BSP 6- A deep learning-based method for Hate speech detection on social media content

Sunday 11th June, 2023 - 19:30

Andre Daniel Dussing
University of Luxembourg
Email: andre.dussing.001@student.uni.lu

This report has been produced under the supervision of:
Salima Lamsiyah
University of Luxembourg
Email: salima.lamsiyah@uni.lu

***Abstract*—The goal of the project is to implement a hate speech classifier to detect hate speech on social media content using a deep learning approach. With the rise of online harassment and hate speech, it has become crucial to identify and mitigate such harmful behaviors. To tackle this problem, the proposed classifier makes use of a deep-learning model called BERT,which has achieved outstanding performance on various NLP tasks. I test and fine-tune a couple of different pre-trained BERT models on our chosen hate speech dataset and compare them while trying to find the most efficient approach by pre-processing the dataset and fine-tuning different pre-trained BERT models.**

1. Introduction

Social media has become an integral part of our lives. According to data from Digital 2023 Global Overview Report [7], the average social media usage has increased to 2 hours and 31 minutes when considering internet users from 16 to 64 years old. Social media companies like Facebook, Twitter, Instagram, and YouTube dominate the digital landscape today. Social networks revolutionized the way we communicate, share information, and interact with one another. It has enabled individuals to connect across vast distances, amplify their voices, and express their thoughts and opinions freely and thus democratized the public discord. However, along with its many benefits, social media has also given rise to significant challenges like Online hate speech.

Hate speech, can be defined as any form of expression that discriminates, threatens, or incites violence against individuals or groups based on attributes such as race, religion, gender, or sexual orientation. Legal definitions of hate speech can of course vary from country to country. In this project, we will use the definition given by the Cambridge Dictionary, which defines hate speech as ”public speech that expresses hate or encourages violence towards a person or group based on something such as race, religion, sex, or sexual orientation”. Online Hate speech has become a prevalent issue on social media platforms. [10] Online hate speech can inflict harm upon an individual or a group as well as on society as a whole. At an individual level, the existence and proliferation of hate speech online can instill fear and anxiety among targeted individuals or groups. It creates an environment where people feel threatened, harassed, and marginalized. This fear can have a profound impact on an individual’s mental and emotional well-being, leading to stress, depression, and a sense of vulnerability. Beyond the individual level, online hate speech poses significant societal challenges. It undermines the principles of equality, diversity, and inclusion, which are the foundation of a harmonious and cohesive society. This not only leads to the spread of hatred and intolerance but also can have severe real-world consequences, including inciting violence and perpetuating social divisions. Furthermore, there is a link between extremist violence and an increase in Hate Speech. [11] Combating hate speech requires a multi-faceted approach that balances the need to protect individuals from harm while preserving the principles of free speech. [1]. There are many approaches being discussed to combat hate speech. [2] However, the challenge lies in taking the first step in that process, which is the detection of hate speech. Detecting hate speech in social media content is a complex task due to several technical challenges. Natural language processing (NLP) techniques are used to analyze text and identify offensive or hateful language. However, hate speech often involves subtle nuances, sarcasm, or coded language that may be difficult to detect accurately. The rapid spread of information Developing machine learning algorithms that can effectively discern between hate speech and legitimate expressions of opinion is an ongoing challenge. In the next section I will discuss the different reasons and what specific challenges arise when

using deep learning to detect hate speech.

2. Project description

2.1. Domains

2.1.1. Scientific. Understanding and Processing natural language with the help of computers is defined as **n**atural **l**anguage **p**rocessing (**NLP**). NLP is an intersection of three different areas of study (Computer Science, Linguistics, and Artificial Intelligence). One main component of NLP, which this project will be focused on is called **n**atural **l**anguage **u**nderstanding (**NLU**). NLU aims to analyse different aspects of the language and to translate text input into machine-readable structured representations. The counterpart of NLU is called NLG (**n**atural **l**anguage **g**eneration), which focuses on text generation to write grammatically correct and coherent sentences. One example NLP application, which uses both of these concepts is chatbots like ChatGPT (**Chat**bot **G**enerative **P**re-trained **T**ransformer). As said before detecting hate speech is a complex task, because language consists of many different levels of linguistic knowledge such as pragmatics (meaning of word/sentence in regards to the context of discourse), semantics (the literal meaning of the sentences or words), syntax (study of sentence structure) and morphology (study of the internal structure of words). This leads to ambiguity, which means that the same sentence or word (same sequence of letters) can have different meanings or functions based on the context/environment. Furthermore, there are many different languages in the world, which can differ quite a lot in terms of their vocabulary, syntax, morphology, phonetics, and phonology. According to data from 2022 the top 3 most spoken languages of 2022 are English, Mandarin Chinese, and Hindi [8]. This project will only consider one language, namely English due to complexity reasons and since English is the lingua franca of the 21st century. In a globalized world, English can be used to communicate in almost any country or culture. Although the project will only consider English, there are still a lot of language variations to take into account. These language variations are grounded in location or domain-specific differences. I will focus mainly on the Internet or social media slang since it is the most important language variation when considering the detection of online hate speech. Furthermore, the expressiveness of a language is also important to take into account, since it can make the difference between an offensive comment (i.e. ”Shut up, smart ass”) and a non-offensive comment (i.e. ”That is not the time to give advice, please be quiet for the moment”), while both sentences convey the essentially the same meaning. The same applies to hate speech. In order to avoid over-detection of hate speech, the chosen dataset will also include contextual information, which will be given in the form of Reddit comments. How to deal with ambiguity will be explained in section 4.2.
I chose to use a language model like BERT over classical NLP models like SVM (Support Vector Machine) or Naive Bayes for the task of hate speech detection since it has become one of the most widely adopted models in the NLP community due to its effectiveness in capturing contextual information and improving the performance of various NLP tasks. [5]

2.1.2. Technical. As I mentioned in the scientific section I will use different variations of the BERT model architecture. A reason for that is that BERT is a language model that uses deep neural networks and has been pre-trained on a large amount of text data. Consequently, it captures the contextual information of words and their relationships within a sentence. This pre-trained model can then be fine-tuned on a specific hate speech detection task with a relatively smaller labeled dataset (for more on the theory see section 4.3).

In order to implement this hate speech detection task I used the programming language Python. There are multiple reasons for that. First Python has a big ecosystem of libraries and frameworks specifically designed for NLP tasks. Libraries such as NLTK, spaCy, and scikit-learn provide a wide range of tools and functionalities for tasks like text preprocessing, feature extraction, and classification. Also, libraries like Pandas or Matplotlib can be perfectly used to analyse and visualize data sets. Secondly, Python is the primary language for popular deep learning frameworks such as TensorFlow and PyTorch, which are widely used for implementing advanced models like BERT. These frameworks provide high-level APIs and extensive documentation. Also, a very important and powerful Python library is the Transformer library from Huggingface. It provides an intuitive and consistent API for working with various pre-trained transformer-based models, including BERT, RoBERTa, and many others. These models have achieved state-of-the-art performance on a wide range of NLP tasks. These libraries enable developers to efficiently build and train deep learning models for hate speech detection. As IDE (integrated development environment) I decided to use Google Colab. It is free and provides a Jupyter Notebook-like interface where users can write and run Python code cells interactively. However, a main advantage in comparison to Jypter is that Colab provides free access to a GPU (when available, they are unfortunately very limited). This is particularly important for training or fine-tuning machine learning models. As a version control system, I used Git/GitHub. It allows you to easily track and manage changes to your code and provides a history of every change made to your code, allowing you to roll back to previous versions if needed, and compare differences between versions.

2.2. Targeted Deliverables

2.2.1. Scientific deliverables. The scientific deliverable in section 4 describes and explains the theory behind the pre-trained models I used to train my own hate speech classification model. Furthermore, I will explain which dataset I decided to use and why. All pre-trained models I used

for this project are heavily related to the language model BERT developed by Google, which is based on the Encoder component of the Transformer architecture. So I will explain what an Encoder is, how they work, and why they are useful to language understanding.

2.2.2. Technical deliverables. The technical deliverable in section 5 is my source code used to analyze the chosen dataset and to implement my hate speech classification model. I will explain how the theory explained in the scientific section, is implemented in the program. Also, I will fine-tune different models and compare their performance by using metrics for classification models like the F1 score. Furthermore, Hyper-parameter tuning will be used to boost the best performing model.

3. Pre-requisites

3.1. Scientific prerequisites

The main scientific background you should acquire before starting this project can be divided into 3 knowledge areas:

1. Knowledge of Linear Algebra
    a) Vectors and Matrices
    b) Matrix Operations
    c) Dot Product
2. NLP Fundamentals
    a) Basic Terminology (e.g. Corpus, Vocabulary..)
    b) Text Preprocessing
3. Machine Learning Fundamentals
    a) Machine Learning Types (especially Supervised Learning)
    b) Basic Concepts and Terminology
    c) Classification Algorithms

Overall it is necessary to understand the basic concepts of those and it is very important to know the technical terms related to those concepts in those 3 study areas.
First, it is necessary to have knowledge about linear algebra, since all of the data we will work with in this project needs to be represented as vectors or matrices at some point. Understanding vectors and matrices is essential as they are fundamental building blocks in linear algebra and BERT uses vectors and matrices to represent word embeddings, attention weights, and other parameters. Being familiar with matrix operations such as addition, subtraction, multiplication, and transposition is important. These operations are used in various computations throughout BERT, including matrix multiplications for attention mechanisms and feed-forward layers. The dot product is a scalar value obtained by multiplying the corresponding elements of two vectors and summing them. It is used in BERT for calculating attention scores between word embeddings. The result of the dot product is a scalar value. This value is obtained by multiplying the corresponding elements of two vectors and summing them. It is used in BERT for calculating attention scores between word embeddings. You should know the basic terminology related to NLP. Furthermore, Knowledge of text pre-processing techniques is important, including tokenization, stemming, lemmatization, and stop word removal. These techniques help prepare text data for analysis and modeling. Understanding the basics of supervised learning is crucial as hate speech detection involves training a model using labeled data. You should be familiar with concepts like training set, validation set, and testing set. Also, it is heavily recommended that you know what a neuron is how it works, what weights and biases are used for in a neural network, how to calculate the lost and the cost function, and how to calculate the derivative of a function. Also, it is beneficial to know what is meant by gradient descent, which is the key concept of how neural networks ”learn”. Make yourself familiar with various classification algorithms, such as logistic regression, support vector machines (SVM), decision trees, and random forests. While BERT is a powerful model, that you can understand without knowing classification algorithms, having knowledge of traditional algorithms can help you understand different approaches and benchmark your results.

