Toxic Comment Classification

The Role of Self-Training in Enhancing Model Performance

Bachelor Thesis
Faculty of Science, University of Bern

submitted by

Leroy Borgeaud dit Avocat

from Biel, Switzerland

Supervision:

PD Dr. Kaspar Riesen
Institute of Computer Science (INF)
University of Bern, Switzerland

Abstract

In recent years, social media platforms have become increasingly popular, with comment sections serving as essential spaces to express and discuss opinions and share information. Unfortunately, the anonymity of social media allows malicious users to post toxic comments, disrupting these discussions. While artificial intelligence methods have shown promise in detecting such content, they often rely on a large amount of labeled training data. However, there is a lack of labeled data due to the costly and time-consuming labeling process. While semi-supervised learning demonstrates effectiveness in various NLP tasks, its use in identifying toxic comments is still not well-researched, especially for the German language.

This thesis aims to explore semi-supervised learning as a potential solution for the lack of labeled data. Consequently, we investigate two self-training approaches. First, training algorithms with a combination of labeled and pseudo-labeled data. Second, training algorithms with a combination of labeled data and pseudo-labeled data that is specifically classified as toxic. We evaluate the effectiveness of these approaches using various machine learning algorithms, including Logistic Regression, Support Vector Machines, Decision Trees, Random Forests, and XGBoost, alongside different text embedding techniques. To evaluate the approaches, we also collect a dataset of German comments from Reddit.

Our results reveal that for both approaches, self-training generally diminishes the overall performance for detecting toxic comments. Furthermore, our findings suggest that SVM handles pseudo-labels better than the other algorithms. However, it is important to note that these results are not fully conclusive, given the poor performances of the base models. Notably, Logistic Regression emerges as the best-performing base model, albeit with a low F1-score of 0.286. Furthermore, FastText emerges as the best-performing embedding during the evaluation. Finally, our findings also suggest that there may be an optimal ratio between pseudo-labeled and labeled data in training sets.

Acknowledgements

I would like to extend my sincere thanks to PD Dr. Kaspar Riesen, who helped define the research questions and whose support and guidance throughout the thesis was invaluable. Finally, I would like to express my gratitude to my family and girlfriend for their their feedback and moral support.

Contents

Chapter 1

Introduction

In this chapter the research topic is outlined. First the topic of toxic comments is introduced in Section 1.1. Next, the research goals and methods of the thesis are outlined in Section 1.2. Finally, an overview of the structure of the thesis is provided in Section 1.3.

1.1 Toxic Comment Classification

In recent years, the internet and social media have become indispensable tools for communication. Participating in online discussions by means of commenting is an essential way to exercise one's right to freedom of expression. Comments allow users to express their opinions, add crucial information, correct inaccuracies or misinformation. However, some malicious users try to disrupt these discussions with toxic comments. Toxic comments are comments that are rude, disrespectful or unreasonable [1].

Consequently, toxic comments become a problem for social media, blogs and news platforms, according to Risch [1]. First, they decrease the amount of users engaging in discussions, thus lowering the number of platform visitors. Second, legal obligations might require that platforms implement countermeasures against hate speech. For example, in Germany, due to the Network Enforcement Act, platforms are obliged to remove obviously illegal content, including comments, within 24 hours of notification. Hence, both problems could lead to revenue losses for the platforms. However, manual moderation of comment sections is costly, time-consuming, and often impractical [2].

Artificial Intelligence (AI) provides various automated solutions for the problem of toxic comments. The field of AI includes several subfields concerning comment

section moderation, for instance, Machine Learning (ML), Deep Learning (DL), and Natural Language Processing (NLP) [3]. These subfields provide distinct approaches to identify and filter out toxic comments. Thus, enabling platforms to effectively moderate comment sections [1].

Natural Language Processing, in particular, enables machines to understand, interpret and generate human language [4]. It includes different tasks such as speech recognition, machine translation and sentiment analysis [3]. Sentiment analysis provides valuable insights into the emotional reactions of users [5]. Meanwhile, text classification, one of the most important problems in NLP [6], enables machines to categorize texts into predefined classes or categories based on their content, tone, or context [7]. Toxic comment classification combines the methodologies of text classification and sentiment analysis [1]. Its primary goal is to automatically identify and flag toxic comments.

Recently, there has been a notable increase in research about toxic comments, with various challenges being explored [2]. Challenges include, for example, the lack of labeled data [1], data imbalance problems [6], difficulties due to language variations [8] and the obstacles in the annotation process [9]. Current research is increasingly focused on improving the robustness and accuracy of models [1].

While substantial research is dedicated to detecting toxic comments in English [2], research on this topic in German is relatively limited in comparison [10]. However, initiatives such as the GermEval 2018 and 2021 shared tasks on the detection of offensive language and toxic comments represent significant steps towards addressing this gap [11].

Despite these advancements, a persistent challenge remains: the scarcity of labeled data [1]. Risch et al. [12] observe that the shortage of human-labeled data is primarily due to the costly nature of the labeling process and the inherent imbalance in toxic comment datasets. Specifically, there is a bias towards non-toxic comments because most platforms contain only a small proportion of toxic comments. Duarte [13] reports that, to address the lack of labeled data, semi-supervised learning (SSL) emerges as a promising approach. By leveraging a combination of a small amount of labeled data and a larger quantity of unlabeled data, SSL techniques can enhance model performance. Although SSL shows success across various NLP tasks [14], its application to detecting toxic comments remains relatively unexplored [15], particularly in the German language.

1.2 Research Goals

This thesis aims to investigate the utilization of SSL specifically for detecting toxic comments in the German language. Semi-supervised learning techniques have the potential to overcome the scarcity of human-labeled data and improve model effectiveness [13]. The objective of this thesis is to investigate the influence of SSL, particularly self-training, on the performance of five different algorithms, including Logistic Regression, Support Vector Machines (SVM), Decision Trees, Random Forests, and XGBoost. The central research questions addressed in this thesis are:

> **Research Question 1:** *How does self-training with pseudo-labels influence the performance for detecting toxic comments in German?*
>
> **Research Question 2:** *How does augmenting the training set with only toxic pseudo-labeled comments affect the model's performance compared to using the complete pseudo-labeled dataset?*
>
> **Research Question 3:** *What impact do variations in the quantity of pseudo-labeled data in the training set have on model performance?*

To investigate these research questions, the following preliminary steps are necessary. First, a dataset of German comments from Reddit will be collected, with a portion manually labeled to support the training and evaluation process. Second, a preliminary experiment will evaluate four types of text embeddings: TF-IDF, Word2Vec, FastText, and Doc2Vec, to determine the most effective embedding for each algorithm.

Following these preliminary steps, the effect of self-training will be examined using two methods. First, by training each algorithm with labeled data and varying amounts of pseudo-labels. Second, by training each algorithm with labeled data and varying amounts of pseudo-labels classified as toxic. The performance of the two methods are compared to determine which approach yields superior results in improving algorithms. The models are evaluated on various metrics such as precision, recall and F1-score to provide a detailed performance analysis.

In conclusion, this thesis will contribute to understanding how self-training with pseudo-labels can enhance the detection of toxic comments in German. It will offer insights into the influence of self-training on various algorithms. Additionally, the thesis provides insights into how text embeddings impact the performance of different algorithms.

1.3 Outline of the Thesis

The remainder of the thesis is organized as follows. Next, Chapter 2 provides a formal definition of toxic comments and describes SSL and its potential for toxic comment classification. Additionally, the chapter outlines the embeddings and algorithms used in the experiments. In Chapter 3, the research methods for the evaluation of self-training are explained and justified. The chapter first discusses the dataset collection and labeling process. Subsequently, the preliminary and self-training experiments are explained in detail. In Chapter 4, the evaluation results are presented and discussed, starting with an analysis of the labeling results. Moreover, the chapter also discusses the results of the embedding selection and the self-training results. Finally, Chapter 5 revisits and discusses the research questions of the thesis, presenting the conclusions of the thesis.

Chapter 2

Basic Concepts

In this chapter, we formally introduce several fundamental concepts essential for the evaluation of self-training. Section 2.1 provides a formal definition for toxic comments and toxicity classes. Following this, Section 2.2 discusses self-training and its potential for classifying toxic comments. Section 2.3 introduces embeddings and provides explanations for the four embeddings used in the experiment. Next, Section 2.4 similarly discusses the algorithms employed in the experiment. Lastly, Section 2.5 provides a detailed overview of the evaluation metrics utilized.

2.1 Toxic Comment

A major problem in the area of hate speech detection is that toxic comments are difficult to define precisely [10]. Various definitions can be used to characterize toxic comments [2]. For example, different research papers use numerous classification classes to classify toxic comments [9]. However, Fortuna et al. [16], in their annotation guideline for toxic comment datasets, advise against creating new definitions and classes. They suggest to instead use existing concepts. Consequently, this thesis uses the definition proposed by the Perspective API, which is developed by Jigsaw and Google's Counter Abuse Technology team [16]:

Definition 1. *A toxic comment is defined as a rude, disrespectful, or unreasonable comment that is likely to make other users leave a discussion.*

This definition provides several advantages. It not only considers the content of a comment but also focuses on its impact. As a result, this approach defines toxicity in a more flexible manner, covering a broad spectrum of toxic comments. Furthermore, the definition is utilized in multiple research papers, including those by Poojitha et

al. [17] and Risch [1].

2.1.1 Classes of Toxicity

Recently, there is a shift from binary classification to a more nuanced, fine-grained classification according to Risch [1]. Binary classification oversimplifies the reasons for labeling a comment as toxic, making it unclear why a comment is flagged. In contrast, a fine-grained approach provides insight into the specific reasons a comment is deemed unacceptable [16].

Research papers employ various classification classes, such as racist, sexist, and hate for their evaluation of toxic comments. The Kaggle Challenge on Toxic Comment Classification [18] in 2017 introduces the following classes: toxic, severe toxic, obscene, threat, insult, and identity hate [12]. Androcec [2] notes that these classes are used in the majority of reviewed papers. Following the recommendation by Fortuna et al. [16] to use already established classes, this thesis incorporates the classification framework from the Kaggle Challenge [18]. Furthermore, Fortuna et al. [16] recommend that detailed explanations and examples for each class should be included in research papers. Consequently, the following paragraphs provide a detailed explanation of each class using definitions by Risch [1], along with examples to illustrate their characteristics

Obscene

Comments categorized as obscene often include inappropriate language, offensive words, or graphic descriptions, such as sexual or violent content. Detecting some of these types of comments can be accomplished using blacklists designed specifically for swear words and curse words [1]. In summary, comments categorized in this class contain obscene language in the form of specific words or explicit content.

Example 2.1.1. *Der Kuhle fickt den jetz [sic]*

Insults

Insults contain rude or offensive statements directed at an individual or a group. In contrast to the obscene class, comments in the insult class always have a specific target. Often, other users are the targets of insults, but this is not always the case [1]. Generally, comments in this class contain language that is hurtful or insulting towards others.

Example 2.1.2. *sorry aber der Müller ist so 1 pimmel*

Threats

Threats contain statements that advocate or imply causing harm or injury to oneself or others. This class also includes comments threatening to have another user's account closed or statements that hint at property destruction [1]. In general, comments in this class contain language that implies a threat of physical harm, violence, or other forms of damage.

