Assignment Automatic Classification of Hate Speech (Methods)

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

November 16, 2023

Link to Jupyter Notebook
Link to Model Files

1 Data

1.1 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[1] 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[2] 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.

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.

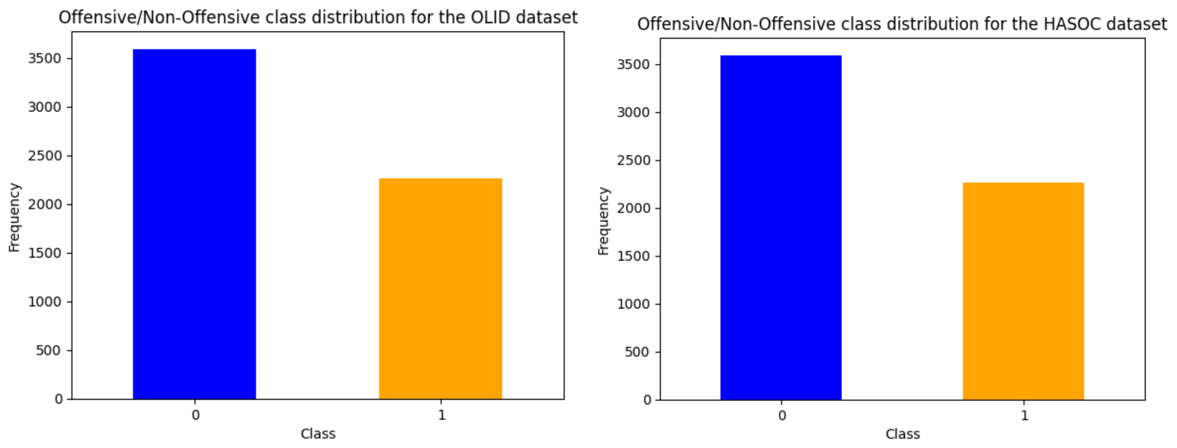

Figure 1: Distributions of Offensive and Non-Offensive classes.

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 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[3]. 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.

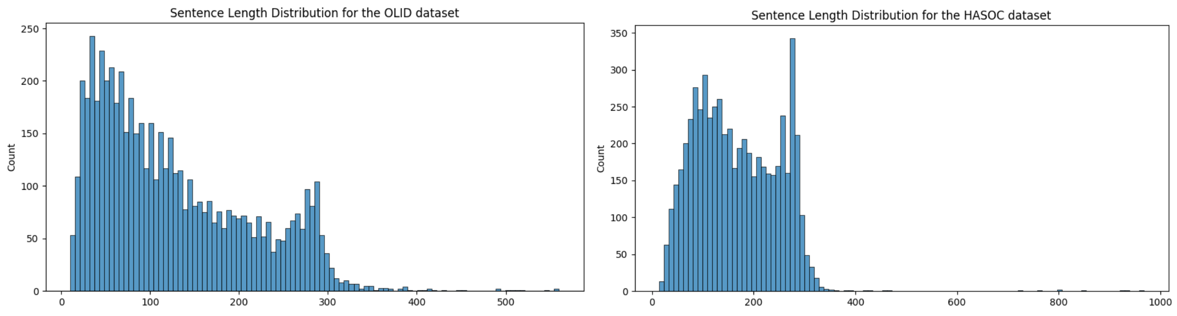

Figure 2: Distributions of text size in both datasets.

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.

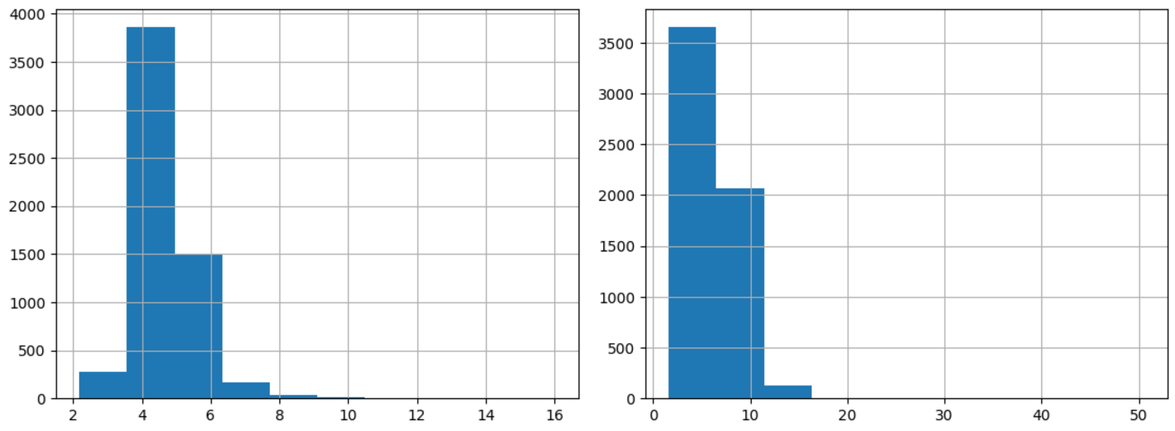

Figure 3: Average word length for both datasets.

