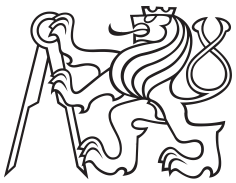
Bachelor's Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Toxic Content Recognition in Conversational Systems

Adam Černý

Supervisor: Ing. Jakub Konrád
Study program: Open Informatics
Specialisation: Artificial Intelligence and Computer Science
May 2023

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Černý Adam** Personal ID number: **499126**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence and Computer Science**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Toxic Content Recognition in Conversational Systems

Bachelor’s thesis title in Czech:

Rozpoznání toxického obsahu v konverzačních systémech

Guidelines:

The goal of the thesis is to devise a method of recognition of toxic content in conversational systems.
1) Review the current methods used for recognizing toxic content, with focus on methods based on embedding clustering and neural Language Model (LM) methods.
2) Select two LM methods, implement them, and compare the results using datasets such as the Jigsaw toxicity and Wikipedia toxicity datasets.
3) Evaluate the performance of different models and identify the most suitable method for toxic content recognition.
4) Use appropriate evaluation metrics to compare the results with current state-of-the-art (SOTA) methods in the field.
In addition to technical challenges, this task also includes ethical considerations and challenges, as the definition of toxic content can vary depending on the context and the use case. Therefore, it's important to carefully choose or extend the dataset and the evaluation criteria.

Bibliography / sources:

[1] Davidson, Thomas, Dana Warmsley, Michael Macy, and Ingmar Weber. "Automated hate speech detection and the problem of offensive language." In Proceedings of the international AAAI conference on web and social media, vol. 11, no. 1, pp. 512-515. 2017.
[2] Xu, Jing, Da Ju, Margaret Li, Y-Lan Boureau, Jason Weston, and Emily Dinan. "Recipes for safety in open-domain chatbots." arXiv preprint arXiv:2010.07079 (2020).
[3] Zhao, Zhixue, Ziqi Zhang, and Frank Hopfgartner. "A comparative study of using pre-trained language models for toxic comment classification." In Companion Proceedings of the Web Conference 2021, pp. 500-507. 2021.

Name and workplace of bachelor’s thesis supervisor:

Ing. Jakub Konrád Department of Cybernetics FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **01.02.2023** Deadline for bachelor thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

Ing. Jakub Konrád	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

. _____ Date of assignment receipt

_____ Student's signature

Acknowledgements

I would like to express my gratitude to my supervisor, Ing. Jakub Konrád, for his invaluable guidance throughout the entire duration of writing this thesis. I would also like to thank Ing. Jan Šedivý, CSc. and Ing. David Herel for their suggestions and advice that helped me solve numerous problems. Lastly, I would like to thank my family for their support and encouragement.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 24, 2023

Adam Černý

Abstract

Conversational AI systems, such as chatbots, are becoming increasingly popular in various industries and are often used to assist with customer service, help users navigate websites and perform other related tasks. However, these systems can be vulnerable to users who may try to insult, harm or be offensive to the system or deceive the system to make it say something toxic. In this project, we investigate the problem of toxic language recognition in conversational AI systems. We review the current state-of-the-art in this area and propose methods to solve the problem. We evaluate the effectiveness of proposed techniques through a series of experiments on a dataset of tweets, present the results and compare them to SOTA methods. Based on the results, we suggest potential directions for future research in the area.

Keywords: natural language processing, machine learning, text classification, toxicity recognition

Supervisor: Ing. Jakub Konrád

Abstrakt

Systémy konverzační umělé inteligence, jako jsou chatboti, nabírají na popularitě v mnoha odvětvích průmyslu a jsou často používány pro pomoc s péčí o zákazníky, orientací uživatelů na webové stránce či pro jiné úlohy podobného charakteru. Tyto systémy mohou mít problém rozpoznat toxické uživatele, kteří se je snaží urazit, chovají se k nim nevhodně či se je snaží zmanipulovat, aby je donutili říct něco urážlivého. V této práci se na problém rozpoznání toxickémho obsahu v dialogových systémech zaměřujeme podrobně. Popisujeme aktuální state-of-the-art metody v oblasti NLP a navrhujeme postupy, které je možné použít k řešení problému. Efektivitu navhrhovaných řešení vyhodnocujeme na datasetu. Na základě výsledků popisujeme kvalitu jednotlivých navrhovaných metod, porovnáváme je se SOTA metodami a navrhujeme směr, kterým by se mohl ubírat náš budoucí výzkum v oblasti.

Klíčová slova: zpracování přirozeného jazyka, strojové učení, klasifikace textu, rozpoznání toxického obsahu

Překlad názvu: Rozpoznání toxického obsahu v konverzačních systémech

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Motivation and goal

Toxic content recognition has become an important problem in conversational AI. Such content includes hateful speech such as racism or sexism, trolling or using offensive language. Detecting toxic content helps dialogue systems and other platforms to provide a satisfactory user experience by creating a safe environment.

Besides customer experience, many countries have laws and regulations that require online platforms to remove toxic content. Failure to comply with these laws can result in legal action against the platform. Therefore, accurate identification of toxic content is necessary for compliance and avoiding legal repercussions.

Moreover, platforms that fail to create a healthy, non-toxic environment may suffer great harm to their reputation and therefore lose popularity. Thus, identifying and removing harmful content is critical for maintaining a positive brand image and reputation as well as an active user base.

Overall, toxicity recognition in conversational AI is a complicated task, and SOTA models often use context in order to improve performance and robustness against adversarial attacks [1]. However, such models require a lot of computation time and memory. In this work, we will explore the capabilities of methods that are not as computationally expensive. Our methods could then be used in situations where low memory usage and fast training are sought-after. We will focus on supervised fastText and recurrent neural networks and propose our own methods that use these algorithms. Our classifiers will be designed for toxicity recognition in general - they will be usable in conversational AI as well as other use cases, such as filtering toxicity on social media.

1.2 Structure

Firstly, we will provide a theoretical background that is crucial for understanding and potentially improving state-of-the-art techniques. Secondly, we will give an overview of related work introduced to this day, and thirdly we will propose our own methods for toxicity recognition. We will describe our proposed methods in detail and illustrate their performance on a dataset. Lastly, we will compare our methods to state-of-the-art approaches and provide conclusions and suggestions for future research.

Chapter 2

Theoretical background

In this chapter, we will provide an overview of the theory needed to understand the machine learning, natural language processing and text classification concepts that will be used in this thesis.

2.1 Traditional machine learning

Machine learning is a subset of artificial intelligence that focuses on the design and implementation of algorithms and models capable of learning from and making predictions based on data inputs. These systems are able to continuously improve their performance through exposure to additional data without the need for explicit programming for specific tasks, which allows them to make more accurate predictions.

ML has a wide range of applications, including image and speech recognition, natural language processing, and predicting outcomes in various industries such as finance or healthcare.

2.1.1 Unsupervised learning

Unsupervised learning is a type of machine learning in which a model is trained on an unlabelled dataset and is expected to discover patterns and relationships in the data on its own. That means unsupervised learning does not require any predetermined outcomes or labels.

2.1.2 Supervised learning

In supervised learning, a model is trained on a labelled dataset, where the correct output is provided for each input. The model makes predictions based on this training and is then evaluated on how well it is able to make predictions for new, previously unseen data.

2.1.3 Logistic regression

Logistic regression is often used as a baseline classifier for supervised learning tasks in NLP. Since we use it in our experiments, it will be described in this section.

In its simplest form, logistic regression classifies into two classes - positive and negative [2]. The classification process is based on estimating the probability that an input vector $\mathbf{x}$ belongs to class 0 or 1. Then the predicted label y is the one that has a higher probability. Formally, it can be written as follows:

$$\hat{y} = \begin{cases} 1 & \text{if } P(y = 1|\mathbf{x}) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

where $P(y = k|\mathbf{x})$ denotes the probability that an input $\mathbf{x}$ belongs to class $k \in \{0, 1\}$.

The above-mentioned probability is constructed using a vector $\mathbf{w}$ and a bias term b. The probability of an input $\mathbf{x}$ belonging to class 1 is than calculated using the following formula:

$$P(y = 1|\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$$

Since there are only two classes, the probability of $\mathbf{x}$ belonging to class 0 can be computed as:

$$P(y = 0|\mathbf{x}) = 1 - \sigma(\mathbf{w} \cdot \mathbf{x} + b)$$

In both above-mentioned formulas, σ denotes the sigmoid function, which is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

This function is used to convert the predicted value to a probability.

Multinomial logistic regression

In many cases, we need to classify inputs into more than two classes. For these purposes, we can generalize logistic regression into its multinomial form. The only difference is that to classify an input $\mathbf{x}$, we use separate weights and biases for each class in order to obtain a vector of predicted probabilities $\hat{\mathbf{y}}$.

This is done using the $softmax$ function instead of $sigmoid$. The purpose of these functions is the same, but softmax can be used to convert a vector into a probability distribution. It is a generalization of the sigmoid function and is defined as follows:

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, K$$

where z_i is the i-th component of an input vector $\mathbf{z}$ and K is the number of classes.

With the softmax function, we can get a vector of probabilities $\hat{\mathbf{y}}$ the following way:

$$\hat{\mathbf{y}} = softmax(\mathbf{W}\mathbf{x} + \mathbf{b})$$

where $\mathbf{W}$ is a matrix of weight vectors for each class in rows, $\mathbf{x}$ is an input vector and $\mathbf{b}$ is a vector of biases for each class.