Example 2.1.3. *Schon mal eins aufs mal gekriegt wegen deinem frechen Mundwerk?*

Severe Toxic

The severe toxic class contains comments that threaten the lives of specific individuals or groups [1]. Moreover, comments categorized in this class typically contain extremely harmful or abusive language.

Example 2.1.4. *Ja echt ungerecht. Ich finde man muss doch dafür einstehen können die Juden endlich töten zu dürfen, Adolf hat zu wenig getan!!! :(*

Identity Hate

In contrast to insults, identity hate specifically targets groups based on characteristics such as religion, sexual orientation, ethnicity, gender, or other social identifiers. It involves attributing negative qualities to these groups as if these traits are universally applicable [1]. For instance, comments that are racist, homophobic, or misogynistic are examples of identity hate.

Example 2.1.5. *ich bin auch mit einem Schlitzauge befreundet und ihn stört das nicht wenn ich ihn so nenne (hab ihn aber auch noch nie gefragt)*

Toxic

Comments likely to discourage other users from participating in a discussion are categorized as toxic. For instance, trolling, which involves posting irrelevant comments to disrupt the conversation, falls under this classification. Similarly, spam messages belong to this class [1]. Comments from all other classes could also be considered toxic as well. Furthermore, this class is based on the presumed effect

of comments on readers, which naturally depends on the readers' personalities. [16].

Example 2.1.6. *Deine Meinung ist Mist*

The specified classes are not mutually exclusive. As a result, toxic comments can belong to multiple classes. For instance, Example 2.1.4 could also belong to the classes threats, identity hate and toxic. Similarly, Example 2.1.5 could also be classified as both an insult and a toxic comment.

2.1.2 Multi-label Classification

According to Risch [1], approaches to toxic comment classification can be categorized as either multi-class or multi-label methods. Multi-label classification allows a comment to belong to multiple classes simultaneously, whereas multi-class classification assumes classes are mutually exclusive. For instance, in multi-class classification, a comment could be categorized as either a threat or obscene, but not both. However, as demonstrated by Example 2.1.4 and Example 2.1.5, comments often exhibit characteristics of multiple classes. As a result, multi-label classification provides a better representation of real-world scenarios [1]. Furthermore, it is better suited for applications where identifying and addressing multiple aspects of toxicity within a single comment is essential [16].

2.2 Self-Training

Natural Language Processing plays a crucial role in the classification of toxic comments [19]. Various ML and DL techniques are employed to analyze the language and context within comments. Text classification can be broadly divided into unsupervised learning, supervised learning and SSL approaches [13]. Each of these approaches addresses different objectives and presents unique advantages and disadvantages [3].

Unsupervised learning focuses on finding patterns and structures in unlabeled data [3]. Typical tasks include clustering data points based on similarity and dimensionality reduction, where the objective is to simplify the data while retaining as much information as possible [19]. In contrast to unsupervised learning, supervised learning works with labeled data, where each example is annotated with the correct label [19]. For instance, a supervised algorithm learns to generate a function that can predict the correct label when presented with new data [3]. Common supervised tasks include predicting a value based on a set of features or performing

classification [19].

However, supervised approaches struggle when labeled data is scarce [5]. These methods require substantial labeled data to create robust models [15]. A lack of labeled data often leads to overfitting, where the model tends to memorize the limited training examples, including outliers, rather than learning generalized patterns. This results in poor performance on new, unseen data [14]. Furthermore, Rastogi et al. [20] emphasize that in text classification "the size of labeled training data still dictates the performance of a model".

Semi-supervised learning might overcome these limitations by leveraging both labeled and unlabeled data [15]. This approach is particularly useful when acquiring a large labeled dataset is challenging, but large amounts of unlabeled data are easily available [14]. In such scenarios, purely supervised or unsupervised methods are usually insufficient [14]. Common SSL methods include co-training, self-training and graph-based methods [13].

Self-training is a widely adopted category of SSL techniques in text classification [14]. It belongs to wrapper methods, which work by generating pseudo-labels for unlabeled data and combining these newly labeled instances with existing labeled data to train a classifier [13]. According to Sosea [14], the process begins with training a model on a small labeled dataset to create a base model. This base model then predicts labels for the unlabeled data, thus generating pseudo-labels for the unlabeled data. These new labels are referred to as pseudo-labels because they may be inaccurate. The pseudo-labels are combined with the original labeled dataset to create an expanded training set. Subsequently, the model is retrained on this new dataset, allowing it to learn from a larger corpus. This process can be repeated iteratively [5], with each iteration exposing it to more data and potentially improving its performance [13].

Semi-supervised learning, particularly self-training, has the potential to enhance the effectiveness of toxic comment classification [15]. Since, collecting a comprehensive labeled dataset is time-consuming and costly [1], self-training offers a practical solution by reducing the reliance on extensive manual labeling [14]. Not only does self-training reduce labeling costs [15], but it can also lead to improved model performance [13]. By leveraging self-training, the classifier could periodically be updated with new data, potentially helping it remain effective against the evolving language of toxic comments. This approach could enhance its ability to detect various forms of toxicity that may not be present in the original labeled dataset. Overall, self-training may offer a cost-effective strategy for improving toxic comment

classification [15].

2.3 Embeddings

Patil et al. [21] define NLP as comprising two main steps: "first, the representation of the input text (raw data) into a numerical format (vectors or matrices), and second, the design of models for processing the numerical data to achieve a desired goal or task". Embedding refers to the initial step, enabling ML algorithms to easily process the data. There are several techniques to create embeddings, ranging from traditional methods like bag of words to more advanced approaches such as Word2Vec, GloVe, and BERT [22]. The present thesis uses the following four embeddings for its experiments:

2.3.1 TF-IDF

Term frequency-inverse document frequency (TF-IDF) is a technique used to quantify the importance of a word in a document (e.g. a comment) relative to the rest of the collection of documents [21]. Since it is based on the prevalence of words occurring in a document, it is classified as a frequency-based embedding [22]. It combines the two statistics term frequency (TF) and inverse document frequency (IDF) [17]:

Term frequency measures how frequently a particular term is used in a document's text. [17]. Since documents can vary significantly in length, words are likely to appear more frequently in longer documents compared to shorter ones [21]. To account for this, the term frequency is normalized by dividing it by the total number of words in the document. Term frequency is defined by the following formula [17]:

$$TF_{t_d} = \frac{\text{Number of times } t \text{ appears in a document } d}{\text{Total number of terms in a document } d}$$

Conversely, IDF assesses how infrequent a term is across the entire document collection. Inverse document frequency evaluates the importance of a word by considering its rarity across all documents [17]. It penalizes words that are not unique to specific documents [21]. For instance, common words like "the" or "and" are affected by this measure. Inverse document frequency is defined by the following formula [17]:

$$IDF_t = \log\left(\frac{\text{Total number of documents}}{\text{Number of documents with term } t \text{ in it}}\right)$$

The TF-IDF score is calculated for every word using the formula $TF - IDF_{t_d} = TF_{t_d} * IDF_t$ [21]. Subsequently, we generate a term-by-document matrix, with the matrix containing TF-IDF values [22]. Words with higher TF-IDF scores are deemed more significant, while those with lower scores are regarded as less important [22].

Using TF-IDF, document vectors can effectively reveal underlying concepts and topics. However, TF-IDF matrices are typically high-dimensional and sparse [21]. To address this issue, dimensionality reduction techniques are often applied to TF-IDF vectors. Additionally, TF-IDF fails to establish connections between terms, thus failing to link synonyms and distinguish antonyms in the vector space [22]. Overall, in terms of accuracy, TF-IDF vectors generally outperform simpler methods such as Bag of Words or One Hot Encoding [21].

2.3.2 Word2Vec

Word2Vec is an example of a static word embedding [21]. Whereas TF-IDF focuses on frequency, Word2Vec aims to capture the semantic relationship and meanings between words using a neural network [23]. Essentially, words with similar contexts are represented by similar vectors [22]. For instance, the distance between the words "*Doctor*" and "*Physician*" will be less than the distance from "*Doctor*" to "*Tomato*", due to their semantic relationship [22]. Unlike TF-IDF which maps documents to vectors, Word2Vec converts individual words into a vector [21]. As a result, word vectors are usually dense and have lower dimensionality compared to the vocabulary size [22].

Word2Vec is used in two distinct ways: Continuous Bag of Words (CBoW) and skip-gram [22]. The CBoW approach aims to predict a target word based on its surrounding context words. In contrast, skip-gram attempts to predict the context, given the word [21]. Both approaches use a fully connected neural network with a single hidden layer [22]. During training, the neural network learns to adjust the weights to minimize the prediction error, resulting in the final word vectors [22].

Word2Vec represents each word as a vector. However, this approach ignores the morphology of words, making it unsuitable for morphologically rich languages [21]. Moreover, Word2Vec struggles to handle out-of-vocabulary words, which are words not present in the training data, resulting in a lack of embeddings for these words [22].

2.3.3 FastText

FastText, an extension of Word2Vec [24], addresses the limitations of Word2Vec by utilizing a skip-gram approach [21]. In FastText, words are represented as collections of character n-grams, where each character n-gram is represented by a vector [22]. The representation of a word is the sum of these n-gram vectors [25]. Consequently, FastText incorporates subword information, which is absent in Word2Vec [21]. In contrast to Word2Vec, FastText can generate embeddings for misspelled words, rare words, and words that are not present in the training data [24]. Furthermore, Risch [1] highlights that FastText is effective for managing toxic comments, as its subword embeddings address out-of-vocabulary words. This is particularly important because toxic comments frequently contain obfuscation and misspellings, which are common in online discussions. For example, users might write "*loseeer*" instead of "*loser*".

2.3.4 Doc2Vec

Similar to FastText, Doc2Vec is another model derived from Word2Vec [26]. However, unlike Word2Vec, which focuses on individual words, Doc2Vec extends its embeddings to encompass larger text sequences such as sentences, paragraphs, or even entire documents [27]. This characteristic makes Doc2Vec particularly advantageous for tasks such as sentiment analysis, especially when dealing with longer texts [28].

Lau [28] explains that Doc2Vec employs two different approaches: Distributed Bag of Words (DBoW) and Distributed Memory Paragraph Vectors (DMPV). The DBoW model simplifies the representation by ignoring word order and operates similarly to the CBOW approach of Word2Vec. In this model, the document is represented by a single token that replaces the input words. On the other hand, DMPV functions similarly to the skip-gram model of Word2Vec. However, it combines a token specifically for the document with the target words during training.

2.4 Algorithms

To effectively classify toxic comments, various ML and DL algorithms are utilized in different research papers [2]. In this section, we briefly introduce each algorithm used for the experiments. An overview of the key hyperparameters for each algorithm utilized in the experiment is provided in Section A.1 of the appendix.

2.4.1 Logistic Regression

Géron [19] explains that Logistic Regression is frequently used to estimate the probability that a given example belongs to a specific class. It is a regression algorithm that calculates a weighted sum of its input features. The output is then transformed using the logistic sigmoid function, which ensures that the predictions are constrained between 0 and 1. For instance, it can estimate the probability that an email is spam. It is one of the most popular classification algorithms due to its simplicity and interpretability [3].