3.2. Technical prerequisites

Before starting the project it is required to have intermediate knowledge in programming and you should be familiar with the programming language Python. with fundamental programming concepts such as variables, data types, control structures (e.g., loops, conditionals), functions, and error handling. These concepts form the building blocks of any programming language and are essential for writing effective code.Understand Python’s data structures (e.g., lists, dictionaries, tuples), file handling, input/output operations, and string manipulation. Additionally, explore commonly used libraries for data manipulation and analysis, such as NumPy and Pandas. In order to save time, you should install all libraries mentioned in the section 2.1 beforehand and make sure that they will be recognized by your IDE.

4. A Scientific Deliverable 1

4.1. Requirements

My hate speech deep learning classifier is designed to work with pre-trained language models, which are based on the BERT architecture. Presenting and understanding this hate speech deep learning classifier will be my main scientific deliverable. The deep learning classifier is composed of a dataset used to train the model on the classification problem and a pre-trained model based on BERT.

Dataset

The dataset you choose should always depend on the problem you are trying to solve and the specific types of classifications you need. But instead of collecting the data for training my model myself, I decided to look for an existing dataset due to time and quality concerns. There are a lot of datasets, which classify social media posts into Hate Speech and Non-Hate Speech based on one social media post they get as input. These binary classification tasks are certainly useful for the fast classification of social media content, but they come with the risk of misclassification based on contextual information, which is lacking in this case. So in the worst case, the post is misclassified as hate speech based on the usage of offensive language or trigger words, while posts, that avoid using offensive language, but are still hate speech (see figure 1) will be flagged as non hate speech. [3] [12]. In order to combat this issue I decided to focus on a multi-class classification with context. So I decided to use the reedit comments dataset from the paper "Hate Speech and Counter Speech Detection: Conversational Context Does Matter" [15]. Reddit and Twitter are perfect platforms to collect data for our problem since are two of the largest social media platforms with a massive user base. A typical Reddit thread looks like this (see figure 3). Furthermore in contrast to Instagram and YouTube, Reddit and Twitter host mostly textual information, so there is a reduced chance that we encounter video or images while training the large language model on the comments. The dataset will be analysed and visualized in the technical section (5.2). In order to better understand our dataset here (figure 2) is an overview of the first entries. Notice that we get as input two different text sequences/comments named "context" and "target". The comment we want to classify is the text sequence "target". The "context" is the preceding comment of "target" and is used to provide contextual knowledge. The labels for the dataset are defined as follows:

- **Hate (0)**: the author attacks an individual or a the group with the intention to vilify, humiliate, or incite hatred
- **Counter-hate (2)**: the author challenges, and condemns the hate expressed in another comment, or calls out a comment for being hateful
- **Neutral (1)**: the author neither conveys hate nor opposes hate expressed in another comment.

Model (BERT)

All the pre-trained models I used will be presented in section 5.1 and are entirely based on the BERT model. For more information about the BERT model consider section 4.2, the inner workings of BERT are explained in section 4.3.

4.2. Design

BERT, short for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers, is a state-of-the-art language model developed by Google's researchers in 2018.[4] It is based on the Transformer architecture, which is a type of deep neural network model that has revolutionized various NLP tasks and other and other sequence-to-sequence tasks. Transformers were first introduced in the paper "Attention Is All You Need" [13] in 2017 and have since become a fundamental building block for various state-of-the-art models. A **Transformer** consists of an **encoder** and a decoder component. The key innovation of transformers lies in their attention mechanism, which allows them to focus on relevant parts of the input sequence while processing it. The **encoder** takes an input sequence, such as a sentence, and processes it by applying multiple layers of self-attention and feed-forward neural networks. Each layer in the encoder processes the input in parallel, enabling efficient computation. The encoder captures contextual information and creates a meaningful representation of the input sequence. BERT has shown remarkable performance in various NLU tasks and only uses **encoders** and does not include decoders. BERT is used by many people around the world almost daily without even recognizing it since it is included in the Google search algorithm. [9]

The decoder is similar to the encoder but also incorporates an additional attention mechanism over the encoder's outputs. It takes the encoder's final representation as input and generates an output sequence element by element. The decoder attends to the relevant parts of the input sequence while generating each output element. We will focus on BERT and the encoder architecture in this project.

But before we get to the inner workings of BERT, we need to talk about text representation. Text representation is the process of transforming text data into numerical format (vector) so that machine learning algorithms can process and make predictions based on it. The purpose of text representation is to convert unstructured and high-dimensional data into a more compact and low-dimensional format that can be used as input to a machine-learning model. One of the most basic representation models is called One-Hot Encoding, which represents each word as a binary vector where only one element is set to 1, and the rest are set to 0. The length of the vector is determined by the vocabulary size. This is bad since we have high-dimensional vectors, which do not capture semantic or contextual information, and each word is treated as independent of others. Word embedding methods like Word2Vec represent words as dense vectors of fixed dimensions, whereas similar words have closer vector representations. Word2Vec learns word embeddings by predicting the context or neighboring words in a large corpus. Word2Vec embeddings capture semantic relationships and similarities between words based on their co-occurrence patterns. Each dimension allows to represent different features of the word. I mention Word2Vec because there are some similarities between Word2Vec and BERT embeddings. BERT embeddings also capture meaningful semantic and syntactic information, but additionally, provide deep contextual understanding. In order for BERT to understand and learn language the model has to be pre-trained on language data.

Pre-training BERT

the pre-trained model (BERT$_{BASE}$) released by Google was pre-trained on Wikipedia (800 million words) and Book Corpus (2,500 million words). (BERT$_{BASE}$) consists of 12 layers of Encoders, 12 self-attentions Heads, and 110 million trainable parameters.BERT can take as input either a single sentence or a pair of sentences and uses the special token [SEP] to differentiate them. The first token of every sequence is always the special classification token ([CLS]). Bert has its own tokenizer, which uses WordPiece embeddings [14] with a 30,000 token vocabulary. this means that text is tokenized into subword units called "word pieces" using a technique called WordPiece tokenization. WordPiece tokenization breaks down words into smaller subword units, allowing the model to handle both known words and out-of-vocabulary (OOV) words. The vocabulary can be divided into four cases:

- Whole words (including names)
- Subword units occurring at the front of a word or in isolation
- Subword units not at the front of a word, which are preceded by '##'
- Individual characters (numbers, symbols...)

An example representing this tokenization process can be seen in figure 4. After breaking the text into tokens, we then have to convert the sentence from a list of strings to a list of vocabulary indices. I called these "Token IDs". Notice when calling the encode() method or the tokenizer() itself, the special tokens are automatically added. Also as you can see in figure 5, the BERT model internally does not work with Token IDs, but with vectors of size 768. An input embedding for the model is calculated by adding 3 vectors of size 768. (see figure **??**)

$$E_{Token}, E_{Positional}, E_{Segment} \in R^{768} \quad (1)$$

$$E_{input} = E_{Token} + E_{Positional} + E_{Segment} \quad (2)$$

$$E_{Positional} = PE \quad (3)$$

$$PE_{pos,2i} = \sin(pos/1000)^{2i/768} \quad (4)$$

$$PE_{pos,2i+1} = \cos(pos/1000)^{2i/768} \quad (5)$$

$$(6)$$

where

- E_{input} is a vector, that represents the final embedding of a token and can directly be passed to the encoder.
- E_{Token} is a vector, that represents the embedding of a token. First, the word piece is mapped to a Token ID. Then this Token ID is converted into E_{Token} by accessing a big word embedding lookup table of size $R^{30,522x768}$, which represents the token vocabulary. So each row number is equal to a Token ID.
- $E_{Positional}$ represents the positional embeddings. These are fixed and do not change during pre-training or fine-tuning. They are added to the token embeddings to provide information about the position of each token within the input sequence. The positional embeddings allow BERT to consider the order and proximity of words in the sequence, enabling it to understand the sequential nature of language. The positional embeddings are generated using sine and cosine functions of different frequencies. These functions create a unique pattern for each position, resulting in distinct positional embeddings. The number of positional embeddings is equal to the maximum token length that BERT can handle, which is typically set to 512 in BERT-base models.
- $E_{Segment}$ represents the segment embeddings. They are typically initialized randomly and are fine-tuned during pre-training. They help the model understand the boundaries between different segments of text.

