MASTER'S THESIS 2023

Multi-Label Toxic Comment Classification Using Machine Learning: An In-Depth Study

Matilda Froste, Mosa Hosseini

ISSN 1650-2884
LU-CS-EX: 2023-27

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2023-27

Multi-Label Toxic Comment Classification Using Machine Learning: An In-Depth Study

Multi-Label klassificering av hatiska kommentarer: en omfattande studie

Matilda Froste, Mosa Hosseini

Multi-Label Toxic Comment Classification Using Machine Learning: An In-Depth Study

(Toxic comment detection to combat hate speech online)

Matilda Froste
ma7874fr-s@student.lth.se

Mosa Hosseini
mo0708ho-s@student.lth.se

July 3, 2023

Master's thesis work carried out at
Prevas, Malmö

Supervisors: Björn Granvik, bjorn.granvik@prevas.se
Pierre Nugues, pierre.nugues@cs.lth.se

Examiner: Jacek Malec, jacek.malec@cs.lth.se

Abstract

The classification of toxic comments is a well-researched area with many techniques available. However, effectively managing multi-label categorization still requires a considerable amount of work. In this thesis, we performed a classification experiment on over 200 thousand comments from the Jigsaw toxic comment competition data available on Kaggle. We aimed to optimize a model to identify six different categories of hate speech. Initially, we implemented a baseline model using a simple vectorization technique and logistic regression. Subsequently, we compared this model with more advanced approaches that employed elaborate vectorization techniques in conjunction with recurrent neural networks and transformers. After thorough analysis, we found that a fine-tuned transformer-based model called RoBERTa yielded the best performance, achieving a mean macro average F1-score of 0.808. This model surpassed the previous state-of-the-art set by van Aken et al. (2018), which achieved an F1 score of 0.791. Finally, we integrated the optimized model in a web application to visualize the toxicity of messages.

Keywords: Natural language processing, machine learning, offensive speech detection, transformers, multi-label classification

Acknowledgements

We want to truly thank our supervisor Pierre Nugues at LTH for all the help and ideas we have received during the master thesis. His input has been invaluable, and having him as a supervisor has been a pleasure. Our examiner Jacek Malec has also assisted us with many answers along the way, for which we are very grateful.

We express our sincere gratitude to Björn Granvik at Prevas for his contribution, ideas, and enthusiasm for this thesis. His dedication to making the world a less hateful place is truly admirable. Finally, thanks to all the colleagues at Prevas who have assisted us along the way and welcomed us as colleagues.

Contents

Chapter 1
Introduction

1.1 Context

Hateful comments on the internet are a significant concern. Discussions, posts, and articles can quickly become attacked by discriminatory, offensive, threatening, or otherwise harmful comments. This problem is particularly destructive as it can increase the division between groups of people or lead to severe mental health consequences targeting individuals.

Toxic comment classification is a significant area of research that involves developing and applying techniques to identify and categorize offensive or harmful comments. Despite the extensive study conducted in this field, effectively managing the multi-label categorization of toxic comments remains challenging. Accurately categorizing toxic comments becomes complex due to the subjective nature of determining the appropriate labels for each comment. Furthermore, multiple labels add a layer of complexity, requiring a robust approach to handle diverse categories effectively.

This thesis addresses the challenges above by exploring various machine-learning techniques designed explicitly for multi-label classification in the context of toxic comment classification. This research identifies the most suitable methods for accurately categorizing toxic comments with multiple labels by investigating and comparing different methodologies.

1.2 Problem Formulation

Natural language processing (NLP) has experienced a surge in popularity since its groundbreaking advancements in the mid-twentieth century (Foote, 2019). One of the prominent applications of NLP is text classification, which encompasses a wide range of areas, including sentiment analysis of movie reviews, customer feedback analysis, and categorization of news articles (Tunstall et al., 2022). Although text classification may appear simple at first glance, finding the optimal model for this task can be challenging.

The process of finding the optimal model involves several steps. Firstly, selecting a suitable dataset that aligns with the specific classification objective is essential. Once the dataset is determined, we need to vectorize the text data, which means transforming raw text into a numerical representation based on the features of the text. This step requires careful consideration of various techniques and approaches, such as TF-IDF, word embeddings, or transformer encoders.

After preprocessing the data, the next step is to choose an appropriate model architecture. Many models can classify text, including logistic regression, recurrent neural networks (RNNs), and transformer-based models. Each model has its own set of hyperparameters that we must fine-tune for optimal performance. Deciding on the best model architecture beforehand can be a complex task due to the diverse characteristics of different datasets and the wide range of possibilities and combinations of different parameters. Therefore, this thesis investigates and compares numerous model architectures, as well as parameters, to determine the most suitable approach for the given dataset. The ultimate aim is to integrate the optimized model into a web application, thus having a product able to filter out hate speech without human interaction.

1.3 Contributions

1.3.1 Contribution to Research

This thesis gives an account of toxic comment classification using machine learning. It thoroughly explains the attempted implementations and their performance, which could inspire future research and projects. The mean macro average F1 score obtained beats the previous record found by van Aken et al. (2018). Therefore, the best model obtained is a state-of-the-art model we can apply to online platforms. The actual method, in the form of code online, is available for anyone who wants to test or improve the final tool [1].

Additionally, this report explains why a hate-comment categorization tool is needed. The hope is that this project will serve as a tool for diminishing hate speech online.

1.3.2 Division of Work Between Authors

In the early parts of the master thesis, Mosa researched existing models and attempted simple natural language processing methods. Mosa compared different data sets and performed introductory exploratory data analysis on the data sets of interest. Matilda, on the other hand, focused on structuring the work. She did the planning of the project, setting dates, communicating with professors and Prevas, as well as constructing guidelines.

Once we had chosen the data set and the objective, we jointly started researching the techniques we had found in agreement with our LTH supervisor, Nugues. Mosa started by illustrating and analyzing the data in a visual sense. When we had prepared the data, Mosa started with the TF-IDF implementation, whereas Matilda researched and attempted the more complex techniques. Simultaneously, both attempted the fastText and GloVe embeddings and picked the best performing. Mosa continued this part by constructing the recurrent

[1] `https://huggingface.co/spaces/4stra/mean_or_clean`

neural network model architectures for the fastText and GloVe embeddings, whereas Matilda went on to experiment with transformers.

Matilda performed the initial feature extraction for the transformers method. She created the code for the feature extraction, whose output we wanted to send as the input to the logistic regression model. However, she needed a powerful computer to perform the extraction. Since Mosa could do it, he ran the code for the feature extraction. Both authors were making multilingual transformer attempts. Mosa performed the final parts of the transformer procedure, where he compared multiple pre-trained models and fine-tuned the models.

At the end of April, we held our final presentation at Prevas. We contributed equally to preparing the slides, choosing the content, and talking during the presentation.

Once we had presented at Prevas, Matilda focused almost entirely on writing the report, whereas Mosa continued to find the optimal model. He simultaneously visualized the results and added them to Section 5.1. Matilda finished chapters 1, 3, 4, and 5.

In the final stage, Matilda mainly focused on finalizing, cleaning, organizing the report, and creating designs using Canva, whereas Mosa focused on perfecting the results, application, and visualization parts. He also cleaned the code, organized the files, and created a GitHub repository. Matilda wrote the popular science summary. We put equal effort into the final presentation at LTH.

1.4 Related Work

Davidson et al. (2017) introduced significant contributions to hate speech categorization. The authors created a dataset of tweets and annotated them as toxic or non-toxic, providing a valuable resource for hate speech classification research. Additionally, they presented a binary classification approach for identifying hate speech based on this dataset. Furthermore, they highlight the difficulty of letting a computer predict toxicity in text, as text is nuanced and toxic speech has many interpretations. They discuss the bias of annotation and the variation of toxic language on online platforms. Various feature extraction techniques are discussed in the paper, capturing different characteristics. Support Vector Machine (SVM) operated as the machine learning classifier and ultimately presented effective results compared to baseline model results. Davidson et al. (2017) provide insights into the challenges and advancements in hate speech categorization.

In 2012 Warner and Hirschberg presented a categorization approach to hate speech. In their work, the authors discuss their definition of hate speech and detail the process of creating a corpus. They also provide techniques for detecting hate speech that can identify ways individuals try to bypass common “dirty word” filters. Due to the early studies, their results are sufficient, with an F1 score of 0.63, but far from being as good as some later models.

Georgakopoulos et al. (2018) explain the need for prominent hate speech models, the uses and limitations of existing models, and the abundance of information on the internet. They specifically compare models based on convolutional neural networks with traditional bag-of-word models and find that the CNN approach provides better results than the more straightforward approaches.

A common trait of the papers mentioned is the difficulty defining toxic hate speech. The bias of comment annotation is evident since language is nuanced, and human annotators have different perceptions. Schmidt and Wiegand (2017) addresses subjectivity in labeling hate

speech. Apart from describing hate detection methods, the authors also describe strategies to combat the issue of subjectivity. Ross et al. (2017) delves deeper into the reliability of hate speech annotations, specifically in the context of the European refugee crisis in 2015. They performed a study with two internet groups. They presented a definition of toxic speech to one group, whereas the other group did not see a definition beforehand. Ross et al. concluded that toxic speech is not always binary and that more detailed instructions on toxic speech annotation are needed before labeling comments.

In 2020, Kauranen classified hate speech as either hateful or not, in other words, binary classification. In doing so, he got an arguably good F1 score as a result. In this Master's thesis, we aim to build upon the approaches presented by Kauranen. However, we decided to explore the possibility of finding a better model and doing multi-label categorization since the field has advanced drastically.

Wang et al. (2017) explain the challenges and difficulties with multi-label classification and mentions that the F1 score is a good metric for this problem. We used their paper as a source of information for multi-label categorization.

Additional relevant work for our master thesis was research within neural networks and Transformers. By examining previous work in these areas, we gained valuable insights and knowledge that contributed to developing and implementing our models. Regarding neural networks, and recurrent neural network in particular, the research within long short-term memory (LSTM) by Hochreiter and Schmidhuber (1997) and gated recurrent unit (GRU) by Cho et al. (2014) was essential sources of information.

The second complex model architecture we use is based on Transformers, first introduced by Vaswani et al. (2017). The module "Transformers" on Huggingface, introduced by Wolf et al. (2019), has pre-trained transformer tokenizers and pre-trained models and was crucial for our thesis work.

The dataset in this thesis was initially used in a Kaggle competition called 'Toxic Comment Classification Challenge' (cjadams et al., 2017). The comments belong to between zero and six different categories of toxicity. Another dataset regarding offensive language classification is the OffensEval dataset, introduced by Zampieri et al. (2019). The comments in their dataset are either labeled as offensive or not. They can further be labeled as a targeted insult or threat if offensive.

van Aken et al. (2018) explain how they performed multi-label classification on the Kaggle dataset we use. The authors tackle the difficulties of identifying toxic text but still successfully obtain positive outcomes on the dataset. We will use their findings as a foundation to outperform.

1.5 Background

In this section, we will explain the concept and terminologies that will appear in this paper.

Artificial Neural Networks

Artificial Neural Networks (ANN) are computing systems that are inspired by biological neural networks. It is a network of interconnected nodes with assigned weights. Neural networks are a fundamental component of machine learning and have been successful in

solving complex tasks such as classification, regression, and decision-making. The training procedure involves adjusting the weights of neural networks so that the loss is minimized. Loss is the error between the predicted and the actual label.

Machine Learning

Machine learning, a branch of artificial intelligence, empowers computers to learn from data and make predictions without relying on explicit programming.

Transfer Learning

Transfer learning is a machine learning technique that involves using a pre-trained model, which was initially trained on one task, to solve a different but related task. Instead of starting the learning process from the beginning, the pre-trained model's knowledge is transferred and adjusted (fine-tuned) for the target task. This approach results in enhanced performance and faster convergence compared to starting from scratch.

Hyper parameter optimization

When training a machine learning model, certain parameters cannot be learned directly from the data but need to be set manually. These parameters are known as hyperparameters.

Hyperparameters play a vital role in optimizing the model, and finding the optimal values for these hyperparameters is a crucial task for machine learning engineers. The specific hyperparameters that need to be adjusted depend on the chosen model architecture and the learning algorithm being used.

Some commonly optimized hyperparameters are:

- **Batch size**
  Most of the time, it is computationally expensive to train all data at once, so we divide them into smaller batches. Batch size refers to the number of training samples processed together. The choice of batch size can affect performance.

- **Optimizer**
  To minimize loss, we need an optimization method. Choosing a suitable optimization method is an essential part of training. Common optimization methods for classification tasks include SGD, Adagrad, Adam, and Nadam. SGD and Adagrad move in the negative gradient direction, while Adam and Nadam are based on momentum.

- **Learning rate**
  All of these optimizers rely on a hyperparameter called the learning rate. The learning rate determines the magnitude of steps towards the minima. If the step is too large, we might miss the minima; if it is too small, convergence requires many iterations.

- **Number of hidden layers and neurons**
  These hyperparameters determine the architecture and capacity of the neural network. The number of hidden layers and neurons in each layer affects the model's ability to learn complex representations. Too few layers or neurons can result in underfitting,

while too many can lead to overfitting. Experimentation and understanding of the problem at hand are necessary to find the right balance.

- **Dropout**
  Dropout is commonly used during model training to prevent overfitting. This technique randomly disables nodes with a certain probability, effectively cutting some connections. It promotes generalization in the model.

- **Maximum word length**
  In some cases, limiting the number of words in the training data can lead to faster training and potentially better performance. Determining the appropriate number of words to include is a hyperparameter that requires experimentation.

Chapter 2
Data

Data is essential to any machine learning model, and toxic text classification is no exception. There are many open-source hate speech corpora on the internet. Some of these corpora offer binary labels, distinguishing between offensive and non-offensive content. Others provide a limited set of specific categories for classification. However, most hate speech datasets are designed to be binary or multi-class, wherein each comment belongs to a single category. Nevertheless, it is essential to acknowledge that toxic messages often incorporate different forms of toxicity. For instance, a hateful comment can be both threatening and insulting. Hence, our objective was to explore a dataset encompassing numerous distinct categories while allowing for a comment belonging to multiple hate classes. The dataset we chose, the 'Toxic Comment Classification Challenge' dataset from a Kaggle competition (cjadams et al., 2017), satisfies these criteria effectively.

2.1 Dataset

The dataset contains 223,548 manually annotated comments from Wikipedia, of which 159,571 belongs to the train and 63,978 to the test set. Figure 2.1 shows an example, where the labels toxic, severe toxic, obscene, threat, insult, and identity hate are seen. A comment which does not belong to any of the categories is considered not offensive.

The language in the dataset is predominantly English, but in the training set, there are some non-English sentences wrapped in English text. An example of such a sentence is

> \n\n Spventi: ""but perhaps it's \n safest to discuss ... Plus, the explanation
> 「松浦靜山の『甲子夜話』にみえる川柳。おそらくはよみ人知らず」
> sounds much more reliable than
> 「江戸後期の平戸藩主・松浦静山が詠んだ句」 ...
> #Cultural_point_of_interest?