2.4.2 Support Vector Machines

Support Vector Machines (SVM) are powerful ML models that can handle linear classification, regression, and novelty detection [19]. Support Vector Machines attempt to identify the optimal hyperplane that divides classes in a high-dimensional space. Furthermore, they aim to maximize the margin between the classes to improve generalization and reduce the risk of overfitting [3]. They can use the kernel trick to map data into a higher-dimensional space, making nonlinear decision boundaries possible [19]. Consequently, data that is not linearly separable in the original space often becomes easily separable in this higher-dimensional space [3].

2.4.3 Decision Trees

Decision Trees are flexible ML algorithms capable of handling classification, regression, and even multi-output tasks [19]. A Decision Tree makes its decision by conducting a series of tests, beginning at the root and proceeding down the appropriate branches until it reaches a leaf. Each internal node represents a test on the value of an input attribute [3].

The main advantages of Decision Trees are their interpretability and visualization capabilities. Trees can easily be visualized and used to explain the decision-making process [19]. However, if the trees are very deep, predicting a new example can be computationally expensive. Decision trees are also unstable, adding a single new example could affect the entire tree [3].

2.4.4 Random Forests

Random Forests address some of the limitations of Decision Trees [3]. They are an ensemble method where multiple Decision Tree classifiers are trained on different random subsets of the training data. The predictions from each tree are then

aggregated to achieve better accuracy than a single classifier [19]. Furthermore, by aggregating multiple trees, the model becomes less sensitive to the addition of new examples and it also reduces the risk of overfitting. Similar to Decision Trees, Random Forests can be used for both classification and regression tasks [3].

2.4.5 XGBoost

Extreme Gradient Boosting (XGBoost) implements gradient boosting [3]. Boosting is an ensemble method that combines multiple weak learners into a strong learner. The core idea behind most boosting methods is to train classifiers in sequence, with each one attempting to improve the errors of its predecessor [19]. According to Russel [3], the most widely used boosting methods are AdaBoost and gradient boosting. XGBoost is commonly used for large-scale applications handling billions of examples. Furthermore, it ensures efficiency by optimizing memory organization to minimize cache misses and enabling parallel computation across multiple machines.

2.5 Evaluation Metrics

Evaluation metrics are crucial to assess the performance of classifiers. Yet, the evaluation of a classifier can be challenging [2]. One fundamental metric is accuracy, which measures the proportion of correctly classified comments [19]. However, accuracy is not useful when datasets are imbalanced, as is often the case with toxic comment datasets [1]. For example, suppose a dataset contains 5% toxic comments and 95% non-toxic comments. A model that classifies each comment as non-toxic would achieve a 95% accuracy, despite not being able to recognize a single toxic comment.

There are several metrics to assess a model's performance more comprehensively. One method used to visualize the performance of a model is the confusion matrix [19]. Figure 2.1 shows an example of a confusion matrix. In a confusion matrix, each row corresponds to an actual class, while each column corresponds to a predicted class [19]. With the help of the matrix we can define the following evaluation metrics [19]:

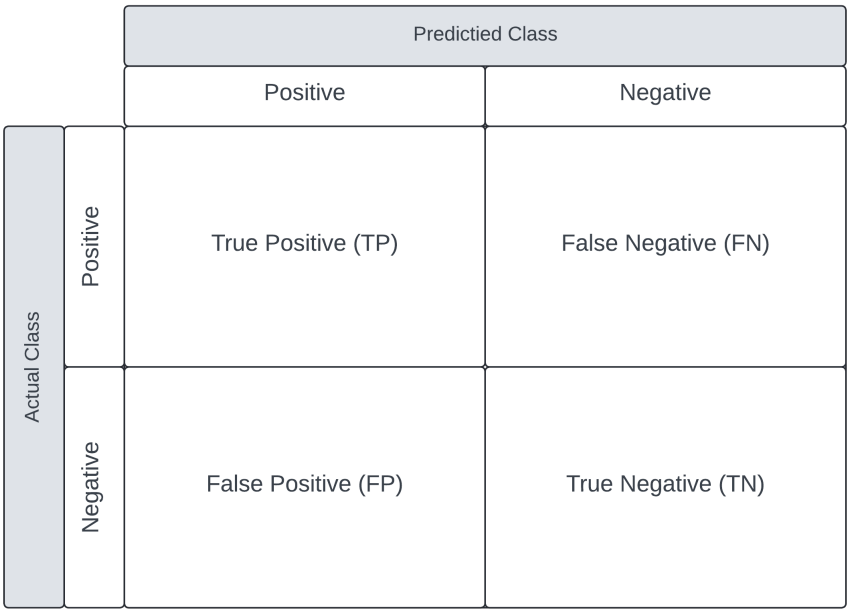

Figure 2.1: Confusion Matrix [19]

$$Precision = \frac{TP}{TP + FP} \tag{2.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.2}$$

$$F1\text{-}score = 2 \times \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.3}$$

Precision measures the accuracy of the positive predictions or in other terms it indicates the proportion of positive predictions that is actually correct. Recall, or sensitivity, indicates the ratio of positive instances that are correctly detected [17]. However, these two metrics are not independent and an increase in one might lead to a decrease in the other one [19].

The F1-score is the harmonic mean of precision and recall [17]. While the regular mean assigns equal importance to all values, the harmonic mean places significantly more emphasis on lower values. Consequently, a classifier will achieve a high F1-score only when both recall and precision are high [19]. Androcec [2] points out

that the most used evaluation metric in research papers about toxic comment classification is the F1-score. Therefore, it is chosen as the main evaluation metric for the present thesis.

Chapter 3

Methods

In this chapter, the research methods are explained in detail and justified. Section 3.1 explains the framework used for the classification. Next, Section 3.2 discusses the dataset collection and labeling process. Subsequently, Section 3.3 details the process for selecting the embeddings for the main experiment. Finally, Section 3.4 describes the self-training approach utilized in the experiment.

3.1 Text Classification Phases

This thesis applies Mironczuk's [7] framework for text classification to classify toxic comments. The framework encompasses the following six steps:

1. Data acquisition
2. Data analysis and labeling
3. Feature construction and weighting
4. Feature selection and projection
5. Classifier training
6. Solution evaluation

According to Mironczuk [7], the process begins with acquiring data to create a dataset as the foundation for the classification process. Second, the dataset is labeled to obtain a training and test set. Next, the dataset is preprocessed to prepare it for the chosen classifier. This involves constructing features from the data and then applying a selected feature representation algorithm, such as word embeddings, to appropriately weigh these features. Subsequently, feature selection methods may be employed to reduce the number of features, followed by projecting

the reduced features into a lower-dimensional space to achieve the most effective data representation. The classifier is then utilized to categorize the texts. Finally, the performance is assessed using the chosen evaluation metrics.

3.2 Dataset

3.2.1 Collection

Following the framework proposed by Mironczuk [7], evaluating the impact of self-training requires a large dataset with both labeled and unlabeled data. As research on toxic comment classification has expanded, an increasing number of annotated datasets have become available [12]. However, both Risch et al. [12] and Demus et al. [10] observe that the majority of datasets focus on English comments and are sourced from Twitter, with few considering German comments. To address this research gap, a dataset comprising German comments is collected.

The data is collected from Reddit, a popular social media platform [29]. Reddit offers multiple distinct advantages: First, a cross-platform study shows that Reddit typically exhibits higher toxicity levels compared to other platforms such as Twitter [8]. Consequently, this helps address the bias towards non-toxic comments in datasets [1]. Second, Reddit offers diverse content and allows nested responses, resulting in a wide range of discussions and comments [8]. Additionally, Reddit allows longer comments, which can provide more context and depth to the discussions.

To reduce biased data collection [16] and obtain a representative sample of toxic comments on Reddit, comments are gathered from the 16 largest German subreddits, excluding those where the majority of comments are in English. An overview of the subreddits is provided in the Table 4.1. For each subreddit, 10,000 comments are collected, ensuring coverage of a wide range of topics. In total, 160,000 comments are collected across various subreddits. This approach helps mitigate the bias that Risch [1] and Demus et al. [10] noted as a common issue in datasets.

Data acquisition is accomplished using the Reddit Scraper API, which is built on the Apify SDK [30]. Given a subreddit name, the web scraper crawls through posts on the "*hot page*" of Reddit, which ranks post based on upvotes and submission time [8]. Generally, newer posts will rank higher than older ones. Starting with the top-ranked post, the scraper collects all its comments. It then moves to the next ranked post and gathers all comments from there. This process continues until a total of 10,000 comments are collected.

The web scraper's approach offers two benefits: First, comments are gathered from

multiple posts, ensuring a wide range of comments from posts with various topics are included. Second, the web scraper collects comments from newer posts, which are less likely to have been manually moderated, thereby increasing the probability of capturing toxic comments. In summary, the use of the Reddit Scraper API not only enhances the diversity of the dataset but also increases the likelihood of capturing toxic comments.

3.2.2 Clean Up

The collected data includes irrelevant content that needs to be eliminated to improve the learning efficiency of the algorithms [17]. First, the dataset contains comments removed by moderators, deleted comments and duplicates, all of which need to be filtered out. Second, comments with fewer than four characters are excluded. Third, the web scraper has issues with scraping the > and < signs, replacing them with "&gt" and "&lt", which also need to be removed. Finally, comments containing only URLs are removed. This process is performed using the Python pandas library. By eliminating unnecessary comments, such as duplicates, the quality of the dataset is significantly increased. This ensures that the dataset captures only unique comments, consequently leading to more efficient model training. In summary, the cleanup process enhances the dataset's quality, providing a solid foundation for the further text classification process.

3.2.3 Labeling

Following the next step in the framework [7], the collected comments are manually labeled by one person. 1,000 comments from the ten largest subreddits are labeled, resulting in a total of 10,000 labeled comments. This quantity ensures there is a sufficient amount of labeled data as well as a substantial amount of remaining unlabeled data to explore self-training possibilities [13]. Comments are taken from multiple subreddits, ensuring a diverse range of comments is covered in the labeling process. Initially, the comments are labeled using binary classification. For this labeling process, Definition 1 by the Perspective API [16] is used to classify the comments. Afterwards, the comments identified as toxic during the binary classification are reclassified using multi-label classification, incorporating the classes defined in Section 2.1.1. In summary, four datasets are created: one dataset with 10,000 binary labels, the same dataset but with multi-labels, a dataset containing only the remaining unlabeled comments and finally a dataset without labels containing all collected comments.

3.3 Preliminary Experiment

Before conducting the main experiment involving self-training, a preliminary experiment is conducted. The goal of this preliminary experiment is to select the best-performing embedding for each algorithm detailed in Section 2.4. The embeddings used in the preliminary experiment are outlined in Section 2.3. Each algorithm will then use its best-performing embedding in the self-training experiment.

3.3.1 Preprocessing

Prior to evaluating and selecting the embeddings, it is essential to preprocess the comments [7]. Preprocessing ensures that the data is in a standardized and usable format. The preprocessing phase plays an important role in improving the quality of embeddings, consequently improving their performance [17].

For preprocessing the comments, a combination of steps proposed by Datta et al. [31] and Poojitha et al. [17] are executed in the given sequence. First, URLs, special characters, e-mails and numbers are removed [31]. Removing numbers is also a part of preprocessing, because it improves model performance and decreases the complexity of the data [17]. Next, the text is lemmatized using spaCy, a library for NLP processing. Additionally, punctuation is removed since it causes issues for ML algorithms [17]. Afterwards, extra whitespace is removed and the text is transformed to lowercase. Details of the preprocessing code are available in Section A.2 of the appendix.

