Spring 2023

Hate Speech Detection in Hindi

Pranjali Prakash Bansod
San Jose State University

Hate Speech Detection in Hindi

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Pranjali Prakash Bansod

May 2023

The Designated Project Committee Approves the Project Titled

Hate Speech Detection in Hindi

by

Pranjali Prakash Bansod

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2023

Dr. Fabio Di Troia	Department of Computer Science
Dr. William Andreopoulos	Department of Computer Science
Dr. Robert Chun	Department of Computer Science

ABSTRACT

Hate Speech Detection in Hindi

by Pranjali Prakash Bansod

Social media is a great place to share one's thoughts and to express oneself. Very often the same social media platforms become a means for spewing hatred.The large amount of data being shared on these platforms make it difficult to moderate the content shared by users. In a diverse country like India hate is present on social media in all regional languages, making it even more difficult to detect hate because of a lack of enough data to train deep/ machine learning models to make them understand regional languages.This work is our attempt at tackling hate speech in Hindi. We experiment with embeddings like fastText and GloVe combined with machine learning classifiers like logistic regression and decision tree classifier. We also experiment with transformer based embeddings like distilBERT and MuRIL.The transformer based models perform better in our task and we achieve an F1 score of 0.73 with the help of MuRIL embeddings.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my project advisor Dr. Fabio di Troia for guiding, encouraging and correcting me throughout the project. This project would not have been possible without the constant support and inputs of Prof. Di Troia. I would also like to thank my committee members Prof. William Andreopoulos and Prof. Robert Chun for evaluating this work and providing valuable guidance.

TABLE OF CONTENTS

CHAPTER

1 Introduction 1

2 Background 4

2.1 Related Work 4

2.1.1 Hate speech detection in Urdu 4

2.1.2 Hate Speech Detection in Marathi 6

2.1.3 Hate Speech Detection in Hindi 6

2.2 Embeddings 7

2.2.1 GloVe 9

2.2.2 fastText 11

2.2.3 BERT 12

2.2.4 DistilBERT 13

2.2.5 BERT Embeddings for Indic languages 14

2.3 Techniques 15

2.3.1 Under-sampling and Oversampling 15

2.3.2 Class Weights 15

2.3.3 Logistic Regression 15

2.3.4 Decision Tree Classifier 17

2.3.5 Dropout 17

2.3.6 Layer Normalization 19

3 Experiments 20

3.1 Hardware Setup for experiments 20
3.2 General process for experiments 20
3.3 Description of the dataset 22
3.3.1 Experiment 1: Finetuning DistilBERT pretrained model . 22
3.4 Experiment 2: Finetuning MuRIL pretrained model 23
3.5 Experiments with fastText and GloVe embeddings 23
3.6 Experiment 3: fastText embeddings with logistic regression classifier 23
3.7 Experiment 4: fastText embeddings with Decision Tree Classifier 24
3.8 Experiment 5: GloVe embeddings with logistic regression classifier 24
3.9 Experiment 6: GloVe embeddings with Decision Tree Classifier . . 24
4 Results 25
5 Conclusion and Future Work 40
5.1 Conclusion 40
5.2 Future Work 41
LIST OF REFERENCES 42
APPENDIX
A 45
B 46

CHAPTER 1

Introduction

The internet is a wonderful place to share one's thoughts, knowledge and experiences. Unfortunately, it sometimes also becomes a place where hurtful things are said to a person or targeted at a group of people. Hatred can also be directed at people belonging to certain community, race, ethnicity, religion or gender. These are traits of a person on which that person has no control.As a consequence hateful speech can have lot of negative psychological effects on a person. These negative effects include feelings of worthlessness and a low self-esteem and in worst cases might also make a person suicidal. The targeted individuals might also feel isolated from the rest of the society and they might choose to withdraw from participating in any discussion on any social media platforms. In the worst case online hatred might turn into crimes in real world if it invokes feelings of revenge amongst the targeted people. Tackling online hate speech is a complicated problem since the volume of data is so large that checking every written item for hate is difficult.There is a pressing need for AI based moderation to detect hateful content with minimum human intervention. Secondly, laying out the guidelines for classifying any written item as hateful is challenging since hate is very subjective, what is hateful for one person might be mildly offensive to another person. So, when data is collected for training models for hateful speech detection it is difficult to agree on labels for the given data. If the data is related to any issue going on in some part of the society then there might be varied opinions about that issue based on the knowledge level of the people labeling the data[1]. The problem of hate speech becomes manifold in India since it is such a diverse country in terms of language, culture and ethnicity. Almost every state has it's own language and hateful content is present in the online world in most of those languages accompanied with hateful content in English. To further compound the situation, the content from

these languages is present in a code-mixed form where languages are mixed together or a script of a different language is used to write something in another language. More and more data is required to train language models on these languages in order to identify hate speech and mitigate it. The problem of hate speech in India needs to be taken seriously since it is one of the largest markets for social media platforms like Facebook, Twitter, Instagram, YouTube etc. Fig. 1 gives a glimpse of the language diversity found in India. One can see that Hindi is spread over a larger portion of the

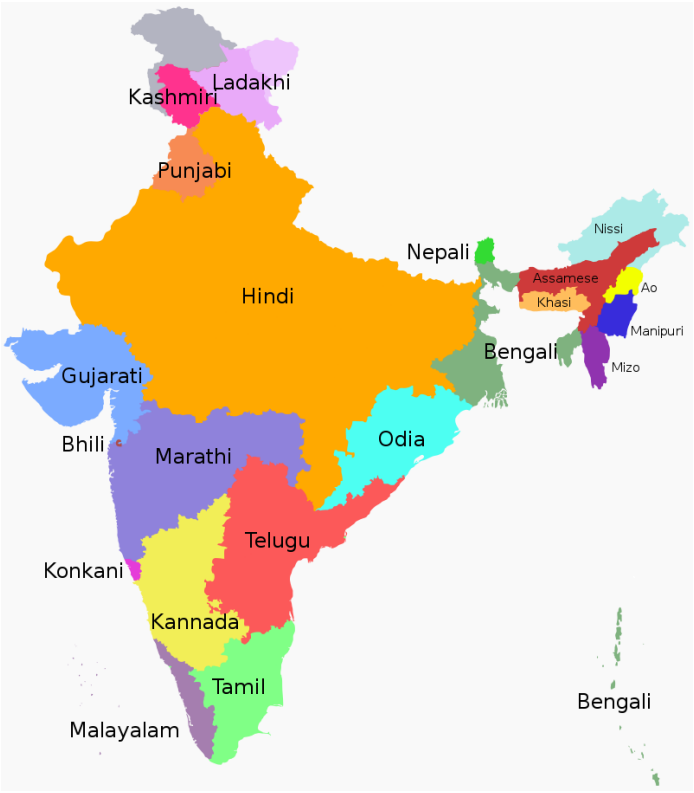


Figure 1: Indian languages and their spread through out the country.

country and in this work we plan to tackle hate speech detection in Hindi and Hindi written using English letters.67

The remnant of this work has been divided as follows. Chapter 2 covers the work related to hate speech detection in languages similar to Hindi. It also covers the

techniques that are used for sentiment analysis in general. Chapter 3 contains all the experiments that we have conducted for hate speech classification in Hindi. Chapter 4 discusses the results of the experiments discussed in Chapter 3. In Chapter 5 the possible future research for hate speech detection in Indian languages is discussed.

CHAPTER 2

Background

This chapter discusses the work done so far in the field of hate speech detection in Indian languages. It also discusses the different approaches whether it is embedding, deep-learning or machine-learning models used for sentiment analysis in general.

2.1 Related Work

This section discusses the approaches used for hate speech detection in other indic languages that are similar to Hindi.Languages like Marathi and Urdu share many features with Hindi. Secondly, it discusses the existing approaches to tackle hate speech detection in Hindi.

2.1.1 Hate speech detection in Urdu

Urdu is a language that originated in India and is spoken in countries like Pakistan and some parts of Afghanistan. It shares a lot of grammar and words with Hindi and is known commonly because of the beautiful poetry compositions written in Urdu. In India it is mostly spoken in states like Uttar Pradesh, Telangana and Jammu and Kashmir. Urdu has a perso-arabic script. There have been efforts to detect hate speech in Urdu. In one such attempt [2] Bilal et al. tried to detect hate speech for Urdu written in the Roman script. They created a dataset RU-HSD-30K comprising of 30,000 Roman Urdu text messages and trained the BERT model from scratch on this dataset known as the RU-BERT model. In addition to this, they have tried models like multilingual BERT and BERT-English models on the RU-HSD-30K dataset. They used 2 approaches for fine-tuning the transformer models. One way was to freeze the weights of the pre-trained transfomer models and use them with BiLSTM and BiLSTM with attention networks for classification and the other way was to update the weights of the pretrained models as they get trained along with BiLSTM and BiLSTM with attention networks. They compared the results of transformer

based models with that of Word2Vec embeddings combined with machine learning algorithms. The transformer models outperformed the Word2Vec embeddings and ML algorithms by a huge margin. The highest accuracy was achieved by the random forest classifier amongst the traditional ML models and was 71 %. The highest accuracy achieved by transformer based models was 97 %. In 2021 Saha et al.[3] presented an approach to classify Urdu text as abusive and threatening. They used the Urdu HASOC 2021 dataset for their experiments.As one approach, they combined BERT embeddings with XGBoost classifier and LGBM classifier turn by turn and in the other approach they used multilingual-BERT, dehatebert-mono-arabic. Since, the dataset for threatening vs non-threatening faced a class imbalance the authors had to apply class weights to their model training. The highest F1 score achieved for the abusive tweet classification was 0.88 and was due to the dehatebert-mono-arabic model. For the threatening text classification the highest F1 score was achieved by the same model and was 0.54. [4] addressed the lack of guidelines for classifying any speech as hateful. In 2021 [4]Malik et al. formed the dataset HS-RU-20 of Roman Urdu text where they classified speech into 3 categories - hateful, offensive and neutral. According to guidelines laid out by the authors of [4] a speech is hateful when it is directed towards any trait of a community/person on which they have no control such as gender, race, physical features etc. Offensive speech is when a person/community is attacked but not based on traits on which they have no control, meaning the content is still offensive but not hateful. This dataset has then been used to train classical machine learning models on embeddings like count vectorizer, n-gram vectorizer and character level features. The logistic regression performed the best giving an accuracy of 84 per-cent when combined with the count vectorizer embeddings.

2.1.2 Hate Speech Detection in Marathi

Marathi happens to be the cousin language of Hindi. It has the same script has Hindi and many words in both the languages are common. It is spoken in the western part of India, in the state called Maharashtra. In 2021 Velankar et al. [5] tackled Marathi hate speech detection on the Marathi dataset provided by HASOC for the year 2021. The authors of [5] used embeddings like fastText and combined the fastText embedding with deep learning methods like 1D-CNN, LSTM and BiLSTM one by one .They also used transformer based embeddings like multilingual-BERT, IndicBERT[6] and Roberta-mr for hate speech detection in Marathi.They have adopted a hierarchial approach for training.If a tweet is classified as hateful in the first level of classification then it is further classified as profane, offensive or just hateful. The non-trainable version of fastText embeddings combined with 1D CNN achieved an accuracy of 83 %. Amongst the transformer based approaches, the IndicBERT performed the best giving an accuracy of 88 %. Joshi et al.[7] created a large dataset of Marathi tweets with over 25000 samples known as L3Cube-MahaHate. The dataset has categories like hateful, offensive, profane and non-hateful. The authors of [7] trained the multilingual BERT model on this dataset and shared it as MahaBERT on HuggingFace[8].

