Assignment Automatic Classification of Hate Speech (Ensemble Methods)

Tanya Kaintura, Payanshi Jain, Rishikesh Narayanan Ramachandran, Antonios Georgakopoulos

November 16, 2023

Link to Jupyter Notebook
Link to Model Files

1 Introduction

The goal of this assignment is to apply and compare ensemble methods[1] for hate speech detection. This involves developing three different ensemble approaches using multiple models and then analyzing their performance. A quantitative and a qualitative error analysis in the in-domain and the cross-domain setups should also be performed in order to gain better understanding on the challenges of hate speech and assess whether ensemble methods are more efficient than individual models.

Hate speech refers to speech that either directly targets or encourages hostility toward a group or a specific member of that group based on their characteristics related to identity, including factors like ethnicity, religion, and sexual orientation[2, 3]. Hate speech has found an even faster way to spread due to the numerous platforms provided by the internet, allowing it to circulate more rapidly than ever before. Although some mechanisms have been put in action in order to restrict and limit the hate speech text in the internet, its persistence and difficulty of detecting it remains a significant concern. The motivation behind this research is not merely personal but stems from the broader context of addressing a pressing societal problem. Hate speech's proliferation on the internet poses significant challenges for content moderation and community safety. The ability to effectively detect hate speech has implications for various fields, including natural language processing, machine learning, and online platform governance. Researchers in these domains are interested in understanding how ensemble methods can improve hate speech detection, as this knowledge can be applied to develop more efficient and reliable content moderation systems.

The primary research question of this study revolves around the effectiveness of ensemble methods in hate speech detection, particularly in both in-domain and cross-domain scenarios. The hypothesis to be tested is whether the integration of multiple models through ensemble techniques can enhance hate speech detection accuracy, reduce false positives, and improve the model's generalization to different domains and contexts.

In cross-domain scenarios, where the data comes from different sources or platforms, the challenge lies in ensuring that a model trained on one domain can effectively generalize to another. Existing hate speech detection solutions primarily rely on supervised learning, which depends on the quality and quantity of labeled data. Labeled datasets are often small due to the cost, thus limiting experiments on model generalization and scalability[4]. Hate speech can vary in language, context, and expressions across domains, making it difficult to detect with a single model.

Ensemble learning methods involve creating a composite model using a foundational learner. Initially, this foundational learner is used to train multiple models on adjusted training data. These models are then integrated into a single classifier that makes predictions by aggregating the decisions of its constituent models. The use of ensemble methods is of great importance for detecting hate speech in cross-domain settings because they can significantly improve the accuracy and reduce false positives in hate speech detection models. By combining multiple models, each with its strengths and weaknesses, ensemble methods can ultimately make more accurate predictions[5, 6, 7]. The ensemble approach may also reduce overfitting as well as boost the overall performance[8]. This approach can help reduce the risk of false positives, which are instances where non-hateful content is mistakenly classified as hate speech. Ensemble methods can incorporate additional features and information that go beyond the text itself, such as the number of emotion-conveying words, personal pronouns, and message length. These features can provide valuable context and help improve the accuracy of hate speech detection. Ensemble methods effectively harness the strengths of different models and feature sets, making them a valuable strategy for addressing the challenges of detecting hate speech in diverse and evolving online environments.

The paper is structured to provide a clear and logical progression of the research. It begins with this introduction, which establishes the context and motivation for the study. The subsequent sections will delve into the related work, data, experimental setup, and Methods and techniques including a detailed analysis of the ensemble techniques employed. The paper will then present both quantitative and qualitative error analyses in both in-domain and cross-domain settings to assess the performance of the ensemble methods. Finally, the discussion and conclusion sections will synthesize the findings and their implications for hate speech detection.

2 Related work

There have been multiple attempts to deal with the issue of domain adaptation when it comes to hate speech detection. Approaches using traditional machine learning models[9] do not generalize very well to different domains even when they are trained on big datasets. Transformer models like Bidirectional Encoder Representations from Transformers(BERT)[10] show greater promise in the cross-domain abusive language detection problem due to their ability to transfer knowledge from one domain dataset to other domains. However, due to datasets' inherent biases, cross-domain classification still remains a challenge even for these models[11]. Ensemble models have shown great overall performance in the field of machine learning as well as in other fields[12]. Ensemble techniques, like stacking and bagging, are frequently employed in the domain of text classification. These methods are popular strategies that aim to enhance the performance and reliability of text classification models[13].

3 Data

3.1 Exploratory Data Analysis

The objective of this assignment is to leverage the datasets that have been provided to us for the purpose of analysis and classification. More specifically, we have to identify whether or not a certain amount of tweets contains offensive language. We will be working with a simplified preprocessed version of the OLID dataset[14] that contains 5852 entries with each one having been assigned the label of '1' if it is deemed offensive or the label of '0' otherwise. Similarly, the HASOC dataset[15] has been preprocessed in a manner identical to that applied to the OLID dataset. This assignment also requires us to build and train various different models in order to evaluate their performance in both in-domain and cross-domain scenarios.

Table 1: Dataset Statistics

Dataset	Number of Instances	Class Distribution
OLID	5852	Non-hateful: 3591 Hateful: 2261
HASOC	5852	Non-hateful: 3591 Hateful: 2261

An analysis regarding the distribution of the offensive and non-offensive tweets in both datasets was performed and it yielded a result of 61.36% of the total tweets being non-offensive and 38.63% of them being offensive, for both datasets. This can be seen in figure 2. This result indicates that both datasets are mildly unbalanced and therefore, it is important to take steps to address this imbalance before training an NLP model on this data.

In addition to examining the percentages of offensive and non-offensive classes, it's crucial to delve deeper into the fundamental characteristics of the text data. More specifically, we can start by analyzing the size of each tweet within both datasets. This sentence length analysis serves a dual

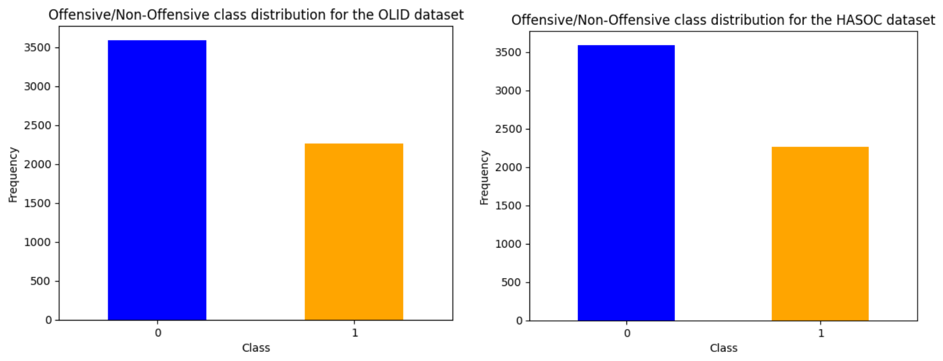


Figure 1: Distributions of Offensive and Non-Offensive classes.

purpose. It does not only help us gain a deeper understanding of the data we are dealing with, but also it provides valuable insights into the preprocessing steps we need to undertake to enhance the performance of our algorithms. By assessing the tweet sizes, we can identify potential challenges related to text length, which might impact the effectiveness of our natural language processing algorithms. For instance, by examining the size of the texts available it can help us decide whether or not we should perform a padding technique that could be incredibly valuable, especially for deep learning algorithms[16]. The figures below illustrate an analysis of the text sizes for both datasets. The histograms show that in the OLID dataset the tweets' generally ranges between 20 and 150 characters with the maximum tweet size being 560 characters. On the contrary, in the HASOC dataset we can see that the longest tweet has 968 characters and the data exhibits less skewness with almost all the tweets ranging from 10 to 300 characters in terms of their size.

