Engineering Degree Project

Detection of bullying with Machine Learning

- Using Supervised Machine Learning and LLMs to classify bullying in text

Author: Seif-Alamir Yousef
Author: Ludvig Svensson
Supervisor: Elias Holmér
Lnu Supervisor: Arslan Musaddiq
Semester: Spring 2024
Subject: Computer Science

Abstract

In recent years, there has been a increase in the issue of bullying, particularly in academic settings [1]. This degree project examines the use of supervised machine learning techniques to identify bullying in text data from school surveys provided by the Friends Foundation. It evaluates various traditional algorithms such as Logistic Regression, Naive Bayes, SVM, Convolutional Neural Networks (CNN), alongside a Retrieval-Augmented Generation (RAG) model using Llama 3, with a primary goal of achieving high recall on the texts consisting of bullying while also considering precision, which is reflected in the use of the F3-score. The SVM model emerged as the most effective among the traditional methods, achieving the highest F3-score of 0.83. Although the RAG model showed promising recall, it suffered from very low precision, resulting in a slightly lower F3-score of 0.79. The study also addresses challenges such as the small and imbalanced dataset as well as emphasizes the importance of retaining stop words to maintain context in the text data. The findings highlight the potential of advanced machine learning models to significantly assist in bullying detection with adequate resources and further refinement.

Keywords: Natural Language Processing, Bullying, Text Classification, SVM, Logistic Regression, Naive Bayes, Convolutional Neural Networks, Retrieval-augmented generation, GPT-4o

Preface

Our deepest appreciation goes to Elias Holmér, our project supervisor at Tietoevry, as well as Arslan Musaddiq, our project supervisor at Linnaeus University. Their expertise, knowledge, and guidance have been instrumental to the project's successful completion.

Lastly, profound appreciation is extended to Tietoevry for providing a conducive environment for this project, and to the Friends foundation for their generous data and expertise sharing, which significantly strengthened the project's outcome.

Contents

1 Introduction

This degree project investigates how Machine Learning can be used to identify bullying in text surveys. Specifically, it explores the use of Supervised Machine Learning and Large Language Models (LLMs) in collaboration with *Tietoevry* and the *Friends* foundation. Access to a unique dataset provided by the Friends Foundation allows for the evaluation of various Machine Learning models and algorithms. The aim is to determine the most effective methods for detecting instances of bullying, thus contributing valuable insights to the field. The authors hypothesize that LLMs will outperform traditional supervised machine learning models in detecting bullying in text surveys, due to their greater complexity and advanced capabilities.

1.1 Background

Bullying is a significant issue in Sweden, unfortunately increasing in recent years, especially in schools. Moreover, bullying increasingly occurs digitally due to the digital development in society. Current traditional detection methods rely on self-reporting or direct observations and are often ineffective and discover incidents too late [1].

Friends is a foundation dedicated to combat bullying [2]. They currently rely on manually reviewing surveys to identify cases. With the advent of artificial intelligence (AI), there is potential to automate the bullying detection process which will enhance efficiency, save valuable resources, and enable quicker interventions.

AI, or more specifically in this study's case, its subset Machine Learning, offers powerful tools for analyzing textual data. ML techniques such as Natural Language Processing (NLP) allow computers to understand and interpret human language, making them ideal for evaluating text-based surveys [3] .

Tietoevry [4], the partner company for this degree project, has expressed an interest in evaluating various ML techniques to detect bullying in text surveys. This includes more traditional methods like Logistic Regression, Naive Bayes, and SVM, but also more complex and modern approaches such as CNN and LLMs.

By leveraging our established partnership with Tietoevry and the Friends Foundation, we get unique access to their dataset. With this dataset, we have a good foundation for training and evaluating the different ML models for this task.

1.2 Related work

Last year, a group of students from Linnaeus University did a similar degree project together in collaboration with TietoEvry. Their objective was to detect possible instances of bullying using Unsupervised Graph Machine Learning in a dataset obtained from Friends. The dataset was structured as a graph database which was created utilizing Neo4j, a graph database management system. Their methodology included the use of similarity algorithms and community detection algorithms,

among other techniques, to identify patterns or clusters indicative of bullying behavior. The findings of their study suggest that Graph Machine Learning can effectively identify potential bullying scenarios. They achieved their best results with the Louvain algorithm, attaining a 78.3% overlap in identifying at-risk students, closely aligning with assessments made by a domain expert [5].

Another study investigates the use of various supervised machine learning algorithms for text categorization which is essential in managing textual information systems. By utilizing a comparative analysis, the study evaluates the effectiveness of Logistic Regression, SVM, Naive Bayes, Random Forest, and AdaBoost in the classification of text into predefined categories based on various metrics such as accuracy and precision. The results demonstrate that SVM scored the highest accuracy of 96.86% while AdaBoost trailed behind with the lowest accuracy of 79.49%. Additionally, other supervised machine learning algorithms were compared in this study [6].

1.3 Problem formulation

The goal of this study is to explore new approaches for detecting bullying in text surveys, leveraging the Friends dataset. While the previous work done by Christoffer Eid and Olof Enström last year successfully identified some bullying patterns, there is still room for improvements [5]. Instead of using Unsupervised Graph Machine Learning techniques as they did, this study will focus on Supervised Machine Learning. Eventually, the best-performing model will be compared with a locally implemented Retrieval-Augmented Generation (RAG) model.

In addition to the exploration of unsupervised techniques by previous work at LNU and Tietoevry, the second related work in the domain of text classification has demonstrated the potential of various supervised machine learning algorithms in accurately classifying text into predefined categories [6], providing a valuable reference for this study's selection of algorithms. This study's insight will potentially guide our evaluation and fine-tuning of similar algorithms to detect instances of bullying in the Friends dataset.

Research Questions:

- Which classification algorithm, among Logistic Regression, Naive Bayes, SVM, and CNN, performs best for classifying bullying in textual data?
- How does the performance of a locally implemented RAG model using a vectorized database compare to the best-performing supervised machine learning model in detecting instances of bullying in textual data?

The selection of the mentioned classification algorithms is influenced by prior research on text classification techniques. According to [7], these algorithms are all well-suited for handling textual data. They can effectively learn from the features extracted from texts and make informed predictions about whether a new piece of text constitutes bullying. The choice of these algorithms was also inspired by the works of [8] and [6].

To train well-functioning machine learning models, a large amount of data is required. Simpler models might require thousands of instances while other complex models (e.g. image recognition) will require millions [9].

The labeled dataset provided by Friends was manually labeled by them. However, its size, with about 3600 labeled instances, might introduce limitations in the ability to achieve well-performing models. Additionally, since the data for this project must be in Swedish, it is challenging to find more data due to the limited online resources, compared to the more readily available ones for a widely-used language like English.

Regarding the second research question, the plan is to implement a RAG model locally, utilizing a vectorized database alone with an open-source LLM. The reason behind running it locally is that the Friends dataset is confidential, so it cannot be exposed online on third-party LLMs like ChatGPT. However, our supervisor Elias from TietoEvry, suggested that it's possible to create a curated dataset from the original one, containing only examples that don't include any names or sensitive information. This smaller dataset can then be used to prompt GPT-4 via the OpenAI API [10]. This additional approach will be tested to evaluate the efficiency and cost-effectiveness of using GPT-4 in identifying bullying instances, while the primary focus remains on the local implementation of the RAG model with a different LLM.

Since the machines intended for use in this study have limited computational resources, integrating an LLM with RAG could be a resource-intensive process. This limitation influences the choice of the LLM that can be run locally. For instance, a model like GPT-4 is computationally demanding and isn't suitable for the hardware available for this study. The selection of an open-source LLM will therefore be based on the computational demands of the model.

The expected result of this study is to find out how well various Supervised Machine Learning algorithms and a locally implemented RAG model can identify instances of bullying in the Friends dataset. The primary evaluation metric used for comparing the models will be the F3-score of the bullying class.

1.4 Motivation

Bullying can have severe consequences on human psychological health and well-being. It has long-lasting effects that can make someone feel rejected and lonely and some people can even get depressed and anxious because of bullying [11].

Getting bullied at a young age can even have negative impacts on the psychological long after childhood. At a young age, kids are trying to learn about themselves and their roles as well as growing their personalities. If they get bullied it might make it harder for them to trust others, it can make them angrier and feel bad about themselves. Additionally not being able to socialise and create relationships with others at a younger age can make it challenging to do so when they are older. Getting subjected to criticism about who they are or what they do, can cause them to develop a negative self-image and assume that others see them the same way [11].

The motivation for this work is to help Friends with efficient and automated detection of bullying rather than relying on manually reading through the surveys. Hopefully, this work will additionally assist teachers and mental health experts in tackling bullying among young people.

This project will also provide insights into the performance of various AI techniques in tasks similar to those encountered in this project. These insights could be valuable for others undertaking related tasks, not limited to bullying detection. The findings could be relevant for a wide range of language processing applications.

1.5 Milestones

M1	Literature review: Research techniques and tools to be used in the project	April 10
M2	Data collection and preprocessing: Acquire the data and perform necessary preprocessing steps to prepare the data	April 13
M3	Implement and evaluate supervised machine learning algorithms	April 25
M4	Implement and evaluate a RAG model	May 5
M6	Analysis and interpretation of results: analyze the performance of the models to identify most effective approach	May 15
M7	Final report writing	May 22

1.6 Scope/Limitation

In this study, the investigation is focused on exploring and evaluating a subset of selected supervised machine learning algorithms to detect bullying within a Swedish-language text survey dataset provided by Friends.

Due to data confidentiality, all models, except for the smaller GPT-4o evaluation, are run and evaluated locally. This introduces computational limitations which may restrict the scalability of the model training and the LLMs possible to evaluate.

This work is also constrained by the limited size and language specificity of the Friends dataset.

1.7 Target group

This project will particularly interest organizations focused on children's well-being in educational environments, such as schools and the Friends Foundation. It may also interest the machine learning community for its technical aspects. Those working on similar tasks but focusing on different classifications can still gain valuable insights from our findings.

1.8 Outline

The rest of the report consists of the following chapters:

- Chapter 2 (**Theory**): This chapter introduces and explains the theoretical concepts that are fundamental to study.
- Chapter 3 (**Method**): This chapter explains the methodologies used to investigate the problem. It outlines the steps for the controlled experiment and discusses aspects such as validity, reliability, and ethical considerations are discussed.
- Chapter 4 (**Implementation**): This chapter presents and explains the software implementation to address the research questions. It includes key code snippets with explanations and presents figures and a flow chart to illustrate the implementation process.
- Chapter 5 (**Experimental Setup, Results, and Analysis**): This chapter provides an overview of the hardware and software used in this study, followed by a detailed presentation of the results obtained from the experiments. It concludes with an analysis of these results.
- Chapter 6 (**Discussion**): This chapter provides an in-depth analysis of the results and answers the research questions. It compares the findings with related work and discusses the contextual challenges encountered during the study.
- Chapter 7 (**Conclusion**): This chapter offers a summary of the thesis and discusses potential future work that could extend and build upon the findings of this study.

2 Theory

This section outlines the theoretical foundations crucial for understanding machine learning techniques used in bullying detection from text. It covers essential NLP methods, focusing on how data is prepared through preprocessing and feature extraction. The differences between supervised and unsupervised learning are also examined, setting the stage for a deeper discussion of the algorithms used in the analysis.

2.1 NLP

NLP is a branch of AI that focuses on enabling machines to understand and manipulate human natural language. NLP originates from computational linguistics and it combines engineering and computer science to perform useful language-based tasks [12].

NLP is growing rapidly in our everyday lives and is getting applied in various fields such as chat-bots, medicine, and conversational agents such as Alexa and Siri that use NLP to understand user questions and find answers. It can be used for various tasks such as giving answers to questions, having conversations with users, and a wide variety of text classification methods [12].

In order to train the selected supervised machine learning algorithms, two sequential steps are required, these steps are provided by the NLP architecture and are called data preprocessing and feature extraction [12].

2.1.1 Text Preprocessing

Before using the textual data with the selected supervised machine learning algorithms, the text must be preprocessed to turn the text into a more convenient and standard form. Preprocessing consists of NLP techniques such as tokenization, stop word removal, and stemming/lemmatization [7]. There are more techniques such as spelling correction, but only those relevant to this study are covered.

2.1.1.1 Tokenization

One crucial preprocessing step is splitting a character sequence or a sentence into word units, also called *tokens*. This step is called *Tokenization* and it eases the subsequent steps. In addition to tokenizing the text, unwanted characters such as punctuation or semicolons are removed [13]. Here is an example:

```
Input: "Växjö är en fin stad som ligger i Sverige"
Output: [Växjö, är, en, fin, stad, som, ligger, i, Sverige]
```

These tokens are then carried out to the next preprocessing step which is presented in the following section.

2.1.1.2 Stop words removal

In this step the goal is to eliminate words that occur frequently in a language but have little or no meaning, these are referred to as *stop words* [12]. These are removed to reduce noise and improve performance [14]. For this study, there is no need to construct a custom list of stop words, as there are several online resources available that provide lists of Swedish stop words.

Building upon the earlier example, the token that remains after the removal of stop worlds would be:

```
Output: [Växjö, fin, stad, ligger, Sverige]
```

2.1.1.3 Stemming and lemmatization

After the removal of step words, two other text preprocessing techniques, namely stemming and lemmatization are utilized to reduce inflectional forms of words to one base form [15]. These techniques can help improve the accuracy and precision of the machine learning algorithms by reducing the dimensionality [16].

Lemmatization involves identifying that two words have the same root or stem, despite their different appearance. For instance, "eats" and "ate" both derive from the root verb "eat", making "eat" the lemma for both. Lemmatization is the process of mapping these variations back to their root form. Although lemmatization algorithms are complex, they are often preferred over *stemming* which is a simpler method to find the root of a word by truncating the ends of words, such as transforming "eats" and "eating" to "eat". However stemming can be less precise and lead to mistakes, making lemmatization a more accurate method for finding roots of words [7]. An example of stemming error would be that the Swedish word "får" can mean different things, as a verb it means "get" and the lemma will be "få" ("get"), and as a noun the meaning becomes ("sheep") and it remains "får" as the lemma.