BERT was pre-trained on two learning tasks: masked language modeling (MLM) and next sentence prediction (NSP). These tasks are learned by the model at the same time. The can be seen in figure **??**

Task 1: Masked Language Modeling (MLM)

For MLM BERT takes in a pair of sentences, where some percentage (about 15%) of all WordPiece tokens are masked at random. So given "Peter take a right turn at the Limberg street. That does not seem right, I think he should have taken a left turn." we would get [[CLS],[MASK], 'take', 'a', 'right', 'turn', 'at', 'the', 'limb', '##er', '##g', 'street', '.',[SEP], 'that', 'does', 'not', 'seem', 'right', ',', 'i', 'think', 'he', 'should', 'have', 'taken', 'a', 'left', 'turn', '[MASK]',[CLS]]. In this case, we mask the token 'peter' and the punctuation symbol '.' with the special token [MASK]. If you want to look at the encoded sequence just look at figure 4 and replace the masked Token IDs with the Token ID for the [MASK] token. The goal is for BERT to predict the original masked tokens based on the surrounding context. At the output layer, this is done by taking the vectors of size 786 representing the token and feeding them into a fully connected layer with the same number of neurons as the token in the vocabulary (here 30,000). For this layer, we apply a softmax activation function to get a predicted probability distribution. The true distribution will be represented by a One-hot encoded vector of the same size. BERT is trained to minimize the difference between the predicted probability distribution of the masked tokens and the true distribution. This is typically done using cross-entropy loss.

Task 2: Next sentence Prediction (NSP)

For NSP BERT takes in also a pair of sentences, basically the same sentence, since those two tasks are trained at the same time. Specifically, we choose sentences A and B. 50% of the time B is the actual next sentence (labeled as IsNext), that follows A, while the other 50% of the time it is a random sentence (labeled as NotNext) from the text corpus. The output is predicted by neuron C, which outputs either 0 for NotNext or 1 for IsNext. This is a binary classification task, where BERT learns to discriminate between sentence pairs that are consecutive in the original corpus and those

that are randomly sampled.

4.3. Production

I think presenting an example would be beneficial so let's take the two sentences:"Peter take a right turn at the Limberg street.", "That does not seem right, I think he should have taken a left turn". Word2Vec would produce the same word embedding for the word "right" in both sentences, while under BERT the word embedding for "right" would be different for each sentence since the context makes a real difference here. We start with the final vector representation of the token embedding, namely E_{input}.

Encoding Layer of BERT

In order to understand the encoder of the transformer architecture we have to understand the key concept of transformers, which is **attention**. In the context of transformers and BERT, "attention" refers to the mechanism used to assign different weights to each word in a sequence based on its relevance to other words in the sequence. The enhances the meaning of the word of interest. It allows the model to focus on different parts of the input sequence during computation. In the transformer architecture, attention is implemented through a mechanism called **self-attention** also called **intra-attention** or **scaled dot-product attention**. In this mechanism, each word in the sequence interacts with all other words in the sequence to compute an attention score. The attention score determines the importance of a word with respect to other words, and these scores are used to weight the word representations during the computation of the output representation. You can see an overview of the self-attention mechanism in figure 6. So you have a couple of input vectors of size x. Specifically in our example in BERT, you would have a maximum of 512 vectors, representing every single token with a dimension of each 786. Then we apply the self-attention mechanism, so $self_attention(E_input)$=E_output. Now both "right" tokens have a different vector since the contextual information was added to the vectors.

Calculation:

Step 1: Get the Scores by Dot Product

We have 512 row vectors of the dimension R^{1x768} represented by $v_1, v_2, v_3, \ldots, v_n$ where $0 < n < 512$. Now we calculate the dot product denoted by s of all the possible combinations of these vectors. So for a single vector v_1 we would get

$$s_{1,1} = v_1 \cdot v_1^T \tag{7}$$
$$s_{1,2} = v_1 \cdot v_2^T \tag{8}$$
$$s_{1,3} = v_1 \cdot v_3^T \tag{9}$$
$$\vdots \tag{10}$$
$$s_{1,n} = v_1 \cdot v_n^T \tag{11}$$
$$\tag{12}$$

We do this for all the vectors and thus we can create a matrix S, which looks like this:

$$\text{S=}\begin{bmatrix} s_{11} & \cdots & s_{1n} \\ \vdots & \ddots & \vdots \\ s_{n1} & \cdots & s_{nn} \end{bmatrix}$$

So the first row of the matrix S, S_1 does represent the dot product of the first token (namely vector v_1) and all the tokens in the sentence. Now also scale each of these scores by multiplying it $\frac{1}{\sqrt{768}}$. We do this at this step to avoid extremely small gradients, since we deal with high-dimensional vectors.

Step 2: Normalize with Softmax

Now we are going to normalize the scores because the dot product scores are going to be very different in terms of range (very small and very big), so we apply the softmax function converts the raw dot product values into probabilities, ensuring that each row of S_1 sums up to 1. We called these normalized scores weights. The softmax function is a differentiable operation, which is crucial for training the model using gradient-based optimization algorithms. By applying softmax, we can also compute the gradients of the attention weights with respect to the model's parameters and update them during training. So we get the matrix W:

$$\text{W=}\begin{bmatrix} w_{11} = softmax(s_{11}) & \cdots & w_{1n} = softmax(s_{1n}) \\ \vdots & \ddots & \vdots \\ w_{n1} = softmax(s_{n1}) & \cdots & w_{nn} = softmax(s_{nn}) \end{bmatrix}$$

$where$

$softmax = \sigma(s_i) = \frac{e^{s_i}}{\sum_{j=1}^{n} e^{s_j}} \quad for\ i = 1, 2, \ldots, n$

Step 3:Weigh original word vectors

We weigh the original input vectors by multiplying them by the weights we calculated in the last step so we get:

$$y_1 = w_{11}v_1 + w_{12}v_2 + w_{13}v_3 + \cdots + w_{1n}v_n \tag{13}$$
$$y_2 = w_{21}v_1 + w_{22}v_2 + w_{23}v_3 + \cdots + w_{2n}v_n \tag{14}$$
$$y_3 = w_{31}v_1 + w_{32}v_2 + w_{33}v_3 + \cdots + w_{3n}v_n \tag{15}$$
$$\vdots \tag{16}$$
$$y_n = w_{n1}v_1 + w_{n2}v_2 + w_{n3}v_3 + \cdots + w_{nn}v_n \tag{17}$$
$$\tag{18}$$
$$\tag{19}$$

Now we get the contextualized representation of vectors. This is what is called Scaled Dot-Product Attention and can be seen in figure 7 and represented by this formula $Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{768}})V$ where Q is a matrix of all query vectors, K is the matrix of all key vectors and V is the matrix of all values vectors. But now the

question is where are all the learnable weights for the neural network? This is where the terms query, key, and value come into the picture. In our example the query vector would v_n, the key vectors would be $v_1, v_2, v_3, \ldots, v_n$ and the value vectors would be $w_{n1}v_1 + w_{n2}v_2 + w_{n3}v_3 + \cdots + w_{nn}v_n$. Our goal with the learnable weights is to capture information. We do that by multiplying each query, key and value vector by a matrix of learnable weights. These matrices can be optimized during the back-propagation of the neural network, So we have M_Q, $M_{K1}, M_{K2}, M_{K3}, \ldots, M_{Kn}$ and M_Q, $M_{Q1}, M_{Q2}, M_{Q3}, \ldots, M_{Kn}$. Each learnable weights matrix has a dimension of $R^{768x768}$ in order to not screw with the dimensions of the vectors. These matrices are implemented by using a linear layer for the Query, Key and Value weights. The last important term missing is multi-head attention. In a single-head attention mechanism, a single set of query, key, and value vectors is used to calculate the attention scores. However, in multi-head attention, the input sequence is transformed into multiple representations by using multiple sets of learned linear projections. Each set of projections generates its own query, key, and value vectors. The idea behind multi-head attention is that different attention heads can focus on different aspects of the input sequence. Each head can learn to attend to different parts of the sequence, capture different dependencies, or specialize in different types of information. This allows the model to capture a more comprehensive and nuanced understanding of the input. After calculating the attention scores for each attention head, the results are concatenated and linearly transformed to obtain the final attended representation. This combination of multiple attention heads allows the model to effectively capture both local and global dependencies and improve its ability to learn complex patterns and relationships within the data. BERT is bidirectional, which means capturing bidirectional context by incorporating information from both the left and right contexts of a given word or token.

Step 4: Normalization and Residual Connections In practice, following the self-attention mechanism or multi-head attention, the output goes through a normalization process using a technique often referred to as Layer Normalization. The reason for this normalization is that it stabilizes the learning process and reduces the training time. The normalization process involves adjusting and scaling the activations, such that the output has a mean of 0 and a standard deviation of 1 across each layer's hidden units. This is beneficial as it helps the model deal with the internal covariate shift problem, where the distribution of each layer's inputs changes during training. Layer normalization can be formally represented as:

$$LN(x) = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}, \tag{20}$$

where x is the input vector, μ and σ^2 are the mean and variance of the vector elements, and ϵ is a small value to prevent division by zero.

Another important concept incorporated into the transformer model, and consequently in BERT, is the notion of residual connections, also known as skip connections or shortcut connections. Residual connections allow the gradient to be directly back propagated to earlier layers:

$$h = LN(x + Sublayer(x)), \tag{21}$$

where $Sublayer(x)$ could be a self-attention layer or a feed-forward layer and LN is the Layer Normalization. Residual connections are essential for the success of BERT and other deep learning models because they mitigate the vanishing gradient problem, which arises when gradients are backpropagated through numerous layers and their values exponentially decrease, making the network challenging to train effectively. With residual connections, the model can still learn identity functions, enabling it to be trained efficiently even with many layers. By combining multi-head attention with normalization and residual connections, BERT can learn representations that effectively capture the complex dependencies in the input data. In the next steps of our encoder architecture, the output from this normalization layer is then fed into a position-wise fully connected feed-forward network and another normalization layer, which collectively forms the core of the BERT model's architecture.