2.1.3 Hate Speech Detection in Hindi

In [9] the authors again used the HASOC 2021 dataset along with another dataset like CONSTRAINT – 2021 for detecting hate speech in Hindi. They have done hate speech detection for Code mixed Hindi, Hindi written in Devnagari script and also for English sentences written in Devnagari script. The authors of this work have made use of the multilingual transformer model (m-bert) for classification.In [9] the authors categorized hate speech into 34 different categories out of which 28 categories were for hateful/ non - hateful speech in single language (Hindi) and the remaining 6

categories were for code mixed hate speech. Not all sentences containing hateful words are hate speech, for example "I hate apples" does not target any person or group, it is just somebody expressing their dislike for apples. The motivation for creating these categories was to separate such non – hateful sentences like the one above from actual hateful speech. This work also dealt with hateful speech with missing offensive words and missing names of the protected groups against which the hate is being directed. Such sentences also form some of the categories. In [10] Kanade et al. used embeddings like TF-IDF, Word2Vec and Bag-of-Words (BOW) on the HASOC 2021 dataset to classify Hindi tweets as hateful and non hateful. They passed these embeddings to traditional machine learning models like Naive Bayes, Logistic Regression, SVM and Random Forest Classifiers. They achieved a macro F1 score of 69 per-cent and an over-all accuracy of 75 per-cent on the binary classification task when they combined Word2Vec with an SVM classifier. The authors of [11] experimented with BERT embeddings and also experimented with the combination of BERT embeddings with convolutional networks. They have run their experiments separately on the HASOC 2020 and HASOC 2021 dataset. To address the class-imbalance issue in these datasets they have applied over-sampling to the datasets and the experiments were run on the over-sampled version and the original version of the dataset separately. On the over-sampled version of the HASOC 2020 dataset they achieved an F1 score of 85 per-cent and on the HASOC 2021 dataset they achieved an F1 score 77 per-cent with the BERT combined with 1D-CNN approach.

2.2 Embeddings

This section discusses the different Embeddings and approaches used for converting text into vectors.Embeddings have come a long way and are crucial to natural language processing. Word embedding is a way of converting a word into vector while

also extracting meaningful information about the words of a corpus. Embeddings store lot meaningful information about words such as the semantic relationship amongst words whether 2 words are synonyms or antonyms, they also capture syntactic relationship between words ,whether words form a noun-to-verb or verb-to-noun relationship and the types.They also store contextual information about words. Research regarding word embeddings goes back to early 2000s but the first break through came with Word2Vec embeddings[12] Mikolov et al. created the Word2vec embeddings at Google. Word2Vec quickly became popular and was being widely used for NLP tasks. The basic idea behind Word2Vec embeddings is that similar words appear together in similar context and hence their vectors are close to each other. Word2Vec brought 2 ways to train the neural network, one was the skip-gram approach and the other was continuous bag of words (CBOW). In the skipgram approach the neighbouring words of a given word are predicted. In the CBOW approach given the surrounding words of a word, the word needs to be predicted. Word2Vec uses a single hidden layer of

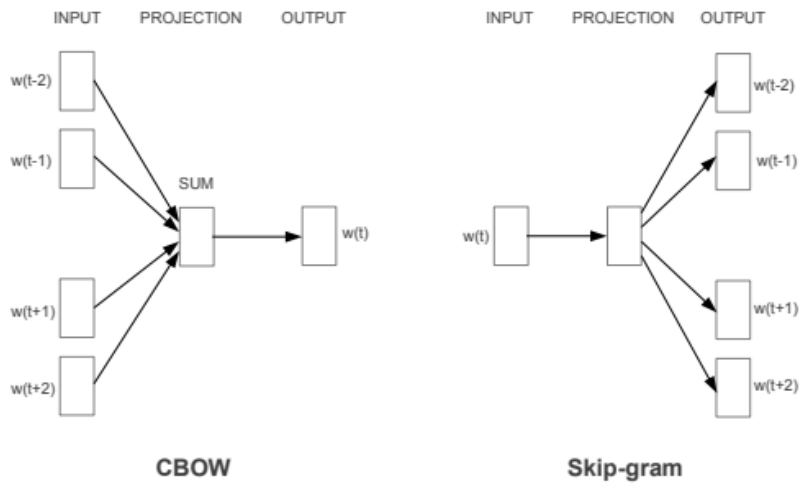


Figure 2: Idea of CBOW and skip-gram training [12]

length N and width d for training where N is the size of the vocabulary and d is the

dimension of the output vector, in the original word2vec paper a dimension size of 300 was used. 1-hot encoded vectors of size N are passed to the this layer. If we are training using the skip-gram approach then there will be only one place marked as a 1 in the input vector.The output of the hidden layer is passed to a softmax layer. In case of a skip-gram approach there will be an output vector of size d returned at the end of the training. Although, word2vec was a significant development in the

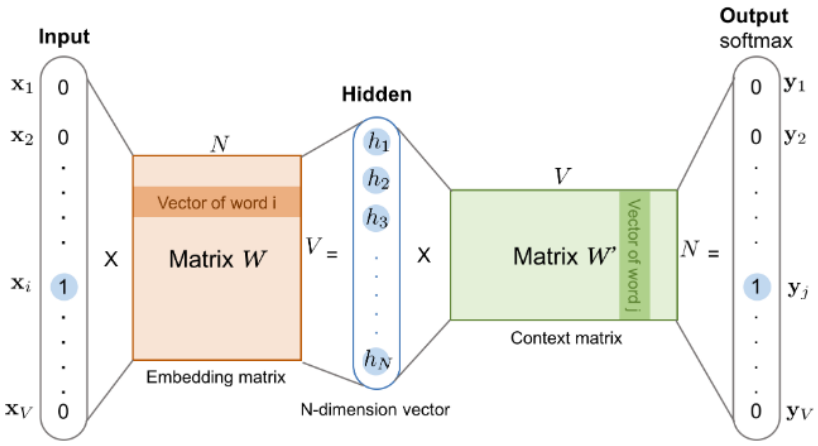


Figure 3: Skip-gram training with hidden layer [13]

world of embeddings it has some shortcomings. Word2Vec embeddings are static they don't change with their context. Word2Vec embeddings cannot handle out of vocabulary words.These limitations gave room for further improvements and other types of embedding were further developed.

2.2.1 GloVe

GloVe is an unsupervised method for converting words into vectors.It was introduced at Stanford University in 2014[14].It figures out how frequently 2 words appear together through out a given corpus. For maintaining this information it uses a co-occurrence matrix. Fig.4 shows an example of a co-occurrence matrix.

GloVe also borrows the sliding window method used in Word2Vec for capturing

Doc 1: All that glitters is not gold

Doc 2: All is well that ends well

*	<START>	all	that	glitters	is	not	gold	well	ends	<END>
<START>	0	2	0	0	0	0	0	0	0	0
all	2	0	1	0	1	0	0	0	0	0
that	0	1	0	1	0	0	0	1	1	0
glitters	0	0	1	0	1	0	0	0	0	0
is	0	1	0	1	0	1	0	1	0	0
not	0	0	0	0	1	0	1	0	0	0
gold	0	0	0	0	0	1	0	0	0	1
well	0	0	1	0	1	0	0	0	1	1
ends	0	0	1	0	0	0	0	1	0	0
<END>	0	0	0	0	0	0	1	1	0	0

Figure 4: Example of a co-occurrence matrix [15]

the local context of a word, that is the words surrounding a word.When forming the co-occurrence matrix for a corpus the window slides over the corpus and the words falling within the window are the context words for the center word of that window and those will get incremented in the matrix.The loss function of GloVe can be derived using the log-bilinear regression model.

$$J = \sum_{i,j} f(X_{ij})(w_i^T w_j - logX_{ij})^2 \tag{1}$$

The above loss function captures the information of Pij/ Pik[16]. That is the probability of the words i and j occurring together divided by the probability of j and k occurring together. To briefly summarize the loss function. The term Xij is derived from the co-occurrence matrix and it is the number of times the words i and j occurred together.the function f(Xij) is the power law function to make sure that frequent pairs of of words i and j are not given too much weight nor rare co-occurrences are given

too much weight.Before the training begins, every word in the corpus is assigned a numeric value.A matrix of size (vocabulary * embedding size) is formed with random values as the initial vector representations for words like wi and wj.Here wi and wj are words within a context.The loss is calculated by passing all the 3 values wi, wj and Xij to the loss function and the gradient is then applied to the vectors of the center word wi and the surrounding context vector wj. In [17] the GloVe algorithm has been implemented from scratch and gives a clear idea about the inner workings of GloVe.

2.2.2 fastText

fastText[18] embeddings were created by Facebook. FastText embeddings implementation forms character n-grams of a given word because there is lot of valuable information hidden at the character level in a word. fastText makes use of negative

Figure 5: 3-grams example [19]

sampling where negative samples are words randomly picked from the corpus and are not the neighboring words of a given word t. The vector representation of the word 't' is formed by taking the sum of it's character n-grams. This sum is then added to the vector of that whole word itself. A table containing random vectors for every word is used before the training is started, similar to the GloVe approach.The dot product of the vectors for the actual context words of the word t is taken. The dot product of the negative samples and the word t are also taken. The sigmoid function is then applied to both the vector products to get a score between 0 and 1. The training for fastText is basically bringing the score of the actual context to 1 and reducing the

score of the negative samples.[19]. The loss used for doing this type of training is the binary logistic loss. Once the loss is calculated then those gradients get applied to the vectors and their values get updated. This is the skip-gram way of training the fastText model.

Figure 6: Idea of fastText training [19]

2.2.3 BERT

BERT stands for bidirectional encoder representation from transformers[20]. The BERT model is available in different sizes. The BERT-Base model is formed by stacking 12 encoder blocks and uses 12 attention heads whereas the BERT-Large is formed by stacking 24 encoder blocks and it uses 16 attention heads.The architecture of an encoder is shown below: To briefly describe the working of an encoder [21]we send the positional embeddings of a word to the encoder and the self attention is calculated. Self-attention is the most important part, in this part the word embeddings are treated as 3 values Q, K and V where Q stands for Query , V stands for Value and K stands for Key. We can consider Q as a search term we input in a search engine where it will be matched against K keys that are a potential matches for Q and V is the values returned after the search is completed. Analogous to this example, Q is the current word in a sentence and K, V are the remaining words of the sentence. So, these 3 vector values are passed through a linear layer each. A dot product of

Q and K is taken which gives us a table or a matrix showing the relevance value of every word to every other word. This matrix is scaled down since multiplication can result in exploding values. The softmax function is applied on this matrix to give us probabilities for every value in the matrix. If we split the Q, K and V vectors into pieces and pass it down to multiple attention units working in parallel then we are using something called a multi-headed attention. The encoder also makes use of residual connections. Once we compute the product of the probabilities-matrix and the V vector we add this product to the input positional embeddings, this addition back into the input is called residual connection. The residual connection output then passes through a layer normalization. As the last step, the normalized residual output is passed through 2 feed-forward network layers with ReLU activation. Fig. 7 shows the inner workings of a transformer block.

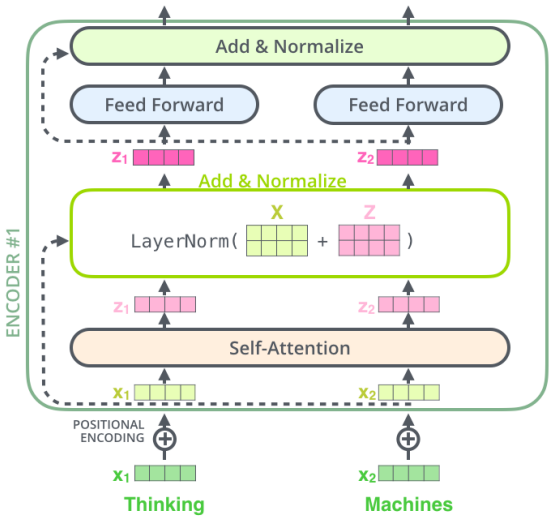


Figure 7: Architecture of an encoder block [22]

2.2.4 DistilBERT

DistilBERT [23]is a smaller and compact version of the original BERT embeddings. The original BERT embedding has many layers and it takes lot of computation

resources and time to train BERT embeddings. Using such large models is not feasible for research activities and real-life applications. To take an example of our own project it takes a couple of hours to fine - tune distilBERT on our corpus whereas it takes 4 to 5 hours to fine tune other embeddings like MuRIL on our dataset, for the same number of epochs, GPU configuration and training data due to their huge size.So, in order to reduce the size of the BERT embeddings alternate layers of the BERT model are removed and the knowledge of the larger BERT model is 'distilled' into the smaller model. The DistilBERT is trained using the Teacher-Student training. In teacher-student training the output logits of the teacher or the larger model are also given as input to the loss function. The calculated loss is then used to update the weights of the student model. Fig. 8 shows the idea of distillation.

