



OULUN AMMATTIKORKEAKOULU

Yury Sergeev

INAPPROPRIATE CONTENT CLASSIFICATION - NATURAL LANGUAGE PROCESSING

INAPPROPRIATE CONTENT CLASSIFICATION - NATURAL LANGUAGE PROCESSING

Yury Sergeev
Master Thesis
Autumn 2023
The Degree Programme in Data
Analytics and Project Management
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
The Degree Programme in Data Analytics and Project Managements

Author(s): Yury Sergeev

Title of the thesis: Inappropriate content classification - Natural Language Processing

Thesis examiner(s): Ilpo Virtanen

Autumn 2023

Pages: 66

The main objective of this work was to investigate the effectiveness of various Natural Language Processing (NLP) techniques in processing and analyzing text data. The study focused on identifying and detecting toxic comments, which is becoming increasingly important in the digital age due to the massive amount of user-generated content on social media and online platforms. Robust methods are needed to monitor and filter harmful or inappropriate language to create a safer and more respectful online environment. This research aimed to contribute to this effort by efficiently identifying toxic commentary.

The study utilized the Jigsaw Multilingual Toxic Comment dataset to train several models. This dataset is valuable because it contains a diverse and comprehensive collection of comments, which can simulate real-world online interactions. The research involved experimenting with various word embedding techniques, such as Word2Vec, TF-IDF, GloVe, FastText, and specialized embedding layers, to represent text data effectively. These techniques play a crucial role in translating human language into formats that can be understood and processed by machine learning models. For the classification task, a diverse set of models was employed, including both traditional machine learning algorithms like Naive Bayes, Random Forest, Logistic Regression, LinearSVM, and XGBoost, as well as advanced deep learning models such as Convolutional Neural Network (CNN), Recurrent Network (RNN), Long-short Term Memory (LSTM), Gated Recurrent Units (GRU), Bidirectional Encoder Representations from Transformers (BERT), a distilled version of BERT (DistilBert), and XLM-RoBERTa (XLM-R). This comprehensive approach was designed to assess and compare the capabilities of different models in accurately identifying toxic comments.

XLM-RoBERTa was the most effective model, with an accuracy of 96% and an F1 score of 88% in detecting toxic comments. This high level of performance indicates the model's robustness and reliability in detecting toxic comments in diverse contexts. Further solidifying its practical applicability, the best-performing model was then tested on real-world data obtained from Twitter, aiming to detect inappropriate tweets in a live environment.

Keywords: Natural Language Processing, Machine Learning, Deep Learning, Toxic Comment Classification

CONTENTS

1.	Introduction.....	1
1.1	Overview	1
1.2	The structure of the thesis.....	2
2.	Data and Preprocessing.....	3
2.1	Dataset overview	3
2.2	Text pre-processing	5
2.2.1	Remove Punctuation	6
2.2.2	Stopword removal	6
2.2.3	Stemming.....	6
2.2.4	Lemmatization	6
2.2.5	Bag of Words	6
2.2.6	Word embeddings	7
2.2.7	Tokenization.....	9
2.3	Imbalanced Data	9
3.	Methods	10
3.1	Basic classification techniques.....	10
3.1.1	Naïve Bayes Classifier	10
3.1.2	Random forest.....	11
3.1.3	Support Vector Machine	11
3.1.4	Logistic Regression.....	12
3.2	Gradient tree boosting.....	13
3.2.1	XGBoost	13
3.3	Convolutional Neural Network	14
3.4	Recurrent Neural Network	15
3.4.1	Long Short Term Memory	16
3.4.2	Gated Recurrent Network	16
3.5	Transformer	17
3.5.1	BERT	18
3.5.2	DistiBert.....	19
3.5.3	XLM-RoBERTa	19
4.	Experiments	20
4.0.1	Experimental Setup.....	20
4.0.2	Grid Search for model tuning	20
4.0.3	Model evaluation	20
4.0.4	Performance evaluation on the test set.....	24
4.0.5	Logistic regression + TF-IDF	26
4.0.6	Naive Bayes + TF-IDF.....	29
4.0.7	Random forest + TF-IDF	31
4.0.8	LinearSVM + TF-IDF.....	33

4.0.9	XGBoost	36
4.0.10	BERT Base	39
4.0.11	DistiBert.....	42
4.0.12	XLM-RoBERTa.....	44
4.0.13	CNN.....	46
4.0.14	RNN.....	50
4.0.15	LSTM.....	54
4.0.16	GRU	57
5.	Testing.....	60
6.	Deployment.....	61
7.	Conclusion	62
8.	Limitations and future work.....	63
8.0.1	Limitation	63
8.0.2	Future work.....	63

LIST OF SYMBOLS AND ABBREVIATIONS

NLP	Natural language processing
AI	Artificial Intelligence
DNN	Deep Neural Network
TF-IDF	Term Frequency-Inverse Document Frequency
SVM	Support Vector Machine
API	Application Programming Interface
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory
GRU	Gated Recurrent Unit
BERT	Bidirectional Encoder Representations from Transformers
XLM-R	XLM-RoBERTa, Unsupervised Cross-lingual Representation Learning at Scale
MLM	Masked language modelling
NSP	Next Sentence Prediction
BPTT	Back Propagation Through Time
CNN	Convolutional Neural Network
LinearSVM	Linear Support Vector Machine
GloVe	Global Vectors for Word Representation

1. Introduction

1.1 Overview

With the rapid advancement of internet technology and the growth of online communication platforms in recent years, a vast amount of data is flooding every aspect of our lives. Unfortunately, this increase in text information has caused some issues for people. Many users frequently write inappropriate content, such as toxic comments, hate speeches, and offensive messages, which can create an unpleasant atmosphere and harm the online community. Removing toxic content and banning users who post it quickly is crucial to maintain a positive environment. Ignoring this problem can negatively impact the platform and users' overall experience. Several companies have faced problems due to inappropriate discussions on their platforms. For example, Facebook has been accused of not doing enough to combat hate speech and misinformation. Over 1,000 companies participated in a Facebook advertising boycott in 2020 to protest the company's handling of this issue (Hsu and Lutz 2021). Twitter has also received criticism for not moderating harmful content effectively, leading to calls for more robust moderation policies and increased transparency (Dang and Paul 2022). According to CASM Technology and ISD research, anti-Semitic tweets doubled from June 2022 to February 2023 (*Antisemitism on Twitter Before and After Elon Musk's Acquisition* 2023). The study found that while takedowns of such content increased, they did not keep pace with the surge.

It is important to note that certain legal obligations regarding moderation are mandated by law. One such requirement is the Digital Services Act, which the EU enforced on November 1, 2022 (*DSA: landmark rules for online platforms enter into force* 2022). The purpose of this act is to limit the spread of illegal content online. To achieve this, platforms are required to establish clear and transparent content moderation policies and implement mechanisms for the swift removal of illicit content, such as hate speech, upon receiving notifications from users or authorities.

Many companies rely on manual content moderation instead of automated systems, which is very time-consuming and can cause delays in responding to information. Additionally, monitoring and reviewing every comment can be challenging due to the rapid accumulation of data and insufficient resources to hire full-time moderators.

Advanced algorithms are necessary to handle the tremendous quantities of textual data produced. Natural Language Processing (NLP), a branch of Artificial Intelligence, enables machines to read, comprehend, learn, and extract meaning from human language. NLP is currently used for various tasks such as sentiment analysis, spam detection in emails, identifying fake news, text translation, virtual assistants, chatbots, etc. Recent years have

seen significant progress in this field, with deep neural networks (DNN) (Badjatiya et al. 2017; Aken et al. 2018; Aluru et al. 2020) like recurrent neural networks (RNN) (Yin et al. 2017) and, more recently, Transformers, achieving state-of-the-art results.

Our work aims to improve the efficiency and accuracy of automated content moderation systems by exploring the effectiveness of various NLP and machine learning techniques in classifying, transforming, and representing text data.

Machine learning is at the core of our approach, which involves training computer algorithms to make predictions or decisions based on data. During the modeling process, we employ statistical techniques to unearth patterns and relationships within the data. This learning process can either be supervised, where the model is trained on datasets with known outcomes, thereby learning to predict similar outcomes in new data, or unsupervised, where the model discerns patterns and structures in data without any pre-defined labels or outcomes. Once adequately trained, these models can make well-informed predictions on new and unseen data.

These techniques promise to provide scalable solutions to manage the ever-increasing volume of digital content, replacing labor-intensive manual moderation with more sophisticated automated methods. Our study aims to enhance the moderation process and streamline it, resulting in a more efficient and effective content moderation system.

1.2 The structure of the thesis

1. The first chapter introduces the background of inappropriate content classification, the related research.
2. The second part provides an overview of the dataset and a detailed explanation of the text preprocessing.
3. In the third chapter, the various machine learning and deep learning techniques that were employed for classification are described.
4. The fourth chapter demonstrates the environment and process of implementation and analyses the results.
5. The fifth section evaluates our top-performing model using data collected from Twitter.
6. The sixth part describes the deployment process of our model.
7. The seventh part concludes the whole process.
8. The eighth part describes future work and limitations encountered during implementation.

2. Data and Preprocessing

2.1 Dataset overview

When it comes to data science and machine learning, training, validation, and testing data play a crucial role in developing and evaluating models.

Training data refers to a portion of the original dataset that is used to train or fit a machine-learning model. Typically, it constitutes around 60% to 80% of the dataset. During training, the machine learning model learns to identify patterns and relationships between the input features and the target variable. The primary objective of using training data is to create a model that can accurately predict or make decisions based on these learned relationships.

During the process of tuning a model's hyperparameters, validation data is used to provide an unbiased evaluation of the model, which was fitted on the training dataset. This subset is crucial for preventing overfitting and is typically smaller than the training set but larger than the test set. It is used to fine-tune the model's parameters and indicate its performance during training. Validation data is not used for training the model but for making decisions about which models or model configurations are the best.

On the other hand, testing data is used to evaluate the model's performance. This subset is distinct from the training data and is not used during the training phase. The main purpose of testing data is to assess how well the model performs on new, unseen data. It involves evaluating various performance metrics like accuracy, precision, recall, and F1 score, depending on the problem type.

We are utilizing the Jigsaw Multilingual Toxic Comment dataset, which was created by the Conversation AI team. This dataset was used for a Kaggle competition in July 2020. The data were collected from two different sources: Civil Comments and Wikipedia. These sources contained page comment messages from 63 million users and articles that were manually annotated via crowdsourcing between 2004 and 2015.

The dataset has been divided into three sections:

Training dataset

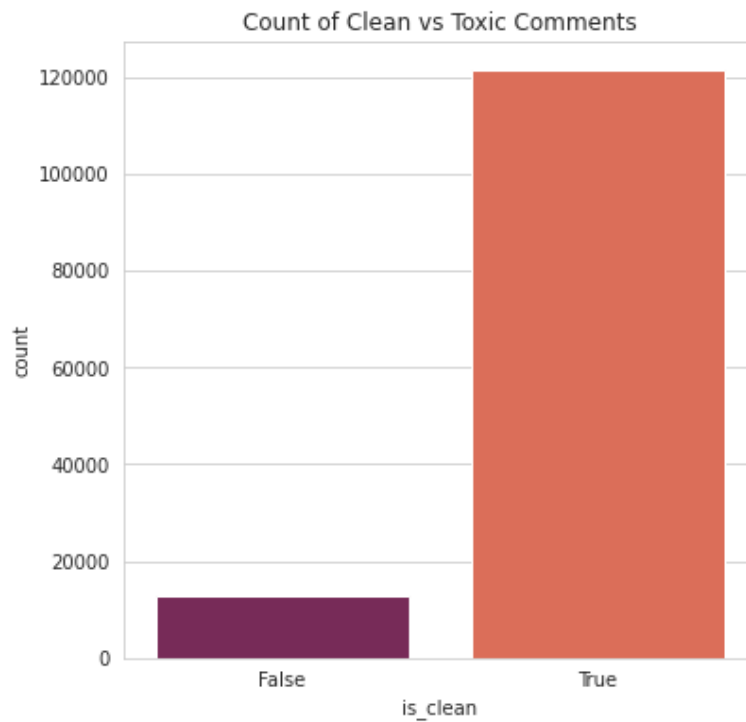


Figure 2.1. The count of clean vs toxic comments in the training dataset

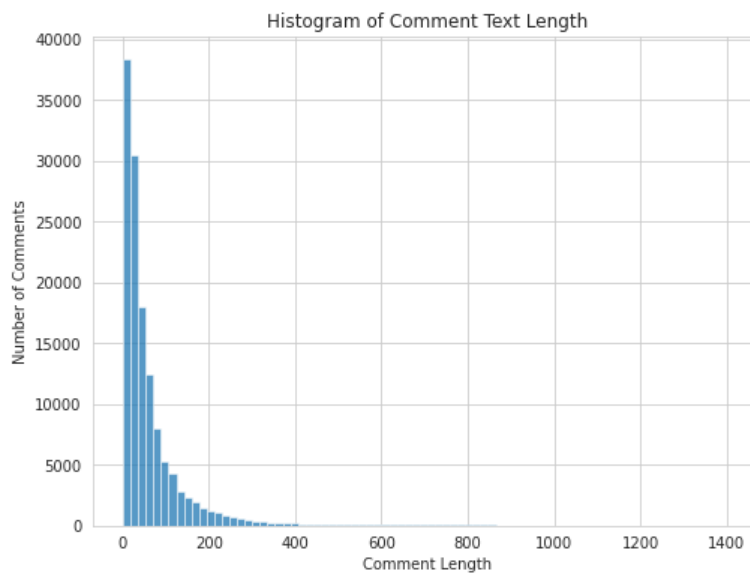


Figure 2.2. Histogram of Comment Text Length

The training set contains 134 129 comments, all in English. There are 10.58 % toxic comments in the training data. The training dataset represents 60% of all comments.

Validation dataset

The validation set contains 44710 comments. There are 10.58 % toxic comments. The validation dataset represents 20% of all data.

Test dataset

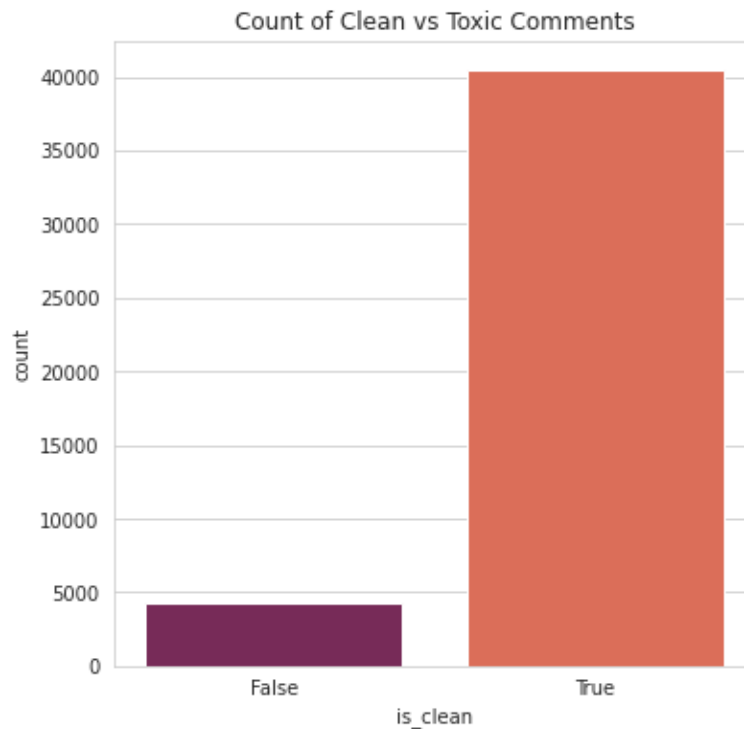


Figure 2.3. The count of clean vs toxic comments in the test dataset

The test set contains 44710 comments. There are 10.58 % toxic comments. The test dataset represents 20% of all the data.

2.2 Text pre-processing

Text preprocessing is a crucial step in the text classification workflow, providing numerous advantages. This step involves converting raw text data into a more comprehensible form, enabling the algorithms to process input data of better quality. Consequently, this leads to improved accuracy in the output results.

Preprocessing plays an important role in ensuring consistency in data processing. Removing noise and irrelevant details creates uniformity and clarity in the text, which is essential for accurate classification.

Converting unstructured text into structured data not only improves the accuracy of classification but also increases computational efficiency. This is because it reduces data dimensionality, resulting in faster processing times and lower resource usage.

Text preprocessing sets the stage for text classification models to perform optimally. Without it, the models may be less accurate, slower, and more complex to train and run.

2.2.1 Remove Punctuation

Punctuation can provide grammatical context to a sentence that supports our understanding, but punctuation is another character for the machine.

