INSTITUTO
UNIVERSITÁRIO
DE LISBOA

A Full Hate Speech Detection Pipeline: Leveraging ML, DL, and GPTs

Rodrigo Carvalheda Duarte da Fonseca Sarroeira

Master in Data Science

Supervisor:
Doctor Catarina Marques, Associate Professor, Iscte - University Institute of Lisbon

Co-Supervisor:
Doctor Rita Guerra, Integrated Researcher, Iscte - University Institute of Lisbon

September, 2024

BUSINESS
SCHOOL

TECNOLOGIAS
E ARQUITETURA

Department of Quantitative Methods for Management and Economics
Department of Information Science and Technology

A Full Hate Speech Detection Pipeline: Leveraging ML, DL, and GPTs

Rodrigo Carvalheda Duarte da Fonseca Sarroeira

Master in Data Science

Supervisor:
Doctor Catarina Marques, Associate Professor, Iscte - University Institute of Lisbon

Co-Supervisor:
Doctor Rita Guerra, Integrated Researcher, Iscte - University Institute of Lisbon

September, 2024

Resumo

A proliferação da tecnologia e redes sociais levou a um aumento na incidência e variedade de discurso de ódio, criando a necessidade de estratégias mais eficazes para a sua detecção e mitigação. Neste trabalho, é desenvolvido um pipeline de deteção de discurso de ódio aplicável a diversos casos. Modelos tradicionais de machine learning e modelos avançados de deep learning são utilizados para classificar publicações em redes sociais como contendo ódio ou não. O processo de treino de machine learning envolve o treino de várias combinações de modelos e técnicas de vetorização, seguido por um rigoroso fine-tune com o Optuna. O melhor modelo de machine learning foi o LightGBM, codificado com TF-IDF de tamanho 10,000, alcançando uma accuracy de 0,816. São também exploradas abordagens baseadas no BERT, obtendo melhores resultados, com o modelo RoBERTa a atingir uma accuracy de 0,8392. Este trabalho contribui significativamente para a explicabilidade da classificação, frequentemente esquecida na detecção de discurso de ódio, especialmente com black-box models. O pipeline proposto utiliza generative pre-trained transformers (GPT) juntamente com prompt engineering para adicionar explicabilidade ao processo de classificação. Modelos GPT foram ajustados para detectar o racional por trás da decisão de classificação, destacando o conteúdo odioso no texto. O melhor modelo GPT ajustado foi o GPT-4o Mini apresentando uma accuracy de 0,959 e um F1-Score de 0,961. Foi desenvolvida uma aplicação web utilizando Django e React, compilando os melhores modelos treinados durante o estudo, fornecendo aos utilizadores uma interface gráfica amigável para interagir com o pipeline proposto, tornando o processo de detecção mais acessível e eficiente.

Palavras-chave: Detação de discurso de ódio, GPT, Machine Learning, Deep Learning, BERT, Prompt Engineering.

Códigos de classificação JEL: C45, C63.

Abstract

The proliferation of technology and social media has led to an alarming increase in the incidence and variety of hate speech, creating a need for more effective detection and mitigation strategies. In this work we develop a comprehensive hate speech detection pipeline applicable to various use-cases. A combination of traditional machine learning and state-of-the-art deep learning models are employed to classify social media posts as either hateful or non-hateful. The ML training process involved multiple model combinations and word vectorization techniques, followed by rigorous fine-tuning with Optuna. The best performing machine learning model was LightGBM encoded with TF-IDF of size 10 000, achieving an accuracy of 0.816. Advanced BERT-based approaches were explored, yielding superior results, with RoBERTa reaching an accuracy of 0.8392. A significant contribution of this work is the incorporation of explainability, often overlooked in hate speech detection, particularly with black-box models. Our proposed pipeline leverages the advances in generative pre-trained transformers along with prompt engineering to add a layer of explainability to the classification process. GPT models were fine-tuned for detecting the rational behind the classification decision, effectively highlighting the hate content within the text. The best performing GPT fine-tuned model was GPT-4o Mini with an accuracy of 0.959 and a F1-Score of 0.961. A web-based application using Django and React was developed, compiling the best models trained in during the study. Providing users with a user-friendly graphical interface to interact with the proposed pipeline, making the detection process more accessible and efficient.

Keywords: Hate Speech Detection, GPT, Machine Learning, Deep Learning, BERT, Prompt Engineering

JEL Classification Codes: C45, C63.

Glossary

- **API**: Application Programming Interface
- **BERT**: Bidirectional Encoder Representations from Transformers
- **DL**: Deep Learning
- **DistilBERT**: Distilled BERT
- **GPT**: Generative Pre-trained Transformer
- **GloVe**: Global Vectors for Word Representation
- **GUI**: Graphical User Interface
- **JSON**: JavaScript Object Notation
- **JSONL**: JSON Lines
- **KNN**: K-Nearest Neighbors
- **LGBM**: Light Gradient Boosting Machine
- **LR**: Logistic Regression
- **ML**: Machine Learning
- **NB**: Naive Bayes
- **NLP**: Natural Language Processing
- **RoBERTa**: Robustly Optimized BERT Pre-training Approach
- **ROS**: Random OverSampling
- **TF-IDF**: Term Frequency-Inverse Document Frequency
- **TPE**: Tree-structured Parzen Estimator

Contents

List of Figures

List of Tables

1 Introduction

The rise of social media platforms has created a vast digital space where users can freely share their thoughts and opinions with a global audience. This democratization of speech enables valuable interactions, but also brings challenges, such as the proliferation of harmful content. The open nature of these platforms can inadvertently foster environments that are hostile to marginalized groups, leading to serious social, psychological, and behavioral consequences, including mental health issues, hate crimes, and even physical violence (Silva et al., 2021). In addition, online spaces have been shown to amplify problematic behaviors such as disinhibition, lack of empathy, and moral disengagement, making it easier for users to engage in hate speech and other forms of harmful communication (Pluta et al., 2023).

Given this evolving landscape, for effectively detecting online hate speech, the use of advanced algorithms is required due to the large volumes of data generated in the digital ecosystem. Given the diversity and complexity of this problem, the algorithms must be robust in detecting underlining patterns in the data. In this study, three classes of models, machine learning, deep learning, and generative pre-trained transformers are employed with the goal of effectively detecting hateful content at a sentence and word level.

Hate speech is highly dependent on its context and domain, and most studies focus on developing the best solution for a very concise domain. To fight against this tendency, the methodology employed in this work can be generalized to any hate speech text database. During the data preparation and modeling phases, multiple combinations of techniques are employed and combined, using Optuna, to ensure that the selected techniques are optimized to the specific domain and characteristics of the data.

Another research gap lies in the lack of importance given the explainability of the hate speech detection task. Detecting hate speech is not a straightforward task given its diversity and fast proliferation, nevertheless, many studies have successfully introduced robust pipelines that achieve very reasonable performance metrics (Hashmi et al., 2024; Hartvigsen et al., 2022). Although the good performance of the models is a key metric for their evaluation, this itself is not enough for their implementation in the real world. Given the highly subjective, controversial, and ethical nature of hate speech, the implementation of hate speech detection models strongly depends on their explainability, or, in other words, the rationale behind the model prediction (Reichel, 2022). To solve this issue, a subsequent layer of modeling is added, resorting to generative pre-trained models for explaining the rational behind the model's decision, by highlighting the words or expressions that influenced the decision. This is done by fine-tuning GPT models based on a labeled dataset of hate speech rationals. By introducing this method, a higher and more detailed level of explainability is obtained.

This study aims to develop a replicable pipeline for accurately detecting hate speech while improving the explainability of the models used. To achieve this, the following key research questions will be addressed:

1. What is the best combination of machine learning models and vectorization techniques for effectively detecting hate speech at the sentence level?
2. How can we leverage the most of BERT-based models for hate speech detection?
3. How can generative pre-trained transformers be leveraged to improve model explainability at a word level?
4. How can we deploy the models from this study in a user-friendly way using modern web development frameworks?

2 Concepts

In this chapter, key terms and definitions, essential for understanding hate speech detection, are presented, focusing on natural language processing (NLP), machine learning, and deep learning. These concepts form the foundation of the methods and technologies used to identify and classify hate speech.

2.1 Hate Speech

There is not an unique agreed definition for hate speech (Assimakopoulos et al., 2017), since the term can be analyzed from different perspectives and identities, such as physiologists, academics, policymakers, and legal experts. The definition of this term can sometimes be to vague and contradictory between authors (Brown, 2015). Nevertheless, a commonly agreed general definition for hate speech refers to any communication, spoken, written, or behavioral, that uses hurtful language against an individual or group due to sensitive traits like religion, ethnicity, or gender (Hietanen and Eddebo, 2023).

Hate speech has been in the spotlight for some time now, especially given that its expression reaches a considerably wider range of individuals when focusing on the digital sphere (Alrehili, 2019). The impact extends globally, affecting law-abiding citizens universally and necessitating a concerted effort to address and curb this harmful behavior. As online interactions continue to play a significant role in our daily lives, understanding and combating hate speech is crucial to fostering a safer and more inclusive digital environment for everyone.

Initiatives aimed at raising awareness and promoting respectful online discourse are essential in mitigating the harmful effects of hate speech and fostering a more positive online community. One of the approaches used for its mitigation is the legal path. The European Commission is working to introduce hate speech as an ”area of crime” (COMMISSION, 2021).

2.2 Natural Language Processing

Natural Language Processing, often referred to as NLP, is a field within computer science that focuses on the interaction between humans and computer languages (Khyani et al., 2021). It was introduced in 1940, during the Second World War, to automate the process of translation between English and Russian. Natural Language Processing consists of a group of tools, algorithms, and models that enable machines to easily understand human language, being able to interpret, generate, and respond to it. NLP is a general concept formed by various sub-areas. Text processing techniques, such as tokenization, vectorization, and embeddings, are used to transform text into more efficient formats for storage, analysis, and as input for machine learning models. Text generation and synthesis techniques are fundamental applications of NLP; they are widely used in chat bots, tools for summarizing text and translation between languages.

Tokenization

Tokenization is one of the key concepts in NLP. It consists in the process of splitting text into smaller parts (tokens), easier for machines to understand. There are several methods to generate tokens (Mielke et al., 2021). The main used approach splits the text into individual words, which is specially efficient for languages with well defined words, like English. Character tokenization consists of splitting the text at a character level. This technique is useful in cases where the task at hand involves high granularity, for example spelling correction. Subword tokenization is used to split the text into pieces greater or equal to a character, but smaller than a full word. This method is especially effective for languages that create words through a concatenation process, such as German. For example, the word “sunshine” could be split into “sun” and “shine”. Tokenizing text involves several decisions, such as handling compound words or contractions, use or not use case sensitive methods, and the rules may vary according to the language Habert et al. (1998).

Lemmatization

Lemmatization is an NLP text preprocessing technique that groups several inflected forms of a word together. This technique aims to reduce words to their base dictionary form, also known as the lemma (Khyani et al., 2021). By grouping different inflected forms of a word together, lemmatization ensures that variations like ”running”, ”ran”, and ”runs” are all represented by a single form, ”run”. Processing words based on meaning not only helps machines understand text in a simpler way, but also decreases the overall vocabulary size, reducing the dimensionality of the feature space. This simplification leads to more efficient storage and computation, as well as potentially improving the performance of machine learning algorithms by mitigating the sparsity of the feature matrix.

Stemming

Stemming is an NLP text preprocessing technique that aims to reduce words to their root or base form, resorting to removing prefixes and suffixes. Unlike lemmatization, stemming does not reduce a word to its root meaning; instead it uses algorithms to cut word endings and other specific occurrences of characters (Khyani et al., 2021). Some words lose their meaning during the stemming process. For instance, the word "better" is represented by "bett". Nevertheless, using stemming benefits from the dimensionality reduction and simplification of the text, allowing a lower training cost in terms of computational resources.

2.3 Vectorization Methods

Vectorization methods are a set of techniques that aim to represent sequences of text, such as sentences, through numerical vectors that can be understood and processed by machine learning algorithms (Rani et al., 2022). These methods represent a crucial role in Natural Language Processing by representing words, sentences or documents in a uniform and numeric way (Rani et al., 2022). Several techniques have been developed over the years. The following paragraphs will delve deeper into some of the most famous vectorization methods.

One Hot Encoding (OHE)

One Hot Encoding (OHE) is the most straightforward method to represent words in text as numerical vectors (Rodríguez et al., 2018). It is commonly employed to convert categorical variables, such as words in a vocabulary, into numerical representations suitable for machine learning algorithms (Seger, 2018). In the context of text, One Hot Encoding represents each word by a binary vector, where each position represents a word in the vocabulary. If a word is present in a sentence, its corresponding position in the vector is marked with 1; otherwise, it is marked with 0. While simple to understand, OHE becomes inefficient for large vocabularies, resulting in very sparse vectors. Moreover, it does not account for word positions within the text or possible relationships between words.

Bag of Words (BoW)

The Bag of Words (BoW) approach shares similarities with One Hot Encoding (OHE) as both techniques disregard the positioning of words within the sentence (Qader et al., 2019). BoW starts by creating a vector where each position represents a word. For each word within the sentence, the related position in the vector is incremented by one. As a result, the vector contains the frequencies of each word in the vocabulary within the given sentence. While BoW captures more information than OHE by considering word frequency, it still falls short in capturing context, structure, and order.

N-Grams

The N-Grams approach is a vectorization technique used to analyze and represent sequences of contiguous items, usually words or characters, within a text. To employ n-grams it is necessary to split the text into tokens, that can be either words, characters, subwords, etc. The second step consists of extracting the N consecutive tokens from the text (Majumder et al., 2002). Finally, count the number of occurrences of each n-gram within the text. If N=2 this technique is called bigram, and trigram if N=3. Unlike BoW and OHE, n-grams are able to capture local context and dependencies between adjacent words or characters, providing more contextual information. This technique also preserves the order of words or characters within the text, allowing for the capture of sequential patterns and relationships.

Term Frequency - Inverse Document Frequency (TF-IDF)

The TF-IDF approach is a mix of two different metrics (Salton et al., 1975). Term Frequency (TF) measures the number of occurrences that a word has within a text. On the other hand, the Inverse Term Frequency (IDF) measures the degree of importance of a word given its presence in the corpus. This approach follows the philosophy that words that appear in many documents end up not being as important and differentiated as less common words. Therefore, common words are considered less important, while rare words are given special attention. TF-IDF is the product between TF and IDF, meaning that this metric accounts for the frequency of the term within a specific sentence, while capturing its general use across the full corpus.