There are various stemming algorithms, but they generally operate by removing suffixes from words. This is done by comparing word endings to a predefined list of common suffixes and removing them if they match certain conditions without violating specific rules related to those suffixes [17].

2.1.2 Feature Extraction

You can't feed a machine learning model with text or words. At some point, you need to convert it into numerical representations so that the machine learning model can "understand" and process it. The conversion should aim to preserve useful information and patterns present in the text. This is done in the *Feature Extraction* step. There are a variety of feature extraction techniques that can be used but the ones used and explained in this degree project are *Bag-of-Words (BoW)* and *Term Frequency-Inverse Document Frequency (TF-IDF)*.

2.1.2.1 BoW

The BoW model is a straightforward approach to text representation for machine learning. It works by transforming text data into a numerical format that algorithms can process [12].

Initially, BoW constructs a vocabulary list from all unique words across an entire text corpus without considering the syntax and word order. Each unique word found in the text becomes a feature in the final data representation [12].

Once the vocabulary is established, each piece of text (such as a document or a sentence) is converted into a vector. The length of this vector is equal to the size of the vocabulary [12].

Each element in the vector corresponds to a word in the vocabulary. For a given piece of text, the vector is populated with the count of how many times each vocabulary word appears in that text. If a word in the vocabulary is present in the text, the corresponding vector element is the count of that word; if it is absent, the element is zero [12].

For instance, in the documents "blue car and blue window" and "red car", with a vocabulary of blue, car, and, window, red, the vectors would be [2, 1, 1, 1, 0] and |[0, 1, 0, 0, 1]|. Here, the number 2 in the first vector indicates that the word "blue" appears twice in the first document, while the 0s denote the absence of the corresponding words [12].

Its main limitations are that it doesn't account for the order of words, and the vectors can become quite large, which can be computationally expensive [12].

2.1.2.2 TF-IDF

The TF-IDF vectorization measures how important a word is to a document in a corpus. The TF-IDF score increases if a word appears frequently in a document but decreases if it appears in many documents in the corpus, balancing relevance and commonality [18].

Scikit-learn's `TfidfVectorizer` uses the following approach [19]:

Term Frequency (TF): The term frequency $tf(t, d)$ of a term t in a document d is the ratio of the term's occurrences to the total number of terms in the document.

$$tf(t, d) = \frac{\text{count of term } t \text{ in document } d}{\text{total terms in document } d}$$

Inverse Document Frequency (IDF): The inverse document frequency $idf(t, D)$ reduces the weight of terms that appear in many documents. It is calculated as:

$$idf(t, D) = \log\left(\frac{1 + \text{total number of documents } D}{1 + \text{documents with term } t}\right) + 1$$

TF-IDF Score: The TF-IDF score for a term t in a document d is the product of its TF and IDF values:

$$\text{TF-IDF}(t, d) = tf(t, d) \times idf(t, D)$$

L2 Normalization: To ensure that documents of different lengths can be compared on the same scale, TF-IDF vectors are normalized using the L2 norm, making the vector length 1:

1. Calculate the L2 norm of the TF-IDF vector:

$$\|\mathbf{v}_d\|_2 = \sqrt{\sum_{t \in d} (\text{TF-IDF}(t, d))^2}$$

2. Normalize each TF-IDF score:

$$\text{TF-IDF}'(t, d) = \frac{\text{TF-IDF}(t, d)}{\|\mathbf{v}_d\|_2}$$

The normalized vector has a length of 1, allowing fair comparison between documents.

2.2 Supervised vs Unsupervised

Machine learning systems can be categorized based on the level and kind of supervision required during the training. There are primarily four major types of learning categories: supervised learning, unsupervised learning, semisupervised learning, and reinforcement learning [9].

2.2.1 Supervised Machine Learning

Supervised Machine Learning trains on a labeled dataset. This means that for every instance in the dataset (in the case of this study an instance is a text that is to be classified), there is a label attached to it ("1" for bullying and "0" for not bullying").

2.2.2 Unsupervised Machine Learning

Unsupervised Machine Learning uses unlabeled data. The goal is to find patterns or structures in the data without guidance on what outcomes to predict. The model learns from the data itself, and this type of learning is particularly useful for exploratory analysis or discovering hidden patterns in the data [9].

While the previous project by Eid and Enström at Linnaeus University employed unsupervised machine learning techniques to detect bullying patterns within a graph database [5], this study aims to explore and validate the effectiveness of supervised learning methods to detect bullying in the labeled dataset from Friends.

2.3 Supervised learning algorithms

In the context of this study, where the objective is to classify textual data as either "bullying" or "not bullying", *classification* serves as a crucial technique. At its core, classification involves assigning a category to an input based on specific features extracted from the data. This process is fundamental in human decision-making and machine learning applications, which include tasks ranging from recognizing objects in images to analyzing sentiments in textual content [7].

For this particular study, binary classification is employed. This approach categorizes texts into one of two discrete classes: bullying or not bullying, similar to how basic sentiment analysis classifies text reviews as positive or negative based on the words they contain [7].

Supervised machine learning is the most frequently used learning method in language processing for text classification [7]. The Friends dataset consists of labeled data, each piece of text in the dataset is an observation that Friends has labeled as 1 in the case of "bullying" or 0 in the case of "not bullying". By training different models using the training data, the model will be able to make predictions on new, unseen data and classify it as "bullying" or not "bullying".

The methods chosen for this classification task are Logistic Regression, Naive Bayes, SVM, and CNN. These methods are all well-suited for handling textual data. They can effectively learn from the features extracted from texts and make informed predictions about whether a new piece of text constitutes bullying [7].

2.3.1 Naive Bayes

Bayes' theorem is the main method for calculating the probability of some event when other certain probabilities are known [20].

The formula is:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} \tag{1}$$

where:

- $P(x|y)$ is the posterior probability of class x given predictor y.
- $P(y|x)$ is the likelihood, which is the probability of the predictor given class x.
- $P(x)$ is the prior probability of class x.
- $P(y)$ is the prior probability of the predictor.

In machine learning, the naive Bayes classifier is an application of Bayes' theorem that is used specifically for classification tasks [20]. Simply a naive Bayes classifier is based on:

$$P(y|x_1, \ldots, x_j) = \frac{P(x_1, \ldots, x_j|y)P(y)}{P(x_1, \ldots, x_j)} \tag{2}$$

where:

- $P(y|x_1, \ldots, x_j)$ is the posterior probability that an observation belongs to class y give the values of its j features $(x_1, \ldots, x_j)$.
- $P(x_1, \ldots, x_j|y)$ is the likelihood of observing the values for the features $(x_1, \ldots, x_j)$ give a specific class y.
- $P(y)$ is the prior probability of class y before any evidence.
- $(x_1, \ldots, x_j)$ is called the marginal probability of features x.

The Naive Bayes classifier treats a text document as a collection of words without regard to their order in the sentence. This model focuses on the frequency of each word within the document. For example, in a text containing phrases like "I love this movie" and "I would recommend this movie", rather than maintaining the sequential order of words, the model would simply record the occurrences of each word as shown in Table 2.1 [7].

Word	**Frequency**
I	2
love	1
this	2
movie	2
would	1
recommend	1

Table 2.1: Word frequencies in the sentences mentioned above

In naive Bayes, the posterior probabilities of observation for each possible class are compared. Since the marginal probability remains constant when comparing, the focus relies on comparing the numerators of the posterior probabilities for each class. By selecting the class with the highest numerator in the posterior probability, the predicted class is determined, which is denoted $\hat{y}$, for the given observation [20].

Naive Bayes classifiers assume that all features are independent of each other given the class, this is why it's considered "naive". This assumption makes the computations simple and is key to the efficiency of the algorithm. Although this assumption can be unrealistic because features in real-world scenarios are often dependent on each other, naive Bayes classifiers still perform very well in many text classification tasks [20].

In naive Bayes classifiers, an assumption about the distribution of the input features given the class is required. Depending on the nature of the data, various distributions can be assumed. One type is *multinomial* distribution which is used for discrete data and is useful in text classification tasks [20]. This is due to its suitability for classification with discrete features, such as word counts. The multinomial distribution typically requires integer counts, which is why the BoW approach was selected. Although fractional counts like TF-IDF could potentially be effective, this study will utilize the BoW vectorizer for the multinomial classifier [21].

To classify bullying using a naive Bayes classifier, the probability that a document x contains bullying can be calculated using Bayes' theorem:

$$P(Bullying|x) = \frac{p(x|Bullying) * P(Bullying)}{P(x)} \tag{3}$$

where x is a vector representation of the words in a document, where each element w_ii in the vector $x = [w1, w2, w3, w4 \ldots, wn]$ corresponds to a feature derived from the document, typically the frequency of words.

A common challenge encountered when working with Naive Bayes classification is

the zero probability issue, where unseen words in the training dataset lead to a probability of zero during predictions, which affects the classifier's performance. To address this issue, smoothing techniques are used, specifically Laplace and Lidstone smoothing, which adjust the calculation of probabilities and prevent zero probability estimates [22].

2.3.2 Logistic Regression

Logistic regression is a statistical model that is used for classification problems, where the goal is to predict the likelihood that a specific instance falls into a given category. For example, determining if an email is spam or not, or if a text document consists of bullying content or not [9].

$$\hat{p} = h_\theta(x) = \sigma(\theta^T \cdot x) \tag{4}$$

It takes the weighted sum $\theta^T \cdot x$ of the input features(where θ represents the parameter vector, including the bias term and weights, and x represents an instance's feature vector) and runs it through the logistic function, visualised in Figure 2.1 [9].

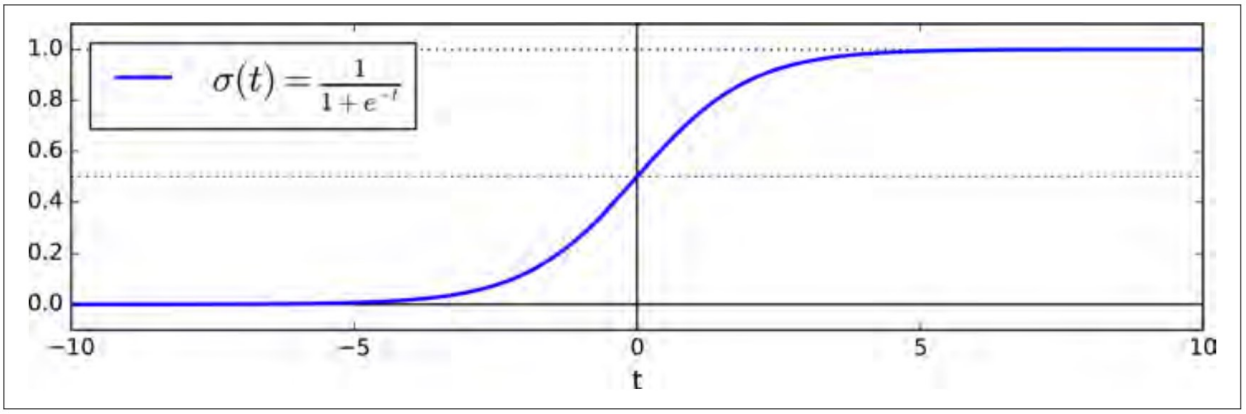

Figure 2.1: Logistic function.

The predicted class label, $\hat{y}$, is then determined based on the probability estimate:

$$\hat{y} = \begin{cases} 0 & \text{if } \hat{p} < 0.5 \\ 1 & \text{if } \hat{p} \geq 0.5 \end{cases} \tag{5}$$

So it $\hat{p}$ is less than 0.5, it will be classified as the negative class, or for example non-bullying class, otherwise the positive class.

To boost the performance of the logistic regression model, hyperparameters can be tuned using grid search together with cross-validation (CV). Some of the hyperparameters are:

- **Inverse of regularization strength (C)**: Controls the amount of overfitting. Smaller values of C cause stronger regularization, while large values might lead to overfitting.
- **Solver**: This parameter specifies the algorithm to use for the optimization. 'liblinear' is good for small datasets while 'saga' and 'lbfgs' are better suited for larger ones.
- **Maximum number of iterations**: Determines the number of iterations for the solver to converge.
- **Penalty**: Specifies the norm used in penalization. The common penalties are 'l1' and 'l2' [23].

2.3.3 SVM

The concept of hyperplanes plays a crucial role in understanding SVM. Hyperplanes are a geometrical structure that exists as a subspace within a dimension that is one less than the dimension of the space in which it is defined i.e. the ambient space. To put it simply, in one-dimensional space, a hyperplane is a dot. In a two-dimensional space, a hyperplane is represented as a line, in a three-dimensional space, a hyperplane corresponds to a plane. In a formal sense, a hyperplane can be defined as an $n-1$ subspace in an n-dimensional space [20].