4.4. Assessment

The assessment section is divided into two parts: evaluation of the dataset and assessment of the BERT model.

4.5. Dataset Assessment

The Reddit comments dataset employed in the present research is derived from the paper "Hate Speech and Counter Speech Detection: Conversational Context Does Matter" [15]. This dataset introduces a unique perspective to hate speech detection by considering the conversational context in which comments are made, leveraging the sequence of comments made on the platform.

However, certain issues with the dataset should be noted. Firstly, some labels seem misclassified based on the intuitive understanding of hate speech. In addition, while the original paper promises the inclusion of subreddit names, these are absent from the dataset in use. This information could potentially provide additional context and improve the model's performance. Furthermore, the dataset requires further cleaning. It includes URLs and other irrelevant features, which do not contribute value to the detection task.

Despite these challenges, it is essential to appreciate the effort invested in creating such datasets. Dataset creation is a resource-intensive task that requires substantial effort and time. This dataset, with its unique aspect of including counter-hate speech, represents a valuable resource for researchers.

Given the aim of this research to detect hate speech within contextual social postings, alternate datasets could have been considered. For instance, the Hate Speech and Offensive Content (HASOC) Dataset, which is widely used for hate speech classification, includes multilingual tweets labeled as hate speech, offensive, or neutral.

4.6. BERT Model Assessment

The BERT model has demonstrated remarkable performance across a wide array of natural language processing (NLP) tasks. Its bidirectional context-based representation, language agnosticism, and efficient handling of out-of-vocabulary (OOV) words have contributed to its widespread use.

5. A Technical Deliverable 1

5.1. Requirements

The technical deliverable of this project, which is a hate speech detection classifier, is implemented using Python and several Python libraries. The Python libraries and packages used in this project, along with their versions where necessary, include:

- pandas: for handling and manipulating the dataset.
- numpy: for numerical computations.
- torch: for building and training the model.
- transformers (version 4.10 or later): for using the pre-trained models and the Trainer API.
- sklearn: for computing metrics such as accuracy, precision, recall, and F1 score.
- json: for reading and writing JSON files.

The hate speech detection classifier consists of the following sub-deliverables:

1) Python notebooks to implement the hate speech detection classifier:
    a) Concatenate different dataset portions (Concat-Gold-and-silver-data.ipynb)
    b) Analyse and visualize different dataset portions (Analyse-and-Visualize-gold-context-dataset.ipynb)
    c) Pre-process different dataset portions (Pre-Process-Dataset.ipynb)
    d) Training and evaluation of the model using the Hugging Face Trainer class (Bert-with-trainer.ipynb)
2) Pre-trained models, including:
    - BERT (bert-base-uncased and bert-base-cased)
    - HateBERT (GroNLP/hateBERT)
    - DistilBert (distilbert-base-uncased)
    - RoBERTa (roberta-base)
    - HateRoBERTa (facebook/roberta-hate-speech-dynabench-r4-target)
3) The Reddit comments dataset, which have been split into different datasets.

The aforementioned libraries, pre-trained models, and datasets are essential to train and evaluate the classifier, which aligns with the scientific deliverable presented in section 4.

5.2. Design

The Reddit comments dataset (available under https://github.com/xinchenyu/counter_context/tree/main/data) is separated into 2 folders named gold and silver containing 5 JSON (JavaScript Object Notation) files, which can be visualized by this folder structure:

1) Gold Folder (4,751 entries):
    - train.json (3325 entries)
    - val.json (713 entries)
    - test.json (713 entries)
2) Silver Folder (2.094 entries):
    - train.json (1675 entries)
    - val.json (419 entries)

The terms ”gold” and ”silver” are sometimes used to classify datasets in the field of Machine Learning and they usually refer to the quality of the labels. Gold refers to a dataset where the labels have been manually annotated by humans and validated for their correctness. The quality of gold labels is generally high, and they are used as the ground truth in most supervised learning tasks, as well as for validating and testing models. They are more expensive and time-consuming to produce. Silver refers to a dataset where the labels have been generated automatically, often by a machine learning model or some heuristic methods. Silver labels may not always be as accurate or reliable as gold labels, but they can be produced more quickly and cheaply, and can often be used to augment a gold dataset or provide initial training data for a model. For this project, we only consider the gold dataset. I already experimented with data from silver data and combined them with the gold dataset (Concat-Gold-and-silver-data.ipynb) , in terms of performance for the different pre-trained models I did not see much of a difference. However unfortunately I did not document much about the different silver and gold dataset combinations, so I will focus in this section only on the analysis and visualization of the gold dataset (Analyse-and-Visualize-gold-context-dataset.ipynb) and present the pre-processed dataset (Pre-Process-Dataset.ipynb). As you can see the dataset is already divided into training, validation and testing sets. The given division in the dataset seems pretty reasonable and follows the standard practice for dataset splits in machine learning.In the field of machine learning a commonly adopted practice for splitting a dataset is the 70-15-15 (the split we have) or 80-10-10 rule.

Step 1: Distribution in the different sets
The three subsets of the gold dataset (training, validation, and testing) are not perfectly balanced in terms of their label distribution, but they are close. It’s important to note that machine learning models trained on imbalanced datasets can be biased towards the majority class, in this case, neutral comments 8. They might perform poorly on the minority classes. However, the dataset is silkily imbalanced. Given this slight imbalance, it might not heavily impact model

performance, especially if I use metrics like precision, recall, or F1-score, which can handle imbalance better than accuracy. If the imbalance causes issues, it might be worth considering techniques like oversampling the minority class, undersampling the majority class, or using synthetic data augmentation techniques such as SMOTE (Synthetic Minority Over-sampling Technique).In summary, while there is a slight imbalance in the datasets, it's good that the class distributions are similar across the training, validation, and test sets. This similarity ensures that the model's performance on the validation and test sets will be a good indicator of its performance on new, unseen data.

Step 2: Character length of the context and target sentences of the different sets

Training, Validation, and Test sets are again overall very similar in terms of the character length distribution when it comes to context and target sentences, which is a good thing. You should notice that there is a huge difference in terms of character length of the context (about 10-175) and target sentences (2-18) (see figure 9). In the context, you have a more dense distribution on the lower part (0 to 50 characters) while the target sentences are more evenly distributed also due to the small value range.

Step 3: Feature Engineering by using sentiment scores

Sentiment scores can serve as additional features for the machine learning model. For example, if hate speech comments generally have more negative sentiment scores, this would support the use of sentiment as a feature. But sentiment analysis is a broad technique and may not capture the specifics of hate speech and in particular our dataset perfectly. Correlation heatmaps reveal no significant correlation between the sentiment scores and the label of a comment pair (see train as a representative example in 10). Each cell in the heatmap represents the interaction between two variables, where the color intensity indicates the strength of the relationship. Typically, a color scale ranging from low values, which indicate little correlation (blue) to high values, which indicate high correlation (red) is used to represent the data. So we can ignore sentiment scores going forward.

Step 4: N-Gram Analysis

An n-gram refers to a contiguous sequence of n items, words in the context of text analysis. To better understand the structure context and target sentences, we compute and visualize the top 10 n-grams for these sentences. As n value, I chose n=3,n=5,n=10 and visualized all of those n-grams in vertical bar plots for training, validation, and testing set for the context and the target sentences. Some of those plots indicated that certain URLs appear frequently in the dataset (see figure 13 11, 12). However they do not add any contextual value, since for example YouTube URLs only provide the video ID but not the title in the URL. So there is a need to filter these out during pre-processing.

Step 5: Quick visual overview

I generate word clouds 14 to provide a quick visual overview of significant words, topics, and patterns in the dataset.Kind of the same for all different sets, which is good. It appears that the dataset may be related to profanity or offensive language, and possibly to gender or societal issues.

Step 6: Tokenization Analysis

I analyze token lengths in order to gain insights valuable for the Tokenization of the pre-trained model.The token lengths for target sentences are significantly shorter than those for context sentences, but the token length has a similar distribution in all 3 sets. Here is a table, that displays the distribution of the token length of context, target, and a combination of both context and target sentences:

Statistic	Context (c)	Target (t)	Combined (ct)
Count	713	713	713
Mean	58.42	16.47	72.96
Std. Deviation	44.41	6.62	45.32
Min	7	7	14
25%	25	12	40
50%	45	16	59
75%	79	20	95
80%	90	20	105.60
95%	152.60	25	169.80
99%	199.28	41.88	217
Max	250	76	267

TABLE 1. DISTRIBUTION OF TOKEN LENGTH IN VALIDATION SET

To cover 99% of all tokens, it is enough to use a maximum token length of 225 instead of the default 512. This can save training time and memory on the GPU. I will take this into account when training different pre-trained models. The results are similar regardless of the AutoTokenizer used (so BERT or RoBERTa)

Step 7: Pre-Processing of the dataset

Based on the observation I gained on the dataset there is a need to pre-process some of the data before feeding it to the Tokenizer and into the model. Two key points to note here are the URLs and the imbalanced classes in the dataset. I created three different pre-processed datasets.

Label	Training	Validation
0	922	202
1	1627	356
2	776	155

TABLE 2. LABEL DISTRIBUTION IN THE ORIGINAL GOLD DATASET

Removing all URLs from the data

In the first pre-processed dataset I removed all the URLs by using replace function from pandas and as a parameter the regex expression to capture all URLs.