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	004176f28a17bf45	"\n\nWell, after I asked you to provide the di...	0	0	0	0	0	0
1	0044cf18cc2655b3	What page shoudld there be for important chara...	0	0	0	0	0	0
2	00472b8e2d38d1ea	A pair of jew-hating weiner nazi schmucks.	1	0	1	0	1	1
3	00480b6e1f19601b	I tend to think that when the list is longer t...	0	0	0	0	0	0
4	0048de0c9422f64f	"\n\n What's up with this? \n""If you are a re...	0	0	0	0	0	0

Figure 2.1: The figure illustrates an example from the training data. The column "id" is the unique id of that particular comment. The column "comment_text" contains the comments, and the other six columns contain a binary number indicating whether the comment belongs to that category.

On the other hand, the test set included some non-English comments; see Figure 2.2. We checked for Chinese, Greek, Arabic, and Hindi texts and retrieved 235 comments. This number would have been higher if we had included more languages. Due to the existence of non-English comments, we will also test some multi-lingual solutions.

	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
54	...السلام عليكم و رحمة الله و بركاته الا الجميع ت	0	0	0	0	0	0
546	...لڑکی اور لڑکے میں جومحبت پیدا ہوتی ہے، ہوسکتاہ	0	0	0	0	0	0
724	李白 月下獨酌 中古漢語朗讀 *李白 將進酒 中古漢語朗讀 *李白 靜夜思 中古漢語朗讀 *...	0	0	0	0	0	0
776	...لقد طفح الكيل من جرم وكفر ونهب السفاح الدجال ا	0	0	0	0	0	0
1070	"灌装设备|封口设备|打码设备包装机|包装机械 包装机械 封口机 收缩机|热收缩机|热收缩包...	0	0	0	0	0	0

Figure 2.2: Example of non-English comments in the test set.

2.2 Exploratory Data Analysis

This section explores the dataset and presents related statistics. Please note that most of the statistics and graphs are based on the training data unless otherwise specified. Figure 2.3 shows the frequency of comments based on the number of words. We can see that only a small fraction of the comments exceed 200 words, suggesting that it might be a good idea to set a limit of 200 words for all comments during the training procedure. Although Figure 2.3 suggests that a 200-word limit might be suitable, it is important to consider the possibility that longer comments exceeding this limit could predominantly belong to a specific type of toxic class. This raises concerns that by removing longer comments, we might inadvertently eliminate a significant portion of valuable information related to that specific toxic class.

To address this concern and ensure that our analysis is comprehensive, the box plot in Figure 2.4 demonstrates the word frequencies per category. This allows us to visually examine the distribution of word frequencies across different categories and assess whether there is

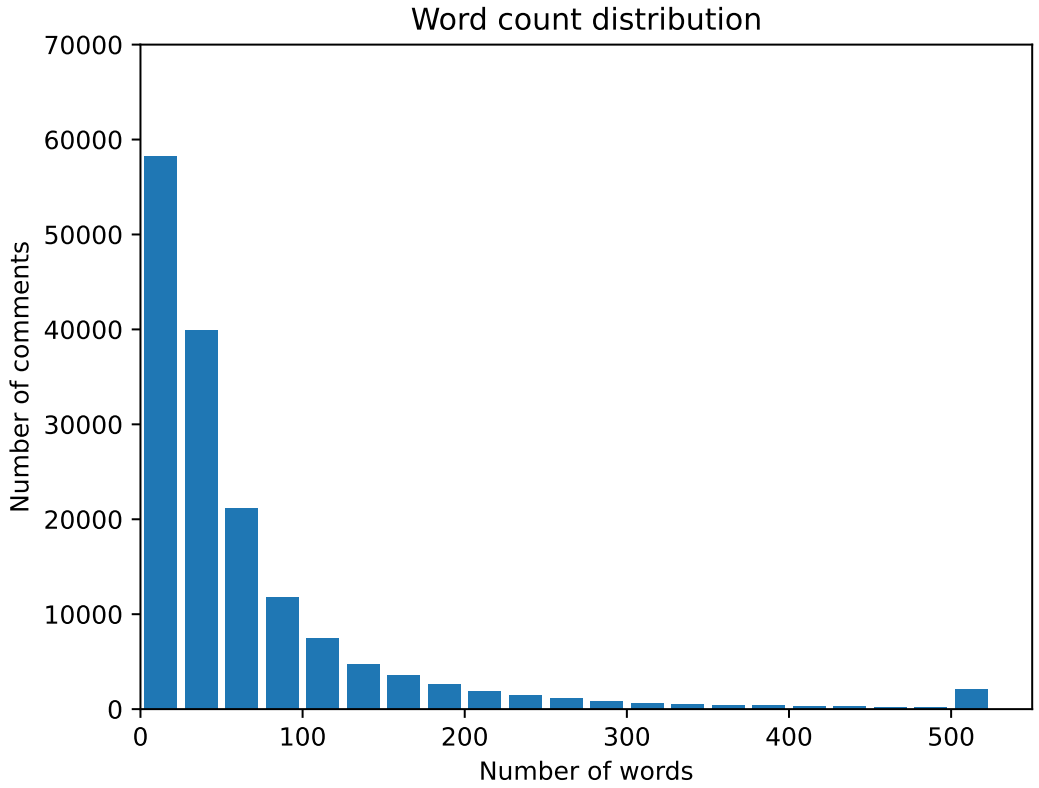

Figure 2.3: Plot of frequency of comments based on the number of words. The width of each bar represents 25 words. For example, the first bar represents how many comments are between 1 to 25 words. Note that the last bar is for comments longer than 500

a significant disparity between longer comments and any particular toxic class. As evident from the figure, we have relatively homogeneous data regarding the number of words in the hateful categories, whereas the not-offensive comments have a distinct difference in word length.

We also investigated the distribution of comments in each category. Table 2.1 shows the descriptive statistics for each class. First, we notice that the number of comments labeled "not offensive" is significantly larger than other categories. Furthermore, the number of comments

Table 2.1: Descriptive data statistics of the dataset. The std, mean, median, min, and max are calculated based on the number of words.

	total	not offensive	toxic	severe toxic	obscene	threat	insult	identity hate
num_comm	159571	143346	15294	1595	8449	478	7877	1405
std	99.2	98.2	106.5	186.1	108.6	137.8	107.7	114
mean	67.3	68.9	51.3	75.6	49.6	55.2	48.3	52
median	36	38	22	17	20	23	21	21
min	1	1	2	2	2	3	2	2
max	1411	1250	1411	1403	1403	1403	1403	1247

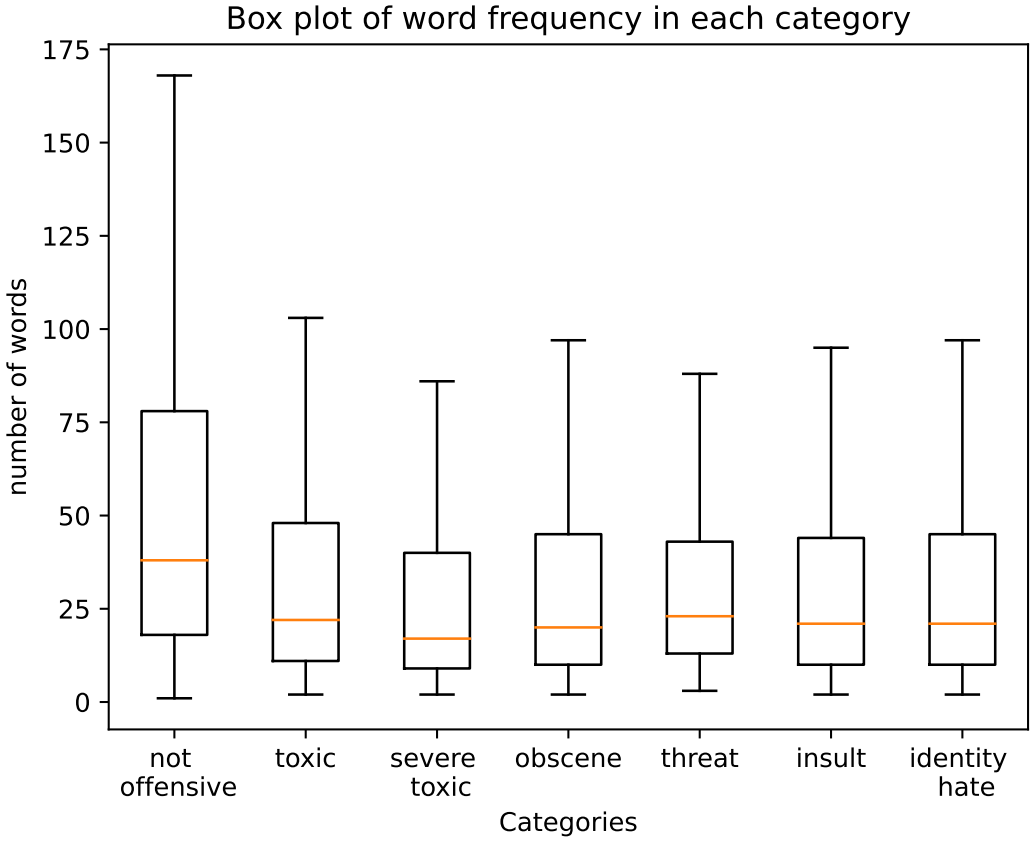

Figure 2.4: This chart displays the distribution of the number of words per category. The orange lines in the middle represent the median word length of each category. The boxes contain 50% of comments in each class. The lower and upper whiskers correspond to the 25% of comments with fewer words and the 25% of comments with more words, respectively.

labeled "threat" is small. Table 2.2 shows the data distribution of the train and test set. In all categories except severe_toxic, a third of the data is reserved for the test set.

To understand the extent of the imbalance of each category compared to the number of non-toxic comments, we plot the number of comments labeled as one against the rest for each of the six categories. As shown in Figure 2.5 for the class threat, there is a significant difference in the number of comments not labeled as "threat" that the bar is barely visible in the plot. The number of samples in the under-represented categories is difficult to see in Figure 2.5. Thus Figure 2.6 illustrates the same bars as Figure 2.5 but without the bars of non-labeled comments.

Table 2.2: Test and train ratio in each category.

%	not_off	toxic	severe_toxic	obscene	threat	insult	identity_hate
Train	71.29	71.52	81.29	69.6	69.38	69.68	66.37
Test	28.71	28.48	18.71	30.4	30.62	30.32	33.63

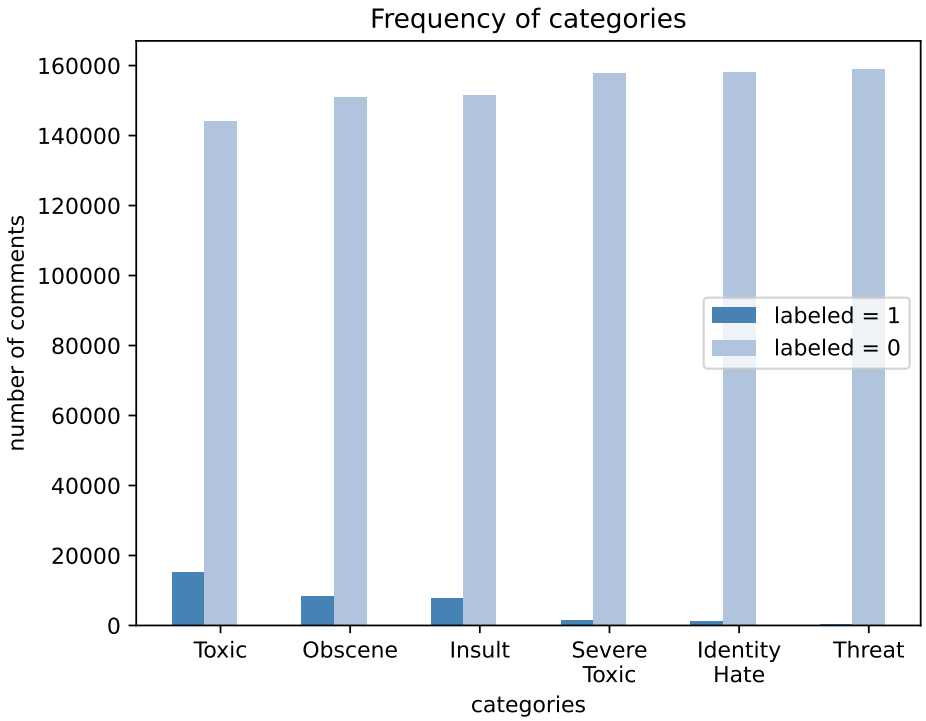

Figure 2.5: Plot of label 1 and 0 in each category. Note that for the category threat, the number of samples labeled as 1 is so small that it is not visible in the plot.

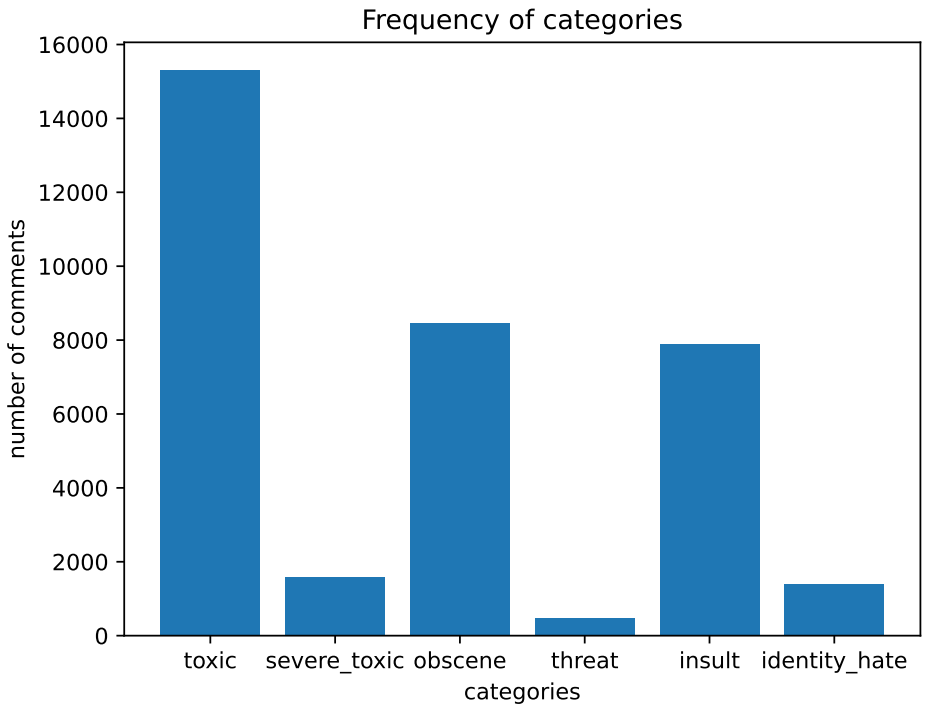

Figure 2.6: Bar plot of the frequency of comments in each category

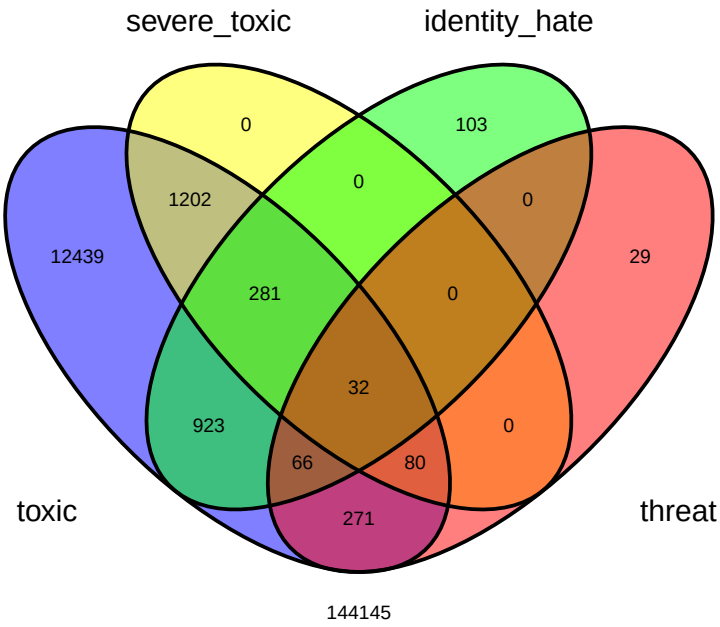

Figure 2.7: Venn diagram of 4 categories. "severe_toxic" is a subset of "toxic". Some comments are only labeled as "identity_hate" or "threat" without belonging to any additional category.