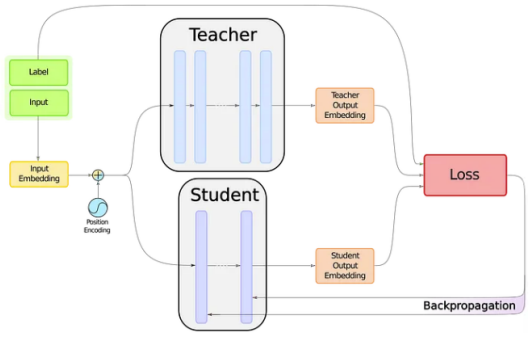


Figure 8: Idea of student-teacher training [24]

2.2.5 BERT Embeddings for Indic languages

Various transformer based models have been trained on corpora from Indian languages. One such embedding that is particularly applicable to the task of Roman-Hindi data is the MuRIL embedings[25] MuRIL stands for Multilingual Representations for Indian Languages. MuRIL is the BERT model trained not only on 17 Indian languages but also their transliterated and translated counterparts. English is also included in the training. The MuRIL model has been trained on corpora like Wiki,

COMMONCRAWL, Dakshina and PMINDIA. The researchers who created this model have also shared it on HuggingFace for others to use. Multilingual models when trained on a large number of languages do not pick up the nuances of a languages and don't learn the language properly in depth. The creators of MuRIL have focused only on 17 Indian languages such that the model properly learns each language.

2.3 Techniques

This section discusses the finer details of machine/ deep learning models, the small tricks that come handy in improving the model's performance.

2.3.1 Under-sampling and Oversampling

In many real life classification problems it is very common that the number of examples for a given category is way less than the number of examples for the other category. In that case there are 2 techniques which are used to tackle this problem. Using under-sampling we randomly remove a portion of the class which is in majority. Oversampling is quite the opposite of under-sampling, in which we randomly duplicate examples of the minority class. These techniques are applied on the training data.

2.3.2 Class Weights

Class weights is a way of altering the loss function in order to address class imbalance in a dataset. The loss function is changed in such a way that when it mis-classifies an example from the minority class it will be penalized more than it will be penalized for mis-classifying an example from the majority class.By adopting this method more weight is given to the minority class.

2.3.3 Logistic Regression

Logistic regression is a supervised machine learning algorithm for used for classification. It makes use of the sigmoid function to separate the values into classes also

known as dependent variables.

$$S(x) = \frac{1}{1 + e^{-x}}$$

(2)

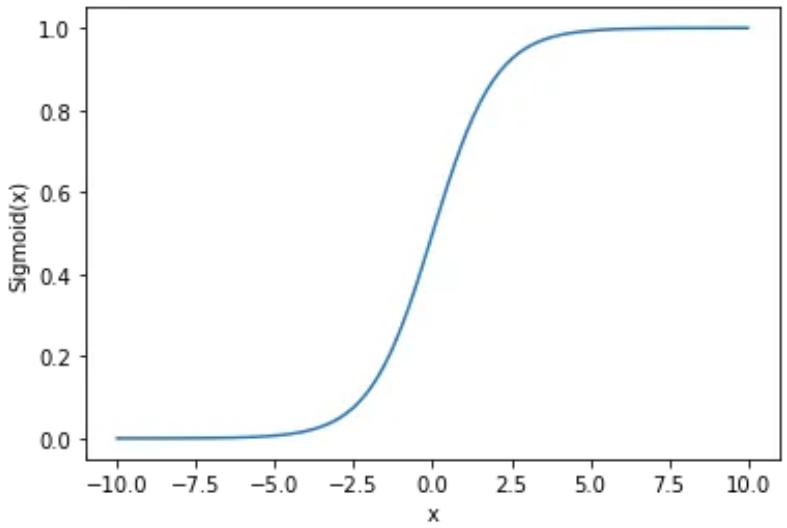


Figure 9: Sigmoid function curve [26]

To briefly summarize the working of a logistic regression classifier, the input parameters are sent to a linear classifier first. Linear classifiers are susceptible to outliers and might not do well in separating the members into their respective classifiers. Therefore, the output of the linear classifier is sent to the sigmoid function.The sigmoid function takes any numeric values and returns a value between 0 and 1. A threshold can be used on the sigmoid function output to do the final classification. A threshold of 0.5 is usually used, it can be seen from the curve of sigmoid function because it cuts the y-axis at 0.5. So, if a value is greater than 0.5 then the label would be 1 and if it is lesser than 0.5 then it will be 0. The logistic regression cost function is then minimized and the best parameters for classification are then learned.

2.3.4 Decision Tree Classifier

Decision tree classifier is a supervised machine learning algorithm. Decision trees are easier to understand.They can also be easily visualized and are useful when the relationship between the dependent and independent variables is complicated It works on the principle of splitting the dataset in such a way that all the examples fall into their appropriate classes by dividing the dataset based on a feature of the input data.The root node of a decision tree is the entire dataset, it recursively keeps splitting the dataset using a feature of the dataset. The quality of the split can be calculated with the help of measures like gini index, entropy, information gain, gain ratio, chi-square. If the split contains samples belonging only to a single class then the split is considered good but if the split contains examples from more than 1 class then it needs to be split further.The leaf nodes of a decision tree are splits which are homogeneous. Decision tree performs splitting on all features of the input data and then selects the feature that gave the best split. The splitting will continue until the split is pure or homogeneous and this can lead to deep trees. Such trees with great depth can cause overfitting and pruning is adopted to keep the depth of the tree in check and hence to avoid overfitting. Branches are removed from the tree such that the overall accuracy of the algorithm is not reduced, branch removal begins from the leaf nodes of the trees. To do the removal of branches the dataset is divided into training and validation dataset and after the branch removal the algorithm is tested on the validation dataset to check if the accuracy is dropped.

2.3.5 Dropout

Dropout [28] is technique used for tackling the issue of over-fitting. When a particular model shows a great accuracy on training data but a drastically low accuracy on the test data. It means that model has memorized the examples from the training

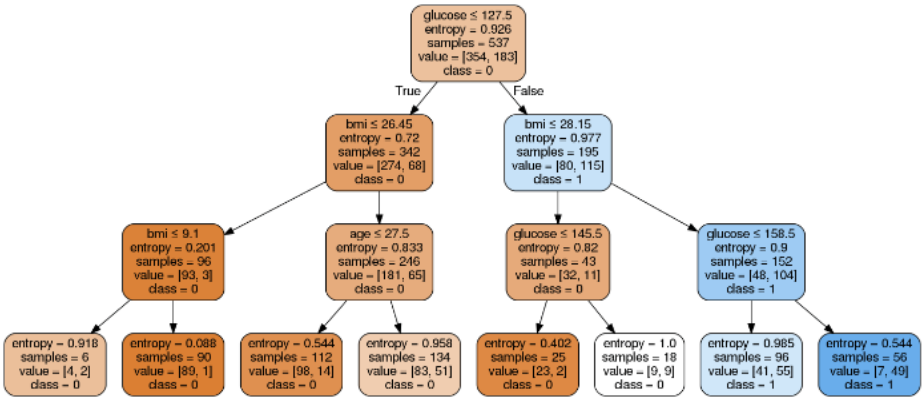
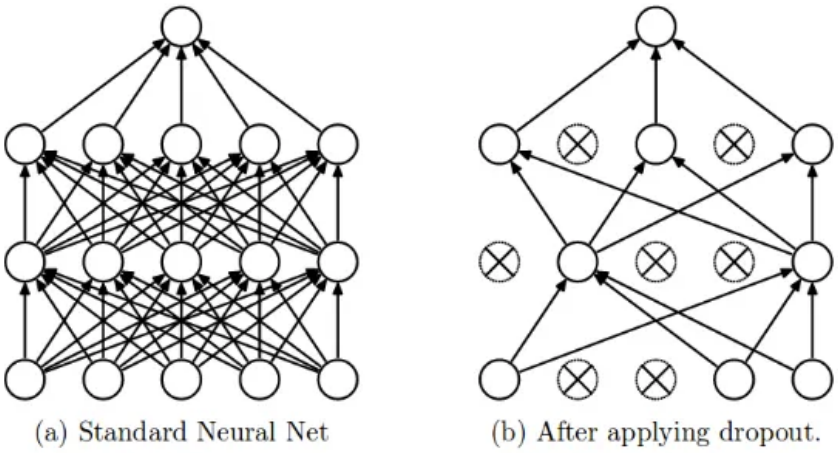


Figure 10: Decision tree splitting [27]

data set instead of learning the underlying pattern to distinguish between classes. Overfitting is caused when there is a dependence between the neurons of a neural network such that an update in one neuron improves/ fixes the result of other neurons. Such a dependence causes overfitting. By using a dropout we are simply removing some of the neurons of the network such that every neuron is just responsible for their own output and the dependence between neurons is reduced. The dropout layer has been used in the BERTforSequenceClassification model which has been used in our work.



(a) Standard Neural Net (b) After applying dropout.

Figure 11: Dropout [28]

2.3.6 Layer Normalization

The encoder in the transformer makes use of layer normalization. It is a crucial step in the encoder architecture. If a neural network receives inputs that vary along a large range then it will take longer to train because while training a neural network the weights are updated to minimize the error and large values might lead to large updates. When these large updates accumulate over several layers they cause exploding gradients. In order to solve this issue Hinton et al. [29] came up with the approach of layer normalization. The idea is to normalize the input to the layer such that there are no fluctuating values in the input. Layer normalization is suitable for small batch sizes because a small batch size will not show a very large variation in inputs as compared to large batch sizes. In natural language processing usually a small batch size is used, layer normalization is suited for such tasks.

CHAPTER 3

Experiments

This chapter goes over the experiments we conducted on the HASOC dataset. We have divided the experiments into 2 categories. The first category explores the transformer based embeddings like DistilBERT and MuRIL. embeddings . The second category contains experiments with the GloVe and FastText embeddings. We tried a series of experiments from which we found some to be successful and we have documented those in this section.

3.1 Hardware Setup for experiments

The MuRIL model and embeddings were fine tuned on the SJSU High Performance Computing cluster using the NVIDIA Kepler 40 (K40) GPU. We fine tuned the models using 2 GPUs. The GloVe and FastText embeddings were trained on an intel i7 10th gen processor. DistilBERT embeddings and model were finetuned on the Google Colab Pro runtime that supports GPU.

3.2 General process for experiments

1. Text cleaning Tweets contain hash-tags, emojis, URLS and special characters. These add unnecessary noise to the model and removing these helps any model learn better.So we remove emojis, URL links, punctuation and all the special characters from a tweet.

2. Nukta or diacritic character is a dot that is added to a letter in the Devanagari script to represent sounds that were not part of the script originally and were borrowed from other languages like Urdu as an example.

3. Since our data is written in Hindi we transliterate the data into roman script by making use of a transliteration library[30].

4. We form the embeddings for fastText and GloVe mebeddings for the tweets and

Figure 12: Nukta characters

run traditional machine learning models on the embeddings to classify them as hateful or non-hateful.

5. We pass down the tweets to the BERTforSequenceClassification model with DistilBERT embeddings as base layer and fine tune the model.

6. Similar to the previous step, we pass the tweets to the BERTforSequenceClassification with MuRIL as the base layer for fine tuning.

Fig. 13 summarizes the entire process of the project.