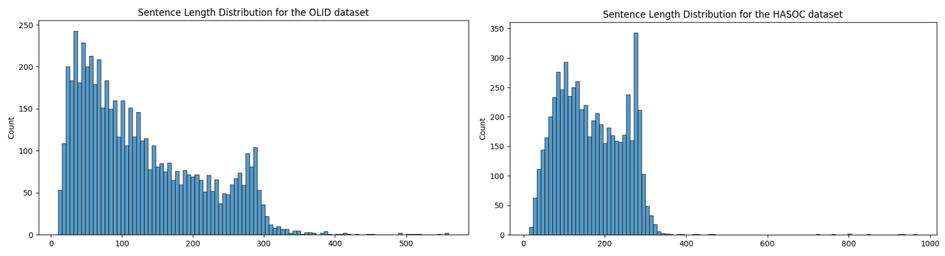


Figure 2: Distributions of text size in both datasets.

We can also move to data exploration at a word-level and check the average word length of each sentence in both datasets. The two plots below show that the average word length roughly falls between the range of 3 and 7 words for both the OLID and the HASOC dataset.

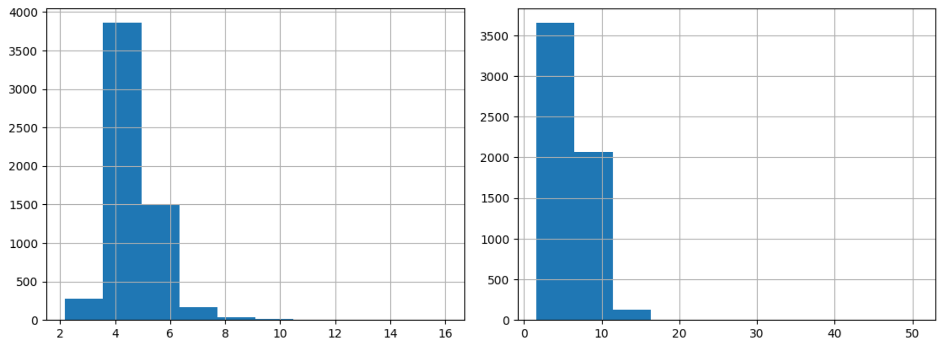


Figure 3: Average word length for both datasets.

It is important to note that this low range could be a result of too many stopwords, such as "the", "a", "an" etc., being used in the tweets. We can remove the stopwords using the NLTK library [17] and try to perform the same analysis once more so we can get a more accurate representation of the average word length. The result of removing the stopwords can be seen in figure 4. The HASOC data shows the bigger change when it comes average word length while the OLID shifts very slightly to the right.

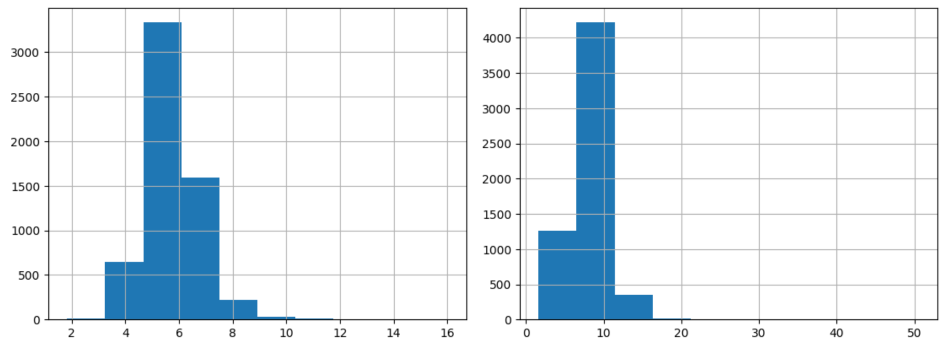
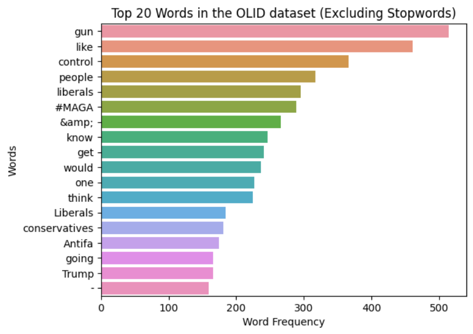


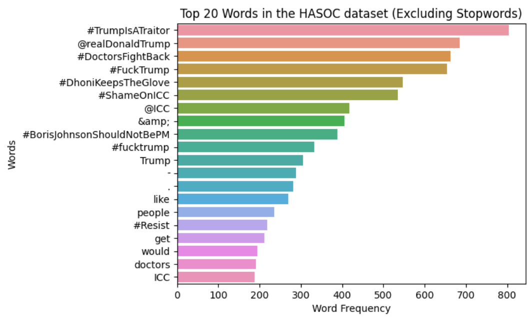
Figure 4: Average word length for both datasets without stopwords.

By analyzing the frequency of the top words in each dataset we can gain a better understanding of the overall context of the comments in the datasets and if there is any correlation between them. We can also understand what are the most prevalent topics discussed in Twitter as well as some common language patterns and trends. The figures 5a, 5b below display the top 20 words present in both datasets after removing stopwords.

In the case of the OLID dataset, it becomes evident that the content of the tweets mainly revolves around political subjects, with discussions possibly centered around gun control, ideological differences, and various perspectives. Additionally, we observe mentions of specific political figures, such as ”Trump,” indicating that the dataset may contain commentary related to political personalities. This analysis gives us important information about what topics are discussed and how people use language in the dataset.



(a) Top 20 more frequent words in the OLID dataset.



(b) Top 20 more frequent words in the HASOC dataset.

