Hate Speech in Social Media
Hate and Offensive Language Identification
Subtask: H2

Aaditya Gupta
2020552

Charvi Jindal
2020045

Chetan
2020046

Harjeet Singh Yadav
2020561

1 Problem Statement

Social media has seen immense growth over the past decades. The amount of hate speech has also significantly increased on these platforms, which poses a threat to society. This hate speech includes offensive language that can be hurtful, derogatory, and insulting and is usually directed to hurt someone's sentiment. These platforms must identify such hate speech and prevent it from reaching the millions of users of the forum without imposing any rigid censorship. Natural Language Processing is used to tackle the problem and devise ways to automate the process. Our project works along the same lines and uses the HASOC Subtask A dataset and aims to identify and flag hate speech as accurately as possible.

2 Related Works

Following are some of the previous works done on the above problem statement.

2.1 HateCheckHin

A lot of research has been done in the field of hate speech detection using AI and NLP. The current models are evaluated using performance on test data and accuracy. This can result in an incapability of identifying the weakness in the models due to biases present in hate speech detection models.

The paper(Das et al., 2022) evaluates existing models for Hindi hate speech detection and identifies the possible weaknesses in the model.

It uses mBERT as a base model fine-tuned on two datasets and uses Hindi as the base language. Class weights are introduced to emphasise minority classes. HateCheckHIn uses 28 functionalities and 6 different multilingual settings to identify any flaws in existing models.

The paper reveals functional weaknesses in the existing models and concludes that the current models work poorly for multilingual test cases. The paper gives an overview on the increasing need to find ways of identifying hate in multiple languages as people on social media are now not limited to a single language and a mixed version of hindi and english is gaining extreme popularity among the platforms.

2.2 Hindi-English Hate Speech Detection: Author Profiling, Debiasing, and Practical Perspectives

The paper(Chopra et al., 2020) aims to model and analyze low-resource code-switched Hinglish hate speech on social media and text for graph-based author profiling based on linguistic homophily. It uses two datasets for testing and incorporates homophily with Node2Vec. Multiple experiments were performed and reported using combinations of layers like LSTM, convolution layers and attention layers to make a backbone network.

It observes a significant performance increase for the two datasets compared to existing models with bias elimination. It is also noted that a combination of CNN, bidirectional LSTM and attention yielded the best results.

The paper reveals gender, social and religious biases induced into existing models used in analysis and proposes a bias elimination algorithm to decrease its value in the final results.

3 Methodology

We started by downloading the given datasets(The one provided for subtask H2), training and testing from google drive and preprocessing it in a few different ways. Then we tried various models with both contextual and non-contextual embeddings to increase the accuracy of the tweets' predictions in the test dataset.

3.1 Pre-processing

We followed the steps given below for preprocessing.

3.1.1 Google Translate

The tweets with a few words in Hindi or entirely in Hindi were translated to English using the python module googletrans, version 3.1.0. A few sentences weren't completely translated or translated at all, so those sentences in the training dataset were manually translated.

3.1.2 Tweet Preprocessor

Tweet-preprocessor(Shah, 2020) is a tweet-preprocessing library in python. It deals with URLs, Usernames, Mentions, Reserved Words, Emojis etc. We used this on the translated tweets obtained in the previous step.

3.1.3 Tokenisation, Removal of Digits, Stop Words and Punctuations

Using NLTK (Natural Language Toolkit) Library, the remaining preprocessing, on the sentence level, was done. In this step, we removed digits, lower cased the entire text, removed the punctuations and even lemmatized the tokens and saved this and the final preprocessed data in the dataframe and the preprocessed csv file.

3.2 Vectorizer -Tf_IDF and Classifier: Random forest classifier

TF-IDF (Term frequency-inverse document frequency) (Ramadhan, 2021) is used as a vectorizer that uses the number of times a term occurs in the document in the form of a matrix with the number of documents as rows and the number of distinct terms as columns. Inverse document frequency, calculated as the inverse of the number of documents containing a specific term, is used as the weight to reduce the scattered data.

The Random Forest Classifier (Yiu, 2019) is a classification algorithm that uses multiple decision trees as an ensemble. Each tree generates a class prediction, and the class with the most votes becomes the model's prediction. The tree generation is carried out with bagging and randomness to create an uncorrelated forest working on the "wisdom of the crowd" to achieve the most likely prediction.

Random forest classifier is accessible through sklearn in the class sklearn.ensemble.RandomForestClassifier with multiple paramters that can be changed to get different sets of predicitons.

3.3 Vectorizer -Tf_IDF and Classifier: Adaboost algorithm

The Adaboost algorithm arranges the models sequentially in the ensemble and, in each step, tries to boost the weak learners (base model) based on the mistakes of previous models (Veronica, 2020). In the end, Adaboost generates on robust ensemble model where each has a speciality over a part of the problem, and when fit together, they cover all the vital areas of the given problem.

The Adaboost classifier can be accessed in python through sklearn using the class sklearn.ensemble.AdaBoostClassifier with multiple customizable paramters for classifications.

3.4 Vectorizer -Tf_IDF and Classifier:SVM(Support Vector Machine)

SVM (Support Vector Machine) is a supervised machine learning algorithm for classification and regression problems. It uses a decision boundary to segregate n-dimensional space into classes under a hyperplane. SVM uses extreme vectors known as support vectors in the creation of the hyperplane for the best decision boundary.