To perform classification with SVM, the idea is to find the hyperplane that maximizes the margin between the nearest data points belonging to opposing classes. For example, in a two-dimensional space, where there are two classes, the hyperplane can be visualized as the widest straight line with margins that separate the two classes [20]. The number of features in the training data will determine the type of the hyperplane. Given the possibility of finding multiple hyperplanes for class differentiation, finding the hyperplane that maximizes the margin between points increases the algorithm's ability to find the optimal decision boundary between classes. Therefore, this enhancement enables the algorithm to exhibit strong generalization capabilities against new data and perform accurate classifications. In Figure 2.2 the lines that are neighboring to the optimal hyperplane are called *support vectors* as they traverse through the data points that help find the maximal margin [24]

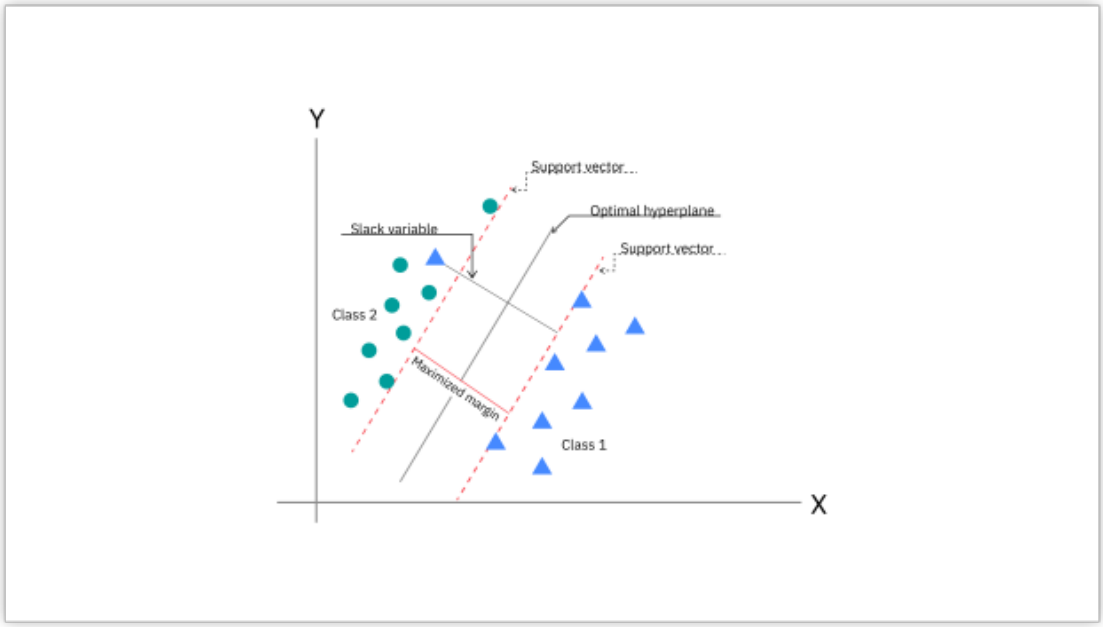

Figure 2.2: SVM Classification Boundary and Margins [24]

SVM can work with both linear and nonlinear classification tasks. Yet, when it's possible to separate the data with a line, *kernel functions* are utilized in a space with higher dimensions to enable linear separability. There are different types of kernel functions that can be used for classification tasks, some common kernel functions are *polynomial kernel*, *Radial basis function kernel (RBF kernel)* and *Sigmoid kernel* [24].

To boost the performance of an SVM model, hyperparameters can be tuned using grid search together with CV. Some of the hyperparameters are:

- **Regularization Parameter (C):** This parameter controls the trade-off between achieving a wider margin and minimizing classification errors. A smaller C value leads to a wider margin but allows misclassification. However, a larger C strives for a narrower margin with fewer misclassifications [20].
- **Kernel:** The choice of the kernel can impact the decision boundary that the SVM constructs [9].
- **Gamma value:** the gamma parameter γ is for non-linear hyperplanes. The higher the gamma values, the more it tries to exactly fit the training data. On the other hand, lower gamma values can cause the model to underfit [21].

When applying feature extraction, the TF-IDF vectorizer processes the textual data and calculates a score for each word in each document. The result will be a matrix where each row is a vector that represents a document, and each dimension of the vector corresponds to a word's TF-IDF score. This matrix will be used as input to train an SVM model [21]. The labels or targets for the SVM model will correspond to the classification outputs (e.g. 1 for "bullying or 0 for "not bullying")

2.3.4 CNN

CNN is designed to extract features from data, traditionally and most commonly images. However, it has also shown significant results in areas like voice recognition and text classification [25],[9]. The CNN architecture consists of two main layers: the **convolution layer** and the **pooling layer**. The convolution layer detects important features in the data and the pooling layer reduces the size of the feature map while retaining the most important information [26],[27]. Instead of using two-dimensional convolution layers, which is the case for the traditional image-recognition use cases, the convolution layers are one-dimensional for text classification.

The output of the pooling layer is fed into a fully connected neural network. This part is similar to a traditional feed-forward neural network, where the output is passed through one or more hidden layers and an output layer to produce the final result [26].

For text classification using CNNs, the input data is numerical representations of the texts. These can be results from *one-hot encodings* or trained *word embeddings*.

When using word embeddings, which is the case in this project, each word is represented as a dense vector of numbers of typically hundreds of dimensions. These vectors capture semantic relationships between words. For example, the word embeddings of the words 'man' and 'king' or 'women' and 'queen' will be closely related(the vectors will have a similar direction in the high-dimensional vector space).

2.4 Overfitting and Underfitting

Overfitting occurs when a model is trained too well, resulting in a significantly better performance on the training data compared to the test data [28]. Instead of learning fundamental patterns in the data, the model ends up capturing noise and intricate details present in the training data [29]. Therefore, the model will lack sufficient generalization capability, leading to poor performance on new unseen data [29]. Overfitting is a significant concern in machine learning and can be prevented through various approaches. To assess if the model is overfitted, the data can be split into a training set and a test set, where the model is trained on the training set and evaluated on the test set [28].

On the other hand, underfitting arises when the model fails to capture the variance in the data. An underfitted model is too simple, resulting in low variance and high bias [29], [30]. In both overfitting and underfitting, the model fails to capture the dominant trend present in the training set, resulting in poor generalization to new unseen data. However, unlike overfitting, underfitted models have high bias and lower variance in their predictions. This highlights the bias-variance trade-off, which occurs when an underfitted model alters into an overfitted state. When the model learns, the bias decreases, but there is a potential increase in variance as it becomes overfitted. The objective when training a model is to find the optimal "sweet spot" where the model can effectively capture the dominant trend and apply

it to new unseen data.

Figure 2.3 represents an illustration of three different model states.

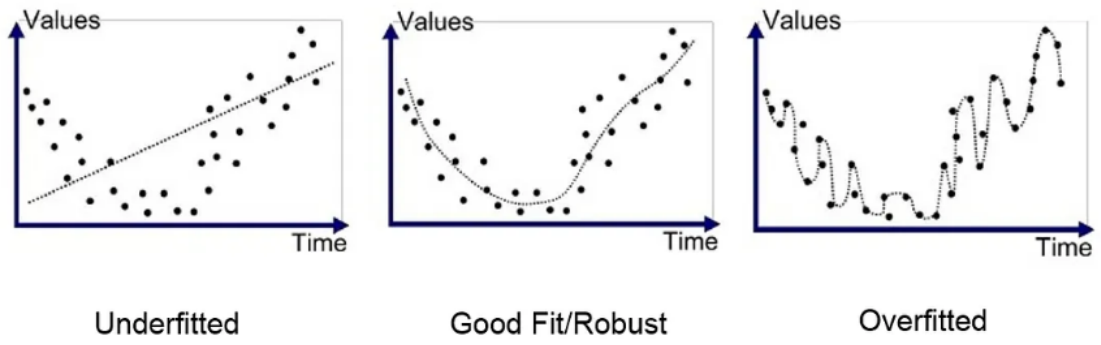

Figure 2.3: A graphical illustration of an underfitted model, ideally fitted model, and overfitted model [31]

One common technique to avoid overfitting is regularization, it can provide significant assistance in addressing this issue. It involves applying a penalty on the input parameters that have large coefficients, effectively restricting the amount of variability in the model [32].

Additionally, increasing the amount of training data is an effective approach to mitigate overfitting [33]. When dealing with a large dataset, it is possible to incorporate a validation set alongside the standard training and testing sets. For instance, if the split ratio is 80:20, the validation set can be formed by selecting 20 percent from the 80 percent training set to evaluate the model. If the dataset is smaller, a suitable method is CV [30], [34] which is presented in 2.5.

2.5 CV

CV is a technique to test and evaluate the effectiveness of machine learning models. Additionally, CV can be used as a re-sampling procedure that can be utilized to evaluate a model if the data is limited [35].

One common CV approach is k-folds CV which involves splitting the data into k-folds of equal size. One of the folds serves as the test set, also known as the validation set, while the remaining folds are used to train the model. This process is repeated until each fold has been used as a test set. After each iteration, a score is recorded, and once all iterations are completed, these scores are averaged to evaluate the overall performance of the model [32].

2.6 Evaluation Metrics

In this subsection, various methods for evaluating the performance of a model will be discussed. The evaluation can be represented using a confusion matrix, while

metrics such as accuracy, precision, recall, and F1 Score are commonly utilized to determine the effectiveness of different classification models [36]–[38].

2.6.1 Confusion Matrix

The primary objective of a confusion matrix is to provide a visual representation of how the input data is classified by the classifier model. It categorizes the classification results into 4 different categories that measure the correctness of the model, as shown in Figure 2.4. The first two categories, true positives (TP) and true negatives (TN) indicate the accurate classification of the input data. The third one, false positives (FP), occurs when the input data is considered negative but is classified as positive, and false negatives (FN) mean the data was wrongly classified as negative when it actually is positive [39].

		True class	
		Positive	Negative
Predicted class	Positive	True Positive Count (TP)	False Positive Count (FP)
	Negative	False Negative Count (FN)	True Negative Count (TN)

Figure 2.4: Confusion Matrix: Predicted vs true classes correlation [40]

2.6.2 Accuracy

Accuracy measures the proportion of correct predictions i.e. true positives and true negatives among the total number of predictions made by the model. Accuracy can be calculated using equation 6 [41].

$$Accuracy = \frac{TP + FP}{TP + TN + FP + FN} \tag{6}$$

2.6.3 Precision

Precision measures the proportion of true positive predictions among all positive predictions made by the model. Higher precision means fewer false hits. In other words, precision answers the question: "Out of all positively predicted instances, how many were actually positive?". Precision can be calculated using equation 7 [41].

$$Precision = \frac{TP}{TP + FP} * 100\% \tag{7}$$

While precision provides insight into the accuracy of positive predictions, it doesn't reflect the model's overall accuracy. To better assess the completeness of the model's positive predictions, the recall metric is used [41].

2.6.4 Recall

Recall, also known as Sensitivity or True Positive Rate, measures the proportion of true positive predictions among all actual positive instances. To put it simply, recall answers the question: "Out of all actual positive instances, how many were correctly predicted as positive?". Recall can be calculated using equation 8 [41].

$$Recall = \frac{TP}{TP + FN} * 100\% \tag{8}$$

2.6.5 F1-score

Precision and recall can exhibit an inverse relationship in certain cases. When the model increases its recall by identifying more actual instances (TP), the model may also incorrectly label more non-instances (FP) in the process, which in turn reduces precision [42]. The F1-score is used to balance this trade-off between precision and recall to reflect the model's overall class-wise accuracy. The F1-score can be calculated using equation 9 [43].

$$\text{F1-score} = 2 * \frac{precision * recall}{precsion + recall} \tag{9}$$

2.6.6 F_β-score

In certain situations, it isn't desired to value recall and accuracy equally, as in the standard F1-score. For instance, in medical screenings such as cancer detection, a high recall is desired, even if it is at the cost of low precision. A lot of false positives is acceptable as long as you don't miss a lot of actual cancer cases.

$$F_\beta\text{-score} = \frac{(1+\beta^2)\cdot(\text{precision}\cdot\text{recall})}{(\beta^2\cdot\text{precision})+\text{recall}} \tag{10}$$

F_β-score is used to favor either recall or precision. A high value of β means recall will be prioritized over precision. For example β=2 will weigh recall twice as much as precision [44], [45].

There is no single optimal β value for a given task. The decision will be up to the developers to decide, based on the requirements of the task.

2.7 LLMs

LLMs are advanced neural network architectures trained on extensive textual data to perform a variety of tasks, one being NLP. These models, built using the *transformer architecture*, learn to "understand" and generate language by processing vast amounts of data. The data is usually text, but it can also be for example images and audio. This extensive training allows them to develop an "understanding" of language and the world, giving them the ability to perform all sorts of complex tasks [7].

Transformers, the backbone of LLMs, utilize *self-attention* mechanisms to analyze words in relation to each other across large text spans.

They operate by predicting the next word in a sequence given the previous words. This capability also extends to *prompting* where models perform tasks directed by specific instructions in natural language, a method that simulates understanding and responding to user questions.

2.7.1 Models

There are a lot of LLMs available today, ranging in size and performance. The *Llama 3* model, released April 18, 2024, is a state-of-the-art open-source LLM from Meta [46], which comes in different sizes of 8 billion and 70 billion parameters. The amount of parameters correlates with how computationally intensive the model is to run, as well as how "smart" the model is, measured in various benchmarks. Its open-source nature allows it to be run locally.

Another famous LLM is the *GPT-4o* model, released may 13, 2024, by *OpenAI* [47]. OpenAI hasn't shared how many parameters GPT-4o has but the benchmarks show it's among the top-performing models in various tasks. It's closed-source, so even if you had the computational resources to run it locally, you currently can't. Instead, you have to rely on the OpenAI API [10].

2.7.2 Temperature

When prompting an LLM, a `temperature` variable can be adjusted to influence the randomness of predictions made by the models. The variable can be set in the range of 0.0-2.0, where a higher temperature (e.g. 1.0 or more) makes the output more random and creative. A lower temperature (e.g. 0.2) makes the output more focused and deterministic [48].

2.8 RAG

One issue with LLMs is the lack of reliability in the responses. It just gives you an answer with the highest probability, based on the data it was trained on. RAG is a technique that helps with increasing reliability. It achieves this by combining a given prompt with relevant information retrieved from a vector database. This allows the LLM to give a more accurate response because it has the context of actual data from the database [49].

To create the vector database, you store the desired data as numerical representations using embeddings. When a user asks a question, the system embeds the question into a numerical representation, finds the most similar information in the database by calculating vector distances, and retrieves it. The retrieved data is then added to the LLM prompt, enabling the model to provide answers that are accurate, up-to-date, and sourced from authoritative references [49].

3 Method

3.1 Research Project

In this research question, the goal is to try different methods for detecting bullying in the Friends dataset by employing supervised machine learning algorithms. In our comparative analysis, we want to implement and evaluate several algorithms to identify the most effective one in detecting bullying in the dataset.

For the second research question, the study explores the efficacy of a RAG model and compares how well it performs on the dataset against the best-performing supervised model.

Addressing the two research questions of this study will be based on **controlled experiments** which is explained in the following section.