Lemmatization and stemming are both common techniques used in preprocessing to reduce the size of the vocabulary and decrease the spareness of the data [32]. Lemmatization reduces words to their dictionary form while considering the context of the vocabulary and the morphological analysis [33]. Considering the context ensures that the resulting words are valid words [33]. For instance, both "lief" and "gelaufen" will be reduced to "laufen". On the other hand, stemming reduces words by removing prefixes and suffixes without considering the context [21]. This may result in words that do not exist [33]. While stemming is usually sufficient in English, lemmatization is preferred in German due to the more complex morphology of words [10].

Although stopword removal is also a common preprocessing technique in NLP [21], it is not implemented in this thesis. Stopwords, such as "und" or "nicht", are words that might not add substantial meaning to a sentence [17]. However, in sentiment analysis, their removal may negatively impact model performance [31]. Therefore, this particular preprocessing step is excluded.

To illustrate the result of preprocessing, consider Example 2.1.5 introduced earlier. After going through the preprocessing steps, the comment transforms significantly due to lemmatization:

Example 3.3.1. *ich sein auch mit ein schlitzaug befreunden und ihn stören der nicht wenn ich ihn so nennen hab ihn aber auch noch nie fragen*

3.3.2 Embedding Training and Vectorization

For all embeddings the preprocessed labeled dataset is split into a training set and test set with each containing 5,000 comments. Next, each embedding model is trained and subsequently each comment is converted into vectors:

TF-IDF

A TF-IDF vectorizer is used to transform the text data into numerical representations. First, the vocabulary and IDF are learned from the training and test dataset. Next, the training and test set are transformed into TF-IDF feature vectors. A standard scaler is then employed to normalize the data, which is particularly important given the sparse nature of the TF-IDF vectors [21].

Word2Vec and FastText

Both Word2Vec and FastText follow the same process to generate embeddings. In particular, Word2Vec uses the CBoW approach. Initially, the model is trained on the entire set of collected comments to learn the embeddings for each individual word. Then, the model transforms each sentence into fixed-size vectors. In Code 3.1, the vectorization of sentences is illustrated in Python.

```python
def vectorize(sentence):
    words = sentence.split()
    words_vecs = [model.wv[word] for word in words if word in model.wv]
    if len(words_vecs) == 0:
        return np.zeros(params['vector_size'])
    words_vecs = np.array(words_vecs)
    return words_vecs.mean(axis=0)
```

Code 3.1: Function to convert a sentence to its average word vector using pretrained embeddings

The method's goal is to convert a sentence into a vector using the trained embedding model, which can be either FastText or Word2Vec. Initially, the function splits the

input sentence into individual words. Subsequently, it attempts to retrieve the word vector for each word from a pre-trained model. If a word is not found in the model's vocabulary, it is ignored. This process results in a list of word vectors. If none of the words in the sentence are found in the model, the function returns a zero vector with the same size as the word vectors. Next, the vectors are converted into a NumPy array and the mean vector is computed across all word vectors. Finally, the mean vector is returned.

Doc2Vec

Doc2Vec uses the DBoW approach during training. Furthermore, Doc2Vec requires tagged documents in order to create embeddings. Initially, each comment is split into sentences and each sentence is assigned a unique tag. After initialization, the model builds its vocabulary from the tagged sentences and is subsequently trained on the data. Subsequently, the model infers vectors for each sentence in the training and test sets using its learned vector representations.

3.3.3 Hyperparameter Tuning

To identify the optimal embedding and algorithm combination, a comprehensive hyperparameter tuning approach is employed. Algorithm 1 presents a pseudocode example of the preliminary experiment, specifically demonstrating the process for FastText and SVM. The pseudocode serves as the basis for explaining the hyperparameter tuning process in detail.

Algorithm 1 Pseudocode for the preliminary experiment

```
Define FastText parameter grid (vector size, window, min count, epochs)
for each combination of FastText parameters in the grid do
    Train FastText and vectorize data
    Find best SVM model using GridSearch
    Train and evaluate the best SVM model
    if F1-score > best_F1_score then
        Update best parameters and metrics
    end if
end for
Print parameters of the best FastText-SVM combination
```

The process begins by loading the training and test datasets. Then, a parameter grid is defined for the FastText model. Table A.6 in the appendix outlines the parameter grid used for Word2Vec, FastText, and Doc2Vec. Meanwhile, Table A.7 displays the parameter grid for TF-IDF. The tuning process involves iterating through every combination of these embedding parameters.

For each combination, the FastText model is trained on the entire dataset. Subsequently, the training and test sets are vectorized using the trained FastText model. The SVM algorithm is then tuned using GridSearchCV, focusing on maximizing the F1-score. Following this, the SVM model is trained using the optimal parameters. Next, the model's performance is evaluated on the test set, calculating accuracy, precision, recall, and F1-score. If the current F1-score exceeds the previous best, the parameters and metrics are updated. Finally, the parameters and evaluation scores of the overall best model are printed.

This process enables detailed hyperparameter tuning for both the algorithm and embedding, ensuring the optimal parameters are identified for each. Furthermore, each combination is evaluated allowing for a detailed performance evaluation. This process is repeated for each combination of algorithms and embeddings to find the best embedding for each algorithm.

3.4 Main experiment

3.4.1 Base Model

In order to evaluate the effect of self-training, a base model is necessary. The base model in self-training refers to the initial model trained solely on the labeled dataset [14]. Before training the base model, preprocessing is needed to ensure the data is clean and properly formatted. The preprocessing steps, as explained in detail in Section 3.3.1, are also utilized in the self-training experiment.

Next, the labeled dataset is split into the same training and test set as in the preliminary experiment. Each algorithm then uses its most effective embedding configuration. The specific parameters used for the embeddings are available in Section A.4 in the appendix. The embedding is then applied to vectorize the training and test set, as outlined in Section 3.3.2.

Afterwards, the optimal hyperparameters of the algorithm are determined using GridSearchCV with stratified 5-fold cross-validation. Demus et al. [10] note that k-fold cross-validation is highly effective for imbalanced datasets. In 5-fold cross-validation, the training set is divided into 5 equal subsets. Each fold is then used once as a test set, while the remaining four folds are combined to form the training set. Stratified cross-validation is preferred for toxic comment datasets due to their class imbalance [1], as it preserves the percentage of samples for each class in each fold [34].

Subsequently, the algorithm is fine-tuned to optimize the F1-score. Once the op-

timal parameters are identified, the algorithm is trained using these parameters. Finally, the model's performance is evaluated on the test set. An example of a base model code is available in available in Section A.5.

3.4.2 Self-Training Process

Self-training is typically implemented in an iterative manner, according to Lee [5]. Pseudo-labels are generated and then added to the training set. Subsequently, the classifier is retrained and used to generate new pseudo-labels, repeating the process. However, this approach creates difficulties for evaluating the effect on the classifier's performance.

Due to the inherent class imbalance of toxic comments datasets [1], the pseudo-labeled dataset might only contain comments labeled as non-toxic. This bias towards non-toxic pseudo-labels significantly impacts the model's performance metrics. First, precision might increase as the model becomes more conservative in its toxic comment labeling. On the other hand, recall might decrease because the model fails to identify many toxic comments. As a consequence, the F1-score for the toxic class typically declines, with the drop in recall outweighing any gains in precision.

To address this issue, the present thesis uses a non-iterative approach that involves creating pseudo-labels for the entire unlabeled dataset. Figure 3.1 illustrates the complete process of self-training used in the thesis.

Dataset	**Number of comments**
Expansion Set 1	2,500
Expansion Set 2	5,000
Expansion Set 3	50,000
Expansion Set 4	115,433

Table 3.1: Sizes of pseudo-labeled datasets

After training the algorithm on the labeled data, the base model is used to make predictions on the unlabeled data, thereby creating pseudo-labels. Subsequently, multiple datasets are generated using the pseudo-labeled data. All pseudo-labeled datasets are designed to maintain the same proportion of toxic to non-toxic comments. In total, four datasets are created out of the unlabeled dataset. The sizes of the pseudo-labeled datasets are displayed in Table 3.1. In order to better distinguish the different pseudo-labeled datasets, they will be referred to as expansion sets from now on. The first expansion set is half the size of the training set. The

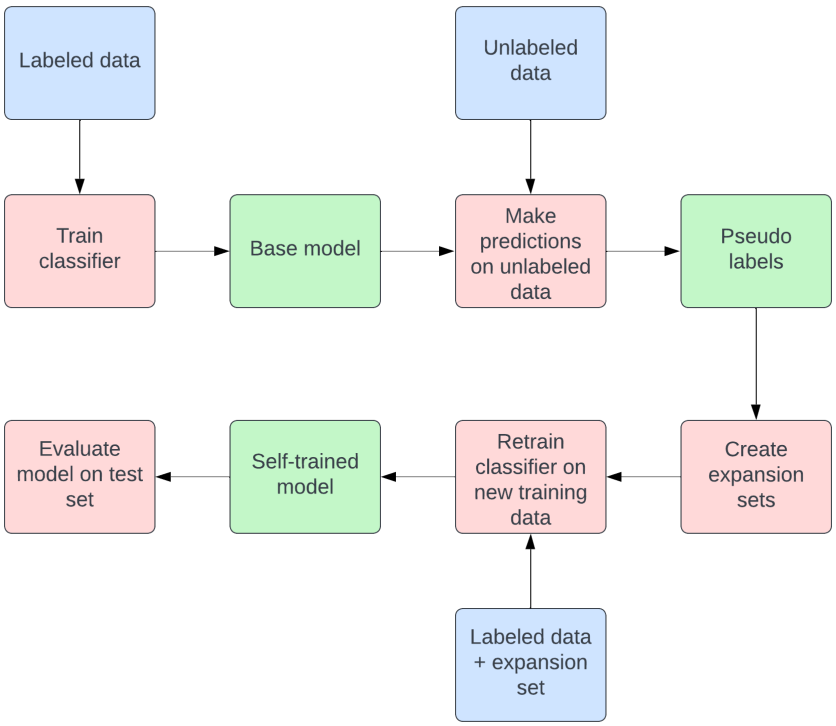

Figure 3.1: Self-Training process for a supervised classifier [15]

second expansion set has the same size as the training set. The third expansion set is ten times larger than the training set. Finally, the fourth expansion set contains all pseudo-labeled data.

In addition to the expansion sets, a secondary experiment is performed where only the toxic comments from the expansion set are added to the training set. This approach aims to address the class imbalance issue inherent in toxic comment datasets [1]. The decision to focus solely on toxic comments is supported by a study by Stanescu et al. [35]. Their research across multiple datasets shows that the approach of adding only positive instances, in this case toxic comments, consistently yields superior results compared to other techniques designed to mitigate class imbalance. By concentrating exclusively on toxic comments, there is potential to enhance the model's performance, particularly in identifying and classifying toxic content. To distinguish these datasets from the expansion sets, they are referred to as toxic sets from this point forward. The sizes of the toxic sets are presented in Table 3.2.