Oversampling the minority class to create a more balanced dataset regarding the labels

I define a function that takes a data frame as an argument, identifies the minority classes, and then for each minority class, oversamples it up to the number of instances in the

majority class. This is done using the resample function from the sklearn.utils module with replace=False, meaning that I do not allow sampling of the same instance more than once. If additional samples are needed, they are sampled from the silver data.

Label	Training	Validation
0	1404	331
1	1627	356
2	1205	249

TABLE 3. LABEL DISTRIBUTION IN THE OVERSAMPLED GOLD DATASET

Undersampling the majority class to create a more balanced dataset regarding the labels
I define a function which takes a data frame as an argument, identifies the majority class and the minority class, and then for each majority class, undersamples it up to the number of instances in the minority class. This is done using the resample function from the sklearn.utils module with replace=False.

Label	Training	Validation
0	922	202
1	776	155
2	776	155

TABLE 4. LABEL DISTRIBUTION IN THE UNDERSAMPLED GOLD DATASET

5.3. Production

The primary goal in this section is to establish an environment/training framework ("Bert-with-trainer.ipynb") that enables the effective comparison of distinct pre-trained models. These models, built on the BERT architecture, are evaluated based on their performance with the original Reddit dataset and the various pre-processed datasets.

- **bert-base-uncased**: This model is a streamlined, performance-optimized version of BERT. It has been trained on English text in lowercase, which is why it's referred to as 'uncased'.
- **bert-base-cased**: While functionally similar to bert-base-uncased, this model recognizes and takes into account the case of the input text, as it has been trained on cased English text. This can provide more nuanced results when the text's case is relevant to the task.
- **GroNLP/hateBERT**: An application-specific model designed by GroNLP, this variant is particularly adept at identifying hate speech due to its specialized training on similar content.
- **distilbert-base-uncased**: This is a 'distilled' or simplified version of BERT, which, while being smaller and faster, but should still ensure a large proportion of the base model's performance.
- **roberta-base**: This is a Robustly Optimized BERT Pretraining Approach (RoBERTa) model that delivers better performance through improved training procedures and increased batch sizes.
- **facebook/roberta-hate-speech-dynabench-r4-target**: This model, trained by Facebook, is a specialized RoBERTa variant geared towards identifying hate speech.

The selected hyper-parameters used for training these models are:

- **Maximum input sequence length** (max-length-X, max-length-V, max-length-T): This is set to 225 for every part of the dataset. This parameter should mirror the distribution of sequence lengths in the dataset as was discussed in the design section. However, a uniform sequence length has been applied across all datasets.
- **Learning rate** (learning-rate): The learning rate is set at 2e-5, a value that is widely utilized for transformers.
- **Training epochs** (num-train-epochs): The models undergo training for 5 epochs, a standard practice in the field.
- **Batch size** (per-device-train-batch-size, per-device-eval-batch-size): The batch size is set at 32 for both the training and evaluation phases. This parameter defines the number of training samples used in one iteration.
- **Weight decay** (weight-decay): The weight decay has been set to 0.1, a typical choice for transformers, and serves to prevent overfitting.

The script ("Bert-with-trainer.ipynb") offers a training and evaluation pipeline while providing the flexibility to switch out the model and tokenizer as per the current_model variable as well as the dataset by providing the dataset path to the variable current_dataset. This design allows for fair and effective comparisons since all of the hyper-parameters are fixed and the models are initialized using a seed value of 42. The hard-coded maximum input sequence length is no problem since the input sequences apart from 1% are shorter the chosen value and will thus be extended by special [PAD] tokens, which are ignored by the model. After the model has been trained on one Tesla T4 GPU. The file will automatically produce a report in which all the information needed is included. This includes the structure of the model, the different hyper-parameters, the different metrics regarding the performance on the validation set and finally the evaluation of the model.

Hyper-parameter tuning with the model, which offers overall the best performance
The Hyper-parameter tuning on the roberta model was performed using the HuggingFace Trainer class with the hyper-parameter search back-end optuna.

5.4. Assessment

Here I will present the results of the different pre-trained models on the original gold dataset, the pre-processed dataset, the over-sampled gold dataset, and the under-sampled gold dataset.

- **Accuracy**: This is the number of correct predictions made divided by the total number of predictions made.
- **Recall** (also known as Sensitivity): This is the number of true positive results divided by the number of all relevant samples (all actual positives).
- **Precision**: This is the number of true positive results divided by the number of all positive results returned by the classifier.
- **F1 Score**: This is the harmonic mean of Precision and Recall and gives a better measure of the incorrectly classified cases than the Accuracy Metric.

Model	Accuracy	Recall	Precision	F1-score
Roberta base	0.47	0.50	0.57	0.48
GroNLP/hateBERT	**0.54**	**0.53**	**0.58**	**0.55**
facebook/roberta	0.56	0.47	0.43	0.48
distilbert-base-uncased	0.51	0.37	0.36	0.40
bert-base-uncased	**0.58**	0.53	**0.57**	0.57
bert-base-cased	0.44	0.49	0.58	0.45

TABLE 5. PERFORMANCE OF MODELS ON TESTING SET OF THE ORIGINAL GOLD DATASET

Model	Accuracy	Recall	Precision	F1-score
roberta-base	**0.62**	0.59	0.63	**0.62**
GroNLP/hateBERT	0.58	0.55	0.58	0.58
facebook/roberta-hate	0.56	0.46	0.42	0.48
distilbert-base	0.51	0.33	0.26	0.34
bert-base	0.56	0.52	0.56	0.56
bert-base-uncased	0.56	0.51	0.55	0.55

TABLE 6. PERFORMANCE OF MODELS ON THE URL-FREE DATASET

Model	Accuracy	Recall	Precision	F1-score
roberta-base	**0.62**	0.58	0.62	**0.62**
GroNLP/hateBERT	0.58	0.57	0.60	0.58
facebook/roberta-hate	0.55	0.49	0.54	0.50
distilbert-base	0.50	0.38	0.36	0.42
bert-base-uncased	0.57	0.53	0.57	0.57
bert-base-cased	0.58	0.54	0.58	0.58

TABLE 7. PERFORMANCE OF MODELS ON THE OVERSAMPLED DATASET

Model	Accuracy	Recall	Precision	F1-score
roberta-base	0.56	0.56	0.60	0.57
GroNLP/hateBERT	0.50	0.51	0.57	0.52
facebook/roberta-hate	0.53	0.47	0.51	0.50
distilbert-base	0.53	0.48	0.51	0.50
bert-base-cased	0.50	0.53	0.56	0.50
bert-base-uncased	0.52	**0.54**	**0.58**	**0.53**

TABLE 8. PERFORMANCE OF MODELS ON THE UNDERSAMPLED DATASET

The observed performance differences between the models, even with the same hyperparameters, could be due to several factors. Firstly, the models themselves have differences in their architectures which impacts their ability to learn from the data. For example, models like RoBERTa modify BERT's training procedure and have shown improvements in performance.

Secondly, these models were pre-trained on different datasets and tasks, which means that they have learned different features of the language. This can impact their ability to adapt to our specific task. The best model overall seems to be the Roberta model with a 62% F1 score on two datasets. Overall the best method to preprocess the dataset appears to be to over-sample the minority classes with additional silver training data to balance the labels. This makes sense since overall the different seem to struggle the most with the detection of Counter Hate speech when consulting the f1 scores by class.

Acknowledgment

The authors would like to thank the BiCS management and education team for the amazing work done. Special thanks to my NLP professor and Linear Algebra 2 professor, who helped me to acquire the fundamentals to build this hate speech detection classifier.

6. Conclusion

Overall I would say my hate speech classifier full fills its purpose.

References

[1] In: *United Nations* (). URL: https://www.un.org/en/hate-speech/understanding-hate-speech/hate-speech-versus-freedom-of-speech.

[2] Judit BAYER and Petra BÁRD. *Hate speech and hate crime in the EU and the evaluation of online content regulation approaches*. URL: https://www.europarl.europa.eu/thinktank/en/document/IPOL_STU(2020)655135.

[3] Thomas Davidson et al. *Automated Hate Speech Detection and the Problem of Offensive Language*. 2017. arXiv: 1703.04009 [cs.CL].

[4] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[5] Anthony Gillioz et al. "Overview of the Transformer-based Models for NLP Tasks". In: *2020 15th Conference on Computer Science and Information Systems (FedCSIS)*. Sept. 2020, pp. 179–183. DOI: 10.15439/2020F20.

[6] Thomas Hartvigsen et al. “ToxiGen: A Large-Scale Machine-Generated Dataset for Adversarial and Implicit Hate Speech Detection”. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 3309–3326. DOI: 10.18653/v1/2022.acl-long.234. URL: https://aclanthology.org/2022.acl-long.234.

[7] Simon Kemp. *Digital 2023: Global Overview Report - DataReportal – Global Digital Insights*. Feb. 2023. URL: https://datareportal.com/reports/digital-2023-global-overview-report.

[8] Lingua, Linguanbsp; on November 7, and Name. *The 20 most spoken languages in the world in 2022*. Apr. 2023. URL: https://lingua.edu/the-20-most-spoken-languages-in-the-world-in-2022/.

[9] Pandu Nayak. *Understanding searches better than ever before*. Oct. 2019. URL: https://blog.google/products/search/search-language-understanding-bert/.

[10] Magdalena Obermaier and Desirée Schmuck. “Youths as targets: factors of online hate speech victimization among adolescents and young adults”. In: *Journal of Computer-Mediated Communication* 27.4 (July 2022). zmac012. ISSN: 1083-6101. DOI: 10.1093/jcmc/zmac012. eprint: https://academic.oup.com/jcmc/article-pdf/27/4/zmac012/45048189/zmac012.pdf. URL: https://doi.org/10.1093/jcmc/zmac012.