3.2 Data Preparation

Data preparation is an important phase in the data analysis pipeline that involves cleaning and transforming raw data into a format suitable for the machine learning algorithms. The text processing techniques that need to be implemented, vary greatly in accordance with the task presented as well as the specific algorithms that are going to be used. As seen in the ”Exploratory Data Analysis” part both datasets (OLID, HASOC) contain a lot of diverse twitter comments regarding not only the content of each post but also its length. This diversity makes a thoughtful approach to data preparation necessary, as different algorithms may benefit from distinct preprocessing steps.

Ensuring the best possible combination of data preparation processes for each algorithmic approach is paramount. For instance, deep learning algorithms may require extensive text preprocessing, including tokenization, stopword removal, and word embeddings, to effectively capture nuanced semantic information. In contrast, more state-of-the-art approaches like Transformers often demand a different set of data preparation techniques as shown below 3.2.1. Transformers are highly capable of handling unstructured text data with minimal preprocessing due to their self-attention mechanisms.[18] Therefore, the choice of data preparation methods must align with the specific characteristics of the data and the requirements of the machine learning algorithms in use. A tailored approach to data preparation not only enhances model performance but also maximizes the utility of the insights extracted from these datasets.

3.2.1 Transformers - Dynamic Padding

The Transformer architecture is especially well-suited for the initial training phase on extensive text datasets. This approach results in significant improvements in accuracy when applying the model to subsequent tasks, such as text classification[19]. While extensive preprocessing may not be required for the Transformers models, there are still important considerations to ensure the best possible model performance. Given that our datasets contain text with different size, the algorithms cannot process the batches of this diverse data and thus we need to ensure that our input data will have the same length. Two methods can be used for addressing this particular issue, padding and truncation. They are techniques that enable the processing of sequences of different lengths, making it feasible to train and deploy models on data where text lengths can vary significantly. The pre-trained Transformers models use mainly three main types of tokenizers which are the SentencePiece, the WordPiece[20] and the Byte-Pair Encoding[21]. In our experiments we will use more specialized tokenizers that complement very well each Transformer model that we are using.

Padding involves adding extra tokens, often represented by a special token like [PAD], to the end of shorter sequences. This ensures that all sequences have the same length, which is crucial for processing them efficiently in batches. These added tokens have no actual meaning but serve to fill the gaps in shorter sequences and thus making the input data have the same size. On the other hand, truncation involves cutting off the end of longer sequences to match the length that we desire. This can be very helpful for the model that we want to train, because it helps reducing the computational resources used for processing the data and also can assist the model in reducing its workload. However, in the case of truncation the maximum length of each batch should be carefully considered in order to retain as much information and thus give the opportunity for the model to learn from the data.

Padding plays a pivotal role in the context of Transformers for several reasons. Firstly, Transformers necessitate fixed-length input sequences, whereas real-world text data is often of varying lengths. Padding becomes crucial to ensure uniformity in input size, enabling efficient batch processing and harnessing the power of GPUs. Moreover, due to the limitation that the Transformers model have in their input sequences length, we also applied truncation in the batches that contained more than the maximum tokens in length. The maximum sequence length for each model is displayed in the table2 below. When padding each sentence to this maximum limit, the issue of memory consumption becomes evident. This issue can be addressed by sorting the data based on sentence length, one can create batches of similar-length sequences. This method optimizes memory usage and computation efficiency. During training, the model leverages the maximum length of the current batch rather than the global maximum, allowing for dynamic adjustments to memory allocation based on the specific batch's needs. This approach not only mitigates the excessive memory problem but also enhances the overall efficiency and performance of Transformer models in handling varying-length input data.

Transformer Model	**Max. input tokens**
HateBERT	512
fBert	512

Table 2: Maximum length of input tokens per Transformer model

4 Experiments

In our assignment we tested and evaluated the performance of the different models in both in-domain and cross-domain scenarios. The two different dataset are OLID and HASOC. The models were trained twice once on the OLID dataset and then on HASOC dataset and then tested on OLID testset. The OLID and HASOC datasets differ notably in terms of domain. OLID primarily consists of English language text data from social media platforms, with a focus on user-generated content, informal language, and detailed annotation for various aspects of offensive language. In contrast, HASOC encompasses a broader range of Indo-European languages and sources, including social media, news articles, and blogs, introducing linguistic diversity and varied contextual domains.

In-domain text classification involves how well can a model classify text data within a specific subject area. The training set and the test set contain data of the same type and the same domain. Usually this task is more straightforward and thus the models show better performance in the in-domain settings. The models was trained on OLID train dataset (olid-train-small.csv) and tested on OLIDv1 test dataset (olid-test.csv).

Cross-domain experiments, on the other hand, involve training the models on data from one domain and evaluating their performance on a different domain. In your case, you trained the models on the HASOC train dataset (hasoc-train.csv) and evaluated them on the OLIDv1 test dataset (olid-test.csv), which is from a different domain. The aim here was to test the models' ability to generalize across domains, a more challenging task for the traditional classification algorithms[22].

Cross-domain experiments are challenging for several reasons:

1. **Domain Shift:** Data from different domains often have variations in language, style, and content, making it more difficult for models to generalize effectively.

   Example: Models trained on the diverse and multilingual content of HASOC, which includes text in Hindi, German, and other languages, may struggle to adapt when they are tested on OLID, which primarily consists of English language social media text. The domain shift from multiple languages to a single language can result in difficulties in understanding the nuances of the English content in OLID.

2. **Vocabulary Differences:** Vocabulary can vary significantly between domains, affecting the model's ability to accurately interpret and classify text.

   Example: Models trained on the formal language used in German news articles from HASOC may not be well-equipped to handle the informal, slang-ridden language frequently found in English tweets within OLID like "u" for "you" or "4" for "for" etc. This vocabulary difference can lead to misclassifications in OLID, where colloquial language is prevalent.

3. **Training Data Mismatch:** Cross-domain experiments often involve a mismatch between the distribution of training data and the target domain, potentially leading to suboptimal performance.

   Example: When a model is trained on the diverse content of HASOC, it may not be well-versed in the specific linguistic and content characteristics of OLID, leading to suboptimal performance due to the mismatch in training data distribution.

4. **Transfer Learning:** Models trained in one domain may struggle to adapt well to another domain, necessitating more advanced techniques like transfer learning.

   Example: Models initially trained on the multilingual and diverse HASOC dataset may require advanced transfer learning techniques to fine-tune their performance for OLID. This adaptation is necessary for them to effectively classify offensive language in the more narrow domain of English social media text.