Algorithm	**Toxic Set 1**	**Toxic Set 2**	**Toxic Set 3**	**Toxic Set 4**
Logistic Regression	205	410	4,100	9,473
SVM	53	106	1,059	2,444
Decision Trees	435	870	8,701	20,080
Random Forests	12	26	261	603
XGBoost	280	559	5,584	12,911

Table 3.2: Amount of comments labeled as toxic in each expansion set

In a final step, the labeled training set is merged with either an expansion set or a toxic set to generate a new training set. The base model is then retrained on the new training data. The new model then undergoes the same process in hyperparameter tuning and optimization as the base model described in Section 3.4.1. Subsequently, the self-trained model is evaluated on the same test set as the base model.

Chapter 4

Experimental Evaluation

This chapter presents a thorough analysis of the experimental results and a detailed discussion. First, Section 4.1 examines the dataset and labeling process. Next, Section 4.2 details the results of the embedding selection. Subsequently, Section 4.3 offers an analysis of the self-training results. Finally, Section 4.4 presents a comprehensive error analysis on the outcome of the self-training experiment.

4.1 Dataset Results

4.1.1 Binary Classification Dataset

Table 4.1 displays an overview of the subreddits, their subscriber counts, and the number of comments remaining after the data cleanup. Additionally, the table indicates the number of toxic comments among the 1,000 comments labeled per subreddit, with only the 10 largest subreddits incorporated in the labeling process. The subreddits cover a broad range of topics, from football and financial issues to relationship advice. This diverse content ensures a comprehensive representation of discussions and comments on Reddit.

Furthermore, the comments column shows that approximately 22% of the comments are removed during the data cleanup outlined in Section 3.2.2. Interestingly, the removal of comments ranges from around 9% to 30% for individual subreddits. This variance is due to some subreddits facilitating more in-depth and specific discussions, thus decreasing the probability of duplicate comments. Additionally, different subreddits have varying moderation rules and methods, which also affect removal of comments.

In terms of toxic comments, we observe that out of the 10,000 labeled comments,

	Subreddit	Subscribers	Comments	Toxic Comments
1	r/de	2.2 mil.	7,871	47
2	r/ich_iel	1.5 mil.	7,693	39
3	r/Bundesliga	1.1 mil.	8,037	27
4	r/de_IAmA	577k	7,542	14
5	r/Austria	542k	7,424	56
6	r/Finanzen	556k	8,731	21
7	r/Lustig	402k	7,713	114
8	r/FragReddit	388k	9,052	38
9	r/wasletztepreis	387k	7,825	33
10	r/BinIchDasArschloch	354k	7,370	46
11	r/zocken	322k	8,492	
12	r/tja	275k	6,906	
13	r/fussball	291k	7,779	
14	r/duschgedanken	238k	8,344	
15	r/deutschememes	234k	7,102	
16	r/mauerstrassenwetten	199k	7,513	
Total			**125,394**	**435**

Table 4.1: Overview of the dataset containing comments from various subreddits

only 435 are classified as toxic. This number is notably low despite numerous measures to increase the amount of toxic comments captured. This demonstrates a clear class imbalance, with toxic comments representing only a small portion. This issue is highlighted in several research papers, including the studies conducted by Risch [1] and van Aken et al. [9]. This substantial imbalance presents difficulties in both collecting datasets and training models designed to detect toxic comments [6].

Notably, the dataset's labeling process is extremely time-consuming. Binary classification requires around 55 hours to label 10,000 comments. In contrast, labeling only the toxic comments for multi-label classification takes approximately five hours. This challenge, coupled with class imbalance issues [1], highlights the need to address the inefficiencies in the labeling process to improve overall efficiency.

It is noteworthy that during the labeling process, toxic comments frequently appeared in close proximity to one another. A phenomenon supported by research conducted by Shankaran [8]. This clustering of toxic comments suggests that certain discussions might escalate quickly, leading to a higher density of toxic comments.

Further analysis of toxic comments indicates several notable differences between toxic and non-toxic comments. Toxic comments have an average word count of 23.2 and a mean character count of 145.2. In contrast, non-toxic comments have an average word count of 28.3 and a mean character count of 177.7. This indicates that

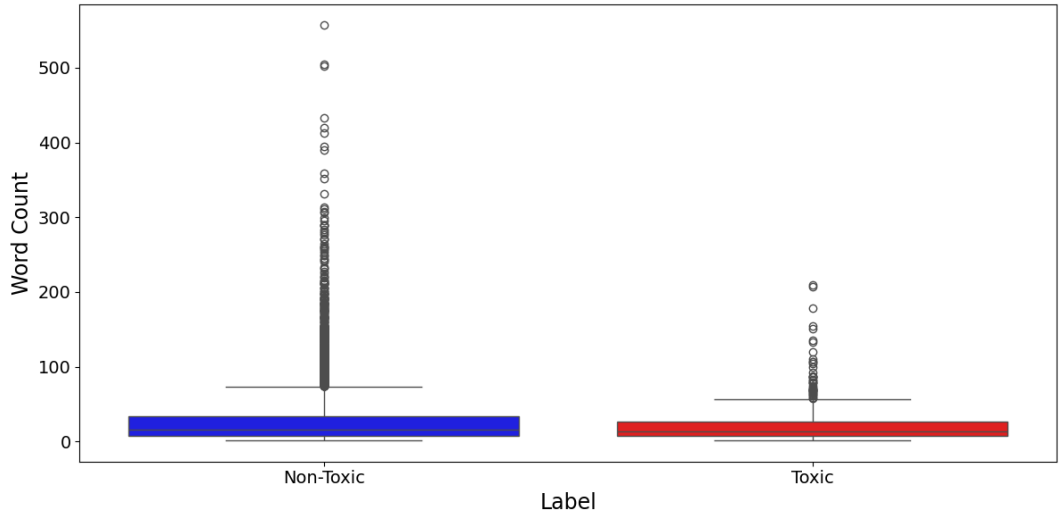

Figure 4.1: Box-and-Whiskers Plot illustrating word count distribution

toxic comments are generally shorter in length compared to non-toxic comments.

Figure 4.1 illustrates the word count of both toxic and non-toxic comments in a box-and-whiskers plot. The figure demonstrates that the interquartile range is slightly larger for non-toxic comments, indicating greater variability in word count for non-toxic comments. Furthermore, no toxic comment exceeds 250 words, whereas non-toxic comments have outliers exceeding 300 words. Therefore, toxic comments are typically shorter in both word count and character count compared to non-toxic comments.

4.1.2 Multi-label Classification Dataset

Unlike the binary classification dataset, the multi-label classification dataset involves categorizing comments based on the toxic classes outlined in Section 2.1.1. The dataset for multi-label classification is derived from the same subreddits as the binary classification dataset. However, each comment is annotated with one or more toxic classes, providing a richer set of information for each comment.

Figure 4.2 illustrates the distribution of each individual toxic class. The figure demonstrates that the classes severe toxic, threat and identity hate occur far less frequently than the other three classes. The rare occurrence of severe toxic, threat, and identity hate comments intensifies the class imbalance problem [1], making it more difficult for models to learn to identify these comments accurately. The limited availability of these classes makes it challenging for the model to effectively learn and generalize from them.

Regarding the toxic class, there is a decline from 435 to 423 when moving from

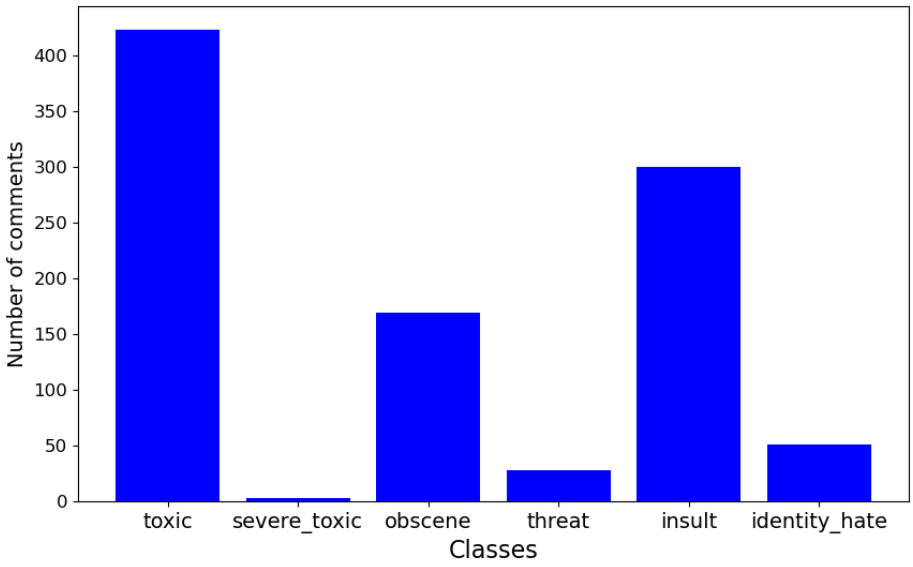

Figure 4.2: Distribution of toxic classes

binary to multi-label classification, due to two reasons: First, some toxic comments are reclassified as only belonging to the obscene class but not to the toxic class. Secondly, some comments that are previously considered toxic are no longer classified as belonging to any toxic class. This may be attributed to the fact that in multi-label classification, comments are classified without further context. Whereas, during binary classification, the comments are labeled with context. Thus, a comment might be perceived as toxic in one context but not in another [1]. This issue highlights the struggle to accurately label toxic comments [36].

The decrease in toxic comments also reflects the difficulties involved in the annotation process [9]. Demus et al. [10] note that there are often borderline cases that are difficult to label unambiguously, as hate speech is difficult to define precisely. These doubtful labels are a major factor contributing to false classifications, particularly false positives [9]. As a result, the performance of models trained on such data tends to suffer.

To address these issues, several solutions can be implemented. First, the selection of annotators can influence the quality of labels [9]. Moreover, Waseem [37] concludes that models trained with expert annotations significantly outperform those trained with annotations from amateurs. Furthermore, using multiple annotators could help to improve annotation quality. In practice, inter-annotator agreement is often used as a measure of annotation quality: the greater the agreement, the better the annotations [10]. In conclusion, the labeling process might influence the overall

effectiveness of models.

4.2 Embedding Selection

Table 4.2 illustrates the results of the preliminary experiment detailed in Section 3.3. The table presents an analysis of the algorithms across four embeddings. For each algorithm-embedding pair, the table reports the precision, recall, and F1-score for the best-performing parameters. Furthermore, for each algorithm, the embedding that achieves the highest F1-score is highlighted.

Algorithm	Embedding	Precision	Recall	F1-Score
Logistic Regression	**TF-IDF**	**0.267**	**0.307**	**0.286**
	Word2Vec	0.108	0.518	0.179
	FastText	0.125	0.573	0.206
	Doc2Vec	0.095	0.468	0.157
SVM	TF-IDF	0.339	0.092	0.144
	Word2Vec	0.160	0.427	0.232
	FastText	**0.205**	**0.289**	**0.240**
	Doc2Vec	0.128	0.188	0.152
Decision Tree	TF-IDF	0.052	0.262	0.086
	Word2Vec	0.081	0.335	0.130
	FastText	**0.091**	**0.372**	**0.146**
	Doc2Vec	0.069	0.381	0.117
Random Forest	TF-IDF	0.070	0.211	0.105
	Word2Vec	0.333	0.023	0.043
	FastText	**0.284**	**0.115**	**0.163**
	Doc2Vec	0.189	0.064	0.096
XGBoost	TF-IDF	0.151	0.220	0.179
	Word2Vec	0.166	0.239	0.184
	FastText	**0.164**	**0.243**	**0.196**
	Doc2Vec	0.093	0.266	0.138

Table 4.2: Comparison of algorithms with different embeddings

Across the board, FastText emerges as the best-performing embedding for SVM, Decision Tree, Random Forests and XGBoost. This suggests that FastText's skip-gram approach plays a key role in improving classification performance across various algorithms. The results further reinforces Risch's observation [1] that FastText is effective in managing toxic comments, as its subword embeddings address out-of-vocabulary words.