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 [4] 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.

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 5, 6 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

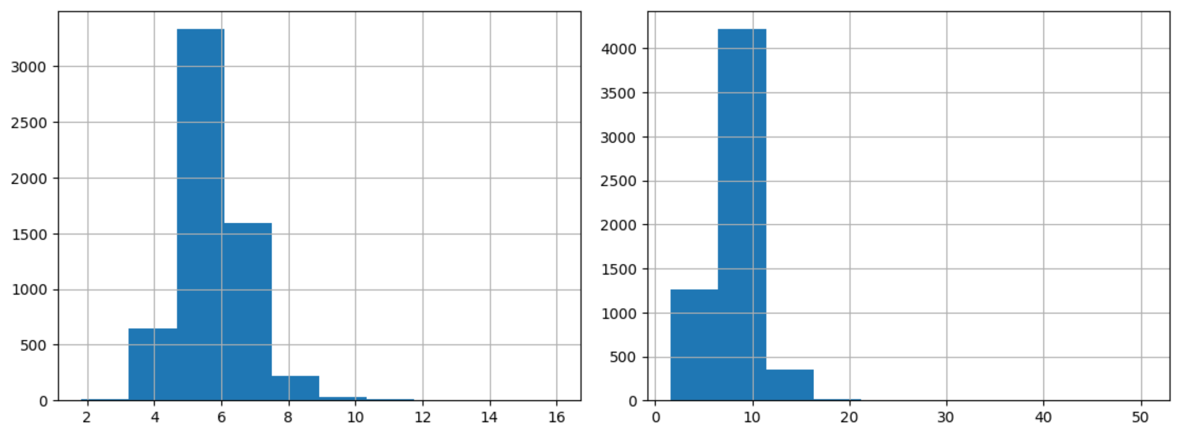

Figure 4: Average word length for both datasets without stopwords.

analysis gives us important information about what topics are discussed and how people use language in the dataset.

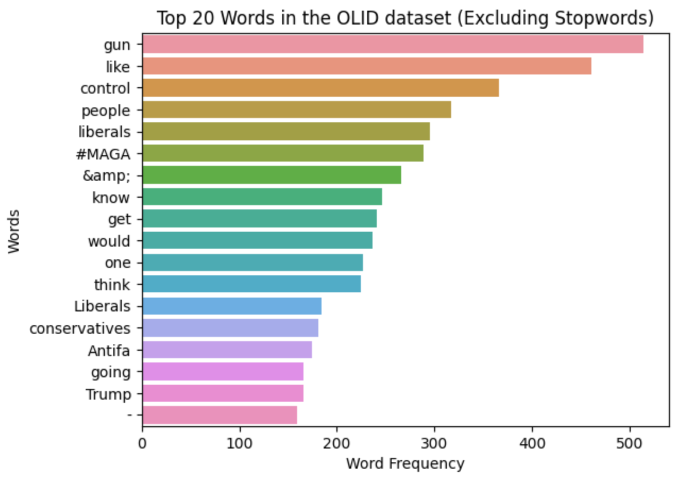

Figure 5: Top 20 more frequent words in the OLID dataset.

1.2 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. Transformers are highly capable of handling unstructured text data with minimal preprocessing due to their self-attention mechanisms.[5] 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.

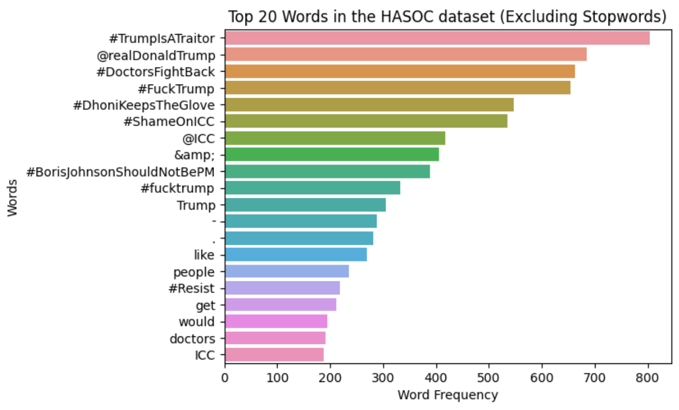

Figure 6: Top 20 more frequent words in the HASOC dataset.