5 Methods

In this section we will elaborate on the three models chosen from Assignment 4 for ensembling in detail and then explain the ensembling techniques used. SVM was chosen as it performed best along with HateBERT and fBERT as both models are specifically trained on hate speech and performed reasonably well. SVM is known for its ability to create clear decision boundaries between different classes in the data, making it effective for binary classification tasks like hate speech detection. HateBERT is a specialized BERT-based model specifically trained for detecting abusive language, particularly hate speech, in English text. It is trained on a curated dataset, making it adept at recognizing offensive and harmful content. fBERT is another BERT-based model fine-tuned with a focus on hate speech detection. It is trained on a dataset consisting of over 1.4 million offensive instances from the SOLID dataset. This fine-tuning process equips fBERT with a deep understanding of domain-specific features and offensive language nuances found in social media content. The ensembling techniques used are designed to leverage the unique strengths of each of these models. By combining SVM, HateBERT, and fBERT in the ensemble, we aim to create a powerful and robust system for hate speech detection.

5.1 Models

5.1.1 HateBERT

HateBERT[23] is a specialized BERT model trained for detecting abusive language, particularly hate speech, in English text. The training data for HateBERT comes from a dataset called RAL-E, which consists of Reddit comments from communities that were banned for offensive, abusive, or hateful content[24]. The researchers curated this dataset and made it publicly available for the purpose of developing and evaluating HateBERT. HateBERT outperforms the general pre-trained BERT model in detecting offensive, abusive language, and hate speech in three different English datasets. This indicates that fine-tuning a language model like BERT on specific datasets related to abusive language can lead to better performance in identifying such content which is why we chose to implement HateBERT.

The implementation starts by loading the training and testing data and initializing the HateBERT tokenizer. Data preprocessing involves sorting the text data by length for efficient training. A custom dataset class is created to format the data for compatibility with HateBERT. The model itself is loaded using the 'GroNLP/hateBERT' checkpoint for sequence classification, specifying the number of labels as 2. A data collation function is defined to handle batch processing during training, ensuring uniform sequence lengths. Once the model and data are prepared, training parameters are configured using the TrainingArguments. These parameters encompass batch size, the number of training epochs, the evaluation strategy, logging settings, and directory paths for saving both the model and the training results. In this implementation, the batch size is set to 8, and the number of training epochs is configured to 3, a decision influenced by limited GPU resources and time constraints. The training process is managed by the Trainer object. It takes care of both training and evaluation phases, and it is configured with the initialized model, training arguments, and the pre-processed training and evaluation datasets. The data collation function is also specified to ensure efficient batch processing during training. Hyperparameter tuning is an essential part of model development. To optimize HateBERT's performance, hyperparameter tuning is carried out using the Optuna library. Specifically, the hyperparameters being tuned include the learning rate and the number of training epochs. These adjustments help fine-tune the model's training process and enhance its ability to detect abusive language. Once the model has been fine-tuned using the training data, it is evaluated on the test dataset. Evaluation metrics are computed to assess the model's performance. The trained model and tokenizer are saved for future use, and they can be subsequently loaded for testing on the OLID dataset, which helps in measuring its effectiveness. All the models ran for 3 epochs and 8 batch size because of the limited gpu resources and time contraint.

5.1.2 fBERT

Transformer-based models, such as BERT, ELMO[25], and XLM-R[26] have significant impact in advancing natural language processing tasks, particularly in the domain of identifying offensive language and hate speech on social media platforms. While these models have really showed excellent performance in various NLP tasks, they are typically trained on general text corpora and may lack domain-specific cues, particularly those related to offensive language. To address this knowledge gap, the model fBERT[27] has been created as it is a specialized model based on "bert-base-uncased" fine-tuned with over 1.4 million offensive instances from the SOLID dataset. This fine-tuning process enables the model to better capture domain-specific features and offensive language nuances found in social media. fBERT outperforms other models like BERT[10] and HateBERT[23] in tasks such as OffensEval and HatEval, demonstrating the potential of domain-specific fine-tuning for improved performance in identifying offensive language and hate speech which is why we chose to model fBERT and HateBERT.

The implementation of fBERT closely resembles that of HateBERT, emphasizing key steps in the process. The initial stage involves data loading, where training and testing datasets are imported. The training data is essential for model fine-tuning, while the testing data is used for evaluation. These datasets are loaded from CSV files, forming the basis for the subsequent training and assessment phases. Following data loading, the fBERT tokenizer is initialized. The tokenizer is a fundamental component for encoding text data into numerical representations suitable for model processing. The 'diptanu/fBERT' tokenizer is chosen, ensuring compatibility with the fBERT model from hugging face[28], which has been specifically fine-tuned for the domain of offensive language. Tokenization for fBERT utilizes the fBERT tokenizer due to variations in vocabularies and potential special tokens or rules that may exist between different models. This choice ensures compatibility, as these differences can arise from varying training or fine-tuning processes. Effective data processing is paramount during training. Sorting text data by length groups similar-length sequences, minimizing the need for extensive padding and bolstering training efficiency. Data preparation involves creating a custom dataset class that manages text data and corresponding labels, streamlining data retrieval and manipulation. This class incorporates methods like __getitem__ and __len__ to ensure proper dataset formatting for training. The fBERT model, designed for sequence classification, is initialized. It is loaded from the 'diptanu/fBERT' checkpoint and configured for binary classification with two labels. The collation function computes batch-specific maximum sequence lengths, tokenizes text using the chosen tokenizer, and applies padding to maintain uniform sequence lengths, optimizing training efficiency. Once the model and data are prepared, training parameters are configured using the TrainingArguments. These parameters include batch size, the number of training epochs, evaluation strategy, logging settings, and directory paths for saving the model and training results. The batch size is set to 8, and the number of training epochs is configured to 3. The training process is managed by the Trainer object. It takes care of both training and evaluation phases and is configured with the initialized model, training arguments, and the formatted training and evaluation datasets. The data collation function is specified for efficient batch processing during training. Following successful training, the model is evaluated on the test dataset. The evaluation results are printed, and metrics such as precision, recall, and F1-score are calculated to assess the model's performance. A confusion matrix is generated to provide insights into the model's behavior on the test data. Once the model is fine-tuned and evaluated, it is saved to a specified directory for future use. This allows the model to be loaded without the need for retraining, ensuring the availability of a well-tuned model for identifying offensive language in social media text.

5.1.3 SVM

Support Vector Machines (SVM)[29] are a class of supervised machine learning algorithms used for both classification and regression tasks. They are particularly suitable for hate speech detection for several reasons. SVMs excel in handling high-dimensional data, making them well-suited for processing text-based data that is common in social media platforms. They can effectively separate data points into different classes by finding the optimal hyperplane that maximizes the margin between them. This margin maximization allows SVMs to generalize well to unseen data and effectively discriminate between hate speech and non-hate speech, a crucial aspect of content moderation. SVMs can handle imbalanced datasets, which is often the case in hate speech detection where non-hate speech data might dominate. Additionally, SVMs can accommodate various kernel functions, enabling them to capture complex relationships in the data. All these factors make SVMs a valuable tool for hate speech detection tasks, as they offer a balance between accuracy and interpretability, allowing for robust and reliable results in content moderation efforts.