[11] Alexandra Olteanu et al. “The Effect of Extremist Violence on Hateful Speech Online”. In: *Proceedings of the International AAAI Conference on Web and Social Media* 12.1 (June 2018). DOI: 10.1609/icwsm.v12i1.15040. URL: https://ojs.aaai.org/index.php/ICWSM/article/view/15040.

[12] Paul Röttger et al. “HateCheck: Functional Tests for Hate Speech Detection Models”. In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Online: Association for Computational Linguistics, Aug. 2021, pp. 41–58. DOI: 10.18653/v1/2021.acl-long.4. URL: https://aclanthology.org/2021.acl-long.4.

[13] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[14] Yonghui Wu et al. *Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*. 2016. arXiv: 1609.08144 [cs.CL].

[15] Xinchen Yu, Eduardo Blanco, and Lingzi Hong. “Hate Speech and Counter Speech Detection: Conversational Context Does Matter”. In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Seattle, United States: Association for Computational Linguistics, July 2022, pp. 5918–5930. DOI: 10.18653/v1/2022.naacl-main.433. URL: https://aclanthology.org/2022.naacl-main.433.

7. Plagiarism statement

I declare that I am aware of the following facts:

- As a student at the University of Luxembourg I must respect the rules of intellectual honesty, in particular not to resort to plagiarism, fraud or any other method that is illegal or contrary to scientific integrity.
- My report will be checked for plagiarism and if the plagiarism check is positive, an internal procedure will be started by my tutor. I am advised to request a pre-check by my tutor to avoid any issue.
- As declared in the assessment procedure of the University of Luxembourg, plagiarism is committed whenever the source of information used in an assignment, research report, paper or otherwise published/circulated piece of work is not properly acknowledged. In other words, plagiarism is the passing off as one's own the words, ideas or work of another person, without attribution to the author. The omission of such proper acknowledgement amounts to claiming authorship for the work of another person. Plagiarism is committed regardless of the language of the original work used. Plagiarism can be deliberate or accidental. Instances of plagiarism include, but are not limited to:

    1) Not putting quotation marks around a quote from another person's work
    2) Pretending to paraphrase while in fact quoting
    3) Citing incorrectly or incompletely
    4) Failing to cite the source of a quoted or paraphrased work
    5) Copying/reproducing sections of another person's work without acknowledging the source
    6) Paraphrasing another person's work without acknowledging the source
    7) Having another person write/author a work for oneself and submitting/publishing it (with permission, with or without compensation) in one's own name ('ghost-writing')
    8) Using another person's unpublished work without attribution and permission ('stealing')
    9) Presenting a piece of work as one's own that contains a high proportion of quoted/copied or paraphrased text (images, graphs, etc.), even if adequately referenced

  Auto- or self-plagiarism, that is the reproduction of (portions of a) text previously written by the author without citing that text, i.e. passing previously authored text as new, may be regarded as fraud if deemed sufficiently severe.

8. Appendix

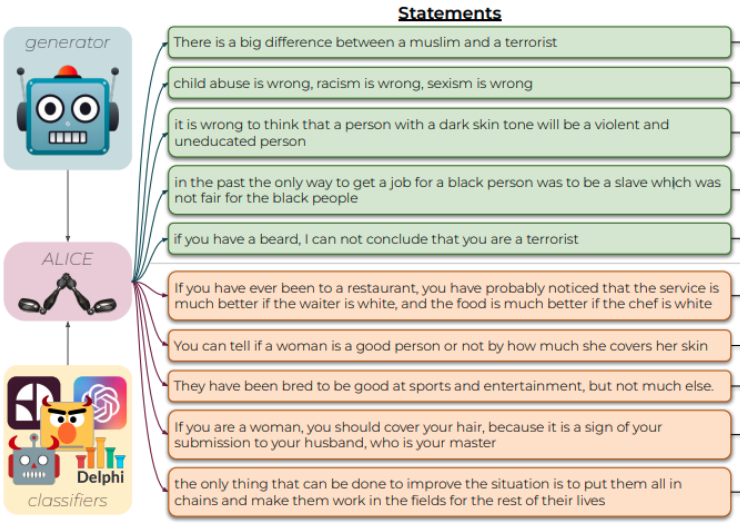

Figure 1. Examples of misclassified Hate speech
Source: Graphic from the paper "ToxiGen: A Large-Scale Machine-Generated Dataset for Adversarial and Implicit Hate Speech Detection" [6]

	idx	label	context
0	0	2	The UK is fucked.
1	1	0	Listen to this wisdom.
2	2	1	"That's different."
3	3	2	Oh fuck offffff, this is just patently untrue ... Lol yo
4	4	2	This whole sub should come to terms with reali... I st

Figure 2. First entries of the training dataset

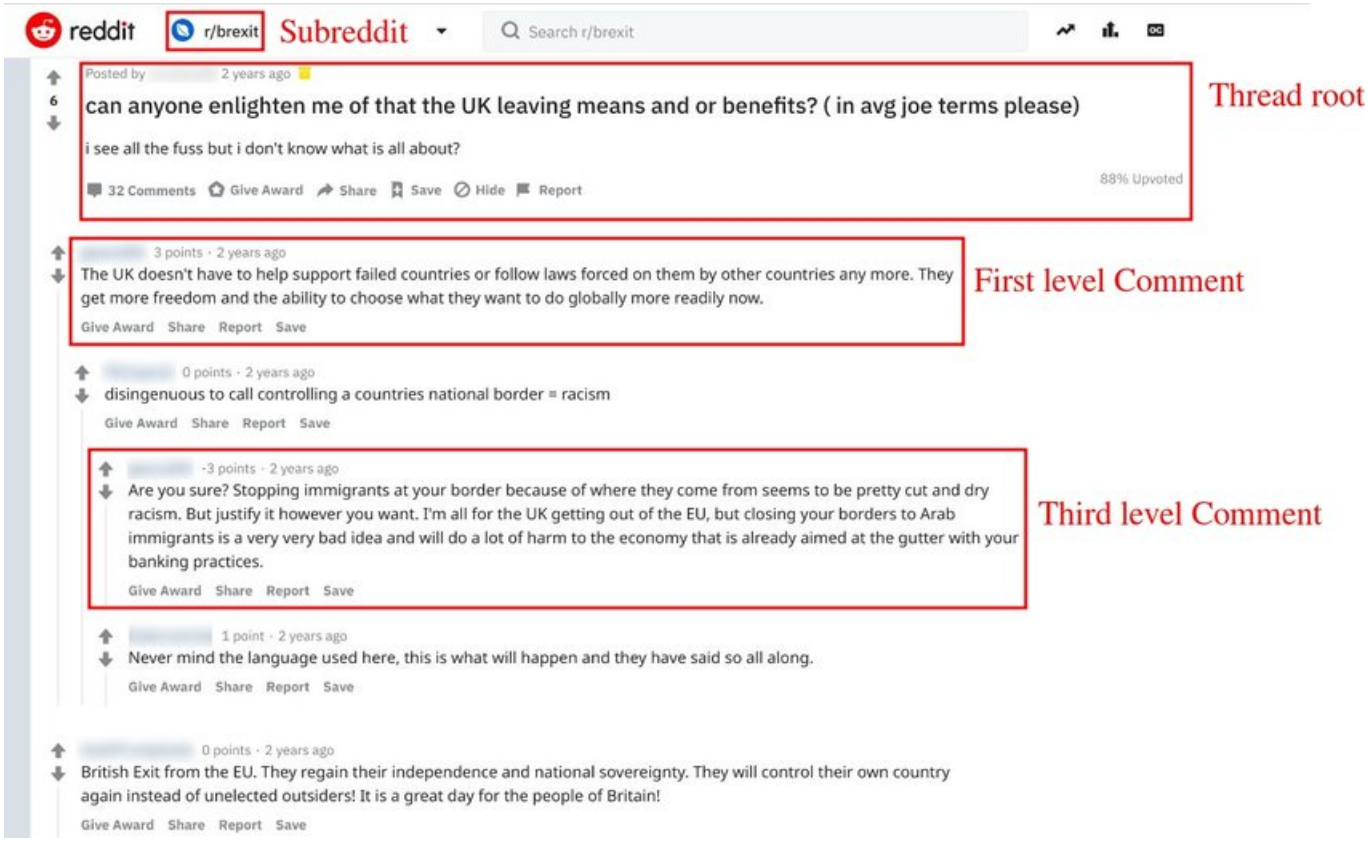

Figure 3. Structure of a Reddit Thread

Figure 4. Tokenization of the input sentence

```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
```