2.2.2 Stopword removal

Stop words are common words in any language that often occur but carry much less meaningful information about the phrase's meaning. Here are examples of some common stop words: a, an, and, or, of, on.

2.2.3 Stemming

Stemming is a process in which words are reduced to their root meaning.

When searching for a certain keyword, looking for word variations is helpful. For example, searching for "boat" may also show results for "boats" and "boating".

Porter Stemmer Porter's Algorithm, developed by Martin Porter in 1980, is one of the most common and effective stemming tools.

Snowball Stemmer It is essential to clarify that the term "Snowball" might be misleading. "Snowball" is the name of a stemming language developed by Martin Porter. The algorithm is also known as the "English Stemmer" or "Porter2 Stemmer". This algorithm offers a slight improvement in terms of its logic and speed as compared to the original Porter stemmer.

2.2.4 Lemmatization

Lemmatization and stemming are both techniques used to reduce words to their root form by removing inflections. However, while stemming simply chops off the ends of words, lemmatization takes a more sophisticated approach by applying a set of rules to transform words into their base or dictionary form. This results in more accurate word mapping. For example, stemming might convert "better" to "bett", while lemmatization would correctly map it to "good".

2.2.5 Bag of Words

The Bag of Words (BoW) algorithm is used to count the frequency of words in a document. This helps in assessing the similarity between different documents. BoW is widely used for various tasks such as search, document classification, and topic modeling. It is also popular for preparing text for input in a deep-learning network.

2.2.6 Word embeddings

Word embedding techniques are used to represent each word using real-valued vectors with a fixed vector length. These vectors are typically learned using deep learning methods. Using word embeddings allows for capturing word meanings, as words with similar meanings are represented using similar vector representations. Some frequently used word embedding methods include TF-IDF, Word2Vec, and FastText.

Term Frequency-Inverse Document Frequency or TF-IDF

TF-IDF is an acronym for “Term Frequency — Inverse Document Frequency”. It is a technique used to evaluate the significance of words in a collection of documents by assigning a score to each word.

Terminology

t — term (word)

d — document (set of words)

N — count of corpus

corpus — the total document set

TF (Term Frequency)

$$TF(t) = \frac{\text{Number of times term } t \text{ appears in a document}}{\text{Total number of terms in the document}}$$

Term Frequency is the ratio of the number of target terms in the document to the total number of terms.

IDF (Inverse Document Frequency)

We use Inverse Document Frequency (IDF) to calculate a term’s importance. This is because when we calculate Term Frequency (TF), all terms are seen as equally important, even though certain terms, such as “is”, “of”, and “that”, may appear frequently but have little importance. To address this issue, we need to reduce the weight of frequent terms and increase the weight of rare ones.

We can achieve this by using IDF, which is calculated as follows:

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

TF-IDF

TF-IDF is an information retrieval technique that weighs a term's frequency (TF) and its inverse document frequency (IDF) (Aji, Abdi, and Rakhmadi 2021; *tf-idf - Wikipedia* 2023). Each word has its respective TF and IDF score. The product of the TF and IDF scores of a word is called the TFIDF weight of that word. The higher the TFIDF score (weight), the rarer the word and vice versa. TF-IDF is a technique used in information retrieval that considers a term's frequency (TF) and its inverse document frequency (IDF). Each word is assigned a corresponding TF and IDF score. The product of these two scores is called the TFIDF weight of the word. If a word has a higher TFIDF score (weight), it is considered to be rarer and vice versa. This technique is used to determine the relevance of a word in a document.

$$TF - IDF = TermFrequency(TF) * InverseDocumentFrequency(IDF)$$

Word2Vec

In 2013, a technique for natural language processing was published named Word2vec (Mikolov et al. 2013). The algorithm employs a neural network model to learn how words are associated with one another based on a large corpus of text. Once the model is trained, it can suggest similar words or additional words that can be used in a sentence. Word2vec uses a unique set of numbers, called a vector, to represent each word to determine how semantically similar it is to other words. It is a popular method for creating word embeddings known for its rapid computation and open-source availability.

FastText

FastText is an extension of Word2Vec, which was proposed by Facebook in 2016. One of the major improvements of FastText over Word2Vec is that instead of feeding individual words into the neural network, FastText divides words into several n-grams (sub-words). As a result, it can generate word representations for words not present in the training data. FastText vectors are trained on Wikipedia and Crawl and available in 157 languages. They are super-fast to train.

GloVe

GloVe is an unsupervised learning algorithm for obtaining vector representations for words (Pennington, Socher, and Manning 2014). Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space (Pennington 2021).

2.2.7 Tokenization

Tokenizing is splitting some string or sentence into a list of words to structure our non-structured text. That is because the machine cannot read directly from the words.

2.3 Imbalanced Data

Addressing imbalanced datasets is a common challenge in the field of machine learning. In machine learning, imbalanced data is a term used to describe a situation where the classes or categories in a classification problem are not equally represented. This means that one class, known as the majority class, is much more prevalent than the other class or classes, known as the minority class or classes. Such an imbalance's impact can be significant, leading to biased predictions, inaccurate decision-making, and suboptimal results. This chapter will explore imbalanced data, its consequences, and practical techniques to address the problem. Understanding the impact of imbalanced data and implementing appropriate measures to ensure accurate predictions and decision-making is crucial.

Several strategies and techniques can help mitigate the challenges associated with imbalanced datasets:

1. **Resampling** is a common technique used to modify datasets, which can involve oversampling the minority class, undersampling the majority class, or a combination of both.
2. **Oversampling**
 - **Random Oversampling:** Duplicates minority class samples to balance the class distribution.
 - **SMOTE (Synthetic Minority Over-sampling Technique):** Generates synthetic examples based on the existing minority class samples.
3. **Undersampling**
 - **Random Undersampling:** Removes random samples from the majority class.
 - **Tomek Links:** Removes the majority class samples close to the minority class samples.
4. **Algorithmic Approaches.** Some machine learning algorithms can handle imbalanced datasets better than others. Consider using algorithms less sensitive to class imbalance, such as Random Forest or Gradient Boosting.

Figure 2.1 shows that only 10.58% of comments are toxic in the training dataset. We intend to employ the SMOTE technique and algorithmic approaches like Random Forest or Gradient Boosting to address this data imbalance in our work.

3. Methods

3.1 Basic classification techniques

Classification is a type of supervised learning in machine learning. The primary goal of classification is to use past observations to predict the categorical class labels of new instances. To achieve this, the algorithm is trained on a dataset containing instances or examples already labeled with the correct output. After the algorithm is trained, it can be used to classify new examples. This process involves analyzing the input data and assigning a class label to it based on what the algorithm learned from the training dataset.

In text classification tasks, such as detecting toxic comments, the goal is to categorize text snippets (like comments, tweets, or posts) into classes - typically binary classes such as "toxic" or "non-toxic." This involves teaching a model to recognize patterns, features, or contexts that are indicative of each category.

3.1.1 Naïve Bayes Classifier

The Naïve Bayes algorithm is a type of supervised learning algorithm. It uses Bayes' Theorem to solve classification problems. This algorithm is simple yet effective and is commonly used for building fast machine-learning models that can make quick predictions. Its primary use case is in text classification.

Bayes' Theorem calculates the probability of an event based on the probability of another event occurring. Mathematically, it is represented by an equation.

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}, \quad (3.1)$$

where A and B are events and $P(B) \neq 0$.

- $P(A | B)$ is a conditional probability: the probability of event A occurring given that B is true. It is also called the posterior probability of A given B (*Bayes' theorem - Wikipedia 2023*).
- $P(B | A)$ is also a conditional probability: the probability of event B occurring given that A is true. It can also be interpreted as the likelihood of A given a fixed B because $P(B | A) = L(A | B)$ (*Bayes' theorem - Wikipedia 2023*).
- $P(A)$ and $P(B)$ are the probabilities of observing A and B, respectively, without any given conditions; they are known as the marginal probability or prior probability (*Bayes' theorem - Wikipedia 2023*).

-
- A and B must be different events.

Advantages of Naïve Bayes Classifier:

- Naive Bayes is a simple and efficient machine learning algorithm used for predicting a class of datasets.
- This algorithm can be applied to binary and multi-class classification problems.
- Compared to other algorithms, it has been observed to perform better in multi-class predictions.
- Naive Bayes is a popular solution for text classification problems.

Disadvantages of Naïve Bayes Classifier:

- Naive Bayes assumes independence between features, limiting its ability to learn feature relationships.

3.1.2 Random forest

Random Forest is a machine learning algorithm used for classification and regression problems. It is based on the concept of decision trees, which is a flowchart-like tree structure that an individual uses to make decisions based on the data attributes' conditions. Decision trees are widely used in various machine-learning tasks, including classification and regression.

The Random Forest algorithm combines several decision trees to make predictions, where each tree is trained on a different subset of the data. The algorithm builds each decision tree in the forest using a random subset of the dataset's features, which helps reduce overfitting and improve the model's accuracy. To make a prediction for a new input, the algorithm passes the input through each decision tree in the forest and takes the average of the outputs as the final prediction. Combining the predictions of multiple decision trees helps improve the overall accuracy and stability of the model while reducing the risk of overfitting.

Random Forest is a popular machine learning algorithm due to its simplicity, high accuracy, and ability to handle large datasets with many features. It is commonly used in applications such as image classification, text classification, and predictive modeling.

3.1.3 Support Vector Machine

Support Vector Machines (SVMs) are supervised learning algorithms used for classification, regression, and detecting outliers (*Support Vector Machine (SVM) Algorithm - Javatpoint 2022*). The main concept behind SVM is to identify the best hyperplane (a line in two-dimensional space) that can accurately separate a dataset into classes.

Here are the key concepts of SVM:

- **Hyperplane:** This decision boundary separates feature space classes. In a two-dimensional space, it is a line, but in higher dimensions, it is a plane or a hyper-surface.
 - **Support Vectors:** These critical data points closest to the hyperplane influence its position and orientation, making them essential in the training set.
- Margin:** This is the gap between the two classes separated by a hyperplane. SVM aims to maximize this margin to improve the model's generalization abilities.

SVMs can be used in both linear and non-linear classification:

Linear SVM: Linear SVM is a classification algorithm that divides datasets into two classes using a single straight line. Such datasets are called linearly separable data. When the data is linearly separable, the classifier used is called a Linear SVM classifier.

Non-linear SVM: Non-linear SVM classifiers are used for datasets that cannot be linearly separated. This means a non-linear SVM classifier is utilized if a dataset cannot be classified using a straight line.

3.1.4 Logistic Regression

Logistic regression is a machine learning classification algorithm that predicts the probability of an outcome with discrete and categorical values. Instead of providing exact values like 0 and 1, it provides probabilistic values between 0 and 1. The curve from the logistic function indicates the likelihood of something happening.

Logistic Regression Assumptions:

- In binary logistic regression, the dependent variable must have only two possible outcomes, which are typically represented as 0 or 1.
- For binary regression, factor level 1 of the dependent variable should indicate the desired outcome.
- Only the meaningful variables should be included.
- The independent variables must be uncorrelated to avoid multicollinearity.
- The independent variables are linearly related to the log odds.
- Logistic regression requires quite large sample sizes.
- Logistic regression is not able to handle a large number of categorical features/variables.
- Logistic regression doesn't require high computation power, easy to implement, easily interpretable, used widely by data analyst and scientist

Types of Logistic Regression

- **Binary Logistic Regression:** The categorical response is a binary outcome with only two possibilities. For instance, when classifying emails, the emails are either considered Spam or Not Spam.
- **Multinomial Logistic Regression:** Multinomial logistic regression is used when there are three or more categories without ordering. For example, predicting food preferences such as veg, non-veg, and vegan.
- **Ordinal Logistic Regression:** Ordinal Logistic Regression applies to three or more categories that are ordered, such as a movie rating scale from 1 to 5.

3.2 Gradient tree boosting

Gradient boosting is a powerful algorithm used to build predictive models for both regression and classification tasks. It is based on decision trees and works by combining multiple weaker models to create a stronger, more accurate one. Unlike random forests, which create decision trees for each sample, gradient boosting creates trees one after the other. Each new tree is built to improve upon the results of the previous one without altering it. This makes gradient boosting a highly effective technique for machine learning.

3.2.1 XGBoost

XGBoost is a high-performance implementation of gradient-boosted decision trees, available as an open-source library. The library was developed by scholars at the University of Washington. It combines an underlying C++ codebase with a Python interface, making it a powerful and easy-to-implement package.

Advantages of XGBoost (*XGBoost ML Model in Python - Javatpoint 2023*)

- **Execution and Speed:** The software was initially developed using the C++ programming language.
- **Center calculation is parallelizable:** Calculating centers in XGBoost is parallelizable. It can utilize the power of multi-core CPUs as well as GPUs and computer networks, making it possible to train on large datasets.
- **Reliably outflanks other technique calculations:** It consistently outperforms other AI algorithms, as demonstrated by superior results on multiple benchmark datasets.
- **Wide assortment of tuning boundaries:** XGBoost has limitations for scikit-learn API, lacks features like regularization, tree boundaries, cross-validation, etc.

3.3 Convolutional Neural Network

Convolutional Neural Networks (CNN) are a type of network architecture used in deep learning for computer vision. Although they were initially developed for image recognition, CNNs have shown promising results when applied to various Natural Language Processing (NLP) tasks as well. In fact, many of the models submitted by ImageNet teams since 2014 have been based on CNNs. As word embedding algorithms like Word2vec and GloVe gained popularity, they became one of the most common text processing methods. Therefore, utilizing CNNs to extract features from word embedding matrices to handle NLP tasks became inevitable and has shown promising results.

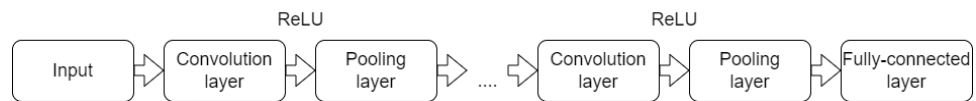


Figure 3.1. Basic structure of CNN

The basic structure of a CNN is shown in Figure 3.1. Convolutional neural networks (CNNs) consist of an input layer, multiple hidden layers, and an output layer. The hidden layers, the core component of a CNN architecture, consist of convolutional layers, pooling layers, and a fully connected layer that exports the output. The convolutional layer is the main building block of a CNN. It abstracts the input with a specific shape into a feature map using a set of learnable filters (or kernels). The pooling layer progressively reduces the spatial size of the feature map generated by the previous convolutional layer and identifies important features. The dense layer (fully connected neural network) has each neuron in a layer receiving information from all neurons in the previous layer, making them densely connected. This means that all neurons in a layer are connected to the next parts, and the input image is classified through this layer. Activation functions like Relu, Softmax, and Sigmoid are used to define the output of the dense layer. The choice of activation function depends on the data type. For instance, Softmax is used for multiclass classification, while Sigmoid is used for binary classification.

3.4 Recurrent Neural Network

Recurrent neural networks (RNNs) are a type of neural network where the outputs from previous steps are passed as input to the current step.

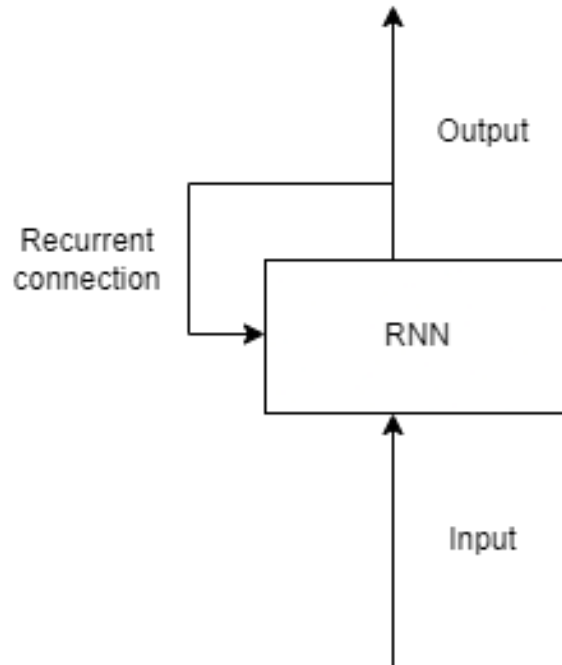


Figure 3.2. A recurrent network: a network with a loop

As shown in Figure 3.2, the network uses both the output of the previous step and the internal state from the previous step as inputs for the current step. The network has a hidden state and loops, which enable it to store past information in the hidden state and operate on sequences. Additionally, the looping structure helps to enhance the network's performance on current and future inputs.

Unlike feed-forward neural networks, RNNs can handle sequential data and memorize previous inputs due to their internal memory. A feed-forward neural network only allows information to flow in the forward direction, from the input nodes through the hidden layers and to the output nodes. Due to this reason, RNN is a perfect solution for Natural Language Processing (NLP) tasks since they can take in text as full sequences of words, starting from a single sentence up to an entire document.