On the other hand, for Logistic Regression, TF-IDF is the most effective embedding.

Furthermore, this combination achieves the best performance overall. Interestingly, while all other embeddings result in higher recall scores for Logistic Regression, they also exhibit lower precision scores. This suggests that TF-IDF might offer a better balance between precision and recall in this particular context. Moreover, this highlights the trade-off between the two, where the other three embeddings tend to prioritize recall at the expense of precision. Additionally, the lower F1-scores for the other embeddings can be attributed to the harmonic mean, which places significantly more emphasis on lower values.

Across all algorithms, Doc2Vec consistently underperforms compared to the other embeddings. This suggests that it might not be the most suitable for toxic comment classification. Additionally, although Word2Vec generally shows similar scores as FastText, it performs notably worse with Random Forests.

In conclusion, FastText emerges as the best-performing embedding across multiple algorithms, while Doc2Vec performs poorly in comparison. Furthermore, the best performance only achieves a F1-score of 0.286, indicating that there is still considerable room for improving the classifiers.

4.3 Self-Training Results

4.3.1 Expansion Sets

The results of the self-training experiment are presented in Table 4.3. The table provides a comprehensive comparison of the algorithms across the various expansion sets outlined in Section 3.4.2. Furthermore, the table includes a baseline, which serves as a reference point to evaluate performance changes. Additionally, the top-performing dataset for each algorithm is highlighted.

The table reveals that the addition of expansion sets generally influences performance across all algorithms. However, adding expansion sets typically decreases the F1-score. In fact, the base model outperforms the expanded models for all algorithms, except for SVM. Nevertheless, in the case of SVM, the base model performs almost as well as the two better-performing expanded models.

Examining the algorithms individually reveals diverse patterns. For Decision Trees, recall increases noticeably with each expansion set, excluding expansion set 1. Similarly, Logistic Regression shows an improvement in recall. However, while the expansion sets increase recall, precision tends to decrease, resulting in a lower F1-score compared to the base model.

Algorithm	Dataset	Precision	Recall	F1-Score
Logistic Regression	**Baseline**	**0.267**	**0.307**	**0.286**
	Expansion Set 1	0.217	0.317	0.258
	Expansion Set 2	0.206	0.372	0.265
	Expansion Set 3	0.162	0.349	0.221
	Expansion Set 4	0.186	0.404	0.255
SVM	Baseline	0.175	0.262	0.210
	Expansion Set 1	**0.176**	**0.271**	**0.213**
	Expansion Set 2	0.173	0.271	0.211
	Expansion Set 3	0.172	0.271	0.210
	Expansion Set 4	0.167	0.257	0.202
Decision Trees	**Baseline**	**0.095**	**0.349**	**0.150**
	Expansion Set 1	0.060	0.202	0.093
	Expansion Set 2	0.062	0.413	0.107
	Expansion Set 3	0.057	0.633	0.104
	Expansion Set 4	0.053	0.596	0.098
Random Forests	**Baseline**	**0.284**	**0.115**	**0.163**
	Expansion Set 1	0.146	0.028	0.046
	Expansion Set 2	0.140	0.037	0.058
	Expansion Set 3	0.138	0.060	0.083
	Expansion Set 4	0.127	0.046	0.067
XGBoost	**Baseline**	**0.102**	**0.353**	**0.158**
	Expansion Set 1	0.086	0.312	0.135
	Expansion Set 2	0.082	0.372	0.134
	Expansion Set 3	0.096	0.307	0.146
	Expansion Set 4	0.087	0.266	0.131

Table 4.3: Comparison of algorithms with different expansion sets

SVM demonstrates a slight performance improvement for only the first two expansion sets. Specifically, for the first expansion set, the recall and precision scores reach their peak. Conversely, Decision Trees and Random Forests display the largest decrease in F1-score with the expansion sets. Specifically, Random Forests experiences an average drop of 9.95 percentage points. Interestingly, as seen in Table 3.2, Random Forests classifies the least number of comments as toxic in each expansion set compared to the other algorithms.

In conclusion, the results indicate that utilizing expansion sets generally diminishes performance. Interestingly, the decrease in performance is not linearly related to the increase in training data. This observation suggests that simply increasing the size of expansion sets may not always lead to worse model performance, highlighting the need for a balanced approach between labeled data and pseudo-labels.

4.3.2 Toxic Sets

Table 4.4 presents the findings of the self-training experiment using toxic sets instead of expansion sets. As the experiment employs the same base model as the expansion set self-training approach, it also includes the base model as a reference point. Additionally, Section A.6 in the appendix presents the results of the expansion and toxic set experiments in one combined table.

Algorithm	Dataset	Precision	Recall	F1-Score
Logistic Regression	**Baseline**	**0.267**	**0.307**	**0.286**
	Toxic Set 1	0.234	0.326	0.273
	Toxic Set 2	0.216	0.317	0.257
	Toxic Set 3	0.148	0.413	0.218
	Toxic Set 4	0.128	0.436	0.198
SVM	Baseline	0.175	0.262	0.210
	Toxic Set 1	**0.181**	**0.321**	**0.232**
	Toxic Set 2	0.163	0.326	0.217
	Toxic Set 3	0.133	0.385	0.198
	Toxic Set 4	0.129	0.399	0.195
Decision Trees	**Baseline**	**0.095**	**0.349**	**0.150**
	Toxic Set 1	0.064	0.436	0.111
	Toxic Set 2	0.048	0.399	0.086
	Toxic Set 3	0.048	0.628	0.090
	Toxic Set 4	0.051	0.849	0.096
Random Forests	**Baseline**	**0.284**	**0.115**	**0.163**
	Toxic Set 1	0.159	0.046	0.071
	Toxic Set 2	0.143	0.051	0.075
	Toxic Set 3	0.076	0.128	0.096
	Toxic Set 4	0.076	0.142	0.099
XGBoost	**Baseline**	**0.102**	**0.353**	**0.158**
	Toxic Set 1	0.071	0.353	0.118
	Toxic Set 2	0.071	0.321	0.117
	Toxic Set 3	0.073	0.518	0.128
	Toxic Set 4	0.061	0.628	0.111

Table 4.4: Comparison of algorithms with different toxic sets

In the table, we observe a similar effect to that of the expansion set experiment. The vast majority of toxic sets result in a decrease in the F1-score. In terms of F1-score, only SVM has toxic sets that marginally outperform the base model. Furthermore, Toxic Sets 1 and 2 exceed the performance of their counterparts, Expansion Sets 1 and 2, for SVM.

With the exception of XGBoost, Toxic Set 1 demonstrates better performance for

each algorithm compared to Expansion Set 1. However, this relationship is not observed with larger toxic sets. Furthermore, recall reaches its peak for all algorithms with Toxic Set 4.

Similar to the first experiment, there is no linear relationship between the increasing size of the toxic sets and an increase or decrease in performance. However, the expanded datasets, particularly Toxic Set 4, lead to an increase in recall at the expense of precision. In conclusion, the addition of toxic sets generally appears to diminish overall performance.

4.4 Error Analysis

Our evaluation results suggest that adding pseudo-labels generally negatively affects model performance. This may be attributed to the fact that self-training is usually unsuccessful if the base model performs poorly [5]. Various studies propose numerous approaches to improve the base model such as oversampling [31], undersampling [5] and Synthetic Minority Oversampling Technique (SMOTE) [17]. However, these approaches lead to worse results in our experiment. The best-performing base model only achieves a F1-score of 0.286. This score is lower than the F1-scores in a study by Roy [10] that uses Random Forests and SVM, where the scores were around 0.54.

Faruque [15] emphasizes that the performance of SSL depends on the quality of the labeled and unlabeled data. Consequently, ensuring high-quality data is crucial for successful SSL. Given the outcomes in our experiments, we conduct a thorough error analysis in this section, drawing on the insights from Van Aken et al. [9]. The analysis identifies potential sources of error and factors contributing to the poor results.

4.4.1 Common Challenges in Toxic Classification

Out-of-vocabulary words

Models struggle to identify toxic comments that contain words that are not present in the training set [9]. These words may include new swear words, spelling errors, or deliberately hidden toxic language. For instance, consider Example 4.4.1 that contains a comment from the test set which was misclassified by each algorithm, because the swear word "*pussy*" never appears in the training set.

Example 4.4.1. *Sport ist einfach nur noch widerlich.Überall nur noch pussies und Marketing...*

Long Comments

According to Van Aken et al. [9], toxic comments are often determined by the language used in the initial parts of the comments. However, the impact of these initial parts diminishes in longer comments, specifically those with more than 50 words. Consequently, this can affect the classification of comments. For example, if a comment begins with an insult but the remainder is lengthy and non-toxic, the overall classification might be non-toxic. As seen in Figure 4.1 some toxic comments exceed the 50-word-limit. As a result, this might explain some of the misclassifications of the base models.

Doubtful labels

Another issue may be the presence of comments in the dataset for which the original labels appear questionable. Consequently, this can lead to false positives and false negatives during classification [9]. This problem is made worse by the fact that labeling is done by a single amateur annotator. Employing multiple annotators rather than relying on a single person tends to improve label quality, consequently improving performance [9]. Furthermore, models that are trained with expert annotations perform significantly better than those with annotations from less experienced annotators [37]. Therefore, the results in our experiment could be attributed to the reliance on a single annotator.

Toxicity in Comments

Davidson et al. [38] note that a frequent issue is that toxic comments often lack explicit swear words or overtly hateful language, which complicates accurate classification. Consequently, van Aken et al. [9] explains that this often leads to false negatives in the classification. On the other hand, there are also comments that contain swear words, but they are considered non-toxic. For instance, comments referring to toxic comments or quoting other toxic comments. Example 4.4.2 does not contain any swear words, but is clearly an insult. Furthermore, it was also misclassified by each algorithm.

Example 4.4.2. *Ey nicht böse gemeint, wirklich, aber man hätte ja ein besseres Bild nehmen können. Man denkt der hat Down-Syndrom mit dem offenen Mund und den Augen.*

4.4.2 Challenges in Implementing Pseudo-Labeling

One of the challenges explored in SSL is the degradation of classifier performance due to the increasing quantity of unlabeled data added to the training data [13]. Moreover, self-training relies heavily on the quality of pseudo-labels generated by the base model [15]. According to Sosea [14], self-training methods address this by ignoring examples where the base model is uncertain. However, relying solely on the base model's confidence is risky, especially if the base model performs poorly. This issue is further worsened by the challenges mentioned earlier, which also impact the quality of pseudo-labels during self-training. Consequently, the decline in pseudo-label quality negatively affects the overall performance [15].