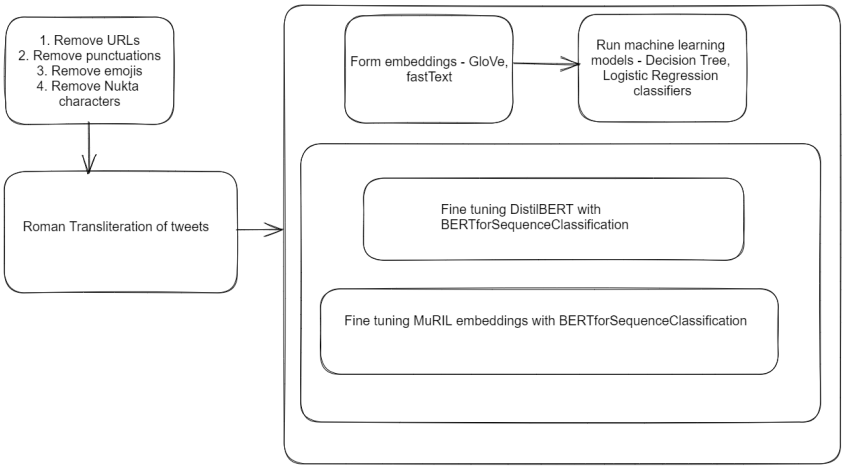


Figure 13: General Idea of the project

3.3 Description of the dataset

We have combined 3 datasets containing tweets in mixed Hindi and English provided by HASOC into one dataset. The combined dataset contains 9753 tweets. Out of the 9753 tweets 2281 are hateful tweets and remaining are non-hateful. HASOC stands for Hate Speech and Offensive Content Detection in English and Indo-Aryan Languages. It is a competition that is being held every year since the year 2019 where teams submit their models for detecting hate-speech in Hindi, English and Marathi. HASOC provides these datasets to researchers for use and we have made use of the same.

Table 1: Table showing the number of hateful and not hateful tweets in the dataset

Tweet Type	Number of Tweets	Percentage
Hateful	2281	23.38%
Not Hateful	7472	76.61%

3.3.1 Experiment 1: Finetuning DistilBERT pretrained model

Transfer learning is the technique of using pre-trained models on tasks similar to our own task. The weights of the existing models are updated by training the model further which is known as fine-tuning the model. Transfer learning is more suitable for our problem because our dataset is not very large and pretraining is suitable only if we have a large dataset.Secondly, finetuning is faster than pretraining a model from scratch. We have made use of the HuggingFace version of the BERTforTextClassification with distilbert-base-uncased model for generating word embeddings. Fig.14 shows the layers of the bert text classification model. We trained the model with a varying learning rate dictated by the polynomial decay, we started with an initial learning rate of 5e-5.Adam optimizer has been used for the training. We have applied class

weights to the model to tackle the class-imbalance problem. The model was ran for 4 epochs.

```
==========================================================================
Layer (type:depth-idx)                        Param #
==========================================================================
├─DistilBertModel: 1-1                        --
|    └─Embeddings: 2-1                        --
|    |    └─Embedding: 3-1                    23,440,896
|    |    └─Embedding: 3-2                    393,216
|    |    └─LayerNorm: 3-3                    1,536
|    |    └─Dropout: 3-4                      --
|    └─Transformer: 2-2                       --
|    |    └─ModuleList: 3-5                   42,527,232
├─Linear: 1-2                                 590,592
├─Linear: 1-3                                 1,538
├─Dropout: 1-4                                --
==========================================================================
Total params: 66,955,010
Trainable params: 66,955,010
Non-trainable params: 0
==========================================================================
```

Figure 14: Individual layers of the BERT classification model.

3.4 Experiment 2: Finetuning MuRIL pretrained model

Similar to the distilBERT pretraining, the MuRIL model is finetuned on on our dataset. We used the BERTForSequenceClassification model with MuRIL for generating the embeddings. We have again applied class-weights to the model and trained it for 8 epochs at a very low learning rate of 1e-5 using the Adam optimizer.

3.5 Experiments with fastText and GloVe embeddings

This section explores the fastText and GloVe embeddings along with classic machine learning classifiers. Each experiment was conducted once with oversampling and then without oversampling.

3.6 Experiment 3: fastText embeddings with logistic regression classifier

We form the fastText embeddings for the entire corpus using the fastText implementation of the Gensim library. We go over every word in each sentence of the dataset and we add the vectors of all the words in the sentence to form a single vector

of size 50, this approach is adopted for all the following fastText/GloVe experiments. This vector is then sent to the Logistic Regression classifier. The fastText model was trained for 25 epochs with a window-size of 1.

3.7 Experiment 4: fastText embeddings with Decision Tree Classifier

Similar to the previous experiment, we have created fastText embeddings for the corpus. We have then passed down these embeddings to a decision tree classifier.

3.8 Experiment 5: GloVe embeddings with logistic regression classifier

For this experiment we have used GloVe embeddings for converting words into vectors.We have used the glove-python library implementation for forming the word embeddings.The GloVe model was trained for 20 epochs with a window size of 5. These GloVe embeddings are passed down to a logistic regression classifier.

3.9 Experiment 6: GloVe embeddings with Decision Tree Classifier

Similar to the previous experiment, GloVe embeddings are formed for the entire corpus and then passed down to the decision tree classifier.

CHAPTER 4

Results

This section discusses the results of the experiments described in the previous section. We trained different embeddings accompanied by deep learning and machine learning models. In Experiment 1 we combined fastText embeddings with the logistic regression classifier. We get an overall accuracy of 68 per-cent when we oversample the minority class in the training dataset. To summarize the result we have made use of the precision-recall curves since we have a class imbalance in our dataset.

Fig.15 shows the PR-curve for the logistic regression when trained with fastText embeddings. The classification report for this experiment is shown in the table 2. The recall for the hateful class is 0.66. The precision is 0.38 which appears low but a closer look at the confusion matrix (Fig.16)of this experiment shows that the classifier has 356 false positives which is still a small number when compared to the total number neutral tweets.The number of true negatives is still a high number. The total area under the curve for this experiment is 0.81 and the F1 score for this model is 0.62.

Table 2: Classification report for fastText combined with Logistic Regression

Class	Precision	Recall	F1	Support
Hateful	0.38	0.66	0.48	338
Non-Hateful	0.87	0.68	0.77	1125
Accuracy	-	-	0.68	1463
Macro Avg	0.63	0.67	0.62	1463
Weighted Avg	0.76	0.68	0.70	1463

For this experiment we sent fastText embeddings of sentences to a decision tree classifier. The decision tree classifier has an overall accuracy of 66%. It shows a great recall for the hateful class. However, the number of false positives in this case are higher as shown in the confusion matrix in Fig. 18.The classifier was able to identify

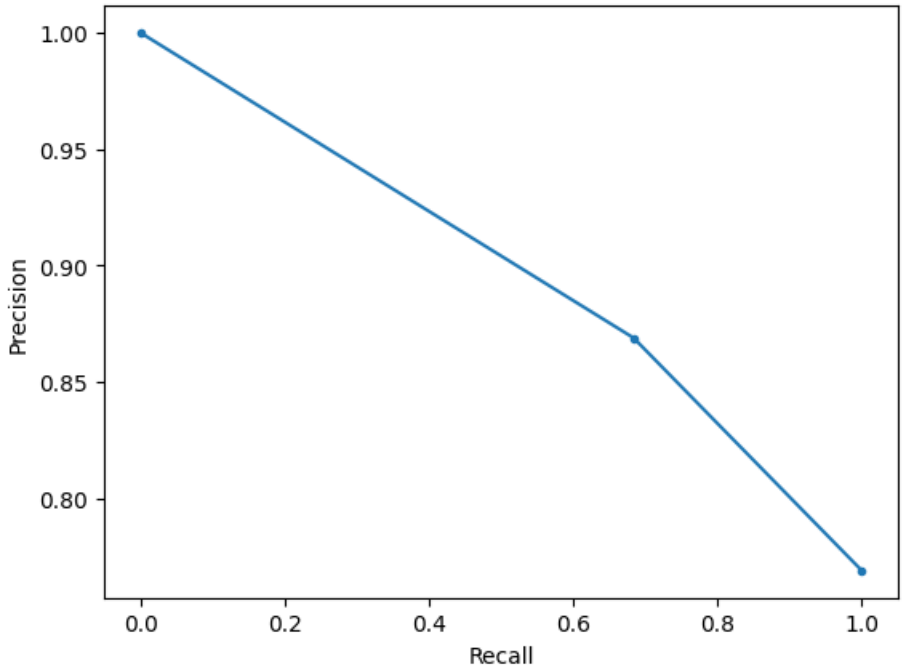


Figure 15: PR curve for logistic regression with over-sampling

677 neutral tweets correctly as shown in Fig.18. The PR curve for this experiment is shown in Fig.17 and the area under this curve is 0.88. The F1 score for this model is around 0.63.

Table 3: Classification report for fastText combined with Decision Tree

Class	Precision	Recall	F1	Support
Hateful	0.39	0.84	0.53	338
Non-Hateful	0.93	0.60	0.73	1125
Accuracy	-	-	0.66	1463
Macro Avg	0.66	0.72	0.63	1463
Weighted Avg	0.80	0.66	0.68	1463

Fig. 19 and Fig. 20 show the result of both the classifiers when no over-sampling is done on the data. It can be seen from the confusion matrix of logistic regression that the model classifies the large number of neutral tweets correctly, giving a high

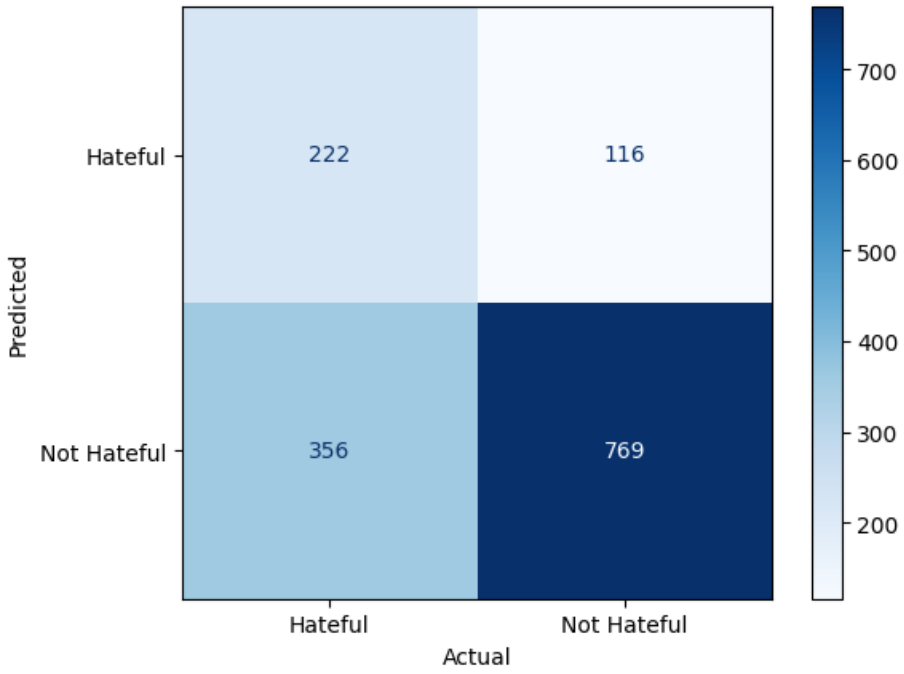


Figure 16: Confusion Matrix for fastText combined with logistic regression classifier with over-sampling

accuracy. However, it is hardly able to identify any hateful tweets correctly and has a higher number of false negatives since it classifies 291 neutral tweets as hateful. The decision tree classifier has a similar outcome however it has a slightly higher number of false positives.