We were interested in the correlation between categories and decided to illustrate the interdependence between categories. A Venn diagram demonstrates the distribution of comments throughout the classes. In Figures 2.7 and 2.8, we illustrate two different Venn diagrams with four categories each. We chose not to illustrate the six categories in the same figure because that would imply more than 40 sections. Thus, it would be hard to read the information. As Figure 2.7 shows, the "severe_toxic" category is a subset of "toxic". Further, some comments are only labeled as "identity_hate" or "threat" without belonging to any additional category. For the set "threat", few comments do not share the intersection with toxic. This may suggest that the category "threat" might also be a subset of toxic, and the comments labeled as "threat" without being labeled as "toxic" were mistakenly annotated. In Figure 2.8, we see numerically more comments only belonging to one category ("obscene" and "insult"). However, since these are categories with many registers in total, they are percentage-wise still relatively few.

Another practical way of visualizing the dependencies between categories is to plot a correlation matrix of the categories. Figure 2.9 shows that the correlation between the "toxic" and "severe_toxic" categories is relatively low even though, as Figure 2.7 demonstrates, all comments labeled "severe_toxic" are also labeled as "toxic". The low correlation is because the toxic category contains considerably more comments not labeled as "severe_toxic" leading to a lower correlation between those two categories. Therefore, Figure 2.8 demonstrates the importance of the size of the *intersection* between two classes compared to those not in that intersection. For instance, most comments labeled as "obscene" and "insult" are found in the intersection between the two. Thus, we see a more significant correlation between the two. The correlations between "toxic" and "obscene" and "toxic" and "insult" are also significant due to their large intersection. The lowest correlation is between "severe_toxic" and "threat", which Figure 2.7 explains since they share very few comments.

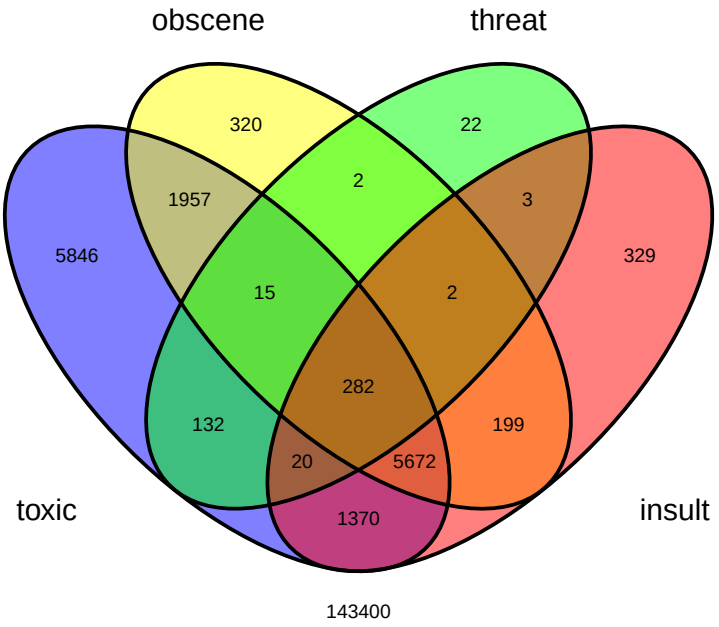

Figure 2.8: Most comments are in the category "toxic". For the set "threat", few comments are not in the intersection with toxic.

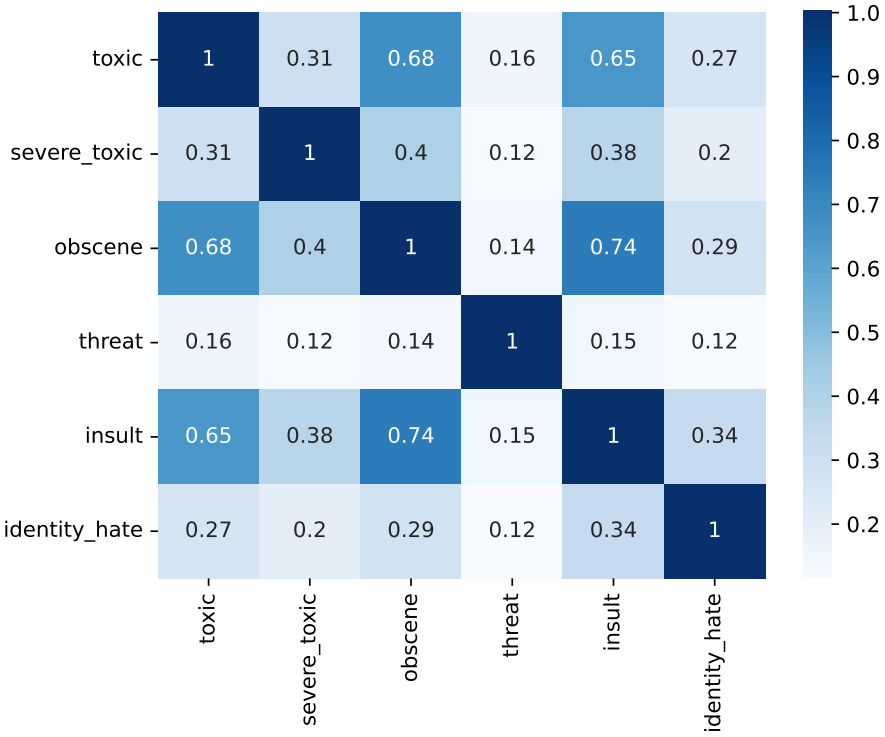

Figure 2.9: The correlation matrix of categories. A number closer to one means a stronger correlation between the categories.

Chapter 3
Approach

This chapter explains the approach to obtaining a model for recognizing and categorizing hate speech. It describes the most basic procedure as well as more complex methods to obtain more precise models. We give a comprehensive explanation of the process, starting from cleaning and tokenization, followed by token vectorization, word embedding, and classification. Ultimately, there is a description of how we evaluated the models to choose the optimal one.

3.1 Data Cleaning

We performed data cleaning on the training data for all procedures except for the transformer, which we explain further in Section 3.4.3. This process stage is *standardizing* the text (Chollet, 2021). In other words, we cleared the data column containing the comment texts from URLs, substituted uppercase letters with lowercase letters, and removed stopwords, i.e., common words or words providing no contextual meaning. We also removed characters that were not letters or numbers. By doing so, the strings on each row were as simple as possible without losing their context. Standardization takes the following comment, part of the training set, from

> "Please stop. If you continue to vandalize Wikipedia, you will be blocked from editing. | Talk If this is a shared IP address, and you didn't make the edit, consider creating an account for yourself so you can avoid further irrelevant notices."

to

> "please stop continue vandalize will blocked editing talk shared ip address didnt make edit consider creating account can avoid irrelevant notices"

The advantage of standardization is that words appear only once, regardless of capitalization. For example, the word "please" appears in lowercase, although we initially have the

word “Please” in the comment above. However, when doing this part of the process, one has to take careful consideration of what the outcome will be. Punctuation can be crucial depending on what kind of text analysis one performs. For example, maintaining question marks is essential when identifying questions in a text, as Chollet (2021) discusses.

3.2 Tokenization

In the next step, we split the text into tokens, transforming the complete strings into units of text. There are several ways to do tokenization, each chosen depending on the task. The most common ones are *word-level tokenization* and *N-gram tokenization*, whereas *character-level tokenization* is less popular (Chollet, 2021). On the word level, each token corresponds to one unit of text separated by white space, and character level intuitively means treating each character as a token. N-gram tokenization treats N number of consecutive words as a token. A subcategory of word tokenization is subword tokenization, where a word can be split up into shorter words. When treating text, either the order of the words matters, or it does not. Chollet refers to these models as either *sequence models* or *bag-of-word* models. The models where the sequence is important use word-level tokenization and the N-gram tokenization when creating a bag-of-words mode. The latter one, due to the loss of structure of the text, is normally not used in deep learning models due to lacking the ability to provide as good predictions as the sequence models.

3.3 Text and Token Vectorization

Before training any model, we needed to transform the sequence of raw text tokens into a feature vector. Different methods are available to achieve this goal. This section discusses bag-of-words methods and word embedding. We explain the concepts behind these methods and provide a methodology for using them.

3.3.1 Bag-of-Words

Bag-of-words is a simple way of representing text, its usage widespread in analyzing sentiments (El-Din, 2016). When using the bag-of-words technique, we kept track of the occurrence of a word within a document without considering the grammar and order. To demonstrate the procedure, let us assume we have the following documents:

D1 : The cat sat on the mat.
D2 : I love my cat.

	the	cat	sat	on	mat	i	love	my
D1	2	1	1	1	1	0	0	0
D2	0	1	0	0	0	1	1	1

Table 3.1: Bag-of-word representation of D1 and D2.

Table 3.1 shows the bag-of-word representation of these two documents. It counts the number of times a certain word occurs in a document. For instance, the word "the" shows up twice in the first document, but not once in the second one. We could improve several things with this representation. To begin with, it is a good practice to normalize the input vectors to the neural network. Another problem is that connecting words will have more occurrences in each document, even if they are not crucial for capturing meaning.

For that reason, a modification of the bag-of-word representation exists. Sparck Jones (1972) introduced the term frequency-inverse document frequency, TF-IDF, which measures the importance of a word in a document relative to all other documents in the corpus. TF stands for term frequency. It counts how many times a specific word appears in the document and divides the count by the total number of words. For example, if we want to calculate the term frequency of the word w in document/comment d we do it according to Equation 3.1.

$$tf(w,d) = \frac{count(w \in d\ )}{\sum_{word \in d}\ count(word)} \tag{3.1}$$

However, term frequency alone is not a good measure of the importance of a word. Words such as "the", "and", "or", "how", "where", and "was" for connecting phrases will have a high term frequency even if they do not have any noticeable importance in the document. We need a method to reduce the significance of these words.

Hence, we introduce Inverse Document Frequency, IDF, which measures the relative rarity of a term in a corpus. IDF is defined as

$$idf(w,D) = \log(\frac{count(D)}{count(D_w)}), \tag{3.2}$$

where D is all documents in the corpus and D_w is all documents containing the word w. Looking at Equation (3.2) we notice that it does not depend on individual documents. Therefore, the IDF of a specific word stays the same for all documents and can be interpreted as a weight for the term frequency. Consequently, words that appear only in one document will have higher importance than words that appear in all documents. Combining TF and IDF gives us TF-IDF, defined as

$$TF - IDF(w,d,D) = tf(w,d) \cdot idf(w,D). \tag{3.3}$$

During the TF-IDF implementation on our data, we utilized scikit-learns TF-IDF vectorizer. Scikit-learn's implementation of TF-IDF slightly differs from the usual definition and is shown in Equation 3.4. Using this version, a word present in all documents will not be completely disregarded and will retain some degree of significance.

$$idf(w,D) = \log(\frac{count(D)+1}{count(D_w)+1}) + 1,) \tag{3.4}$$

Scikit-learn's TF-IDF vectorizer receives a list of documents, referred to as a corpus, and returns a matrix where columns are the number of unique words in the whole corpus and the rows correspond to the number of documents in the corpus. This is the *feature matrix*. In our case, a document was a comment and the corpus was the whole training data. The cleaning described in Section 3.1 was performed on the training comments.

However, there are several drawbacks to the TF-IDF method. First of all, it creates a sparse feature matrix. The resulting feature matrix has dimensions $R^{D \times V}$, where D is the number of documents and V is the number of unique words in all the documents. When dealing with many documents, each containing a high number of unique words, storing the resulting matrix would require considerable memory. There are approximately 289,000 unique words and approximately 160,000 comments in the training set, resulting in a matrix dimension of 159,000 × 289,000. Additionally, the longest comment in the training set is 5,000 words long, meaning that, at most, only 5,000 elements in each row are non-zero, making the matrix very sparse.

Second, this method is too simplistic and captures neither context nor semantics of a sentence. For example, words like "tea" and "coffee" will get completely different vectors even though they are highly related.

Due to these limitations, we needed to try more complex vectorization methods, leading to word embedding.

3.3.2 Word Embedding

A better way of representing words in documents is to use word embedding. Word embedding is mapping each word to a real-valued fixed-sized vector. These vectors are learned representations of the words such that they can encapsulate some properties of the word. In contrast to the bag-of-words method, which results in a sparse high-dimensional one-hot encoded vector, the result of word embeddings is dense and of a much smaller dimension (Chollet, 2021). By using word embeddings, similar words will have a similar vector representation. With similar words, we mean words that share some properties. In Figure 3.1, eight

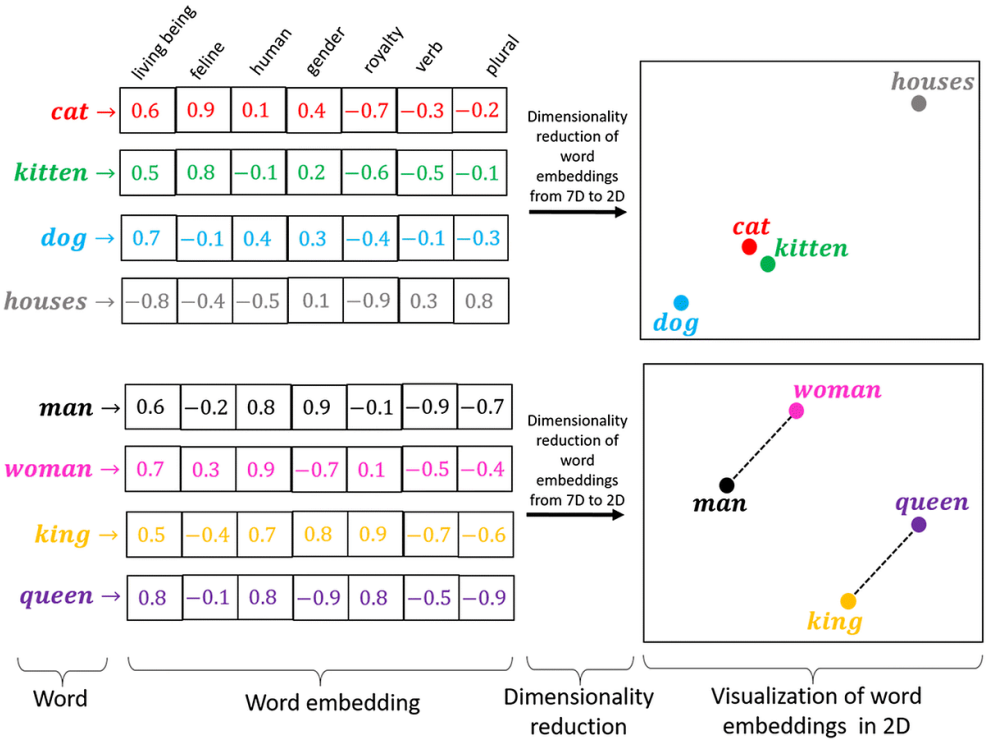

Figure 3.1: Word embedding representation (Gautam, 2020)

different words and their mapping to a seven-dimensional vector are illustrated. The words "man" and "woman" are diverse when it comes to the feature "gender"; 0.9 in comparison to −0.7, suggesting that the number 1 refers to a word of strong masculinity and −1 strong femininity. However, they are barely associated with "gender", resulting in values close to 0. However, the words "king" and "queen" are strongly connected to royalty and their respective gender.