By following this computation process, we get an output vector $\hat{\mathbf{y}}$ that has K elements (the number of classes) and the element y_k is the probability that vector $\mathbf{x}$ is from class k. We classify an input $\mathbf{x}$ as the class that has the highest probability.

Training

To obtain $\mathbf{w}$ and b, we need to train the classifier and to do so, we need to define an objective (loss) function. For logistic regression, that is typically cross-entropy loss which determines how different the expected output is from the actual output. It is defined as:

$$L_{CE}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{k=1}^{K} y_k \log(\hat{y}_k)$$

where K is the number of classes, $\mathbf{y}$ is the expected output (one-hot vector) and $\hat{\mathbf{y}}$ is the actual output.

That can be simplified into negative log-likelihood of the correct class, giving us the following formula:

$$L_{CE}(\hat{\mathbf{y}}, \mathbf{y}) = -\log(\hat{y}_c)$$

where $\hat{y}_c$ is the output probability corresponding to the correct class c.

The whole learning process consists of minimizing the average value of the loss function over the training set. This is done using an optimization algorithm - usually gradient descent or stochastic gradient descent.

Gradient descent

Thanks to the fact that the gradient of a function is a vector that points in the direction of the steepest increase in a function, gradient descent can minimize a function by moving in the opposite direction than the gradient.

Formally, GD is an iterative algorithm that performs an update of parameters with the aim to minimize a function. In each step t, it makes the following computation:

$$\theta^{t+1} := \theta^t - \gamma \nabla L(f(x;\theta), y)$$

where θ denotes the vector of parameters (in the case of logistic regression, it would be $(\mathbf{w}, b)$), $f(x;\theta)$ is the representation of our prediction $\hat{\mathbf{y}}$ that shows that it is dependent on the parameter vector θ, ∇L denotes the gradient of the loss function with respect to the parameters and γ is the learning rate - a hyper-parameter that determines how much the algorithm moves on each step.

Stochastic gradient descent

Stochastic gradient descent is an algorithm that aims to compute similar results as GD without the need for as many computational resources and as much time. It achieves this goal by computing the gradient after just one randomly chosen training example or a mini-batch (small subset) of random examples instead of going through all the examples like GD does.

For both GD and SGD, it is very important to pick the learning rate carefully because it determines the step size of the algorithm. It is common to either pick a fixed value, start with a high learning rate that decreases over time or use an optimizer such as Adam [3] [2].

For logistic regression, the cross-entropy loss function is convex, and therefore there is no risk of getting stuck in local minima [2].

2.2 Artificial neural networks

Artificial neural networks are a type of machine learning algorithm that has the ability to learn and model complex patterns in data. The algorithm is inspired by the human brain, where information is processed through a network of interconnected neurons [2]. This concept is relatively old, dating back to 1943 [4]. In recent decades, the idea has really found its use and has achieved state-of-art results in many areas of machine learning, such as computer vision, robotics, finance, gaming and natural language processing. In NLP, various

architectures of ANNs are used for text classification, sentiment analysis, language translation, named entity recognition or text summarization.

In artificial neural networks, the basic building block is a neuron (sometimes called a unit), which receives a set of real values as input and produces an output based on a set of weights and biases. These weights and biases are learned through a training process that adjusts them to minimize the difference between the predicted output and the actual output. The computation in each unit is done as follows:

$$y = f(\mathbf{w} \cdot \mathbf{x} + b)$$

where $\mathbf{w}$ is the weights vector, b is the bias term, $\mathbf{x}$ is the input vector, and f is an activation function. The most widely used activation functions include relu, sigmoid, tanh and softmax. They are defined as follows:

- $relu(z) = max(0, z)$
- $sigmoid(z) = \frac{1}{1+e^{-z}}$
- $tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = \frac{1-e^{-2z}}{1+e^{-2z}}$
- $softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, K$

Activation functions are important because they can be non-linear and therefore allow the model to fit well to data that are not linearly separable.

2.2.1 Feedforward neural networks

In feedforward neural networks, neurons are organised into layers and connected without cycles. That means that the outputs of each layer are passed to the units in the next layer (hence the name feedforward). Each unit in the n-th layer of the network is connected to each unit in layers $n-1$ and $n+1$ (fully connected).

In vector form, a basic feedforward neural network can be computed as follows:

$$\mathbf{h} = f(\mathbf{W}\mathbf{x} + \mathbf{b_0})$$

$$\mathbf{y} = g(\mathbf{V}\mathbf{h} + \mathbf{b_1})$$

where $\mathbf{x}$ is the input (real valued vector), $\mathbf{W}$ and $\mathbf{V}$ are weight matrices, $\mathbf{b_0}$ and $\mathbf{b_1}$ are bias vectors, $\mathbf{h}$ is the hidden layer and $\mathbf{y}$ is the output.

In text classification, $softmax$ is typically used to get a probability distribution over the possible classes as output.

More layers can be added analogically by using the output of the last layer ($\mathbf{y}$ in the example) as input to compute the next hidden layer.

Training

In supervised text classification, the training data typically consists of annotated inputs that are fed to the network and used for optimization. The network uses the labels in the training data to adjust the weights accordingly so that the outputs of the network on the training dataset and the expected (real) outputs are as close as possible.

In this process, it is crucial to define a loss function that can be optimized. This is typically cross-entropy loss which is described above.

The whole learning process consists of minimizing the average value of the loss function over the training set. This is done by an optimization algorithm such as gradient descent or stochastic gradient descent.

To be able to use GD or SGD, we need to know the gradient of the loss function with respect to each of the parameters. With only one layer and a sigmoid or softmax activation function (logistic regression), this is straightforward because we can simply use the derivative of the activation function. In ANNs, this is not an easy task because they often have millions of parameters and multiple layers. The problem of computing the partial derivatives of the weights over all previous layers is solved by an algorithm called backpropagation [5], which utilizes the chain rule to compute partial derivatives of the loss function with respect to each of the parameters in a backward pass of the network.

The optimization problem for multi-layer neural networks is not convex, and therefore we face the risk of (S)GD getting stuck in a local minimum. This problem is partially solved by following best practices such as random weights initialization, using dropout [6] and tuning hyper-parameters.

2.2.2 Recurrent neural networks

A recurrent neural network (RNN) is a type of neural network that is particularly well-suited for processing sequential data, such as text. RNNs have a memory component that allows them to maintain information about previous inputs and use it to inform their processing of subsequent inputs. This makes them particularly useful for tasks that require an understanding of context and the relationships between words in a sequence.

In an RNN, each unit in the network has a connection to itself, as well as connections to the units in the previous and next time steps. The simplest form of an RNN is an Elman network [7], which consists of an input layer, a hidden layer and an output layer. In each timestep, the hidden state of the

network is computed and used in the next timestep in combination with the input. This allows the network to use information from previous time steps when processing the current input.

A recurrent neural network can be viewed as an extension of a feedforward neural network. We only need to add a matrix $\mathbf{U}$ that contains the weights between hidden layers in time t and $t-1$ and a bias vector $\mathbf{b_2}$ for matrix $\mathbf{U}$. Computation of the current hidden layer can then be done as follows:

$$\mathbf{h_t} = f(\mathbf{W x_t} + \mathbf{b_0} + \mathbf{U h_{t-1}} + \mathbf{b_2})$$

Other than this addition, we do not need to modify anything, and the process stays the same as in FNN (previous section). (S)GD and backpropagation can again be used for optimization. However, backpropagation through time described in [8] can improve the performance [9].

Such a network can capture context relatively well but has its problems. Mainly the exploding/vanishing gradient problem [10] where the propagation of the gradient causes it to be very large/small. Because of this issue, the model can not efficiently remember long context. This problem can be partially solved [11] by more complex RNN variants such as long-short-term memory (LSTM) [12].

2.2.3 Transformers

The transformer architecture was introduced in the 2019 paper called Attention is All You Need [13]. It has outperformed many preexisting solutions and helped to create state-of-the-art language models such as GPT4 [14] or BERT [15], which is a model that is very well suited for finetuning for tasks such as question answering or language inference.

The key idea behind the transformer architecture is the use of self-attention mechanisms, which allow the model to attend to different parts of the input sequence at different times rather than processing the input in a fixed order. This allows the model to capture long-range dependencies in the input sequence.

The transformer architecture consists of a sequence of encoder and decoder layers, each of which is composed of multiple self-attention layers and feedforward layers. The input to the encoder is a sequence of word embeddings, and the output is a sequence of hidden states that capture the meaning of the input sequence. The decoder then processes the hidden states and produces an output sequence.

One of the key advantages of transformer architecture is that it is fully parallelizable, which means that it can be efficiently trained on large datasets using multiple GPUs. This has made it possible to train large transformer

models that achieve state-of-the-art performance on a wide range of NLP tasks [2].

2.3 Natural language processing

Natural language processing (NLP) is a subfield of artificial intelligence that focuses on giving computers the ability to understand, interpret, and generate human language. There are several tasks that NLP systems are capable of performing, including language translation, text classification, sentiment analysis, named entity recognition and question answering.

2.3.1 Text classification

Text classification is a process that involves assigning text to one or more predefined categories based on its content. It is often used to classify documents, such as news articles or emails into predefined categories or labels, such as news, spam, or advertisement.

There are several approaches to text classification, including rule-based systems, which rely on a set of predefined rules to classify text, and machine learning-based systems, which learn to classify text through training on a dataset.

2.3.2 Bag of words

The simplest way of representing a word in a dictionary (corpus) is to have a vector of the size of the dictionary where all elements except the one on the index of the represented word are zeros. Only the element on the word's index is one. This is called one-hot encoding.