3.2 Method

3.2.1 Controlled Experiment

In a controlled experiment there are two types of variables:

- *Independent variable*s: are the variables that can be manipulated or changed in the experiment. These variables have a direct effect on the dependent variable.
- *Dependent variables*: are the factors that will be observed or measured as the independent variables change.

In the context of this study, the dependent variable is the model performance metrics such as accuracy, precision, recall, F1-score, and F3-score, which will help identify the best model for detecting bullying in the dataset. These will be different based on the supervised machine learning model that is chosen. Therefore the independent variable is the chosen model among Logistic Regression, Naive Bayes, SVM, CNN, and the RAG model with Llama 3. However, due to GPT-4o's cost, closed-source nature, and inability to handle sensitive data, It isn't a viable option for this particular application.

3.2.2 The Controlled Experiment Setup

- **Preparing the data**:
  - **Text Preprocessing**: Apply text preprocessing techniques such as tokenization, stop words removal, and lemmatization to clean and normalize the data.

  - **Feature Extraction**: Implement a feature extraction method, including BoW and TF-IDF to transform the textual data into a numerical format suitable for the machine learning models. Additionally, for the CNN model, pre-trained word embeddings from FastText were used.

- **Model Implementation and Comparative Analysis**
  - **Model training**: Implement the previously mentioned classification algorithms using the dataset with the help of machine learning libraries. The choice of these algorithms was guided by prior research on text classification techniques, as detailed in the problem formulation in section 1.3.
  - **Hyperparameter tuning**: Adjust the model hyperparameters through a systematic process using GirdSearchCV to find the best settings for each model. For Multinomial Naive Bayes, various values of the smoothing parameter were evaluated. Logistic Regression involved tuning regularization value C, solver type, maximum iterations, and penalty type. For SVM, the grid search focused on optimizing hyperparameters C, gamma, and kernel type.
  - **Controlled testing**: Evaluate each model in CV by measuring and recording performance metrics such as accuracy, precision, recall, F1-score, and F3-score for each model.
  - **Selecting a model**: The model that provides the best performance based on the metrics will be identified as the best supervised learning model and will later be compared to the offline LLM.

- **LLM Selection and RAG implementation**: An appropriate offline LLM from the Hugging Face platforms will be downloaded to be utilized with a RAG implementation [50]. The model will then be tested and evaluated using the same metrics used for the traditional method.

- **Testing GPT-4o**: Following a suggestion from our supervisor Elias, OpenAI's newest model was prompted with anonymized entries to evaluate its efficiency and cost-effectiveness.

- **Final Comparative Analysis**: Compare the performance of the best traditional model against the implemented RAG model as well as analyze and document differences in the performance.

- **Presentation**: summarize the findings and use visual tools like graphs and charts to present the results.

The results gathered from the models will be the foundation for a proposed solution, i.e. the most efficient model in detecting bullying in the Friends dataset.

3.3 Reliability and Validity

3.3.1 Reliability

No data collection process was conducted in this study; instead, the data was pre-collected by Friends, and TietoEvry had permission to share it with the researchers.

If this project were to be replicated using the same dataset and following the same methodology, the results should be very similar to those obtained in this project. Simpler models, such as Logistic Regression, should yield identical results when reproduced with the same dataset, while more complex models like CNNs and LLMs, might introduce slight variations due to their inherent randomness. These variations, however, are not significant enough to be critical for the overall conclusions of the study.

3.3.2 Validity

To build models that accurately identify bullying, it's crucial that the dataset contains genuine examples of bullying. One could imagine a situation where the models score well on tests like the F1 score, but the dataset doesn't truly reflect real-world bullying. This would indicate that the data isn't accurately capturing what it aims to measure. Fortunately, the dataset provided by the Friends Foundation is of high quality, which enhances the validity of the research, ensuring that the study measures what it is supposed to measure.

3.4 Ethical considerations

In this study, ethical considerations are given significant attention as the models are trained on a confidential dataset that consists of sensitive data such as bullying and mental health within a school context. Both contributors to this work have signed a paper from TietoEvry to not talk about the data, not share it, not expose it, and keep it safe.

3.5 Team Work

To ensure an equitable distribution of work, we made a collaborative plan. Although the majority of work will be conducted jointly, and the responsibilities for writing specific sections were clearly defined.

On the next page is an overview of the responsibilities for each chapter presented in Table 3.2.

Chapter	Section	Owner & responsible
Chapter 1	1 | 1.1 | 1.3 | 1.5 | 1.7	Ludvig
	1.2 | 1.4 | 1.6 | 1.8	Seif
Chapter 2	2.1 | 2.1.1 | 2.1.1.1 | 2.1.1.2 | 2.1.1.3 | 2.2 2.2.2 | 2.3 | 2.3.1 | 2.3.3 | 2.4 | 2.5 | 2.6 | 2.6.1 | 2.6.2 | 2.6.3 | 2.6.4 | 2.6.5	Seif
	2.2.1 | 2.1.2 | 2.1.2.2 | 2.1.2.1 | 2.6.6 | 2.3.2 | 2.3.4 | 2.7 | 2.8	Ludvig
Chapter 3	3.1 | 3.4 | 3.5	Seif
	3.2 | 3.3	Ludvig
Chapter 4	4.1 | 4.2 | 4.4 | 4.4.1 | 4.6	Seif
	4.3 | 4.4.2 | 4.5	Ludvig
Chapter 5	5.4 | 5.5 | 5.8	Seif
	5.1 | 5.2 | 5.3 | 5.6 | 5.7	Ludvig
Chapter 6	6.1 | 6.2 | 6.3	Seif
		Ludvig
Chapter 7	7	Seif
	7.1	Ludvig

Table 3.2: Team work

For the implementation part, Seif will handle the text preprocessing step as well as implementing Naive Bayes and SVM. Additionally, Seif will be responsible for trying out GPT-4o. On the other hand, Ludvig will work on the feature extraction as well as implementing Logistic Regression and CNN. Additionally, Ludvig will be responsible for the implementation and evaluation of the RAG model.

4 Implementation

This chapter explains the steps for preprocessing the data and implementing the different supervised machine learning models as well as the RAG model. Additionally, it describes the process of creating a curated, anonymized dataset and using it to prompt GPT-4 via the OpenAI API. Figure 4.5 presents the process flow of the implementation of the supervised learning models.

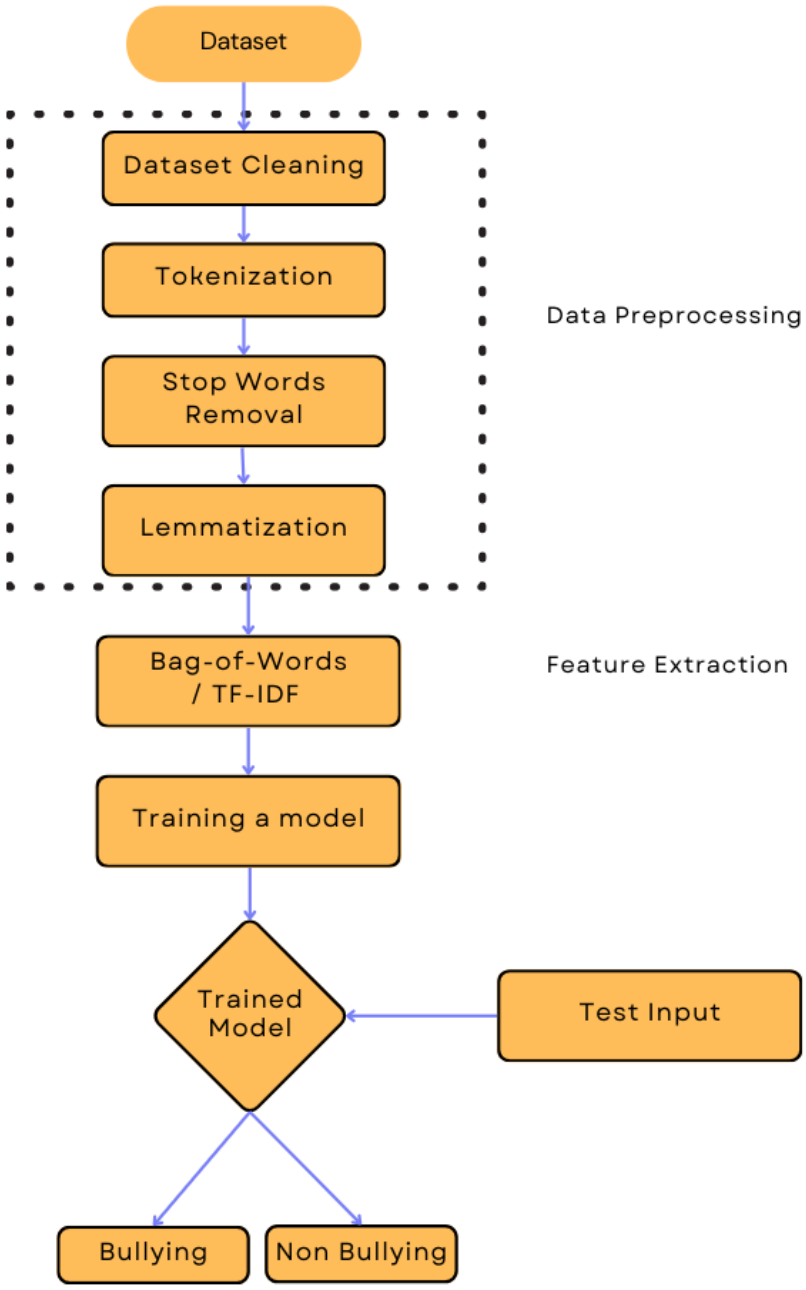

Figure 4.5: Text Classification Pipeline for Bullying Detection

4.1 The Friends dataset

The Friends dataset contains two primary columns, "text" and "label text." The "text" column features responses gathered from surveys, and the "label text" column categorizes each response as ‘0‘ for non-bullying instances or ‘1‘ for texts indicative of bullying, these were manually labeled by Friends. Due to the confidential nature of the data, only examples that don’t contain any names of sensitive information can be presented as examples. Table 4.3 presents an overview of the dataset, and Table 4.4 presents some anonymized entries.

Total rows	3839
Bullying instances	634
Non-bullying instances	3110
Number of rows with NaN label	95
Number of rows without alphabetic characters	34

Table 4.3: Overview of the Friends dataset

Text	**Label**
BÄTTRE SKOLMAT!!!!!!	0
alla kollar på mig	0
känner mig trygg i hela skolan	0
jag är mobbad	1
de slår mig	1
misshandel	1

Table 4.4: Some anonymized examples from the Friends dataset

According to Table 4.3, the dataset exhibits some inconsistencies, including entries without labels and others lacking alphabetical characters, so cleaning the data before training is necessary.

4.2 Text Preprocessing

In this subchapter, a thorough description of the different preprocessing steps is provided. These steps include cleaning the dataset, tokenization, stop word removal, and lemmatization.

4.2.1 Removal of non-essential value

The initial step in text preprocessing involves cleaning the dataset. This process entails removing entries that lack labels or don’t contain any alphabetical characters.

```
data = pd.read_csv('./data/Friends_data_only_user_annotation.csv')

# Drop rows with NaN in 'text' or 'label text' columns
data.dropna(subset=['label text'], inplace=True)

```

```
# a function to check for alphabetic characters, including Swedish
def contains_alpha(text):
    return bool(re.search(r'[a-zA-ZåääöÅÄÖ]', text))

# Remove rows without alphabetic characters in the 'text' column
data = data[data['text'].apply(contains_alpha)]
```

First, the dataset is loaded using pandas [51]. The `dropna` method is employed to eliminate rows where the 'label text' column contains NaN values. Additionally, a function named `contains_alpha` is defined. This function checks whether a text string contains any alphabetical characters, including Swedish letters. It uses a regular expression [52] to search for any occurrences of these characters. If none are found, the function returns `False`. Rows that don't meet the criteria set by the function are then removed from the dataset.

Some examples of text entries that don't contain any alphabetical characters include `[...., -||-, :), ????]`. These entries are removed from the dataset as they don't contribute to the analysis of text for bullying. However, it is worth mentioning that this cleaning approach doesn't filter out text entries such as `[:D, :P, ( o )Y( o )]`.

These entries are retained in the dataset because they include alphabetic characters. This decision to use such an approach is to avoid removing text that could still be relevant to the study, such as entries with mixed content like `???bullied??`. This method ensures preserving texts that, despite their unconventional format, may contain indications of bullying.

An overview of the dataset after the cleaning step is presented in Table 4.5.

Total rows	3710
Bullying instances	634
Non-bullying instances	3076
Number of rows with NaN label	0
Number of rows without alphabetic characters	0

Table 4.5: Overview of the Friends dataset after cleaning

4.2.2 Tokenization, Lemmatization and Stop Words Removal

To perform NLP tasks with Swedish texts, libraries such as *Lemmy* [53], *spacCy* [54], and *Stanza* [55] were investigated. Stanza was selected for this study to perform tokenization and lemmatization. This choice is based on inspiration from a similar study conducted at KTH Royal Institute of Technology, which involved the categorization of Swedish emails using supervised machine learning [8].

Stanza is a robust Python package for NLP tasks that offers a range of tools, which can be used in a pipeline to process various human languages, including tokenization and lemmatization. To use Stanza with Swedish text, the specific language model has to be downloaded [56]. The model utilized the Stockholm-Umeå Corpus (SUC) version 3.0, which is a collection of Swedish texts available since 2012, consisting

of seven million words in total. The corpus is balanced, encompassing diverse text types and stylistic levels [57].

Throughout the process, an investigation was conducted to ascertain the most suitable technique for the Swedish language, either stemming or lemmatization. Eventually after trying out both techniques, it was concluded that lemmatization was the more appropriate choice, due to its ability to reduce words to their lemma form more accurately.

For stop words removal, instead of creating a new list of stop words, accessible lists available online were utilized [58], [59]. This step aims to eliminate frequently occurring words that have little or no meaning. These lists were concatenated into a text file and loaded into a Python list.