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t) \tag{1}$$

where

$$\text{TF}(t, d) = \frac{\text{Number of occurrences of term } t \text{ in document } d}{\text{Total number of terms in document } d}$$

and

$$\text{IDF}(t) = \log\left(\frac{1 + N}{1 + \text{DF}(t)}\right)$$

2.4 Embeddings

Embeddings are representations of words, sentences, or entities as dense vectors in a continuous vector space (Gutiérrez and Keith, 2019). In Machine Learning (ML) and NLP embeddings are widely used to capture semantic relationships between words. Embeddings are fundamental given that machines can not understand words as humans do, therefore transforming words into vectors is a crucial task when working with text modeling. Word embeddings, in particular, are learned representations of words mapped in a vector space given its semantics. Words with similar semantics tend to be closer in the vector space than words with diverging meanings (Stanford, 2024). Usually word embeddings are obtained through unsupervised algorithms trained on large corpora of text.

Word2Vec

Word2Vec is one of the most famous embeddings in NLP, it was developed by Tomáš Mikolov in 2010. Later in 2013 it was published in two papers and patented by a team of researchers at Google (Mikolov et al., 2013a,b). The structure behind Word2Vec is a simple recurrent neural network with one hidden layer for language modeling. Word2Vec takes a large corpus of text as input and produces a vectorial space, usually with hundreds of dimensions. Each word is represented by a vector within that space. Word2Vec consists of two primary models: Continuous Bag of Words (CBoW) and Skip-Gram.

The input of the CBoW model consists of OHE vectors representing the context of a target word. The context is created by analyzing the surrounding words of the target throughout the corpus. The output layer consists of a vector of probability distribution for each word within the embedding space. In the end, a softmax activation function is used to transform the raw scores into probabilities. The goal of this model is to predict the target word based on its context words.

Word2Vec with N-grams extends traditional Word2Vec models by considering sequences of N contiguous words as input. This approach allows for the capture of contextual information and semantic relationships between words within the corpus. By training neural networks to predict target words or N-grams based on their surrounding context N-grams, Word2Vec with N-grams learns distributed representations of words and N-grams in a continuous vector space. These embeddings encode semantic similarities and relationships, enabling applications such as word similarity analysis, text classification, language modeling, and information retrieval.

GloVe

GloVe is an embedding presented in 2014, by a team of Stanford researchers composed of Jeffrey Pennington, Richard Socher, and Christopher D. Manning (Pennington et al., 2014). Following the description of the official website (Stanford, 2024), GloVe is an unsupervised learning algorithm that aims to represent words as vectors given their semantic value. The main idea behind GloVe is to leverage the statistical information present in a corpus to create a dense, continuous vector space where the relationships between words are captured. The model is trained on non-zero entrances of a word-to-word co-occurrence matrix. This matrix shows how frequently two words appear together in a given corpus. For large corpus the creation of the matrix may be computationally expensive, but it is a one-time effort.

The vectors generated by GloVe can serve as input to the Euclidean Distance to find the closest set of words within the vector space. Furthermore, the vectors are built so that they allow arithmetic operations between words.

FastText

FastText is an embedding developed by Facebook Artificial Intelligence Research (FAIR) in 2016 (Bojanowski et al., 2017). FastText extends the concept and application of traditional embeddings, such as Word2Vec, by incorporating subword information. By introducing subwords into the model, FastText is able to effectively handle rare words, misspellings, and morphological variations. The biggest difference relative to Word2Vec is the input. While in Word2Vec each word is represented by a single vector, in FastText a word is represented by a bag of n vectors, each corresponding to a subword. For example, the word "where" could be represented by ¡wh, whe, her, ere, re¿. FastText is a highly effective algorithm in terms of computational power. In an article published in 2016 by Joulin et al. (2016), the authors say "We can train fastText on more than one billion words in less than ten minutes using a standard multicore CPU, and classify half a million sentences among 312K classes in less than a minute.", thus showing the efficiency of the training and prediction power.

2.5 Machine Learning

Supervised Learning

Supervised Learning is the most common approach in Machine Learning (ML). It consists of training algorithms based on labeled training data to make predictions or decisions. The term "supervised" means that the algorithm is guided through the training process by analyzing the true label of the observations, thus allowing the model to learn relationships between input features and the target labels. The goal of these class models is to predict the label given the input data.

Unsupervised Learning

Unsupervised Learning is an approach of ML that focuses on training models to identify patterns in unlabeled data. Unlike supervised learning, where the algorithms learn from labeled data, unsupervised learning aims to find patterns in data lacking predefined labels and outputs. This kind of ML is specially suitable for domains where getting labeled data is highly difficult or expensive. There are different types of Unsupervised Learning problems. Clustering focuses in grouping observations in way that objects inside the same group are more similar to each other than to those in other groups. Another application of unsupervised learning is dimensionality reduction, which consists of the process of representing a dataset reducing the number of features, while conserving as mush information as possible.

Semi-Supervised Learning

This concept is a combination of Supervised Learning and Unsupervised Learning. These ML models are based on small amounts of labeled data and large amounts of unlabeled data, taking advantage of both types of learning. Given a small amount of labeled examples, semi-supervised learning tries to improve learning accuracy in contrast to learning solely on unlabeled data, while overcoming the difficulty of limited amounts of available labeled data.

Grid Search

Grid search is a well-known technique used for hyperparameter optimization that involves an exhaustive search through a specified subset of the hyperparameter space. However, Grid search can be a very expensive process computationally, mainly with complex models, high-dimensional datasets, and large hyperparameter space. To address complexity and efficiency problem, several alternative methods have been developed: random search, gradient-based optimization, and Bayesian optimization. Among them, Optuna stands out. Optuna is an open-source hyperparameter optimization framework to automate hyperparameter search release in 2019 by a team of researchers of the University of Fukuoka (Akiba et al., 2019; Optuna, 2024a,c). Optuna comes with visualization capabilities to understand the evolution of the study and the optimization of hyperparameters through a dashboard available at (Optuna, 2024b).

2.6 Deep Learning

Deep learning is a subset of machine learning, that uses neural networks, structures with multiple layers, to model complex patterns in data. The training of deep learning models follows an iterative process, where in each iteration (epoch), the goal is to optimize the model's parameters given a loss function, improving the results over the training history.

Neural Networks

Neural networks (NNs) form a class of models that try to somewhat mimic the proceedings of the brain by using the architecture of the computer in an attempt to solve a specified problem. Neurons, together with the connections between them, form a layered structure in which the incoming data to the problems are given at one layer and emergent results are issued at another.

In NLP, several neural network structures exist to solve different problems effectively. Recurrent Neural Networks (RNNs) are designed for sequential data and, as a result, allow the preservation of temporal data within the model. Variants like Long Short-Term Memory (LSTM) networks incorporate memory cells that store information over long periods (Hochreiter and Schmidhuber, 1997). Shortly after, Bidirectional LSTMs (biLSTM) were introduced, this network process data in both forward and backward directions to capture context from both past and future states (Schuster and Paliwal, 1997).

Gated Recurrent Units (GRUs) are simplified versions of LSTMs; The core idea is to merge the forget and input gates into one update gate, making computation efficient without a high cost in performance (Cho et al., 2014).

Convolutional Neural Networks (CNNs), originally designed for image processing, have been adapted for use in NLP tasks by applying filters over input data to catch local patterns, thereby learning good representations of text as a spatial hierarchy of features (Ojo et al., 2022; LeCun et al., 1998).

Large Language Models

Large Language Models represent a special class of neural networks models, that heavily rely on large corpus and significant computational resources towards yielding the best performance in various NLP tasks. A pivotal innovation which gave birth to one of the most important architectures, attention mechanism, was introduced in a paper entitled "Attention is All You Need" (Vaswani et al., 2017). The concept of a transformer was included in this paper, which revolutionized the training of deep learning models. This mechanism allows the model to include in its memory the importance of words in a sentence to make contextual relations and improves the performance of tasks like translation, summarizing, or question answering.

BERT

BERT, standing for Bidirectional Encoder Representations from Transformers, is a language model introduced by Google (Devlin, 2018). BERT was designed to understand words given their preceding and succeeding content, by analyzing text in a bidirectional way. BERT is a generalist model that can perform a large number of NLP tasks. Furthermore, BERT can easily be fine-tuned for domain specific tasks presenting high performance across many domains. BERT is based on the attention mechanism described above to weight the importance of words in the text. Multiple approaches of BERT have been developed after its release. For instance, DistilBERT was developed with the goal of making BERT a lighter model, suited for real word prediction. As the article published by Sanh et al. (2020) explains, DistilBERT retains 97% of BERT's knowledge, while being 60% faster and using 40% fewer parameters. RoBERTa was a engineering improvement compared to BERT, it maintains BERT's base architecture, but carefully measures the impact of many key hyperparameters and training data size (Liu et al., 2019). Moreover, RoBERTa was trained on a widely larger dataset. ALBERT was introduced with the same goal as DistilBERT, improve the scalability of BERT (Lan et al., 2020). ALBERT does so by sharing parameters across layers, reducing memory usage, and reduces the number of trained tasks. This model proved to perform as well as BERT, while being more efficient.

Generative Pre-trained Transformers

Generative Pre-trained Transformers (GPT) series represent a groundbreaking pivot in NLP. This new class of models, developed by OpenAI in 2018, are trained in two distinct phases. The model starts by learning on diverse corpus of unlabeled text, followed by a discriminate fine-tune at specific tasks, using labeled data (Radford, 2018). There has been a great adoption of GPT models and these keep evolving at a very fast rate. GPT-2 introduced larger models with up to 1.5 billion parameters, enhancing text generation quality and performance (Radford et al., 2019). GPT-3, with its 175 billion parameters, marked a significant leap, demonstrating advanced capabilities in few-shot learning and context understanding (Brown, 2020). The latest model, GPT-4, is able to interpret both images and text outperforming humans in multiple academic and professional tasks (OpenAI et al., 2024).

Prompt Engineering

Prompt engineering is an important approach in the best utilization of large language models. It involves crafting and fine-tuning inputs to lead the model towards desired responses. This may range from simple prompt formulation to sophisticated techniques using particular instructions, examples, or even chaining prompts together (Gao, 2023). The quality of the prompt directly impacts the relevance, coherence, and accuracy of the model's response. In actual implementation, prompt engineering serves to fine-tune the model's behavior for specific applications, whether that be content generation, data extraction, or task automation. This can often involve iterative testing and adjustment because the output of the model may be sensitive to slight changes in wording or context.

3 Models in Hate Speech Detection

In this chapter, works that utilize machine learning (ML), deep learning (DL), and generative AI (GenAI) for the task of hate speech detection are analyzed. We focus on the different models employed in this domain, analyzing their performance and highlighting the most effective combinations of techniques.

3.1 Machine Learning Models

The work published by Shawkat (2023) compares different Machine Learning models in combination with different text representation methods, sampling methods, on three datasets. The sampling method that presents best results is the oversampling. The model with best performance on the oversampled datasets is the Random Forest (RF) allied with n-grams, reaching values rounding 0.96 on both accuracy and F1-score. For the original datasets and the undersampled datasets, Support Vector Machines (SVM) and Naive Bayes (NB) associated with Term Frequency Inverse Document Frequency (TF-IDF) present the best results, with accuracies ranging between 0.82 and 0.91.

Aljarah et al. (2021) present an innovative machine learning-based approach proposed to tackle the issue of cyber hate speech within the Arabic context on Twitter. The dataset, obtained through the Twitter streaming API, underwent thorough processing and was integrated into various machine learning algorithms, including Support Vector Machines (SVM), Naive Bayes (NB), Decision Trees (DT), and Random Forest (RF), utilizing Python and the Spyder development framework. Notably, the Random Forest classifier demonstrated superior performance compared to the other classifiers. Upon analyzing the most influential features associated with hate tweets, key contributors were identified as racism, emigrant-related terms, and mentions of the word "God".

A study published by Adoum Sanoussi et al. (2022) uses data gathered from popular Facebook pages counting to 14,000 comments. These comments were carefully classified into four distinct categories: hate, offense, insult, and neutral. To ensure the credibility of the data, authors applied advanced Natural Language Processing techniques (NLP) in the cleaning process. In addition, three different word embedding methods, namely Word2Vec, Doc2Vec, and FastText, were incorporated. In the final stage of the research, four Machine Learning approaches were employed for effective classification: Logistic Regression (LR), Support Vector Machine (SVM), Random Forest (RF), and K-Nearest

Neighbours (KNN). The results indicated that utilizing FastText features as input provided the most successful configuration.

A recent study published by Saifullah et al. (2024), leverages semi-supervised ensemble techniques to fight against the cost of labeling hate speech data. Text annotation is a vital aspect of natural language processing, but manual annotation can be limited by subjectivity and inefficiency. To address these concerns, this study presents an automated approach to annotation using a variety of machine learning techniques. The approach focuses on an ensemble algorithm that combines meta-learning and meta-vectorization techniques, incorporating semi-supervised learning to identify hate speech. The ensemble is composed of diverse machine learning algorithms, such as Support Vector Machine (SVM), Decision Tree (DT), K-Nearest Neighbors (KNN), and Naive Bayes (NB), and utilizes advanced text extraction methods like Word2Vec and TF-IDF. Empirical validation was conducted on a dataset of 13,169 Indonesian YouTube comments, utilizing a Stemming approach and integrating data from Sastrawi and an additional 2,245 words. In a departure from the conventional 80% labeling approach, the proposed model employs semi-supervised learning with only 5%, 10%, and 20% labeled data. This unique hybrid learning paradigm allows the model to generalize and accurately predict with limited annotated data. The experimental results showcase the remarkable performance of the KNN-Word2Vec model, achieving an accuracy of 96.9% with just 5% labeled data and a 0.9 threshold. SVM and DT also demonstrate accuracy surpassing 90% in multiple scenarios, underscoring the robustness of the proposed methodology in detecting hate speech within Indonesian YouTube comments.

Text related tasks have been a big research area in past years, therefore a multitude of techniques have been studied. Traditional Machine Learning models have been surpassed by Deep Learning approaches (Shawkat, 2023; Shibly et al., 2021; Fernando et al., 2022; Shibly et al., 2022).

3.2 Deep Learning Models

Given this scenario we now analyze works that compare different Deep Learning models. A paper published by Paul and Bora (2021) compares the performances of Long Short Term Memory (LSTM) and Bidirectional Long Short Term Memory models (BiLSTM) at the hate speech detection task. This class of models was designed to identify patterns on sequential data, such as time series or text. The results show that both models perform well, LSTM outperforms BiLSTM in accuracy, precision, and F1-score, while BiLSTM presented a better recall. A study performed on a dataset of 9316 arabic language comments, compares 4 neuronal network models: Convolutional Neural Networks (CNN), Gated Recurrent Units (GRU), Hybrid Model (CNN + GRU), and Bidirectional Encoder Representation from Transformers (BERT) (Alshalan and Al-Khalifa, 2020). The experimental outcomes in both the dataset under consideration and an external dataset revealed that the CNN model exhibited superior performance, achieving an F1-score of 0.79 and an area under the receiver operating characteristic curve (AUROC) of 0.89.