To represent a whole sentence, we can add one-hot encoded words together and get what is called a Bag of Words (BoW). It is clear that the order of words in the original sentence is ignored in BoW [2].

For example, if we consider the following documents (word sequences):

- Document 1: *Thomas went out and it was raining*
- Document 2: *Thomas likes it when it is raining*

We get the vector representations in table 2.1.

	Thomas	went	out	and	it	was	raining	likes	when	is
Document 1	1	1	1	1	1	1	1	0	0	0
Document 2	1	0	0	0	2	0	1	1	1	1

Table 2.1: Example of two documents encoded into vector space using BoW.

2.3.3 N-grams

An n-gram is a sequence of words that appear right after each other. For example, in the sentence "What a beautiful day!", the bigrams (sequences of two words) would be "What a", "a beautiful", and "beautiful day!". We can form trigrams and longer n-grams analogically.

N-grams are used in the simplest language models to capture context and predict the next word. Frequently occurring n-grams such as "New York" can also be added to a corpus and viewed as phrases [16].

Besides the above-described word n-grams, we can also add character n-grams to the corpus to improve the robustness of a model [17].

2.3.4 Word embeddings

Word embeddings are numerical representations of words or phrases in a high-dimensional vector space. These representations are used to capture the meaning and context of words in a way that is much more flexible and powerful than traditional one-hot encoding schemes.

One of the main advantages of word embeddings is that they can capture complex relationships between words and the context in which they are often used. That means that two words that are similar in meaning may have very similar embeddings even though they are not spelled the same and do not share any common subwords. This allows word embeddings to capture the semantics of a language in a way that is more robust and generalizable.

We can combine word embeddings with the above-mentioned BoW representation by averaging the vectors of all words in a sentence to get a sentence vector.

2.3.5 Word2vec

Word2vec [18] is the first-ever method for learning fixed-length word embeddings based on the context in which the word appears in a large corpus of text. These vectors are learned in such a way that words that appear in similar contexts are closer to each other in vector space. For example, the vectors for the words "king", "queen", and "princess" will likely be close together in vector space.

Simple algebraic operations can be used to capture relationships between words. For example, embedding("king") - embedding("man") + embedding("woman") results in a vector that is closest to the vector representation of the word "queen" [19]. However, it is important to note that this only works in some cases. It has been shown that male-female analogies are one of the easiest to capture for the algorithm [20]. Popularising such analogies has been

criticised for portraying the capabilities of the algorithm in an unrealistic way and making incorrect conclusions based on them [21].

There are two main approaches used to capture context in word2vec: the Continuous Bag-of-Words (CBOW) model and the Skip-Gram model.

In the CBOW model, the goal is to predict the current word, given the context of the words surrounding it. For example, given the sentence "the cat sat", the goal is to predict the word "cat" given the context "the" and "sat". In our example, we use a window size of only 1, but in practice, this number would be larger. Generally, the CBOW model tries to learn the probability distribution P(w|C), where w is the target word, and C is the context.

On the other hand, in the Skip-Gram model, the goal is to predict the context given a target word. For example, given the word "cat," the goal is to predict the words "the" and "sat". The Skip-Gram model tries to learn the probability distribution P(C|w), where C is the context and w is the target word.

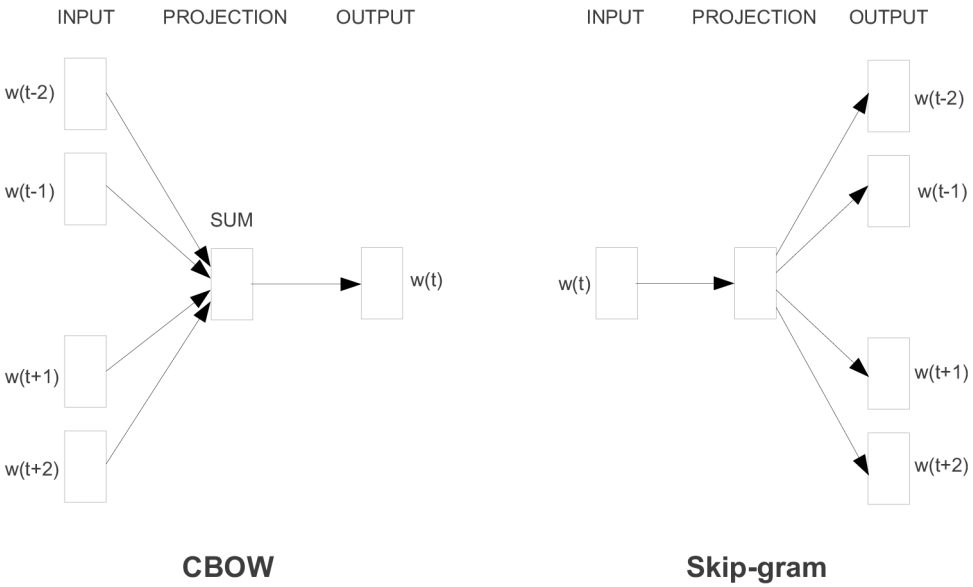


Figure 2.1: The CBOW architecture predicts a word given its context, whereas Skip-gram predicts the context given the word [18].

2.3.6 FastText

FastText [17] is an algorithm for efficient learning of word representations and text classification. It was developed by Facebook and is based on the idea of representing words as a bag of character n-grams rather than as a sequence of letters. In a way, it is an extension of word2vec. The main difference between these two algorithms is that fastText uses character n-grams as opposed to word2vec, which uses whole words. This is why fastText works well for morphologically rich languages such as Turkish or Finish and misspelt

or infrequent words that contain character n-grams similar to some more common words. This approach also helps fastText capture partial information about a word even if it has not been seen in the training data.

Similarly to word2vec, fastText can be trained using either the Continuous Bag-of-Words (CBOW) model or the Skip-Gram model.

The algorithm can be used for learning both supervised [22] and unsupervised [17] word embeddings. In this thesis, we mainly use supervised fastText embeddings, and therefore we describe the training process needed to obtain them.

Training

The training of supervised word embeddings is very similar to the CBOW model for unsupervised ones. The main difference is that in supervised training, the goal label is put in the context instead of the word itself.

FastText is a shallow neural network with an input layer, a linear layer and an output layer with two weight matrices $\mathbf{A}$ and $\mathbf{B}$. $\mathbf{A}$ is used to compute the hidden representation of input vector $\mathbf{x}$ (normalized bag of features) as $hidden = \mathbf{A} \cdot \mathbf{x}$ and $\mathbf{B}$ represents the weights of a linear classifier (multinomial logistic regression).

During training, the goal is to minimize the negative log-likelihood over the classes:

$$-\frac{1}{N}\sum_{n=1}^{N} \mathbf{y_n} \log(softmax(\mathbf{BAx}_n))$$

where N is the number of documents (input sequences), $\mathbf{y_n}$ is the label (one-hot vector) of the n-th document, and $\mathbf{x_n}$ is the normalized bag of features of the n-th document.

The optimization of this objective function is achieved using SGD.

Classification

To classify an input, we can use the following formula:

$$\hat{\mathbf{y}} = softmax(\mathbf{BAx})$$

where $\mathbf{B}$ and $\mathbf{A}$ are weight matrices, x is an input vector (normalized bag of features), and $\hat{\mathbf{y}}$ denotes the probability distribution over the classes that occur in the training data.

The computation of softmax can be very expensive. Therefore hierarchical softmax is used to speed up the process.

2.3.7 Language models

Language models are artificial intelligence algorithms trained to predict and generate text given some context. These models are designed to understand the structure and patterns of language to generate coherent, relevant and grammatically correct text [2].

Language models are built using large amounts of text data, such as books, articles, and mainly online content, such as Wikipedia or CommonCrawl, and are trained using machine learning techniques. They use statistical patterns and algorithms to analyze and learn from this data and then generate new text based on the patterns they have identified. Language models have a wide range of applications, from chatbots and virtual assistants to machine translation and text summarization [2].

RNN language models

Recurrent neural network language models are a revolutionary concept because, in contrast to FNN LMs [23], they don't need a fixed context length [9]. In theory, an arbitrarily long sequence can be stored in the hidden state. In practice, it is not that simple due to the exploding/vanishing gradient problem [11]. This problem can be partially solved using a more complex RNN architecture such as LSTM [12].

Chapter 3

Related work

Even though, in some situations, it may seem that toxicity recognition can be solved by a simple blacklist of banned words that imply toxicity, it is not the case due to various reasons [24]. The main ones include altering words to fool the blacklist (e.g. @ss, f!ck etc.), implicit toxicity, sarcasm, context or cultural differences (some words may be offensive or hateful only in certain cultures) [25] [26]. From these examples, it is evident that a blacklist would not be a sufficiently robust solution. Therefore, it is vital to introduce solutions that are smarter and more effective.

Many papers focused on the topic of toxicity recognition in the past and presented different approaches and solutions such as decision trees [27], TF-IDF, POS and custom linguistic features [28], unsupervised fastText [29], supervised [29] or semi-supervised [26] deep learning, multi-task learning [30] or language models [29] [31] [32].

Our work focuses on supervised fastText to show that results comparable to SOTA feature-based approaches are achievable with low dimensional embeddings and high training efficiency. We believe this will be the case because supervised fastText was already shown to perform very well compared to SOTA [22]. Besides fastText, we experiment with training specialised RNN language models that can learn contextual information. We believe that a sense of word order and context of words can help the model make more informed decisions.

Besides designing the best possible classifier, there are other challenges present in the task. The biggest ones include data collection and annotation and the definition of categories to detect. The following sections will address these problems and present an overview of the most widely used datasets.