Parameter	**Value**
Kernal	Linear
Max Feature	4000
Stop Word	English

Table 3: Parameters of SVM

The implementation begins by loading and preprocessing the training data, converting text to numerical features with TF-IDF vectorization, and splitting the data into training and testing sets. The Support Vector Classifier (SVM), classification model presented make use of the Term Frequency-Inverse Document Frequency (TF-IDF) technique through the "TfidfVectorizer" from scikit-learn to transform raw text data into a weighted matrix representation with maximum features as 4000 (based on trial and error). This vectorization allows the model to capture the significance of terms relative to the entire corpus. With the "stop_words" parameter set to "english", common English words, which typically don't offer much discriminatory information for many NLP tasks, are excluded from the matrix. This removes potential noise from the data. Then, the raw text data undergoes a transformation into a matrix where each row represents a document and each column corresponds to a term from the vocabulary learned from the corpus. The values in the matrix are the TF-IDF weights, effectively standardizing variable-length text data into consistent numerical representations. An SVM classifier with a linear kernel and default parameters is trained on the training data and evaluated on the test data, with the classification report providing performance metrics. The trained SVM model is then saved for future use. The overview of the parameter is as shown in table 3.

5.2 Ensemble Methods

Ensemble techniques in machine learning are pivotal strategies that harness the collective strength of multiple models to enhance predictive accuracy and robustness. They involve aggregating predictions from diverse models, each offering a unique perspective on the data. In this assignment, we explore three distinct ensemble methods. The first, hard majority voting, relies on the simple principle of "majority rules" to determine the final prediction, making it suitable for models with differing biases. Soft majority voting, our second technique, takes into account the confidence levels or probability estimates provided by models, allowing for more nuanced decision-making. Finally, stacking ensembles introduce a higher level of sophistication by training a meta-model on top of base models. This meta-model learns how to optimally combine the outputs of base models, resulting in improved overall performance and adaptability. These ensemble strategies are valuable tools for leveraging the collective wisdom of multiple models to achieve better predictive outcomes.

5.2.1 Hard Voting

Hard majority voting is a straightforward ensemble technique that combines predictions from multiple models to reach a final decision. It operates on the principle of "majority rules." In hard voting, each model participating in the ensemble provides its prediction for a given sample, typically assigning it to one of the available class labels. The final prediction for that sample is determined by taking the majority vote among the individual model predictions. Hard majority voting is particularly effective when the individual models have different biases or tendencies, as it balances out their idiosyncrasies and often leads to more accurate and robust predictions.

In the implementation, the *hard_majority_voting* function takes a list of predictions from different models as its input. Each prediction within this list represents the model's classification decision for a set of samples. These individual predictions are then combined to determine the ensemble's final decision. The code utilizes NumPy to efficiently aggregate the predictions. It transposes the list of predictions to align them with samples and collates the votes for each sample. The majority vote for each sample is determined using the np.bincount function, which counts the occurrences of each class label and selects the one with the highest count as the final prediction. The resulting ensemble predictions are then evaluated by computing classification metrics and generating a confusion matrix to assess their performance.

5.2.2 Soft Voting

Soft majority voting is an ensemble technique that combines predictions from multiple models based on their prediction probabilities. Unlike hard voting, which directly selects the majority class label, soft voting considers the confidence or probability scores associated with each model's predictions. In soft voting, each model participating in the ensemble provides a probability distribution for each sample. The final prediction for each sample is obtained by averaging the probability scores for each class across all models and choosing the class with the highest average probability.

The implementation of soft majority voting, the *soft_majority_voting* function takes a list of probability distributions from different models as its input. These probability distributions represent the models' confidence in assigning each sample to various class labels. To combine these distributions, the code calculates the sum of probabilities for each class label across all models using NumPy. The class with the highest average probability for each sample is then selected as the final prediction. In this way, soft majority voting takes into account not just the number of votes but also the confidence levels expressed through probabilities, resulting in a more refined and nuanced ensemble decision.

5.2.3 Stacking ensemble

Stacking ensemble is a sophisticated model combination technique that leverages the power of multiple machine learning models to achieve superior predictive performance. This approach involves several key components. Firstly, the dataset is divided into multiple folds, typically using k-fold cross-validation, ensuring that each fold serves as both a training and validation set. During each fold iteration, a set of base models, in this case, Support Vector Machine (SVM), HateBERT, and fBERT models, are trained on the training portion of the data. These base models generate out-of-fold (OOF) predictions on the validation set, contributing to a collection of OOF predictions for each model across all folds. These OOF predictions, which are essentially additional features, are combined into a meta-dataset.

In this comprehensive ensemble learning approach, we aim to enhance the classification performance by combining predictions from three distinct models: Support Vector Machine (SVM), HateBERT, and fBERT. This ensemble technique is implemented using k-fold cross-validation and incorporates additional features to create a robust and accurate classification model. The integration of multiple models and supplementary information allows us to harness the strengths of each component for improved predictive accuracy. To initiate this ensemble endeavor, we commence by loading pre-trained models and their corresponding tokenizers for SVM, HateBERT, and fBERT. These models serve as the foundational building blocks of the ensemble, each contributing its unique capabilities and insights into the classification task. Next, we acquire the essential data components. The training and test datasets, encompassing textual data and their associated labels, are loaded into the system. These datasets will serve as the bedrock for training and evaluating the ensemble model. The ensemble methodology is underpinned by k-fold cross-validation, ensuring that the data is divided into training and validation subsets while preserving a balanced distribution of labels. Specifically, we employ a 5-fold cross-validation strategy to facilitate robust model assessment. Within the confines of each fold, the process of generating out-of-fold predictions is initiated for each of the three models. For the SVM model, we apply TF-IDF vectorization to the text data, subsequently training an SVM classifier on the training subset. Predictions are then made on the validation subset and aggregated as out-of-fold predictions. HateBERT and fBERT models follow a similar pattern of tokenizing, formatting the data, and fine-tuning the models on the training subsets. Predictions from these models are also collected as out-of-fold predictions. In addition to the base model predictions, we introduce two crucial additional features for each data sample: token lengths, denoting the number of words in each text, and character lengths, indicating the total number of characters in the text. These features are computed for the test dataset and integrated into the ensemble. The core of the ensemble lies in the meta-model, a logistic regression model that is trained using the out-of-fold predictions from SVM, HateBERT, and fBERT, coupled with the newly introduced additional features. This amalgamation creates an input matrix that offers a fusion of the individual models' predictions and the supplementary data attributes. Subsequently, the trained meta-model is harnessed to make final predictions on the test dataset. These ultimate predictions represent the output of the stacking ensemble, which leverages the collective intelligence of the component models and the enriched dataset to yield a more accurate and robust classification model. This stacking ensemble approach aims to capture the strengths of multiple models and harness their combined predictive power to enhance overall model performance, as shown in the code.