The rest of this subsection addresses the word embedding techniques used in this thesis, starting with fastText and followed by GloVe.

FastText

FastText is widely regarded as one of the most popular word embedding techniques because it can overcome critical limitations in other methods, such as word2vec introduced by Mikolov et al. (2013). While word2vec ignores the internal structure of words, i.e. the combination and arrangement of characters in words, FastText considers these internal structures, making it a more powerful and effective approach for word embedding.

Before we delve into the workings of fast Text, let us introduce the two primary architectures used to train word embeddings in fastText: continuous bag-of-words (CBOW) and Skip-gram. Both techniques consider a certain number of consecutive words at a time in a document. The number of words to consider as context for the current word is determined by a context window in both CBOW and Skip-gram models.

For the CBOW model, the context words are the input, and we try to predict the current word. Skip-gram model is the opposite of CBOW, where the input is the current word, and we predict the context. Both techniques are illustrated in Figure 3.2.

It is worth noting that CBOW and Skip-gram models are used in training word2vec

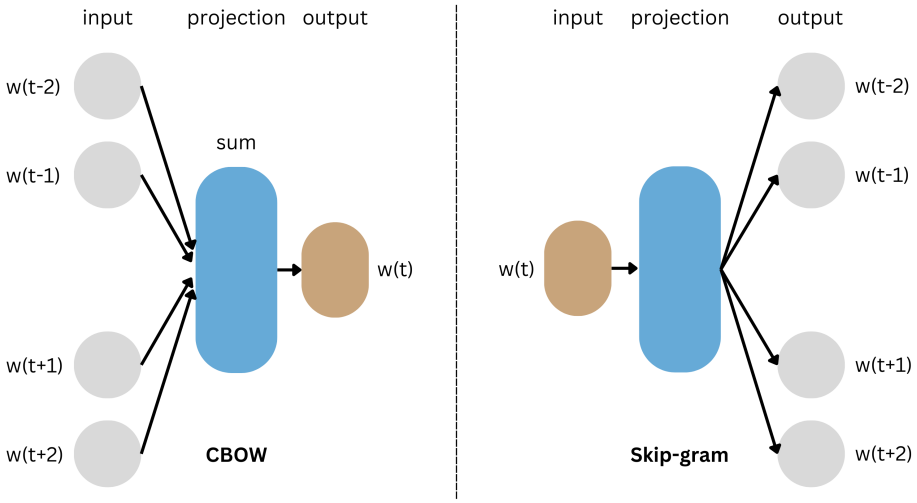

Figure 3.2: This illustration was inspired by the work of Mikolov et al. (2013) and depicts the CBOW and Skip-gram models. In the figure, $w(t)$ is the current word and $w(t-2), w(t-1), w(t+1), w(t+2)$ are the context words. The window size in this case is 2. The word "projection" in the figure refers to the usual vector-matrix multiplication.

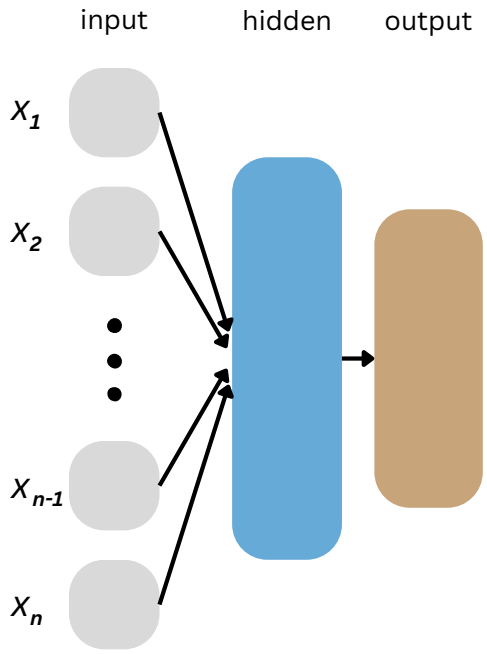

Figure 3.3: Illustration of CBOW for fast Text. Character N-grams features, $x_1, x_2, \cdots, xn$ of context words are input, the hidden layer contains the representation of words, and outputs are probabilities of each word in vocabulary being current words.

and fastText. However, one of the differences is that the input of word2vec is words, while fastText utilizes a combination of character n-grams features as input. It improves the vector representations of words by handling both words and sub-parts of words with the character n-grams.

Figure 3.3 illustrates an architecture similar to CBOW where the inputs are the N-gram features of text, and the outputs are the probabilities of each word being the current word. The softmax function, f in equation 3.5 calculates the probabilities.

The loss function for this architecture is the negative log-likelihood, defined as

$$-\frac{1}{N}\sum_{n=1}^{N} y_n \log(f(BAx_n)), \tag{3.5}$$

where N is the number of documents, i.e., training inputs, and y_n is the target. Matrices B and A are weights, and x_n is the normalized bag of features (Joulin et al., 2016).

FastText is a highly efficient algorithm because it utilizes hierarchical softmax instead of the traditional softmax. This choice enables faster computation and makes the algorithm time-efficient in processing large amounts of text data. The hierarchical softmax is especially useful when the architecture has many output classes.

Another advantage of fastText word embedding is that it can represent rare words since some of their n-gram properties are similar to common words. FastText uses n-gram information when creating word representations, allowing for embeddings for out-of-vocabulary words (words that have never existed in the training corpus).

GloVe

One of the drawbacks of models like fastText and word2vec is that their context is defined locally by their context window. Thus, they cannot capture contextual relations between words that are too far apart but are still semantically connected.

In 2014, Pennington et al. addressed this issue by developing a new word embedding technique called GloVe. GloVe stands for global vectors and is a type of word embedding that takes advantage of both local and global properties of words. A context window captures the local properties. The local properties refer to the relationship between a target and current words. A word-to-word co-occurrence matrix captures the global properties. Let $\boldsymbol{X}$ denote the co-occurrence matrix. $\boldsymbol{X_{ij}}$ is the number of times the word $\boldsymbol{j}$ occurs in the context of the word $\boldsymbol{i}$. Further, let $\boldsymbol{X_i} = \sum_k \boldsymbol{X_{ik}}$ be the number of times any word appears in the context of the word $\boldsymbol{i}$. Then the co-occurrence probabilities are defined as

$$p(i|j) = \frac{X_{ij}}{X_i} \tag{3.6}$$

The GloVe paper, published by Pennington et al. (2014), includes a simple example demonstrating how co-occurrence probabilities cluster contextually similar words. The example in Table 3.2 shows how co-occurrence probabilities retrieve certain semantic aspects. For words that frequently appear as context for both target words, like the word "water", the ratio $\boldsymbol{P(k|ice)/P(k|steam)}$ is close to one. The same goes for words that rarely appear as context for the selected target words, like the context word "fashion". Further, we notice that if the probability ratio is close to zero, the context word is semantically related to the target word "steam" and if it is larger than one, the context word correlates to the "ice".

Probability and Ratio	**k = solid**	**k = gas**	**k = water**	**k = fashion**
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Table 3.2: The simple example from Pennington et al. (2014) shows co-occurrence probabilities of target words "ice" and "steam" for selected context words.

The probability ratio is a better way to separate relevant and irrelevant words. It is also better for discriminating the two relevant words from each other (Pennington et al., 2014). Therefore the ratio of co-occurrence probabilities is a good starting point for word vectors in GloVe representation. Thus the most general model for word vectors can be formulated as

$$\boldsymbol{F(w_i, w_j, \tilde{w}_k) = P_{ik}/P_{jk}}, \tag{3.7}$$

where $\boldsymbol{w_i, w_j} \in \mathbb{R}^d$ are target words and $\boldsymbol{\tilde{w}} \in \mathbb{R}^d$ are separated context words. Starting from Equation 3.7, We can derive the following cost function:

$$J = \sum_{i,j}^{v} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}j - log(X_{ij}))^2, \tag{3.8}$$

where v is the size of vocabulary, $f(X_{ij})$ is a weighting term to scale down stop words, w_i is i:th word and $\tilde{w}_j$ is j:th context word. The terms b_i and $\tilde{b}_j$ are bias terms, and lastly, X_{ij} is the

number of times the word j appeared in the context of the word i. We skip the derivation of these cost functions and refer the curious reader to the paper Pennington et al. (2014). Equation 3.8 can be minimized using optimization methods like stochastic gradient descent or the Adam method to obtain word vectors. According to Dharma et al. (2022), GloVe can capture semantic and syntactic aspects of words.

To summarize the section about word embeddings, fastText, and GloVe are models that map words into vector spaces, capturing semantic relationships between words. However, they do not consider the contextual meaning of words. Section 3.4.3 introduces an improvement.

3.4 Model Architectures

We used logistic regression, neural network variants, and transformers as model architectures. Creating a baseline using logistic regression was our initial approach. We did this for comparative purposes and to help us move on to more complex models.

Neural networks are another approach to categorizing text. The two main types of neural networks are convolutional neural networks (CNNs) and recurrent neural networks (RNNs). According to Yin et al. (2017), hierarchical CNN architecture is traditionally preferred for extracting position-invariant features. However, when considering sequential architectures, RNN performs better on average. The sequential architecture of RNN makes it a preferred choice in NLP, and two further developed methods based on RNN have gained significant popularity. The long short-term memory (LSTM) was first introduced by Hochreiter and Schmidhuber (1997), and gated recurrent unit (GRU) by Cho et al. (2014). Section 3.4.2 explains the bidirectional gated recurrent unit in more depth.

The transformer architecture uses an altered version of neural network architecture. Vaswani et al. (2017) argues that transformer models can replace traditional deep neural networks in NLP since they can capture long-term dependencies.

3.4.1 Logistic Regression

In logistic regression, we aim to predict the probabilities that a given sample belongs to a class. The likelihood $\hat{y}_i$ that sample i belongs to class 1 is given by

$$\hat{y}_i = f(z_i), \tag{3.9}$$

where

$$f(x) = \frac{e^x}{1 + e^x}, \tag{3.10}$$

is called the logistic function z_i is:

$$z_i = \boldsymbol{x_i}\boldsymbol{\omega} = \sum_{k=0}^{m} x_{ik}\omega_i. \tag{3.11}$$

The vector $\boldsymbol{x_i} = (1, x_{i1}, \cdots, x_{im})$ are the features for sample $\boldsymbol{i}$. If we use TF-IDF, the elements $x_{i1}, \cdots, x_{im}$ correspond to the TF-IDF vector of size $\boldsymbol{m}$ representing comment $\boldsymbol{i}$. The first element, 1, in the vector, represents the bias, a constant value. The elements in

the vector $\boldsymbol{\omega} = (\omega_0, \omega_1, ...\omega_m)$ are weights of the network that will be adjusted during the training phase.

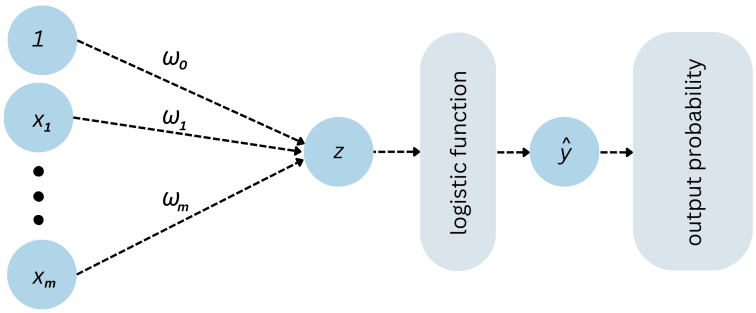

Figure 3.4: Illustration of logistic regression. The value of $\hat{y}$ is the output probability of a given sample.

Figure 3.4 illustrates Equations 3.10 and 3.11 as a neural network. The input vector in Figure 3.4 is the final output from the feature extraction step explained in Section 3.3.2. We can regard Equation 3.10 as an activation function of a simple feed-forward neural network with $m + 1$ input nodes and one output node. The probability output predicts the label as zero or one, using a threshold of 0.5. Ultimately, we compare the predicted label against the true label of the sample. In this procedure, individual models are created to predict whether a sample belongs to a specific class or not. Consequently, the process is repeated for each class. For instance, when we employ logistic regression, six models are generated, each dedicated to predicting one of the six classes.

3.4.2 Bidirectional Gated Recurrent Unit

Unlike the simple neural network depicted in Figure 3.4, the recurrent neural network architecture can capture temporal dependencies through feedback connections. Figure 3.5 provides a visual representation of a recurrent neural network, where the arrows pointing to the left or looping back to the same node signify the recurrent behavior inherent in the network. The six output nodes in the illustration signify that six categories can be predicted, as in the case of this thesis.

Figure 3.5 illustrates the fundamental concept of a recurrent neural network. In our thesis, we specifically directed our attention to the GRU, which belongs to the subgroup of RNNs and represents an enhanced version of the conventional RNN. By examining a single GRU unit, we can gain valuable insights into its functionality. This unit comprises two crucial gates, namely the update gate and the reset gate, as discussed in the research conducted by Cho et al. (2014).