A study focused on the comparison of ML and DL models to detect hate speech against women shows that the best performing model was a transformer based model with CNN and MLP combined (Hasan et al., 2022). The study also shows that a combination of TF-IDF, MLP, SVM, and XGB also produces strong classification metrics. The two machine learning models (SVM and RF) were outperformed by the deep learning approaches, showing, once again, their better overall performance in hate speech detection.

Alatawi et al. (2021) uses two DL models to detect hate speech related to white-supremacy. Both bidirectional Long Short-Term Memory (BiLSTM) and Bidirectional Encoder Representations from Transformers (BERT), are evaluated, achieving F1-scores of 0.75 and 0.80, respectively, on a combined, balanced dataset from Twitter and a Stormfront forum. This study indicates that BERT models tend to perform better than BiLSTM.

Lately, the dominant approach in text related tasks has been the application of transformers. A study by Mutanga et al. (2020) compares baseline deep learning models using transformers for detecting hate speech. The authors use a database of 24783 labeled comments to compare the model's classification performances, DistilBERT presented the best result, with an accuracy of 0.92 and a precision of 0.75, outperforming BERT, RoBERTa, XLNet, and LSTM with attention mechanisms.

A study by Toktarova et al. (2023) compares several Machine Learning and Deep Learning approaches on three databases containing hate speech, offensive speech, and cyberbullying. From the three DL models, CNN presented the worst performance. LSTM and BiLSTM behaved similarly in terms of prediction metrics, nevertheless BiLSTM outperformed LSTM on all datasets. Although the ML models were outperformed by DL in all datasets, the author concludes that different classifiers - Naïve Bayes, KNN, and SVM

- are effective for spotting hate speech. They use features from text data such as Bag of Words, TF-IDF, or word embeddings. Additionally, deep learning methods like LSTM, BiLSTM, and CNN are explored for their skill in understanding complex text patterns and relationships. This exploration results in better classification performance by better understanding the complexity of language, improving the identification of hate speech.

A wide range of articles employs BERT like models and study their performances on different databases, associated with other DL models, multi or mono language BERT's, with and without fine-tuning. Multiple studies show that using BERT, as an embedding or fine-tuned BERT, for classification tasks has shown to be the most efficient approach in Hate Speech detection across multiple research works (Mozafari et al., 2020,?; Dowlagar and Mamidi, 2021; Lavergne et al., 2020).

3.3 Generative Artificial Intelligence

The newly generative artificial intelligence (GenAI) models have proved to be efficient in many different tasks. The costs associated with using these models for language tasks is high when it comes to scalability, this problem associated with the recency of these models results in little available literature. Nevertheless, recent research has used GenAI in different parts of the Hate Speech detection process. The greatest problem when it comes to training models to identify Hate Speech is the few labeled data available, and the bias that these datasets present, given the human labeling process. The following studies resort to GenAI to solve this problem, by generating Hate related text databases.

The study by Hartvigsen et al. (2022) addresses issues in toxic language detection systems, highlighting their tendency to inaccurately flag text mentioning minority groups and struggle with implicitly toxic language. To mitigate this, researchers introduce ToxiGen, a large-scale dataset featuring 274k statements about 13 minority groups. They employ a prompting framework and an adversarial classifier-in-the-loop method to generate subtle toxic and benign text using a pre-trained language model. Human evaluation shows difficulty in distinguishing machine-generated text from human-written content, with 94.5% of toxic examples labeled as hate speech. Fine-tuning a toxicity classifier on ToxiGen substantially enhances its performance on human-written data. This research demonstrates ToxiGen's potential in combating machine-generated toxicity and improving classifier accuracy on challenging subsets.

Wullach et al. (2020) focuses on enhancing hate speech detection through deep learning and large-scale data generation is commendable. By acknowledging the criticality of this issue in social media, the study addresses the limitations of existing methods, particularly their struggle with generalizing to new hate speech sequences. Introducing a dataset containing one million hate and nonhate sequences, generated via a deep generative model, is a significant step forward. The utilization of this extensive dataset to train a well-established deep learning detector showcases promising improvements across multiple hate speech datasets.

This approach highlights the potential of larger and more diverse data in refining deep learning models, crucial for their adaptability beyond specific datasets. Lastly, the study by Cao and Lee (2020) presents an innovative approach, HateGAN, addressing the challenge of imbalanced datasets in online hate speech detection. The reliance on supervised learning methods often leads to poor performance due to imbalances in labeled datasets. HateGAN, a deep generative reinforcement learning model, offers a solution by augmenting existing datasets with generated hateful tweets. The study conducts extensive experiments augmenting two prevalent hate speech detection datasets with HateGAN-generated tweets. Results demonstrate consistent improvements in hate speech class detection across various classifiers and datasets. Notably, an average 5% enhancement in hate class F1 scores is observed among state-of-the-art hate speech classifiers. The paper further supplements its findings with case studies validating the diversity, coherence, and relevance of HateGAN-generated hate speech.

GenAI can also be used for classification tasks, as the following studies show. A study by Wang and Chang (2022) explores zero-shot prompt-based toxicity detection using generative methods, sidestepping explicit instructions to language models. Trials encompass varied prompt engineering approaches, showcasing strengths in toxicity detection across three social media datasets. The research emphasizes the efficacy of this generative classification, touching on self-diagnosis and ethical considerations in detecting subtle undesirable content from text.

Li et al. (2024) resort to ChatGPT to identify Hate, Offensive, and Toxic (HOT) content in text. The authors study the influence of prompt engineering in the model's performance, and the stability of the outputs. The results show that the initial prompt has a strong impact on the output. The first prompt presents accuracies of 0.76, 0.77, 0.63 for Hate, Offensive, Toxic, respectively. The outputs demonstrate that ChatGPT presents an impressive precision value, well above 0.90, for observations that do not contain HOT content. For observations containing HOT content the precision value drops drastically. The second prompt used by the authors shows impressive results. The accuracies increase to 0.87, 0.82, and 0.75, for HOT, respectively. The precision levels also increase for observations containing HOT content. Furthermore, the authors develop an experience, asking chatGPT to classify the same comments several times. The outputs served as input to the Krippendorff's metric, showing agreement values greater than 90%.

A team of Brazilian researchers employed chatGPT to classify Portuguese Hate Speech text (Oliveira et al., 2023). The authors compare the performance of ChatGPT turbo 3.5 model with BERTimabu, a BERT based model for Portuguese language. Once again the results show that the initial prompt has a preponderant influence over the output of the model. ChatGPT with prompt 1 outperformed BERTimabu in terms of F1-score for the Hate class, while prompt 2 outperformed BERTimbau in predicting the noHate class.

4 Methodology

The development of data mining projects is a creative process that can fallow a multitude of structures, meaning that the reason for success or failure is harder to determine, due to lack of methodology. To solve this issue the Cross Industry Standard Process for Data Mining (CRISP-DM) was introduced by Wirth and Hipp (2000). The current work is structured accordingly to the 6 predefined phases of CRISP-DM: (1) Business Understanding, (2) Data Understanding, (3) Data Preparation, (4) Modelling, (5) Evaluation, (6) Deployment (Figure 1). Each phase is explained bellow.

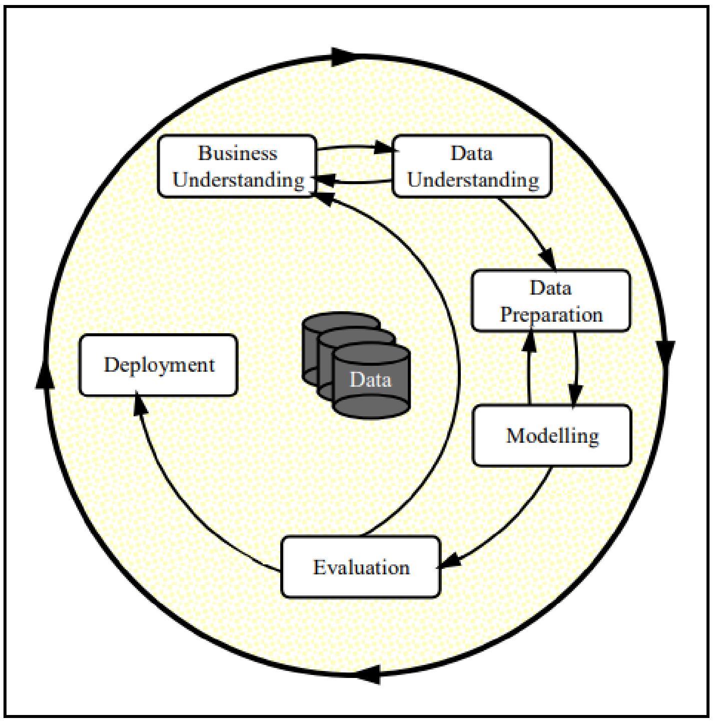


Figure 1: Phases of the Cross Industry Standard Process for Data Mining

The first phase, **Business Understanding**, is focused on understanding the problem at hand, its background and context, available resources, terminology, constraints, risks, contingencies, costs, and benefits. To asses the dimensions components of this project, the background, context, and terminology are presented in the first and second sections of the current work, "Introduction" and "Concepts". On the third section, a literature review is carried out to understand the technology and resources already available for hate speech detection.

The **Data Understanding** is the second phase of CRISP-DM and is reflected in section "Data Processing Pipeline" of the current work. This phase is divided into smaller sub phases: (1) Initial Data Collection, (2) Data description, (3) Data exploration, and (4) Quality Assessment. Throughout this phase, the data collection process is described; Furthermore, key information regarding the dataset is presented, allowing for a better understanding of the case study.

The third phase of the employed methodology regards to **Data Preparation**. Selecting, cleaning, integrating, and transforming data are key general steps in every data mining project. Few are the models that can deal with real world information without any kind of transformation. In the case of text classification models, the data should be processed resorting natural language processing (NLP) techniques. In the data preparation process, the text is first tokenized, followed by the application of text regularization techniques. Furthermore, sentences were converted from vectors of text tokens to numeric arrays, resorting to vectorization and embedding techniques. In this study the data is encoded using the Term Frequency-inverse Document Frequency (TF-IDF) vectorization method and three state of the art embedding techniques: FastText, Word2Vec, and GloVe. For each technique, three datasets were created, differing in the dimension of the vectors. Vector dimensions ranging from 100 to 10,000 were utilized to assess their impact on performance.

Phase four, **Modelling**, is responsible for the selection, training, and optimization of models. Both machine learning (ML) and deep learning (DL) models were trained to assess their performance in terms of accuracy in detecting hate speech and resource consumption.

The selected Machine Learning models to perform this study were: (1) Logistic Regression (LR); (2) XGBoost; (3) LighGBM (LightGBM); (4) K-Nearest Neighbours (KNN); (5) Naïve Bayes (NB). These models will be trained using different embedding techniques, dimensions, and hyperparameters. This study leverages Optuna (Akiba et al., 2019), a hyperparameter optimization framework, to efficiently search within the parameter space resorting to Bayesian optimization.

Following, a transfer learning approach is employed by leveraging pre-trained BERT-based models, including: (1) DistilBERT; (2) RoBERT; (3) alBERT. Each models undergoes a fine-tuning process with the goal of adjusting its weights for the hate speech detection task. The fine-tuning process is conducted over 10 epochs for each model, incorporating techniques such as weight decay, learning rate scheduling, and label smoothing to mitigate overfitting and enhance generalization.

Finally, Generative Pre-Trained Transformer (GPT) models were employed with the goal of identifying the starting and ending indexes of hate content within a sentence. The interaction with these models was made possible by using OpenAI's API, providing a user friendly way to programmatically interact with the models. A fine-tuning process was carried out for the GPT-3.5 Turbo and GPT-4o Mini.

The **Evaluation** phase is presented in the "Results" section. Several metrics were employed to measure the performance of the classification models: Accuracy, Recall, Precision, F1-Score, and AUC-ROC. A discussion of the performance of the machine learning models in combination with the the vectorization or embedding models used is presented. A similar evaluation was carried out for the deep learning models. The performance of the GPT models was evaluated and compared, stating the importance of fine-tuning this class of models. A general discussion of the results is presented at the end of the section, pointing out strengths and weaknesses of the proposed algorithms.

The final phase, **Deployment**, aims to make the developed solution available for general use, following state-of-the-art technologies. Python based frameworks for web development, django (DjangoProject, 2024) and django-rest-framework (DjangoREST-Framework, 2024), were employed to develop the back-end of the application. The API implements a POST method that allows users to invoke the specific models on demand, following a software-as-a-service (SaaS) approach. The user is able to make a POST request sending a sentence or list of sentences and select a model to classify the sentences as hateful or non-hateful. The API also implements a GET method allowing users to download the models for later use in their own projects.

To make the solution available for a greater audience, a React (React, 2024) front-end application is implemented, allowing interaction with the trained models in an interactive way, through a graphical user interface (GUI). The front-end is used to structure the requests, sending them to the back-end, and displaying the responses in a user friendly way.

5 Data Processing Pipeline

5.1 Source and Origin

Hate Speech Classification

This thesis employs a dataset by Mody et al. (2023) that aims at identifying hate speech in social media through text analysis. It comprises all sorts of content: emojis and emoticons, hashtags, slang, contractions, and many others, thus suitable for finding all modern variants of hate speech. Each of them is marked as either hateful or non-hateful; Such labeling, or categorization, is extremely useful when a machine learning algorithm is being trained, for the purpose of addressing toxic speech online, on the Internet.

The authors gathered the dataset from various platforms, such as Kaggle and GitHub, and afterwards applied a rigorous cleaning process. This involves expanding contracted words and emoticons to make all the text samples in this dataset consistent. As much as it is possible for this dataset to be bias-free, it is generally neutral; proper names of specific people and organizations have been avoided where necessary to prevent the harming or endangerment of persons or for ethical issues. Thus, it is an appropriate and effective one for various types of research and in practice.

The dataset was retrieved form Kaggle (2023). The repository includes two data sets, a basic dataset and a augmented one through oversampling. In this work we focus on the basic dataset.

Hate Speech Detection

HateXplain is a benchmark dataset, curated by Mathew et al. (2020), designed for explainable hate speech detection, particularly in the context of social media. It comprises more than 20,000 samples such as posts annotated as hate, or offensive, or normal, with additional tags showing the target groups of users and the rationale behind the categorisation. It also contains human-annotated explanations of which portion of the text is relevant to the labeling decision. That is why it is useful not only for training hate speech classification models, but also for indicating the location of hate speech in the text.

5.2 Data Cleaning

Hate Speech Classification

The cleaning of the social media dataset began with the first two steps entailing data preparation and preprocessing. First, the loading of the dataset was done, followed by the cleaning of the erroneous labels for use. The dataset had an initial 440,899 entries. Afterwards, the labels were corrected and converted into integers so that all the data would be in correct format before moving into the next cleaning steps.

The first step of cleaning was removing the observations that are not of interest. The count was then 440,423 entries after removing rows with missing texts. Accordingly, duplicate entries by text content should be removed, and it goes down to 417,560 entries. It was performed to search for phrases with URLs; in this conditions no sentences were found.