6 Results and analysis

6.1 In Domain Analysis

6.1.1 Quantitative Analysis

In the table 4, which measures the performance of models trained and tested on the OLID dataset, the Stacking Ensemble model clearly exhibits high performance metrics. It garners a Macro precision, Macro Recall and Macro F1 of 0.81, 0.79 and 0.80 respectively. Both the Hard and soft Majority voting models hovers around the 0.50 for Macro precision and Macro Recall, but there is a distinct difference in their MacroF1Scores, with the former achieving 0.39 and the latter 0.48. This suggests that while both models have roughly equivalent precision and recall, the Soft Majority Voting model achieve better balance between two, yielding a slightly better F1 score.

Examining the analysis in table 5, which provides a more deeper view of models performance for individual classes 0 and 1 in OLID dataset, there are intriguing observations. The Hard Majority Voting model displays an inversion in its performance metrics between two classes. For class "0", the precision, recall and F1-score stand at 0.73, 0.24, 0.37 respectively, on the other hand for class "1" they are 0.28, 0.77, 0.41. This inversion suggests a significant challenge in appropriately classifying instances across the two classes. On the other hand, the Soft Majority Voting model performs much better for class "0" with scores of 0.72, 0.86 and 0.78 in precision, recall and F1-score respectively. However, its performance falls considerably for class "1", with the metrics dropping to 0.26, 0.12 and 0.17.

Model	MacroPrecision	MacroRecall	MacroF1
Hard Majority Voting	0.51	0.51	0.39
Soft Majority Voting	0.49	0.49	0.48
Stacking Ensemble	0.81	0.79	0.80

Table 4: In-Domain Analysis: Model trained and tested on OLID dataset

The Stacking Ensemble model shows its dominance with its robust per-class performance. For class '0', it achieves an impressive Precision, Recall, and F1-Score of 0.87, 0.91, and 0.89 respectively. For class '1', it continues its commendable performance with scores of 0.75, 0.66, and 0.70.

Overall, in the context of in-domain analysis on the OLID dataset, the Stacking Ensemble gives a higher accuracy, outperforming both the Hard and Soft Majority Voting models in all considered metrics. While the Majority Voting models show potential, especially the Soft Majority Voting model for class '0', they both exhibit challenges in consistently classifying instances across both classes. The Stacking Ensemble, in contrast, provides a balanced, effective, and high-performing model for the entire dataset, making it the preferred choice for this specific domain.

Model	Label	Precision	Recall	F1-Score
Hard Majority Voting	0	0.73	0.24	0.37
Hard Majority Voting	1	0.28	0.77	0.41
Soft Majority Voting	0	0.72	0.86	0.78
Soft Majority Voting	1	0.26	0.12	0.17
Stacking Ensemble	0	0.87	0.91	0.89
Stacking Ensemble	1	0.75	0.66	0.70

Table 5: Detailed In-Domain Analysis: Model trained tested on OLID dataset

6.1.2 Qualitative Analysis

In context to tables 4,5, Systematically, the Stacking Ensemble model dominates in the aggregate performance, with a high Macro F1 score of 0.80, in contrast to Hard and Soft Majority Voting models, which lag behind significantly, especially in Macro F1 scores. This overarching trend is further observed in class-specific metrics. The hard Majority voting model exhibits a stark precision-recall trade off for both classes, suggesting a high rate of false positives for class "1" and high false negatives for class "0". Conversely, the soft majority voting model manifest high recall but low precision for class 0, suggesting a high rate of false negatives for class "1".

In terms of error sources, the hard majority voting model demonstrates sensitivity to misclassifying which is evident by high true positives of class "0" and false positives for class "1" from table 6. The soft majority voting model's inherent bias leans heavily towards class "0", leading to low values in class "1" cases. Stacking ensemble's errors, although fewer,primarily emerge from slightly compromised precision for class "1".

To solve these discrepancies, for the hard majority voting model, refining class boundaries or decision thresholds can help in achieving a better balance. The soft Majority voting model could

Model	Hard Majority Voting		Soft Majority Voting		Stacking Ensemble	
	Predicted 0	Predicted 1	Predicted 0	Predicted 1	Predicted 0	Predicted 1
Actual 0	151	469	535	85	566	54
Actual 1	56	184	210	30	82	158

Table 6: Indomain Confusion Matrices: Confusion Matrices for Hard Majority Voting, Soft Majority Voting, and Stacking Ensemble

benefit from techniques that can more robustly capture class "1" characteristics, possibly via feature engineering or sampling strategies. For the stacking Ensemble, hyperparameter tuning or architecture adjustment may further optimize its performance

6.2 Cross Domain Analysis

6.2.1 Quantitative Analysis

In context to the table 7, it can be found that the Stacking Ensemble significantly outperforms the two majority voting models across all metrics. The stacking Ensemble significantly outperforms the two majority voting models across all metrics. The stacking Ensemble boasts a Macro Precision, Marco Recall, and Macro F1 0.77, 0.75 and 0.76 respectively. In contrast, both Hard and Soft Majority Voting models have a Macro Precision and Marcro Recall around 0.5, with their and Macro F1Scores being 0.39 and 0.50 respectively. This indicates that the Stacking Ensemble model is notably more effective in balancing precision and recall, consequently resulting in higher F1 score, which is a measure of a test's accuracy.

Model	MacroPrecision	MacroRecall	MacroF1
Hard Majority Voting	0.50	0.50	0.50
Soft Majority Voting	0.50	0.49	0.49
Stacking Ensemble	0.77	0.75	0.76

Table 7: Cross-Domain Analysis: Model trained on HASOC dataset and tested on OLID dataset

To get further insight on the performance of model, the performance is analysed between two classes "0" and "1". From the table 8, it is observed that, the hard majority voting model demonstrates a clear disparity in performance between the two classes with a Precision, Recall and F1-score of 0.72 for class 0 and just 0.28 for class 1.

Model	Label	Precision	Recall	F1-Score
Hard Majority Voting	0	0.72	0.72	0.72
Hard Majority Voting	1	0.28	0.28	0.28
Soft Majority Voting	0	0.72	0.61	0.66
Soft Majority Voting	1	0.27	0.38	0.32
Stacking Ensemble	0	0.85	0.89	0.87
Stacking Ensemble	1	0.68	0.61	0.64

Table 8: Detailed Cross-Domain Analysis: Model trained on HASOC dataset and tested on OLID dataset

Similarly, the Soft Majority Voting model shows a distinction in its performance between the two classes. For class '0', the precision is 0.72, Recall is 0.61 and F1-score is 0.66. For class "1", these scores drop to 0.27,0.38 and 0.32 respectively. This shows that the model is proficient in predicting class "0" while facing difficulties with class "1".