3.1 Definition of toxic content

Intuitively, toxic content can refer to any harmful, offensive, or dangerous content to individuals, groups, or society as a whole. However, it lacks a standardised definition.

There are several categories into which toxic content can be split, but no generally followed typology exists. Moreover, the distribution of classes is highly dependent on their definitions and the annotators who assign them. For example, some messages considered hate speech by [33] were only considered offensive by [28] [34].

3.1.1 Hate speech

Similarly to toxic content, hate speech has no official, universally agreed-on definition [24] [34]. Still, there is a general consensus that it is speech that targets disadvantaged racial, religious, ethnic, national or other social groups in a manner that is potentially harmful to them [35].

Since we decided to use the dataset collected by Davidson et al. in 2017 [28], we also use their definition of hate speech, which is the following:

"Language that is used to express hatred towards a targeted group or is intended to be derogatory, to humiliate, or to insult the members of the group."

This definition is similar to those of Twitter [36], Meta (Facebook) [37], or OpenAI [38].

Aside from the fact that hate speech is hard to define, it is also often highly subjective, which brings further complicates its automated detection. For example, in [28], 5% of the samples in the whole dataset were labelled as hate speech, and only 1.3% of all samples were decided unanimously.

3.1.2 Offensive speech

It is important to distinguish offensive speech from hate speech because there are many situations in which vulgar and toxic language is part of regular communication that often does not need to be censored or processed the same way as hate speech because it is not meant to insult anyone. These situations include but are not limited to African-American people using the word n*gga in everyday communication [39], users quoting explicit song lyrics [28] and curse words being part of a joke or used to emphasise a strong emotion [40].

Hate speech
No police department that hires gypsies, half-breeds, and nigglets should be trusted.
<user> Hit a bitch it's not like you can see the bruises
You have my word we will RT @JRwrz: @Thotcho beat those fucking faggots pls
Offensive speech
Im just a young niggah tryna live my life long
<user> nah he pussy he won't
Bad bitch good head I think she ah keeper !

Table 3.1: Illustration of the difference between hate speech and offensive speech. The examples are taken from [28].

3.1.3 Explicit and implicit toxicity

Toxicity is not always implied by the choice of words. In some cases, the classified text might consist of words that can be used in non-toxic contexts. The usage of such context-dependent words makes it harder for both annotators and machines to detect toxicity [34]. Examples of such text are in table 3.2.

Explicit toxic speech
Then i fuck yo bitch
You like them bitch niggas.
this shit so trash lol
Implicit toxic speech
<user> ugly emo trash go cut
#SomethingIGetAlot Are you... asian? black? Hawaiian? gay? retarded? drunk?
People who live in #theNetherlands are unwashed trash.

Table 3.2: Illustration of the difference between explicit and implicit toxic speech. The examples are taken from [28].

3.2 Datasets

To this day, there is no dataset that could be used as a general benchmark for classifier evaluation. The reason is closely tied to the previously described problem of defining what categories of toxicity to recognize and how to define them [34].

Most available datasets don't only have two categories but rather aim to detect various types of toxicity. Several previous works overcame this issue by binarizing the datasets into two classes - sensitive (toxic) and normal [29] [26]. This solution allows the authors to test their classifiers in a unified environment, but it needs to be noted that it dramatically simplifies the

classification task. Another method that allows the usage of datasets that do not have the same classes is multi-task learning. Using multi-task learning to improve the robustness and out-of-domain performance was proposed in [30].

A further problem of commonly used datasets is bias against minorities such as racial [41] [42] or gender [43] bias. It has been shown that there is a clear correlation between samples written in African-American English and samples classified as offensive or abusive [43] and that there are often fewer examples of offensive or hateful speech written by white people than people of colour. Therefore, models trained on such datasets are more likely to be biased and, in the worst-case scenario, might discriminate against the minority groups they were meant to protect [42]. It is not the scope of our work to solve this problem, but it is essential to be aware of the possibility of bias in datasets.

Davidson et al. 2017 Twitter dataset

The data collection process of this dataset began with a lexicon of hate speech compiled by `hatebase.org`. 85.4 million tweets containing terms from the lexicon were extracted using the Twitter API. Then a set of 25k tweets was sampled randomly, and each was labelled by at least three annotators using crowdsourcing. The resulting categories are hateful, offensive (but not hateful), and neither [28].

The dataset is imbalanced because of the usage of hateful and offensive terms from `hatebase.org` during data retrieval.

Founta et al. 2018 Twitter dataset

This dataset aims to overcome some of the shortcomings of previously created datasets. These include

- Difficulty of annotators to distinguish between categories
- Different occurrence rates for different classes
- Scaling up the multi-label annotation process while maintaining the quality of annotation and time-budget constraints.

To address these challenges, the authors used a three-round annotation process. In the first two rounds, they allowed the annotators to use more than one label (category of toxic behaviour). The categories were offensive, abusive, hateful, aggressive, cyberbullying, spam and normal, and each tweet was labelled by at least five annotators. After these two rounds, they determined correlations between the categories and modified the goal labels accordingly. They decided to merge their original abusive, offensive and aggressive categories into abusive and remove cyberbullying altogether [44].

The final categories in the dataset are hateful, abusive, normal and spam. Besides spam, the definitions are similar to [28] because the abusive class contains various types of toxicity (but not hate speech), which can also be said about the offensive class from [28].

The spam class was previously discarded in several papers since there are specialized classifiers to detect spam, and the class is not closely related to the task of toxicity recognition [45].

Other datasets

There are many different datasets for toxicity recognition [24] that detect various subcategories of toxicity such as racism and sexism [33] [46], hate speech against immigrants and women [47], trolling [48] [49], targets of insults [50] [51] or types and levels of toxicity [52].

Most of the hate speech and toxicity datasets come from Twitter [33] [46] [28] [50] [53], but other sources include Wikipedia [52], Facebook [54], Reddit [26] [55], Instagram [56] [57] or niche forums such as Gab [55] [51] and Stormfront [58].

Some of the datasets were also created for the purposes of a public competition. For example, the Toxic Comment Classification Challenge held on Kaggle by Google Jigsaw aimed to distinguish between various categories of toxicity and allowed multiple labels per input [52]. Another such dataset is OLID which is annotated on three levels - detection, type and target. Each of these levels was a separate task in the OffensEval 2019 competition [50] [59].

In table 3.3, we describe some of the most popular datasets. The table clearly illustrates that it is not simple to create a benchmark corpus for the task at hand because it is very broad and is often split into different subtasks.

Dataset	Class distribution	Source	Sample count
Davidson et al. [28]	Hate 6% Offensive 77% Neither 17%	Twitter	25k
Founta et al. [44]	Hateful 5% Abusive 27% Normal 54% Spam 14%	Twitter	100k
Kaggle (Jigsaw) [52]	Toxic 10% Severe toxic 1% Obscene 5% Threat 0.3% Insult 5% Identity hate 1% Neither 90%	Wikipedia	222k
SemEval 2019 (Task A) [50]	Offensive 33% Not offensive 67%	Twitter	14k
Waseem and Hovy [33]	Racism 12% Sexism 20% Neither 68%	Twitter	17k
Waseem [46]	Racism 1% Sexism 12% Neither 86% Both 1%	Twitter	7k
De Gilbert et al. [58]	Hate 11% Not Hate 86% Relation 2% Skip 1%	Stormfront	10k

Table 3.3: Comparison of some of the most widely used and cited datasets for toxicity and hate speech recognition.

Chapter 4

Method

In this chapter, we will describe the methods that were used in our experiments to achieve the presented results.

4.1 Supervised fastText

Supervised fastText has been shown to work very well for typical text classification tasks such as sentiment analysis [22]. Since sentiment might imply toxicity in some cases and the task of toxicity recognition is similar to sentiment analysis overall, we believe that it will work well for our task as well.

4.1.1 Concatenation of sentence embeddings

To improve the robustness of a supervised fastText classifier, we experimented with concatenating the embeddings of a given sentence from several supervised fastText models trained on different dataset variations.

The motivation for experimenting with this approach is the assumption that even if one model from the ensemble doesn't produce a very good sentence vector, the concatenated embedding can still be saved by other models that work better in the situation.

Description of the classifier

To get a sentence embedding, we first need to train several models on different datasets. Then we can feed a sentence to each classifier independently and concatenate the individual sentence embeddings to form one final embedding which contains more information than the individual ones.

After creating the final embeddings, a classifier can be trained and used for predictions on a test set. We use multinomial logistic regression because it is a strong baseline classifier often used in NLP [2] and is also used in supervised fastText classification [22]. However, any feature based classifier could be used instead.

There are other ways that could be used to combine embeddings from the classifiers (e. g. average or weighted average). However, the advantage of concatenation is the fact that the embeddings can be of arbitrary lengths. This allows the algorithm to use different sentence embedding methods easily because there is no need to make the vectors all the same length.

The process of embedding one sentence can be described by the following pseudocode:

Algorithm 1 Function used to acquire a sentence embedding

```
function GET_SENTENCE_EMBEDDING(sentence)
    e ← []
    for each c ∈ C do
        e ← concatenate(e, c.get_sentence_vector(sentence))
    end for
    return e
end function
```

where C is a set of supervised FT models that can be used to embed a sentence, and e is the final embedding.

It can also be illustrated by the schematic in figure 4.1

4.2 RNN language model for text classification

In this section, we describe a modification of the recurrent neural network language model architecture [9], that can be used for text classification,

This approach aims to train a language model on a slightly modified dataset where each line contains a sequence of words followed by a unique end-of-sentence token and a label. The training data is then processed line by line.