In conclusion, the combination of a weak base model [14], questionable labels [15], and the inherent challenges of toxic comment classification [9] may contribute to the poor performance of self-training. Addressing these challenges is crucial for successfully implementing SSL in toxic comment classification.

Chapter 5

Conclusions and Future Work

This chapter revisits the main methods and results. Additionally, it provides an outlook for future research. Section 5.1 reflects on the results of the experiment and its implications. Subsequently, Section 5.2 discusses potential future areas of research in toxic comment classification.

5.1 Conclusions

This thesis investigates the use of self-training for detecting toxic German comments. The thesis addresses the challenge of limited labeled data for toxic comment classification [1] by employing self-training techniques to enhance model performance [13].

To comprehensively analyze the effects of self-training, five ML algorithms are employed in our evaluation: Logistic Regression, SVM, Decision Trees, Random Forests, and XGBoost. Moreover, two distinct self-training approaches are utilized. The first approach involves self-training with pseudo-labels. The second approach focuses on self-training with only toxic comments, specifically targeting the minority class to address class imbalance issues.

To address the challenges in toxic comment classification and explore the potential of self-training, three key research questions are formulated. These questions focus on different aspects of self-training and its impact on model performance. The experiment, including the choice of ML algorithms and self-training approaches, are designed to comprehensively answer these questions.

RQ1: *How does self-training with pseudo–labels influence the performance for detecting toxic comments in German?*

Our results suggest that self-training with pseudo-labels diminishes the overall performance for detecting toxic comments. Only SVM demonstrates a slight performance increase with the two smaller pseudo-labeled datasets. This outcome is unexpected, as multiple research papers, such as the studies by Ahmadinejad et al. [32] and Faruque [15], successfully implement self-training in toxic comment classification. The diminishing effects of self-training may be largely attributed to the poor performance of the base model. Since the base model performs poorly, it consequently predicts incorrect pseudo-labels, which then in turn reduces the performance of the expanded model. In conclusion, self-training does not seem to significantly enhance the performance of a weak classifier.

RQ2: *How does augmenting the training set with only toxic pseudo-labeled comments affect the model's performance compared to using the complete pseudo-labeled dataset?*

Similar to the results observed in RQ1, augmenting the training set with only toxic pseudo-labeled comments generally leads to a decrease in performance. Furthermore, SVM also experiences a slight increase with two toxic sets suggesting that SVM may handle wrong pseudo-labels better than other algorithms. Additionally, adding only toxic pseudo-labeled comments generally leads to an increase in recall at the expense of precision.

RQ3: *What impact do variations in the quantity of pseudo-labeled data in the training set have on model performance?*

Interestingly, improvements in SVM performance are observed only when the pseudo-labeled datasets are smaller than or equal to the size of the original training set. For both approaches, there is no linear relationship between model performance and the increase in pseudo-labeled data. This suggests that there may be an optimal ratio of pseudo-labeled data and labeled data during. In other words, merely increasing the pseudo-labeled data will not always lead to further decreases or increases in performance.

In conclusion, this thesis provides multiple insights into toxic comment classification and self-training. First, FastText emerges as the best-performing embedding for SVM, Decision Trees, Random Forests, and XGBoost, suggesting, that it is particularly suitable for toxic comments. Second, the experiment reveals that self-training leads to a decrease in performance if the initial performance of the base model is already poor. The results obtained were not as robust as initially anticipated due to limitations in the base model's performance. Consequently, this leads to the assumption that self-training might only improve already well-performing

models. Lastly, the relationship between model performance and the amount of pseudo-labeled data is not linear, indicating that there may be an optimal amount of pseudo-labeled data in the training set.

It is worth noting that even a poorly performing base model can still be useful for toxic comment classification. The base model could be used to create pseudo-labels, which then could be used as a basis for further labeling processes. In this context, the quality of pseudo-labels is less critical, since the comments pseudo-labeled as toxic only serve as a suggestion, which comments should be manually labeled. Consequently, this approach may increase the likelihood of identifying toxic comments, thus potentially reducing the issue of class imbalance [1].

5.2 Future Work

A potential limitation of this thesis is the poor performance of the base model. Consequently, the research questions outlined in Section 1.2 are not fully addressed in the present thesis. Self-training might yield better results if the base model already performs well. Therefore, future work could first focus on creating a well-performing base model before employing self-training approaches. This could involve using state-of-the-art DL methods to create a robust classifier. Subsequently, the experiment using self-training could be repeated and its results evaluated. Furthermore, an iterative approach to self-training [5] could also be examined.

Another area for future research could be using other methods of SSL instead of self-training. For instance, in co-training, two or more algorithms are trained on the same labeled data and then utilized to create pseudo-labels for the same unlabeled dataset [13]. Subsequently, only the most confident predictions from each classifier are utilized further as training data. This approach may help reduce the issue of having a single poorly performing base model.

Ultimately, SSL shows significant promise for toxic comment classification due to its ability to utilize both labeled and unlabeled data. However, further research is essential to fully realize its benefits and address emerging challenges.

Appendix A

Appendix Title

A.1 Hyperparameters of the Algorithms

Hyperparameter	Description
C	Inverse of the regularization strength.
max_iter	Maximum number of iterations for the solver to converge
class_weight	Weights associated with classes

Table A.1: Description of Logistic Regression hyperparameters utilized in the study. [34]

Hyperparameter	Description
classifier__C	Inverse of the regularization strength.
kernel	Specifies the kernel type to be used in the algorithm.
gamma	Kernel coefficient for 'rbf'

Table A.2: Description of SVM hyperparameters utilized in the study [34]

Hyperparameter	Description
_max_depth	Maximum depth of the decision tree.
min_samples_split	Minimum number of samples required to split an internal node.

Table A.3: Description of Decision Tree hyperparameters utilized in the study [34]

Hyperparameter	Description
n_estimators	Number of trees in the ensemble
min_samples_split	Minimum number of samples required to split an internal node.
max_depth	Maximum depth of the tree.

Table A.4: Description of Random Forest hyperparameters utilized in the study [34]

Hyperparameter	Description
n_estimators	Number of trees in the forest.
max_depth	Maximum depth of a tree.
learning_rate	Controls the shrinkage of each tree's contribution

Table A.5: Description of XGBoost hyperparameters utilized in the study [34]

A.2 Preprocessing Code

```
import pandas as pd
import string
import spacy as sp
import re

nlp = sp.load('de_core_news_md')

def lemmatize(text):
    doc = nlp(text)
    return ' '.join([token.lemma_ for token in doc])
def remove_punctuation(text):
    translator = str.maketrans('', '', string.punctuation)
    return text.translate(translator)
def remove_extra_whitespace(text):
    return re.sub(r'\s+', ' ', text).strip()
def remove_numbers(text):
    return re.sub(r'\d+', '', text)
def remove_emails_mentions(text):
    text = re.sub(r'\S+@\S+', '', text)
    text = re.sub(r'@\w+', '', text)
    return text
def replace_ascii(text):
    return re.sub(r'?', '', text)
def remove_urls(text):
    return re.sub(r'http\S+|www\S+|https\S+', '', text)
def preprocess_text(text):
    text = remove_urls(text)
```

```
    text = replace_ascii(text)
    text = remove_emails_mentions(text)
    text = remove_numbers(text)
    text = lemmatize(text)
    text = remove_punctuation(text)
    text = remove_extra_whitespace(text)
    text = text.lower()
    return text

test_data = pd.read_excel("TestSet.xlsx")
train_data = pd.read_excel("TrainingSet.xlsx")

train_data['comment'] = train_data['comment'].apply(preprocess_text)
test_data['comment'] = test_data['comment'].apply(preprocess_text)

train_data.to_excel('Preprocessed_TrainingSet.xlsx', index=False)
test_data.to_excel('Preprocessed_TestSet.xlsx', index=False)
```

Code A.1: Text preprocessing pipeline

A.3 Hyperparameter Grids for Embedding Models

Hyperparameter	Description	Values
`vector_size`	Dimensionality of the word vectors	50, 100, 200, 300
`window`	Maximum distance between the current and predicted word within a sentence.	5, 10, 15
`min_count`	Ignores all words with a total frequency lower than this threshold.	1, 5, 10, 100
`epochs`	Number of iterations over the entire training dataset.	10, 20, 50, 100, 200

Table A.6: Tuned hyperparameters for Word2Vec, FastText, and Doc2Vec

Hyperparameter	Description	Values
`tfidf_max_features`	Maximum number of features to be extracted by the TF-IDF vectorizer	None, 1000, 5000, 10000
`tfidf_ngram_range`	The lower and upper boundary of the range of n-values for different n-grams to be extracted	(1,1), (1,2), (2,2), (1,3), (3,3), (1,4), (4,4)

Table A.7: Tuned hyperparameters for TF-IDF Vectorizer

A.4 Embedding Parameters

Algorithm	vector_size	window	min_count	epochs
SVM	300	5	1	100
Decision Trees	100	5	1	100
Random Forests	50	15	1	20
XGBoost	50	5	1	100

Table A.8: Specific parameters used for FastText during the self-training process

A.5 Base Model Code

```
import pandas as pd
import numpy as np
from gensim.models import FastText
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score,
                            recall_score, f1_score
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import StratifiedKFold, GridSearchCV

def vectorize(sentence):
    words = sentence.split()
    words_vecs = [ft_model.wv[word] for word in words if word in ft_model.wv]
    if len(words_vecs) == 0:
        return np.zeros(300)  # Directly use the vector_size here
    words_vecs = np.array(words_vecs)
    return words_vecs.mean(axis=0)

# Load datasets
train_data = pd.read_excel("Preprocessed_TrainingSet.xlsx")
test_data = pd.read_excel("Preprocessed_TestSet.xlsx")
data = pd.read_excel("Data.xlsx")  # Data file for training FastText
pseudo_data = pd.read_excel("PseudoLabels.xlsx")
X_train = train_data['comment']
y_train = train_data['label']
X_test = test_data['comment']
y_test = test_data['label']

# Train FastText model
sentences = [sentence.split() for sentence in data['comment']]
ft_model = FastText(vector_size=300, window=5, min_count=1,
                    workers=4, epochs=100)
ft_model.build_vocab(sentences)
```

```
ft_model.train(sentences, total_examples=len(sentences), epochs=100)

# Vectorize train data
X_train = np.array([vectorize(sentence) for sentence in X_train])

# Define pipeline and parameter grid for GridSearchCV
scaler = StandardScaler()
model = Pipeline([('scaler', scaler),
                  ('classifier', SVC(class_weight='balanced'))])

param_grid = {
    'classifier__C': [0.01, 0.1, 1, 10, 100],
    'classifier__kernel': ['rbf'],
    'classifier__gamma': ['scale', 'auto', 0.1, 1, 10, 100]
}

# Grid search with stratified cross-validation
kf = StratifiedKFold(n_splits=5, shuffle=True)
grid_search = GridSearchCV(model, param_grid, scoring='f1', cv=kf, n_jobs=-1)
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_

# Train the best model on the entire training set
best_model.fit(X_train, y_train)

# Vectorize test data and predict test labels
X_test = np.array([vectorize(sentence) for sentence in X_test])
predicted = best_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, predicted)
precision = precision_score(y_test, predicted, average='binary',
                            zero_division=1)
recall = recall_score(y_test, predicted, average='binary', zero_division=1)
f1 = f1_score(y_test, predicted, average='binary', zero_division=1)

print(f"\nTest Set Evaluation")
print(f'Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1-score: {f1:.4f}')
print(f'Best SVM Parameters: {grid_search.best_params_}')

# Create a file with misclassified comments
misclassified = test_data[predicted != y_test].copy()
misclassified['Predicted_Label'] = predicted[predicted != y_test]
```

```
misclassified['Real_Label'] = y_test[predicted != y_test]
misclassified.to_excel("Misclassified_Comments.xlsx", index=False)

# Save PseudoLabels
X_pseudo = pseudo_data['comment']
X_pseudo = np.array([vectorize(sentence) for sentence in X_pseudo])
predicted_pseudo = best_model.predict(X_pseudo)
pseudo_data['Predicted_Label'] = predicted_pseudo
pseudo_data.to_excel("PseudoLabels_Predictions.xlsx", index=False)
```