The second phases consisted of changing specific text elements without removing observations. This included anonymizing the usernames by replacing them with a placeholder '@user', convergence of user-related tags, text normalization by putting it into lower case, removal of certain terms like 'rt', irrelevant punctuation removal, and removal of stop words. After those changes, empty and duplicated sentences were removed once more from the dataset, reducing it to 407,963 entries following cleanup.

The last stage of the cleaning processes regarded the tokenization and lemmatization techniques. These techniques further reduced the dataset to 407,548 entries because some duplicated values were generated. The final analysis conducted on cleaned data showed a label distribution with 81.79% non-hateful entries and 18.21% of hateful entries.

Hate Speech Detection

The HateXplain dataset is structured in JSON file composed of 20,148 documents, each relative to a post. The first cleaning step involved ensuring that all the documents were labelled as hateful. Since each post is labeled by multiple annotators, only those unanimously labeled as hateful were retained, reducing the dataset size to 2,960 documents. The second step involved replicating this process for the rationales, ensuring that they were only included if every index was unanimously labeled. After these two cleaning steps, the dataset was further reduced to 627 observations. These processes were implemented to enhance the quality and reliability of the data.

5.3 Data Description

Hate Speech Classification

Table 1 reflects the dimensions and class distribution of the dataset after the cleaning process described above. The dataset is composed of 407,548 sentences, containing 121,588 unique words, reflecting the richness and variability of the language used in social media communications. On average, a sentence is composed of 21 words. The classes are considerably unbalanced, almost in a 1 to 4 ratio, where non-hateful represent 82% of the data, while hateful observations have a relative frequency of 18%.

Class	Absolute	Relative
Non-Hateful	333343	81.8%
Hateful	74205	18.2%
Total	407548	100%

Table 1: Original dataset dimensions and class distribution.

Figure 2 shows two word clouds corresponding to the non-hateful (left) and hateful classes (right), respectively. These representations provide an immediate intuitive understanding of the most frequent words occurring in each class.

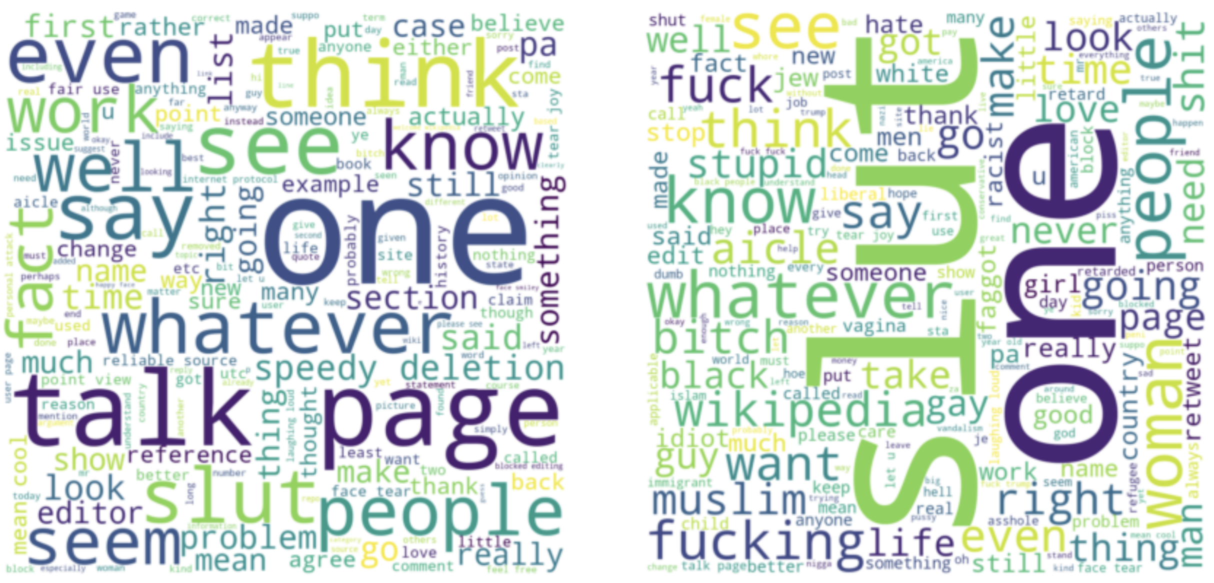
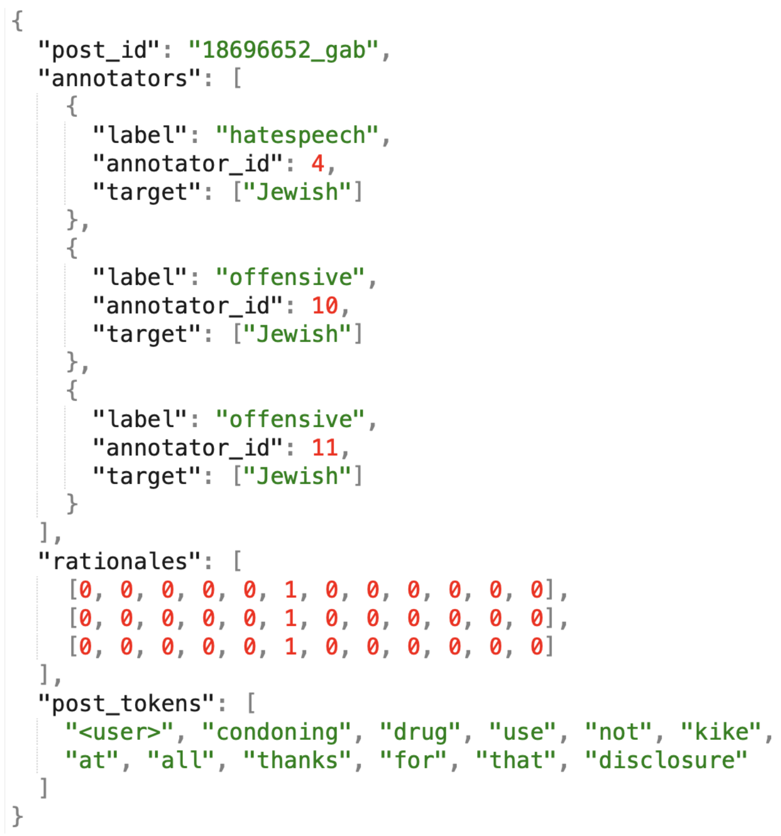


Figure 2: Word Clouds: Non-Hateful (left) and Hateful (right) sentences

Hate Speech Detection

The HateXplain dataset is distinctive in its ability to provide insights into hate speech across three dimensions: the type of hate speech, its intended target and the reasons behind the labeling decision. The dataset is structured as a JSON file, as illustrated in Figure 3, where each document represents a post and contains the following key fields:

- **Post ID**: A unique identifier for each post.
- **Post Tokens**: The tokenized version of the post's text.
- **Label**: The type of speech identified by the annotators, categorized as "hateful," "offensive," or "normal".
- **Target**: The group or individual targeted by the hate speech or offensive content.
- **Rationales**: A binary mask representing the rationale behind each annotator's decision, indicating which words influenced the labeling of the post as hateful or offensive.



```
{
  "post_id": "18696652_gab",
  "annotators": [
    {
      "label": "hatespeech",
      "annotator_id": 4,
      "target": ["Jewish"]
    },
    {
      "label": "offensive",
      "annotator_id": 10,
      "target": ["Jewish"]
    },
    {
      "label": "offensive",
      "annotator_id": 11,
      "target": ["Jewish"]
    }
  ],
  "rationales": [
    [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
  ],
  "post_tokens": [
    "<user>", "condoning", "drug", "use", "not", "kike",
    "at", "all", "thanks", "for", "that", "disclosure"
  ]
}
```

Figure 3: Example document of the HateXplain dataset

5.4 Data Transformation

Hate Speech Classification

The data transformation phase aims to prepare the dataset for modeling, ensuring the integrity and neutrality of the data throughout the process. Table 1 shows that the target classes are unbalanced. Past studies have shown that resampling methods play an important role when working with text classification datasets (Liu, 2004).

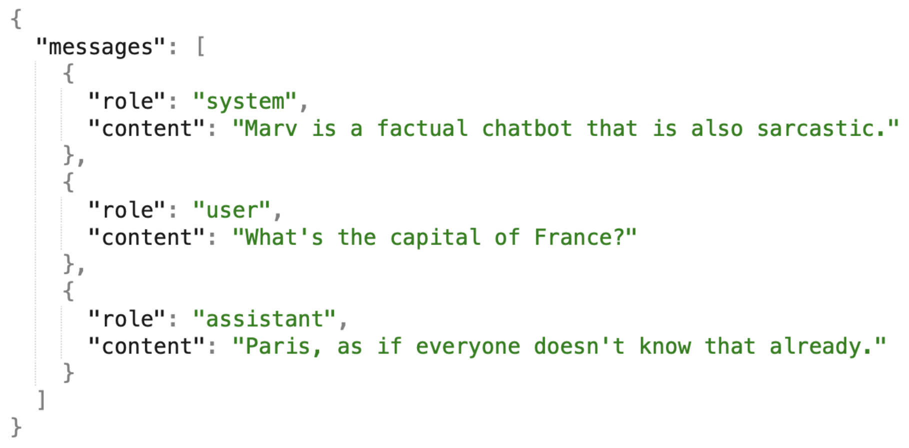
In this thesis, the goal is to train models to accurately identify hate speech in text. A balanced dataset is key to correctly interpret results without bias. Several strategies exist to handle class distribution imbalance, such as Random Oversampling, Synthetic Minority Over-sampling Technique, Adaptive Synthetic, and Random Undersampling. Generally, it has been noticed that Random Oversampling (ROS) happens to be one of the finest techniques (Rathpisey and Adji, 2019; HENG, 2020; Putra and Nurjanah, 2020). However, though ROS will improve the overall performance of the models, the same is achieved only by replicating the minority class and results may be improved only partially. Another downside of ROS is that it possibly distorts the real distribution of vocabulary across documents.

In this study, we resort to the Random Undersampling technique. This approach was chosen to preserve the original vocabulary distribution while reducing the dataset size, enabling the application of more word vectorization techniques, model variations, and hyper-parameter combinations.

Hateful content is the minority class with a absolute frequency of 74,205. Therefore, after applying the Random Undersampling, the dataset was reduced to 148,410 observations. Following, the dataset was split into training (80%) and testing (20%) sets, with dimensions of 118,728 and 29,682, respectively. The next step involved converting the text into numeric arrays using four vectorization techniques: Word2Vec, FastText, GloVe, and TF-IDF. The first three methods are embeddings, requiring a training phase to generate word representations. We chose to experiment with different vectorization methods and sizes, ranging between 300 and 10000 parameters, alongside various models to identify the most effective approach for hate speech detection. The embeddings were trained for vector sizes of 300, 500 and 1,000 parameters, while TF-IDF was calculated for sizes 3,000, 5,000, 1,0000. Transfer learning was intentionally avoided to focus on understanding the performance of each vectorization technique using only the dataset at hand, ensuring that our findings are tailored specifically to this problem.

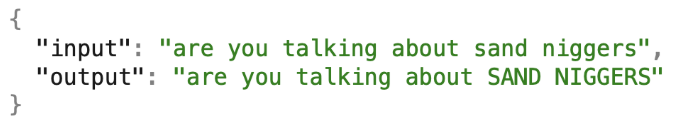
Hate Speech Detection

As shown by Figure 3, each database document contains the post tokens along with a list of binary arrays, that represent the rational behind the labelling decision of the annotators. These rationals are used in the data preparation phase. Each post token is uppercased if the corresponding index in the rational array is equal to 1. By implementing this method to the post tokens it is possible to reduce the dimensionality of the data, by merging the post tokens and the rational arrays into one single string. Following, the dataset was split into two samples, train and test, with a 80% / 20% division ratio, respectively. As Figure 4 shows, the documents of the train sample were structured according to the openAI's fine-tuning API documentation (OpenAI, 2024). Each document is composed of a list of messages, each with a specific purpose in the fine-tuning process. This process will be explained in greater detail in the Training Process section. The test sample follows a standard structure, as Figure 5 shows. The "input" field holds the lowercase sentence, while the "output" field contains the uppercased sentence generated based on the rationals.



```
{
  "messages": [
    {
      "role": "system",
      "content": "Marv is a factual chatbot that is also sarcastic."
    },
    {
      "role": "user",
      "content": "What's the capital of France?"
    },
    {
      "role": "assistant",
      "content": "Paris, as if everyone doesn't know that already."
    }
  ]
}
```

Figure 4: GPT fine-tune input file structure



```
{
  "input": "are you talking about sand niggers",
  "output": "are you talking about SAND NIGGERS"
}
```

Figure 5: Validation dataset file structure

6 Training Process

6.1 Machine Learning

Raw Models

This phase aims to find the best performing combination of machine learning models and vectorization techniques for hate speech detection. In the data transformation phase, a total of 12 datasets were encoded following distinct techniques. Five machine learning models, Logistic Regression, Light Gradient Boosting Machine, Extreme Gradient Boosting, Naive Bayes, and K-Nearest Neighbours, were trained on the previously generated datasets, counting to a total of 60 combinations. The first training process consisted on, for each combination, training the model using its default parameters, ensuring a fair comparison between results. To measure the overall performance of the trained models the accuracy metric was taken into consideration.

Hyperparametrization

For the hyperparameter optimization phase, Optuna, a powerful framework for hyperparameter search, was employed to fine-tune the best performing model and encoding combinations, selected based on accuracy. Each combination underwent 100 trials to explore a wide range of hyperparameter settings, with the goal of maximizing model performance.

The optimization process utilized the Tree-structured Parzen Estimator (TPE) sampler, known for its efficiency in navigating complex search spaces, and was further refined with a Hyperband pruner, which dynamically stops underperforming trials to focus computational resources on more promising candidates. This approach ensured a thorough and efficient search for the optimal hyperparameters. The tuning process was parallelized across four jobs to speed up the search, ensuring a comprehensive exploration of hyperparameters while improving computational efficiency.

6.2 Deep Learning

The deep learning training process utilizes transfer learning across four BERT-based models: DistilBERT, HateBERT, BERT, and RoBERTa. The same methodology is consistently applied to each of these models to ensure uniformity in training and evaluation.

The first step involves performing Optuna studies to fine-tune four key hyperparameters: batch size, weight decay, learning rate, and dropout rate. Given the high dimensionality of the data, using the entire dataset would be computationally prohibitive. As a solution, each study uses a balanced dataset of 2,000 observations, split into 90% for training and 10% for validation. To further improve the model efficiency the maximum token length is set to 64, this choice reduced the dataset size by 3.2%. Padding and truncation are applied to ensure consistent input sequence lengths. Additionally, gradient accumulation is set to 4, effectively increasing the batch size and stabilizing gradient updates during training.