2 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 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[6].

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.

3 Methods

3.1 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[7]. 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[8] and the Byte-Pair Encoding[9]. 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 table1 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 1: Maximum length of input tokens per Transformer model

3.2 Models

3.2.1 HateBERT

HateBERT[10] 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[11]. 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 with the appropriate configuration for sequence classification, specifically binary classification with two labels. A data collation function is defined to handle batch processing during training, ensuring uniform sequence lengths. Training arguments and a Trainer object are configured to initiate and manage the training process. Training parameters are configured, and the model is fine-tuned using the HateBERT architecture on the OLID 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.

3.2.2 fBERT

Transformer-based models, such as BERT, ELMO[12], and XLM-R[13] 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[14] 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[15] and HateBERT[10] 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 is very similar to HateBERT, it includes data loading, tokenizer initialization, sorting the data by text length, formatting the data for model training, configuring training parameters, training the transformer model "fBERT" from hugging face[16] with default parameters, evaluating its performance, and saving/loading the model and tokenizer. Finally, it conducts in-domain and cross domain testing and computes classification metrics, such as precision, recall, and a confusion matrix, to assess the model's effectiveness on the OLID dataset. For Tokenization fBERT tokenizer is used as the vocabularies and potential special tokens or rules might differ between the two models, depending on how they were trained or fine-tuned.

3.2.3 SVM

Support Vector Machines (SVM)[17] 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 2: 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 2.

3.2.4 LSTM

Long Short-Term Memory (LSTM)[18] networks are a type of recurrent neural network (RNN) architecture known for their effectiveness in sequence modeling and text analysis tasks. LSTMs are particularly well-suited for hate speech detection due to their ability to capture and analyze context and long-range dependencies in text data. Hate speech often exhibits complex and context-dependent patterns, making it crucial to consider the broader context of a statement to accurately identify offensive language. LSTMs, with their inherent memory mechanisms, can model these intricate relationships within text data, enabling them to recognize subtle linguistic nuances that may indicate hate speech. They are especially effective when dealing with sequences of varying lengths, making them suitable for processing social media posts, comments, or other user-generated content. LSTMs, with their capability to capture temporal dependencies and context, have been proven valuable in the detection of hate speech, providing a robust and context-aware approach to content moderation and online safety efforts.

The implementation consists of several key steps. First, it loads the training and test datasets from CSV files and tokenizes the text using a Tokenizer. It then converts the text data into sequences and pads them to ensure consistent input lengths. The model architecture is defined, comprising an embedding layer, 2 LSTM layers with different units, dropout for regularization, and two dense layers for classification. The code compiles the model using binary cross-entropy as the loss function and the Adam[19] optimizer with a specified learning rate. It trains the model for 5 epochs, with validation on the test data, and evaluates its performance.

Parameter	**Value**
Activation Function	Relu, Sigmoid
Optimizer	Adam
Learning Rate	0.001
Dropout Rate	0.5

Table 3: Parameters of LSTM

The choice of specific parameters is significant in achieving a well-performing LSTM model. For instance, the Tokenizer is initialized with a vocabulary limit of 10,000 words, which can help capture a wide range of vocabulary while excluding infrequent words. In the model architecture, the embedding layer has an output dimension of 16, which is typically set to be relatively small to capture essential features. The use of two LSTM layers, one with 64 units and the other with 32 units, allows the model to learn both short and long-term dependencies in the text data. The dropout layer with a rate of 0.5 helps prevent overfitting. The model uses binary cross-entropy as the loss function for binary classification tasks and the Adam optimizer with a learning rate of 0.001.The overview of the parameter is as shown in table 3.

3.2.5 LLM GPT2

GPT-2, a powerful language model developed by OpenAI[20], has shown immense potential for various natural language processing tasks, including hate speech detection. One of its significant advantages is its exceptional ability to understand and generate human-like text. When employed in hate speech detection, GPT-2 can effectively analyze and contextualize text data, recognizing subtle nuances and the underlying intent in statements. Its large pre-trained knowledge base, combined with its ability to generate coherent text, can assist in identifying not only explicit hate speech but also more subtle forms, such as coded or veiled language. Additionally, GPT-2 can be fine-tuned on specific hate speech datasets, enabling it to adapt to the language and trends commonly found in online hate speech.