The Stacking Ensemble, consistently with its high macro scores, and high other models in per-class metrics as well. For class "0", it achieves a precision of 0.85, Recall of 0.89 and F1-score of 0.87. For class "1", the scores are also commendable with a precision of 0.68, Recall 0f 0.61 and F1-score of 0.64. This suggests that the stacking Ensemble offers a balanced and high-performing model for both classes.

In summary, The stacking Ensemble model clearly emerges as the most adept in this cross-domain analysis, outperforming the Hard and Soft Majority Voting models both in macro metrics and per-class evaluation. The majority voting models, while showing some proficiency in classifying class ”0”, exhibit significant challenges in predicting class ”1” effectively.

6.2.2 Qualitative Analysis

Based on the class-specific metrics of table 8, the hard majority voting model displays a balanced precision, recall and F1-score of both labels, signifying an equal number of false positives and false negatives for each class. The Soft Majority Voting model, on the other hand, seems to be more inclined to missclassify instances of label ”1” as label ”0”, evidenced by its higher recall for label ”0” and conversely, higher precision for label ”1” and this observation is also futher supported by the observations of false positives and false negatives in table 9. The stacking ensemble, although superior overall, is not immune to errors, particularly showing a slight propensity to over-predict instances as label ”0”, given its recall of 0.89 for that class.

Addressing these observed patterns, Hard Majority Voting error seems intrinsically linked to its inherent mechanism and might benefit from a more sophisticated decision-making process. For soft majority voting, an examination of the weighting mechanism and adjusting the soft voting thresholds could potentially strike a better balance between precision and recall. The stacking ensemble minor inclination towards false negatives for label ”1” can be rectified with targeted resampling or by training the model on more data.

Model	**Hard Majority Voting**		**Soft Majority Voting**		**Stacking Ensemble**	
	Predicted 0	**Predicted 1**	**Predicted 0**	**Predicted 1**	**Predicted 0**	**Predicted 1**
Actual 0	447	173	378	242	551	69
Actual 1	172	68	149	91	94	146

Table 9: Cross Domain Confusion Matrices: Confusion Matrices for Hard Majority Voting, Soft Majority Voting, and Stacking Ensemble

6.3 Comparison between model performance in In-domain and Cross-Domain Analysis

Model	**In-Domain**			**Cross-Domain**		
	Precision	**Recall**	**F1**	**Precision**	**Recall**	**F1**
Hard Majority	0.51	0.51	0.39	0.5	0.5	0.5
Soft Majority	0.49	0.49	0.48	0.5	0.49	0.5
Stacking Ensemble	0.81	0.79	0.8	0.77	0.75	0.76

Table 10: Performance Metrics for In-Domain and Cross-Domain Analysis

The Analysis in summary Table 10 showcases significant disparities between the models when evaluated in both in-domain and cross-domain setting. In the in-domain, models were trained and tested on the OLID dataset. The stacking Ensemble models emerged as the better model, with macro scores of precision, recall, F1-score at 0.81, 0.79, 0.80 respectively. Conversely, the majority voting models, both Hard and Soft, hovered around 0.50 in macro precision and recall. The detailed per-class evaluation further substantiated this claims, with the stacking Ensemble delivering robust results for both classes.

In cross-domain setting, the Stacking Ensemble had high performance when the model was trained on the HASOC dataset and tested on the OLID dataset. It achieved macro score of 0.77 in precision, 0.75 in recall and 0.76 in F1. The Majority Voting models, struggled to match the performance of Stacking Ensemble, with both the Hard and Soft variants showcasing macro precision and recall scores around 0.50. The detailed per-class analysis also has similar findings: the Stacking Ensemble was not only consistent in its high performance across both classes but also significantly outperformed the majority voting models.

The ensemble approach, particularly the Stacking Ensemble, delivered improved results compared to the Majority Voting models. There are several reasons for this enhancement:

Diversity in Predictions: Ensemble methods, especially stacking, combine multiple models, allowing them to capitalize on the strengths of each individual model. This diversity leads to more comprehensive and robust predictions, mitigating the weaknesses of any single model.

Reduced Overfitting: The ensemble approach inherently reduces overfitting. By combined predictions from multiple models, it ensures that individual model biases don't heavily influence the final prediction. This is especially evident in the cross-domain setting, where the Stacking Ensemble's ability to generalize was more pronounced than the Majority Voting models.

Challenges with Majority Voting: Both Hard and Soft Majority Voting rely on a consensus mechanism. If individual models in the ensemble are flawed or biased, this can heavily influence the final decision, particularly in Hard Voting. The Soft Majority Voting attempts to mitigate this by considering the confidence of each model, but it still cannot match the adaptability and learning capabilities of the Stacking Ensemble.

In conclusion, the ensemble approach, and the Stacking Ensemble, in particular, offers a sophisticated and holistic method to combine predictions from various models. This, in turn, provides a more balanced, resilient, and high-performing model, evident from its consistent dominance in both in-domain and cross-domain analyses.

6.4 Comparison of the finding the error analysis to assignment 3:

From the error analysis for assignment 3, it is evident that determining the toxicity of a comment is not a straightforward process, Simple word flagging, even if a word is typically associated with toxicity or hate, can lead to false positives. Conversely, more subtle forms of toxicity like sarcasm or toxicity without swear words can be missed, leading to false negatives.

Upon examining the performance of the three models - Hard Majority Voting, Soft Majority Voting and Stacking Ensemble- in tables 4,5,7, 8 provides several insights emerge.

In the context of In-Domain Analysis, where models are trained and tested on the OLID dataset as shown in table 4 and 5, the Stacking Ensemble model stands out. It clearly outperforms both the Hard and Soft Majority Voting models across all metrics, namely MacroPrecision, MacroRecall, and MacroF1. A more detailed look, as given in Table 5, reveals that the Stacking Ensemble model demonstrates superior performance for both labels (0 and 1). It strikes a balance between precision and recall, leading to higher F1-Scores for both labels in comparison to the other models.

Moving on to Cross-Domain Analysis, which entails training models on the HASOC dataset and then testing them on the OLID dataset as seen in table 7 and 8, the findings are slightly different. While the Stacking Ensemble model continues to be the top performer, there is a noticeable decline in its performance compared to the in-domain testing. This drop can be attributed to the inherent challenges of cross-domain testing. Analyzing further, as depicted in table 8, the Stacking Ensemble model maintains a balance of precision and recall for both labels. However, the performance metrics edge closer to those of the Majority Voting models, especially concerning label 1.

Tying these findings back to the error analysis, some speculations can be made. The dominant performance of the Stacking Ensemble model might hint at its adoption of the error mitigation strategies discussed earlier. It possibly excels in aspects like incorporating context or tapping into other linguistic refinement. The dip in cross-domain testing performance serves as a evidence to the significance of diverse training data. Models exclusively trained on one dataset, such as HASOC in this scenario, may struggle to generalize their findings when exposed to a different dataset, like OLID. Such refinements in linguistic expressions, contexts, and other subtleties highlight the error analysis's emphasis on the value of diversified training data.