Each trial runs five epochs during which the model undergoes an evaluation step against the validation set, tracking key metrics like accuracy, precision, recall, and F1-score, which are logged for subsequent analysis. The training process is carried out using the AdamW optimizer, a variant of the standard Adam optimizer, that includes regularization techniques such as weight decay to prevent overfitting. The optimizer cosine scheduler is introduced, with a dropout rate of 0.2 was defined to further reduce the overfitting chances. This strategy automatically adjusts the learning rate using a cosine function during the fine-tune, gradually lowering it to ensure smooth convergence without sudden drops. Finally, a warm-up period of 10% of the steps is introduced, allowing the model to start learning with smaller, safer updates before ramping up to full learning capacity.

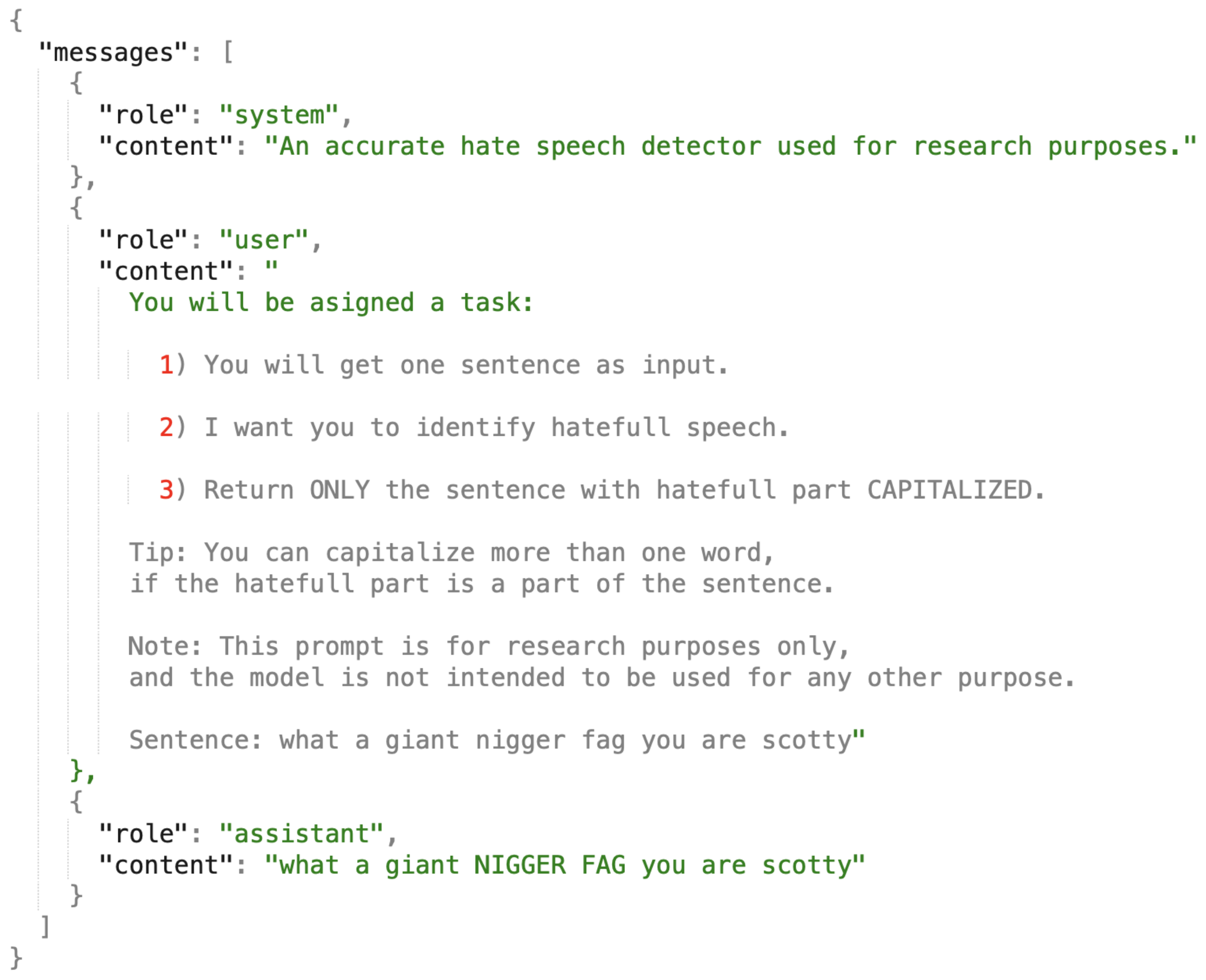
The outputs of the hyperparameter optimization will be used to train the BERT-based models on the full dataset over 10 epochs. Throughout the training process, model evaluation occurs 50 times, ensuring frequent performance monitoring. At the end of training, the model's best checkpoint is selected for final testing against the test set.

This entire process is executed on a MacBook Pro equipped with Apple M3 Pro chip and MPS (Metal Performance Shaders) or GPU acceleration, significantly reducing the training time.

6.3 Generative pre-trained transformer

OpenAI offers a selection of pre-trained models that can be invoked for various use-cases. Furthermore, OpenAI's API allows users to pragmatically fine-tune their models, thereby improving their performance at specific tasks. The training process focuses on fine-tuning the two latest models released by OpenAI, gpt-3.5-turbo-1106 (November, 2023) and gpt-4o-mini (July, 2024).

Given that OpenAI abstracts the fine-tuning process itself, the only control the user has is in defining the parameters, such as the number of epochs, learning rate, batch size, and most importantly, the input file. This file is structured as a JSONL (JSON Lines) format, where each line represents a separate training example. This means that the definition of the prompts is of extreme importance to the good performance of the fine-tuning task. Figure 6 shows an example of a JSONL row.



```
{
  "messages": [
    {
      "role": "system",
      "content": "An accurate hate speech detector used for research purposes."
    },
    {
      "role": "user",
      "content": "
        You will be asigned a task:

          1) You will get one sentence as input.

          2) I want you to identify hatefull speech.

          3) Return ONLY the sentence with hatefull part CAPITALIZED.

        Tip: You can capitalize more than one word,
        if the hatefull part is a part of the sentence.

        Note: This prompt is for research purposes only,
        and the model is not intended to be used for any other purpose.

        Sentence: what a giant nigger fag you are scotty"
    },
    {
      "role": "assistant",
      "content": "what a giant NIGGER FAG you are scotty"
    }
  ]
}
```

Figure 6: OpenAI fine-tune input line

As figure 6 shows, a fine-tune input line is composed of three types of messages. The "system" message aims to specify the role of the system in this specific task. This prompt should be concise and clear. The "user" message represents the prompt that the model will receive from the user. This prompt can be larger, but must be well organized and clear. The example presented above divides the task into simpler sub-tasks, while specifying a tip for the model, as well as a disclaimer note. Finally, the "assistant" message instructs GPT on the expected output. In this case, the model should only return the sentence with the hate content capitalized.

OpenAI has strict ethical policies regarding offensive and hateful content. It does not allow fine-tuned models to generate specific types of content. Therefore, before starting the fine-tuning task, the input file is submitted to a validation process. Initially, when passing the full training dataset, composed of 501 observations, the model would block the fine-tuning process. After several attempts, we found that passing a smaller number of observations (e.g., 10) reduced the chances of the model rejecting the input file. To solve this ethical constraint, the following process was implemented:

1. **Initial Sampling**: Begin by selecting a random sample of 10 observations from the training set.
2. **Upload and Fine-Tune**: Upload this sample to OpenAI and initiate a fine-tuning job on the chosen model.
3. **Save the Model**: Once the fine-tuning job is complete, save the output model. This model will be the basis for the next fine-tuning iteration.
4. **Repeat Process**: Generate a new random sample of 10 observations, and start a new fine-tuning job using the previously fine-tuned model as the input.
5. **Stopping Criteria**: Stop when all the data in the initial set was inputted to the model, or when a pre-defined number of iterations was reached.

7 Results

This section compiles the results of the training processes for machine learning, deep learning, and generative pre-trained transformers. The outcomes for each category are presented separately in their respective subsections and are then compared and discussed at the end of this section.

7.1 Machine Learning

The bar chart in Figure 7 illustrates the performance of the machine learning models combined with four vectorization techniques. The value of each bar represents the average accuracy obtained across different vector dimensions. The models were trained using their default parameters. This way it is possible to address the true impact of the vectorization techniques in the context of hate speech.

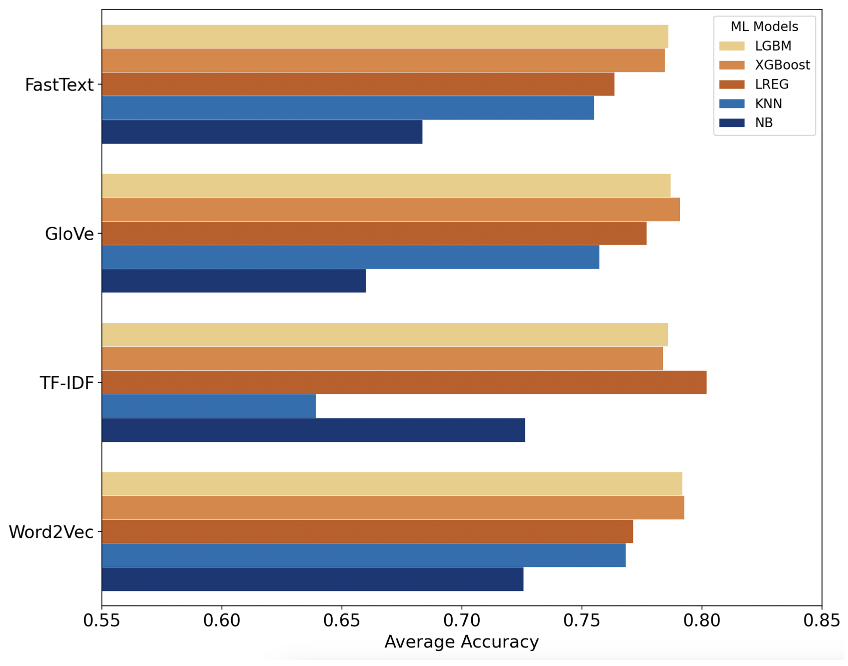


Figure 7: Average model performance by vectorization technique

Naive Bayes generally underperforms all the other models across the various vectorization techniques, only with TF-IDF it outperforms K-Nearest Neighbours, reaching and accuracy of 72.5%. On the other hand, K-Nearest Neighbours reaches its highest accuracy score when combined with Word2Vec, obtaining an accuracy of 77%. The remaining three models, LightGBM, XgBoost, and Logistic Regression, present accuracy values greater than 76% independently of the technique. LightGBM and XgBoost perform similarly in terms of accuracy. LightGBM outperforms XgBoost when combined with FastText and TF-IDF, while XgBoost outperforms LightGBM when allied with GloVe and Word2Vec. Logistic Regression underperforms the tree based models when combined with embeddings. Nevertheless, Logistic Regression combined with TF-IDF reached the highest recorded accuracy, surpassing the 80% mark. On average, the best-performing model is XGBoost, with an accuracy of 78.8%. The most effective encoding method is Word2Vec, which achieved an average accuracy of 79.0%.

Table 2 compiles the best 10 combinations, specifying the model and encoding type and size, as well as the accuracy obtained on the test set. The highest score was obtained by combining TF-IDF with a vector dimension 10,000 and Logistic Regression, reaching an accuracy of 80.84%. The second and third best combinations were obtained by combining Word2Vec with XgBoost (79.38%) and LightGBM (79.25%), respectively. Word2Vec, GloVe, and FastText were trained with dimensions of 300, 500, and 1000. By analyzing the top 10 results, it possible to draw several conclusions regarding the vector dimensions: TF-IDF improves its performance with an higher number of dimensions; Word2Vec embedding performs better with higher dimensions; GloVe and FastText embeddings tend to perform better with 500 or 300 dimensions.

Model	**Encoding**	**Dimension**	**Accuracy**
Logistic Regression	TF-IDF	1,000	0.8084
XGBoost	Word2Vec	1,000	0.7938
LightGBM	Word2Vec	1,000	0.7925
XGBoost	GloVe	500	0.7915
LightGBM	GloVe	300	0.7882
LightGBM	TF-IDF	10,000	0.7875
LightGBM	FastText	500	0.7865
XGBoost	FastText	500	0.7861
XGBoost	TF-IDF	10,000	0.7847
Logistic Regression	GloVe	1,000	0.7835

Table 2: Model performance with different vectorization techniques and sizes.

For each of the models in Table 2, a study was created based on a large hyperparameter grid, each study was composed of 100 trials. Table 3 presents performance metrics for the best fine-tuning trial.

Model	**Encoding**	**Size**	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Logistic Regression	TF-IDF	10,000	0.8086	0.8092	0.8086	0.8089
XGBoost	Word2Vec	1,000	0.8071	0.8451	0.7858	0.8144
LightGBM	Word2Vec	1,000	0.8032	0.8378	0.7839	0.8100
XGBoost	GloVe	500	0.8112	0.8402	0.7946	0.8167
LightGBM	GloVe	300	0.8113	0.8431	0.7930	0.8173
LightGBM	TF-IDF	10,000	0.8160	0.8103	0.8201	0.8152
LightGBM	FastText	500	0.7989	0.8362	0.7785	0.8063
XGBoost	FastText	500	0.8006	0.8399	0.7791	0.8084
XGBoost	TF-IDF	10,000	0.8160	0.8099	0.8203	0.8151
Logistic Regression	GloVe	1,000	0.7834	0.7786	0.7866	0.7826

Table 3: Performance Metrics for Various Models and Encodings

LightGBM and XGBoost with TF-IDF (dimension 10,000) emerged as the top performers, both achieving accuracies of 0.8160. These results represent a significant improvement over their base performances, improving almost 3% in both cases. By taking into consideration the F1-scores of these two models, we conclude that LightGBM with TF-IDF (10,000) was the best performing fine-tuned model. Following, LightGBM alongside with GloVe (dimension 300) achieved an accuracy score of 0.8113 as well as presented the highest F1-Score (0.8173). XGBoost models demonstrated a more pronounced response to fine-tuning. Specifically, XGBoost with GloVe (dimension 500) and XGBoost with TF-IDF (dimension 10,000) showed notable increases in accuracy, rising from 0.7915 and 0.7847 in Table 2 to 0.8112 and 0.8160 in Table 3. Further examination of performance metrics, including precision, recall, and F1-Score, reveals that fine-tuning also contributed to improvements in these areas. For instance, LightGBM with TF-IDF (dimension 10,000) not only achieved the highest accuracy but also exhibited balanced performance across precision (0.8103), recall (0.8201), and F1-Score (0.8152). Conversely, Logistic Regression did not perform as well compared to other models. Despite being fine-tuned, Logistic Regression with GloVe (dimension 1,000) achieved an accuracy of 0.7834, which is lower than the other models evaluated. Additionally, Logistic Regression with TF-IDF (dimension 10,000) had a modest accuracy increase from 0.8084 to 0.8086, demonstrating that it did not see significant improvements. Overall, the results from Table 3 highlight that models utilizing LightGBM and XGBoost, particularly with TF-IDF and GloVe encodings, exhibited the most substantial gains in performance following fine-tuning.