When working with Recurrent Neural Networks (RNNs), a modified form of backpropagation called Back Propagation Through Time (BPTT) is used. RNNs are designed to process sequence data, which means they can be either correct or incorrect at every point in the sequence. The model's weights must be adjusted each time to learn from sequence data effectively. The model starts at the most recent output and works backward to calculate the loss and update the weights at each time step, essentially going "back in time" to learn.

Updating every single weight at every single time step makes BPTT computationally more expensive than traditional backpropagation.

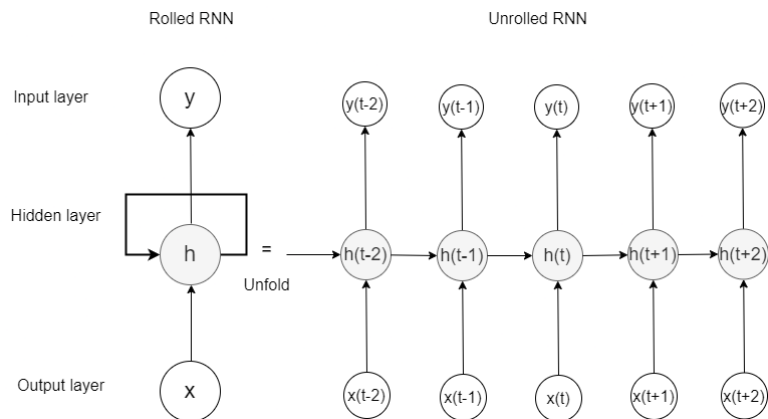


Figure 3.3. Working of Recurrent Neural Network

Recurrent Neural Networks pass information to the middle hidden layer using a loop. The input layer 'x' receives input and processes it before passing it onto the middle layer.

The middle layer 'h' can have multiple hidden layers, each with its activation functions, weights, and biases. If a neural network does not have memory, meaning the various parameters of different hidden layers are not affected by the previous layer, we can use an RNN.

The RNN standardizes the activation functions, weights, and biases of different hidden layers so that each layer has the same parameters. Instead of creating multiple hidden layers, the RNN creates one and loops over it as many times as needed to process the input.

3.4.1 Long Short Term Memory

LSTM (long short-term memory) is a type of recurrent neural network. LSTM is widely used for text classification problems, as it helps maintain the order of words in the text that can preserve the semantic meaning of the text. LSTM is capable of learning long-term dependencies by remembering information for long periods.

3.4.2 Gated Recurrent Network

GRUs are a type of recurrent neural network similar to LSTMs. The key difference between GRUs and LSTMs is that they have a simpler architecture, are faster to train, and are preferred for smaller datasets.

3.5 Transformer

The Transformer architecture was first introduced in the "Attention is All You Need" paper by Google researchers in 2017 (Vaswani et al. 2017). It utilizes an attention mechanism that takes in the entire text input simultaneously to establish contextual relationships between words. Like LSTM, the Transformer employs an encoder and a decoder to convert one sequence into another. However, it stands apart from previously described sequence-to-sequence models because it implements an encoder-decoder structure without utilizing recurrence or convolutions.

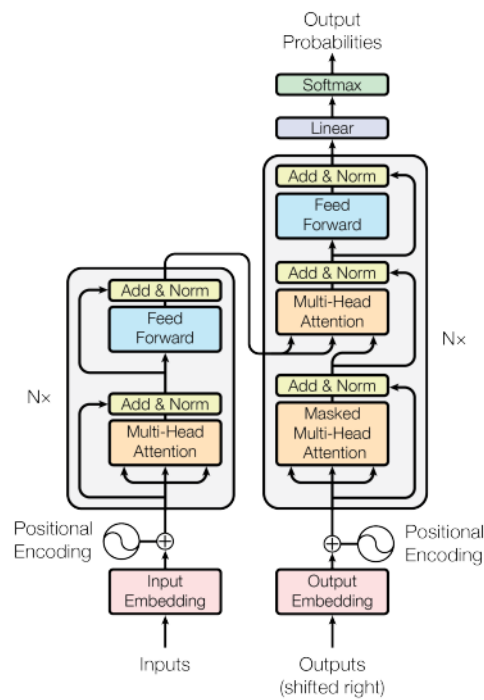


Figure 3.4. The encoder-decoder structure [Image taken from (Vaswani et al. 2017)]

In Figure 3.4, we can observe that the Transformer is comprised of two parts: the encoder and the decoder. The encoder, which is located in the left half of the Transformer architecture, is responsible for mapping an input sequence of symbol representations to a sequence of representations. The encoder is made up of a stack of $N=6$ identical layers, each of which comprises two sub-layers. The first sub-layer contains a multi-head self-attention mechanism, while the second sub-layer is a fully connected feed-forward network. Each of the two sub-layers has a residual connection and a normalization layer. The Transformer architecture is unable to capture information about the relative positions of words in the sequence. To address this, positional encodings are used to inject positional information into the input embeddings. The positional encoding vector has the same dimension as the input embedding.

The decoder is situated on the right half of the architecture. Its function is to receive the output of the encoder along with the decoder output from the previous time step to create

an output sequence. The decoder is made up of six identical layers, each of which contains three sublayers.

The first sublayer takes the previous output of the decoder stack, adds positional information, and applies multi-head self-attention to it. The second layer also applies multi-head self-attention, similar to the first sublayer of the encoder. Finally, the third layer implements a fully connected feed-forward network, like the second sublayer of the encoder.

3.5.1 BERT

BERT(Bidirectional Encoder Representations from Transformers) is a Transformer architecture published by researchers at Google AI in the paper (Devlin et al. 2018). It is an attention mechanism capable of learning contextual relations between words and even sub-words in a text. The Transformer model consists of two mechanisms, namely the encoder and decoder. The encoder reads the text input, while the decoder provides a prediction for the task. In the case of BERT, the focus is on generating a language model, which means that only the encoder mechanism is required. A paper by Google (Vaswani et al. 2017) provides detailed information on how the Transformer works.

When processing text, Transformer encoders differ from directional models in that they process the entire sequence of words at once rather than sequentially (from left to right or right to left). This means that Transformer encoders are bidirectional, as they can learn the context of a word based on its surroundings both to the left and the right. This characteristic is crucial for understanding language meaning.

BERT was pre-trained using the BooksCorpus dataset and English Wikipedia. With just adding a single output layer, the pre-trained BERT model can be fine-tuned for tasks such as question answering, text classification, NER, and language inference without requiring significant modifications to the task-specific architecture.

BERT was trained on two tasks simultaneously:

- Masked language modeling (MLM) — 15% of the tokens were masked and were trained to predict the masked word
- Next Sentence Prediction(NSP) — Given two sentences A and B, predict whether B follows A

There are two different BERT models:

- BERT base is a BERT model with 12 layers of Transformer encoder, 12 attention heads, 768 hidden size, and 110M parameters.
- BERT large is a BERT model with 24 transformer encoder layers, 16 attention heads, 1024 hidden size, and 340M parameters.

In the case of BERT, we can omit preprocessing because of it was trained on the same dirty data and go straight to tokenization and training. Also, linear models often give incorrect results because they do not consider the context of words. Understanding the context is the main advantage of transformers.

3.5.2 DistilBert

DistilBERT is a machine learning model that is an approximate version of BERT (Sanh et al. 2019). It retains 97% of BERT's performance but uses only half the number of parameters, making it 40% smaller than BERT-base (which has 110M parameters). Additionally, DistilBERT is 60% faster than BERT-base. While it is not the State-of-the-art (SOTA) model, it is still competitive. DistilBERT uses a similar architecture to BERT but with fewer encoder blocks (6 blocks compared to BERT base's 12 blocks and BERT large's 24 blocks). Unlike BERT, DistilBERT does not use token-type embeddings or pooling functionalities. Unlike BERT, which was trained using masked language modeling and next-sentence prediction, it is only pre-trained using masked language modeling.

3.5.3 XLM-RoBERTa

In November 2019, the Facebook AI team introduced XLM-RoBERTa as a successor to their original XLM-100 model (Conneau et al. 2019). Both models are based on transformers and use the Masked Language Model objective to process text from 100 different languages. However, XLM-RoBERTa offers a major improvement over the original model in the form of a significantly larger amount of training data. This new model has been trained on over 2.5 terabytes of diverse text data from 100 different languages, making it one of the largest multilingual language models available.

4. Experiments

4.0.1 Experimental Setup

Machine learning models run on the author's PC. The deep learning model training is executed using Google Colab Pro's High-RAM environment using a single NVIDIA P100 GPU. Colab is a Jupyter notebook service that provides free access to computing resources such as GPUs and TPUs. It requires no setup to use. The scikit-learn library was used for the classic machine learning algorithms.

Tensorflow and Keras

TensorFlow is a software library for machine learning and artificial intelligence that is available as an open source. Keras is another open-source software library that provides a Python interface for artificial neural networks and can also act as an interface for the TensorFlow library. The code was written in Python using Keras and Tensorflow as backend.

4.0.2 Grid Search for model tuning

Optimizing machine learning models is a crucial step to ensure optimal model performance. Hyperparameter tuning is an essential part of this process. Hyperparameters are parameters that cannot be learned from data but must be set before training begins. Grid search is a popular technique for hyperparameter optimization. It involves searching through all possible combinations of hyperparameter values to find the best combination for optimal model performance. Grid search creates a grid of possible hyperparameter values, ensuring that a wide range of combinations are explored. However, it is important to note that grid search can be computationally expensive, especially for models with many hyperparameters or a wide range of values.

4.0.3 Model evaluation

The process of model evaluation is a critical step in machine learning. It involves measuring the effectiveness of a trained model by assessing its performance using different metrics and techniques. The goal is to determine how well the model can predict new and unseen data. This evaluation is necessary to confirm the model's reliability, robustness, and suitability for deployment in practical applications.

Confusion matrix

A Confusion Matrix is a performance evaluation tool for machine learning classification (Roepke 2022). It summarizes the prediction results on a classification problem by pre-

sentencing count values for correct and incorrect predictions broken down by each class. Normalizing the confusion matrix involves scaling it to contain values between 0 and 1, making it easier for data scientists to interpret the predictions of labels visually.

		Actual values	
		Positive (1)	Negative (0)
Predictive values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Figure 4.1. Confusion Matrix

True Positive (TP) refers to the number of positive cases properly classified as positive

True Negative (TN) refers to the number of negative cases properly classified as negative

False Positive (FP Type 1 Error) refers to the number of negative cases improperly classified as positive

False Negative (FN Type 2 Error) refers to the number of positive cases improperly classified as negative

Evaluation Metrics

Machine learning evaluation metrics play a crucial role in determining the performance and effectiveness of algorithms. These metrics give a quantitative measure of the model's performance, essential for selecting, optimizing, and validating the model. The selection of the appropriate metric depends on the problem being addressed and the nature of the dataset.

For classification tasks, common metrics include:

1. **Accuracy:** Accuracy measures how often a machine learning model predicts correctly by dividing the number of correct predictions by the total number of predictions. Useful for balanced datasets.

$$Accuracy = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Observations}}$$

2. **Precision:** Evaluates the percentage of true positives out of all positive cases. Useful for prioritizing positive identification over overall accuracy.

$$Precision = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

3. **Recall:** Recall is a measure of how accurately a machine learning model identifies positive instances (true positives) out of all the actual positive observations in the dataset. It is calculated by dividing the number of true positives by the total number of positive instances.

$$Recall = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

4. **F1 Score:** When it comes to sequence labeling tasks like entity extraction and retrieval-based question answering, combining precision and recall can provide a single metric that measures completeness and exactness. This metric is particularly helpful when dealing with imbalanced classes or when false positives and false negatives have varying costs. The F1 score, which is the harmonic mean of precision and recall, offers a balanced view of these two metrics.

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

5. **ROC-AUC Score:** A Receiver in Action Plotting the True Positive (TP) versus the False Positive (FP) at various threshold values produces a characteristic curve, also known as a ROC curve. The ROC curve is created by graphing the True Positive's cumulative distribution function on the y-axis against the False Positive's cumulative distribution function on the x-axis. It is used to compare classification algorithms and is the most preferred metric when the dataset is imbalanced.

Sensitivity is a metric that measures the accuracy of correctly identifying the positive class. It tells us the proportion of true positives that were correctly classified.

$$TPR = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

False Positive Rate (FPR) indicates the proportion of negative class incorrectly classified by the classifier.

$$FPR = \frac{\text{False Positive}}{\text{False Positive} + \text{True Negative}}$$

A desirable classification should have a higher TPR and lower FNR to identify positive cases correctly.

Classification report

When it comes to evaluating the effectiveness of classification models in machine learning, we typically rely on three common metrics: precision, recall, and F1 score. These metrics help us to measure how accurately a classification model predicts the response variable. The good news is that we can easily calculate these metrics in Python by leveraging the `classification_report()` function provided by the `sklearn` library.

4.0.4 Performance evaluation on the test set.

In this section, we focused on evaluating the performance of various NLP models to detect toxic comments. The Jigsaw Multilingual Toxic Comment dataset was used for this purpose, known for its diverse range of user comments in multiple languages. We experimented with several word embedding techniques to represent the text data, including Word2Vec, Term Frequency-Inverse Document Frequency (TF-IDF), GloVe, FastText, and specific embedding layers designed for deep learning models. For the classification task, a total of twelve models were employed. This included five machine learning models: Naive Bayes, Random Forest, Logistic Regression, LinearSVM, and XGBoost; along with seven deep learning models: Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Long-Short Term Memory (LSTM), Gated Recurrent Units (GRU), Bidirectional Encoder Representations from Transformers (BERT), DistilBert (a distilled version of BERT), and XLM-RoBERTa (XLM-R).

All models were trained and tested on the same dataset to ensure consistency in performance evaluation. This approach allowed for a direct comparison of each model's effectiveness in identifying toxic comments.

Hyperparameter tuning played a crucial role in the model training process. For machine learning models, a combination of grid search and random search techniques were utilized to determine the optimal set of parameters. In the case of deep learning models, a combination of manual tuning is employed. This ensured that each model was performing at its highest capacity.

Evaluating the models based on several metrics, such as accuracy, precision, recall, and F1 score, provided a comprehensive understanding of each model's performance. These metrics considered more than just the overall correctness (accuracy) and also assessed the model's ability to correctly identify positive instances (precision and recall), as well as the balance between precision and recall (F1 score).

A summary of the performance of each model on the test set can be found in Table 4.1. This table includes the metrics mentioned above for each model, clearly and concisely comparing their performance.

Table 4.1. Accuracy of machine learning models

Model	Accuracy	Precision	Recall	F1-Score
XLM-RoBERTa	0.9608	0.8959	0.8691	0.8819
DistiBert	0.9579	0.8764	0.8801	0.8782
Bert Base	0.9539	0.8749	0.8676	0.8484
GRU + GloVe Embedding	0.9539	0.8967	0.8169	0.8512
CNN + GloVe Embedding	0.9517	0.8905	0.8081	0.8431
LSTM + GloVe Embedding	0.9514	0.8842	0.8139	0.8446
CNN + FastText Embedding	0.9494	0.8594	0.8379	0.8482
GRU + FastText Embedding	0.9472	0.8783	0.7891	0.8261
LSTM + FastText Embedding	0.9465	0.8795	0.7822	0.8218
XGBoost + TF-IDF	0.9449	0.8321	0.8638	0.8469
Random forest + TF-IDF	0.9436	0.8611	0.7843	0.8168
Naive Bayes + TF-IDF	0.9388	0.8237	0.8168	0.8202
RNN + FastText Embedding	0.9349	0.8687	0.7064	0.7591
LinearSVM + TF-IDF	0.9119	0.7519	0.8748	0.7955
RNN + GloVe Embedding	0.9340	0.8545	0.7127	0.7613
XGBoost + Word2Vec Embedding	0.8417	0.6119	0.6639	0.6289
Logistic regression + TF-IDF	0.8391	0.6614	0.8199	0.6941
XGBoost + GloVe Embedding	0.8360	0.6020	0.6505	0.6174

Among all the other models, the XLM-RoBERTa model achieved the highest accuracy of 96.08% and an F1 score of 88.19%. One of the reasons for this could be that XLM-RoBERTa has been pre-trained on a large amount of text data, enabling it to learn patterns and relationships between words that can be used to identify toxic comments. Deep learning has significant advantages over traditional machine learning methods in text classification. Traditional machine learning methods need to be improved in their ability to understand the semantic and contextual meaning of words, which is critical in most NLP tasks.

4.0.5 Logistic regression + TF-IDF

This section presents the results of applying Logistic Regression in combination with TF-IDF to classify toxic and non-toxic comments.