The implmentation starts by loading the dataset, initializing the GPT-2 tokenizer, and tokenizing the text data, ensuring truncation and padding while limiting sequences to 128 tokens(we were experimenting with different max_lengths and tryign to use dynamic padding but couldnt accomplish that) with the pad token is set to the end-of-sequence token (eos_token) . A custom dataset class is defined to format the data, and the GPT-2 model for sequence classification is initialized with default parameters. Training arguments are set, and a Trainer instance is created for model training and evaluation. After training, the model is saved for future use. The code also includes reporting classification metrics and a confusion matrix to assess the model's performance. The chosen 'gpt2-medium' model, along with these parameters and procedures, equips the GPT-2 model to effectively detect hate speech by comprehensively understanding and contextualizing text data.

4 Results and Analysis

4.1 Cross Domain Analysis

In the cross-domain scenario, the models were trained on the HASOC dataset and then evaluated on the OLID dataset. This type of evaluation helps to ascertain the models beyond the original data on which they were trained.

Looking at the Macro-averages in table 4 provides a Cross-Domain Analysis of different models trained on the HASOC dataset and tested on the OLID dataset. In the evaluation, both SVM and LSTM emerge as notable performers based on their Macro F1 scores, suggesting a good balance between precision and recall in this context. On the other hand, although HateBERT and GPT2 exhibit commendable Macro Precision, their low Macro F1 scores indicate an imbalance in their performance across classes. fBERT strikes a harmonious balance with consistent scores in both precision and recall, showcasing its stable performance in this analysis. The results highlight the intricate nature of model behaviors across varying evaluation settings and the significance of assessing them through multiple metrics.

Model	MacroPrecision	MacroRecall	MacroF1
HateBERT	0.64	0.5	0.22
fBERT	0.52	0.52	0.42
LSTM	0.36	0.5	0.42
SVM	0.51	0.5	0.47
GPT2	0.59	0.51	0.23

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

In the detailed analysis provided by Table 5, one can observe that the LSTM model, despite its consistent performance in other metrics, fails to predict label 1 entirely. This shows a pronounced bias towards label 0. On the Other hand, SVM exhibits a stronger Performance towards label 0 with respectable F1 score of 0.73.

Model	Label	Precision	Recall	F1-Score
HateBERT	0	1	0	0
HateBERT	1	0.28	1	0.44
fBERT	0	0.75	0.29	0.41
fBERT	1	0.29	0.75	0.42
LSTM	0	0.72	1	0.84
LSTM	1	0	0	0
SVM	0	0.63	0.87	0.73
SVM	1	0.39	0.14	0.21
GPT2	0	0.9	0.01	0.03
GPT2	1	0.28	1	0.44

Table 5: Detailed Cross-Domain Analysis: Model trained on HASOC dataset and tested on OLID dataset

For Cross-Domain tasks, SVM remains one of the top-performing models. However, the confusion matrix shown in Table 6 illustrates that its performance drops when compared to in-domain metrics, emphasizing the challenges of adapting to new datasets. Specifically, the matrix highlights false positives and false negatives that SVM encounters in the cross-domain setting. Meanwhile, the consistency of LSTM, despite not being the top scorer, indicates potential for a model that can generalize across domains with some class biases.

	Predicted: 0	Predicted: 1
Actual: 0	465	155
Actual: 1	165	75

Table 6: Cross Domain Confusion Matrix for SVM

4.2 In-Domain Analysis

The In-domain analysis from table 7 for various models, reveals their performance when both trained and tested on OLID dataset. The SVM model notably achieves the highest MacroF1 score (0.66), indicating a relatively well-balanced performance between precision and recall in a macro-averaged context. The other models show some disparity amongst the metrics, underscoring distinct trade-offs between precision, recall, and their harmonic mean (F1 score).

In table 9 it becomes apparent that models behave disparately between two classes, highlighting different model sensitivities and propensities towards either of classes. For example, while the LSTM model excels in recognizing class 0 with an F1-score of 0.84, it utterly fails to recognize class 1, as indicated by the scores of 0 across all three metrics. Conversely, fBERT and GPT2 exhibit an inclination towards class 1 with better scores in Recall and F1 for this particular class.

Model	MacroPrecision	MacroRecall	MacroF1
HateBERT	0.51	0.51	0.48
fBERT	0.52	0.52	0.42
LSTM	0.36	0.5	0.42
SVM	0.71	0.66	0.66
GPT2	0.51	0.51	0.34

Table 7: In-Domain Analysis: Model trained and tested on the OLID dataset.

Overall, SVM outperforms other models when considering the scores from tables 7 and 9. The high macro averages hint at an equilibrium between precision and recall across both classes. This balance is further illuminated by the confusion matrix shown in Table 8. The matrix highlights that SVM correctly classifies a substantial number of instances while maintaining a relatively low rate of false positives and false negatives. This balanced performance, combined with high overall scores, firmly establishes SVM as a strong model for this task. Its demonstrated proficiency in both precision and recall across the classes further solidifies this assessment in this specific context.