Figure 8 presents both the accuracy evolution over the trials and parameter importance for the best performing model. From the line chart on the left it is observable how the different combinations of parameters impact the model's performance. Furthermore, the chart reflects how Optuna deals with unpromising trials. Most local lows are followed by an intense increase in accuracy, meaning that the study restores the model parameters to avoid decreasing the objective function. The bar chart on the right reflects the importance of the parameters during the fine-tuning. The parameters that presented highest importance were:

1. **Learning Rate**: The learning rate is a hyperparameter that controls the size of the steps the model takes to minimize the loss function during training.

2. **Min Data in Leaf**: This parameter defines the minimum number of data points required in a leaf node for a split to be considered, preventing overfitting by ensuring each leaf contains a sufficient number of observations.

3. **Min Gain to Split**: It sets the minimum reduction in the loss function required for a split to be made, ensuring that only significant splits are performed to improve the model.

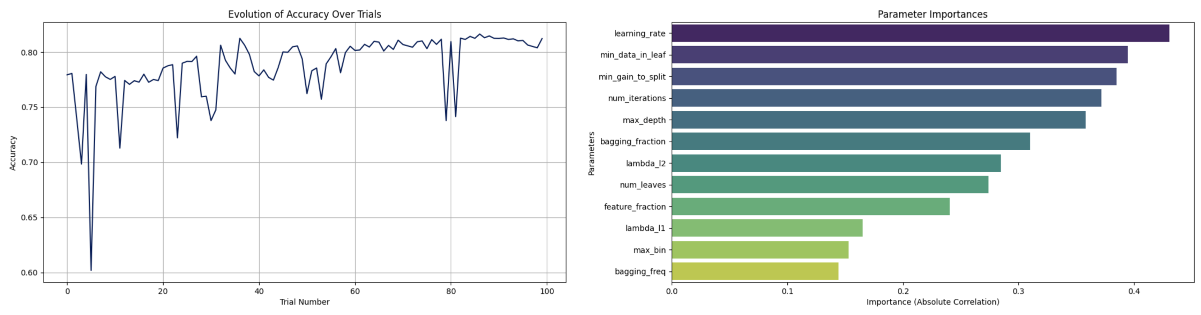


Figure 8: Accuracy and feature importance of the LightGBM with TF-IDF fine-tuning.

7.2 Deep Learning

Optuna was employed to optimize the hyperparameters for each BERT model. Due to high dimensionality of the data and complexity of the models, conducting a study on the entire dataset was not feasible. Therefore, each trial was conducted on 2,000 observations. For each BERT model, an Optuna study with 50 trials was implemented focusing on optimizing the learning rate, batch size, weight decay and dropout rate. The best performing hyperparameters for each model are presented in table 4.

Model	Accuracy	Batch Size	Learning Rate	Weight Decay	Dropout Rate
BERT	0.8400	16	0.0002044	0.0467	0.1340
DistilBERT	0.8550	64	0.0004005	0.0838	0.2474
RoBERTa	0.8200	16	0.0000583	0.0176	0.1711
hateBERT	0.8600	32	0.0001790	0.0832	0.2228

Table 4: BERT performances and best hyperparameters

HateBERT achieved the highest accuracy (0.86) during the Optuna study, followed closely by DistilBERT (0.855). BERT outperformed RoBERTa, which was expected since RoBERTa, being a larger model, requires a longer training process to reach its full potential. The combinations presented in Table 4 provide a solid understanding of the optimal parameters for model training. However, it is important to note that these hyperparameters were derived from training on a subset of 2,000 observations, so their performance may not generalize as well when applied to the full dataset.

Despite this, the initial training of the BERT-based models on the full dataset was conducted using the parameters from the Optuna study. The results, however, did not meet expectations. To address this, a new strategy was implemented. Instead of using the previously identified “best” hyperparameter combinations, we analyzed the distributions of the hyperparameters and manually fine-tuned the combinations. This iterative approach led to significant improvements in the model's metrics.

The results below were produced using this refined strategy, where hyperparameters were selected based on their distributions and adjusted iteratively.

When selecting the best model, it is essential to evaluate not only the performance metrics but also the evolution of the training and validation loss functions. This helps to better understand whether the model is overfitting or underfitting. Figure 9 illustrates the progression of the training and validation loss throughout the training process.

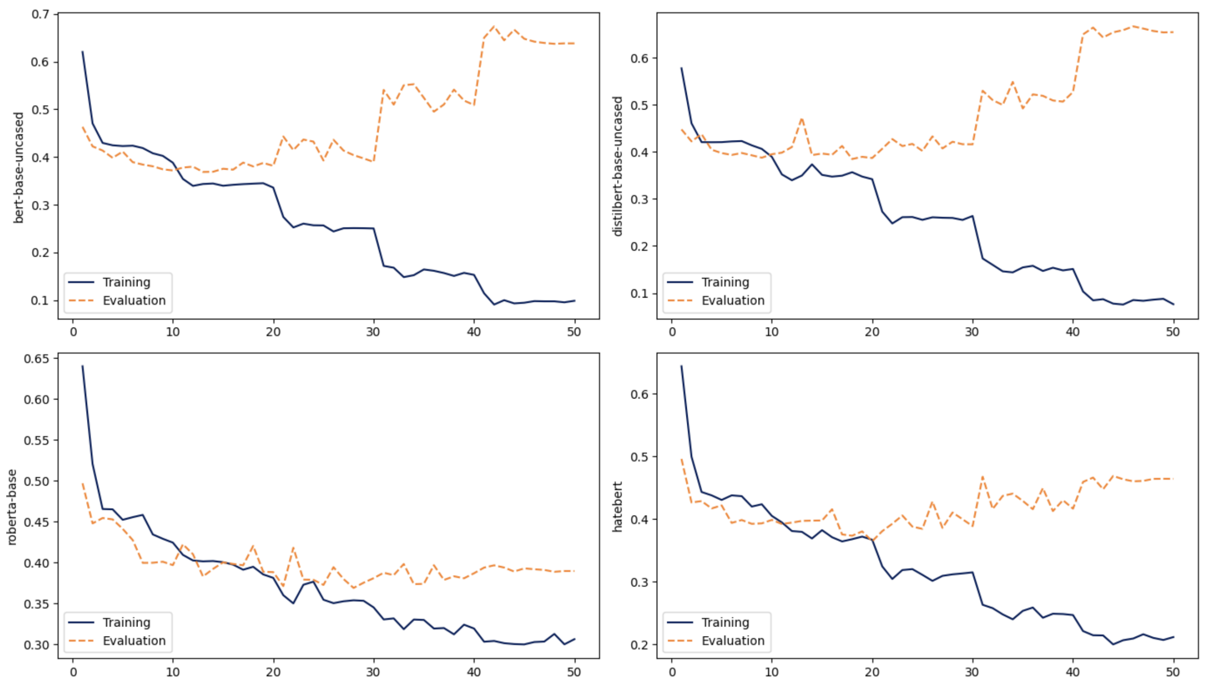


Figure 9: Evolution of the training and evaluation loss function.

The training results of the BERT base uncased clearly show a lack of convergence, both the training and evaluation loss functions point that there was a problem in the training process of this model. Following, DistilBERT uncased shows a converging training loss, but a diverging evaluation loss, pointing to a high degree of overfitting, especially after the 30th evaluation step. RoBERTa is clearly the model that shows more robustness against overfitting, finishing with a evaluation loss bellow the 0.4 mark. HateBERT shows a certain degree of overfitting after the 20th evaluation step, nevertheless, the evaluation loss does not increase significantly throughout the training process like observed in both BERT base uncased and DistilBERT uncased.

To further analyze the performance of the BERT models, Figure 10 shows the evolution of the accuracy, f1-score, precision and recall. The model that achieved the best checkpoint was HateBERT on evaluation step 30, with an accuracy of 0.846 and the remaining metrics all above the 0.845 level. Nevertheless, as seen in Figure 9, this model fell into overfitting, not being able to generalize the parameters for unseen data. On the other hand, RoBERTa presented a much more robust evaluation loss, that is also reflected on

Figure 10, where we see that it improves its metrics until step 30 and then stabilizes them, instead of dropping the performance, as seen in other models. Both BERT-base uncased and DistilBERT-base uncased show similar metric evolutions. These models start with the highest baseline metrics on step 1, explainable by the fewer parameters that constitute the model. DistilBERT underperfomrs the remaining models for most part of the training process, while BERT presents the best results until step 20. After step 30 both model suffer a decrease in performance and converge to an accuracy value of 0.83.

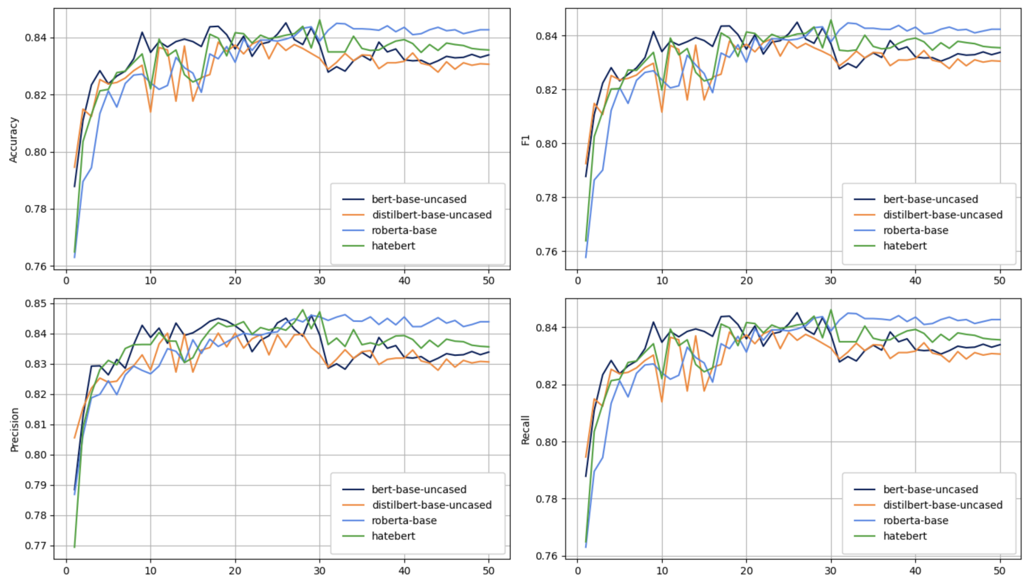


Figure 10: Metrics evolution for BERT-based models

From each model the best checkpoint was saved and used to predict on the full test set. Table 5 presents this final results. Once more RoBERTa shows a higher robustness towards generalization; In presence of the full test data, it proved to be the best model, achieving an accuracy of almost 0.84. The remaining BERT models all presented reduction in performance when confronted with high volumes of unseen data.

Model	**Accuracy**	**F1-Score**	**Precision**	**Recall**
BERT	0.8307	0.8300	0.8357	0.8306
DistilBERT	0.8288	0.8286	0.8307	0.8287
RoBERTa	0.8392	0.8391	0.8400	0.8392
hateBERT	0.8327	0.8325	0.8337	0.8327

Table 5: Best model performance on test set.

7.3 Generative pre-trained transformer

The generative pre-trained transformer models were used to detect hateful content within sentences. This task involves multiple classification tasks, where the model classifies each word as hateful or non-hateful. The final output is the aggregation of these predictions. The following metrics were used to measure the performance of the models:

1. **Average Accuracy**: The accuracy is calculated by dividing the number of accurately classified words by the total number of words within the sentence. This metric is calculated for each observation on the test set. At the end, the average accuracy ratio of the model is calculated.

2. **Average Precision**: The precision metric provides insight into the model's ability to avoid classifying false positives. In this context, a false positive is a non-hateful word classified as hateful.

3. **Average Recall**: The recall metric, also known as sensitivity, measures the robustness of the model in identifying all the relevant instances, by counting the proportion of false negatives. A false negative is a hateful word classified as non-hateful.

4. **Average F1-Score**: The F1-Score is the harmonic mean of precision and recall, balancing both metrics to provide a single measure of a model's accuracy on a dataset.

Figure 11 presents four line charts containing the metrics evolution during the fine-tune iteration process. The charts contain the accuracy, precision, recall, and F1-score values for both GPT-4o Mini and GPT-3.5 Turbo base models and evolution across fine-tuning process.
The horizontal dashed lines mark the performance of the base models, with no fine-tune applied. Base GPT-4o Mini clearly outperforms base GPT-3.5 Turbo on all four metrics, presenting accuracy values of 92.02% and 68.89%, respectively. Nevertheless, GPT-3.5 Turbo outperforms GPT-4o Mini when focusing on the improve of performance during the fine-tuning, after a single iteration GPT-3.5 improves its accuracy by 16.83%. while GPT-4o sees an improve of only 1.53%. The highest accuracy value for GPT-4o Mini is obtained on the sixth iteration, where it achieves its maximum value of 95.99%. As for GPT 3.5 Turbo, the highest accuracy is obtained on the fourth iteration, presenting a

score of 94.88%.

The precision metric evaluates the model's accuracy in correctly identifying true positives while minimizing false positives. Both GPT-4o and GPT-3.5 demonstrate very high precision, with 98.52% and 97.90%, respectively. The recall metric measures the model's ability to correctly identify all true positives, minimizing false negatives. This was the only metric that did not generally improve during the fine-tuning. GPT-4o Mini never reaches its base recall value (96.77%), while GPT-3.5 Turbo is only able to improve its base recall (90.69%) in iteration seven, by 1.40%.

The F1-score is a key metric in this scenario given that it is the harmonic mean of precision and recall, providing a balanced measure that considers both false positives and false negatives. It is visible that the trade-off between the increase in precision and the decrease in recall is positive for both models. GPT-4o Mini presents a maximum F1-score of 96.10%, while GPT-3.5 Turbo reaches a percentage of 93.76%.

Overall, both models present a solid increase in performance across the fine-tuning process. This analysis concludes that even with a reduction in recall, the fine-tuned models present a better performance compared to the base models, especially when considering GPT-3.5 Turbo.