The hyperparameter tuning process through `GridSearchCV` resulted in the following optimal hyperparameters:

- **C (0.01):** This parameter is the inverse of the regularization strength, where smaller values indicate stronger regularization. By setting C to 0.01, the model applies strong regularization, which can prevent overfitting by penalizing larger values in the decision function.
- **Penalty (l1):** This penalization norm is known as Lasso regression and can reduce coefficients to zero for feature selection and resulting in a model with fewer predictors.
- **Solver (liblinear):** This is the algorithm to use in the optimization problem. *Liblinear* is a good choice for small datasets and binary classification problems. It is one of the few solvers that supports $l1$ penalty.

Table 4.2. Classification report of Logistic regression

	precision	recall	f1-score	support
0	0.9753	0.8429	0.9043	40159
1	0.3474	0.7969	0.4839	4214
accuracy			0.8391	44373
macro avg	0.6614	0.8199	0.6941	44373
weighted avg	0.9157	0.8386	0.8644	44373

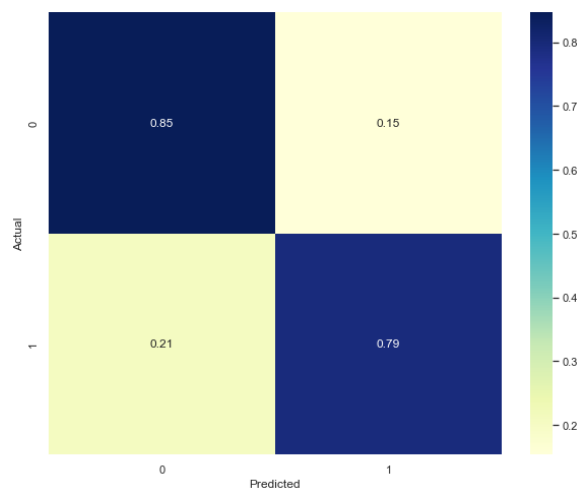


Figure 4.2. Confusion matrix of Logistic regression model

Figure 4.2 displays the confusion matrix, offering an overview of the classification outcomes, while Table 4.2 provides insights through the classification report.

The model has achieved an 83.91% accuracy rate, indicating that it can predict the class labels (toxic or non-toxic) of comments correctly about 83.91% of the time. The precision, recall, and F1-score macro average are 66.14%, 81.99%, and 69.41%, respectively, which suggests that the model has a balanced performance between the two classes. The precision for non-toxic comments (Class 0) is high at 97.53%, indicating that only a small number of non-toxic comments were wrongly classified as toxic. The recall for non-toxic comments (Class 0) is 84.29%, meaning that the model can effectively identify most non-toxic comments. The F1-score for non-toxic comments (Class 0) is 90.43%, demonstrating a good balance. However, the precision for toxic comments (Class 1) is relatively low at 34.74%, indicating that a significant proportion of predicted toxic comments were actually non-toxic. The recall for toxic comments (Class 1) is 79.69%, indicating that the model can identify a large proportion of actual toxic comments. The F1 score for toxic comments (Class 1) is 48.39%, representing a trade-off between precision and recall. The False Positive Rate (FPR) is 15.17%, indicating the proportion of actual non-toxic comments that were wrongly classified as toxic, while the False Negative Rate (FNR) is 20.51%, indicating the proportion of actual toxic comments that were wrongly classified as non-toxic.

Achieving a balance between these two rates is pivotal. While the aim is to minimize false positives and false negatives, the optimal trade-off depends on factors like the platform's nature, user community, and the consequences of each error type. Depending on the circumstances, the priority might be to decrease false positives to prevent mislabeling non-toxic content or to reduce false negatives to ensure prompt action against harmful content.

Here are the advantages and disadvantages of using Logistic Regression with TF-IDF for text classification (Gao and R. Huang 2017; Saif et al. 2018):

Pros:

- **Simplicity and Interpretability:** Logistic regression models have a simple structure, making it easy to interpret their results.
- **Fast Training and Prediction:** Logistic regression is computationally efficient in training and prediction compared to more complex models such as deep learning algorithms.
- **Requires Less Data:** Logistic regression can achieve reasonable results with smaller datasets compared to deep learning models that require large amounts of data to perform well.
- **Probabilistic Output:** Logistic regression predicts the probability of an instance belonging to a particular class, which can help understand a prediction's confidence or set different decision thresholds.
- **Less Prone to Overfitting:** Logistic regression is a simple and effective method that is less likely to overfit, particularly when it is regularized.

-
- **Ease of Implementation:** Many libraries offer support for logistic regression, making it a straightforward process to implement and utilize this statistical method.

Cons:

- **Linearity Assumption:** Logistic regression assumes a linear relationship between input features and the log-odds of the output. This assumption might not always hold true for complex problems like toxic comment detection.
- **Feature Dependency:** Model performance and stability may suffer if features are highly correlated. Careful feature engineering and selection are required.
- **Limited Expressiveness:** Logistic regression's decision boundary may oversimplify complex datasets compared to non-linear classifiers.
- **Sensitive to Outliers:** Logistic regression can be sensitive to outliers, which can skew the decision boundary and negatively impact the model's performance.
- **Binary Classification:** Standard logistic regression is designed for binary classification and may not be as efficient for datasets with multiple toxicity categories.
- **May Not Capture Complex Patterns:** Logistic regression may not capture complex patterns, idiomatic expressions, and context-dependent meanings in text data.

In summary, the Logistic Regression + TF-IDF model effectively identifies non-toxic texts but has room for improvement in precisely classifying toxic texts. The high recall for the toxic class suggests it captures the most toxic instances. However, the lower precision indicates that many non-toxic instances are being misclassified as toxic. Fine-tuning the model, using additional features, or incorporating more sophisticated techniques could improve its precision for the toxic class without compromising its recall. While logistic regression offers simplicity and ease of use, there may be better choices for complex tasks like toxic comment detection, especially when the data has non-linear patterns. However, it can serve as a good baseline, and in some cases, it can perform quite well with the proper feature engineering.

4.0.6 Naive Bayes + TF-IDF

This section presents the results of applying Naive Bayes in combination with TF-IDF.

The optimal hyperparameters obtained from the hyperparameter tuning process identified 'alpha' : 30 as the most appropriate value. The 'alpha' parameter represents the Additive (Laplace/Lidstone) smoothing parameter.

Table 4.3. Classification report of Naive Bayes

	precision	recall	f1-score	support	predicted negative	predicted positive
0	0.9651	0.9675	0.9663	40159	0.9675	0.0325
1	0.6823	0.6661	0.6741	4214	0.3339	0.6661
accuracy			0.9388	44373		
macro avg	0.8237	0.8168	0.8202	44373		
weighted avg	0.9382	0.9388	0.9385	44373		

Figure 4.3 provides insights through the classification report.

The model performed remarkably well, with an overall accuracy of 93.88%. This means that it correctly classified almost 94% of the test data instances. The model demonstrated strong performance for non-toxic texts, with a precision of 96.51% and recall of 96.75%, indicating it can effectively detect non-toxic texts without missing any. However, the precision and recall for toxic texts were 68.23% and 66.61%, respectively. While these metrics are reasonable, there is still room for improvement, especially compared to the performance on non-toxic texts.

The F1 scores for non-toxic and toxic classes were 96.63% and 67.41%, respectively. The model performed exceptionally well for non-toxic texts but only moderately well for toxic texts. The model had 1,307 False Positives (FP), indicating that it mistakenly identified 1,307 non-toxic texts as toxic. This is reflected in a low False Positive Rate (FPR) of 3.25%. However, there were 1,407 False Negatives (FN), meaning 1,407 toxic texts were incorrectly labeled as non-toxic, which gives a relatively higher False Negative Rate (FNR) of 33.39%.

The macro-average F1-score of 82.02% and the weighted average F1-score of 93.85% provide a comprehensive view of the model's performance, giving a balanced score between both classes.

Certainly, here are the pros and cons of using the Naive Bayes classifier with TF-IDF representation for text classification (J. Huang, Lu, and Ling 2003):

Pros:

- **Simplicity:** Naive Bayes is conceptually straightforward, making it easy to implement and understand.

-
- **Efficiency:** Both in terms of computational cost and memory usage, Naive Bayes is efficient, especially when combined with the sparsity of TF-IDF vectors.
 - **Scalability:** The algorithm scales well with the size of the dataset.
 - **Works Well with High Dimensions:** Given that TF-IDF can produce a high-dimensional vector for each document, Naive Bayes handles this well because it treats each feature (word) independently.
 - **Probabilistic Output:** Like logistic regression, Naive Bayes predicts probabilities for classes, which can be insightful for gauging the confidence of a prediction.
 - **Requires Less Data:** Can provide decent results even with smaller training datasets.
 - **Inherent Feature Selection:** TF-IDF inherently ranks features based on their importance (by giving lower scores to less informative words).

Cons:

- **Naivety Assumption:** The assumption that class-independent features (words) are independent is often invalid in natural language data where word context and order can be crucial.
- **Limitations with Sarcasm and Nuance:** Due to its independence assumption, Naive Bayes may struggle to understand context, sarcasm, and nuanced statements.
- **Zero Frequency Problem:** Naive Bayes assigns a zero probability if a word appears in the test data but not in the training data.
- **Data Imbalance Sensitivity:** If one class is dominant, Naive Bayes predictions might be biased towards that class.
- **TF-IDF Limitations:** TF-IDF may miss semantic meanings, unlike word embeddings.
- **Binary Classification Focus:** Standard Naive Bayes is typically used for binary classification but can be extended to multi-class classification.

The Naive Bayes + TF-IDF model performs strongly, especially in identifying non-toxic texts. While the performance is decent for toxic comment detection, there is notable room for improvement, particularly in reducing the number of undetected toxic comments (False Negatives). Given the sensitivity of the task, fine-tuning to improve recall for toxic comments without compromising much on precision would be desirable. The results offer a promising baseline; even better performance might be achieved with proper adjustments.

4.0.7 Random forest + TF-IDF

In this section, the results of Random forest + TF-IDF are presented. Table 4.4 shows the classification report.

Table 4.4. Classification report of Random forest

	precision	recall	f1-score	support	predicted negative	predicted positive
0	0.9577	0.9810	0.9692	40159	0.9810	0.0190
1	0.7644	0.5876	0.6644	4214	0.4124	0.5876
accuracy			0.9436	44373		
macro avg	0.8611	0.7843	0.8168	44373		
weighted avg	0.9394	0.9436	0.9403	44373		

The model has demonstrated an overall accuracy of 94.36%, indicating its effectiveness in distinguishing between toxic and non-toxic comments. This high level of accuracy shows that the model is robust in general classification tasks within this specific domain. When analyzing class-specific metrics, the model has shown remarkable proficiency in identifying non-toxic comments (Class 0). It has achieved a precision of 95.77%, a recall of 98.10%, and an F1-score of 96.92%. The high performance, coupled with a true negative rate (TN) of 98.10%, suggests that the model is highly effective in correctly identifying non-toxic comments, minimizing the risk of false positives (FP rate of 1.90%).

The model performs well in detecting non-toxic comments (Class 0) but struggles when it comes to identifying toxic comments (Class 1). Specifically, the precision for Class 1 is 76.44%, the recall is 58.76%, and the F1-score is 66.44%. While the model can identify toxic comments with reasonable accuracy, it misses many of them, resulting in a false negative rate of 41.24%. This is likely due to an imbalanced dataset, where the model has been trained on more non-toxic comments, leading to a bias towards predicting comments as non-toxic.

The fact that the model performs better on non-toxic comments highlights a significant challenge in toxic comment detection: ensuring sensitivity towards less frequent but crucial toxic comments. The model's higher rate of false negatives for Class 1 indicates a need to improve its ability to detect nuanced or less obvious forms of toxicity.

Using a Random Forest classifier combined with TF-IDF for toxic comment detection has its own set of advantages and drawbacks. Here's a breakdown:

Pros:

- **Robustness to Overfitting:** Random Forests create multiple decision trees, using averaging or majority vote, which reduces overfitting compared to individual trees.
- **Feature Importance:** Random Forest can rank features based on their importance, providing insights into which terms indicate toxicity.

-
- **Non-linearity:** Random Forests can model non-linear relationships in the data, which is an advantage over models like LinearSVM.
 - **Easy to Use:** Random Forests are easy to train without much tuning and do not require feature scaling, making them beginner-friendly.
 - **Handle Noisy Data:** Random Forests are robust to noisy data, which is beneficial for processing unclean data.
 - **Parallel Processing:** The training of individual trees can be parallelized, facilitating faster training on multi-core machines.

Cons:

- **Computational Complexity:** Random Forests can be computationally intensive, both in terms of memory and time, especially as the number of trees increases.
- **Non-contextual Representation:** Random Forests using TF-IDF consider words in isolation and do not capture the order or deeper semantic meanings of the words.
- **Sparse Representation:** TF-IDF can result in very sparse matrices, which increase computational requirements for Random Forests, particularly with large vocabularies.
- **Interpretability:** Although Random Forests provide feature importance metrics, understanding the exact reasoning behind a prediction can be challenging due to the ensemble nature of the model.
- **Potential for Overhead:** If Random Forests are not tuned properly, an excessive number of trees can introduce unnecessary computational overhead with little to no performance benefit.
- **Lack of Sequential Information:** Random Forests ignore the sequence of words in TF-IDF representations, missing important context cues in textual data.
- **Potential to Overfit with Noise:** Although Random Forests are generally robust to overfitting, they can still overfit in situations where the data is excessively noisy or if the trees are too deep.
- **Hyperparameter Tuning:** Despite decision trees' ease of use, tuning hyperparameters such as tree depth, number of trees, and sampling techniques is often necessary for optimal performance.

To summarize, using a combination of Random Forest and TF-IDF can prove to be an effective technique for detecting toxic comments. However, it is important to consider the potential challenges regarding scalability and representation. While advanced models such as deep learning models may have an edge in capturing semantic and contextual nuances present in the data, Random Forests can still serve as a strong, non-linear baseline model in many scenarios.

4.0.8 LinearSVM + TF-IDF

This section presents the results of LinearSVM + TF-IDF.

The parameters C and max_iter were determined using `GridSearchCV`. The C parameter represents the regularization strength, a value of `0.31622776601683794` that balances the trade-off between minimizing errors and maintaining a simple model. The model iterates up to 2000 times to find the optimal decision boundary. These parameter values were selected as the best combination to optimize model performance through cross-validation.

Table 4.5. Classification report of LinearSVM

	precision	recall	f1-score	support	predicted negative	predicted positive
0	0.9811	0.9213	0.9502	40159	0.9213	0.0787
1	0.5254	0.8306	0.6436	4214	0.1694	0.8306
accuracy			0.9126	44373		
macro avg	0.7532	0.8759	0.7969	44373		
weighted avg	0.9378	0.9126	0.9211	44373		

As presented in table 4.5, the LinearSVM classifier utilizing TF-IDF feature extraction has displayed a strong performance in classifying toxic and non-toxic text, achieving an overall accuracy rate of 91.26%. For non-toxic comments (class 0), the model has demonstrated an impressive precision rate of 98.11% and a recall rate of 92.13%, resulting in an excellent f1-score of 95.02%. These statistics suggest the model's robust capability in accurately identifying non-toxic comments.

In identifying toxic comments (class 1), the model performs with reasonable effectiveness, albeit not as robustly as with non-toxic text. It boasts a high recall rate of 83.06%, signifying its ability to detect the majority of toxic comments. However, a lower precision rate of 52.54% results in a modest f1-score of 64.36%. This discrepancy indicates a tendency of the model to generate a significant number of false positives in the process of toxic comment detection.

The model's performance metrics reveal its struggle to strike a balance between sensitivity and specificity. A False Positive Rate (FPR) of 7.87% implies the model occasionally misclassifies non-toxic texts as toxic. Conversely, a False Negative Rate (FNR) of 16.94% indicates that while the model is cautious in labeling texts as non-toxic, it overlooks a substantial portion of toxic content.

Utilizing Linear Support Vector Machine with TF-IDF for toxic comment detection is a common and effective approach (Malmasi and Zampieri 2018; Greevy and Smeaton 2004). Here are the pros and cons of this combination:

Pros:

- **Simplicity and Interpretability:** LinearSVM provides a hyperplane that best separates the classes, making the model straightforward and interpretable. It highlights the influential features in predicting toxicity.
- **Handling of High Dimensionality:** The model excels in high-dimensional spaces, typical for TF-IDF, where the feature set can be extensive.
- **Computational Efficiency:** The LinearSVC class in scikit-learn is optimized for sparse data, offering computational benefits.
- **Strong Baseline:** TF-IDF combined with LinearSVM establishes a robust baseline for text classification tasks, often achieving competitive results.
- **Imbalanced Data Adaptability:** SVMs can be tuned to handle imbalanced data effectively using various techniques.
- **Noise Robustness:** SVMs, particularly with regularization, are adept at managing noisy datasets.