The reset gate determines how much information from the previous hidden state can be discarded or forgotten. It takes the sum of the previous hidden state and the input multiplied by their corresponding weights and passes it through a sigmoid function to produce a value between 0 and 1. If the value is close to 0, the current state is forced to ignore the previous states. The reset gate of the jth GRU unit is

$$r_j = \sigma([\boldsymbol{W}^r \boldsymbol{x}]_j + [\boldsymbol{U}^r \boldsymbol{h}_{t-1}]_j) \tag{3.12}$$

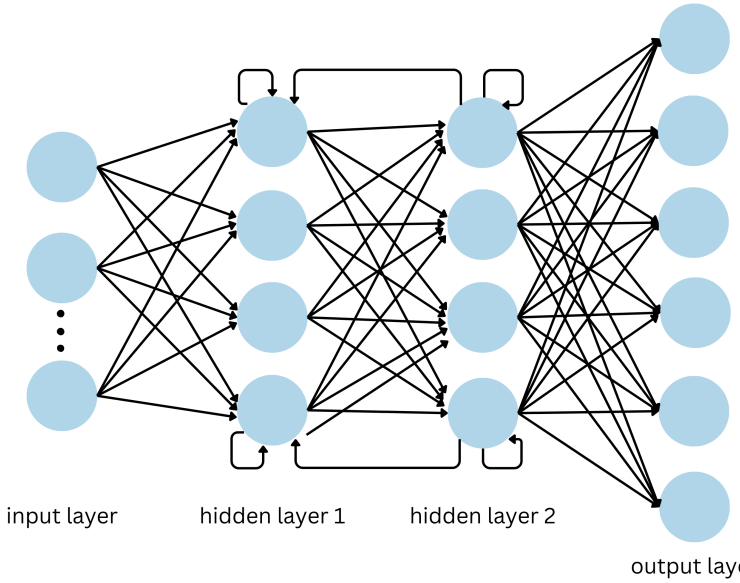

Figure 3.5: An illustration of a recurrent neural network. The arrows going backward in layers or returning to the same layer demonstrate the feedback connections. In the illustration, there are six output nodes, demonstrating the six categories in this thesis.

where $\boldsymbol{x}$ is the input, $\boldsymbol{h_{t-1}}$ is the previous state , $\boldsymbol{W^r}$ and $\boldsymbol{U^r}$ are the weights of input respective previous state. Note that $[.]_j$ means the jth element of the vector.

The update gate will decide how much of the previous state's information will be carried to the current state. It has an equation similar to the reset gate, defined as

$$z_j = \sigma([\boldsymbol{W^z x}]_j + [\boldsymbol{U^z h_{t-1}}]_j). \tag{3.13}$$

Using Equations 3.12 and 3.13, we can write the update formula for the current hidden state as

$$h_j^t = z_j h_j^{(t-1)} + (1 - z_j)\tilde{h}_j^{(t)}, \tag{3.14}$$

where $\tilde{h}_j^{(t)}$ is

$$\tilde{h}_j^{(t)} = \tanh([\boldsymbol{W x}]_j + [\boldsymbol{U}(\boldsymbol{r} \odot h_{t-1})]_j]) \tag{3.15}$$

and $\boldsymbol{W}$ and $\boldsymbol{U}$ are weights and the symbol $\odot$ denotes Hadamard product. Figure 3.6 illustrates the BiGRU mathematics explained above.

A Bidirectional GRU is formed by utilizing two GRU units, one for processing data in the forward direction and the other for processing data backward. We can create a Bidirectional GRU layer by stacking multiple such units together.

3.4.3 Transformers

Vaswani et al. introduced the transformer architecture in 2017. Since then it has become one of the most popular architectures for NLP tasks such as machine translation, text classification, and question-answering. The module 'Transformers' on Huggingface has pre-trained

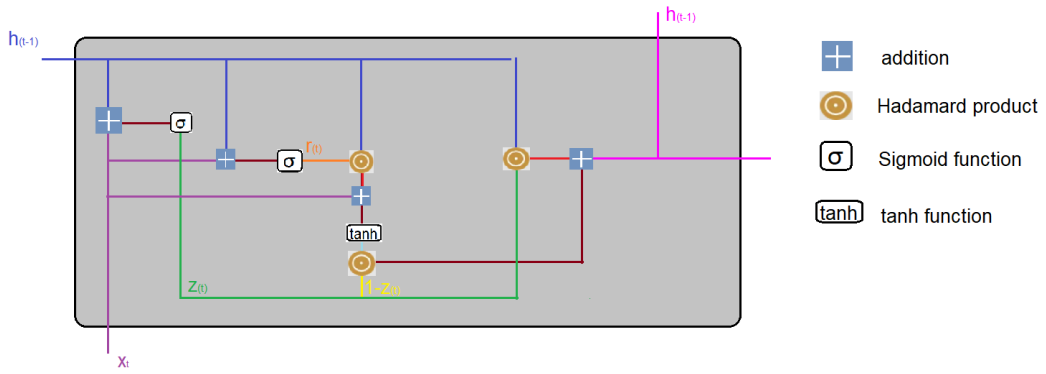

Figure 3.6: Illustration of the gated recurrent unit.

transformer tokenizers and pre-trained models. The transformer library offers these advances to the machine learning community, providing a unified API and a collection of pre-trained models. The Huggingface team designed the transformer library to follow the standard architecture of NLP machine learning models, which involves data processing, model application, and outcome prediction. The transformer architecture steps are more intertwined than the previous methods described. Thus, this section explains the transformer process.

The transformer model architecture is another type of neural network structure. In contrast to traditional neural networks, however, it manages to model long-range dependencies (Vaswani et al., 2017). The self-attention in the transformer model allows for capturing tokens' local and global dependencies in a sequence. The transformer architecture allows for bigger models, and the pre-trained tokenizers and models enable their use in various tasks. In the transformer process, the encoding phase consists of tokenization, encoding, and the self-attention mechanism. The tight interdependence of these steps is the motivation for including a dedicated subsection that describes the transformer process as a whole. Figure 3.7 gives an overview of the whole process. During encoding, the raw input text is converted into a sequence of numerical representations. The key innovation of transformers is the attention mechanism, which allows the model to focus on different parts of the input sequence at different times, instead of processing the entire sequence at once. This allows transformers to handle sequences of varying lengths, making them well-suited for NLP tasks.

In comparison to many other NLP architectures, when using transformers there is no need to remove stop words, punctuation, or stem words. On the contrary, removing stop words or punctuation can harm the classification task since it removes part of the context. Stop words and punctuation have their encoding number in the pre-built architectures, thus increasing the probability that a comment is correctly classified. However, before initializing the encoding phase, the comments had to be shortened. Lengths between 50 and 300 were tried, discussed more in Chapter 4. The number of words in a comment greatly influences the extraction of the hidden states in the feature extraction step. This is due to the parallel procedure when using the transformer method, for which batch size needs to be optimized.

The pre-trained tokenizer splits the raw input text into a sequence of individual subword tokens and directly assigned an embedding, where sequence matters. Subword tokenization is beneficial due to a decrease in vocabulary size without losing information, as discussed in Section 3.2. Each token created is assigned a unique index to indicate its position in the

input sequence. We refer to this as positional encoding, which allows the model to distinguish between tokens based on their position.

The tokenization methods we employ in this thesis use different techniques. The transformer-based models BERT and distilBERT use a tokenization technique called WordPiece. We analyze the frequency and mutual information of subword sequences in the training data to break down words into smaller units. We begin by treating each word as a sequence of characters and merging the most common character sequences to form subword tokens. We merge until we reach a predetermined vocabulary size or the maximum number of merge operations. Wordpiece tokenization allows for handling both known and unknown subwords and allows for a more flexible vocabulary.

The RoBERTa model, on the other hand, uses a tokenization technique called byte-pair encoding (BPE) (Liu et al., 2019). The technique applies byte-pair encoding to represent words as a combination of subword tokens by breaking down words into subword units based on their frequency in a corpus.

To demonstrate the tokenization and encoding process, we will look at the example where 'distilbert-base-uncased' has been used as a tokenizer. The string

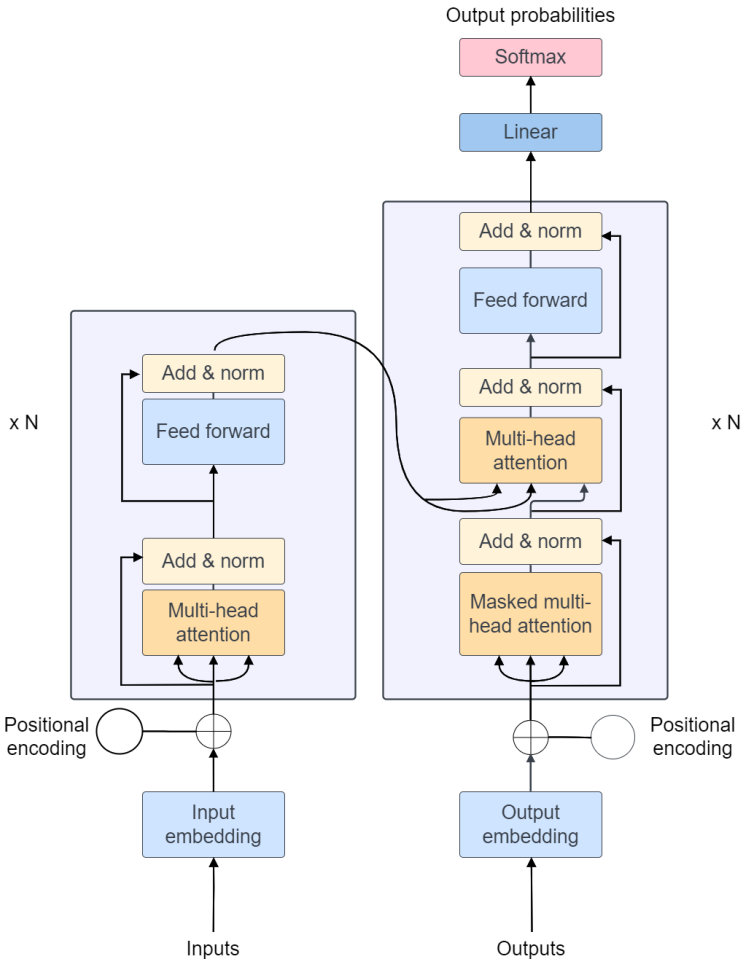

Figure 3.7: Transformer model architecture. The illustration is based on a diagram by Vaswani et al. (2017), and created for this thesis using Visual Paradigm online.

"Quantum entanglement is a strange phenomenon."

returns a dictionary with two keys; 'input_ids' and 'attention_mask' with values

$$\text{input\_ids: } \begin{bmatrix} 101 & 8559 & 4372 & 23395 & 3672 & 2003 & 1037 & 4326 & 9575 & 102 \end{bmatrix}$$
$$\text{attention\_mask: } \begin{bmatrix} [1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1] \end{bmatrix}$$

accordingly. They represent positional encoding. If converted to tokens, the encoded sentence becomes

['[CLS]', 'quantum', 'en', '##tangle', '##ment', 'is', 'a', 'strange', 'phenomenon', '.', '[SEP]']

Ultimately, if wished for, it can become a string again using a tokens-to-sting function within the pre-trained tokenizer of choice. That returns

[CLS] Quantum entanglement is a strange phenomenon. [SEP],

which is the original sentence together with the seemingly unnecessary, however very important, tokens [CLS] and [SEP]. When using a transformer tokenizer, the start and end tokens are always added and play an important role, they mark the start and the end of a sequence.

The Transformer architecture is composed of multiple layers, each outputing a hidden state with dimensions (batch size, sequence length, hidden size), where the hidden size typically corresponds to an embedding dimension of 768. These hidden states are corresponding to the hidden representation of the tokens. The final layer of the Transformer yields the last hidden state, which captures the hidden representations after all the tokens have been processed by the Transformer layers. Although the last hidden state can be utilized for classification tasks, employing it directly can be computationally intensive. Therefore, it becomes necessary to reduce the dimensionality of the last hidden state tensor.

Numerous methods can be employed for dimensionality reduction. For instance, one approach involves using average pooling or another pooling technique. However, a commonly used method is to solely rely on the hidden representation of the CLS token (Tunstall et al., 2022). Because transformer models are based on a self-attention mechanism, the hidden vector of the CLS token serves as an effective representation of the entire sequence. Self-attention empowers the model to encode information from other tokens in the sequence without being constrained by their positions. Through the use of three learnable weight matrices, the self-attention mechanism calculates a weighted sum of token embeddings based on their contextual relevance.

When constructing a model during the training process, it is essential to use the corresponding tokenizer and transformer model. For example, using the 'bert-base-uncased' tokenizer one should use the 'bert-base-uncased' model. This is because different tokenizers might use different positional encodings for the tokens and different transformer models might have different ways of tokenizing words. Various pre-trained transformer models are available, and Chapter 4 provides an overview of the models used in this thesis. To exemplify the concept of pre-trained models, let us explain the BERT model introduced by Devlin et al. (2018). They trained the model using masked language modeling (MLM), meaning that the

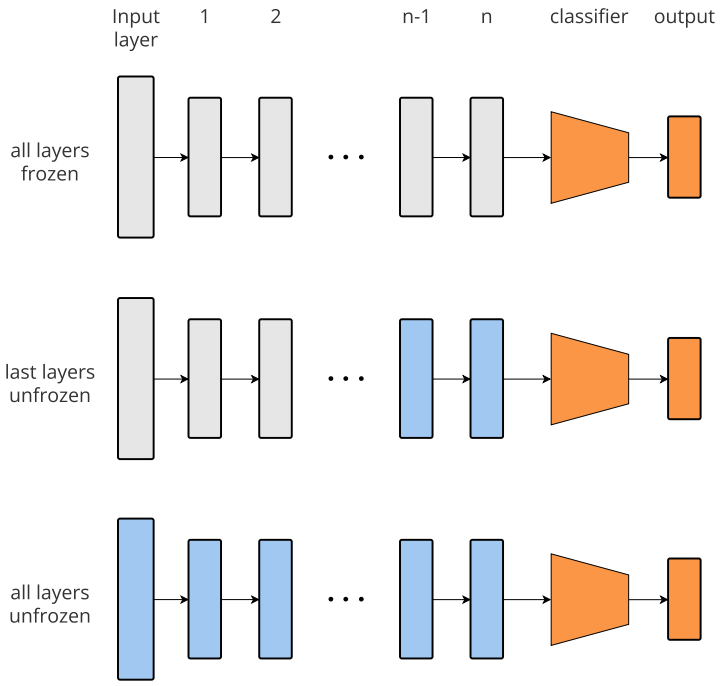

Figure 3.8: Three transformer models techniques. Top architecture illustrates all layers frozen, the middle one when the last layers have been unfrozen, and the last where all layers are unfrozen. The more unfrozen layers, the more training is redone. Illustration made using Visual Paradigm online.

training was performed by hiding one word and, based on the surrounding words, letting the model predict the hidden word. This technique allows for capturing both left and right context, thus resulting in better models. The other pre-trained BERT models, such as RoBERTa and distilBERT, are trained using the same MLM technique. Leveraging these pre-trained transformer models is highly advantageous due to the concept of transfer learning. Transfer learning enables using pre-trained models' syntactic and semantic knowledge to enhance the training process on new tasks with limited labeled data. We can still achieve promising results by fine-tuning the pre-trained models, even with a small amount of task-specific data.