This way, the model should learn that a label must come after an end-of-sentence token. And even if the model predicts a different word than a label as the most likely one, it is possible to pick the highest of the probabilities for all possible classes.

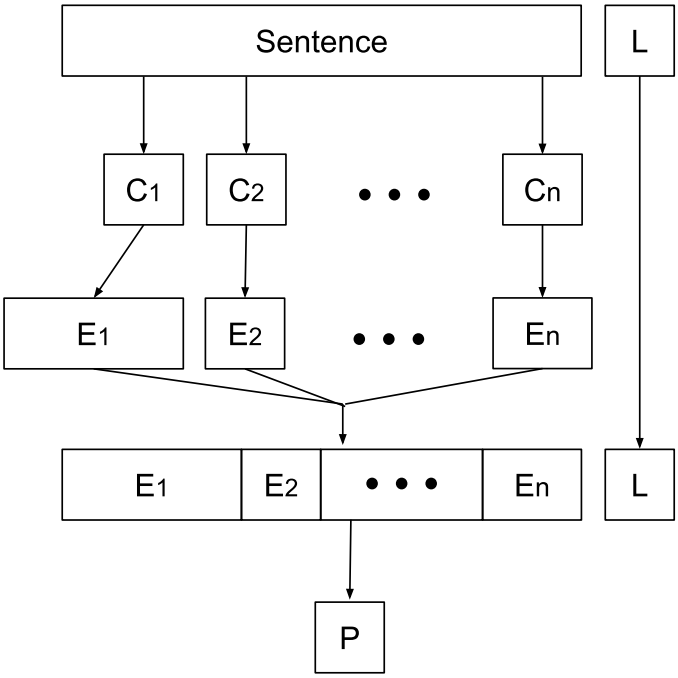


Figure 4.1: In the schematic, we can see the process of acquiring a sentence embedding. First, a sentence is fed to n individual classifiers ($C_1 - C_n$), which generate n arbitrarily long sentence embeddings ($E_1 - E_n$). In the next step, these embeddings are concatenated to form one final embedding of the input sentence. This embedding can later be used to train and evaluate a classifier (e.g. logistic regression).

4.2.1 Loss calculation

Instead of calculating the training loss after each word based on the next word as is done in [9], we only calculate the loss of the last predicted word in a sentence (i. e. the label). This way, the model should adjust to the classification task instead of general word prediction.

4.2.2 Bidirectional RNN ensemble

To make the classifier more robust and counter the exploding/vanishing gradient problem, we trained two models on the same dataset, with the difference between them being the word order. In one model, we process an input sequence normally and in the other, we reverse the word order completely, process the sequence from the end and predict a label at the start of a sentence. To make a prediction, we take the prediction scores for each class from both models, average them out and pick the class with the highest average prediction score.

Bidirectional recurrent neural networks [60] have been shown to work well for text classification [2]. Therefore we assume that our bidirectional ap-

proach, which is very similar to bidirectional RNNs, will improve the model's performance, particularly in situations where the words that carry the most information are at the start of a longer sequence because such words will be at the end in the reversed model.

Chapter 5

Experimental setup

In this chapter, we describe our experimental setup - i.e. the libraries used in our implementation, hyperparameters and the process of their tuning, the data used for training and evaluating our classifiers and the hardware used during our experiments.

5.1 Hardware

All of our models were trained on the same personal laptop with Intel(R) Core(TM) i5-4210H 2.90GHz CPU, NVIDIA GeForce GTX 960M GPU and 8GB of RAM.

5.2 FastText

For our experiments with supervised fastText, we used the library available at their website[1]. This library provides an API to use the fastText algorithm for various tasks such as training models, embedding words or sentences and classifying text.

FastText's website offers pre-trained unsupervised word vectors that were trained on Wikipedia and CommonCrawl [61] and can be used as a starting point for supervised embeddings. The dimension of these vectors is 300, but fastText provides a guide to reduce the dimension. Thanks to that, we were able to adjust these pre-trained vectors for our experiments to fit our chosen dimensions.

To classify the fastText embeddings acquired by our method, we used logistic regression implemented in *scikit-learn* [62].

[1] `https://fasttext.cc/`

5.2.1 Hyperparameters

For supervised fastText, the hyperparameters we tuned are the following:

- lr - learning rate
- epoch - the number of epochs
- dim - dimension of the hidden layer
- minn/maxn - mininmal/maximal length of character n-grams
- wordNgrams - maximal length of word n-grams

To help with the process of hyperparameter tuning, we used the autotune validation feature that is built into the fastText library. This feature tries to optimise the hyperparameters based on a validation dataset automatically. In practice, we found that in some cases, the hyperparameters returned by this method can be improved by further hand tweaking, such as adding more epochs or modifying the learning rate slightly.

During the hyperparameter search, we constrained the maximum model size to 50MB. This functionality is also implemented in the fastText library.

Dataset	pretrainedVectors	lr	epoch	minn	maxn	wordNgrams
Original	None	0.05	10	0	0	0
Balanced	wiki.simple	0.05	1	3	6	3
Full	wiki.simple	0.05	25	3	6	4

Table 5.1: Hyperparameters of the best-performing supervised FT models we trained.

5.3 RNN language model for text classification

The implementation of RNN LM written in PyTorch [63] that we modified for the purposes of text classification was written by David Herel [2], who plans to release it publicly but will not be able to do so before the submission date of this thesis.

5.3.1 Hyperparameters

For RNN language models, the most important hyperparameters include the following:

[2] `https://davidherel.com/`

- learning rate [0.1 - 1]
- number of epochs [5 - 20]
- dropout [0 - 0.3]
- size of hidden layer [100 - 400]
- number of hidden layers [1 - 3]

We experimented with various combinations of hyperparameters to achieve the best possible results on our test data. The ranges of hyperparameters we tried are in square brackets. The hyperparameters of our models could be tuned further in order to achieve even better results, but since we experimented with many variations, we believe the difference in performance would not be significant.

Dataset	lr	epoch	dropout	hidden size	layers
Balanced	0.3	4	0	200	2
Balanced (reverse)	0.2	10	3	200	2

Table 5.2: Hyperparameters of the best supervised RNN models we trained.

Method	Training time	Training data
Supervised fastText	0.6s (10m)	Original
Concatenated FT embeddings + logistic regression	32s (30m)	All datasets
RNN	28m	Balanced
Bidirectional RNN ensemble	88m	Balanced

Table 5.3: Approximate training times of our classifiers. In brackets, we include the time we dedicated to autotuning hyperparameters for FT models. All of the times were measured on the same device.

5.4 Dataset

Although the task of toxicity recognition lacks a universal benchmark dataset, a few stand out as the most widely used ones. These include the dataset of 25k tweets created in 2017 by Davidson et al. [28].

This is our primary dataset of choice because it focuses on the difference between hate speech and offensive speech and provides clear and sufficient definitions for these classes. It is also important for our work that the categories in the dataset are general (i. e. not focused on a particular category of toxicity such as racism, threats or targeted insults). It is also often used by other authors, and therefore, we were able to find results to compare ours with.

The second dataset we decided to use was created by Founta et al. [44] and is suitable for our use case because it provides us with further examples of all three recognised classes. Most importantly, hate speech. As stated above, the definitions of abusive speech in [44], and offensive speech in [28] are similar. Therefore it is ideal for our experiments with data augmentation.

We split the data into train, test and validation sets. The ratio we used is 70% train, 20% test and 10% validation.

5.5 Data augmentation

Using more data has proven to be an efficient method for improving the performance of a hate speech classifier [26] [30]. In this thesis, we experiment with using an additional, larger dataset which distinguishes classes that are very similar to the original ones. This approach allows us to use external annotated data without the need to perform semi-supervision. However, the data are still of lower quality because their retrieval and annotation process was slightly different compared to the original dataset.

We believe that thanks to this method, we will be able to improve the performance and robustness of our classifiers because we will show them more data (even though the data quality won't be as high as before). This is especially important for the hate speech label that is present in only ca. 5% of the original dataset. After training our classifiers on an augmented dataset, we will evaluate them on the original test data to make them directly comparable to the ones trained on the original dataset.

Using this data augmentation process, we have created two new datasets - balanced and full. A comparison of these new datasets to the original one is shown in table 5.4.

Balanced

The first of our datasets is balanced, meaning that the amount of samples from each class is approximately the same. This was achieved by using all available hateful tweets from both datasets and an equal amount (randomly selected) of normal tweets from both datasets and offensive tweets from the original dataset. We aim to show that if a classifier sees more data from each class, it should learn to recognize better those classes that were not as frequent in the original imbalanced dataset. Size-wise, this dataset is comparable to the original one.

Full

In contrast to the balanced dataset, the full one uses all the data from both datasets. It aims to show the difference in results that can be achieved by adding ca. five times more data. Similarly to the original one, this dataset is strongly imbalanced.

Data	# Samples	% Hateful	% Offensive	% Normal	Median len.
Original	17347	5.74	77.28	16.98	7/13
Balanced	16092	33.54	33.38	33.08	9/15
Full	92982	5.74	39.18	55.09	9/16

Table 5.4: Comparison of our training datasets. The median length column contains values for fastText and RNN LM preprocessing, respectively.

5.6 Data preprocessing

Before classifying the data using our proposed methods, we preprocessed it to make it more readable for the classifiers. The preprocessing steps are described in this section.