Cons:

- **Non-contextual Representation:** TF-IDF does not consider the semantic meaning or order of words, which can result in suboptimal performance for context-dependent tasks.
- **Scalability:** Training a LinearSVM on a large dataset can be computationally intensive, and while it handles high dimensionality well, it may need to scale more efficiently for very large corpora.
- **Hyperparameters:** Both SVM and TF-IDF have hyperparameters that require tuning, such as the regularization parameter for SVM and the n-gram range, max/min document frequency for TF-IDF. Additional computation is needed for cross-validation.
- **Lack of Deep Understanding:** Unlike deep learning models, this combination does not inherently capture deeper semantic meanings or hierarchical features of the text.
- **Sparse Representations:** When dealing with large vocabularies, the sparse nature of TF-IDF representations can cause excessive memory consumption.
- **Sensitivity to Irrelevant Features:** While SVM attempts to identify the optimal hyperplane to distinguish between classes, it may still be influenced by irrelevant or less informative features in the TF-IDF representation.
- **Harder to Capture Sequential Information:** TF-IDF neglects word order, hindering meaning extraction from sequences. Recurrent or transformer-based models can handle this.

In summary, using LinearSVM with TF-IDF for toxic comment detection is a widely adopted approach due to its efficiency and effectiveness in handling high-dimensional text data. However, challenges related to imbalanced data and sensitivity to outliers should be addressed. Proper hyperparameter tuning and considering data preprocessing techniques can maximize the benefits of this combination in accurately identifying toxic comments.

4.0.9 XGBoost

In this section, we present the results of the XGBoost model utilizing different text embeddings, specifically TF-IDF, Word2Vec, and GloVe.

Table 4.6. Classification report of XGboost + TF-ID

	precision	recall	f1-score	support	predicted negative	predicted positive
0	0.9749	0.9639	0.9694	40159	0.9639	0.0361
1	0.6892	0.7636	0.7245	4214	0.2364	0.7636
accuracy			0.9449	44373		
macro avg	0.8321	0.8638	0.8469	44373		
weighted avg	0.9478	0.9449	0.9461	44373		

Table 4.7. Classification report of XGboost + GloVe Embedding

	precision	recall	f1-score	support	predicted negative	predicted positive
0	0.9354	0.8796	0.9066	40159	0.8796	0.1204
1	0.2686	0.4215	0.3281	4214	0.5785	0.4215
accuracy			0.8360	44373		
macro avg	0.6020	0.6505	0.6174	44373		
weighted avg	0.8721	0.8360	0.8517	44373		

Table 4.8. Classification report of XGBoost + Word2Vec Embedding

	precision	recall	f1-score	support	predicted negative	predicted positive
0	0.9381	0.8834	0.9099	40159	0.8834	0.1166
1	0.2857	0.4445	0.3479	4214	0.5555	0.4445
accuracy			0.8417	44373		
macro avg	0.6119	0.6639	0.6289	44373		
weighted avg	0.8761	0.8417	0.8566	44373		

The following comparative analysis is based on the classification reports for the XGBoost model. These reports are detailed for TF-IDF embedding in Table 4.6, GloVe embedding in Table 4.7, and Word2Vec embedding in Table 4.8.

XGBoost + TF-IDF:

- The model has a high F1-score of 96.94%, performing well on non-toxic comments with high precision and recall.
- It achieves a moderate precision of 68.92% and a relatively high recall of 76.36% for toxic comments (class '1'), resulting in the highest F1-score (72.45%) among the three models.
- The model's overall accuracy is 94.49%, with a strong performance across the dataset indicated by a weighted average F1-score of 94.61%.

-
- The true positive rate (TP) and true negative rate (TN) are both high, suggesting the model effectively identifies toxic comments while maintaining a good rate of correctly identifying non-toxic comments.

XGBoost + GloVe Embedding:

- Similar to the Word2Vec model, the TF-IDF model has lower precision and recall for non-toxic comments, resulting in a comparable F1-score (90.66%).
- The precision for toxic comments is the lowest at 26.86%, although it has a similar recall to the Word2Vec model (42.15%), leading to a similar F1-score (32.81%).
- Overall accuracy is the lowest among the three models at 83.60%, with a weighted average F1-score of 85.17%.
- The model also has a high false positive and false negative rate, indicating challenges in accurately classifying toxic comments.

XGBoost + Word2Vec Embedding:

- This model has a lower F1-score (90.99%) for non-toxic comments than TF-IDF due to lower precision and recall.
- Precision for toxic comments drops to 28.57% with a recall of 44.45%, indicating the model's misclassification of toxic comments.
- The overall accuracy of the data falls to 84.17%, while the weighted average F1-score is 85.66%.
- The model has a higher false positive rate (FP) and false negative rate (FN), which is not ideal for a toxic comment detection system.

Here are the pros and cons of using XGBoost (Si et al. 2019):

Pros:

- **High Performance:** XGBoost is a highly performant machine learning algorithm that optimizes tree pruning, handles missing values, and regularization, allowing it to deliver superior results compared to other algorithms.
- **Regularization:** XGBoost incorporates L1 (Lasso) and L2 (Ridge) regularization, which can prevent overfitting and make it more robust.
- **Handling Missing Values:** XGBoost can handle missing values, providing more flexibility with incomplete data.
- **Parallel Processing:** XGBoost efficiently utilizes multiple cores for faster training on large datasets.
- **Feature Importance:** XGBoost provides feature importance scores, indicating which terms in TF-IDF indicate more toxicity.

-
- **Flexibility:** It supports linear and non-linear models and can be fine-tuned to specific tasks, offering great flexibility.
 - **Cross-validation:** XGBoost implements efficient cross-validation for hyperparameter tuning.

Cons:

- **Computational Intensity:** Training XGBoost can be computationally intensive, especially when using many boosting rounds. It is essential to consider the potential time and resource requirements when planning to train a model with XGBoost.
- **Memory Consumption:** When dealing with deep trees and large datasets, the algorithm tends to be memory-intensive.
- **Hyperparameter Tuning:** To achieve the best performance with XGBoost, one often needs to spend a significant amount of time tuning the hyperparameters.
- **Non-contextual Representation:** As with other TF-IDF models, this approach isolates words and misses the context and semantic relationships between them.
- **Sparse Representation Challenges:** Handling sparse TF-IDF matrices with gradient boosting is computationally challenging.
- **Potential for Overfitting:** If XGBoost models are not correctly regularized or trained with too many boosting rounds, they can overfit.
- **Lack of Sequential Information:** TF-IDF's representation may need attention to important contextual information derived from word sequence.
- **Complexity:** XGBoost can be more challenging to set up and optimize than simpler models due to its wide range of hyperparameters and configurations.

In terms of performance, the XGBoost model that uses TF-IDF surpasses the models with Word2Vec and GloVe embeddings in nearly all metrics. It displays a better balance in accurately classifying both toxic and non-toxic comments. In contrast, the models that use Word2Vec and GloVe struggle with precision, which implies a tendency to misclassify non-toxic comments as toxic. This is especially troublesome for platforms where wrongly moderating non-toxic content can negatively impact user experience. The overall accuracy and weighted F1-scores are highest for the TF-IDF model, indicating that this feature representation is more effective for the toxic comment classification task in the given dataset. This is attributed to the fact that TF-IDF is a high-dimensional sparse representation that can capture unique aspects of the text, which is more effective for identifying toxic language. On the other hand, GloVe and Word2Vec provide dense representations that may need to be more effective in capturing the nuances of toxic language. The XGBoost with TF-IDF model is the optimal choice for detecting toxic comments due to its higher accuracy, balanced precision, and recall.

4.0.10 BERT Base

In this section, the results of BERT are presented.

Table 4.9. Classification report of BERT Base

	precision	recall	f1-score	support	predicted negative	predicted positive
0	0.9707	0.9786	0.9746	40159	0.98	0.021
1	0.7791	0.7181	0.7473	4214	0.28	0.72
accuracy			0.9539	44373		
macro avg	0.8749	0.8484	0.8610	44373		
weighted avg	0.9525	0.9539	0.9530	44373		

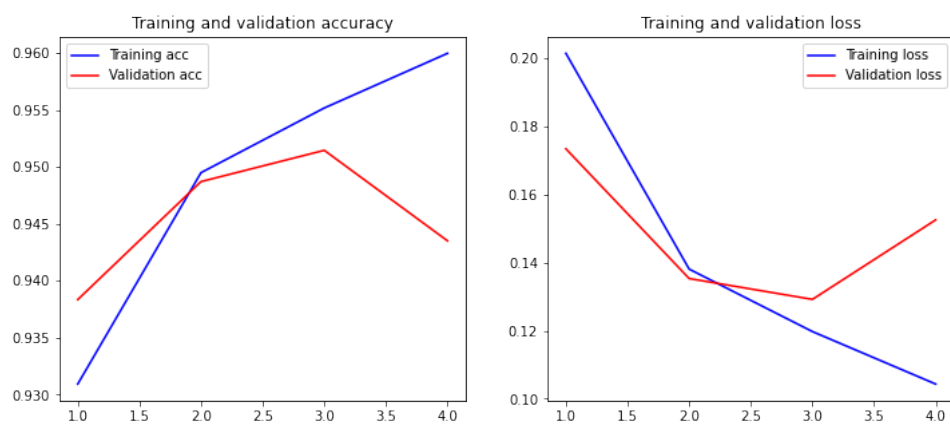


Figure 4.3. Training loss and training accuracy of model Bert Base

According to the classification report shown in Table 4.9, the model effectively determines between non-toxic (class 0) and toxic comments (class 1). The high precision and recall values for class 0 indicate that the model has a solid ability to accurately identify non-toxic text with a low rate of false positives. The precision for identifying toxic comments in class 1 is slightly lower than for non-toxic comments but still reasonably high at 77.91%. This indicates a relatively small number of false positives. However, the recall score is 71.81%, which suggests that the model misses some toxic comments. The f1-score for class 1 is respectable and implies a reasonable balance between precision and recall, but it also shows room for improvement in correctly identifying toxic comments.

The model has an overall accuracy of 95.39%, which is impressive and indicates strong performance. However, the precision and recall macro averages for the minority class (toxic comments) are lower than the weighted average, which means that the model's performance on the minority class is not as high as that on the majority class (non-toxic comments). This is expected in imbalanced datasets.

Using BERT Base for identifying toxic comments has advantages and disadvantages:

Pros:

- **Contextualized Embeddings:** BERT provides contextualized word representations capturing subtle meanings based on context, which is highly beneficial for understanding toxic comments.
- **State-of-the-Art Performance:** BERT is known for its state-of-the-art NLP performance, particularly in text classification tasks.
- **Pre-Trained Knowledge:** BERT is pre-trained on a large corpus of text, which allows the model to understand language nuances even before being fine-tuned for detecting toxic comments.
- **Handling Polysemy:** BERT's bidirectional nature enables it to handle polysemy better than traditional embeddings.
- **Fine-Tuning for Specific Tasks:** BERT can be fine-tuned with additional layers for better accuracy in detecting toxic comments.
- **Transfer Learning:** BERT enables transfer learning, adapting a model from one task to another and reducing the need for a labeled dataset for toxic comment detection.

Cons:

- **Computational Resources:** BERT requires significant computational power for fine-tuning and inference, which can be expensive.
- **Complexity in Fine-Tuning:** Fine-tuning BERT for a specific task can be challenging due to the need for a good understanding of the model and what parameters to adjust.
- **Memory Intensive:** BERT's memory requirements can exceed hardware capabilities, especially for long text sequences.
- **Overfitting Risk:** When dealing with smaller datasets, proper regularization is crucial to prevent overfitting of BERT due to its size and complexity.
- **Real-Time Latency:** Due to its size, BERT may not be optimal for real-time applications.
- **Limited Interpretability:** Due to its complexity, BERT's interpretation can be challenging, and determining the input text's most crucial aspects for predictions is difficult.
- **Tokenization Issues:** BERT uses subword tokenization, which can sometimes split words into pieces that might lose the meaning necessary for detecting toxicity in a nuanced way.
- **Pre-Trained Corpus Bias:** If the pre-trained BERT model has not been exposed to similar types of text or slang commonly found in toxic comments, its performance might be suboptimal.

-
- **Inference Time:** BERT's architecture can cause slower inference times, making it challenging to deploy in environments where immediate response is critical.

While BERT Base is a powerful model for NLP tasks, including toxic comment detection (Mozafari, Farahbakhsh, and Crespi 2019), it requires careful consideration of the trade-offs between performance, computational cost, and deployment complexity.

4.0.11 DistiBert

In this section, the results of DistiBert are presented.

Table 4.10. Classification report of DistiBert

	precision	recall	f1-score	support	predicted negative	predicted positive
0	0.9773	0.9763	0.9767	40159	0.98	0.024
1	0.7754	0.7841	0.7797	4214	0.22	0.78
accuracy			0.9579	44373		
macro avg	0.8764	0.8801	0.8782	44373		
weighted avg	0.9581	0.9579	0.9580	44373		

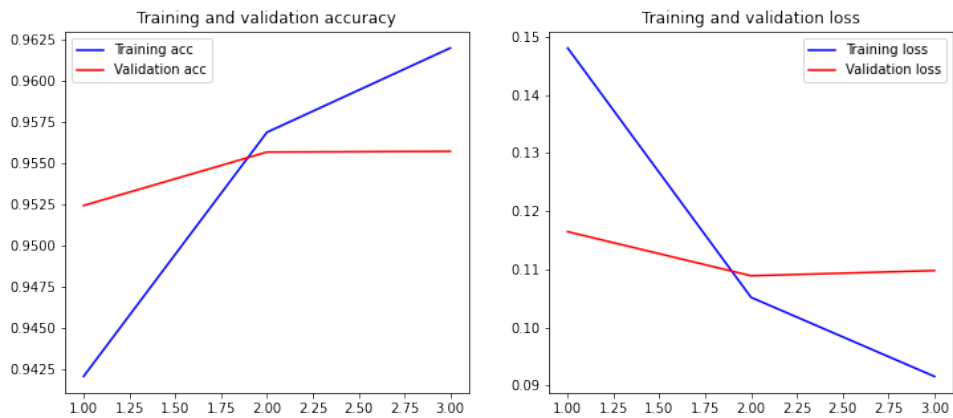


Figure 4.4. Training loss and training accuracy of model DistiBert

Based on the results presented in Table 4.10, it can be concluded that DistilBERT performs exceptionally well in text classification. The classification report indicates that DistilBERT effectively identifies non-toxic text (class 0) with a high degree of precision (97.73%) and recall (97.63%). Consequently, the f1-score is 97.67%, which illustrates the accuracy of DistilBERT in classifying non-toxic comments. Additionally, the minimal rate of false positives and false negatives adds to the effectiveness of DistilBERT in accurately classifying non-toxic comments.

The model performs well in identifying toxic text with a reasonably high precision of 77.54% and recall of 78.41%. The f1-score of 77.97% is also good. Although these figures are lower than for non-toxic text, they still indicate the model’s ability to perform well. The model has a higher recall for identifying true toxic comments than avoiding misclassification of non-toxic text, resulting in a lower precision.

The overall accuracy of the model is excellent at 95.79%. This suggests that DistilBERT performs well in both classes. The macro average precision, recall, and f1-score scores are high, indicating good performance across classes. The weighted averages are also high, which is significant given the difference in support between the classes, with non-toxic comments being more prevalent.

The training and validation performance of the DistilBERT model, as shown in Figure 4.4, demonstrates initial success results, with an improvement in accuracy and a decrease in loss. However, the model's validation accuracy has decreased, and its validation loss has increased after the second epoch, indicating potential overfitting. Overfitting occurs when a model is trained too much on the training data, leading to poor performance on new data.

DistilBERT, a smaller, faster, and lighter version of BERT designed to distill its knowledge into a more compact form, has its advantages and disadvantages when used for detecting toxic comments (Duchene et al. 2023):

Pros:

- **Efficiency:** DistilBERT is more efficient than BERT, requiring less computational resources for practical applications.
- **Performance:** DistilBERT is smaller than BERT but still performs well in detecting toxic comments, striking a good balance between efficiency and effectiveness.
- **Flexibility:** DistilBERT's smaller size enables deployment on devices with limited computational capacity.
- **Faster Inference:** The reduced complexity of the model results in faster inference times, which is crucial for real-time applications.
- **Pre-Trained Knowledge:** DistilBERT, like BERT, benefits from transfer learning; it is pre-trained on a large corpus, which gives it a solid understanding of the language for the task.