The integrated approach to text preprocessing which involves tokenization, lemmatization, and stop-word removal is encapsulated in the function `preprocess_text`.

```
# Ensure the Swedish model is downloaded
stanza.download('sv', verbose=False)


# Initialize the Stanza pipeline for Swedish, without the mwt processor
nlp = stanza.Pipeline(lang='sv', processors='tokenize,lemma', verbose=False, use_gpu=True)

def preprocess_texts(texts, stop_words):
    processed_texts = []
    for text in texts:
        doc = nlp(text)
        tokens = []

        for sentence in doc.sentences:
            for word in sentence.words:

                if word.text.lower() not in stop_words and len(word.text) > 2:
                    tokens.append(word.lemma)

        processed_texts.append(" ".join(tokens))

    return processed_texts
```

After downloading the language model for Swedish, the Stanza pipeline is initialized with two specific processors, the `tokenize` processor for dividing text into tokens or words and the `lemma` processor for reducing words to their lemma. The `use_gpu=True` parameters enable the use of a GPU if available which can speed up the processing. For this study, CUDA support was enabled on an NVIDIA GPU to ensure efficient processing [60].

The `preprocess_texts` function processes each text entry through the Stanza pipeline, where each text is split into sentences and further into words. Each word is then checked against the list of stop words that were loaded and is removed if it is a stop word. Additionally, words with less than three characters are removed because they represent no meaning.

Each valid word's lemma is collected, and the processed text is reconstructed from these lemmas. This approach ensures that the text is distilled down to its most informative components, making it suitable for further analysis in machine learning models. Here is an example of some input text and output processed text:

```
Input: ["Biblioteket har böcker man kan låna.",
        "Vädret var perfekt för en lång promenad idag.",
        "Detta är ett meddelande som innehåller mobning",
        "bo"]

Output: [biblioteket böcker låna,
        vädret perfekt lång promenad,
        meddelande innehåller mobning]
```

The `preprocess_texts` function is used to process the text entries of the original dataset. The result is a new dataset with preprocessed text, which is then carried out to the feature extraction step where it's converted to numerical values before proceeding to training the machine learning models. Additionally, an alternative preprocessed dataset was created without removing stop words. This modification was made because of the realization that it is not always beneficial to remove stop words. Therefore, both approaches, with and without stop words, will be tried to address the differences between removing and keeping such words.

4.3 Feature Extraction

In the implementations of Logistic Regression and SVM, TF-IDF from *scikit-learn* was used for feature extraction. Naive Bayes utilized BoW, also from scikit-learn, while CNN used pre-trained word embeddings from FastText [61].

4.4 Implementation of supervised models

4.4.1 Naive Bayes, Logistic Regression, and SVM

Before implementing the supervised machine learning models, Multinomial Naive Bayes, Logistic Regression, SVM and CNN, the optimal hyperparameters for each model were identified using `GridSearchCV` with `StratifiedKFold` with respect to a custom `f3-score`. This initial step is crucial for optimizing the models' performance, particularly for detecting many positive instances. Given the similar implementation of the three traditional machine learning models mentioned, they are discussed collectively. The deep learning model, CNN, will be discussed later in its own section.

4.4.1.1 Hyperparameter Tuning

For the Multinomial Naive Bayes, `GridSearchCV` was utilized to determine the optimal smoothing parameter α, known as the smoothing prior. With $\alpha \geq 0$, the model accounts for features absent in the training data to prevent zero probabilities in predictions. Setting $\alpha = 1$ is used for Laplace smoothing, while $\alpha < 1$ is for Lidstone smoothing. For the multinomial classifier, various values of alpha were evaluated:

```
    mnb_params = {
    # 'clf__alpha': [0, 0.01, 0.1, 0.165, 0.5, 1, 2]
    # 'clf__alpha': np.linspace(0.01, 1, 20)
    # 'clf__alpha': [0, 0.165, 0.13, 0.14]
    'clf__alpha': np.arange(0.01, 1, 0.01)}
```

Logistic Regression's hyperparameter tuning focused on finding optimal values for regularization value C, solver type, maximum iterations, and penalty type:

```
    logReg_params = {
    'model__C': [0.01, 0.1, 1, 10, 100],
    'model__solver': ['liblinear', 'lbfgs', 'saga', 'newton-cg'],
    'model__max_iter': [100, 200, 300],
    'model__penalty': ['l1', 'l2']}
```

Similarly, for the SVM, the grid search aimed to optimize the hyperparameters C, gamma, and the kernel type:

```
    svm_params = {
    'classifier__C': [0.1, 1, 10, 100],
    'classifier__gamma': ['scale', 'auto'],
    'classifier__kernel': ['rbf', 'linear', 'poly']}
```

4.4.1.2 Data Preparation and Model Workflow

To set up the model workflow, a pipeline was constructed for each model using *ImbPipeline* from *imbalanced-learn* library [62] to prepare the data and define the model workflow.

For the Multinomial Naive Bayes classifier, the pipeline is set up as follows:

```
    pipeline = ImbPipeline([
    ('vect', CountVectorizer()),
    ('smote', SMOTE(random_state=42)),
    ('clf', MultinomialNB())])
```

Where:

- `CountVectorizer()` is a BoW vectorizer that converts text into numerical value suitable for model training
- `SMOTE(random_state=42)` is used to apply Synthetic Minority Over-sampling Technique (SMOTE) to oversample the minority class in the dataset. Consequently making the model has a balanced view of all classes during training [62].
- `MultinomialNB()` is the multinomial Naive Bayes classifier

For logistic regression and SVM, a similar pipeline was constructed except for a change in the vectorization technique and the classifier chosen. While the multinational classifier employed a BoW approach using `CountVectorizer()`, logistic regression and SVM used the `TfidfVectorizer()`, which is advantageous as it emphasizes the importance of terms based on their frequency, providing a weighted approach that often enhances the model performance by highlighting relevant features.

For model validation, StratifiedKFold was selected, which is a variation of KFold used for CV. This method is designed to maintain the percentage of samples for each class in each fold [19], which is particularly useful in dealing with imbalanced datasets. The dataset is divided into 5 folds, shuffling the data before each split to ensure that each fold represents an accurate cross-section of the entire dataset.

```
    cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

Additionally, a custom scoring function was defined to compute the F3-score for the positive class, which places more emphasis on recall than precision:

```
    def f3_score_positive_class(y_true, y_pred):
    return fbeta_score(y_true, y_pred, beta=3, labels=[1], average=
    None)[0]
```

The F3-score is crucial for assessing the model's ability to detect instances of bullying while focusing on minimizing false negatives.

The parameters grid, the CV setup, and the custom scoring function are then passed to `GridSearchCV` to find the optimal hyperparameters for each model that yields the highest F3-score for the positive class. For multinomial classifier the `GridSearchCV` looks as follow:

```
grid_search = GridSearchCV(pipeline, param_grid, cv=cv, scoring=
    f3_scorer, verbose=1, n_jobs=-1)
grid_search.fit(texts, labels)
```

4.4.1.3 Evaluation of models

Once the optimal hyperparameters for each model were identified, each model was then evaluated using its established pipeline. The same CV setup with 5 splits was used. During CV, the model was trained on vectorized text data that had been balanced by applying `SMOTE`. For each validation fold the model's `predict_proba` function was employed:

```
    probabilities = classifier.predict_proba(X_test_vec)[:, 1]
```

`predict_proba` estimates the probability of each class for the test data. Here, it calculates the probability that each test instance belongs to the positive class. These probabilities were then saved for later use in determining the best threshold that yields the highest F3-score for the positive class.

To find the optimal threshold for the classifier, a wide range of decision thresholds were systematically tested. For each threshold, different metrics such as precision,

recall, F1-score, and F3-score were calculated for both classes. Additionally, the macro average F3-score was calculated for each threshold. The rest of the code can be found in the Appendix chapter.

4.4.2 CNN

To implement CNN, the *Keras* library was used [63]. Instead of training custom word embeddings from scratch, based on the relatively small Friends dataset, a pre-trained set of Swedish word embeddings was used from FastText [61]. FastText, developed by Facebook's AI Research lab, has generated pre-trained word embeddings trained on large text corpuses of data from *Common Crawl* and *Wikipedia*.

4.4.2.1 Text vectorization

To convert the Friends dataset into vectors that the CNN can process, the Keras *Tokenizer* first maps each unique word in the dataset to a unique integer e.g.:

```
{'jag': 1, 'vara': 2, 'att': 3, ... }
```

then converts each text to a sequence of unique integers e.g.:

```
[[1, 39, 29], [5, 54, 623, 1635, 6, 125], ...]
```

where `[1, 39, 29]` is a text instance from the dataset that start with `"jag"`. It then finds the length of the longest sequence, and creates a matrix that could look something like this:

```
X:  [[1634   497   ...   0    0]
     [149    116   ...   0    0]
                      ...
     [ 13     9    ...   0    0]
     [ 61     2    ...   0    0]]
```

with a height that of the count of texts in the dataset and a width of the longest sequence(text with most words) 0's are added as padding to get a matrix with uniform row width.

```
data = pd.read_csv('data/new_preprocessed_friends_data.csv')
texts = data['text'].values
labels = data['label'].values

tokenizer = Tokenizer()
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
max_length = max([len(seq) for seq in sequences])
X = pad_sequences(sequences, maxlen=max_length, padding='post')
```

4.4.2.2 Pre-trained word embeddings

Next, the 300-dimensional pre-trained *FastText* word embeddings are loaded using `load_facebook_vectors` from the *gensim* library [64]. A new embedding matrix is created where the words from the Friends dataset get assigned the word embeddings from the pre-trained embeddings. The embeddings that don't appear in the Friends dataset aren't inserted into the new embedding matrix. Words from the Friends dataset that aren't in the FastText model receive zero vectors.

```
fasttext_model = load_facebook_vectors('cc.sv.300.bin')

embedding_matrix = np.zeros((len(tokenizer.word_index) + 1,
    fasttext_model.vector_size))
for word, i in tokenizer.word_index.items():
    try:
        embedding_vector = fasttext_model[word]
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
    except KeyError:
        continue
```

4.4.2.3 Layers

Below are the layers used in the implementation. Notably, trainable was set to `False`. By setting this to False, it is specified that the embedding layer's weights (the word vectors) shouldn't be updated during training. This is to retain the values from the pre-trained vectors, which already encode useful information about the semantics of the words. Initially, the `Dropout` layer and the `kernel_regularizer` weren't used. Those were introduced when overfitting was observed.

```
model = Sequential([
    Embedding(input_dim=len(tokenizer.word_index) + 1,
              output_dim=300,
              weights=[embedding_matrix],
              input_length=max_length,
              trainable=False),
    Conv1D(filters=256, kernel_size=3, activation='relu',
           kernel_regularizer=l2(0.01)),
    Dropout(0.6),
    GlobalMaxPooling1D(),
    Dense(units=64, activation='relu', kernel_regularizer=l2(0.01))
    ,
    Dropout(0.6),
    Dense(units=1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics
    =['accuracy'])
```

`Class weights` from scikit-learn are used to address the imbalanced dataset. They adjust the loss function, making the model penalize misclassifications of the minority class (bullying class) more heavily during training.

```
from sklearn.utils.class_weight import compute_class_weight

weights = compute_class_weight('balanced', classes=[0, 1], y=
    y_train)
```

```
4
5  # transform from array to dictionary, so GridSearchCV can use it
6  class_weights = {}
7  for i, weight in enumerate(weights):
8      class_weights[i] = weight
9
10 # Class weights: {0: 0.6047269763651182, 1: 2.88715953307393}
```

Just like with the other supervised models, hyperparameter tuning using `gridsearchCV` and `StratifiedKFold` was performed with respect to a custom `f3-score`. `KerasClassifier` was used here to wrap the Keras model so it could be compatible with the scikit-learn tools. Below are the hyperparameters tuned.

```
1  param_grid = {
2      'optimizer': ['rmsprop', 'adam'],
3      'kernel_regularizer_val': [0.01, 0.02],
4      'dropout_rate': [0.5, 0.6]
5  }
```

4.4.2.4 Evaluation

Once the best hyperparameters were found, this optimal model was trained in a 5-fold CV, 30 epochs per fold. Early stopping was used together with dropout layers and kernel regularizer to combat overfitting. To monitor potential overfitting, the training and validation loss was plotted. This helped to easily identify if the validation loss started increasing at any stage, indicating overfitting.

After the CV, iteration over multiple thresholds was performed, to find the optimal threshold that maximizes the F3-score for the bullying class. Additionally, a histogram of the predicted probabilities was plotted with the colors of the two classes to easily visualize the results. See plots in the next chapter.

4.5 RAG

To implement the RAG, the *Chroma DB* library [65] was used for creating the vector database, and Llama 3 (8B) was downloaded and set up as a local server using the *LM-Studio* [66] software. The 6 most similar text instances (3 bullying, 3 non-bullying) were retrieved from the vector database and merged with the prompt which was then prompted to the LLM. The flow from entering a text, to receiving a classification can be seen in Figure 4.6 below.

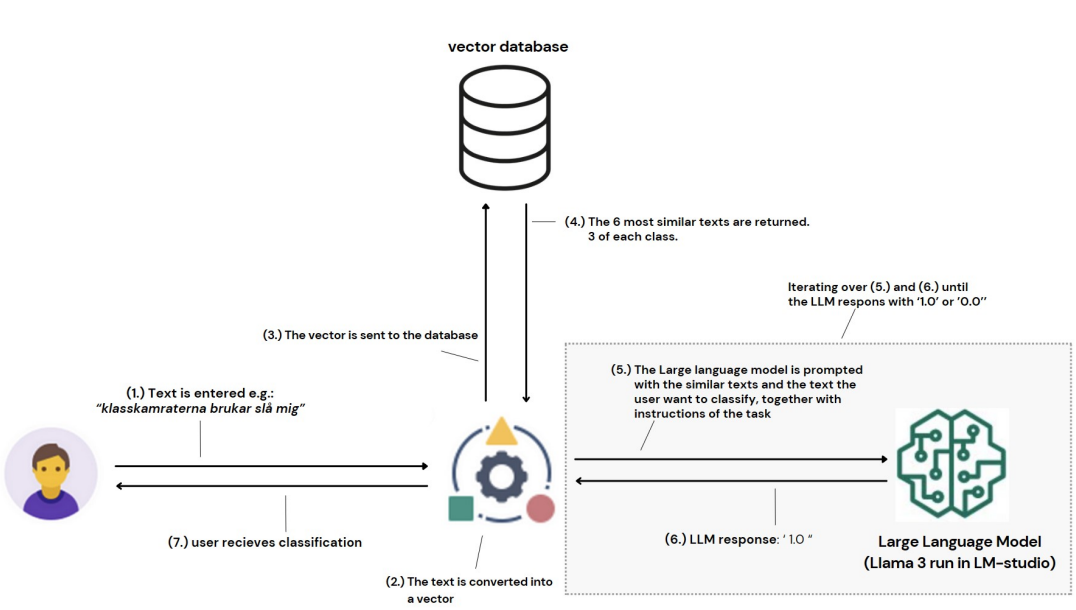

Figure 4.6: RAG implementation

4.5.1 Creating the vector database

The vector database was created from 70% of the Friends dataset. The other rest was be used for prompting the RAG to evaluate it's performance. To create the vector database, the Chroma client is initialized and a `collection` is created. The `documents`(a list of the texts), `metadatas`(the labels), and `ids` are added to the collection.