The number of the pre-trained model's hidden layers differs depending on the chosen transformer type. The 'distilbert-base-uncased' has six layers, whereas both "bert-base-cased" and "roberta-base" have twelve layers. These layers can be treated differently during the training process. Figure 3.8 the transformer-based model architecture. This model can be considered as a transfer learning where the body is the transformer and its corresponding tokenizer and the head is the classifier which usually is a deep neural network. The figure displays three ways to treat the hidden layers. Either, we keep all layers frozen, implying that only classifier weights are trained and the weights of transformers remain untrained. This will help us to establish a baseline for the transformer model, and to check whether fine-tuning the model will have any effect. By keeping all layers frozen, less training is done, but there is the risk that the model does not adapt as well to the data, in this case, toxic speech data. The second option is to unfreeze several layers, starting backward. This way, we

unfreeze a chosen number of layers and expect improved performance since the model can capture more data details. Lastly, all layers could be unfrozen, meaning that all weights of the models are trained.

3.5 Model Evaluation

To assess the performance of the model, we relied on the mean macro average F1 score and the ROC AUC score. While the former is the most crucial metric for our purposes, the latter is a supplementary tool for comparing results.

3.5.1 F1 Score

The evaluation of the model's performance on the test set involved predicting the labels of the test comments and comparing them with the actual labels. While accuracy, recall, and precision were computed in this comparison, the macro avrage F1 score was deemed the most interesting metric. The accuracy measure is a conventional evaluation technique, but it may provide misleading results when working with an imbalanced data set. The F1 score is calculated according to

$$\text{F1 score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}, \tag{3.16}$$

where

$$\begin{aligned} \text{precision} &= \frac{TP}{TP + FP} \\ \text{recall (sensitivity)} &= \frac{TP}{TP + FN}. \end{aligned} \tag{3.17}$$

The values TP, FN, and FP come from the confusion matrix in Figure 3.9. The acronyms are true positive, false negative, false positive, and true negative. A comment correctly labeled as one (hateful) is a true positive. If wrongly predicted as one, it is a false positive. Likewise, a comment correctly labeled as zero (non-hateful) is a true negative, whereas if the actual comment is hateful but predicted as a zero, it is a false negative.

The F1 score was evaluated using the macro-average technique. The macro-average technique gives equal weight to all categories, regardless of label balance, and provides a more reliable measure of overall performance. For a two-class problem, the macro-average F1 score is the average of the F1 scores for both classes. On the other hand, the weighted F1 score considers the data ratio. It may give a false sense of security if the data is highly imbalanced, similar to accuracy.

The ultimate aim is to calculate the mean macro average F1 score, a general comparison method. Specifically, the mean of the macro average F1 score per class is computed to provide a single value per model.

3.5.2 ROC AUC Score

Another commonly used evaluation metric for classification is the ROC AUC score. ROC is short for the receiver operating characteristics. Fawcett (2006) states that ROC visually

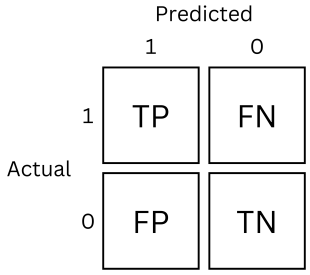

Figure 3.9: Confusion matrix. A comment predicted with label 1 (hateful) with an actual label one gets classified as a TP. Similarly, if predicted as hateful without actually being hateful, it is called a false positive, etc.

represents how well a binary classifier performs. Before explaining how it works, let us define two metrics. True positive rate, or sensitivity, is another name for recall (Equation 3.17). The true negative rate, or specificity, is defined as

$$\text{specificity (true negative rate)} = \frac{TN}{TN + FP}. \tag{3.18}$$

Moreover, we define the false positive rate as

$$\text{false positive rate} = 1 - \textbf{specificity} = \frac{FP}{TN + FP}. \tag{3.19}$$

The model's output is the probability of a comment belonging to each category; these probabilities are real values between 0 and 1. To make the output binary, we need to choose a threshold. Typically a threshold of 0.5 is chosen. We obtain a ROC curve if we plot the true positive rate against the false positive rate for different cutoff values. Hence, if a model is predicting correctly and the predictions are close to their true label, The ROC curve will be drawn to the upper left corner, and thus the area under the ROC curve will be close to one. Figure 3.10 illustrates two curves, where the one closer to the upper left corner gives a better result than the one below. A curve along the diagonal represents a classifier randomly categorizing. Thus a standard threshold to set is a number above 0.5.

The area under the ROC curve (AUC) condenses the classifier's performance into a single scalar. The AUC score allows for easy comparison between different classifiers and informed decision-making for specific tasks. As Fawcett (2006) points out, the ROC AUC score offers an advantage over metrics like accuracy when dealing with imbalanced data sets. Since we have six categories, we will get six ROC AUC scores. Our performance metric will be the mean of ROC AUC of these six categories.

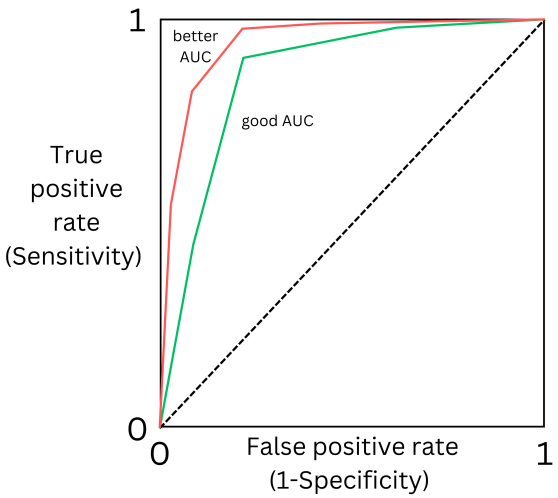

Figure 3.10: The ROC AUC is calculated as the area under the curve. The closer the area is to one, the better the performance.

Chapter 4

Experimental Setup

Deciding on the best model architecture beforehand was a complex task due to the diverse characteristics of the dataset and the wide range of possibilities and combinations of different parameters. Therefore, we wanted to investigate and compare numerous model architectures, as well as parameters, to determine the most suitable approach for the given dataset. Ultimately, we tried countless experimental setups. This chapter accounts for the three main setups used, discussed in Sections 4.1 - 4.3, and their results are discussed in Chapter 5.

4.1 Baseline

Before exploring more complex models, we established a baseline for comparison. The baseline served as a benchmark in evaluating future, more elaborate models. A multilabel classification problem can be decomposed into several binary classifications and this is what we did for creating the baseline.

The initial setup consisted of six binary models, with each model specifically designed for one category. These models were built by combining two components: TF-IDF, which was discussed in Section 3.3, and logistic regression, a binary classifier explained in Section 3.4.

Starting the experimental setup with a baseline model was crucial for the project. We needed to establish a basic model to improve upon using more complex models. No hyperparameter tuning was performed in the baseline architecture since neither TF-IDF nor logistic regression has any parameters to tune. We used sci-kit-learn's built-in function for logistic regression.

4.2 Recurrent Neural Network

After the initial baseline approach, we constructed more complex experimental setups using recurrent neural networks. Table 4.1 shows some attempted setups.

feature extraction	nodes BiGRU	dropout BiGRU	nodes dense	dropout dense	learning rate	num trainable parameters
GloVe50	16	0	[6]	[0]	0.05	6,726
GloVe50	32	0	[6]	[0]	0.05	16,518
GloVe50	64	0.2	[6]	[0]	0.05	45,318
GloVe50	128	0.1	[6]	[0]	0.05	139,782
GloVe50	128	0.1	[16, 6]	[0, 0]	0.05	142,454
GloVe50	128	0	[32, 6]	[0.2, 0]	0.05	146,662
GloVe50	128	0.1	[64, 6]	[0, 0]	0.05	155,078
GloVe50 (200 maxl.)	128	0.2	[32, 6]	[0.1, 0]	0.001	146,662
GloVe50	128	0.2	[32, 6]	[0.1, 0]	0.01	146,662
GloVe50	128	0.2	[32, 6]	[0.1, 0]	0.001	146,662
GloVe100	64	0.2	[6]	[0]	0.005	64,518
ft300	128	0.2	[32, 6]	[0.1, 0]	0.001	338,662
ft300 + GloVe300	64	0.2	[6]	[0]	0.005	256,518
ft300 + GloVe300 (svd)	64	0.2	[6]	[0]	0.005	141,318
ft300 + GloVe300	128	0.2	[32, 6]	[0.1, 0]	0.001	569,062
ft300 + GloVe300 (svd)	128	0.2	[32, 6]	[0.1, 0]	0.001	338,662

Table 4.1: A selection of the RNN architectures tried.

Ultimately, the BiGRU architecture discussed in Section 3.4.2 performed better than the LSTM architecture. Hence it is the model architecture of interest. Table 4.1 shows a large variety of word embeddings and neural network structures. As word embeddings, we used either only fastText, only GloVe, or a combination of these. A combination refers to concatenating the two embedding matrices. The two methods have different strengths. Thus, we attempted concatenation to capture the semantics better. A combination will hypothetically increase the understanding of the context and create a better final model. However, the drawback of such an embedding matrix is the size — a large embedding matrix results in a considerably larger input to the model. Thus a more complex model is created, requiring more training time. To reduce the dimension of the word embeddings, thus reducing the training time, the single-value decomposition technique was tried and is denoted by (svd) in the table.

The bidirectional gated recurrent was used in most of the RNN architectures. The architecture was a BiGRU-layer of 16-128 nodes and then one or two dense layers. The size of the first one varied between 16-64, whereas the last one was always six due to the number of categories. The first dense layer had a RelU activation function, whereas the second had a sigmoid activation function to output probabilities for each of the six classes. Using the ADAM optimizer, we compared different learning rates according to Table 4.1. Normalization was also attempted, as well as varying the maximum length of the comments and the batch size when training the models.

model	truncation	layers unfrozen	num trainable parameters
distilBERT	50	0/7	0.6M
distilBERT	120	0/7	0.6M
distilBERT	200	0/7	0.6M
distilBERT	300	0/7	0.6M
distilBERT	120	4/7	28.9M
distilBERT	120	7/7	67.0M
RoBERTa	120	0/13	0.6M
RoBERTa	120	4/13	29.5M
RoBERTa	120	11/13	79.2M
RoBERTa	120	13/13	125.2M
Twitter-RoBERTa	120	0/13	0.6M
Twitter-RoBERTa	120	10/13	71.5M
Twitter-RoBERTa	120	13/13	125.2M
m-distilBERT	120	7/7	135.3M
m-BERT	200	0/13	0.6M
m-BERT	200	13/13	178.4M
m-BERT	120	13/13	178.4M
xlm-RoBERTa	120	0/13	0.6M
xlm-RoBERTa	120	4/13	28.9M
xlm-RoBERTa	120	13/13	278.6M

Table 4.2: A selection of the transformer architectures tried. The columns "num trainable parameters" shows the number of parameters/weigths that will be tuned during training.

The combinations seen in Table 4.1 are only a fraction of the combinations tried, but the ones resulting in the best results.

4.3 Transformers

The transformer setup was the last architecture to try. Numerous pre-trained models, such as distilBERT and RoBERTa, were attempted when finding the optimal transformer model. These model architectures were used in training either unchanged, letting all layers be frozen, or unfreezing some or even all hidden layers, as explained in Table 4.2. Much altering was unnecessary if we constructed the models using the pre-trained models. However, the possibility of changing the hidden layers of each architecture gave us new opportunities. Depending on which pre-trained model we used, they had different layers, shown in Table 4.2. DistilBERT, a denser version of the original BERT, has fewer layers than RoBERTa. Therefore, it has fewer layers to unfreeze. Figure 3.8 in Section 3.4.3 illustrates the three techniques, from all frozen to all unfrozen. As discussed in Section 3.4.3, the greater the number of unfrozen layers in an architecture, the more training was necessary, increasing the time required to train the model. With more parameters retrained, the model adapts more to the data, which might be helpful.

4.4 Resource Requirements and Computer Specific

Our model training process utilized both a local GPU (GTX 1050 Ti) and Google Colab (Tesla A100). While the local GPU was primarily used for training most models, computationally intensive models like m-BERT or XLM-Roberta were trained using Google Colab. On average, training each experiment with recurrent neural networks took approximately 10 minutes, while experiments with transformer architectures required an average of 2 hours and 30 minutes to complete.

Chapter 5
Evaluation

This chapter first presents the results of the experimental setup, presented in Chapter 4. The results were evaluated according to the performance metric described in Section 3.5. Using the mean macro average F1 score and the ROC-AUC score, we compared our methods and determined the optimal one, which to our delight, was better than the one by van Aken et al. (2018).

In Section 5.2 we discuss the results obtained and describe the limitations of the dataset and the annotations of the data.

5.1 Results

5.1.1 Baseline

We combined the TF-IDF and logistic regression and created a binary model for each category. This gave a mean macro average F1-score of 0.7371 and a ROC-AUC score of 0.7060. Table 5.3 in Section 5.1.4 displays the results together with a few chosen models.

5.1.2 Recurrent Neural Networks

Table 5.1 shows the resulting mean macro average F1 score and ROC-AUC score for the models described in Section 4.2. The combination of the word embeddings fastText and GloVe proved to provide better results. Despite our attempts at single-value decomposition, a common dimension-reducing technique, the setup required more success. Varying the maximum length of the comments, varying the batch size, changing the optimizer, or normalizing the embedding matrices did not improve the performance. The architecture with the best performance uses a combined word embedding and has 128 nodes in the BiGRU layer, which has a 0.2 dropout value. The dense layers are 32 nodes and six nodes, where the first layer has a dropout value of 0.1. The learning rate was 0.001.

feature extraction	nodes BiGRU	dropout BiGRU	nodes dense	dropout dense	learning rate	mean macro avg F1 score	mean ROC-AUC score
GloVe50	16	0	[6]	[0]	0.05	0.6941	0.9605
GloVe50	32	0	[6]	[0]	0.05	0.7018	0.9635
GloVe50	64	0.2	[6]	[0]	0.05	0.7214	0.9672
GloVe50	128	0.1	[6]	[0]	0.05	0.7385	0.9685
GloVe50	128	0.1	[16, 6]	[0, 0]	0.05	0.7304	0.9704
GloVe50	128	0	[32, 6]	[0.2, 0]	0.05	0.7416	0.9707
GloVe50	128	0.1	[64, 6]	[0, 0]	0.05	0.7459	0.9706
GloVe50 200max	128	0.2	[32, 6]	[0.1, 0]	0.001	0.7496	0.9642
GloVe50	128	0.2	[32, 6]	[0.1, 0]	0.01	0.6605	0.9602
GloVe50	128	0.2	[32, 6]	[0.1, 0]	0.001	0.7433	0.9703
GloVe100	64	0.2	[6]	[0]	0.005	0.7547	0.9722
fastText300	128	0.2	[32, 6]	[0.1, 0]	0.001	0.7781	0.9817
ft300 + g300	64	0.2	[6]	[0]	0.005	0.7611	0.9791
ft300 + g300 (svd)	64	0.2	[6]	[0]	0.005	0.7652	0.9782
ft300 + g300	**128**	**0.2**	**[32, 6]**	**[0.1, 0]**	**0.001**	**0.7811**	**0.9803**
ft300 + g300 (svd)	128	0.2	[32, 6]	[0.1, 0]	0.001	0.7723	0.9821

Table 5.1: Results with RNN architectures. ft300 means fastText300, g300 means GloVe300. (svd) mean single value decomposition from 600 to 300. 200 max means a maximum length of 200 words.