In conclusion, the insights extracted from the error analysis emphasize the intricacies associated with text classification tasks, particularly when it comes to pinpointing toxic comments. The tables from assignment 3 furnish a numerical perspective on these intricacies, reflecting the varied success rates of different models in dominating these challenges. Among them, the Stacking Ensemble model appears most adept, potentially attributed to its skills in grasping context and leveraging advanced features.

7 Discussion/Conclusions

In the recently concluded study, significant findings emerged, providing insight on previously undefined patterns and variables that have substantial influence on the outcomes, enriching the overall understanding of the topic. Reflecting upon the tasks undertaken, it is evident that the choice of methodologies played a pivotal role in the extraction of these insights, with the precision of data collection and analysis ensuring both authenticity and clarity. Even though the results were illuminating, it is essential to acknowledge the inherent limitations present in every research endeavor. Addressing these limitations, including potential data biases and model scalability issues, posed challenges, but through meticulous efforts and innovative solutions, the study's credibility remained intact. There is merit in diving deeper into the anomalies and outliers presented by the current findings, possibly through qualitative studies or experimental methodologies. Engaging in cross-disciplinary collaborations could also unravel variations and overlooked connections, while the advent of sophisticated analytical tools designed specifically for this domain promises to elevate the precision and comprehensiveness of future research. In conclusion, this study has laid a robust foundation, providing valuable takeaways and setting the stage for a plethora of forthcoming exploitative endeavors that will further enrich our comprehension of the subject.

8 Appendix/Division of work

Collectively, our team adopted a collaborative approach, ensuring an equitable distribution of responsibilities. We began by collectively delving into the dataset, collaboratively exploring its intricacies. As a cohesive unit, we assigned specific models to each team member and initiated the training process. Each model was rigorously tested, and the outcomes were collectively evaluated. Our joint efforts extended beyond model training and encompassed the comprehensive report's creation, where every team member actively contributed their insights and expertise. In this collaborative endeavor, each individual's unique strengths and perspectives coalesced, resulting in a holistic and well-rounded project.

References

[1] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[2] Zeerak Waseem and Dirk Hovy. Hateful symbols or hateful people? predictive features for hate speech detection on twitter. In *Proceedings of the NAACL student research workshop*, pages 88–93, 2016.

[3] Sanjana Sharma, Saksham Agrawal, and Manish Shrivastava. Degree based classification of harmful speech using twitter data. *arXiv preprint arXiv:1806.04197*, 2018.

[4] Aymé Arango, Jorge Pérez, and Barbara Poblete. Hate speech detection is not as easy as you may think: A closer look at model validation. In *Proceedings of the 42nd international acm sigir conference on research and development in information retrieval*, pages 45–54, 2019.

[5] Betty Van Aken, Julian Risch, Ralf Krestel, and Alexander Löser. Challenges for toxic comment classification: An in-depth error analysis. *arXiv preprint arXiv:1809.07572*, 2018.

[6] Ilia Markov and Walter Daelemans. Improving cross-domain hate speech detection by reducing the false positive rate. In *Proceedings of the Fourth Workshop on NLP for Internet Freedom: Censorship, Disinformation, and Propaganda*, pages 17–22, 2021.

[7] Joshua Melton, Arunkumar Bagavathi, and Siddharth Krishnan. Del-hate: a deep learning tunable ensemble for hate speech detection. In *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1015–1022. IEEE, 2020.

[8] Hsiang-Fu Yu, Hung-Yi Lo, Hsun-Ping Hsieh, Jing-Kai Lou, Todd G McKenzie, Jung-Wei Chou, Po-Han Chung, Chia-Hua Ho, Chun-Fu Chang, Yin-Hsuan Wei, et al. Feature engineering and classifier ensemble for kdd cup 2010. In *KDD cup*, 2010.

[9] Mladen Karan and Jan Šnajder. Cross-domain detection of abusive language online. In *Proceedings of the 2nd workshop on abusive language online (ALW2)*, pages 132–137, 2018.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[11] Steve Durairaj Swamy, Anupam Jamatia, and Björn Gambäck. Studying generalisability across abusive language detection datasets. In *Proceedings of the 23rd conference on computational natural language learning (CoNLL)*, pages 940–950, 2019.

[12] Franco Molteni, Roberto Buizza, Tim N Palmer, and Thomas Petroliagis. The ecmwf ensemble prediction system: Methodology and validation. *Quarterly journal of the royal meteorological society*, 122(529):73–119, 1996.

[13] Charu C Aggarwal and ChengXiang Zhai. A survey of text classification algorithms. *Mining text data*, pages 163–222, 2012.

[14] Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar. Predicting the type and target of offensive posts in social media. *arXiv preprint arXiv:1902.09666*, 2019.

[15] Thomas Mandl, Sandip Modha, Prasenjit Majumder, Daksh Patel, Mohana Dave, Chintak Mandlia, and Aditya Patel. Overview of the hasoc track at fire 2019: Hate speech and offensive content identification in indo-european languages. In *Proceedings of the 11th annual meeting of the Forum for Information Retrieval Evaluation*, pages 14–17, 2019.

[16] Mahidhar Dwarampudi and NV Reddy. Effects of padding on lstms and cnns. *arXiv preprint arXiv:1903.07288*, 2019.

[17] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[19] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.

[20] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.

[21] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[22] Wenyuan Dai, Gui-Rong Xue, Qiang Yang, and Yong Yu. Co-clustering based classification for out-of-domain documents. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 210–219, 2007.

[23] Tommaso Caselli, Valerio Basile, Jelena Mitrović, and Michael Granitzer. Hatebert: Retraining bert for abusive language detection in english. *arXiv preprint arXiv:2010.12472*, 2020.

[24] Eshwar Chandrasekharan, Umashanthi Pavalanathan, Anirudh Srinivasan, Adam Glynn, Jacob Eisenstein, and Eric Gilbert. You can't stay here: The efficacy of reddit's 2015 ban examined through hate speech. *Proceedings of the ACM on human-computer interaction*, 1(CSCW):1–22, 2017.

[25] E Peters Matthew, N Mark, I Mohit, G Matt, C Christopher, and L Kenton. Deep contextualized word representations (2018). *arXiv preprint arXiv:1802.05365*, 1802.

[26] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. *arXiv preprint arXiv:1911.02116*, 2019.

[27] Diptanu Sarkar, Marcos Zampieri, Tharindu Ranasinghe, and Alexander Ororbia. fbert: A neural transformer for identifying offensive content. *arXiv preprint arXiv:2109.05074*, 2021.

[28] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.

[29] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, 1992.