5.6.1 Preprocessing for fastText

For supervised fastText (and bag of features methods in general), the order of words is ignored. Therefore besides standard preprocessing, we also remove stopwords, user mentions and rt strings. The whole process is the following:

1. Remove multiple spaces
2. Remove URLs
3. Remove user mentions
4. Split camel case after # (i. e. "#helloWorld" becomes "hello World")
5. Remove "RT" string (denotes retweet)
6. Remove all special characters
7. Make all text lowercase
8. Remove English stopwords (using a list from NLTK [64])

5.6.2 Preprocessing for RNN

The preprocessing for our RNN approach differs from what we use for supervised fastText because, as opposed to FT, RNNs can capture context and word order. Therefore the classifier could benefit from more information in the data. We use the following preprocessing procedure:

1. Remove multiple spaces
2. Remove URLs
3. Substitute user mentions with a <user> token
4. Split camel case after # (i. e. "#helloWorld" becomes "hello World")
5. Remove all special characters
6. Make all text lowercase

Besides preprocessing the text, we also modified the data so that it could be used as input for our RNN LM variation.

Each document from the training data was put on a separate line followed by a unique end-of-sentence token (<eos>) and a label (0, 1 or 2 for hate speech, offensive speech and normal speech, respectively).

We also had to use padding tokens (<pad>) to make all sentences uniformly long because the code is written in PyTorch [63] in order to run on a GPU.

5.7 Metrics

In this section, we will explain basic metrics for classification. The notation used in the formulas for two classes (positive and negative) is explained in the following list.

- TP (true positives): The number of samples in the test dataset correctly classified positive.
- TN (true negatives): The number of samples in the test dataset correctly classified as negative.
- FP (false positives): The number of samples in the test dataset incorrectly classified as positive.
- FN (false negatives): The number of samples in the test dataset incorrectly classified as negative.

Accuracy

Accuracy measures the ratio between the absolute amount of correct predictions and the number of samples. It is a good baseline metric. However, it has its shortcomings. Especially for unbalanced datasets, it might be misleading. For example, given a dataset with two classes that consists of 95% class 0 and 5% class 1, the accuracy would be 95% if the classifier classified every input as class 0, which seems relatively high. However, it is evident that such a classifier would not work correctly for the input data in practice because it would completely overlook the minority class. Therefore other metrics are often used to ensure that a classifier works as expected.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision

This metric measures the proportion of correctly classified instances among those that were predicted as positive.

$$Precision = \frac{TP}{TP + FP}$$

Recall

Recall is the proportion of correctly classified instances among all actual positive instances.

$$Recall = \frac{TP}{TP + FN}$$

F1 score

F1 score is the harmonic mean of precision and recall and provides a single metric that balances the two.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN}$$

5.7.1 Multiclass metrics

All the above-mentioned metrics are defined for only two classes - positive and negative. In practice, it is often necessary to recognise more than just two

classes. Therefore various means of the metrics are used. We will only describe the macro average F1 score in detail since it is the only metric the reader needs to know to understand the results in the next section correctly.

Macro average F1 score

Macro average F1 score is the arithmetic average over all classes in a dataset. We decided to use this metric because it has been shown to work better than micro average F1 on unbalanced datasets [65]. The following formula is used to calculate this metric:

$$F1_{macro} = \frac{1}{K} \sum_{k=i}^{K} F1_k$$

where K denotes the number of classes to classify and $F1_k$ denotes the $F1$ score for the k-th class.

Chapter 6

Results

In this chapter, we present the results that were achieved using our proposed methods. First, we compare our methods to other publications, and then we analyse each of our methods in depth in order to highlight their advantages and disadvantages.

It is complicated to compare our achieved results to other existing solutions because most papers report results on their train-test-validation splits that are not publicly available. However, even though our reported results were achieved on different splits, it is possible to compare them to other solutions to get a general understanding of how well they perform.

In table 6.1, we present a comparison of our best results and results that were published in other works that use the same dataset. We have achieved results comparable to other embedding-based methods with a much lower embedding dimension. In [30], the authors (similarly to us) employed an additional dataset in order to make their model more robust. However, their scope was to improve the model's performance on out-of-domain data, whereas our motivation for augmenting the data was a better performance on the same single task.

The only feature-based method that outperforms ours by a significant margin uses custom, Twitter-specific features along with some commonly used ones such as TF-IDF, POS and word count. This method is effective, but in contrast to ours, it requires a lot of feature engineering. Moreover, it is only usable for Twitter data because it employs platform-specific features.

The BERT finetuning method presented in [29] has also outperformed ours significantly. However, it can be argued that finetuning a large language model such as BERT requires a lot of time and computational resources as opposed to fastText models that can be trained in less than a minute on a single CPU and even our bidirectional RNN models that can be trained in approximately two hours on a single GPU (see table 5.3).

It is important to note that even though we chose to classify the data into three classes to separate hateful speech from offensive speech, the most essential

part of the task is the detection of toxicity as a whole. For this reason, we present a confusion matrix that was generated by using our best classifier to predict labels normally and merging both toxic classes into one. In figure 6.5, we can see that the classifier performs well. It missclassified 8.7% of toxic tweets and 4.2% of normal tweets from the test set.

Method	Macro F1	Dimension
Multiple FT models (ours)	0.722	30
Bidirectional RNN classifier (ours)	0.696	
BERT embeddings + Bi-LSTM [29]	0.724	768
Unsupervised FT embeddings + Bi-LSTM [29]	0.723	300
BERT finetuning [29]	0.84	
TF-IDF, POS, custom features [28] [1]	0.773	170
Multitask learning and BoW features [30] [2]	0.723	

Table 6.1: Comparison of our results with other similar works.

6.1 Supervised fastText

This section contains a selection of the best results that we achieved using supervised fastText. In table 6.2, we can see an overview of the best fastText models that we trained. We see that even though the macro F1 score is very similar on all variations of the data, it can be improved using our proposed concatenation method.

In table 6.2, we can also see that even though multiple models outperformed each one of the single models, the accuracy has lowered (in comparison to the best performing model). However, if we look at figures 6.1 and 6.2, we can see a very clear improvement in terms of the amount of correctly classified hate speech tweets. We believe that recognizing as much hate speech as possible is crucial in most situations, and therefore, we consider our method to be better than baseline supervised FT classifiers.

6.1.1 Single model

Using supervised fastText alone, we were able to achieve macro average F1 score of 0.702, which is a result that is comparable (even though it performs slightly worse) to other feature based approaches such as [30] or [29]. From

[1] The paper does not report macro F1 score on their test split, but we were able to calculate it based on their codebase that is available at `https://github.com/t-davidson/hate-speech-and-offensive-language`

[2] The paper only reports precision and recall. We calculated the macro F1 score ourselves.

Dataset	Pre-trained vectors	Dimension	Macro F1	Accuracy
Original	None	10	0.702	0.892
Balanced	Yes	10	0.701	0.849
Full	Yes	10	0.702	0.886
All datasets	In some models	30	**0.722**	0.859

Table 6.2: Comparison of the models used for the final FT classifier.

the normalised confusion matrix depicted in figure 6.1, it is evident that the classifier can not distinguish hate speech from offensive speech very well. This is a problem that we can also see in other works that use a feature based approach [29] [30].

	precision	recall	f1-score	support
Hate	0.463	0.255	0.329	290
Offensive	0.926	0.946	0.936	3832
Neither	0.819	0.867	0.842	835
accuracy			0.892	4957
macro avg	0.736	0.689	0.702	4957

Table 6.3: Results achieved using a single supervised fastText model trained on the original dataset.

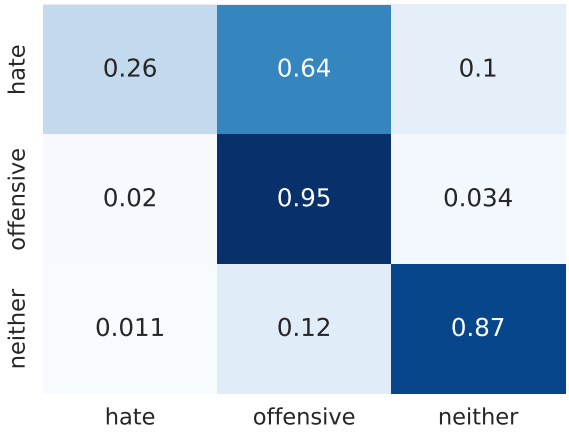


Figure 6.1: Normalised confusion matrix of the classifications made by our best fastText classifier.

6.1.2 Multiple models

Using the proposed method of concatenating supervised fastText embeddings and classifying them with logistic regression, we improved the macro F1 score

to 0.722. This result is comparable to [29], where the authors achieved macro F1 score of 0.724 using BERT-based embeddings and a bidirectional LSTM classifier. Even though our score is slightly lower, the computation time and memory needed to train and use our classifier are significantly lower.

Figure 6.2 depicts the confusion matrix of our best multi-model classifier. It is clear that this classifier works much better for classifying hate speech, and the amount of correctly classified hate speech inputs is comparable to finetuned BERT published in [29]. However, there still is a significant amount of samples that were annotated as hate speech and classified as offensive speech.

	precision	recall	f1-score	support
hate	0.296	0.579	0.392	290
offensive	0.964	0.868	0.913	3832
neither	0.814	0.915	0.861	835
accuracy			0.859	4957
macro avg	0.691	0.787	0.722	4957

Table 6.4: Results achieved using a combination of three supervised fastText models trained on different variations of the original dataset.