```
# loading friends data. this dataset is already cleaned from
    unessesary colunms and nan values
data = pd.read_csv('clean_friends.csv')

# ensure all data['text'] are strings
data['text'] = data['text'].astype(str)

# split the data into two parts: one for the vector database and
    one for testing
data_vector_db, data_rag_test = train_test_split(data, test_size
    =0.3, random_state=42)


```

```
# initialize Chroma client and create a collection
chroma_client = chromadb.Client()
collection = chroma_client.create_collection(name="
    friends_collection")

# Prepare the metadata/labels
metadatas = []
for label in data_vector_db['label']:
    metadatas.append({'label': label})

# Add text documents to the collection with metadata and ids
collection.add(
    documents=data_vector_db['text'].tolist(),
    metadatas=metadatas,
    ids=data_vector_db.index.astype(str).tolist()
)
```

The texts are now stored in the vector database.

4.5.2 Querying the vector database

To query the database with a text, so that it returns the most similar one, the following code is run:

```
all_results = collection.query(
    query_texts=["text to query"],
    n_results=2  # number of similar texts to return
)

print(all_results)
```

```
example output:
{
    'ids': [['871', '211']],
    'distances': [[0.8788206577301025, 0.9084813594818115]],
    'metadatas': [[{'label': 0.0}, {'label': 1.0}]],
    'embeddings': None,
    'documents': [['mest liknande text', 'näst mest liknande text']],
    'uris': None, 'data': None
}
```

As mentioned before, it was desirable to get not only the 6 most similar texts but 3 of each class. This was achieved by querying the 100 most similar texts and then sorting out the first three in each class. The high number was chosen to ensure both classes were covered with the challenge of the unbalanced Friends dataset.

4.5.3 LLM of choice

The available hardware in this project allowed for running the Llama 3 (8 billion parameter version) locally.

To run Llama 3 locally, *LM-Studio* [66] was used. This software allows for to download and run open-source models language models on your computer. Instead of running the model on a remote server, it sets up a local server on your machine. This local server can then be accessed and prompted from your code, allowing you to interact with the language model as if it were running on a remote server, but without the need for an internet connection or paying for cloud services.

4.5.4 Prompting Llama 3

The retrieved texts from the vector database are combined with instructions and the text to classify(see example prompt below) and sent as a prompt to the locally run Llama 3 model. This is done by making an API call to the model using the *openai* library [67]. It is instructed to classify the text by responding with only '1.0' or '0.0'. This is to make it convenient for the code to then easily extract the label from the response. As can be seen in the example prompt, the importance of this is repeatedly punctuated. The reason for this is because of the model's tendency to give long explanations and motivations, not following the instructions. Despite this, it repeatedly avoided answering according to the instructions, making it hard to extract the labels. This was handled by decreasing the amount of tokens it was allowed to respond with to only 2 tokens, combined with iterating until a valid response of either '1.0' or '0.0' was returned. The low token limit forces the language model to give an answer of no longer than around one word, making it more prone to answer either '1.0' or '0.0'.

Example prompt:

```

Klassificera den sista textmeningen utifrån om personen som skrivit
den verkar mobbad i skolmiljön. Om det tyder på mobbning, svara
endast med '1.0'. Annars, svara endast med '0.0'. Klassificeringen
gäller enbart om författaren av texten är mobbad. Innehåll som bara
är stötande eller våldsamt innebär inte automatiskt att texten ska
märkas som '1.0'.

Här är sex märkta exempel:

- "Folk brukar kasta saker på mig" - label: 1.0
- "Lärarna brukar spotta på mig" - label: 1.0
- "Folk brukar kalla mig fula saker osv." - label: 1.0
- "Alla brukar sjunga ramsor om mig" - label: 0.0
- "Mina klasskamrater försöker mata mig" - label: 0.0
- "Vissa brukar jaga mig" - label: 0.0

Klassifiera nu följande text:  "klasskamraterna brukar slå mig."

(OBS! svara endast med '1.0' eller '0.0')
```

When prompting, the `temperature` was set to `0.7`. The motivation behind this was to give Llama 3 some room for "changing its mind" if it got stuck in a loop of not responding with either '1.0' or '0.0', while not responding randomly, allowing for some reasonably consistent results.

4.5.5 Evaluating RAG implementation

To evaluate the model, the training split of the Friends dataset was prompted one text instance at a time. The responses were compared with the actual labels so that a confusion matrix and the various metrics could be computed.

4.6 Prompting GPT-4o

During the final days of the thesis, OpenAI released a new model, which according to them is "their fastest and most affordable flagship model" [47]. Following a suggestion from our supervisor Elias, this model was prompted with anonymized entries to evaluate its efficiency and cost-effectiveness.

To create a smaller, curated, and anonymized dataset from the original one, a script was developed to allow the manual selection of instances that didn't contain any names or sensitive information. The result is 210 instances of the non-bullying class and 200 instances of the bullying class.

Using the OpenAI API and the curated dataset, which consists of raw, unprocessed text, the GPT-4o model was prompted. The results of these prompts were recorded, and a new column containing the GPT-4o predictions was then appended to the dataset.

```
client = OpenAI(api_key='####')
data = pd.read_csv('./GPT_Data/Friends_not_sens_filtered.csv')

def classify_text(prompt, text):

    messages = [
        {"role": "system", "content": prompt},
        {"role": "user", "content": text}
    ]

    try:
        response = client.chat.completions.create(
            model="gpt-4o",
            messages=messages,
            max_tokens=1,
            temperature=0
        )

        # accessing the response content
        label = response.choices[0].message.content.strip()
```

```
21
22          return int(float(label))
23
24      except Exception as e:
25          print(f"Error: {e}")
26          return None
27
28  data['predicted_label'] = data['text'].apply(lambda x:
        classify_text(prompt, x))
```

The `classify_text` function is designed to interact with the OpenAI API to classify text data. The function constructs a list of messages, including a system prompt and user input. It then sends these messages to the GPT-4o model via the API. `max_tokens` is set to 1 to only allow 1 or 0 answers via the model. Additionally, by setting the temperature to 0, the model is instructed to choose the token with the highest probability. This results in deterministic predictable outputs that match the model's training. Finally, the function is applied to each instance in the anonymized dataset. The specific prompts and their impact on the results will presented in the next chapter.

5 Experimental Setup, Results and Analysis

This chapter provides an overview of the hardware and software used in this study as shown in Table 5.6 and Table 5.7, followed by a detailed presentation of the results obtained from the experiments. It presents figures for experiments, with and without stop words removed. This chapter concludes with an analysis of these results, focusing on analyzing the performance metrics for classifying bullying, particularly the F3-score, precision, and recall for the positive class. The objective is to find the most effective model for identifying as many bullying instances as possible without generating too many false positives. Since the results are better when keeping stop words in the text data, the subsequent analysis will focus on the results obtained without removing stop words.

5.1 Hardware

Model	**Log Reg, CNN, RAG**	**Naive Bayes, SVM**
OS	Win 10	Win 11
CPU	Intel i7-4790K	Ryzen 7 5800X
GPU	RTX 2070, 8GB	GTX 1060, 6GB
RAM	16 GB	16 GB

Table 5.6: Hardware specs implemented on different machines

5.2 Software

Model	**Log Reg, Naive B, SVM**	**CNN**	**RAG**
Vscode	1.89.1	1.89.1	1.89.1
Python	3.10.0	3.8.8	3.9.9
scikit-learn	1.3.2	1.3.2	0.24.1
matplotlib	3.3.4	3.3.4	-
Numpy	1.20.1	1.20.1	-
Keras	-	2.4.3	-
Tensorflow	-	2.3.0	-
Gensim	-	4.3.2	-
Seaborn	0.11.1	0.11.1	-
Pandas	1.2.4	1.2.4	1.2.4
LM-studio	-	-	0.2.23
LLM	-	-	Llama 3 8B Instruct
Chromadb	-	-	0.5.0
OpenAI	-	-	1.25.1
Conda	-	-	4.12.0

Table 5.7: Software versions implemented on different machines

5.3 Logistic Regression

Hyperparameters found in the grid search:

```
model__C = 1, model__solver = lbfgs
model__max_iter = 100, model__penalty = l2