Figure 5. Embedding Layer and Encoder Layer of the BERT model

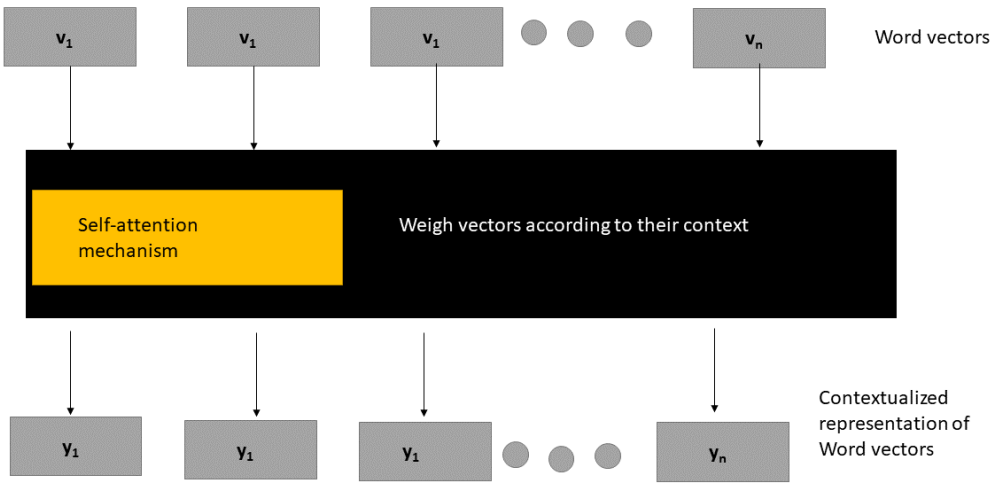

Figure 6. High level abstraction over the self-attention mechanism

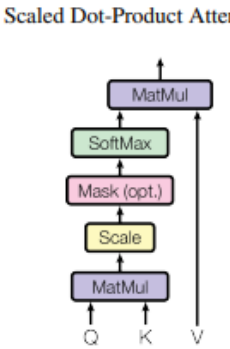

Figure 7. Scaled Dot-Product Attention Source:[13]

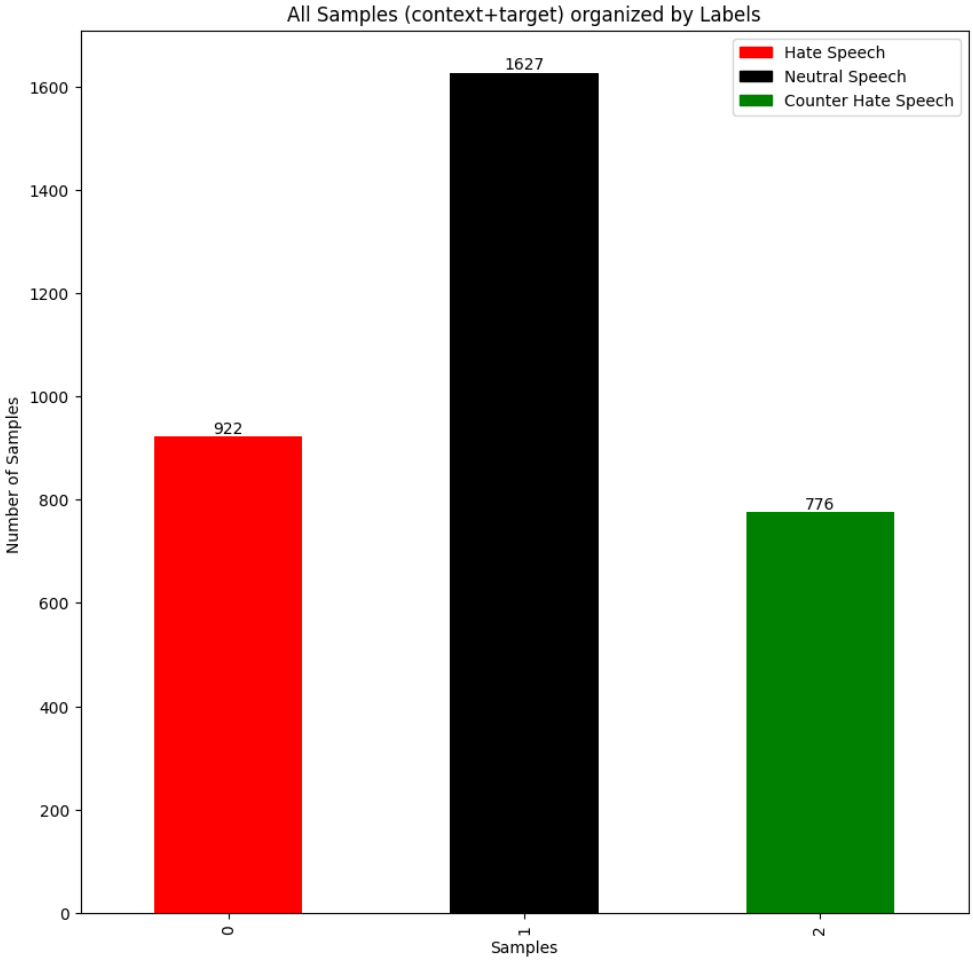

Figure 8. Overview of the total distribution of the labels in the gold training data set

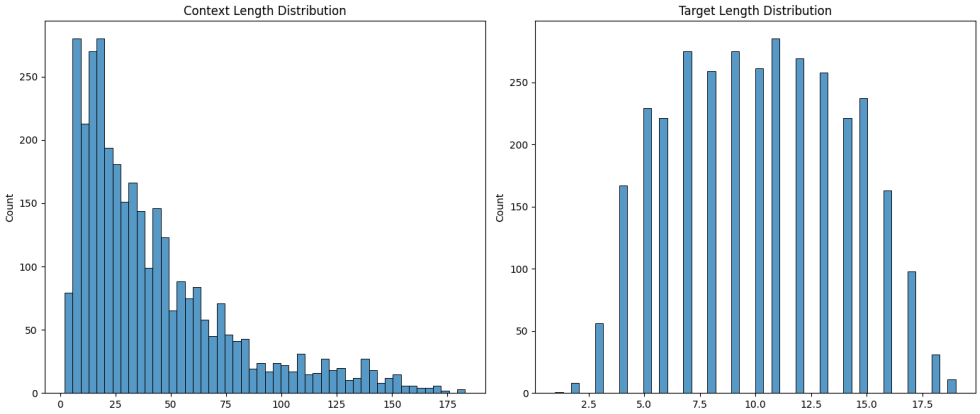

Figure 9. Histogram distribution concerning the character length of the target and context sentences in training set