	Predicted: 0	**Predicted: 1**
Actual: 0	497	123
Actual: 1	197	43

Table 8: In-Domain Confusion Matrix for SVM

Model	**Label**	**Precision**	**Recall**	**F1-Score**
HateBERT	0	0.73	0.5	0.59
HateBERT	1	0.29	0.51	0.37
fBERT	0	0.75	0.29	0.41
fBERT	1	0.29	0.75	0.42
LSTM	0	0.72	1	0.84
LSTM	1	0	0	0
SVM	0	0.7	0.89	0.79
SVM	1	0.72	0.43	0.54
GPT2	0	0.74	0.16	0.26
GPT2	1	0.28	0.85	0.42

Table 9: Detailed In-Domain Analysis: Model trained and tested on the OLID dataset.

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

SVM consistently ranks as the best-performing model in the in-domain context based on macro-averaged scores, as shown in Table 7. However, in the cross-domain setting, as illustrated in Table 5, SVM's performance drops but remains competitive. The LSTM model interestingly maintains consistent performance across the two settings, even if its scores are not the highest. Models like GPT2 and HateBERT display significant performance variations between the two contexts.

HateBERT exhibits a significant decline in the model's macro-averaged F1 score when transitioning to the cross-domain context, dropping from an in-domain score of 0.48 to just 0.22, as reflected in Table 5. On the other hand, fBERT shows slight variations in scores but remains reasonably consistent across both settings. SVM's macro F1 score decreases from 0.66 in the in-domain setting to 0.47 in the cross-domain scenario. GPT2's F1 score also experiences a decline, transitioning from an in-domain score of 0.34 to a cross-domain score of 0.23.

In terms of individual class performance, as illustrated in Table 5, HateBERT's performance dramatically decreases for label 0 in the cross-domain context, while maintaining consistency for label 1. fBERT's results show relative stability for both classes across both contexts. The LSTM model consistently fails to predict label 1, exhibiting a clear bias towards label 0, for which it achieves an F1 score of 0.84. SVM, in the cross-domain setting, experiences a significant drop in performance for label 1. Conversely, GPT2 struggles considerably with label 0 in the cross-domain scenario, yet remains consistent for label 1.

Several factors can explain the differences in performance between in-domain and cross-domain contexts. First, variations in the data distributions between the HASOC and OLID datasets might introduce unique language patterns, complicating cross-domain predictions. Moreover, models like LSTM that exhibit a bias towards a particular class might be overfitting to the training data. LSTM's consistently high predictions for label 0 suggest potential class imbalances in the dataset.The performance discrepancies between in-domain and cross-domain contexts can be further understood by examining specific linguistic challenges:

Toxicity without swear words: Within a specific domain, the language patterns might be more direct, making it easier for models to identify hate speech. However, in a broader context, people might use indirect, subtle language without using explicit hate words, which can be tricky for models to catch.

Rhetorical questions: In-domain data might have more straightforward expressions, while cross-domain contexts might see increased use of rhetorical questions. Such questions can obscure the actual intention or sentiment behind the statement, making detection of hate speech challenging.

Metaphors and comparisons: While in a given domain, the usage of metaphors might be limited and well-understood by models, in cross-domain settings, people might employ a variety of indirect language forms. These can hide the true sentiment, leading models astray.

Rare words: In a specific domain, the vocabulary might be more standardized, so models can be well-trained on those terms. However, in a broader context, users might employ lesser-known words which models might not be familiar with, causing misinterpretations.

Sarcasm and irony: Within a defined domain, sarcasm and irony could be less prevalent or follow specific patterns that models can recognize. However, in cross-domain data, the diverse ways sarcasm and irony are used can flip the meaning of sentences, making them harder to interpret correctly.

Doubtful labels: In the in-domain context, labeling might be more consistent due to the familiarity with the specific type of data. But when dealing with varied data from multiple sources, there's a higher chance of encountering ambiguous labels. This inconsistency can confuse the model, affecting its learning and subsequent predictions.

In summary, while SVM stands out in terms of performance metrics in the in-domain context, LSTM demonstrates a stable, though not necessarily optimal, performance across both domains. The varying results between the two contexts underscore the importance of domain-specific tuning and highlight the necessity of deepening our understanding of linguistic patterns, as evident from Tables 4, 5, 7, and 9.

5 Appendix

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] 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.

[2] 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.

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

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

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

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

[18] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[19] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[20] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.