5.1.3 Transformers

Table 5.2 presents the results from the different transformer experimental setups. The model with the highest performance was the RoBERTa model, with eleven out of thirteen unfrozen layers, comment truncated at 120. It resulted in a mean macro average F1 score of 0.8081, more than 0.01 greater than the best-performing model constructed by van Aken et al. (2018). Also, we beat their mean ROC-AUC with the result of 0.9841, compared to 0.983. Nonetheless, the ROC-AUC score result is such an insignificant difference that we do not dare to claim it much better based on that result.

Figure 5.1 presents the macro average F1 scores of three transformer model architectures based on category. The sample size on the x-axis refers to the number of samples of the particular class, i.e., the category "threat" is the smallest class of less than a thousand samples. Thus it is seen to the far left. In contrast, toxic has the most registers and is thus seen to the far right. The boxes are color-coded by class to enhance clarity and visual appeal. The class "obscene" performs the best, followed by "toxic ". The classes with fewer samples have lower F1 scores compared to the others. The performance of the logistic regression model improves with larger sample sizes. However, categories with low sample sizes exhibit significantly lower scores, while categories with high sample sizes, such as "toxic", perform nearly as well as the RoBERTa and BiGRU models. These findings indicate that when abundant data is available for all categories, employing a more complex model may not be necessary.

Figure 5.2 shows the macro average F1 score based on the number of unfrozen hidden layers. Twitter RoBERTa and RoBERTa perform very similarly when more hidden layers are

model	truncation	layers unfrozen	mean macro avg F1 score	mean ROC-AUC score
distilBERT	50	0/7	0.6192	0.5844
distilBERT	120	0/7	0.6557	0.9492
distilBERT	200	0/7	0.6483	0.6157
distilBERT	300	0/7	0.6100	0.5776
distilBERT	120	4/7	0.7797	0.9824
distilBERT	120	7/7	0.7983	**0.9841**
RoBERTa	120	0/13	0.5737	0.9403
RoBERTa	120	4/13	0.8000	0.9838
RoBERTa	**120**	**11/13**	**0.8081**	**0.9834**
RoBERTa	120	13/13	0.7731	0.9814
Twitter-RoBERTa	120	0/13	0.6861	0.9619
Twitter-RoBERTa	120	10/13	0.7982	0.9840
Twitter-RoBERTa	120	13/13	0.7731	0.9811
m-distilBERT	120	7/7	0.7737	0.9816
m-BERT	200	0/13	0.6504	0.6165
m-BERT	200	13/13	0.7534	0.9763
m-BERT	120	13/13	0.7611	0.9773
xlm-RoBERTa	120	0/13	0.5269	0.9280
xlm-RoBERTa	120	4/13	0.7354	0.9739
xlm-RoBERTa	120	13/13	0.6948	0.9274

Table 5.2: Results from different transformer architectures. The model architecture highlighted is the one giving the best mean macro average F1 score.

unfrozen. Although RoBERTa ultimately gave the best results, the Twitter RoBERTa architecture performed better when fewer layers were unfrozen. The XLM-RoBERTa performed worse than the rest.

feature extraction	model	mm avg F1 score	ROC-AUC score
TF-IDF	logistic regression	0.7371	0.7060
GloVe50	BiGRU [16] + 1 dense	0.6941	0.9605
GloVe300 + fastText300	BiGRU [128] + 2 dense	0.7811	0.9803
RoBERTa	dense, 0/13 unfrozen	0.5737	0.9403
RoBERTa	**dense, 11/13 unfrozen**	**0.8081**	**0.9834**

Table 5.3: Summary of results from baseline, BiGRU, and transformers. From BiGRU and transformers, we display the most simple ones tried as well as the best ones. (mm stands for mean macro,)

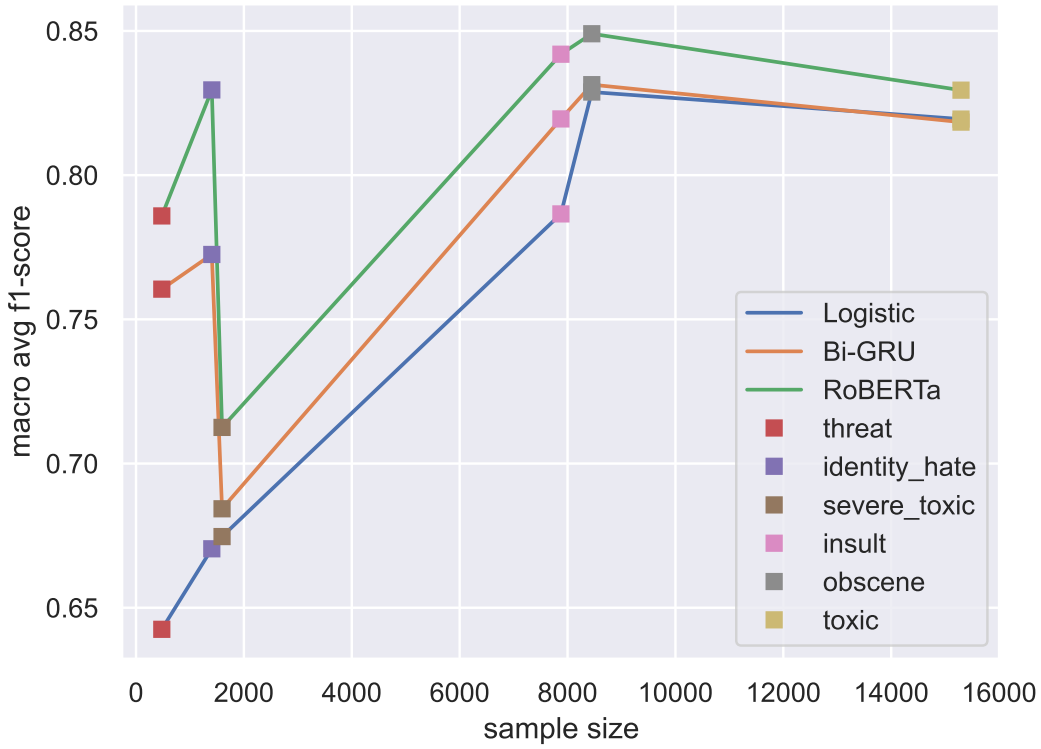

Figure 5.1: Three transformer architectures and their performance for every individual category. The six category performances are seen on six different x-axis locations. They range from "threat" to "toxic", from smallest sample number to greatest.

5.1.4 Results Summarized

Table 5.3 summarizes the most important findings and comparisons. The optimal model is the last one, using RoBERTa with the last 11 layers of the pre-trained model unfrozen and trained. In comparison to the results from van Aken et al. (2018), which achieved a mean macro average F1 score of 0.793 using an ensemble of different models, our transformer-based model is an improvement of **0.015**, suggesting our model scoring **0.808** is better than theirs. The ROC-AUC is also displayed due to being a standard evaluation metric, although the mean macro average F1 score is the most descriptive.

5.2 Discussion

As Table 5.3 suggests, the transformer architecture performs better than a baseline or recurrent neural network-based method. It is not surprising that the best model happens to be a RoBERTa model. RoBERTa is more complex than distilBERT and BERT; hence an improvement was to expect. Surprisingly, distilBERT performs remarkably well, even outperforming all other models with up to six layers. XLM-RoBERTa, on the other hand, does not perform

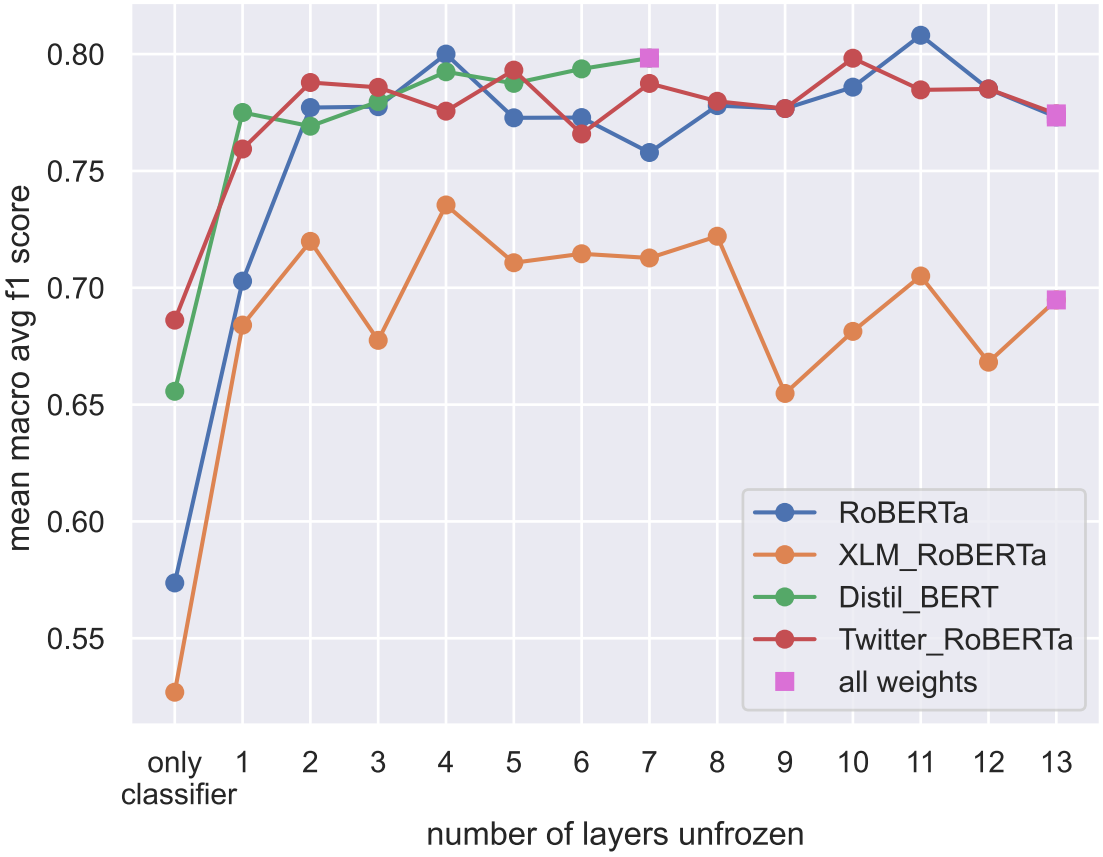

Figure 5.2: Four transformer models and their performance as the number of unfrozen hidden layers is increased.

well. Neither did the m-BERT, suggesting that multilingual models are not well suited for this data set. Although the multilingual models are developed to perform well in English, introducing a model with multilingual capacity might lower the chances of a dataset almost entirely made up of English comments.

RoBERTa likely provides the best results due to having more layers and more trainable parameters. The reason for a transformer-based model being in the lead is likely because of the attention mechanism it possesses. As discussed in Section 3.4.3, the transformer architecture manages to model long-range dependencies, capturing both local and global tendencies. Thus, it is reasonable that it performs better. When categorizing a comment as hateful, it is helpful to look at the whole comment. There might be information at the far end of a comment suggesting that it is, in fact, not hateful, which a transformer-based model can be able to catch. In contrast, the less complex methods will need to consider the entirety of the context to recognize the overall meaning of the comment and focus on much smaller parts.

Although we obtained satisfactory results, there are some limitations and issues with both the dataset and the categorization that are important to address.

One limitation is the perception of what is hateful or not. Depending on the place of origin, cultural background, and language, perceptions of what is hateful will differ. Some individuals may find specific types of language highly offensive, while others may use them in daily conversations without issue. For that reason, categorizing hate speech is a challenging

task for a machine learning model, which might need to be made aware of the background of why this speech is said or not. In our case, the model does not know the subsequent conversation, nor the sender of the comment, thus complicating the task.

Similarly, the annotation process can be subjective. The labeling depends on the individual annotator's cultural, social, and historical background, which can influence their perception of what constitutes hate speech. Ross et al. (2017) addressed the issue of subjectivity in the annotation process, highlighting the problems behind the definition of toxic speech. In their study, they presented a definition of hate speech to one group of people. They did not show the other group a definition. It became apparent that the perception of what is hateful differs. They concluded that detailed instructions and more concise definitions are required to annotate objectively.

Another study discussing subjectivity in hate speech is the one presented by Khurana et al. (2022). They address the issues of not having set definitions of hate speech but also provide guidelines on how hate speech should be annotated based on perspectives within social science and law. They specifically give guidelines based on factors such as target groups and their social status, perpetrator, type of negative reference, and potential consequence. They stress the importance of reducing subjectivity in hate speech data to obtain less biased models.

We could find little information on the annotation of the toxic comment data we used. For that reason, there is uncertainty in how reliable the annotation is. While experimenting with the trained model, we noticed that comments directed towards certain ethnic or religious groups get classified as more toxic than others. That suggests a bias in the annotation process and the identity hate category specifically or of lacking toxic comments towards certain groups of people.

An additional problem with the dataset and its annotation is that there are some highly toxic comments that our model does not catch due to acronyms or rewrites from toxic words. The highly antisemitic number 88 is not recognized as toxic, severe, or identity hate, although it symbolizes Nazis and Hitler. Additionally, acronyms such as *fu*, short for *fuck you*, are not recognized either, meaning that some offensive words and phrases could slip by.

Likewise, the model does not catch irony or sarcasm. That is a challenging task to manage due to the nuances of language. When catching irony and sarcasm, an analysis of previous comments or the topic of the conversation is needed. That would give a context of the *conversation* rather than the *comment*, which we have been looking at so far.

Despite the issues with annotation and the dataset, the optimal model performs very well. Our discussion suggests that future models designed to detect hate speech are likely to perform even better, which is an optimistic thought.

Chapter 6
Application

One of the aims of our thesis was to implement an application visualizing the model's performance when fed a comment. We successfully created a user-friendly interface using Gradio. Gradio is a powerful Python library that streamlines the process of building demos for machine-learning models with just a few lines of code. Figure 6.1 illustrates the system diagram depicting the structure behind a general application. It demonstrates the order of the process flow, where a request is sent to the web server via a REST API when a user submits a comment. The server checks if the comment already exists in the database. If it does, it retrieves the corresponding predictions from the database. If the comment is not found in the database, it gets forwarded to the classifier for the prediction of probabilities of classes. These predictions are then returned to the user and stored in the database for future reference. This process ensures that existing predictions are efficiently retrieved from the database while new comments undergo classification, and their predictions are made available to the user in real-time.