The SVC in python is accessed through the class sklearn.svm, which contains both SVC and SVR for classification and regression, respectively.

3.5 Vectorizer -all-mpnet-base-v2 and Classifier: SVM(Support Vector Machine)

The all-mpnet-base-v2 is a sentence transformer model that maps sentences and paragraphs to a 768-dimensional dense vector space for clustering and semantic search tasks. The model is available on huggingface and can easily be used with the help of sentence-transformers in python.

3.6 Vectorizer -distil-roberta and Classifier: SVM(Support Vector Machine)

distil-roberta is a distilled version of RoBERTa model(Sayar Ghosh Roy, 2021) which uses all the same training procedures containing a total of 82M parameters and with an average twice the speed of base RoBERTa model. It is available on huggingface and can be used directly with the help of sentence-transformers to generate vector embeddings.

3.7 Vectorizer -BERT and Classifier: GBDT

BERT (Bidirectional Encoder Representations from Transformers) applies bidirectional training to a transformer. The paper claims it has a deeper knowledge of context and flow than single-directional models, as it uses an attention mechanism that learns contextual relations between words. It can also be fine-tuned for tasks such as QA and NER.

GBDT (Gradient boosting decision tree) is an algorithm similar to Adaboost with a larger depth of trees and can be used for both classification and regression tasks and can be used with the help of sklearn from the class sklearn.ensemble.GradientBoostingClassifier.

4 Experimental Results

4.1 Non-contexual Embeddings

4.1.1 Vectorizer -Tf_IDF and Classifier: Random forest classifier

Tf_IDf vectorizer gives non-contextual sentence embedding. This causes the inefficient mapping of vectors with the labels {"HOF", "NOT"}.

4.1.2 Vectorizer -Tf_IDF and Classifier:SVM(Support Vector Machine)

Using SVC classifier gives the best output from all the different classifiers used throughout the project. Therefore, the output F1-score icreased from the previous one.

4.2 Contexual Embeddings

4.2.1 Vectorizer -all-mpnet-base-v2 and Classifier: SVM(Support Vector Machine)

We can see that this pair is written twice in the table with two different sets of accuracies. The original accuracy took a jump from earlier high 50(from non-contexual) to 76. The reason being that "all-mpnet-base-v2" is a contextual sentence transformer and hence gives a better relation between the tags and the vectors. Further, this score was improved when the data was pre-processed in a better way. Earlier we had removed all the hashtags, which took away a lot of semantics, but now along with some other modifications, we kept the hashtags, and hence obtained a score of 77.

4.2.2 Vectorizer -distil-roberta and Classifier: SVM(Support Vector Machine)

Using distil-roberta still was better than any non-contexual embedding, however it didnt perform as well as all-mpnet-base-v2. This was expected behaviour as described in the sentence-transformers documentation.

4.3 Contextual embeddings without pre-translation

4.3.1 Vectorizer xlm-roberta-base and Classifier: SVM(Support Vector Machine)

Here we obtained a score of 66. Instead of translation from Hindi to English using google translate, we passed the original data into the Cross-lingual language model. This however performed poorly as compared to previous models. Probably indicating the google-translate's machine translation is better than this direct cross-lingual classification.

5 Analysis and Final comments

From the different models and techniques we have used so far, we observed that pre-processing the data by (1)Translating Hindi tweets to English using Google translate, (2)Adding the keywords after segmenting ¨" hashtags, (3)removing stop words, URLs, emojis, and tags produced better features. Furthermore, using contextual sentence transformer, specifically, "all-mpnet-base-v2," gave the best feature vectors. From a range of ML and DL classifiers used, SVC gave the best accuracies and F1-score.

References

M. Das, P. Saha, B. Mathew, and A. Mukherjee, "Hatecheckhin: Evaluating hindi hate speech detection models," 2022. [Online]. Available: https://arxiv.org/abs/2205.00328

S. Chopra, R. Sawhney, P. Mathur, and R. Ratn Shah, "Hindi-english hate speech detection: Author profiling, debiasing, and practical perspectives," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 01, pp. 386–393, Apr. 2020. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/5374

P. Shah, "Basic tweet preprocessing in python," 2020.

L. Ramadhan, "Tf-idf simplified," 2021.

T. Yiu, "Understanding random forest," 2019.

A. Veronica, "Understanding adaboost and scikit-learn's algorithm," 2020.

T. Z. V. V. Sayar Ghosh Roy, Ujwal Narayan, "Leveraging multilingual transformers for hate speech detection," 2021. [Online]. Available: https://arxiv.org/pdf/2101.03207.pdf

https://scikit-learn.org/stable/:
SKLearn documentation

6 Contribution

Harjeet and Charvi : Pre-processing and making report
Aaditya and Chetan : Resource reading, running different models

Table 1: Macro F1 score comparisons

Vectorizer	**Classifier**	**F1-macro**
`Tf_IDF`	Random forest	0.48
`Tf_IDF`	ADA	0.32
`Tf_IDF`	SVM	0.50
`all-mpnet-base-v2`	SVM	0.76
`distil-roberta`	SVM	0.74
`BERT`	GBDT	0.72
`all-mpnet-base-v2`	**SVM**	**0.77**
`all-mpnet-base-v2`	MLP	0.70
`xlm-roberta-base`	SVM	0.66