Cons:

- **Reduced Capacity:** The distillation process reduces the model's capacity, which may cause a slight decrease in performance compared to BERT, especially for complex tasks.
- **Resource Intensity:** DistilBERT is more resource-intensive than traditional ML models, which could be problematic for some applications despite its greater efficiency than BERT.
- **Risk of Overfitting:** On smaller datasets, there is a potential risk that DistilBERT could overfit if not adequately fine-tuned.
- **Contextual Limitations:** DistilBERT may capture less contextual information than BERT, which could limit its ability to understand subtle language nuances in toxic comments.
- **Pre-Trained Corpus Bias:** DistilBERT's performance can be affected by the pre-training corpus, which may not cover all language variations and slang used in toxic comments, similar to BERT.
- **Tokenization Concerns:** The subword tokenization mechanism can challenge understanding newly created words or slang used in toxic comments.

DistilBERT is a more affordable alternative to BERT for detecting toxic comments. It offers increased efficiency but sacrifices some performance. The use of DistilBERT in real-world applications will depend on specific needs for speed, resource consumption, and the complexity of the language that needs to be processed.

4.0.12 XLM-RoBERTa

The following section presents the outcomes obtained from XLM-RoBERTa.

Table 4.11. Classification report of XLM-RoBERTa

	precision	recall	f1-score	support	predicted negative	predicted positive
0	0.9746	0.9822	0.9784	40159	0.98	0.24
1	0.8171	0.7561	0.7854	4214	0.018	0.76
accuracy			0.9608	44373		
macro avg	0.8959	0.8691	0.8819	44373		
weighted avg	0.9596	0.9608	0.9601	44373		



Figure 4.5. Training loss and training accuracy of model XLM-RoBERTa

Based on Table 4.11 of the classification report for XLM-RoBERTa, the model is highly effective at distinguishing between toxic and non-toxic text. It has a remarkable precision rate of 97.46% and an outstanding recall rate of 98.22% for non-toxic text (class 0). This leads to an almost perfect f1-score of 97.84%. This indicates that the model is exceptionally accurate in identifying non-toxic comments and has a very low false-negative rate. This means that the model rarely misclassifies toxic comments as non-toxic. The model performs well in identifying toxic text, with a precision of 81.71% and a recall of 75.61%, though the precision needs improvement. The f1-score for toxic text is 78.54%, which is strong, but there is still room for improvement in terms of precision. The lower precision indicates that the model is good at detecting toxic comments but also has a higher false-positive rate, labeling some non-toxic comments as toxic.

The model has an impressive accuracy rate of 96.08%, indicating that it can correctly classify most of the text. Additionally, the model's performance across precision, recall,

and f1-score is consistent for different classes, as shown by the high macro average and weighted average scores.

As per Figure 4.5, the training and validation curves of the XLM-RoBERTa model show a successful learning phase, with training accuracy exceeding validation accuracy after some initial epochs. However, there are signs of overfitting, which can be observed as validation accuracy reaches its peak and then declines after the fifth epoch, while validation loss begins to increase. This indicates that although the model can effectively classify toxic and non-toxic text, it may require adjustments such as early stopping to prevent over-learning and ensure that it remains generalizable to unseen data.

XLM-RoBERTa is a cross-lingual language model that is an advanced version of the RoBERTa model. It is designed to work efficiently for multilingual applications as it is trained in multiple languages. However, before using XLM-RoBERTa for toxic comment detection, it is essential to consider the following pros and cons.

Pros:

- **Multilingual Support:** XLM-RoBERTa is trained in 100 languages, making it useful for detecting toxic comments across different languages without separate models.
- **Strong Contextual Understanding:** Like BERT and RoBERTa, it generates deep contextualized word embeddings, leading to a nuanced understanding of language that can be critical for identifying subtle toxic comments.
- **Robust Performance:** It has been fine-tuned on large-scale datasets, improving NLP performance.
- **Transfer Learning Capabilities:** XLM-RoBERTa can improve performance in low-resource languages by leveraging knowledge across multiple languages.
- **State-of-the-Art Models:** For many multilingual NLP tasks, XLM-RoBERTa is the state-of-the-art model, providing high accuracy in text classification tasks such as toxic comment detection (Conneau et al. 2019).

Cons:

- **Resource Intensive:** Training and inference with XLM-RoBERTa may be computationally expensive due to its size and complexity.
- **Fine-Tuning Complexity:** Optimally fine-tuning XLM-RoBERTa for specific tasks can be challenging and requires significant expertise and computational resources.
- **Language Imbalances:** Although it supports multiple languages, performance can be inconsistent due to imbalanced training data or underrepresented languages.
- **Inference Speed:** Due to its complex architecture, XLM-RoBERTa may not be ideal for real-time systems due to slower inference speeds.

-
- **Model Size and Storage:** The model's size may cause storage and memory issues, particularly in low-resource environments.
 - **Complex Preprocessing:** Like BERT-based models, careful preprocessing is required for proper text handling, including tokenization and special token management.

XLM-RoBERTa is ideal for detecting toxic comments across multiple languages. However, deploying the model can be challenging due to its resource demands and complexity, especially in cases where there are limited computational resources or a real-time response is required.

4.0.13 CNN

In this section, we present the results of the CNN model using different text embeddings, specifically GloVe and FastText (Joulin et al. 2016).

Convolutional Neural Networks (CNNs) are often used for image processing but can also be effectively applied to text classification tasks (Kim 2014; Dos Santos and Gatti de Bayser 2014):

Pros:

- **Effective Feature Extraction:** CNNs extract high-level features from text data, capturing local patterns like n-grams.
- **Parallel Computation:** CNNs can be trained and inferred faster by taking advantage of their inherent parallelizability using appropriate hardware.
- **Model Complexity:** CNNs typically have fewer parameters than RNNs, making them less prone to overfitting and more efficient to train.
- **Handling of Variable Input Sizes:** CNNs are ideal for processing variable-length inputs, such as comments.

Cons:

- **Limited Sequential Information:** Although CNNs can capture patterns, they are less effective than RNNs or Transformers at capturing sequential dependencies in text.
- **Hyperparameter Sensitivity:** CNNs require careful tuning of hyperparameters such as filter sizes and number of filters to achieve optimal performance on text data.
- **Less Interpretability:** Compared to simpler models, CNNs are less interpretable because it is challenging to understand the features they use for predictions.
- **Potential Overfitting:** When the dataset is not large enough, CNNs can overfit the training data without proper regularization, hindering generalization.

- **Sequential Information:** CNNs are not well-suited for tasks where word order is crucial for understanding, as their primary strength is recognizing patterns regardless of position.

Table 4.12. Classification report of CNN + GloVe Embedding

	precision	recall	f1-score	support	predicted negative	predicted positive
0	0.9622	0.9854	0.9736	40159	0.9854	0.0146
1	0.8189	0.6308	0.7126	4214	0.3692	0.6308
accuracy			0.9517	44373		
macro avg	0.8905	0.8081	0.8431	44373		
weighted avg	0.9486	0.9517	0.9488	44373		

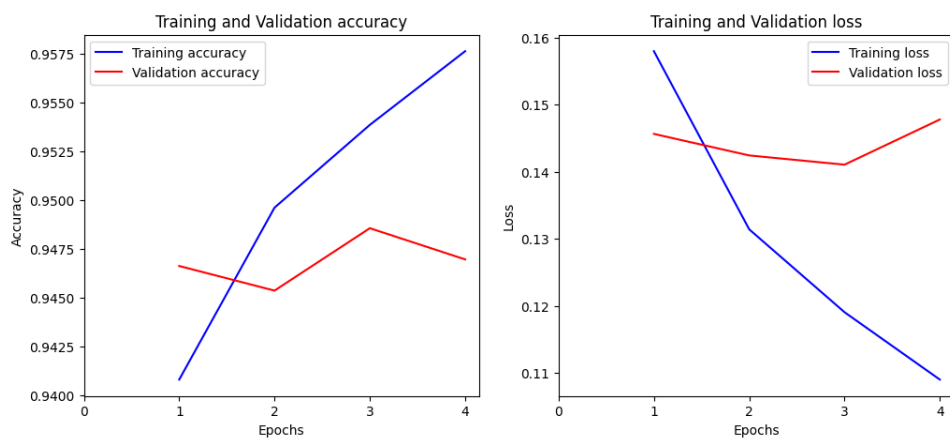


Figure 4.6. Training loss and training accuracy of model CNN + GloVe Embedding

Table 4.13. Classification report of CNN + FastText Embedding

	precision	recall	f1-score	support	predicted negative	predicted positive
0	0.9688	0.9755	0.9721	40159	0.9755	0.0245
1	0.7501	0.7003	0.7243	4214	0.2997	0.7003
accuracy			0.9494	44373		
macro avg	0.8594	0.8379	0.8482	44373		
weighted avg	0.9480	0.9494	0.9486	44373		

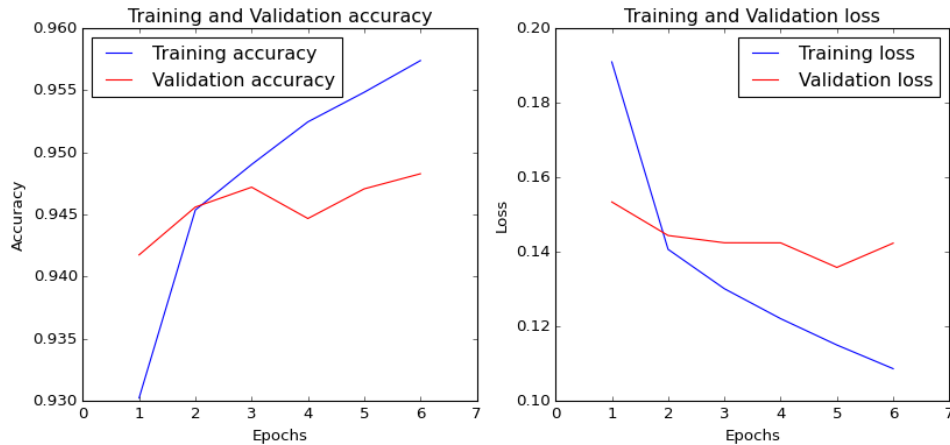


Figure 4.7. Training loss and training accuracy of model CNN + FastText Embedding

Based on the classification reports for the CNN model using GloVe and FastText embeddings in Tables 4.12 and 4.13, respectively, we can make the following observations:

CNN + GloVe Embedding:

- The model achieves a high F1-score (97.36 %) due to its precise (96.22%) and recall (98.54%) performance for non-toxic comments.
- The model’s performance metrics are lower for toxic comments (class ‘1’) with a precision of 81.89 %, recall of 63.08%, and F1-score of 71.26%.
- Overall accuracy is high at 95.17%, and the weighted average F1-score is strong at 94.88%.
- The model’s low false positive rate (1.46%) means it is unlikely to label non-toxic comments as toxic, though it is more likely to miss toxic comments with a higher false negative rate (36.92%).

CNN + FastText Embedding:

- Non-toxic comments have higher precision (96.88%) but slightly lower recall (97.55%) than the GloVe embedding. This results in a comparable F1-score (97.21%).
- For toxic comments, the precision is lower (75.01%) than the GloVe, but the recall is higher (70.03%), leading to a slightly higher F1-score (72.43%).
- The overall accuracy is slightly lower at 94.94%, while the weighted average F1-score is almost identical to the GloVe at 94.86%.
- This model is less conservative and more likely to classify comments as toxic, with a higher false positive rate (FP: 2.45%) and a lower false negative rate (FN: 29.97%).

Both models perform well in classifying non-toxic comments, as indicated by high precision, recall, and F1-scores for class ‘0’. However, there are some differences in their approach to classifying toxic comments (class ‘1’). The GloVe model is more conservative, resulting

in a higher precision but lower recall. On the other hand, the FastText model takes a more balanced approach, achieving a higher recall but lower precision. This suggests that the FastText model may detect more toxic comments but with a higher risk of false positives. Overall, both models have similar accuracy, with GloVe having a slight advantage over FastText. The FastText model performs better in achieving a balance between precision and recall for toxic comments (class '1'), as reflected by the F1 score. To conclude, when choosing between GloVe and FastText embeddings for a CNN-based toxic comment detection model, it is essential to consider the specific requirements of the task. If the goal is to minimize false alarms (i.e., non-toxic comments being classified as toxic), then GloVe might be the preferred option. On the other hand, if the aim is to catch as many toxic comments as possible, then FastText would be a better choice despite causing more false positives. The decision should be based on the trade-off between these rates, considering the cost or impact of misclassifications in the particular application.

4.0.14 RNN

In this section, we present the results of the RNN model using different text embeddings such as GloVe and FastText.

Using Recurrent Neural Networks (RNNs) for toxic comment detection presents various strengths and weaknesses:

Pros:

- **Sequential Data Handling:** RNNs are designed to handle sequential data, making them suitable for processing word order in sentences for understanding context in comments.
- **Contextual Information:** They can capture dependencies between words, maintaining information over a sentence, which is essential for detecting toxicity.
- **Dynamic Outputs:** RNNs can handle variable-length input and output sequences, allowing flexibility in dealing with comments of different lengths.
- **Model Complexity:** Compared to more complex models like BERT, RNNs are more straightforward and require fewer computational resources to train.

Cons:

- **Vanishing Gradient Problem:** The effectiveness of RNNs in capturing long-range dependencies is limited due to the vanishing gradient problem they face.
- **Computational Intensity:** Although RNNs are simpler than Transformers, training them can still be computationally intensive, especially for larger datasets or deeper networks.
- **Training Challenges:** RNNs are challenging to train due to issues like vanishing or exploding gradients. This requires careful tuning and regularization strategies.
- **Inferior to More Advanced Models:** Transformers have surpassed RNNs for tasks requiring an understanding of complex language structures.

Table 4.14. Classification report of RNN + GloVe Embedding

	precision	recall	f1-score	support	predicted negative	predicted positive
0	0.9437	0.9859	0.9643	40159	0.9859	0.0141
1	0.7653	0.4395	0.5583	4214	0.5605	0.4395
accuracy			0.9340	44373		
macro avg	0.8545	0.7127	0.7613	44373		
weighted avg	0.9268	0.9340	0.9258	44373		

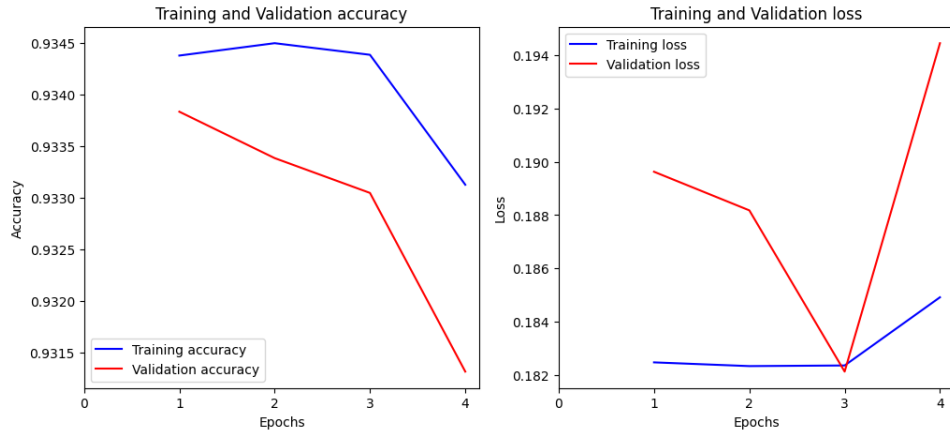


Figure 4.8. Training loss and training accuracy of model RNN + GloVe Embedding

Table 4.15. Classification report of RNN + FastText Embedding

	precision	recall	f1-score	support	predicted negative	predicted positive
0	0.9424	0.9885	0.9649	40159	0.9885	0.0115
1	0.7950	0.4243	0.5533	4214	0.5757	0.4243
accuracy			0.9349	44373		
macro avg	0.8687	0.7064	0.7591	44373		
weighted avg	0.9284	0.9349	0.9258	44373		

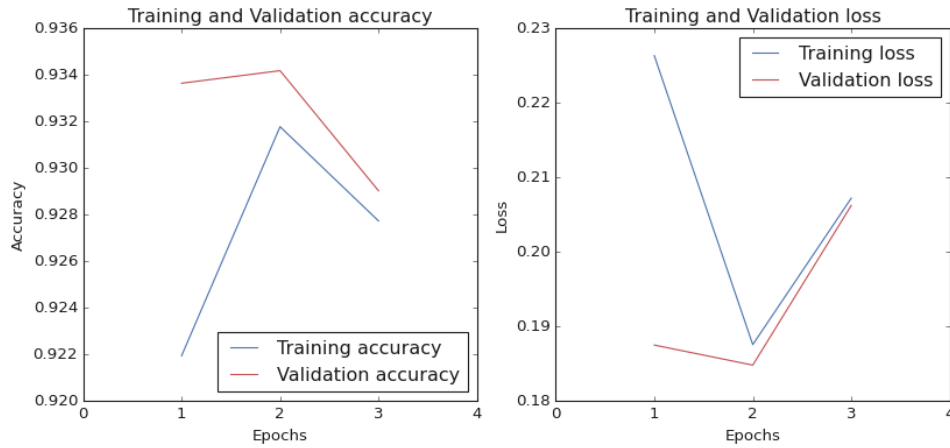


Figure 4.9. Training loss and training accuracy of model RNN + FastText Embedding

Based on the classification reports for the RNN model for GloVe and FastText embeddings detailed in Tables 4.14 and 4.15, respectively, the following analysis can be provided:

RNN + GloVe Embedding:

- The precision of non-toxic comments (class '0') is high (94.37%) with an excellent recall (98.59%), resulting in a high F1-score (96.43%).
- The model's precision is higher for non-toxic comments (class '0') with a score of 94.37%. However, when it comes to identifying toxic comments (class '1'), the precision drops to 76.53%, which means that it is not as accurate in identifying toxic comments. The recall score is also lower at 43.95%, indicating that the model fails to identify more than half of the toxic comments. As a result, the F1 score drops to 55.83%, which is lower than the ideal score.
- The model performs well but struggles with toxic comments (the minority class), as shown by an overall accuracy of 93.4% and a weighted average F1-score of 92.58%.