The results for GloVe combined with logistic regression classifier are as follows.Table 4 shows the classification report for this experiment. The overall precision for the hateful class was around 62 per-cent. The precision for the hateful class was 0.33 because it classified 411 non-hateful tweets as hateful as shown in Fig. 27. This number is higher than the number shown in the fastText and logistic regression classifier experiment. The Precision for the not-hateful or neutral class is 0.72 which is still high enough but lower than the corresponding fastText experiment. The area under curve for this experiment is 0.87. Fig. 21 shows the PR curve for the

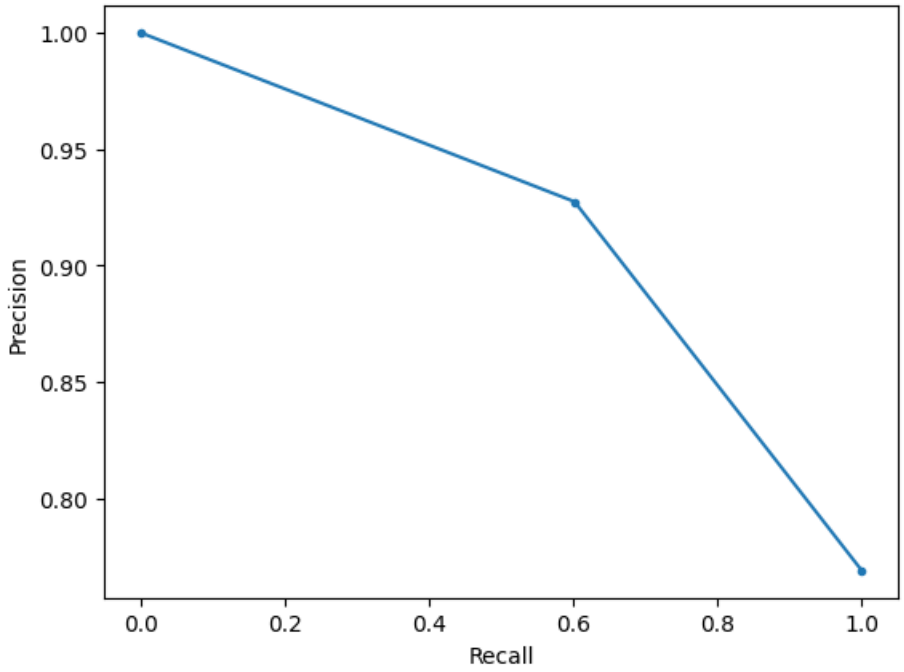


Figure 17: PR curve for decision tree classifier with over-sampling

GloVe-Logistic Regression experiment.

Table 4: Classification report for GloVe combined with logistic regression classifier

Class	Precision	Recall	F1	Support
Hateful	0.33	0.58	0.42	347
Non-Hateful	0.83	0.63	0.72	1116
Accuracy	-	-	0.62	1463
Macro Avg	0.58	0.61	0.57	1463
Weighted Avg	0.71	0.62	0.65	1463

Table 5 shows the classification report for decision tree classifier when used on GloVe embeddings. The recall for hateful class is 0.60 and the precision is 0.28 which is low because the classifier has almost falsely classified half of the neutral tweets as hateful as shown in the confusion matrix in Fig. 23. The area under curve for this experiment is 0.83 and the PR curve is shown in Fig. 24

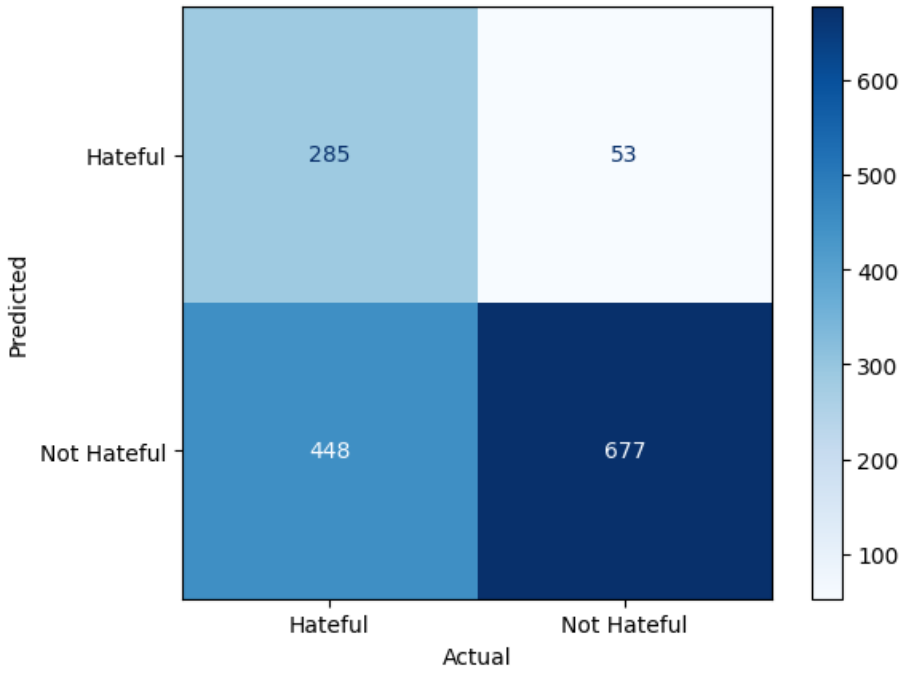


Figure 18: Confusion Matrix for fastText combined with decision tree classifier with over-sampling

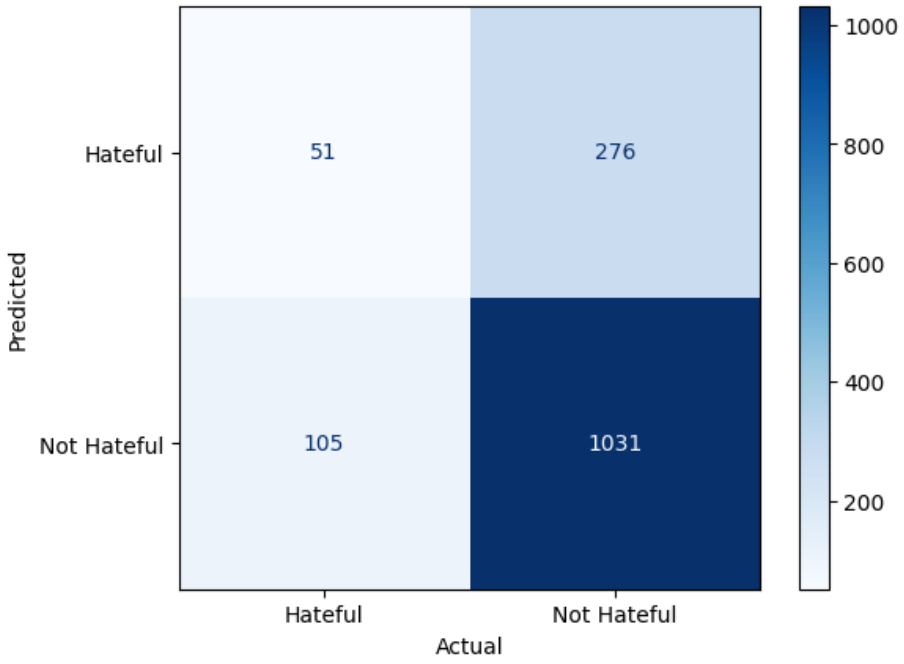


Figure 19: Confusion Matrix for decision tree classifier without over-sampling

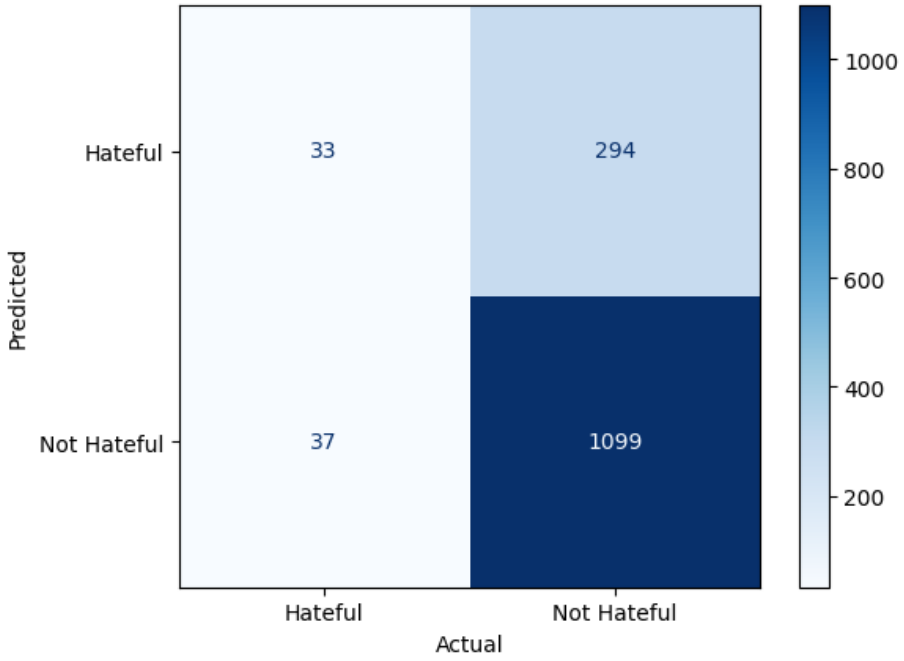


Figure 20: Confusion Matrix for logistic regression classifier without over-sampling

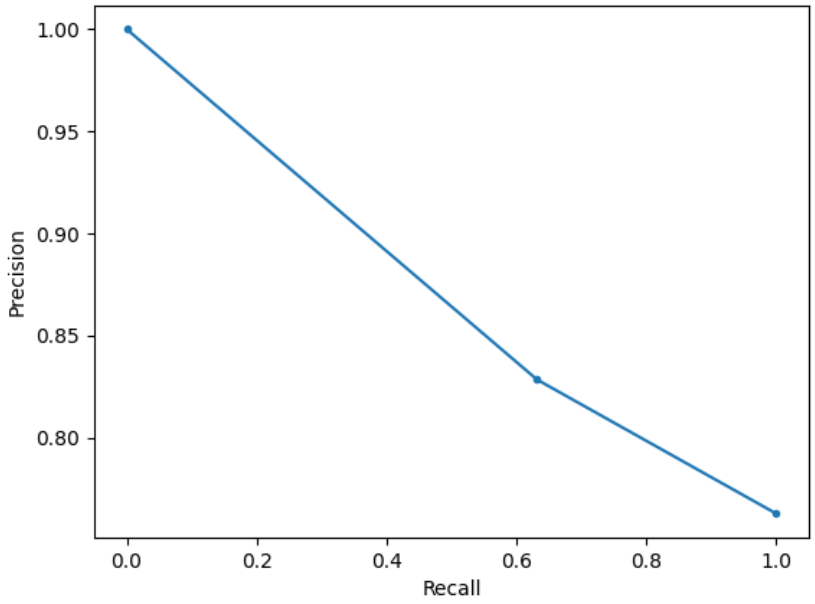


Figure 21: PR curve for GloVe combined with logistic regression classifier

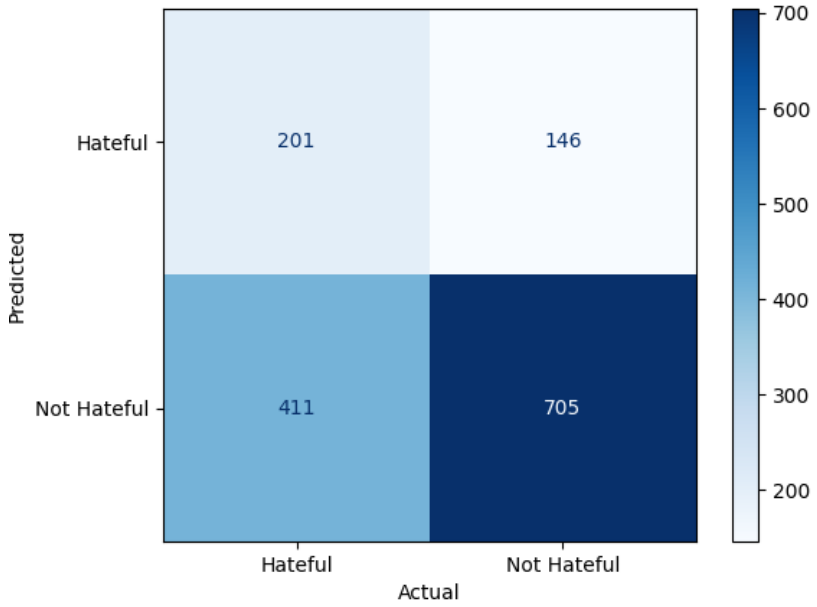


Figure 22: Confusion Matrix for GloVe combined with logistic regression classifier

Table 5: Classification report for GloVe combined with decision tree classifier

Class	Precision	Recall	F1	Support
Hateful	0.28	0.60	0.38	347
Non-Hateful	0.81	0.53	0.64	1116
Accuracy	-	-	0.54	1463
Macro Avg	0.54	0.56	0.51	1463
Weighted Avg	0.68	0.54	0.58	1463

Fig 25 and 26 show the results of not oversampling the training data before sending it to the classifier. Both the classifiers were barely able to classify the hateful tweets correctly.