Code A.2: Example of a base model for self-training using SVM with FastText

A.6 Self-training Results

Algorithm	Dataset	Precision	Recall	F1-Score
Logistic Regression	Baseline	0.267	0.307	0.286
	Expansion Set 1	0.217	0.317	0.258
	Expansion Set 2	0.206	0.372	0.265
	Expansion Set 3	0.162	0.349	0.221
	Expansion Set 4	0.186	0.404	0.255
	Toxic Set 1	0.234	0.326	0.273
	Toxic Set 2	0.216	0.317	0.257
	Toxic Set 3	0.148	0.413	0.218
	Toxic Set 4	0.128	0.436	0.198
SVM	Baseline	0.175	0.262	0.210
	Expansion Set 1	0.176	0.271	0.213
	Expansion Set 2	0.173	0.271	0.211
	Expansion Set 3	0.172	0.271	0.210
	Expansion Set 4	0.167	0.257	0.202
	Toxic Set 1	0.181	0.321	0.232
	Toxic Set 2	0.163	0.326	0.217
	Toxic Set 3	0.133	0.385	0.198
	Toxic Set 4	0.129	0.399	0.195
Decision Trees	Baseline	0.095	0.349	0.150
	Expansion Set 1	0.060	0.202	0.093
	Expansion Set 2	0.062	0.413	0.107
	Expansion Set 3	0.057	0.633	0.104
	Expansion Set 4	0.053	0.596	0.098
	Toxic Set 1	0.064	0.436	0.111
	Toxic Set 2	0.048	0.399	0.086
	Toxic Set 3	0.048	0.628	0.090
	Toxic Set 4	0.051	0.849	0.096
Random Forests	Baseline	0.284	0.115	0.163
	Expansion Set 1	0.146	0.028	0.046
	Expansion Set 2	0.140	0.037	0.058
	Expansion Set 3	0.138	0.060	0.083
	Expansion Set 4	0.127	0.046	0.067
	Toxic Set 1	0.159	0.046	0.071
	Toxic Set 2	0.143	0.051	0.075
	Toxic Set 3	0.076	0.128	0.096
	Toxic Set 4	0.076	0.142	0.099
XGBoost	Baseline	0.102	0.353	0.158
	Expansion Set 1	0.086	0.312	0.135
	Expansion Set 2	0.082	0.372	0.134
	Expansion Set 3	0.096	0.307	0.146
	Expansion Set 4	0.087	0.266	0.131
	Toxic Set 1	0.071	0.353	0.118
	Toxic Set 2	0.071	0.321	0.117
	Toxic Set 3	0.073	0.518	0.128
	Toxic Set 4	0.061	0.628	0.111

Table A.9: Comparison of algorithms with different pseudo-labeled datasets

Bibliography

[1] J. Risch and R. Krestel. *Toxic Comment Detection in Online Discussions*, pages 85–109. Springer Singapore, Singapore, 2020.

[2] D. Androcec. Machine learning methods for toxic comment classification: a systematic review. *Acta Universitatis Sapientiae, Informatica*, 12:205–216, 2020.

[3] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020.

[4] E. Abate-Daga. Künstliche intelligenz zur Analyse natürlicher Sprache - Revolutionierung der Kundenfeedbackauswertung. *Wirtschaftsinformatik und Management*, 16:12–17, 2024.

[5] S. Lee and W. Kim. Sentiment labeling for extending initial labeled data to improve semi-supervised sentiment classification. *Electronic Commererce Research and Applications*, 26:35–49, 2017.

[6] M. Ibrahim, M.Torki, and N. M. El-Makky. Imbalanced toxic comments classification using data augmentation and deep learning. In *17th IEEE International Conference on Machine Learning and Applications, ICMLA 2018, Orlando, FL, USA*, pages 875–878. IEEE, 2018.

[7] M. Mironczuk and J. Protasiewicz. A recent overview of the state-of-the-art elements of text classification. *Expert Systems with Applications*, 106:36–54, 2018.

[8] V. Shankaran and Rajesh Sharma. Analyzing toxicity in deep conversations: A Reddit case study. *CoRR*, 2024.

[9] B. van Aken, J. Risch, R.Krestel, and A. Löser. Challenges for toxic comment classification: An in-depth error analysis. In *Proceedings of the 2nd Workshop on Abusive Language Online, ALW@EMNLP 2018, Brussels, Belgium*, pages 33–42. Association for Computational Linguistics, 2018.

[10] C. Demus, D. Labudde, J. Pitz, N. Probol, M. Schütz, and M. Siegel. *Automatische Klassifikation offensiver deutscher Sprache in sozialen Netzwerken*, pages 65–88. Springer Berlin Heidelberg, Berlin, Heidelberg, 2023.

[11] J. Risch, A. Stoll, L. Wilms, and M. Wiegand. Overview of the germeval 2021 shared task on the identification of toxic, engaging, and fact-claiming comments. In *Proceedings of the GermEval 2021 Shared Task on the Identification of Toxic, Engaging, and Fact-Claiming Comments, GermEval 2021, Düsseldorf, Germany, September 6, 2021*, pages 1–12. Association for Computational Linguistics, 2021.

[12] J. Risch, P. Schmidt, and R. Krestel. Data integration for toxic comment classification: Making more than 40 datasets easily accessible in one unified format. In *Proceedings of the 5th Workshop on Online Abuse and Harms (WOAH 2021)*, pages 157–163, Online, August 2021. Association for Computational Linguistics.

[13] J. M. Duarte and L. Berton. A review of semi-supervised learning for text classification. *Artificial Intelligence Review*, 56(9):9401–9469, 2023.

[14] T. Sosea and C. Caragea. Leveraging training dynamics and self-training for text classification. pages 4750–4762, 2022.

[15] A. Faruque and H. Mustafa. A semi-supervised approach for identifying cyberbullying text in bangla language. In *2024 6th International Conference on Electrical Engineering and Information Communication Technology (ICEE-ICT)*, pages 628–633, 2024.

[16] P. Fortuna, J. S. Company, and L. Wanner. Toxic, hateful, offensive or abusive? What are we really classifying? An empirical analysis of hate speech datasets. In *Proceedings of The 12th Language Resources and Evaluation Conference, LREC 2020, Marseille, France*, pages 6786–6794. European Language Resources Association, 2020.

[17] K. Poojitha, A. S. Charish, M. A. K. Reddy, and S. Ayyasamy. Classification of social media toxic comments using machine learning models. *CoRR*, 2023.

[18] J. Sorensen, L. Dixon J. Elliott, M. McDonald, and W. Cukierski. Toxic comment classification challenge-Kaggle, 2017.

[19] A. Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, 2022.

[20] C. Rastogi, N. Mofid, and F. Hsiao. Can we achieve more with less? Exploring data augmentation for toxic comment classification. *CoRR*, 2020.

[21] R.Patil, S. Boit, V. N. Gudivada, and J. Nandigam. A survey of text representation and embedding techniques in NLP. *IEEE Access*, 11:36120–36146, 2023.

[22] S. J. Johnson, M. Ramakrishna Murty, and I. Navakanth. A detailed review on word embedding techniques with emphasis on word2vec. *Multim. Tools Appl.*, 83:37979–38007, 2024.

[23] X. Rong. word2vec parameter learning explained. *CoRR*, 2016.

[24] A. G. D'Sa, I. Illina, and D. Fohr. BERT and fasttext embeddings for automatic detection of toxic speech. In *International Multi-Conference on: "Organization of Knowledge and Advanced Technologies", Tunis, Tunisia*, pages 1–5. IEEE, 2020.

[25] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *CoRR*, 2016.

[26] H. Aman, S. Amasaki, T. Yokogawa, and M. Kawahara. A doc2vec-based assessment of comments and its application to change-prone method analysis. In *25th Asia-Pacific Software Engineering Conference, APSEC 2018, Nara, Japan*, pages 643–647. IEEE, 2018.

[27] B. Dhariyal, V. Ravi, and K. Ravi. Sentiment analysis via doc2vec and convolutional neural network hybrids. In *IEEE Symposium Series on Computational Intelligence, SSCI 2018, Bangalore, India*, pages 666–671. IEEE, 2018.

[28] J.H. Lau and T. Baldwin. An empirical evaluation of doc2vec with practical insights into document embedding generation. In *Proceedings of the 1st Workshop on Representation Learning for NLP, Rep4NLP@ACL 2016, Berlin, Germany*, pages 78–86. Association for Computational Linguistics, 2016.

[29] N. B. Noor, N. Yousefi, B. Spann, and N. Agarwal. Comparing toxicity across social media platforms for COVID-19 discourse. *CoRR*, 2023.

[30] G. Rudiger. Reddit scraper-apify, Feb 2022.

[31] A. Datta, B. M. Kumar, and A. S. Sairam. Detecting toxic comments from highly skewed social media data. In *2023 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, pages 744–749. IEEE, 2023.

[32] M. Ahmadinejad, N. Shahriar, and L. Fan. *Self-Training for Cyberbully Detection: Achieving High Accuracy with a Balanced Multi-Class Dataset*. PhD thesis, PhD thesis, Faculty of Graduate Studies and Research, University of Regina, 2023.

[33] V. Balakrishnan and E. Lloyd-Yemoh. Stemming and lemmatization: A comparison of retrieval performances. 2014.

[34] 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*, 12:2825–2830, 2011.

[35] A. Stanescu and D. Caragea. Semi-supervised self-training approaches for imbalanced splice site datasets. In *Proceedings of the 6th International Conference on Bioinformatics and Computational Biology, BICoB*, pages 131–136, 2014.

[36] T. Mandl. *KI-Verfahren für die Hate Speech Erkennung: Die Gestaltung von Ressourcen für das maschinelle Lernen und ihre Zuverlässigkeit*, pages 111–130. Springer Berlin Heidelberg Berlin, Heidelberg, 2023.

[37] 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, NLP+CSS@EMNLP 2016, Austin, TX, USA*, pages 138–142. Association for Computational Linguistics, 2016.

[38] T. Davidson, D. W., M. W. Macy, and I. Weber. Automated hate speech detection and the problem of offensive language. In *Proceedings of the Eleventh International Conference on Web and Social Media, ICWSM 2017, Montréal, Québec, Canada*, pages 512–515. AAAI Press, 2017.