RNN + FastText Embedding:

- This model demonstrates high precision (94.24%) and recall (98.85%) for non-toxic comments, resulting in a slightly improved F1-score compared to the GloVe (96.49%).
- The precision of toxic comments is higher than GloVe (79.50%), but recall is slightly lower (42.43%), resulting in a comparable F1-score (55.33%).
- The overall accuracy (93.49%) and weighted average F1-score (92.58%) are similar to the GloVe embedding.

Both models perform well for non-toxic comments, with FastText having slightly better recall. GloVe has better recall for toxic comments, but FastText has higher precision. Both models have similar overall accuracy and F1 scores. However, false-negative rates concerning toxic comment detection are high, as many are not detected. While both the

GloVe and FastText models are effective in identifying non-toxic comments, they face difficulty in accurately classifying toxic comments. The slight differences in performance metrics between the two models must be more substantial to favor one. Factors other than the choice of word embeddings, such as model architecture or dataset balance, might play a more significant role in detecting toxic comments. To enhance the detection of toxic comments, strategies such as resampling the data to address class imbalance, tweaking the model to be more sensitive to the minority class, or incorporating additional contextual features should be considered.

4.0.15 LSTM

This section presents the results of using different text embeddings such as GloVe and FastText with the LSTM model.

Long Short-Term Memory (LSTM) networks belong to a class of recurrent neural networks (RNN) capable of learning long-term dependencies. They are widely used in various text classification tasks (Dubey et al. 2020). Here are their pros and cons:

Pros:

- **Long-term Dependency Learning:** LSTMs are designed to avoid long-term dependency problems, allowing them to remember information for prolonged periods, which is essential for understanding context in text sequences.
- **Contextual Information Processing:** They can process sequences of varying lengths, such as sentences and documents, while preserving word order.
- **Gate Mechanisms:** The input, output, and forget gates in LSTM structures regulate information flow to retain valuable data and discard irrelevant information throughout text sequences.
- **Bidirectional Processing:** Bidirectional LSTMs process data in both forward and backward directions, providing a more comprehensive context than unidirectional LSTMs.
- **Robustness to Sequence Length:** LSTMs are better at processing long sequences than vanilla RNNs, making them more suitable for handling longer text data.

Cons:

- **Complexity and Computation:** LSTMs can be computationally intensive to train with large vocabularies and long sequences.
- **Vanishing Gradients:** Despite improvements over vanilla RNNs, LSTMs can still suffer from vanishing gradient problems during training, particularly with very long sequences.
- **Contextual Limitations:** While LSTMs capture broad context, they may need to capture nuanced word-level context like Transformers.
- **Memory Intensiveness:** LSTMs require significant memory for backpropagation through time, which can be a limiting factor for training on large datasets.
- **Sequential Computation:** The nature of LSTM computations makes parallelization difficult, resulting in longer training times when compared to newer architectures.

Table 4.16. Classification report of LSTM + GloVe Embedding

	precision	recall	f1-score	support	predicted negative	predicted positive
0	0.9634	0.9836	0.9734	40159	0.9836	0.0164
1	0.8049	0.6443	0.7157	4214	0.3557	0.6443
accuracy			0.9514	44373		
macro avg	0.8842	0.8139	0.8446	44373		
weighted avg	0.9484	0.9514	0.9489	44373		

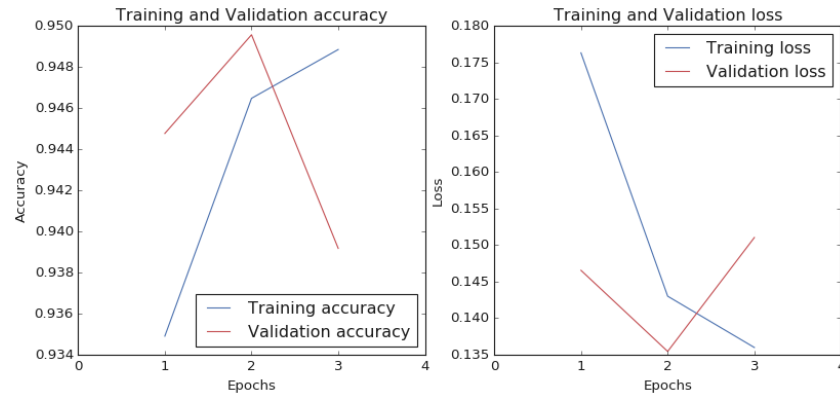


Figure 4.10. Training loss and training accuracy of model LSTM + GloVe Embedding

Table 4.17. Classification report of LSTM + FastText Embedding

	precision	recall	f1-score	support	predicted negative	predicted positive
0	0.9571	0.9850	0.9709	40159	0.9850	0.0150
1	0.8020	0.5795	0.6728	4214	0.4205	0.5795
accuracy			0.9465	44373		
macro avg	0.8795	0.7822	0.8218	44373		
weighted avg	0.9424	0.9465	0.9426	44373		

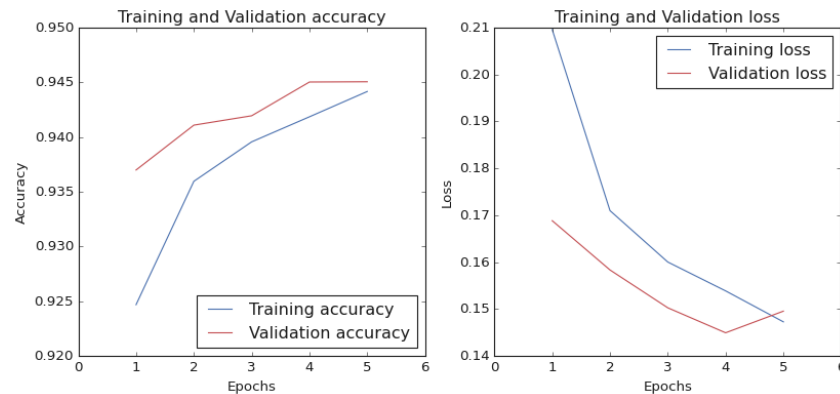


Figure 4.11. Training loss and training accuracy of model LSTM + FastText Embedding

Tables 4.16 and 4.17 show classification reports for LSTM models with GloVe and FastText embeddings and provide insights into their performance in detecting toxic comments:

LSTM + GloVe Embedding:

- The model demonstrates a strong F1-score (97.34%) due to high precision (96.34%) and recall (98.36%) for non-toxic comments (class '0').
- The model achieves a precision of 80.49% and a recall of 64.43% for toxic comments (class 1), resulting in an F1-score of 71.57%.
- The model achieves an overall accuracy of 95.14% and a weighted average F1-score of 94.89%, demonstrating strong performance across all areas.
- The model correctly identifies a good portion of toxic comments with a true positive rate of 64.43% but still misses a number with a false negative rate of 35.57%.

LSTM + FastText Embedding:

- Compared to the GloVe embedding, the model shows slightly lower precision (95.71%) but similar recall (98.50%) for non-toxic comments. The F1-score is almost equivalent (97.09%).
- The recall for toxic comments (57.95%) is notably lower than the model with GloVe (80.2%), resulting in a reduced F1-score (67.28%).
- Overall accuracy is slightly lower at 94.65%, and the weighted average F1-score is 94.26%, slightly below the GloVe performance.
- The model's true positive rate is only 57.95%, while its false negative rate is higher at 42.05%, indicating a lower effectiveness at identifying toxic comments.

Both LSTM models demonstrate outstanding performance in identifying non-toxic comments with high precision and recall. This indicates that they are reliable in recognizing non-toxic content. Regarding detecting toxic comments, the LSTM model that uses GloVe embedding outperforms the one that uses FastText embedding in precision and recall. Hence, it offers better overall performance in identifying toxic content. The GloVe model also achieves higher overall accuracy and weighted average F1-score, reinforcing its superior performance on this dataset. Additionally, the LSTM with GloVe embedding can effectively manage the trade-off between false positives and false negatives, making it a more balanced model.

Although both models exhibit good performance in identifying non-toxic comments, the LSTM with GloVe embeddings outperforms the other model in detecting toxic comments. Therefore, if the detection of toxic content is a crucial issue, using the LSTM model with GloVe embeddings is recommended. To improve the recall for detecting toxic comments without compromising precision, additional training data, hyperparameter tuning, or combining various model types can be considered.

4.0.16 GRU

This section presents the results of using different text embeddings such as GloVe and FastText with the GRU model.

Gated Recurrent Units (GRUs) are a type of recurrent neural network designed to capture dependencies in sequential data such as text. Here are the pros and cons of using GRUs for text classification (Mahajan, Shah, and Jafar 2021; Zhang, Robinson, and Tepper 2018):

Pros:

- **Efficient Learning:** GRUs (Gated Recurrent Units) simplify the gating mechanism used in LSTMs and can lead to faster training without significant loss in capability to capture dependencies (Yin et al. 2017).
- **Less Complexity:** GRUs have a more straightforward gating mechanism compared to LSTMs. The model has two gates, reset and update gates, instead of three, which reduces complexity.
- **Good Contextual Understanding:** GRUs can capture dependencies in a sentence, which is crucial for sentiment analysis.
- **Handling of Variable-Length Sequences:** The GRU is skilled at processing varying-length sequences, a common occurrence in text classification tasks where sentence and document lengths differ considerably.
- **Robustness to Vanishing Gradients:** The GRU's gating mechanisms mitigate the vanishing gradient problem, improving learning from longer sequences compared to traditional RNNs.

Cons:

- **Risk of Overfitting:** Although GRUs are simpler than LSTMs, they can still overfit the training data, mainly when the network is extensive or the training dataset is small.
- **Computational Requirements:** GRUs require significant computational resources despite being less complex than LSTMs, especially when processing large datasets or long sequences.
- **Long Sequences Challenge:** GRUs perform well with long sequences, but highly long dependencies can still be challenging. In some cases, LSTMs or Transformers may capture these more effectively.
- **Vanishing Gradient:** Although GRUs are less common, they can still suffer from vanishing gradients in practice, mainly when sequences are very long or when the model is not properly regularized.

Table 4.18. Classification report of GRU + GloVe Embedding

	precision	recall	f1-score	support	predicted negative	predicted positive
0	0.9639	0.9860	0.9748	40159	0.9860	0.0140
1	0.8295	0.6478	0.7275	4214	0.3522	0.6478
accuracy			0.9539	44373		
macro avg	0.8967	0.8169	0.8512	44373		
weighted avg	0.9511	0.9539	0.9513	44373		

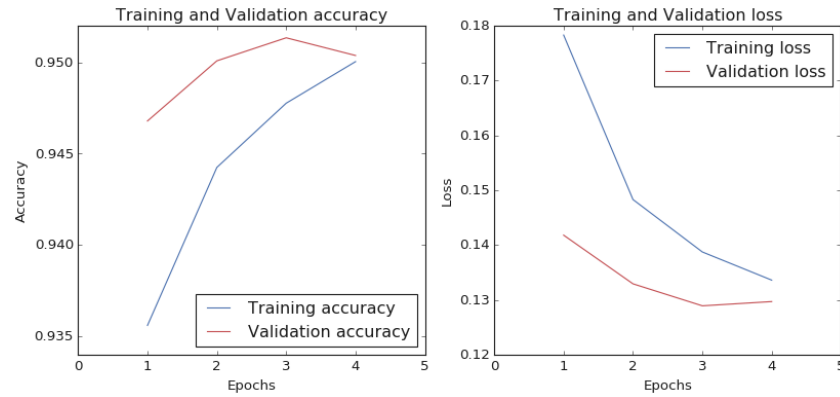


Figure 4.12. Training loss and training accuracy of model GRU + GloVe Embedding

Table 4.19. Classification report of GRU + FastText Embedding

	precision	recall	f1-score	support	predicted negative	predicted positive
0	0.9585	0.9842	0.9712	40159	0.9842	0.0158
1	0.7982	0.5940	0.6811	4214	0.4060	0.5940
accuracy			0.9472	44373		
macro avg	0.8783	0.7891	0.8261	44373		
weighted avg	0.9433	0.9472	0.9437	44373		

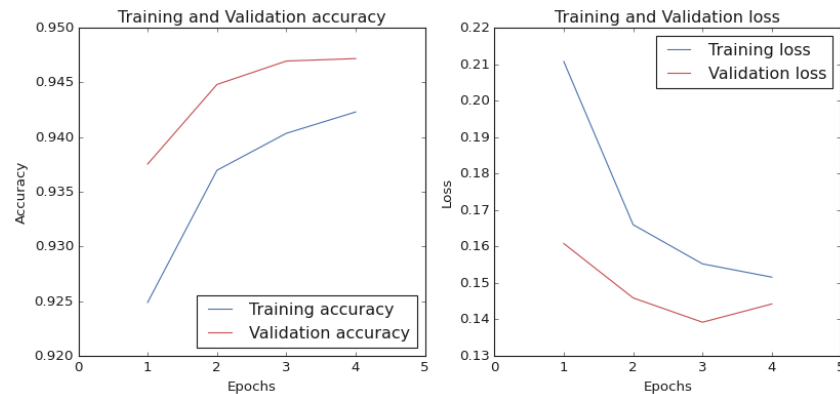


Figure 4.13. Training loss and training accuracy of model GRU + FastText Embedding

When analyzing the classification reports for the GRU models with GloVe and FastText embeddings in the context of toxic comment detection, we can deduce the following:

GRU + GloVe Embedding:

- The model achieved a high F1 score (97.48%) due to its precision (96.39%) and recall (98.60%) for non-toxic comments (class '0').
- The precision for class '1' toxic comments is high (82.95%), with reasonable recall (64.78%) and solid F1-score (72.75%).
- The model achieved an accuracy of 95.39% and a weighted F1-score of 95.13%, indicating strong performance across the dataset.
- The true positive rate (TP: 64.78%) suggests that the model correctly identifies a significant portion of the toxic comments. However, it also misses a number with a false negative rate of 35.22%.

GRU + FastText Embedding:

- Compared to the GloVe model, the non-toxic comments show a slightly lower precision (95.85%) and recall (98.42%), resulting in a slightly lower F1-score (97.12%).
- The precision for toxic comments is 79.82%, with a recall of 59.40%, leading to an F1-score of 68.11%, which is lower than the one obtained by the GloVe.
- The model achieved an overall accuracy of 94.72% and a weighted average F1-score of 94.37%, which is lower than the GloVe model.
- The model's true positive rate (TP: 59.40%) is lower, and its false negative rate (FN: 40.6%) is higher, indicating less effectiveness in identifying toxic comments.

Combining the GRU model with GloVe embeddings has proven to be more effective in identifying toxic comments than the one with FastText embeddings. The higher precision and recall for toxic comments suggest that the GloVe model correctly identifies the toxic content. This could be due to the GloVe model capturing more nuanced semantic relationships in the text, which can be particularly important for detecting the subtleties of toxic language. To further enhance the performance, especially in terms of recall for toxic comments, additional techniques such as upsampling the minority class, tweaking the loss function to penalize false negatives more, or incorporating contextual features could be considered.

5. Testing

To test our best model, we can collect Twitter tweets, which are valuable sources of human opinions, thoughts, feelings, and ideas. Twitter can be helpful in data mining for deriving political and social opinions and trends, as well as location-based brand and product mentions. However, according to a recent study by SimpleTexting, Twitter is also considered the most toxic social media platform (Norton 2022). Therefore, when searching for toxic tweets, it is essential to select a topic that triggers strong emotions, such as abortion rights.