The distilBERT fine-tuned model has an overall accuracy of 76 per-cent. It is higher than the fastText and GloVe combined with traditional machine learning models. Table 6 shows the classification report for the distilBERT model. We can see

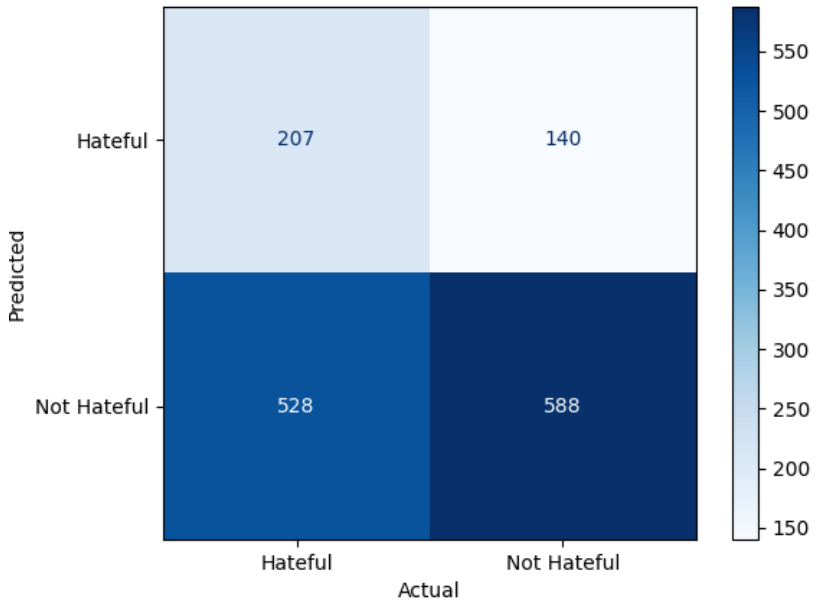


Figure 23: Confusion Matrix for GloVe combined with decision tree classifier

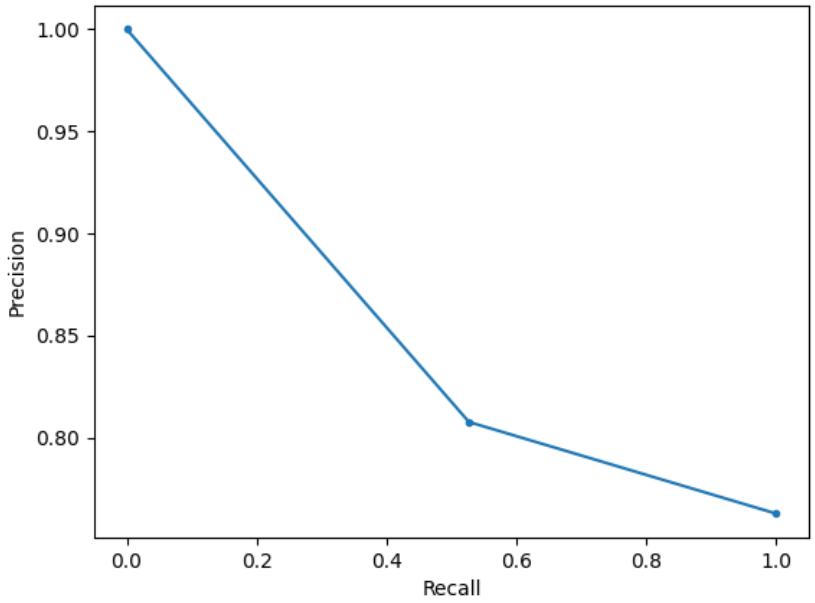


Figure 24: PR curve for GloVe combined with decision tree classifier

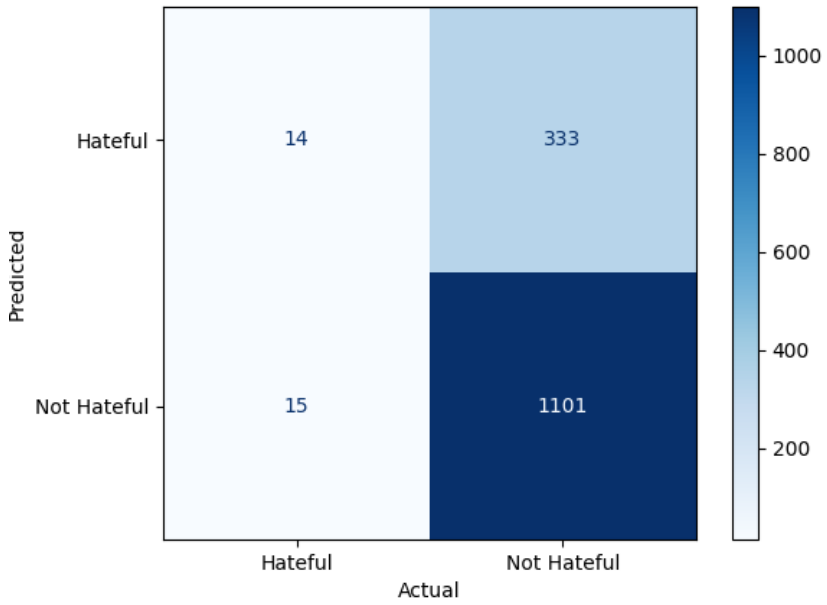


Figure 25: Confusion Matrix for GloVe and logistic regression classifier without oversampling

from the classification report that the precision for the hateful class is much higher in the case of the distilBERT model. It means lesser number of neutral tweets have been classified as hateful. The model has 227 false positives which is lesser than the false positives that the previous experiments gave. The recall for the hateful class is also similar to the previous models in case of the distilBERT model. The precision and recall for hateful class improved without hurting the precision and recall scores of the non-hateful class.The PR curve of the distilBERT model has an area of 0.90 and is shown in Fig. 28 and an F1 score of 0.70.

Fig. 29 shows the overall accuracy trend for training and testing of the model. Fig. 30 shows the loss while training and testing the model. The loss is going downward in both training and validation.

The table 7 shows the classification report for the MuRIL embeddings when they

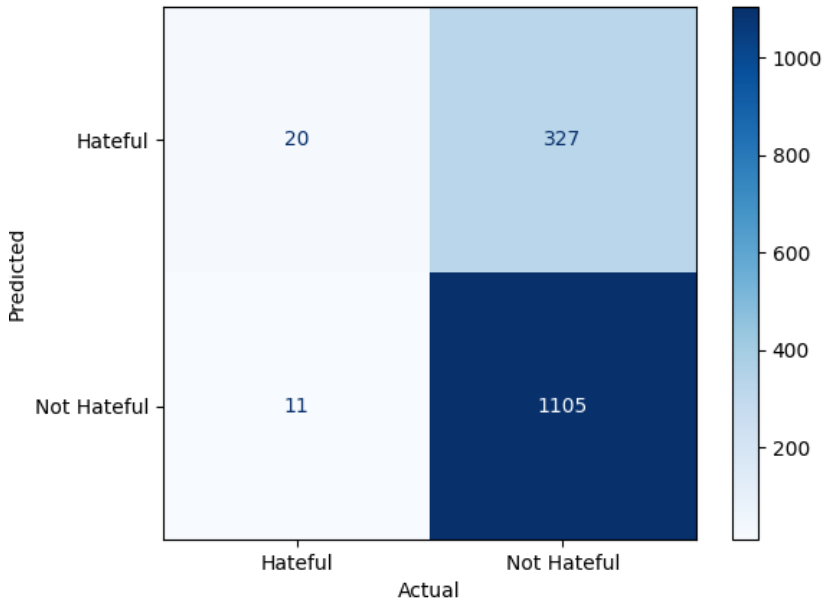


Figure 26: Confusion Matrix for GloVe and decision tree classifier without oversampling

Table 6: Classification Report for DistilBERT finetuning

Class	Precision	Recall	F1-score	Support
Hateful	0.50	0.65	0.56	342
Not Hateful	0.88	0.80	0.84	1121
accuracy			0.76	1463
macro avg	0.69	0.72	0.70	1463
weighted avg	0.79	0.76	0.77	1463

are fine-tuned.

The recall for hateful class in this model is even higher than the distilBERT

Table 7: Classification Report for MuRIL finetuning

Class	Precision	Recall	F1-score	Support
Hateful	0.55	0.66	0.60	342
Not Hateful	0.89	0.83	0.86	1121
accuracy			0.79	1463
macro avg	0.72	0.75	0.73	1463
weighted avg	0.81	0.79	0.80	1463