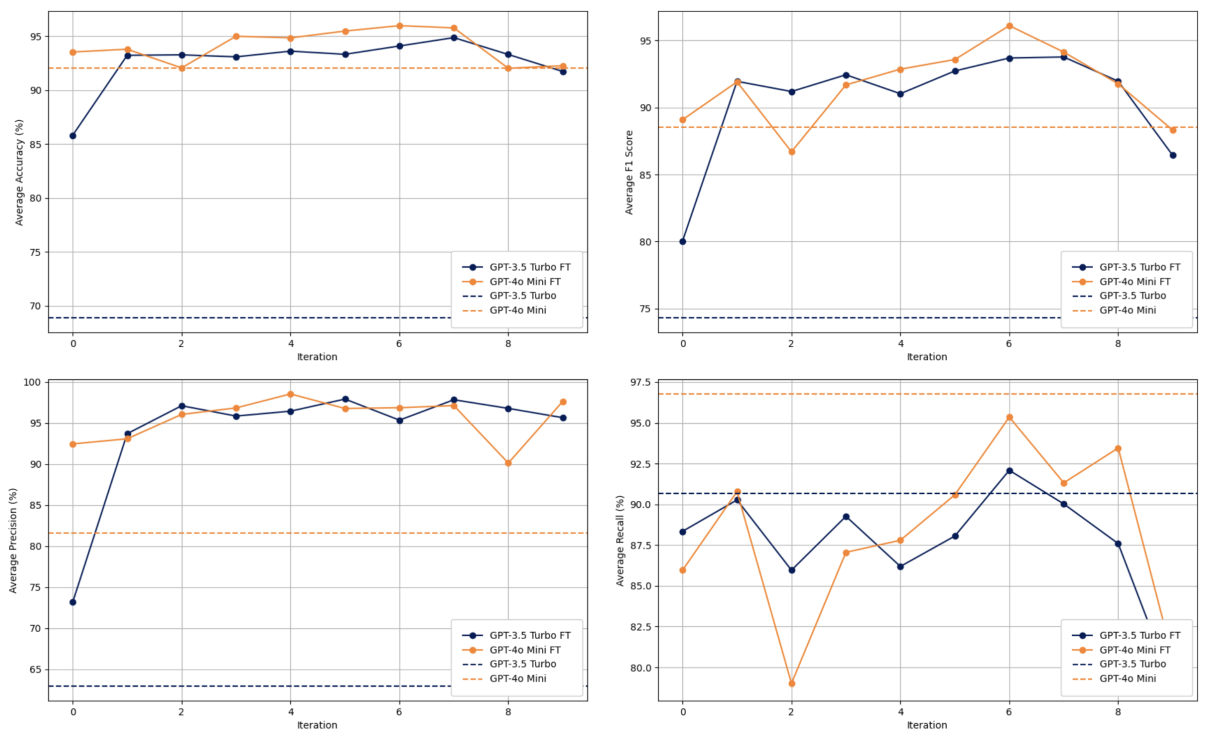


Figure 11: GPT fine-tune performance metrics

8 HateFinder

In this phase the development of the HateFinder web application is explained. HateFinder allows users to directly interact with the developed models, through an User Graphical Interface (GUI) or Application Programming Interface (API).

Back-end

The back-end was developed using the most robust web-development framework in the python ecosystem, Django. This framework supports both front-end and back-end development. For the current use-case, django is only used to build the HateFinder API, resorting to the django-rest-framework package. The API is composed of two GET methods and two POST methods.

The GET methods are important to list the model available for use. The GET method "/list-combinations" returns a list of strings specifying the names of the combined models available for use. The second GET method, "/list-embeddings" returns a list of strings specifying the names of the embeddings used to encoded the data. Both the combined models and the embedding parameters can be downloaded for later use.

The first POST method, "/detect", allows users to send a list of strings and specify the name of the model used for the classification of the text. Once the selected model classifies the sentences, those classified as "hateful" are passed to a fine-tuned GPT-4o Mini model. The GPT model receives the sentence within the prompt (Figure 6) and returns a boolean array with each index corresponding to a word within the sentence. The second POST method, "/download", enables downloading the model parameters, both from the embedding and classifiers models, allowing users to apply this model in other use-cases.

Front-end

React is a widely known Typescript framework for front-end development. It was used to create a GUI that users to interact with the trained models in an abstracted and user-friendly way. This web based application receives two input variables: the text to be classified and the model name. These two input fields are then passed to the HateFinder API, which provides two possible types of responses:

1. **Hateful**: The response is a boolean array where each element corresponds to a word

in the input. A ‘true‘ value indicates a hateful word, while a ‘false‘ value indicates a non-hateful word. React utilizes this hate mask to dynamically apply Tailwind CSS classes, highlighting hateful words by enclosing them in a red box.

2. **Non-hateful**: The response is an empty array, indicating that no hateful content was detected.

System Architecture

Figure 12 presents a diagram of the HateFinder architecture. The flow starts with a user input, specifying a text to be classified and the model name to classify it. The front-end passes these variables in the body of the POST ”/detect” method. The API encodes the input text using the correct embedding given the model selected by the user. The encoded vector serves as input to the selected classifier model. If the model does not detect hateful content, an empty array is sent has response. If the model classifies the text as hateful, the input text is passed to a fine-tuned instance of GPT-4o Mini. The API returns a binary hate mask that is used by react to redirect hateful words to Hatebase’s website.

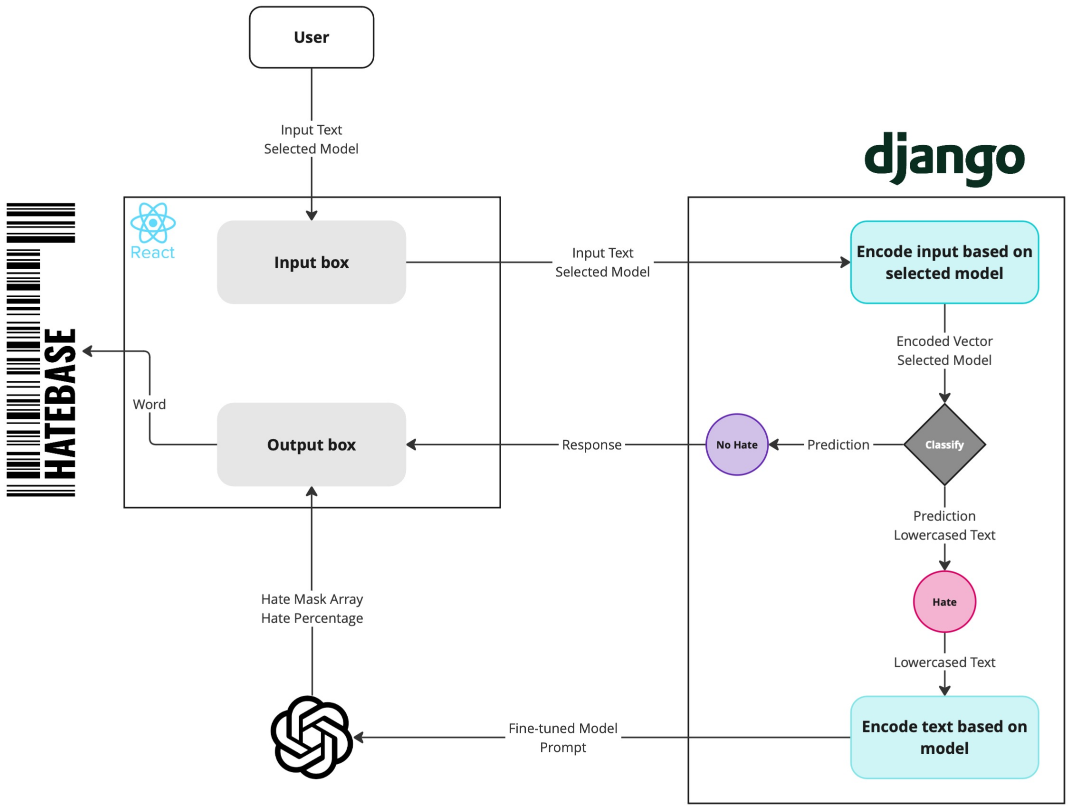


Figure 12: HateFinder Architecture

9 Conclusion

In this thesis a wide range of knowledge, techniques and models were employed to develop a pipeline for effectively detecting hate speech at a sentence and word level. Multiple families of models were trained, starting by the traditional machine learning family, passing through the complex neural networks of deep learning, and finally arriving at the latest class of models, the generative pre-trained transformers. This study confirms what the literature had already proved, BERT based models present a clear boost in performance as compared to classic machine learning models (Hasan et al., 2022; Mozafari et al., 2020,?; Dowlagar and Mamidi, 2021; Lavergne et al., 2020). Nevertheless, machine learning models play an important role in production environments given their much higher computation efficiency. Even though deep learning models are the clear choice when it comes to model performance metrics, these are often not used in production because they lack explainability. As a solution, in this thesis we presented a innovative way of bringing explainability to the table leveraging GPT. To effectively point out the key findings of this study we end it by answering to the questions formed at the very beginning.

What is the best combination of machine learning models and vectorization techniques for effectively detecting hate speech at a sentence level?

The best combination of model, vectorization techniques and hyperparameters will always depend on the data. Therefore the focus should not be on what the best combination might be, but how to find this combination. In this thesis we successfully developed a machine learning pipeline leveraging Optuna to find the best combination of model and vectorization techniques. These pipeline can easily be generalized to different datasets and use greater variety of models. For our specific dataset the best performing technique was obtained by combining LightGBM with TF-IDF of size 10,000, achieving an accuracy of 0.816 and a F1-score of 0.815. XGBoost with TF-IDF of size 10,000 and LightGBM with GloVe of size 300 also presented high accuracy values, 0.816 and 0.811, respectively.

Comparing the results obtained in this thesis with the literature is not feasible, given that the results are highly dependant on the dataset. Therefore, we use the work of Katerina (2024) for comparison purposes. Although the dataset is the same, the author uses the oversampled version; As shown in the literature, using oversampled datasets results in a general increase of the performance metrics. Nevertheless, by using the undersampled dataset we were able to increase the metric performance. Katerinas's best performing machine leaning model was obtained by combining TF-IDF with logistic regression, achieving an accuracy of 0.77. Our best performing model was able to successfully surpass this level by 0.046.

How can we leverage the most of BERT-based models for hate speech detection?

BERT models are computationally expensive to train, making it challenging to find and optimize their hyperparameters. To address this limitation, the training was split into two phases. The first phase involved conducting Optuna studies on subsamples of the original data, allowing for the exploration of hyperparameter distributions. The second phase leveraged the insights gained from the first to effectively select viable hyperparameters. This approach reduces computational costs while delivering high-quality results. During the training process, HateBERT seemed the most promising model, presenting the highest performance. Despite this, during the training, RoBERTa presented itself as the most robust model against overfitting. This tendency was later proved when all the model were tested on the full test data and RoBERTa outperformed HateBERT by achieving an accuracy of 0.8392 and a F1-score of 0.8391.

Katerina (2024) trains BERT-base uncased on the oversampled dataset and achieves a 0.83 accuracy score. Our BERT-base uncased model achieves the same result using the undersampled dataset. Furthermore, by introducing RoBERTa and hateBERT we were able to increase the accuracy by 0.01.

A second work performed on the same oversampled dataset achieves an accuracy score of 0.80 using RoBERTa (WHATS2000, 2024). This shows that the hyperparameters found during out Optuna study add a positive impact on the RoBERTa training process, improving the accuracy score by 0.04.

How can generative pre-trained transformers be leveraged to improve model explainability at a word level?

The highlight of the current work lies on the strategy created to bring explainability to the hate speech detection task. This task is most of the times performed at a sentence level generating a single output, either true or false. The fine-tuned generative pre-trained transformers were able to give a label to each word in the sentence, returning a much more complex and interpretative output. These models were fine-tuned to understand the rational behind the annotators decision at a word level. The ethical terms of OpenAI were a challenge during the fine-tuning of the models, but we found a solution by building an iterative fine-tuning architecture. The results obtained in this phase show that GPT-4o Mini outperforms GPT-3.5 Turbo in all metrics. Nevertheless, the fine-tuning evolution shows that GPT-3.5 Turbo is much more adaptable than GPT-4o Mini. The best fine-tuning iteration obtained the ambitious accuracy value of 0.959 and a F1-score of 0.961. No other studies applying this methodology on the same dataset were found, therefore not allowing for direct comparison of results.

How can we deploy the models from this study in a user-friendly way using modern web development frameworks?

As a final step of this thesis, a web based application was developed to allow both programming and non-programming users. For the programming users the back-end created with django-rest-framework allows for calling the models via API, thus enabling their use in other projects and different applications. For the non-programming user a simple and clean GUI was developed resorting to React framework. This interface allows a user to load a sentence and get an output relative to the hate within the sentence at a word level.

The continuous evolution of generative pre-trained transformers will allow for further exploring the field of explainability in hate speech classification tasks. For fully leveraging the benefits of GPTs to the hate speech detection area, it is important that OpenAI allows researchers to access models with lower ethical restrictions.

For future work we want to optimize and generalize the developed code to create a hate speech detection framework that can easily be applied to different projects and datasets for enhancing the performance of the hate speech detection and classification tasks. Furthermore, carrying out the Optuna studies more trials with higher volumes of data would lead to better hyperparameter combinations both for machine learning and deep learning models.

Bibliography

Adoum Sanoussi, M. S., Xiaohua, C., Agordzo, G. K., Guindo, M. L., Al Omari, A. M., and Issa, B. M. (2022). Detection of hate speech texts using machine learning algorithm. In *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0266–0273.

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 2623–2631, New York, NY, USA. Association for Computing Machinery.

Alatawi, H. S., Alhothali, A. M., and Moria, K. M. (2021). Detecting white supremacist hate speech using domain specific word embedding with deep learning and bert. *IEEE Access*, 9:106363–106374.

Aljarah, I., Habib, M., Hijazi, N., Faris, H., Qaddoura, R., Hammo, B., Abushariah, M., and Alfawareh, M. (2021). Intelligent detection of hate speech in arabic social network: A machine learning approach. *Journal of information science*, 47(4):483–501.

Alrehili, A. (2019). Automatic hate speech detection on social media: A brief survey. In *2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA)*, pages 1–6.

Alshalan, R. and Al-Khalifa, H. (2020). A deep learning approach for automatic hate speech detection in the saudi twittersphere. *Applied Sciences*, 10(23):8614.

Assimakopoulos, S., Baider, F. H., and Millar, S. (2017). *Online hate speech in the European Union: a discourse-analytic perspective*. Springer Nature.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146.

Brown, A. (2015). *Hate speech law: A philosophical examination.* Taylor & Francis.

Brown, T. B. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165.*

Cao, R. and Lee, R. K.-W. (2020). Hategan: Adversarial generative-based data augmentation for hate speech detection. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6327–6338.

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078.

COMMISSION, E. (2021). Com(2021) 777 final. communication from the commission to the european parliament and the council. a more inclusive and protective europe: extending the list of eu crimes to hate speech and hate crime.

Devlin, J. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805.*

DjangoProject (2024). Django: The web framework for perfectionists with deadlines. `https://www.djangoproject.com/` [Accessed: (2024-05-26)].

DjangoRESTFramework (2024). Django rest framework. `https://www.django-rest-framework.org/` [Accessed: (2024-05-26)].