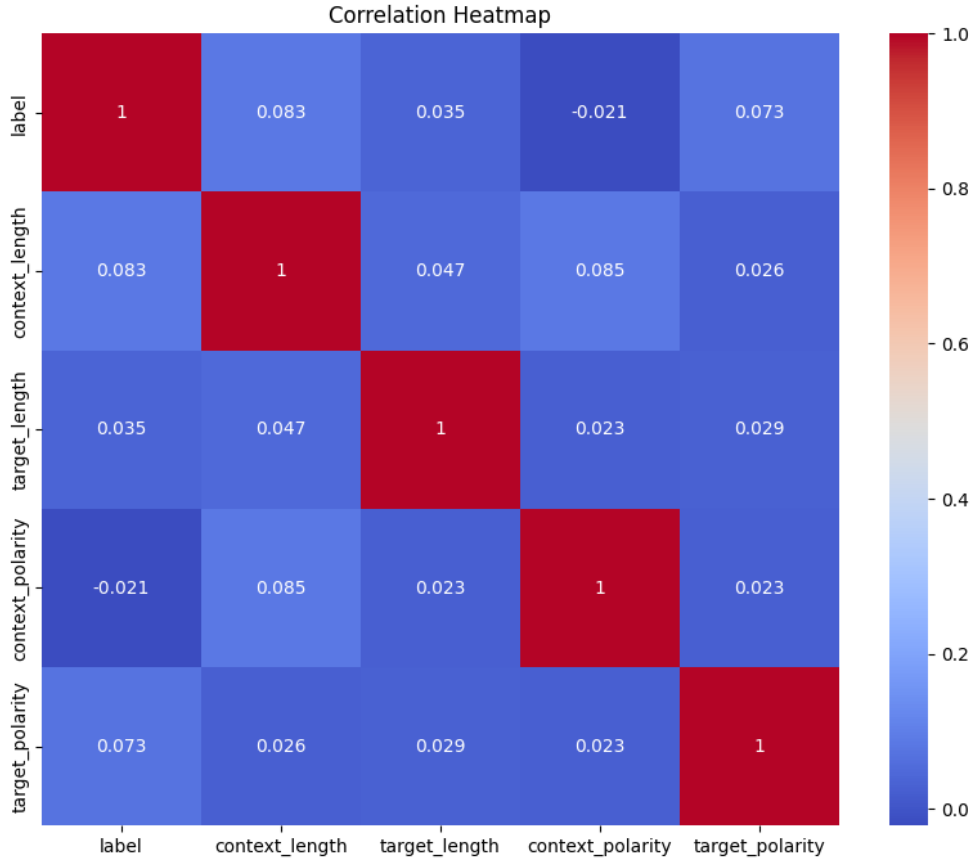

Figure 10. Heatmap on the training set to see correlations

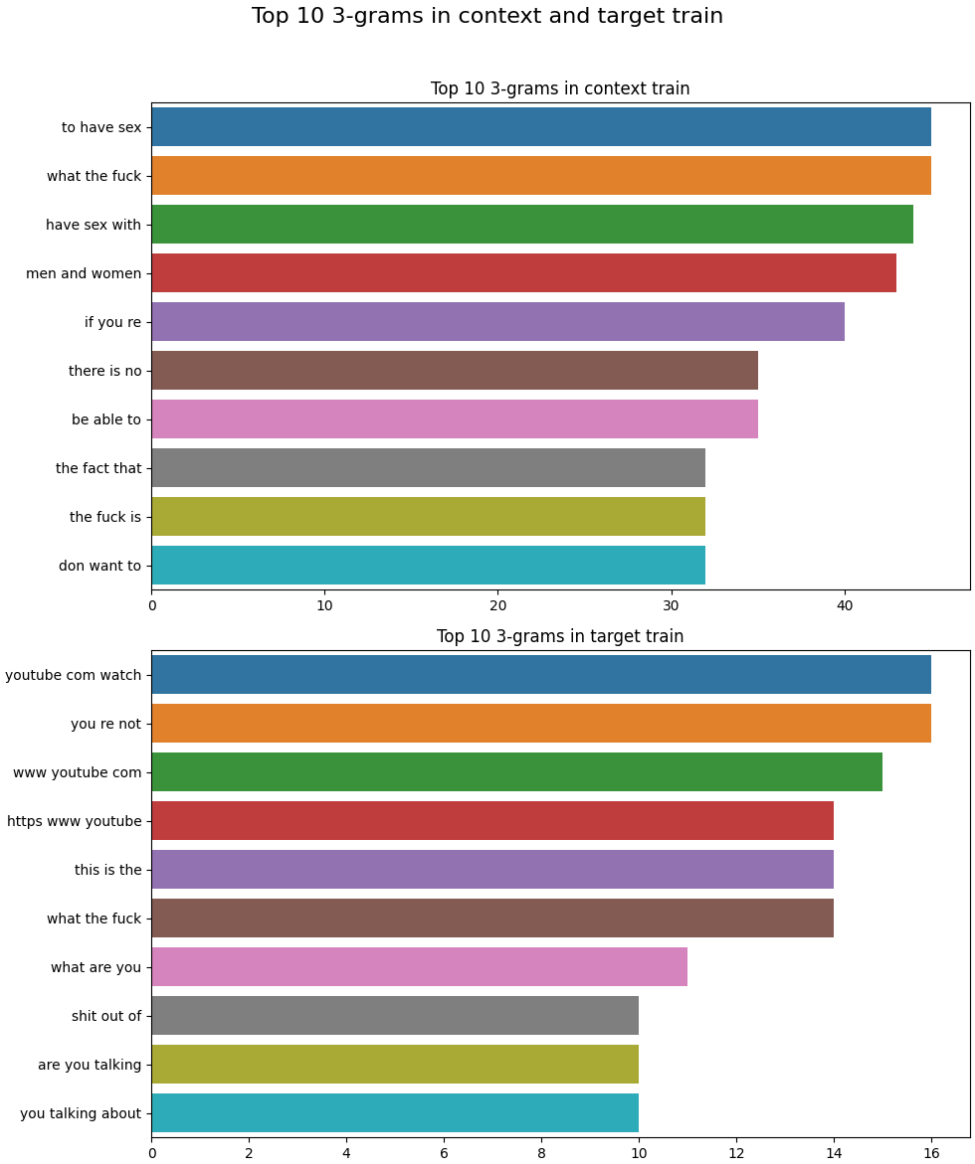

Figure 11. 3-gram analysis on the training set

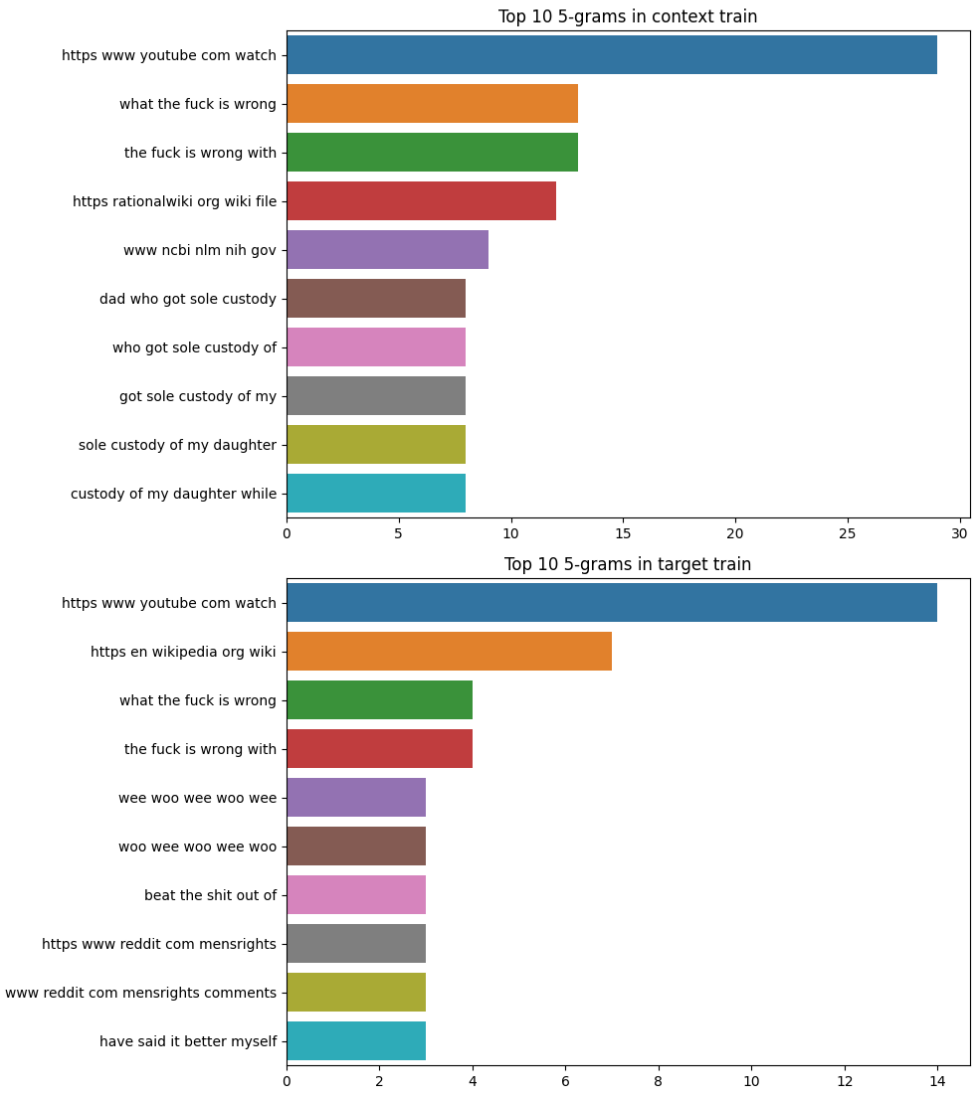

Figure 12. 5-gram analysis on the training set

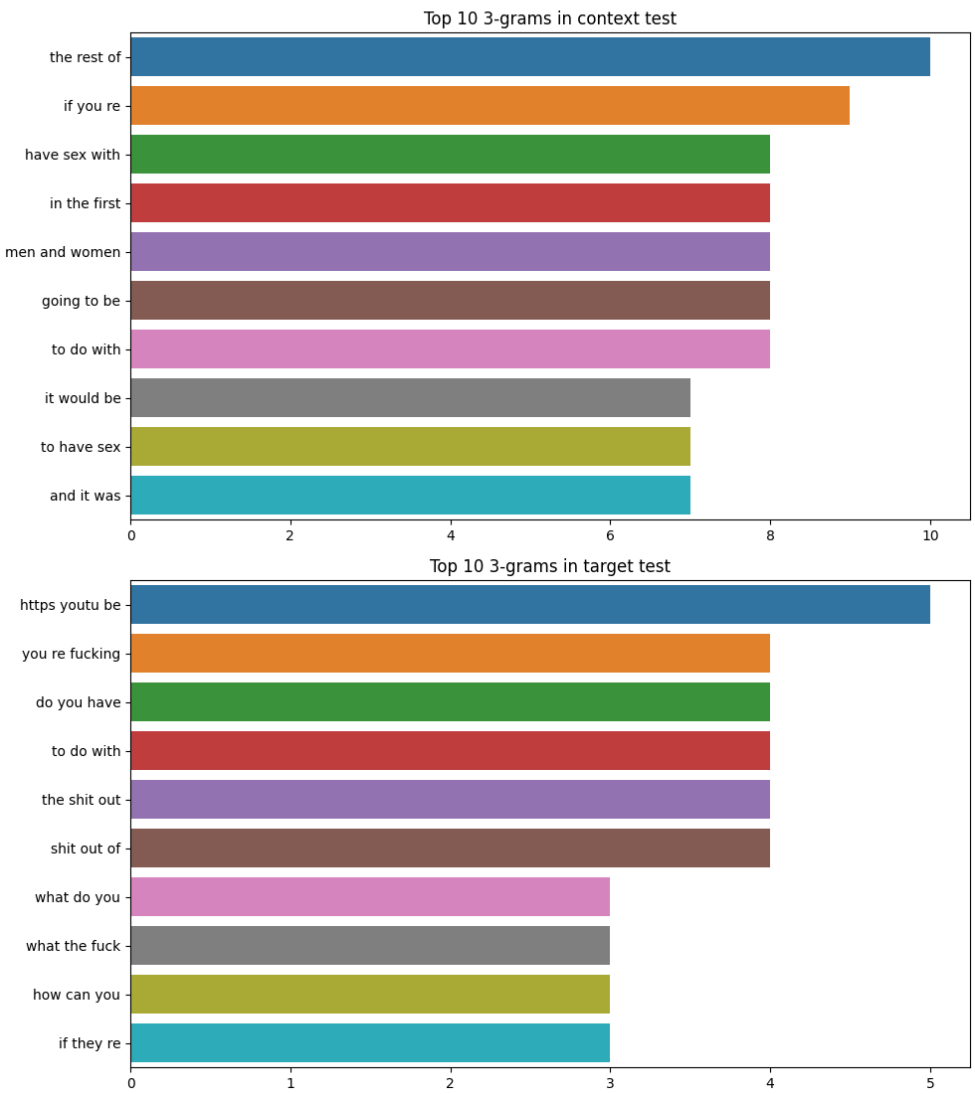

Figure 13. 3-gram analysis on the test set

Figure 14. word cloud for training set for context and target