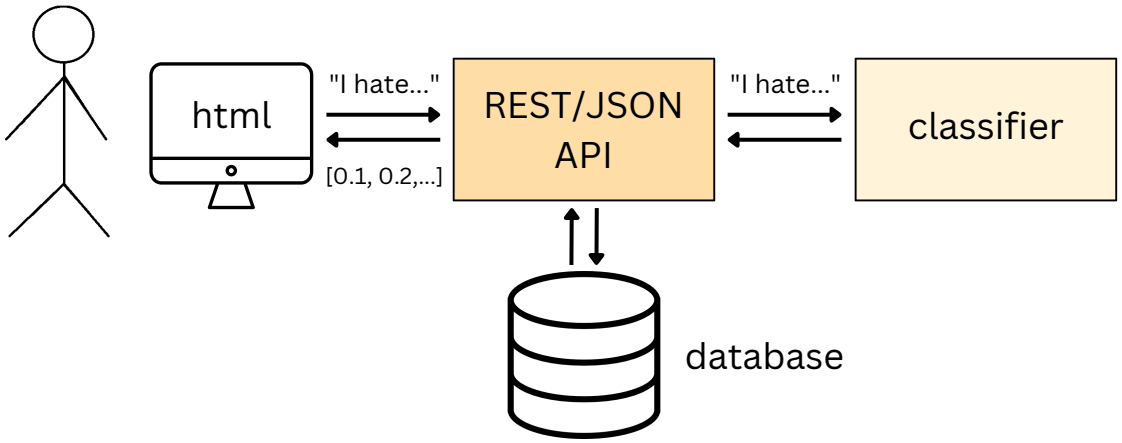

Figure 6.1: The system diagram of the model.

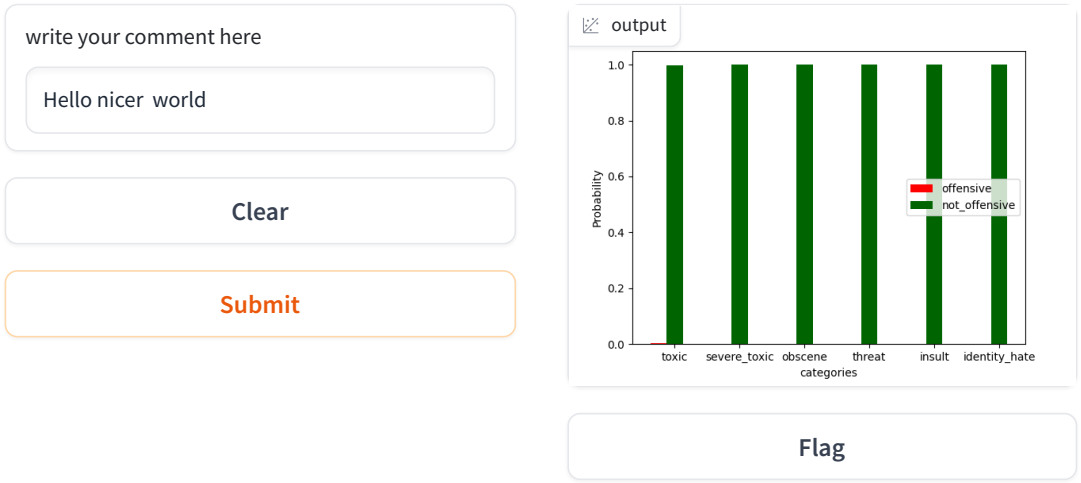

Figure 6.2: A simple application of our model. One can write a comment in the textbox and press submit to get the plot right.

The application possibilities of this model are vast. It can act as a tool for warning against harmful content in comments on social media, particularly for sensitive users. Additionally, it can operate as a writing assistant for individuals with autism who may not be aware that their words can be toxic. This model holds great potential for fostering a safer and more inclusive online environment.

In Figure 6.2, we test non-toxic comments in our Gradio environment. The green bars indicate the probability of that specific category being non-offensive.

Next, we demonstrate an insulting comment. Figure 6.3 shows a comment with a high probability of being classified as toxic, obscene, and insulting. Whether or not the model should classify the comment as obscene can be discussed, depending on the definition of

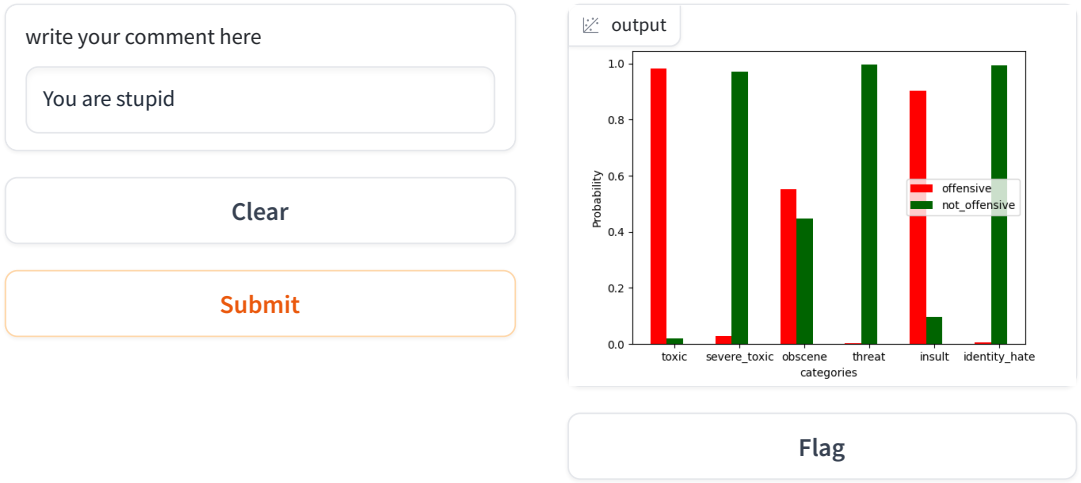

Figure 6.3: The prediction plot when the input is an insulting comment.

obscene, but the other two categories the model predicts correctly.

In Figure 6.4, we present the results from a test with a threatening comment. Our model could predict the two classes “threat” and “toxic” correctly.

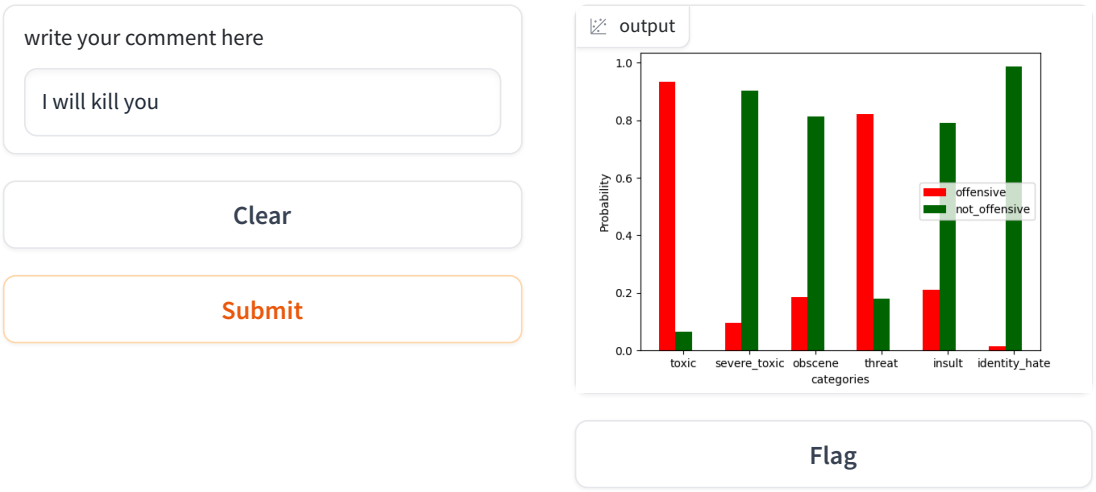

Figure 6.4: The prediction plot when the input is a threatening comment.

The aim is to use the visual representation of the model results, presented in this section, as inspiration for its use on online platforms.

Chapter 7
Conclusions

This thesis presented a comprehensive approach to the multi-label classification of toxic comments. We established a baseline by vectorizing the text using TF-IDF, followed by classifying each comment using logistic regression. To improve our results further, we utilized word embedding to vectorize the text and gated neural networks followed by dense layers to classify the comments. Ultimately, we obtained the best result using a fine-tuned transformer architecture. To our delight, the mean macro average F1 score was better than we have found in previous studies on this dataset.

Further improvement is still possible. One of the limitations discussed in Section 5.2 was the difficulty of detecting irony and sarcasm. One potential enhancement is to utilize large language models (LLMs), which have been trained on vast amounts of data and can detect sarcasm to a certain degree. To determine if a sentence is toxic, one can utilize LLMs to predict the probabilities of categories or the likelihood of the sentence being sarcastic. The classifier and LLM's response can then decide based on that.

According to Huggingface documentation, there are more than 120 thousand transformer models. One future path for this project could be to experiment with more models and improve performance even more. Additionally, we noticed that some models are better at predicting if a comment is toxic, and others are better at predicting the type of toxicity. In the future, one can improve performance further by using an ensemble of different models.

Based on the results, our model is considered a state-of-the-art model. Even though there is room for improvement, we are happy to present our findings to the open-source community. We hope our thesis will inspire others to pursue further studies and that we, collectively, can reduce toxicity online.

References

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.

Chollet, F. (2021). *Deep Learning with Python, Second Edition*. Manning.

cjadams, J. S., Elliott, J., Dixon, L., McDonald, M., nithum, and Cukierski, W. (2017). Toxic comment classification challenge. *Kaggle*.

Davidson, T., Warmsley, D., Macy, M. W., and Weber, I. (2017). Automated hate speech detection and the problem of offensive language. *CoRR*, abs/1703.04009.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Dharma, E. M., Gaol, F. L., Warnars, H., and Soewito, B. (2022). The accuracy comparison among word2vec, glove, and fasttext towards convolution neural network (cnn) text classification. *Journal of Theoretical and Applied Information Technology*, 100(2):31.

El-Din, D. M. (2016). Enhancement bag-of-words model for solving the challenges of sentiment analysis. *International Journal of Advanced Computer Science and Applications*, 7(1).

Fawcett, T. (2006). An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861–874. ROC Analysis in Pattern Recognition.

Foote, K. D. (2019). A brief history of natural language processing (nlp). `https://www.dataversity.net/a-brief-history-of-natural-language-processing-nlp/`. (Accessed on 02/02/2023).

Gautam, H. (2020). Word embedding: Basics. Medium.

Georgakopoulos, S. V., Tasoulis, S. K., Vrahatis, A. G., and Plagianakos, V. P. (2018). Convolutional neural networks for toxic comment classification. In *Proceedings of the 10th Hellenic*

Conference on Artificial Intelligence, SETN '18, New York, NY, USA. Association for Computing Machinery.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9:1735–80.

Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2016). Bag of tricks for efficient text classification. *CoRR*, abs/1607.01759.

Kauranen, M. (2020). Cross-lingual comment toxicity classification. `https://lup.lub.lu.se/student-papers/search/publication/9024850`. (Accessed on 02/02/2023).

Khurana, U., Vermeulen, I., Nalisnick, E., Van Noorloos, M., and Fokkens, A. (2022). Hate speech criteria: A modular approach to task-specific hate speech definitions. *arXiv preprint arXiv:2206.15455*.

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. *CoRR*, abs/1907.11692.

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

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Ross, B., Rist, M., Carbonell, G., Cabrera, B., Kurowsky, N., and Wojatzki, M. (2017). Measuring the reliability of hate speech annotations: The case of the european refugee crisis. *CoRR*, abs/1701.08118.

Schmidt, A. and Wiegand, M. (2017). A survey on hate speech detection using natural language processing. In *Proceedings of the Fifth International Workshop on Natural Language Processing for Social Media*, pages 1–10, Valencia, Spain. Association for Computational Linguistics.

Sparck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21.

Tunstall, L., von Werra, L., and Wolf, T. (2022). *Natural Language Processing with Transformers: Building Language Applications with Hugging Face*. O'Reilly Media, Incorporated.

van Aken, B., Risch, J., Krestel, R., and Löser, A. (2018). Challenges for toxic comment classification: An in-depth error analysis. *CoRR*, abs/1809.07572.

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, B., Li, C., Pavlu, V., and Aslam, J. (2017). Regularizing model complexity and label structure for multi-label text classification.

Warner, W. and Hirschberg, J. (2012). Detecting hate speech on the world wide web. In *Proceedings of the Second Workshop on Language in Social Media*, pages 19–26, Montréal, Canada. Association for Computational Linguistics.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. (2019). Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Yin, W., Kann, K., Yu, M., and Schütze, H. (2017). Comparative study of CNN and RNN for natural language processing. *CoRR*, abs/1702.01923.

Zampieri, M., Malmasi, S., Nakov, P., Rosenthal, S., Farra, N., and Kumar, R. (2019). Predicting the type and target of offensive posts in social media. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1415–1420, Minneapolis, Minnesota. Association for Computational Linguistics.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2023-06-01

EXAMENSARBETE Multi-Label Toxic Comment Classification Using Machine Learning: An In-Depth Study
STUDENT Mosa Hosseini, Matilda Froste
HANDLEDARE Pierre Nugues (LTH), Björn Granvik (Prevas)
EXAMINATOR Jacek Malec (LTH)

Är du lack? AI säger hat online - nej tack

POPULÄRVETENSKAPLIG SAMMANFATTNING **Mosa Hosseini, Matilda Froste**

Förvandla hat till hopp i den digitala världen med vår hatdetektor. Vi använder artificiell intelligens för att identifiera och kategorisera hatiska kommentarer och hjälper moderatorer att skapa en tryggare och mer kärleksfull online-miljö.

Vi lever i en digitaliserad värld. Många av nutidens konversationer sker online, både professionella och privata. De är fyllda av kärlek, hyllning och optimism, men lika ofta är de fyllda av hat. Många har en tendens att tro att det digitala rummet är mer stryktåligt på grund av känslan av anonymitet och avstånd från direkta konsekvenser. I den digitala världen kan hatet flöda fritt, och sprids mer än de skulle göra i det verkliga livet. Det är dags att ta itu med detta problem och skapa en tryggare online-miljö för alla. Vi beslutade att använda vår expertis för att tackla detta problem.

Tack vare maskininlärning, en teknik inom artificiell intelligens (AI), har vi kunnat ta steg för att bekämpa spridningen av hat online. Vi har, med hjälp av stora mängder trevliga och toxiska Wikipedia-kommentarer, skapat en hatdetektor. Målet var att skapa en maskininlärningsmodell bra nog att upptäcka hat i kommentarer. Inte bara skulle den känna igen toxicitet i kommentarer, den skulle även kunna kategorisera dessa kommentarer för mer precision. Det lyckades vi med. Den slutliga modellen fick ett genomsnittligt macro average F1-resultat på 0.808, där ett resultat så nära ett som möjligt är att eftersträva. Vi slog därmed det tidigare rekordet på 0.791.

Maskininlärningsprocessen bestod av fyra huvudsakliga delar: förbehandling av data, extrahering av textegenskaper, träning av en maskininlärningsmodell och utvärdering av modellen med hjälp av testdata. Denna process upprepades många gånger för att hitta en optimal modell.

I figuren ovan syns en visualisering av hur den optimala modellen presterar. En kommentar skickas in och modellen förutser sannolikheten av att kommentaren tillhör en av de sex kategorierna.

Det slutliga målet med detta examensarbete är göra hatdetektorn tillgänglig för användning på sociala medieplattformar och diskussionsforum, där spridningen av hat och skadliga kommentarer är särskilt utbredd. Den ska kunna tillämpas online för att upptäcka illasinnade kommentarer och flagga dem som hatiska. Detta kommer att underlätta för moderatorer att få hjälp i sitt arbete med att identifiera och ta bort skadliga kommentarer.