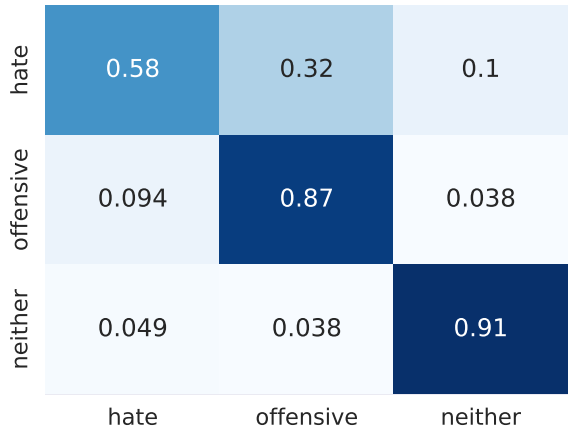


Figure 6.2: Normalised confusion matrix of our supervised fastText-based classifier that uses multiple FT models and is trained on augmented datasets.

6.2 RNN language model for text classification

This section presents the best results we achieved using the previously described method based on RNN LM.

In table 6.5, we can see that this approach was less effective than supervised fastText. Moreover, in comparison to fastText, the training time increased drastically (from seconds to over an hour, see table 5.3), and the number of hidden neurons increased as well from 30 to 200.

Even though supervised fastText outperforms our RNN LM architecture significantly, we can see that the method yields relevant results, and we believe that it could be improved to match or outperform fastText.

	precision	recall	f1-score	support
hate	0.273	0.500	0.353	290
offensive	0.968	0.856	0.909	3832
neither	0.725	0.898	0.803	835
accuracy			0.843	4957
macro avg	0.655	0.752	0.688	4957

Table 6.5: Results achieved using our proposed RNN classification model. The model was trained on the balanced dataset variant.

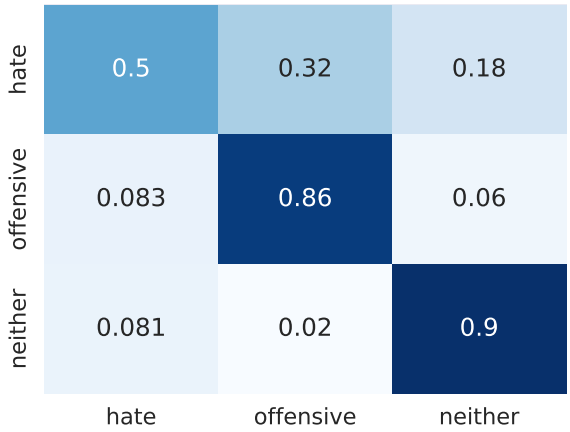


Figure 6.3: Normalised confusion matrix of the classifications made by our best RNN classifier.

6.2.1 Bidirectional ensemble

The first improvement of our RNN architecture is training a second model with reverse word order and using an ensemble of these two models to classify data. In table 6.6, we see that the performance of the classifier did improve in comparison to a single model (table 6.5).

The best macro F1 score for normal order and reverse order models are 0.688 and 0.676, respectively. From that, it is evident that our method works as expected and combines both models to form a better classifier.

	precision	recall	f1-score	support
hate	0.283	0.548	0.373	290
offensive	0.970	0.866	0.915	3832
neither	0.743	0.868	0.801	835
accuracy			0.848	4957
macro avg	0.665	0.761	0.696	4957

Table 6.6: Results achieved on our best RNN ensemble.

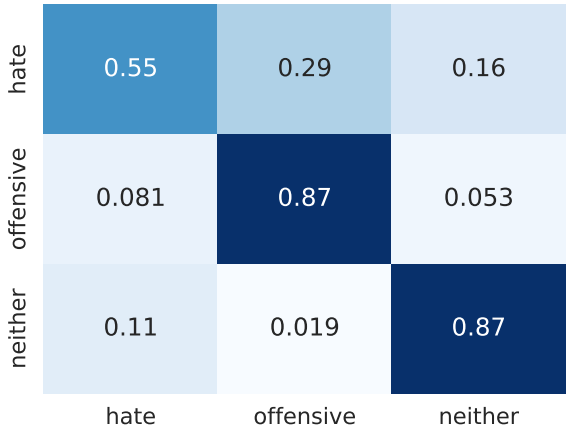


Figure 6.4: Results achieved using our proposed bidirectional ensemble of RNN classifiers. The models were trained on the balanced dataset variant.

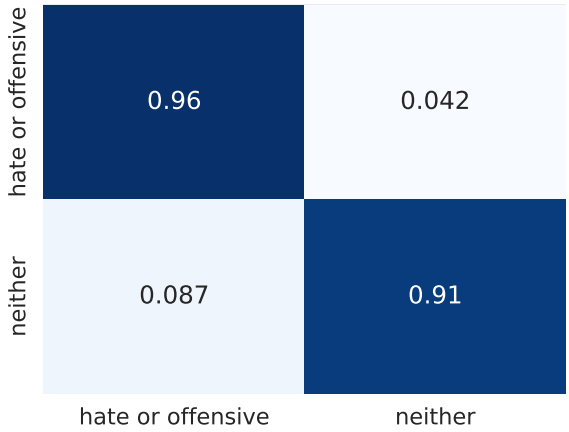


Figure 6.5: Normalised confusion matrix of the classifications made by our best classifier. Both toxic classes are merged in this figure.

6.3 Future work

Even though our classifiers return relevant results, we believe they could be improved further.

First of all, we believe that all of our models would benefit from even more hate speech data because it would help them better distinguish between hate speech and offensive speech. This assumption is based on the fact that even though we trained models on the full dataset, which contained significantly more offensive and normal examples, the amount of hate speech remained the same as in the balanced data. The resulting models performed comparably or worse than those trained on the balanced dataset. We assume that this was caused by the strong imbalance of classes present in the dataset.

More hate speech data could be gathered from other existing datasets or by manually annotating further examples. The latter would be more expensive and time intensive but likely more effective because we could control the annotation process and definitions of classes. Controlling the annotation process could also help to eliminate or at least significantly reduce the bias that is present in currently available datasets [41] [43] [42].

Besides collecting more training data, we believe that finetuning an RNN LM could also be a viable solution because BERT finetuning has been shown to perform significantly better than fastText and other feature-based approaches [29]. However, it needs to be noted that BERT was developed for finetuning [15], and the process of finetuning an RNN language model would not be as straightforward.

Chapter 7

Conclusion

In this thesis, we focused on toxicity recognition. We explained the theory needed to understand the field of machine learning, natural language processing and text classification in general, as well as the details of algorithms used throughout this work.

We reviewed related work in the area of toxicity recognition, discussed the issue of defining the concept of toxicity in conversational systems and some possible subcategories of toxicity, and compared several of the most widely used datasets for the task.

After describing related work in the field, we proposed two methods that can be used for toxicity recognition - one based on supervised fastText embeddings and one based on recurrent neural networks. After explaining our methods, we discussed hyperparameters of our models, datasets, implementation details and metrics to help the reader understand our experimental process and setup.

We presented the results of our experiments, compared them to SOTA results published in other works and proposed possible directions for future work that could improve the performance of our methods.

Our results show that a classifier based on the concatenation of supervised fastText embeddings can be used as a strong baseline for toxicity recognition. Even though LM (BERT) finetuning performs significantly better, it requires considerably more computation. Therefore, we believe that our method is relevant and could be used when it is desirable to train and classify quickly with low memory usage.

We have also shown that RNN language models work reasonably well for toxicity recognition. Although we were not able to outperform fastText with this method, we believe that RNNs could be used to train a high-quality classifier. In this case, the first step in the right direction could be one of the suggestions we presented in section 6.3..

Appendix A

Bibliography

[1] E. Dinan, S. Humeau, B. Chintagunta, and J. Weston, "Build it break it fix it for dialogue safety: Robustness from adversarial human attack," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019, pp. 4537–4546.

[2] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 3rd ed., 2023.

[3] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

[4] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, 1943.

[5] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[6] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[7] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[9] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model." in *Interspeech*, vol. 2, no. 3. Makuhari, 2010, pp. 1045–1048.

[10] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[11] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International conference on machine learning*. Pmlr, 2013, pp. 1310–1318.

[12] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[14] OpenAI, "Gpt-4 technical report," 2023.

[15] J. D. M.-W. C. Kenton and L. K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of NAACL-HLT*, 2019, pp. 4171–4186.

[16] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

[17] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the association for computational linguistics*, vol. 5, pp. 135–146, 2017.

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

[19] T. Mikolov, W.-t. Yih, and G. Zweig, "Linguistic regularities in continuous space word representations," in *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*, 2013, pp. 746–751.

[20] G. Finley, S. Farmer, and S. Pakhomov, "What analogies reveal about word vectors and their compositionality," in *Proceedings of the 6th joint conference on lexical and computational semantics (* SEM 2017)*, 2017, pp. 1–11.

[21] M. Nissim, R. van Noord, and R. van der Goot, "Fair is better than sensational: Man is to doctor as woman is to doctor," *Computational Linguistics*, vol. 46, no. 2, pp. 487–497, 2020.

[22] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Valencia, Spain: Association for

Computational Linguistics, Apr. 2017, pp. 427–431. [Online]. Available: https://aclanthology.org/E17-2068

[23] Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," *Advances in neural information processing systems*, vol. 13, 2000.

[24] S. MacAvaney, H.-R. Yao, E. Yang, K. Russell, N. Goharian, and O. Frieder, "Hate speech detection: Challenges and solutions," *PloS one*, vol. 14, no. 8, p. e0221152, 2019.

[25] S. O. Sood, E. F. Churchill, and J. Antin, "Automatic identification of personal insults on social news sites," *Journal of the American Society for Information Science and Technology*, vol. 63, no. 2, pp. 270–285, 2012.