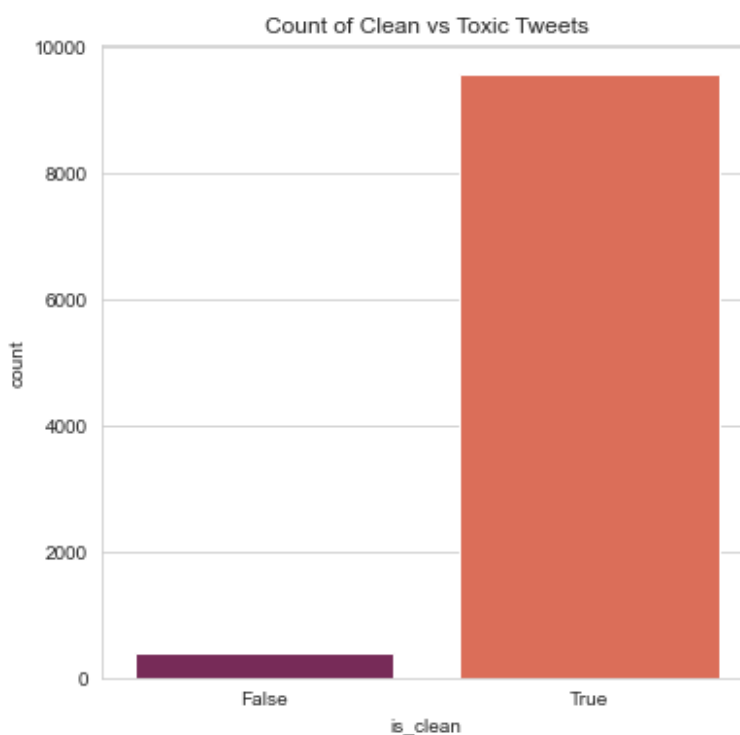


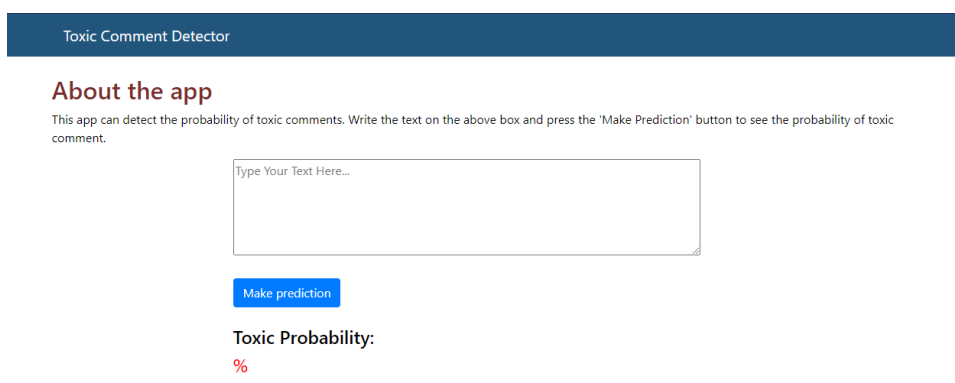
Figure 5.1. *Count of Clean vs Toxic Tweets*

According to Figure 5.1, the test set comprises 9,973 English tweets, out of which 4.28% are toxic. We collected these tweets using the hashtag "abortion" and labeled them using a custom script and Perspective API (Hosseini et al. 2017). The Perspective API is a free tool that leverages machine learning to identify toxic comments. After labeling the tweets, we utilized our best model XLM-RoBERTa to predict toxic tweets and compared the results with those obtained from Perspective. Our model achieved an overall accuracy of 93.23%.

6. Deployment

In order to provide a better and more convenient way to test our model, we decided to build a user interface for our best machine-learning model. So anyone can play with the model through a nice UI. Python offers different options for building web apps. Flask is one of the simplest options for quickly developing web applications. After the model was trained, we exported it to a file, imported it into our web application, and built a simple user interface.

In Figure 6.1, we can observe the web application's user interface, which consists of a text box that accepts English text and a "Make a Prediction" button. Once you click on the button, you can see the probability of toxic content.



The screenshot shows a web application titled "Toxic Comment Detector". Below the title is a section titled "About the app" with a description: "This app can detect the probability of toxic comments. Write the text on the above box and press the 'Make Prediction' button to see the probability of toxic comment." There is a text input field with the placeholder text "Type Your Text Here...". Below the input field is a blue button labeled "Make prediction". Underneath the button, the text "Toxic Probability:" is displayed, followed by a red percentage symbol "%".

Figure 6.1. Toxic Comment Detector web application

7. Conclusion

The aim of this thesis was to evaluate Natural Language Processing (NLP) techniques for detecting inappropriate content through text classification, transformation, and representation. We conducted extensive research, including literature reviews, tutorials from different sources, and practical implementations. As a result, we have developed a product that can take any English statement as input text, process it with a machine learning model, and generate the output as a toxic probability. We have created a simple web interface to make it easy to use. We also tested our best model for detecting negative content in tweets with the topic of "abortion rights".

The process of text classification involves three parts: preprocessing, word embeddings, and classification models. We chose the Jigsaw Multilingual Toxic Comment dataset for the training dataset, which includes comments from Civil Comments and Wikipedia that were manually annotated between 2004 and 2015. In preprocessing, we remove punctuation, English letters, numbers, and stopwords. We then segment the comments into individual words. We used five different techniques for word embedding: Word2Vec, TF-IDF, GloVe, FastText, and embedding layers. Finally, we used five machine-learning models and seven deep-learning models for classification.

As a result, the training model of XLM-RoBERTa showed the best result with an accuracy of 96 % on test data. In summary, based on the sentiment analysis of Twitter data, a large group of people have negative sentiments on the topics related to abortion rights.

8. Limitations and future work

8.0.1 Limitation

As part of our experiments, we utilized transformer-based base models, specifically BERT base. However, using BERT large can lead to better results. The primary difference between these two models is the number of encoder layers they contain. BERT base has 12 encoder layers stacked on top of each other, whereas BERT large has 24 layers of encoders. On the other hand, XLM-RoBERTa-Base has 125 million parameters, while XLM-RoBERTa-Large has 550 million parameters, making it a more potent version of the XLM-RoBERTa architecture. Although larger models can enhance results, they need more computational resources.

8.0.2 Future work

During our research, we encountered a common problem in the field of machine learning - imbalanced datasets. When a particular class has a disproportionate representation in comparison to others, it results in imbalanced data. Only 10.58% of comments in the training set were classified as toxic, while the remaining 89.42% were labeled as non-toxic. Despite using Synthetic Minority Over-sampling Technique (SMOTE) to address this imbalance, we observed that it did not effectively reduce false positives as anticipated.

SMOTE is known for its ability to enhance the representation of minority classes by generating synthetic samples. However, in our case, it failed to rectify the skewed class distribution to the required level for reducing false positive rates. This outcome indicates the complexity of dealing with imbalanced datasets and suggests the need for alternative or additional strategies to mitigate this issue.

We propose exploring advanced resampling techniques like Borderline-SMOTE or Adaptive Synthetic Sampling (ADASYN) to address this challenge. These methods focus on generating synthetic samples near the borderline of decision regions that could be more effective.

Another potential solution is to expand the dataset with more examples of the minority class or use data augmentation techniques to increase the dataset size artificially. These future steps aim to develop a more sophisticated and nuanced approach to handling imbalanced datasets in toxic comment detection, ultimately enhancing the model's performance and reliability.

REFERENCES

- Huang, J., J. J. Lu, and C. X. Ling (Dec. 2003). *Comparing naive Bayes, decision trees, and SVM with AUC and accuracy*. ISBN: 978-0-7695-1978. DOI: 10.1109/ICDM.2003.1250975.
- Greevy, Edel and Alan Smeaton (July 2004). “Classifying racist texts using a support vector machine”. In: *Greevy, Edel and Smeaton, Alan F. (2004) Classifying racist texts using a support vector machine. In: SIGIR 2004 - the 27th Annual International ACM SIGIR Conference, 25-29 July 2004, Sheffield, UK*. DOI: 10.1145/1008992.1009074.
- Mikolov, Tomas et al. (Jan. 2013). “Efficient Estimation of Word Representations in Vector Space”. In: *Proceedings of Workshop at ICLR 2013*. URL: https://www.researchgate.net/publication/234131319_Efficient_Estimation_of_Word_Representations_in_Vector_Space.
- Dos Santos, Cicero and Maira Gatti de Bayser (Aug. 2014). “Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts”. In: *ResearchGate*. URL: https://www.researchgate.net/publication/274380447_Deep_Convolutional_Neural_Networks_for_Sentiment_Analysis_of_Short_Texts.
- Kim, Yoon (Aug. 2014). “Convolutional Neural Networks for Sentence Classification”. In: *arXiv*. DOI: 10.48550/arXiv.1408.5882. eprint: 1408.5882.
- Pennington, Jeffrey, Richard Socher, and Christopher Manning (Oct. 2014). “GloVe: Global Vectors for Word Representation”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Alessandro Moschitti, Bo Pang, and Walter Daelemans. Doha, Qatar: Association for Computational Linguistics, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: <https://aclanthology.org/D14-1162>.
- Joulin, Armand et al. (July 2016). “Bag of Tricks for Efficient Text Classification”. In: *arXiv*. DOI: 10.48550/arXiv.1607.01759. eprint: 1607.01759.
- Badjatiya, Pinkesh et al. (June 2017). “Deep Learning for Hate Speech Detection in Tweets”. In: *arXiv*. DOI: 10.1145/3041021.3054223. eprint: 1706.00188.
- Gao, Lei and Ruihong Huang (Oct. 2017). “Detecting Online Hate Speech Using Context Aware Models”. In: *arXiv*. DOI: 10.48550/arXiv.1710.07395. eprint: 1710.07395.
- Hosseini, Hossein et al. (Feb. 2017). “Deceiving Google’s Perspective API Built for Detecting Toxic Comments”. In: *arXiv*. DOI: 10.48550/arXiv.1702.08138. eprint: 1702.08138.
- Vaswani, Ashish et al. (June 2017). “Attention Is All You Need”. In: *arXiv*. DOI: 10.48550/arXiv.1706.03762. eprint: 1706.03762.
- Yin, Wenpeng et al. (Feb. 2017). “Comparative Study of CNN and RNN for Natural Language Processing”. In: *arXiv*. DOI: 10.48550/arXiv.1702.01923. eprint: 1702.01923.
- Aken, Betty van et al. (Sept. 2018). “Challenges for Toxic Comment Classification: An In-Depth Error Analysis”. In: *ResearchGate*. URL: https://www.researchgate.net/publication/327790215_Challenges_for_Toxic_Comment_Classification_An_In-Depth_Error_Analysis.

-
- Devlin, Jacob et al. (Oct. 2018). “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *arXiv*. eprint: 1810.04805. URL: <https://arxiv.org/abs/1810.04805v2>.
- Georgakopoulos, Spiros V. et al. (Feb. 2018). “Convolutional Neural Networks for Toxic Comment Classification”. In: *arXiv*. DOI: 10.48550/arXiv.1802.09957. eprint: 1802.09957.
- Malmasi, Shervin and Marcos Zampieri (Mar. 2018). “Challenges in Discriminating Profanity from Hate Speech”. In: *arXiv*. DOI: 10.48550/arXiv.1803.05495. eprint: 1803.05495.
- Saif, M. A. et al. (Dec. 2018). “Classification of online toxic comments using the logistic regression and neural networks models”. In: *AIP Conf. Proc.* 2048, p. 060011. DOI: 10.1063/1.5082126.
- Zhang, Z., D. Robinson, and J. Tepper (Mar. 2018). “Detecting hate speech on Twitter using a convolution-GRU based deep neural network”. In: *ResearchGate*. URL: https://www.researchgate.net/publication/323723283_Detecting_hate_speech_on_Twitter_using_a_convolution-GRU_based_deep_neural_network.
- Conneau, Alexis et al. (Nov. 2019). “Unsupervised Cross-lingual Representation Learning at Scale”. In: *arXiv*. DOI: 10.48550/arXiv.1911.02116. eprint: 1911.02116.
- Mozafari, Marzieh, Reza Farahbakhsh, and Noel Crespi (Oct. 2019). “A BERT-Based Transfer Learning Approach for Hate Speech Detection in Online Social Media”. In: *arXiv*. DOI: 10.48550/arXiv.1910.12574. eprint: 1910.12574.
- Sanh, Victor et al. (Oct. 2019). “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter”. In: *arXiv*. DOI: 10.48550/arXiv.1910.01108. eprint: 1910.01108.
- Si, Shukrity et al. (July 2019). “Aggression Detection on Multilingual Social Media Text”. In: *ResearchGate*. DOI: 10.1109/ICCCNT45670.2019.8944868.
- Aluru, Sai Saketh et al. (Apr. 2020). “Deep Learning Models for Multilingual Hate Speech Detection”. In: *arXiv*. DOI: 10.48550/arXiv.2004.06465. eprint: 2004.06465.
- Dubey, Krishna et al. (Dec. 2020). “Toxic Comment Detection using LSTM”. In: *ResearchGate*, pp. 1–8. DOI: 10.1109/ICAIECC50550.2020.9339521.
- Aji, Nurseno Bayu, Musta’inul Abdi, and Ardon Rakhmadi (Apr. 2021). “Tf*Idf and Random Walk For Term Candidate Selection On Automatic Subject Indexing”. In: *JAICT* 6.1, pp. 38–43. ISSN: 2541-6359. URL: <https://jurnal.polines.ac.id/index.php/jaict/article/view/2436/pdf>.
- Hsu, Tiffany and Eleanor Lutz (Oct. 2021). *More Than 1,000 Companies Boycotted Facebook. Did It Work?* URL: <https://www.nytimes.com/2020/08/01/business/media/facebook-boycott.html>.
- Mahajan, Aditya, Divyank Shah, and Gibraan Jafar (May 2021). *Explainable AI Approach Towards Toxic Comment Classification*. ISBN: 978-981-33-4366-5. DOI: 10.1007/978-981-33-4367-2_81.
- Pennington, Jeffrey (June 2021). *GloVe: Global Vectors for Word Representation*. [Online; accessed 29. Mar. 2022]. URL: <https://nlp.stanford.edu/projects/glove>.

-
- Dang, Sheila and Katie Paul (Nov. 2022). “Twitter not safer under Elon Musk, says former head of trust and safety”. In: *Reuters*. URL: <https://www.reuters.com/technology/twitter-not-safer-under-elon-musk-says-former-head-trust-safety-conference-2022-11-29>.
- DSA: landmark rules for online platforms enter into force (Nov. 2022). [Online; accessed 16. Aug. 2023]. URL: https://ec.europa.eu/commission/presscorner/detail/en/IP_22_6906.
- Norton, Lily (Dec. 2022). *America Ranks the Most Toxic Social Media Apps - SimpleTexting*. [Online; accessed 16. May 2022]. URL: <https://simpletexting.com/most-toxic-social-media-apps>.
- Roepke, Brian (Feb. 2022). “Stop Using Accuracy to Evaluate Your Classification Models”. In: *Medium*. URL: <https://towardsdatascience.com/evaluating-ml-models-with-a-confusion-matrix-3fd9c3ab07dd>.
- Support Vector Machine (SVM) Algorithm - Javatpoint* (Aug. 2022). [Online; accessed 10. Aug. 2022]. URL: <https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm>.
- Antisemitism on Twitter Before and After Elon Musk’s Acquisition* (Mar. 2023). [Online; accessed 14. Nov. 2023]. URL: <https://www.isdglobal.org/wp-content/uploads/2023/03/Antisemitism-on-Twitter-Before-and-After-Elon-Musks-Acquisition.pdf>.
- Bayes’ theorem - Wikipedia* (Nov. 2023). [Online; accessed 18. Nov. 2023]. URL: https://en.wikipedia.org/w/index.php?title=Bayes%27_theorem&oldid=1185628279.
- Duchene, Corentin et al. (Jan. 2023). “A benchmark for toxic comment classification on Civil Comments dataset”. In: *arXiv*. DOI: 10.48550/arXiv.2301.11125. eprint: 2301.11125.
- Machine Learning Mastery* (Sept. 2023). [Online; accessed 15. Nov. 2023]. URL: <https://machinelearningmastery.com>.
- tf-idf - Wikipedia* (Nov. 2023). [Online; accessed 19. Nov. 2023]. URL: <https://en.wikipedia.org/w/index.php?title=Tf-idf&oldid=1183151429>.
- XGBoost ML Model in Python - Javatpoint* (Nov. 2023). [Online; accessed 19. Nov. 2023]. URL: <https://www.javatpoint.com/xgboost-ml-model-in-python>.