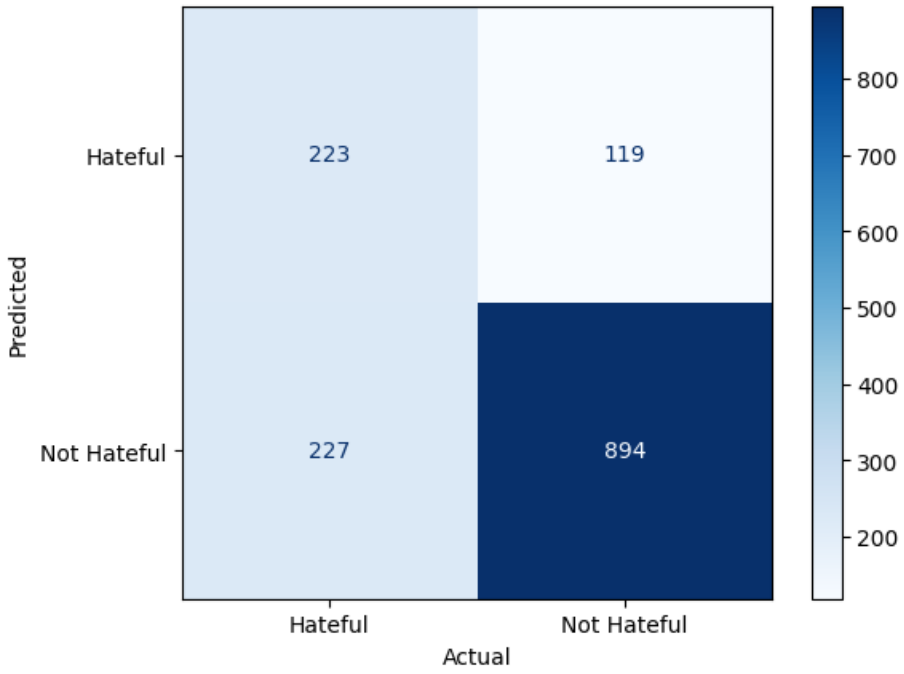


Figure 27: Confusion matrix showing the distilBERT actual numbers

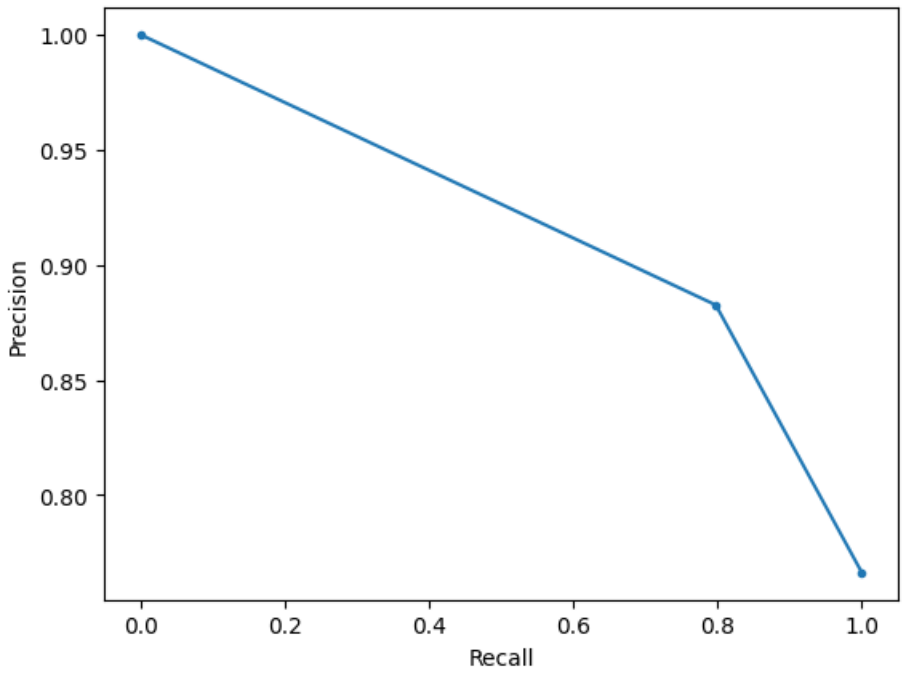


Figure 28: PR curve for distilBERT model

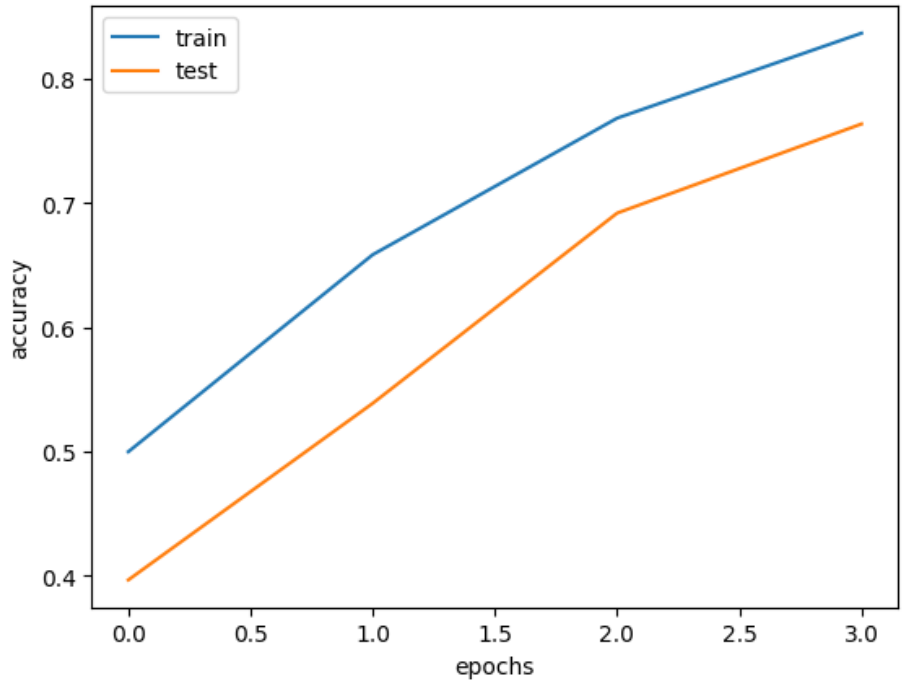


Figure 29: DistilBERT accuracy trend

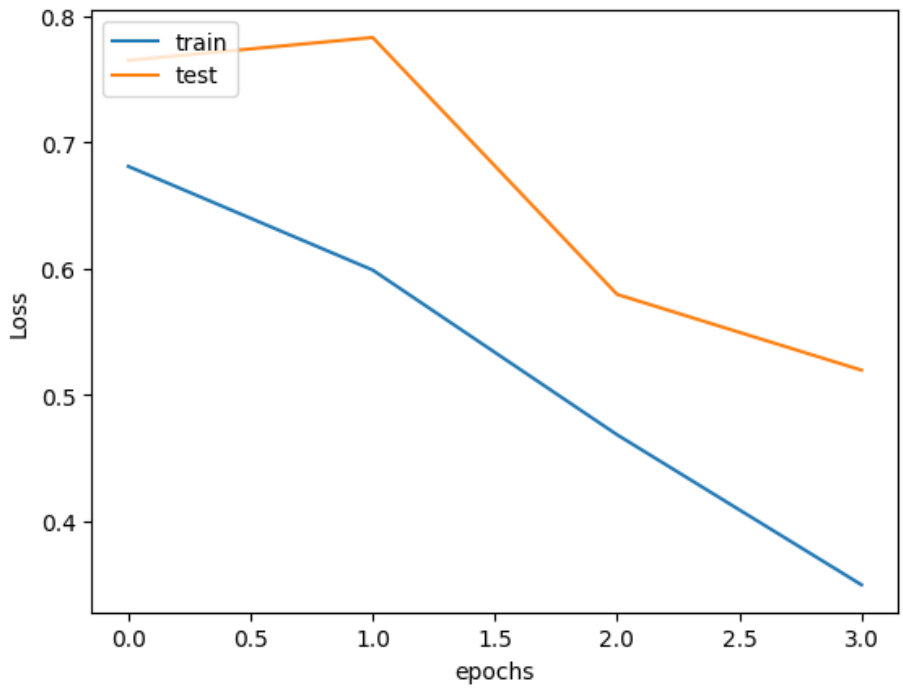


Figure 30: DistilBERT loss trend

model. Even the precision and recall for the neutral class is undisturbed which is a good indication. The precision for the hateful class has also increased and the model has even lesser false positives as shown in the confusion matrix in Fig.31 . The PR

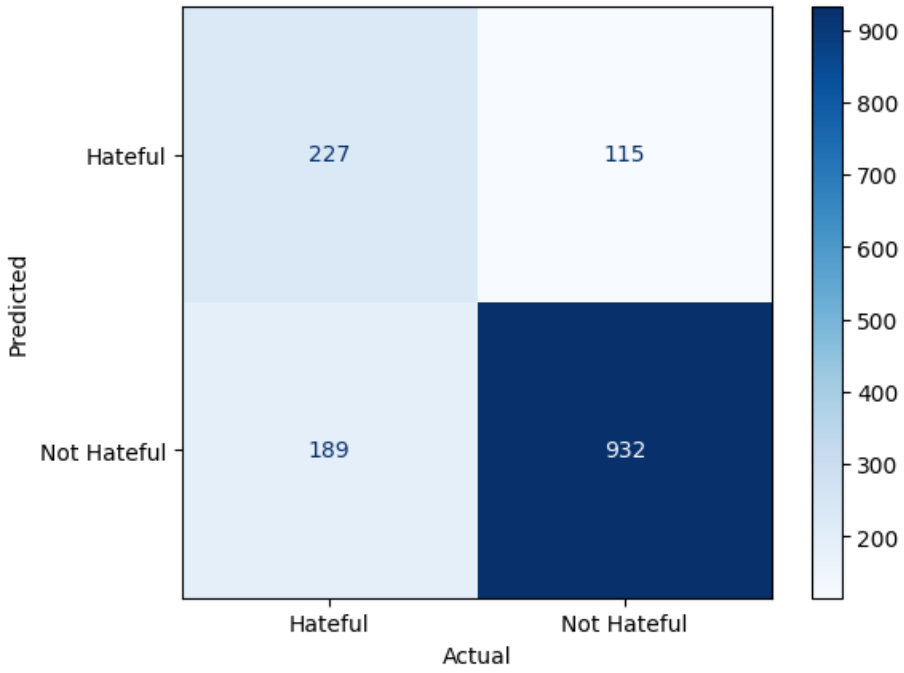


Figure 31: Confusion Matrix for the MuRIL finetuned model

curve for this experiments is shown in the Fig.32. The area under this curve is 0.92 and the F1 score for this model is 0.73. The accuracy and loss trends for the MuRIL model are shown in Fig.33 and Fig.34.

The table 8 shows the results of all the 6 experiments as a summary.