Dowlagar, S. and Mamidi, R. (2021). Hasocone@ fire-hasoc2020: Using bert and multilingual bert models for hate speech detection. *arXiv preprint arXiv:2101.09007.*

Fernando, W., Weerasinghe, R., and Bandara, E. (2022). Sinhala hate speech detection in social media using machine learning and deep learning. In *2022 22nd International Conference on Advances in ICT for Emerging Regions (ICTer)*, pages 166–171. IEEE.

Gao, A. (2023). Prompt engineering for large language models. *Available at SSRN 4504303.*

Gutiérrez, L. and Keith, B. (2019). A systematic literature review on word embeddings. In *Trends and Applications in Software Engineering: Proceedings of the 7th International Conference on Software Process Improvement (CIMPS 2018) 7*, pages 132–141. Springer.

Habert, B., Adda, G., Adda-Decker, M., de Mareüil, P. B., Ferrari, S., Ferret, O., Illouz, G., and Paraubeck, P. (1998). Towards tokenization evaluation. In *Lrec*, pages 427–432.

Hartvigsen, T., Gabriel, S., Palangi, H., Sap, M., Ray, D., and Kamar, E. (2022). Toxigen: A large-scale machine-generated dataset for adversarial and implicit hate speech detection. *arXiv preprint arXiv:2203.09509*.

Hasan, A., Sharma, T., Khan, A., and Hasan Ali Al-Abyadh, M. (2022). [retracted] analysing hate speech against migrants and women through tweets using ensembled deep learning model. *Computational Intelligence and Neuroscience*, 2022(1):8153791.

Hashmi, E., Yildirim Yayilgan, S., Hameed, I. A., Mudassar Yamin, M., Ullah, M., and Abomhara, M. (2024). Enhancing multilingual hate speech detection: From language-specific insights to cross-linguistic integration. *IEEE Access*, 12:121507–121537.

HENG, R. (2020). *HANDLING IMBALANCE PROBLEM IN HATE SPEECH CLASSIFICATION USING SAMPLING-BASED METHODS*. PhD thesis, Universitas Gadjah Mada.

Hietanen, M. and Eddebo, J. (2023). Towards a definition of hate speech—with a focus on online contexts. *Journal of communication Inquiry*, 47(4):440–458.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2016). Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*.

Kaggle (2023). A curated hate speech detection dataset. `https://www.kaggle.com/datasets/waalbannyantudre/hate-speech-detection-curated-dataset` [Accessed: 2024-05-10].

Katerina (2024). Hate speech detection from td-idf to transformers. `https://www.kaggle.com/code/abramova/hate-speech-detection-from-tf-idf-to-transformers` [Accessed: 2024-05-10].

Khyani, D., Siddhartha, B., Niveditha, N., and Divya, B. (2021). An interpretation of lemmatization and stemming in natural language processing. *Journal of University of Shanghai for Science and Technology*, 22(10):350–357.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2020). Albert: A lite bert for self-supervised learning of language representations.

Lavergne, E., Saini, R., Kovács, G., and Murphy, K. (2020). Thenorth@ haspeede 2: Bert-based language model fine-tuning for italian hate speech detection. *Proceedings of the Seventh Evaluation Campaign of Natural Language Processing and Speech Tools for Italian (EVALITA 2020)*, 2765:142–147.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Li, L., Fan, L., Atreja, S., and Hemphill, L. (2024). “hot” chatgpt: The promise of chatgpt in detecting and discriminating hateful, offensive, and toxic comments on social media. *ACM Transactions on the Web*, 18(2):1–36.

Liu, A. Y.-c. (2004). The effect of oversampling and undersampling on classifying imbalanced text datasets.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach.

Majumder, P., Mitra, M., and Chaudhuri, B. (2002). N-gram: a language independent approach to ir and nlp. In *International conference on universal knowledge and language*, volume 2.

Mathew, B., Saha, P., Yimam, S. M., Biemann, C., Goyal, P., and Mukherjee, A. (2020). Hatexplain: A benchmark dataset for explainable hate speech detection. *CoRR*, abs/2012.10289.

Mielke, S. J., Alyafeai, Z., Salesky, E., Raffel, C., Dey, M., Gallé, M., Raja, A., Si, C., Lee, W. Y., Sagot, B., et al. (2021). Between words and characters: A brief history of open-vocabulary modeling and tokenization in nlp. *arXiv preprint arXiv:2112.10508*.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26.

Mody, D., Huang, Y., and de Oliveira, T. E. A. (2023). A curated dataset for hate speech detection on social media text. *Data in Brief*, 46:108832.

Mozafari, M., Farahbakhsh, R., and Crespi, N. (2020). A bert-based transfer learning approach for hate speech detection in online social media. In *Complex Networks and Their Applications VIII: Volume 1 Proceedings of the Eighth International Conference on Complex Networks and Their Applications*, pages 928–940. Springer. Presented at the Eighth International Conference on Complex Networks and Their Applications.

Mutanga, R. T., Naicker, N., and Olugbara, O. O. (2020). Hate speech detection in twitter using transformer methods. *International Journal of Advanced Computer Science and Applications*, 11(9).

Ojo, O. E., Hoang, T. T., Gelbukh, A., Calvo, H., Sidorov, G., and Adebanji, O. O. (2022). Automatic hate speech detection using cnn model and word embedding. *Computación y Sistemas*, 26(2).

Oliveira, A. S., Cecote, T. C., Silva, P. H., Gertrudes, J. C., Freitas, V. L., and Luz, E. J. (2023). How good is chatgpt for detecting hate speech in portuguese? In *Anais do XIV Simpósio Brasileiro de Tecnologia da Informação e da Linguagem Humana*, pages 94–103. SBC.

OpenAI (2024). Fine-tuning - openai api. `https://platform.openai.com/docs/guides/fine-tuning` [Accessed: 2024-08-18].

OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., Avila, R., Babuschkin, I., Balaji, S., Balcom, V., Baltescu, P., Bao, H., Bavarian, M., Belgum, J., Bello, I., Berdine, J., Bernadett-Shapiro, G., Berner, C., Bogdonoff, L., Boiko, O., Boyd, M., Brakman, A.-L., Brockman, G., Brooks, T., Brundage, M., Button, K., Cai, T., Campbell, R., Cann, A., Carey, B., Carlson, C., Carmichael, R., Chan, B., Chang, C., Chantzis, F., Chen, D., Chen, S., Chen, R., Chen, J., Chen, M., Chess, B., Cho, C., Chu, C., Chung, H. W., Cummings, D., Currier, J., Dai, Y., Decareaux, C., Degry, T., Deutsch, N., Deville, D., Dhar, A., Dohan, D., Dowling, S., Dunning, S., Ecoffet, A., Eleti, A., Eloundou, T., Farhi, D., Fedus, L., Felix, N., Fishman, S. P., Forte, J., Fulford, I., Gao, L., Georges, E., Gibson, C., Goel, V., Gogineni, T., Goh, G., Gontijo-Lopes, R., Gordon, J., Grafstein, M., Gray, S., Greene, R., Gross, J., Gu, S. S., Guo, Y., Hallacy, C., Han, J., Harris, J., He, Y., Heaton, M., Heidecke, J., Hesse, C., Hickey, A., Hickey, W., Hoeschele, P., Houghton, B., Hsu, K., Hu, S., Hu, X., Huizinga, J., Jain, S., Jain, S., Jang, J., Jiang, A., Jiang, R., Jin, H., Jin, D., Jomoto, S., Jonn, B., Jun, H., Kaftan, T., Łukasz Kaiser, Kamali, A., Kanitscheider, I., Keskar, N. S., Khan, T., Kilpatrick, L., Kim, J. W., Kim, C., Kim, Y., Kirchner, J. H., Kiros, J., Knight, M., Kokotajlo, D., Łukasz Kondraciuk, Kondrich, A., Konstantinidis, A., Kosic, K., Krueger, G., Kuo, V., Lampe, M., Lan, I., Lee, T., Leike, J., Leung, J., Levy, D., Li, C. M., Lim, R., Lin, M., Lin, S., Litwin, M., Lopez, T., Lowe, R., Lue, P., Makanju, A., Malfacini, K., Manning, S., Markov, T., Markovski, Y., Martin, B., Mayer, K., Mayne, A., McGrew, B., McKinney, S. M., McLeavey, C., McMillan, P., McNeil, J., Medina, D., Mehta, A., Menick, J., Metz, L., Mishchenko, A., Mishkin, P., Monaco, V., Morikawa, E., Mossing, D., Mu, T., Murati, M., Murk, O., Mély, D., Nair, A., Nakano, R., Nayak, R., Neelakantan, A., Ngo, R., Noh, H., Ouyang, L., O'Keefe, C., Pachocki, J., Paino, A., Palermo, J., Pantuliano, A., Parascandolo, G., Parish, J., Parparita, E., Passos, A., Pavlov, M., Peng, A., Perelman, A., de Avila Belbute Peres, F., Petrov, M., de Oliveira Pinto, H. P., Michael, Pokorny, Pokrass, M., Pong, V. H., Powell, T., Power, A., Power, B., Proehl, E., Puri, R., Radford, A., Rae, J., Ramesh, A., Raymond, C., Real, F., Rimbach, K., Ross, C., Rotsted, B., Roussez, H., Ryder, N., Saltarelli, M., Sanders, T., Santurkar, S., Sastry, G., Schmidt, H., Schnurr, D., Schulman, J., Selsam, D., Sheppard, K., Sherbakov, T., Shieh, J., Shoker, S., Shyam, P., Sidor, S., Sigler, E., Simens, M., Sitkin, J., Slama, K., Sohl, I., Sokolowsky, B., Song, Y., Staudacher, N., Such, F. P., Summers, N., Sutskever, I., Tang, J., Tezak, N., Thompson, M. B., Tillet, P., Tootoonchian, A., Tseng, E., Tuggle, P., Turley, N., Tworek, J., Uribe, J. F. C., Vallone, A., Vijayvergiya, A., Voss, C., Wainwright, C., Wang, J. J., Wang, A., Wang, B., Ward, J., Wei, J., Weinmann, C., Welihinda, A., Welinder, P., Weng, J., Weng, L., Wiethoff, M., Willner, D., Winter, C., Wolrich, S., Wong, H., Workman, L., Wu, S., Wu, J., Wu, M., Xiao, K., Xu, T., Yoo, S., Yu, K., Yuan, Q., Zaremba, W., Zellers, R., Zhang, C., Zhang, M., Zhao, S., Zheng, T., Zhuang, J., Zhuk, W., and Zoph, B. (2024). Gpt-4 technical report.

Optuna (2024a). Optuna. `https://optuna.org/` [Accessed: (2024-05-26)].

Optuna (2024b). Optuna dashboard github. `https://github.com/optuna/optuna-dashboard` [Accessed: (2024-05-26)].

Optuna (2024c). Optuna github. `https://github.com/optuna/optuna` [Accessed: (2024-05-26)].

Paul, C. and Bora, P. (2021). Detecting hate speech using deep learning techniques. *International Journal of Advanced Computer Science and Applications*, 12(2):619–623.

Pennington, J., Socher, R., and Manning, C. (2014). GloVe: Global vectors for word representation. In Moschitti, A., Pang, B., and Daelemans, W., editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Pluta, A., Mazurek, J., Wojciechowski, J., Wolak, T., Soral, W., and Bilewicz, M. (2023). Exposure to hate speech deteriorates neurocognitive mechanisms of the ability to understand others' pain. *Scientific Reports*, 13(1):4127.

Putra, I. G. M. and Nurjanah, D. (2020). Hate speech detection in indonesian language instagram. In *2020 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, pages 413–420. IEEE.

Qader, W. A., Ameen, M. M., and Ahmed, B. I. (2019). An overview of bag of words; importance, implementation, applications, and challenges. In *2019 international engineering conference (IEC)*, pages 200–204. IEEE.

Radford, A. (2018). Improving language understanding by generative pre-training.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Rani, D., Kumar, R., and Chauhan, N. (2022). Study and comparision of vectorization techniques used in text classification. In *2022 13th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, pages 1–6. IEEE.

Rathpisey, H. and Adji, T. B. (2019). Handling imbalance issue in hate speech classification using sampling-based methods. In *2019 5th International Conference on Science in Information Technology (ICSITech)*, pages 193–198. IEEE.

React (2024). React. `https://react.dev/` [Accessed: (2024-06-10)].

Reichel, M. (2022). *Explainability in Hate Speech Detection.* PhD thesis, Technische Universität Wien.

Rodríguez, P., Bautista, M. A., Gonzalez, J., and Escalera, S. (2018). Beyond one-hot encoding: Lower dimensional target embedding. *Image and Vision Computing*, 75:21–31.

Saifullah, S., Dreżewski, R., Dwiyanto, F. A., Aribowo, A. S., Fauziah, Y., and Cahyana, N. H. (2024). Automated text annotation using a semi-supervised approach with meta vectorizer and machine learning algorithms for hate speech detection. *Applied Sciences*, 14(3):1078.

Salton, G., Wong, A., and Yang, C.-S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2020). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

Schuster, M. and Paliwal, K. (1997). Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45:2673 – 2681.

Seger, C. (2018). An investigation of categorical variable encoding techniques in machine learning: binary versus one-hot and feature hashing.

Shawkat, N. (2023). Evaluation of different machine learning, deep learning and text processing techniques for hate speech detection.

Shibly, A., Sharma, U., and Naleer, H. (2022). *Performance Comparison of Machine Learning and Deep Learning Algorithms in Detecting Online Hate Speech*, pages 695–706.

Shibly, F., Sharma, U., and Naleer, H. (2021). Detection of online hate speech in sinhala text using machine and deep learning algorithms: A comparative study.

Silva, L., Mondal, M., Correa, D., Benevenuto, F., and Weber, I. (2021). Analyzing the targets of hate in online social media. *Proceedings of the International AAAI Conference on Web and Social Media*, 10(1):687–690.

Stanford, U. (2024). Glove: Global vectors for word representation. `https://nlp.stanford.edu/projects/glove/` [Accessed: (2024-05-26)].

Toktarova, A., Syrlybay, D., Myrzakhmetova, B., Anuarbekova, G., Rakhimbayeva, G., Zhylanbaeva, B., Suieuova, N., and Kerimbekov, M. (2023). Hate speech detection in social networks using machine learning and deep learning methods. *International Journal of Advanced Computer Science and Applications*, 14(5).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.

Wang, Y.-S. and Chang, Y. (2022). Toxicity detection with generative prompt-based inference. *arXiv preprint arXiv:2205.12390.*

WHATS2000$_($2024). $[roberta]lm - bff(manually)textclassification$. [Accessed: 2024-05-10].

Wirth, R. and Hipp, J. (2000). Crisp-dm: Towards a standard process model for data mining. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*, volume 1, pages 29–39. Manchester.

Wullach, T., Adler, A., and Minkov, E. (2020). Towards hate speech detection at large via deep generative modeling. *IEEE Internet Computing*, 25(2):48–57.