[26] C. Khatri, B. Hedayatnia, R. Goel, A. Venkatesh, R. Gabriel, and A. Mandal, "Detecting offensive content in open-domain conversations using two stage semi-supervision," *arXiv preprint arXiv:1811.12900*, 2018.

[27] E. Spertus, "Smokey: Automatic recognition of hostile messages," in *Aaai/iaai*, 1997, pp. 1058–1065.

[28] T. Davidson, D. Warmsley, M. Macy, and I. Weber, "Automated hate speech detection and the problem of offensive language," in *Proceedings of the 11th International AAAI Conference on Web and Social Media*, ser. ICWSM '17, 2017, pp. 512–515.

[29] A. G. d'Sa, I. Illina, and D. Fohr, "Bert and fasttext embeddings for automatic detection of toxic speech," in *2020 International Multi-Conference on:"Organization of Knowledge and Advanced Technologies"(OCTA)*. IEEE, 2020, pp. 1–5.

[30] Z. Waseem, J. Thorne, and J. Bingel, "Bridging the gaps: Multi task learning for domain transfer of hate speech detection," *Online harassment*, pp. 29–55, 2018.

[31] T. Caselli, V. Basile, J. Mitrović, and M. Granitzer, "HateBERT: Retraining BERT for abusive language detection in English," in *Proceedings of the 5th Workshop on Online Abuse and Harms (WOAH 2021)*. Online: Association for Computational Linguistics, Aug. 2021, pp. 17–25. [Online]. Available: https://aclanthology.org/2021.woah-1.3

[32] M. Mozafari, R. Farahbakhsh, and N. Crespi, "A bert-based transfer learning approach for hate speech detection in online social media," in *Complex Networks and Their Applications VIII: Volume 1 Proceedings of the Eighth International Conference on Complex Networks and Their Applications COMPLEX NETWORKS 2019 8*. Springer, 2020, pp. 928–940.

[33] Z. Waseem and D. Hovy, "Hateful symbols or hateful people? predictive features for hate speech detection on twitter," in *Proceedings of the NAACL student research workshop*, 2016, pp. 88–93.

[34] Z. Waseem, T. Davidson, N. Ithica, D. Warmsley, and I. Weber, "Understanding abuse: A typology of abusive language detection subtasks," *ACL 2017*, p. 78, 2017.

[35] S. Walker, *Hate speech: The history of an American controversy.* U of Nebraska Press, 1994.

[36] "Twitter help center - hateful conduct," accessed: 2023-18-04. [Online]. Available: https://help.twitter.com/en/rules-and-policies/hateful-conduct-policy

[37] "Meta transparency center - hate speech," accessed: 2023-18-04. [Online]. Available: https://transparency.fb.com/en-gb/policies/community-standards/hate-speech/

[38] "Openai documentation - moderation," accessed: 2023-24-04. [Online]. Available: https://platform.openai.com/docs/guides/moderation/overview

[39] S. Stephens-Davidowitz, "The effects of racial animus on voting: Evidence using google search data," *Unpublished typescript*, 2011.

[40] W. Wang, L. Chen, K. Thirunarayan, and A. P. Sheth, "Cursing in english on twitter," in *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*, 2014, pp. 415–425.

[41] M. Sap, D. Card, S. Gabriel, Y. Choi, and N. A. Smith, "The risk of racial bias in hate speech detection," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics.* Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 1668–1678. [Online]. Available: https://aclanthology.org/P19-1163

[42] T. Davidson, D. Bhattacharya, and I. Weber, "Racial bias in hate speech and abusive language detection datasets," in *Proceedings of the Third Workshop on Abusive Language Online*, 2019, pp. 25–35.

[43] J. H. Park, J. Shin, and P. Fung, "Reducing gender bias in abusive language detection," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing.* Brussels, Belgium: Association for Computational Linguistics, Oct.-Nov. 2018, pp. 2799–2804. [Online]. Available: https://aclanthology.org/D18-1302

[44] A.-M. Founta, C. Djouvas, D. Chatzakou, I. Leontiadis, J. Blackburn, G. Stringhini, A. Vakali, M. Sirivianos, and N. Kourtellis, "Large scale crowdsourcing and characterization of twitter abusive behavior," in *11th International Conference on Web and Social Media, ICWSM 2018.* AAAI Press, 2018.

[45] A. M. Founta, D. Chatzakou, N. Kourtellis, J. Blackburn, A. Vakali, and I. Leontiadis, "A unified deep learning architecture for abuse detection," in *Proceedings of the 10th ACM conference on web science*, 2019, pp. 105–114.

[46] Z. Waseem, "Are you a racist or am i seeing things? annotator influence on hate speech detection on twitter," in *Proceedings of the first workshop on NLP and computational social science*, 2016, pp. 138–142.

[47] V. Basile, C. Bosco, E. Fersini, D. Nozza, V. Patti, F. M. Rangel Pardo, P. Rosso, and M. Sanguinetti, "SemEval-2019 task 5: Multilingual detection of hate speech against immigrants and women in Twitter," in *Proceedings of the 13th International Workshop on Semantic Evaluation*. Minneapolis, Minnesota, USA: Association for Computational Linguistics, Jun. 2019, pp. 54–63. [Online]. Available: https://aclanthology.org/S19-2007

[48] R. Kumar, A. K. Ojha, S. Malmasi, and M. Zampieri, "Benchmarking aggression identification in social media," in *Proceedings of the first workshop on trolling, aggression and cyberbullying (TRAC-2018)*, 2018, pp. 1–11.

[49] J. Golbeck, Z. Ashktorab, R. O. Banjo, A. Berlinger, S. Bhagwan, C. Buntain, P. Cheakalos, A. A. Geller, Q. Gergory, R. K. Gnanasekaran *et al.*, "A large labeled corpus for online harassment research," in *Proceedings of the 2017 ACM on web science conference*, 2017, pp. 229–233.

[50] M. Zampieri, S. Malmasi, P. Nakov, S. Rosenthal, N. Farra, and R. Kumar, "Predicting the Type and Target of Offensive Posts in Social Media," in *Proceedings of NAACL*, 2019.

[51] B. Kennedy, M. Atari, A. Mostafazadeh Davani, L. Yeh, A. Omrani, Y. Kim, K. Koombs, S. Havaldar, G. J. Portillo-Wightman, E. Gonzalez, J. Hoover, A. Azatian, A. Hussain, A. Lara, G. Olmos, A. Omary, C. Park, C. Wang, X. Wang, and M. Dehghani, "The gab hate corpus: A collection of 27k posts annotated for hate speech," 02 2020.

[52] "Toxic comment classification challenge data," accessed: 2023-25-04. [Online]. Available: https://www.kaggle.com/competitions/jigsaw-toxic-comment-classification-challenge/data

[53] F. Poletto, V. Basile, M. Sanguinetti, C. Bosco, and V. Patti, "Resources and benchmark corpora for hate speech detection: a systematic review," *Language Resources and Evaluation*, vol. 55, pp. 477–523, 2021.

[54] F. Del Vigna12, A. Cimino23, F. Dell'Orletta, M. Petrocchi, and M. Tesconi, "Hate me, hate me not: Hate speech detection on facebook," in *Proceedings of the first Italian conference on cybersecurity (ITASEC17)*, 2017, pp. 86–95.

[55] J. Qian, A. Bethke, Y. Liu, E. Belding, and W. Wang, "A benchmark dataset for learning to intervene in online hate speech," 09 2019.

[56] H. Hosseinmardi, S. Arredondo Mattson, R. I. Rafiq, R. Han, Q. Lv, and S. Mishra, "Detection of cyberbullying incidents on the instagram social network," 03 2015.

[57] H. Zhong, H. Li, A. C. Squicciarini, S. M. Rajtmajer, C. Griffin, D. J. Miller, and C. Caragea, "Content-driven detection of cyberbullying on the instagram social network." in *IJCAI*, vol. 16, 2016, pp. 3952–3958.

[58] O. de Gibert, N. Perez, A. García-Pablos, and M. Cuadros, "Hate speech dataset from a white supremacy forum," in *Proceedings of the 2nd Workshop on Abusive Language Online (ALW2)*. Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 11–20. [Online]. Available: https://aclanthology.org/W18-5102

[59] M. Zampieri, S. Malmasi, P. Nakov, S. Rosenthal, N. Farra, and R. Kumar, "Semeval-2019 task 6: Identifying and categorizing offensive language in social media (offenseval)," *arXiv preprint arXiv:1903.08983*, 2019.

[60] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[61] E. Grave, P. Bojanowski, P. Gupta, A. Joulin, and T. Mikolov, "Learning word vectors for 157 languages," in *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[62] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[63] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[64] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. O'Reilly Media, Inc., 2009.

[65] Z. Zhang and L. Luo, "Hate speech detection: A solved problem? the challenging case of long tail on twitter," *Semantic Web*, vol. Accepted, 10 2018.

Appendix B

List of acronyms

ADAM: Adaptive Moment Estimation

AI: Artificial Intelligence

ANN: Artificial Neural Network

BERT: Bidirectional Encoder Representations from Transformers

BoW: Bag of Words

CBOW: Continuos Bag of Words

FT: FastText

GD: Gradient Descent

GPT4: Generative Pretrained Transformer (version 4)

LLM: Large Language Model

LSTM: Long Short Term Memory

ML: Machine Learning

NLP: Natural Language Processing

NLTK: Natural Language Tool Kit

POS: Part of Speech

RNN: Recurrent Neural Network

SGD: Stochastic Gradient Descent

SOTA: State-of-the-art

TF-IDF: Term Frequency-Inverse Document Frequency