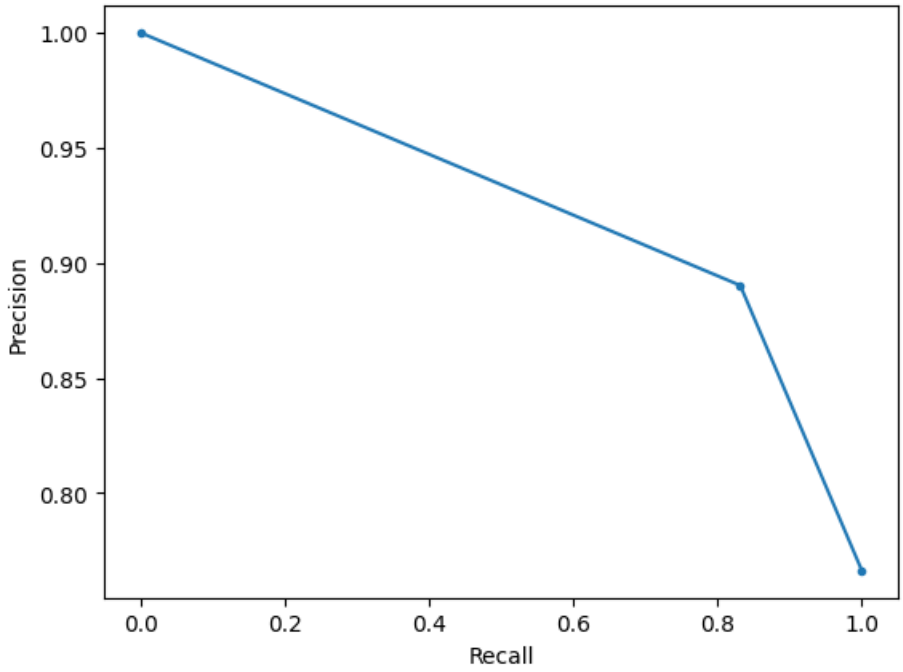


Figure 32: PR curve for the MuRIL finetuned model

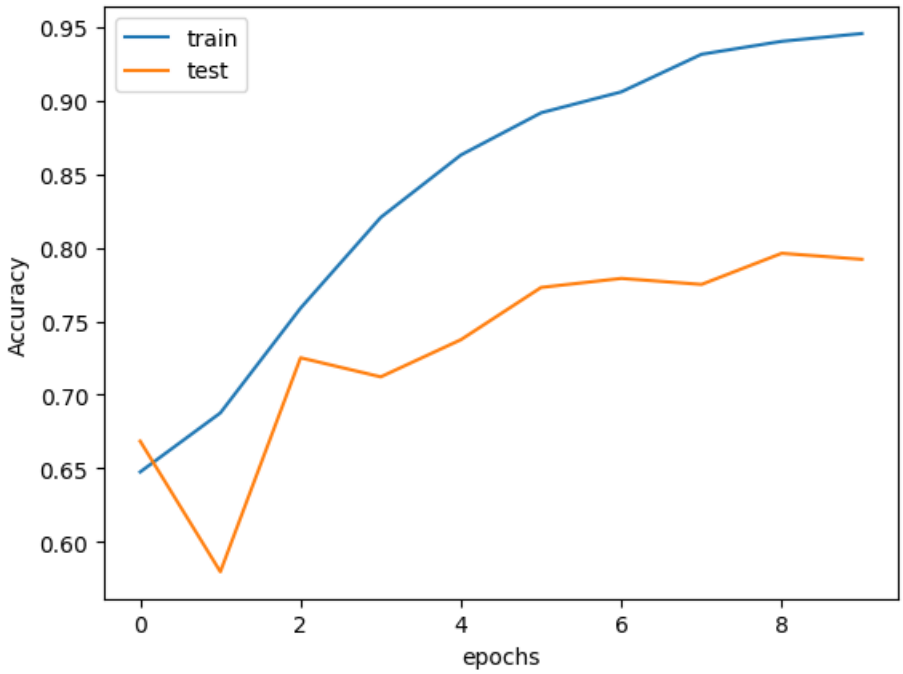


Figure 33: Accuracy for the MuRIL finetuned model

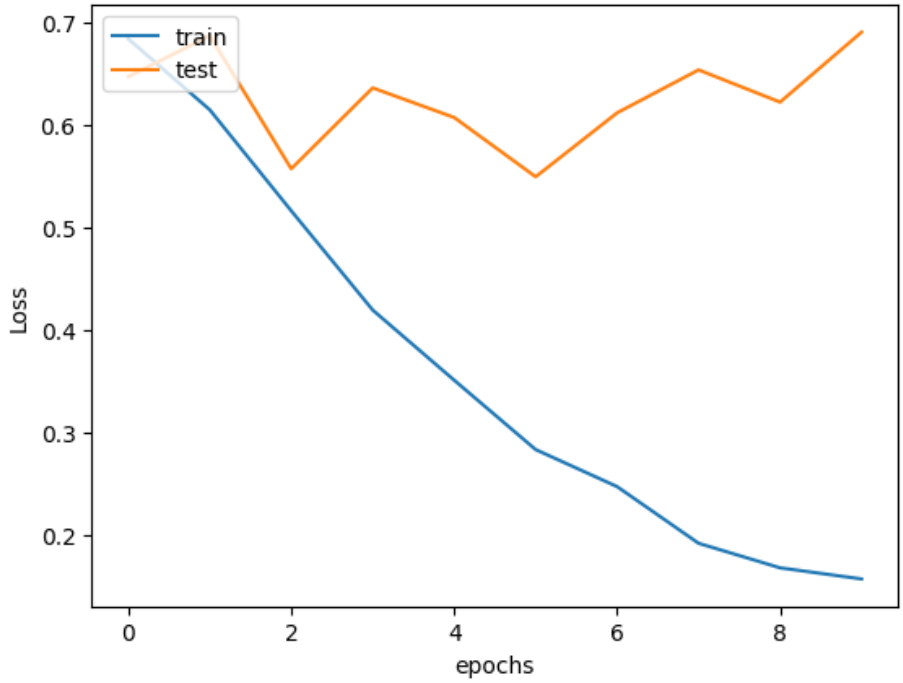


Figure 34: Loss for the MuRIL finetuned model

Table 8: Table summarizing all experiments

Experiment	Precision	Recall	F1-score
fastText and Logistic Regression	0.38	0.66	0.62
fastText and Decision Tree	0.39	0.84	0.63
GloVe and Logistic Regression	0.33	0.58	0.57
GloVe and Decision Tree	0.28	0.60	0.51
DistilBERT	0.50	0.65	0.70
MuRIL	0.55	0.66	0.73

CHAPTER 5

Conclusion and Future Work

This chapter concludes this work by summarizing the best of approaches discussed in this work. Since, a lot more needs to be done in the area hate speech detection for Indian languages, there are lot of directions in which the future research can progress.

5.1 Conclusion

We have adopted a variety of methods in this project to identify hate speech for Roman Hindi. We have used embeddings like fastText, GloVe and transformer based embeddings like distilBERT and MuRIL. We applied classical machine learning models as well as deep learning for classification. The best performing approach seen in this work was the MuRIL embedding combined with the BERTforClassification model. Overall, the application of class weights and the low learning rate give a macro F1 score in the range of 70 - 75 %.The DistilBERT based approach also gave comparable results. The incorrectly classified test samples were examined after the DistilBERT experiment finished.The test samples that we examined contained a large number of tweets from the second wave of COVID that hit India. People took to twitter to express their feelings on the crisis going on in the country. It was found that a large number of the examples were just harsh criticisms and it can be seen that the model did not classify such harsh criticism as hate. Hate is very subjective and for that reason harsh criticism was labeled as hatred in the dataset. There were tweets in which hate was expressed sarcastically. The model was not able to sense sarcasm that was hateful. Some tweets did not contain the name of the person being targeted but rather a veiled reference to the person which can be hard to figure out and requires lot of background knowledge which the model does not have. There were also some tweets about political issues and just lacked enough information about the issue to make any sort of classification. The dataset has a class imbalance and

the application of class weights results in false positives for both the transformer based experiments. False positives is an issue even in the fastText-GloVe experiments because of oversampling.The above reasons pose as barriers to the transformer based model's learning.

5.2 Future Work

The language diversity of India adds lot of complexity to the problem of hate speech detection. Even the languages shown in the map of India show lot of variation from region to region. Language models should be trained to capture those variations as well. There is lot of scope for improvement for standardizing Hindi codes-switched with English. There is already research progressing in this area [31, 32].Standardizing Hindi code-switched with English is necessary because when Hindi is written in the Latin script then there might be different spellings for the same word and this hurts the performance of any model that is trying to classify speech. More and more transformer models with newer and better classification heads can be trained on roman transliterated Hindi data. The challenges mentioned in the previous section can be mitigated by training transformer based models on larger datasets with examples of sarcastic-hate and veiled references.

LIST OF REFERENCES

[1] T. Mandl, S. Modha, G. K. Shahi, H. Madhu, S. Satapara, P. Majumder, J. Schäfer, T. Ranasinghe, M. Zampieri, D. Nandini, *et al.*, ''Overview of the hasoc subtrack at fire 2021: Hate speech and offensive content identification in english and indo-aryan languages,'' *arXiv preprint arXiv:2112.09301*, 2021.

[2] M. Bilal, A. Khan, S. Jan, S. Musa, and S. Ali, ''Roman urdu hate speech detection using transformer-based model for cyber security applications,'' *Sensors*, vol. 23, no. 8, p. 3909, 2023.

[3] M. Das, S. Banerjee, and P. Saha, ''Abusive and threatening language detection in urdu using boosting based and bert based models: A comparative approach,'' *arXiv preprint arXiv:2111.14830*, 2021.

[4] M. M. Khan, K. Shahzad, and M. K. Malik, ''Hate speech detection in roman urdu,'' *ACM Transactions on Asian and Low-Resource Language Information Processing (TALLIP)*, vol. 20, no. 1, pp. 1--19, 2021.

[5] A. Velankar, H. Patil, A. Gore, S. Salunke, and R. Joshi, ''Hate and offensive speech detection in hindi and marathi,'' *arXiv preprint arXiv:2110.12200*, 2021.

[6] D. Kakwani, A. Kunchukuttan, S. Golla, N. Gokul, A. Bhattacharyya, M. M. Khapra, and P. Kumar, ''Indicnlpsuite: Monolingual corpora, evaluation benchmarks and pre-trained multilingual language models for indian languages,'' in *Findings of the Association for Computational Linguistics: EMNLP 2020*, 2020, pp. 4948--4961.

[7] A. Velankar, H. Patil, A. Gore, S. Salunke, and R. Joshi, ''L3cube-mahahate: A tweet-based marathi hate speech detection dataset and bert models,'' *arXiv preprint arXiv:2203.13778*, 2022.

[8] ''Mahabert,'' https://huggingface.co/l3cube-pune/marathi-bert, accessed: 2022-03-30.

[9] M. Das, P. Saha, B. Mathew, and A. Mukherjee, ''Hatecheckhin: Evaluating hindi hate speech detection models,'' *arXiv preprint arXiv:2205.00328*, 2022.

[10] I. Jadhav, A. Kanade, V. Waghmare, and D. Chaudhari, ''Hate and offensive speech detection in hindi twitter corpus,'' in *Forum for Information Retrieval Evaluation (Working Notes)(FIRE), CEUR-WS. org*, 2021.

[11] S. Shukla, S. Nagpal, and S. Sabharwal, ''Hate speech detection in hindi language using bert and convolution neural network,'' in *2022 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*. IEEE, 2022, pp. 642--647.

[12] T. Mikolov, K. Chen, G. Corrado, and J. Dean, ''Efficient estimation of word representations in vector space,'' *arXiv preprint arXiv:1301.3781*, 2013.

[13] L. Weng, ''Learning word embedding,'' https://lilianweng.github.io/posts/2017-10-15-word-embedding/, 2017.

[14] J. Pennington, R. Socher, and C. D. Manning, ''Glove: Global vectors for word representation,'' in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532--1543.

[15] A. Jha, ''Vectorization techniques in nlp [guide],'' https://neptune.ai/blog/vectorization-techniques-in-nlp-guide, 2023.

[16] K. Venugopal, ''Mathematical introduction to glove word embedding,'' https://becominghuman.ai/mathematical-introduction-to-glove-word-embedding-60f24154e54c, 2021.

[17] A. R. G. Devjyoti Chakrobarty, ''An introduction to the global vectors (glove) algorithm,'' https://wandb.ai/authors/embeddings-2/reports/GloVe--VmlldzozNDg2NTQ, 2021.

[18] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, ''Enriching word vectors with subword information,'' *Transactions of the association for computational linguistics*, vol. 5, pp. 135--146, 2017.

[19] A. Chaudhary, ''A visual guide to fasttext word embeddings,'' https://amitness.com/2020/06/fasttext-embeddings/, accessed: 2022-03-30.

[20] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, ''Bert: Pre-training of deep bidirectional transformers for language understanding,'' *arXiv preprint arXiv:1810.04805*, 2018.

[21] M. Phi, ''Illustrated guide to transformers- step by step explanation,'' https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0, accessed: 2022-03-30.

[22] J. Alammar, ''The illustrated transformer,'' https://jalammar.github.io/illustrated-transformer/, 2021.

[23] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, ''Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter,'' *arXiv preprint arXiv:1910.01108*, 2019.

[24] ''Distillation of bert-like models: The theory,'' https://towardsdatascience.com/distillation-of-bert-like-models-the-theory-32e19a02641f, accessed: 2023-03-30.

[25] S. Khanuja, D. Bansal, S. Mehtani, S. Khosla, A. Dey, B. Gopalan, D. K. Margam, P. Aggarwal, R. T. Nagipogu, S. Dave, *et al.*, ''Muril: Multilingual representations for indian languages,'' *arXiv preprint arXiv:2103.10730*, 2021.

[26] Jitender, ''Implement sigmoid function using numpy,'' https://www.geeksforgeeks.org/implement-sigmoid-function-using-numpy/#article-meta-div, 2023.

[27] A. Navlani, ''Decision tree classification in python tutorial,'' https://www.datacamp.com/tutorial/decision-tree-classification-python, 2023.

[28] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, ''Dropout: a simple way to prevent neural networks from overfitting,'' *The journal of machine learning research*, vol. 15, no. 1, pp. 1929--1958, 2014.

[29] J. L. Ba, J. R. Kiros, and G. E. Hinton, ''Layer normalization,'' *arXiv preprint arXiv:1607.06450*, 2016.

[30] R. Mishra, ''devanagari-to-roman-script-transliteration,'' https://github.com/ritwikmishra/devanagari-to-roman-script-transliteration, 2019.

[31] A. Sharma, A. Kabra, and M. Jain, ''Ceasing hate with moh: Hate speech detection in hindi--english code-switched language,'' *Information Processing & Management*, vol. 59, no. 1, p. 102760, 2022.

[32] K. Mehmood, D. Essam, K. Shafi, and M. K. Malik, ''An unsupervised lexical normalization for roman hindi and urdu sentiment analysis,'' *Information Processing & Management*, vol. 57, no. 6, p. 102368, 2020.

APPENDIX A

APPENDIX B