Optimal threshold (stop words not removed) = 0.25
Optimal threshold (stop words removed) = 0.25
```

For Logistic Regression, the different scores for both approaches, with and without stop words are presented in Table 5.8 and Table 5.9. The most effective approach, which retains stop words, achieves a high recall of 0.93, meaning it successfully identifies a large proportion of bullying instances. This is reflected in the confusion matrix in Figure 5.7, where out of 634 positive instances, 589 are correctly identified as bullying (TP), and only 45 bullying instances are missed (FN). However, the low precision of 0.40 indicates a high rate of FP, with 887 non-bullying instances incorrectly classified as bullying out of 3076 negative instances. The model correctly identifies 2198 non-bullying instances (TN). Removing stop words resulted in a slight increase in FP and a slight increase in the other three cases as shown in the confusion matrix in Figure 5.8.

Class	**Precision**	**Recall**	**F1-Score**	**F3-Score**	**Macro F3**
Non-bullying	0.98	0.71	0.82	0.73	0.78
Bullying	0.40	0.93	0.56	**0.82**	

Table 5.8: Results of Logistic Regression (stop words not removed)

Class	**Precision**	**Recall**	**F1-Score**	**F3-Score**	**Macro F3**
Non-bullying	0.97	0.73	0.83	0.75	0.78
Bullying	0.41	0.91	0.57	**0.81**	

Table 5.9: Results of Logistic Regression (stop words removed)

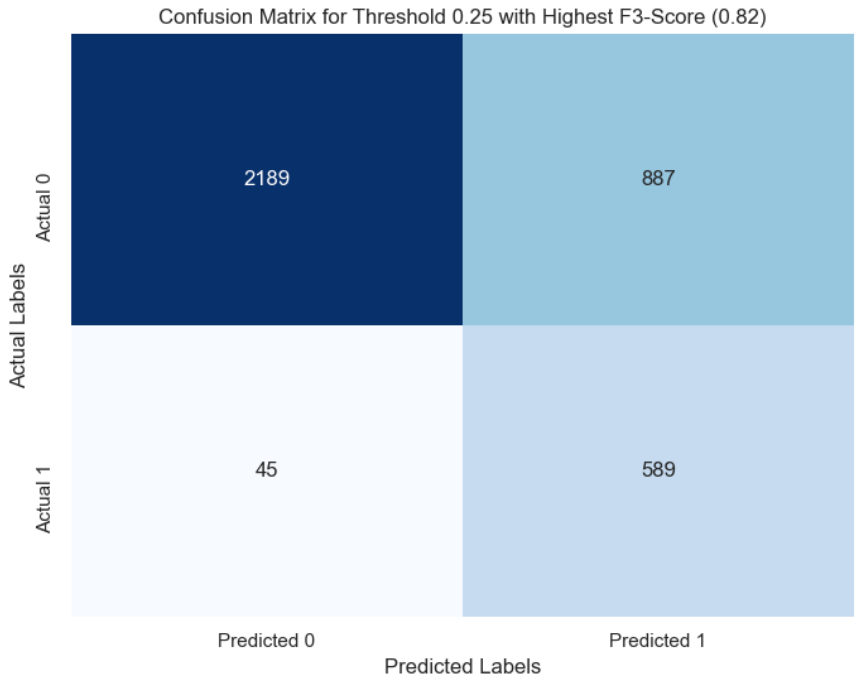

Figure 5.7: Confusion matrix Logistic Regression (stop words not removed)

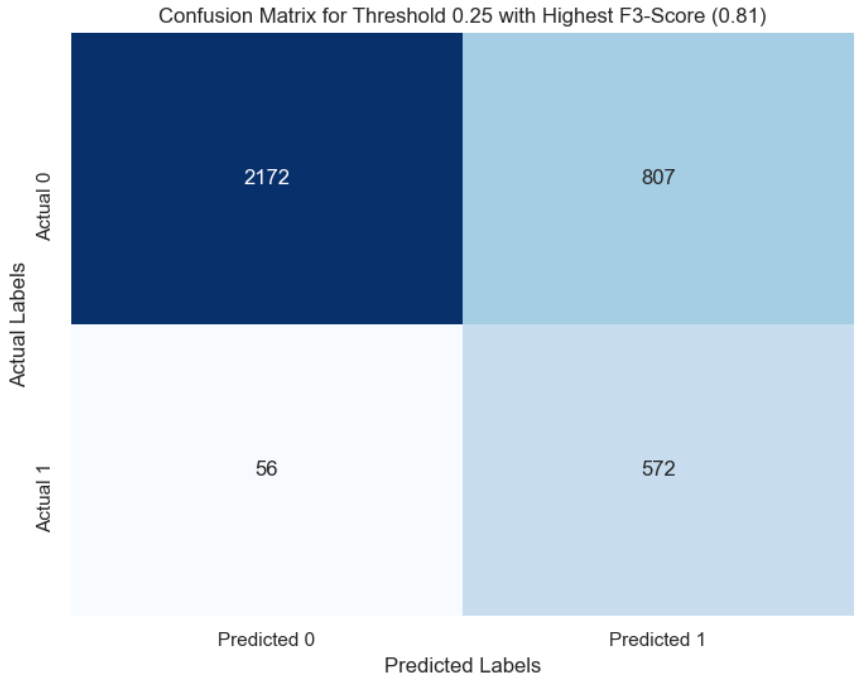

Figure 5.8: Confusion matrix Logistic Regression (stop words removed)

The histograms of the predicted probability distributions for both scenarios are presented in Figure 5.9 and Figure 5.10 illustrate that the Logistic Regression model successfully divides the two labeled classes, although quite a lot of overlapping is

present. It appears to be more confident in classifying non-bullying, compared to bullying.

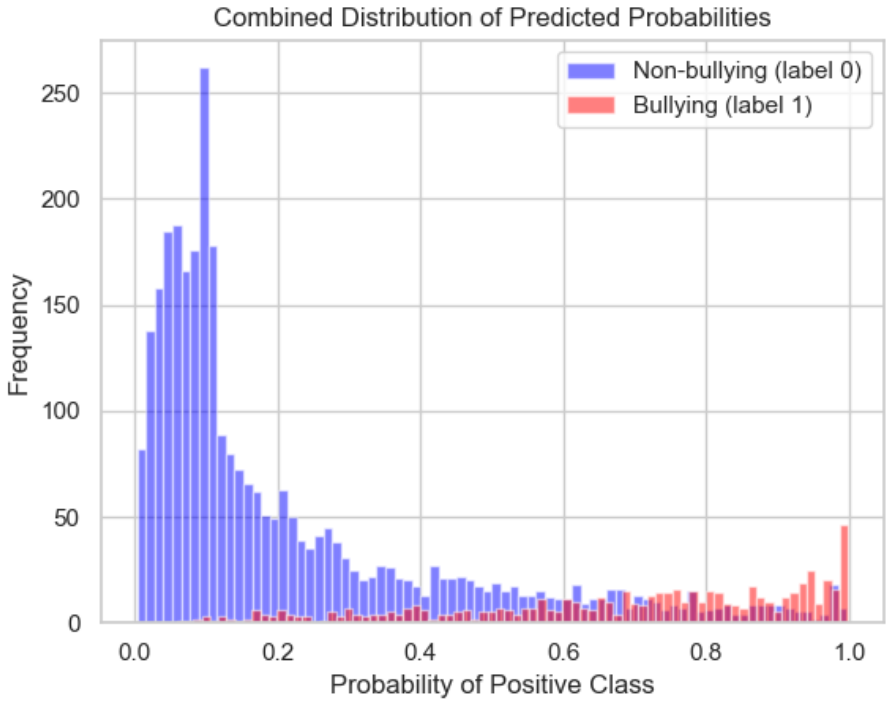

Figure 5.9: Predicted probability distribution Logistic Regression(stop words not removed)

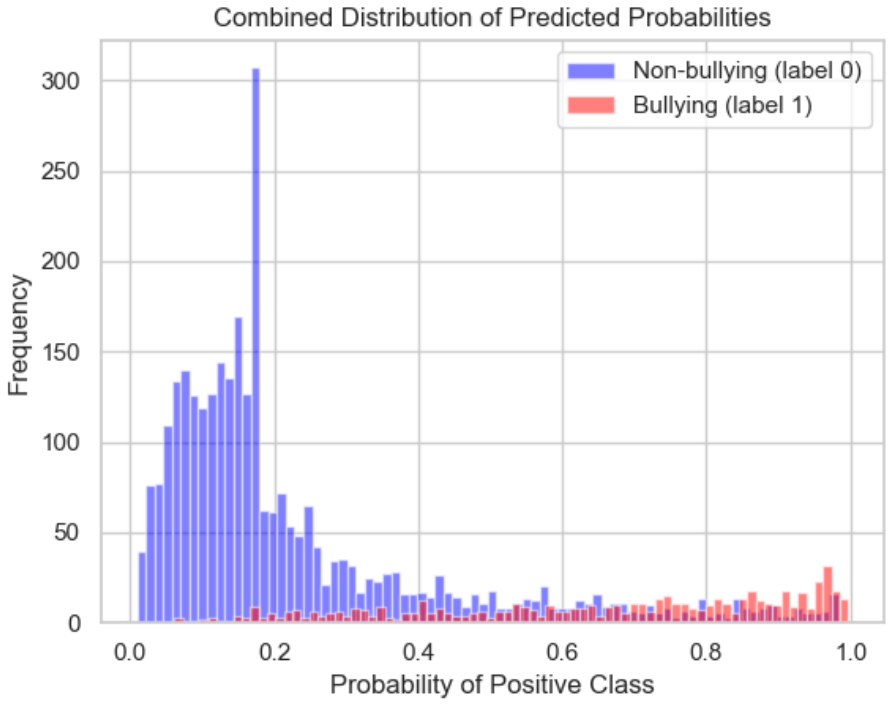

Figure 5.10: Predicted probability distribution Logistic Regression(stop words removed)

5.4 Naive Bayes

Hyperparameters found in the grid search:

```
clf__alpha = 0.16
```

```
Optimal threshold (stop words not removed) = 0.22
Optimal threshold (stop words removed) = 0.10
```

For Multinomial Naive Bayes, the performance metrics for both approaches, with and without stop words, are presented in Table 5.10 and Table 5.11. The Naive Bayes classifier performs similarly to Logistic Regression but with slightly lower recall and precision, scoring 0.85 and 0.35, respectively. This is reflected in the confusion matrix in Figure 5.11, where out of 634 positive instances, 538 are correctly identified as bullying (TP), and 96 are missed (FN). This model also shows a high rate of FP, with 997 non-bullying instances incorrectly classified as bullying out of 3076 negative instances. This lower precision indicates that Multinomial Naive Bayes is less effective at distinguishing between bullying and non-bullying instances, potentially due to its assumption of feature independence, as previously mentioned in 2.3.1. The model correctly identifies 2079 non-bullying instances (TN), indicating it struggles more with specificity compared to Logistic Regression, leading to a higher number of false positives. The removal of stop words results in a lower recall and F3-score as shown in Table 5.11, but it also results in fewer FP as shown in The confusion matrix in Figure 5.12.

Class	**Precision**	**Recall**	**F1-Score**	**F3-Score**	**Macro F3**
Non-bullying	0.96	0.68	0.79	0.70	0.72
Bullying	0.35	0.85	0.50	**0.74**	

Table 5.10: Results of Naive Bayes (stop words not removed)

Class	**Precision**	**Recall**	**F1-Score**	**F3-Score**	**Macro F3**
Non-bullying	0.96	0.50	0.65	0.52	0.62
Bullying	0.27	0.89	0.42	**0.72**	

Table 5.11: Results of Naive Bayes (stop words removed)

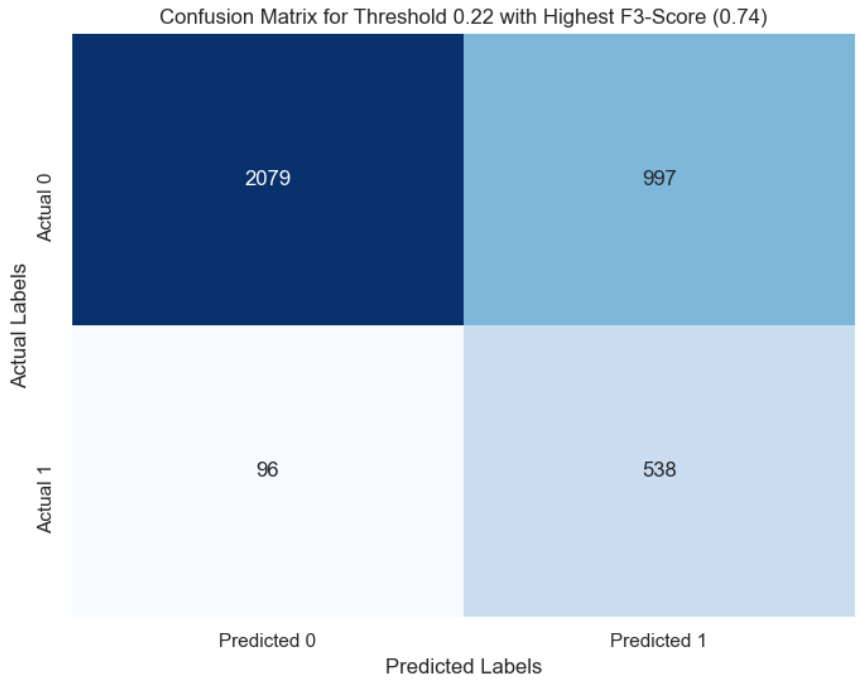

Figure 5.11: Confusion matrix Multinomial Naive Bayes (stop words not removed)

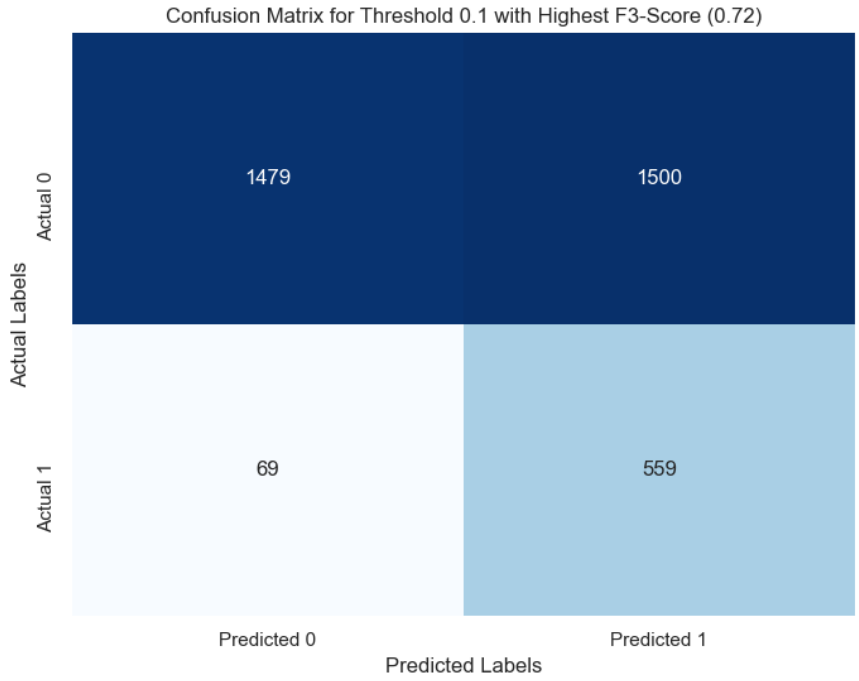

Figure 5.12: Confusion matrix Multinomial Naive Bayes (stop words removed)

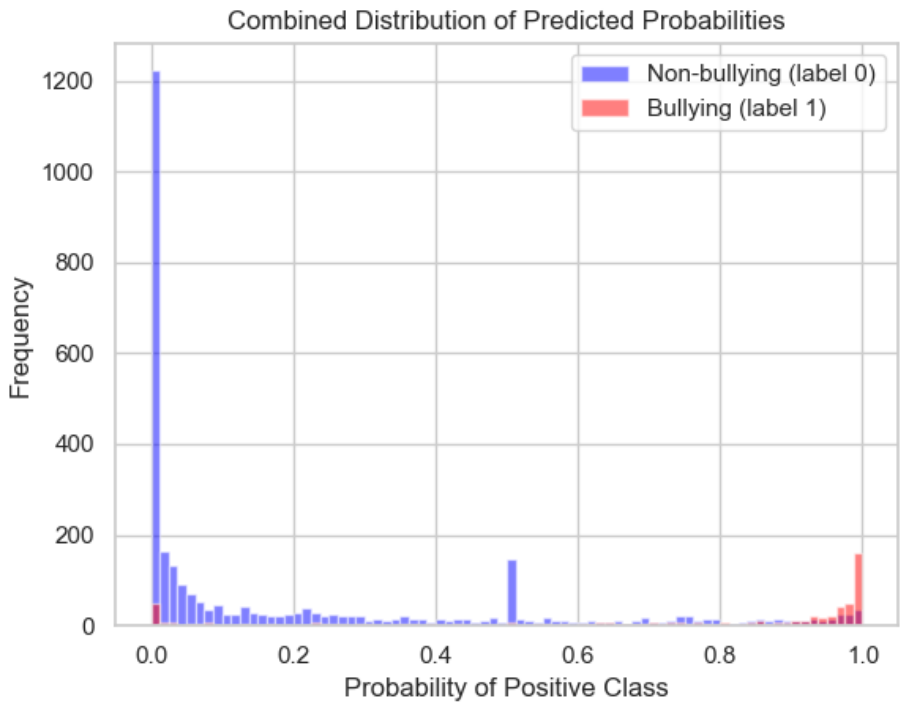

Figure 5.13: NB Predicted probability distribution (stop words not removed)

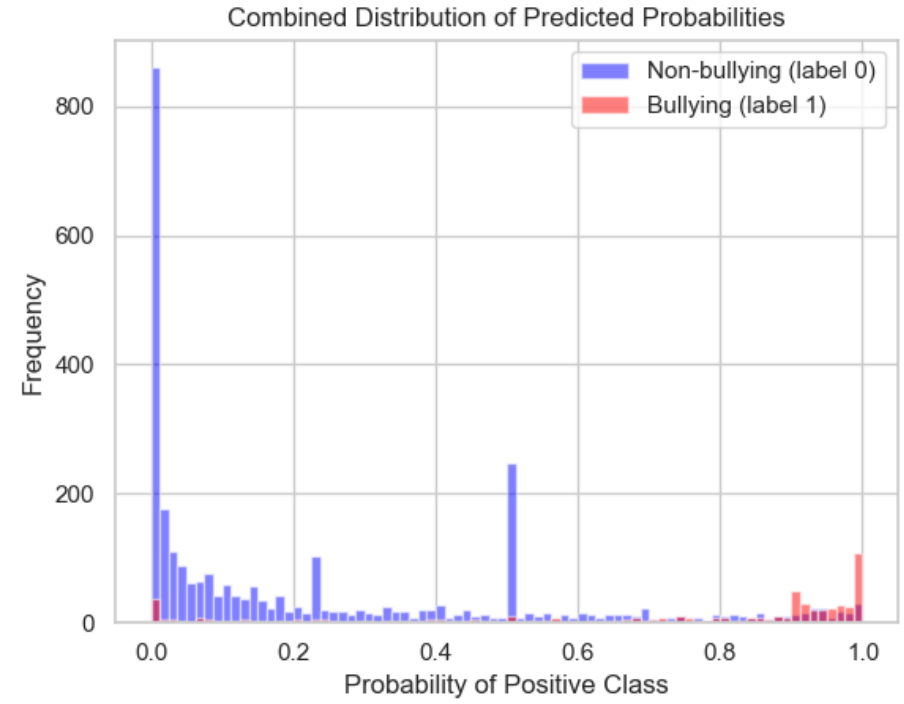

Figure 5.14: NB Predicted probability distribution (stop words removed)

The histograms of the predicted probability distributions for both scenarios presented in Figure 5.13 and Figure 5.14 illustrate that while the model is generally more decisive in its predictions when stop words are kept, It has a clearer separation

of the two label classes towards 0.0 and 1.0, with not a lot of overlapping. A clear spike can be observed at 0.0 of probabilities where it's very confident the prediction is non-bullying. However, removing stop words resulted in a slight overlap of the bullying class. This suggests that the presence of stop words helps the model better delineate the boundary between the classes, despite the higher false positive rate.

5.5 SVM

Hyperparameters found in the grid search:

```
classifier__C = 1, classifier__gamma = 'scale'
classifier__kernel = 'linear'


Optimal threshold (stop words not removed) = 0.05
Optimal threshold (stop words removed) = 0.04
```

For SVM, the performance metrics for both approaches, with and without stop words, are presented in Table 5.12 and Table 5.13. When stop words are retained, SVM shows a balance between recall and precision, performing slightly better than Logistic Regression in terms of precision while maintaining a similar recall. With a precision of 0.43 and recall of 0.92, SVM has fewer false positives, making it a better choice for reducing the over-classification of bullying instances while still capturing most true bullying cases. This is reflected in the confusion matrix in Figure 5.15, where out of 634 positive instances, 586 are correctly identified as bullying (TP), and 48 are missed (FN). Additionally, 772 non-bullying instances are incorrectly classified as bullying out of 3076 negative instances, which is lower than both Logistic Regression and Naive Bayes. The model correctly identifies 2304 non-bullying instances (TN), indicating that SVM maintains a better balance between sensitivity and specificity, reducing the number of FP while still effectively identifying bullying instances. When stop words are removed, the performance slightly diminishes, as seen in the confusion matrix in Figure 5.16. There is a slight increase in FP and a slight decrease in TP. This demonstrates that stop words play a crucial role in SVM's ability to distinguish between bullying and non-bullying cases.

Class	**Precision**	**Recall**	**F1-Score**	**F3-Score**	**Macro F3**
Non-bullying	0.98	0.75	0.85	0.77	0.80
Bullying	0.43	0.92	0.59	**0.83**	

Table 5.12: Results of SVM (stop words not removed)

Class	**Precision**	**Recall**	**F1-Score**	**F3-Score**	**Macro F3**
Non-bullying	0.97	0.71	0.82	0.73	0.77
Bullying	0.40	0.91	0.55	**0.81**	

Table 5.13: Results of SVM (stop words removed)

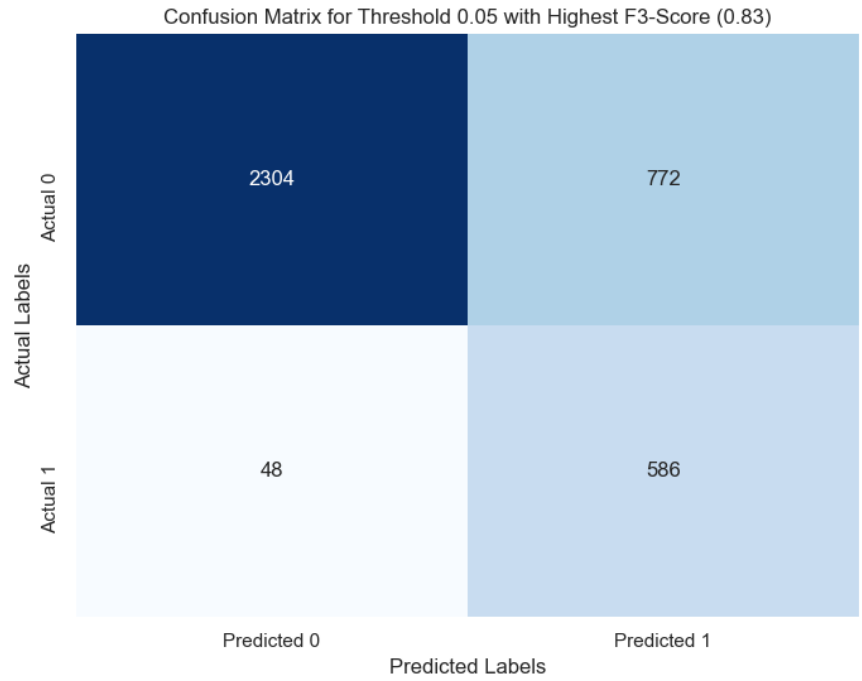

Figure 5.15: Confusion matrix SVM (stop words not removed)

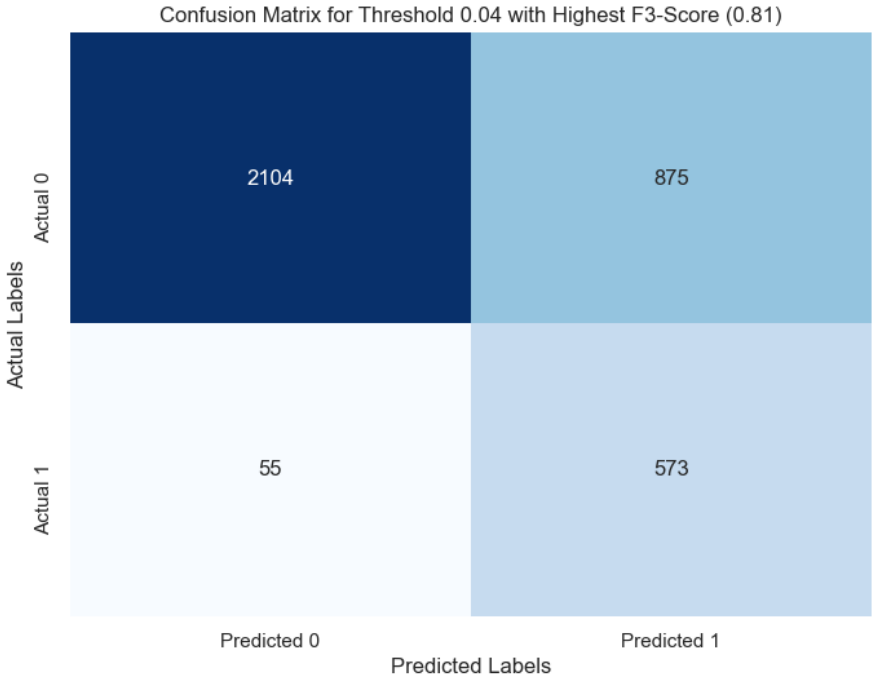

Figure 5.16: Confusion matrix SVM (stop words removed)

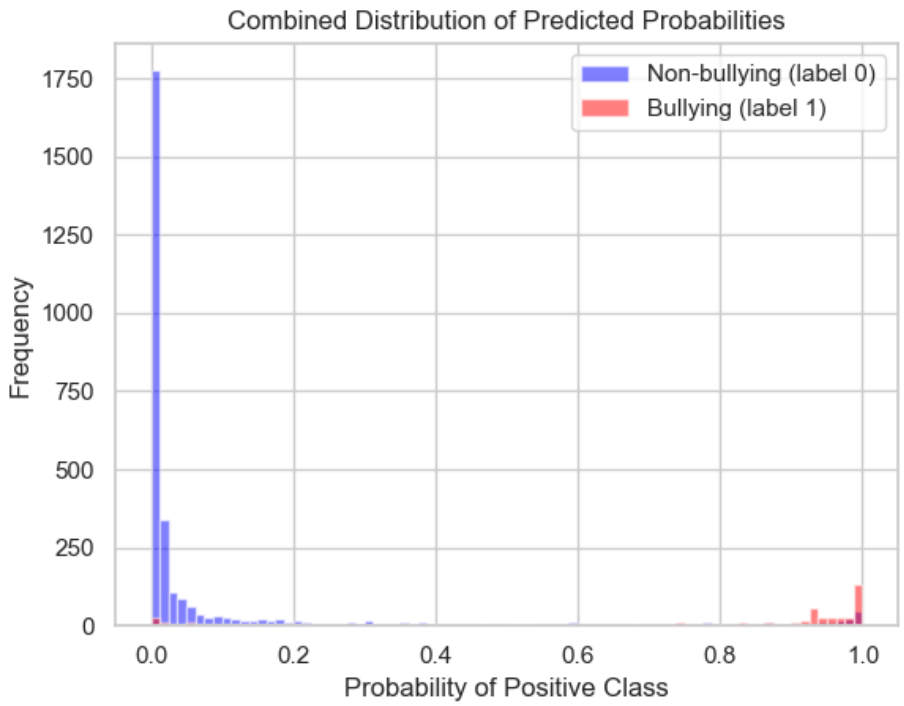

Figure 5.17: SVM Predicted probability distribution (stop words not removed)

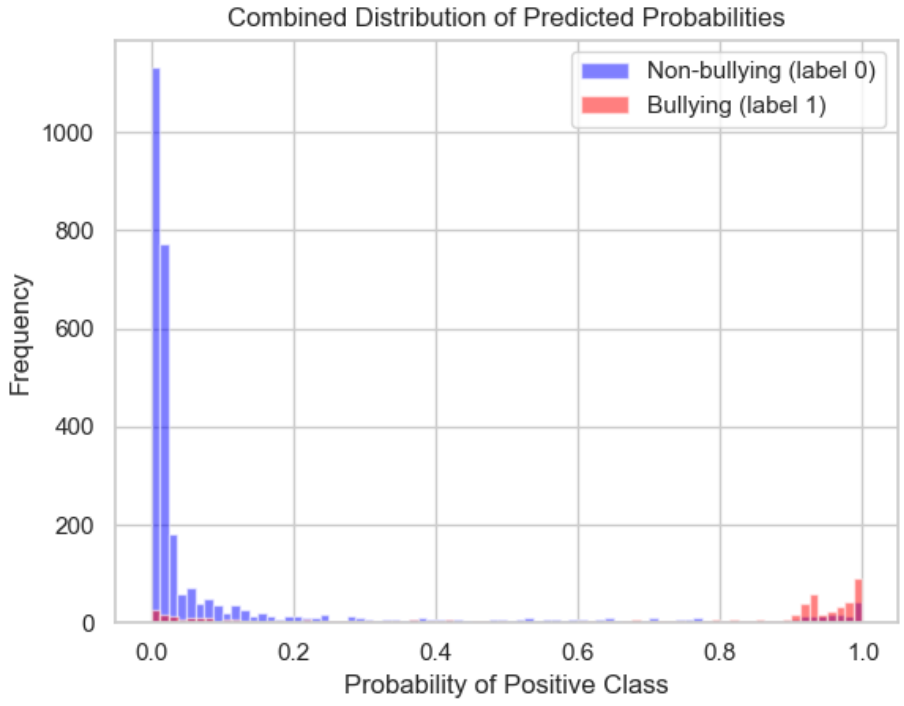

Figure 5.18: SVM Predicted probability distribution (stop words removed)

The predicted probability distribution plots reveal that this model is even more confident in its predictions compared to the two already presented models. It's pushing almost all predictions over 0.9 and under 0.1, while not exhibiting a lot of

overlapping in the classes when stop words are not removed. Figure 5.17 shows a clear separation of probabilities. However, with stop words removed, as shown in Figure 5.18, there is a slight overlap of the bullying class and some probabilities exist in the middle, suggesting a decrease in the model's confidence and clarity in distinguishing between the two classes.

5.6 CNN

Hyperparameters found in gridsearch: `optimizer='adam'`, `kernal_regularisation=0.01`, and `dropout_rate=0.6`.

```
Optimal threshold (stop words not removed) = 0.25
Optimal threshold (stop words removed) = 0.3
```

Similar but slightly better results are achieved without removing stopwords, as can be seen in Figure 5.14 with the F3-score of 0.81, compared to F3-score 0.8 in Figure 5.15.

Class	Precision	Recall	F1-Score	F3-Score	Macro F3
Non-bullying	0.98	0.66	0.79	0.68	0.75
Bullying	0.36	0.94	0.52	**0.81**	

Table 5.14: Results of CNN (stop words not removed)

Class	Precision	Recall	F1-Score	F3-Score	Macro F3
Non-bullying	0.98	0.61	0.75	0.63	0.71
Bullying	0.34	0.93	0.50	**0.80**	

Table 5.15: Results of CNN (stop words removed)

Figure 5.19 shows how the training and validation loss curves decrease at a similar rate for each epoch. This indicates that the CNN model learns effectively without overfitting the training data. The two losses still remain quite high though, possibly indicating some underfitting. This might also explain the relatively uniform distribution of probabilities seen in Figure 5.20 and Figure 5.21. This model doesn't seem as confident in its classification predictions as the models analyzed earlier.

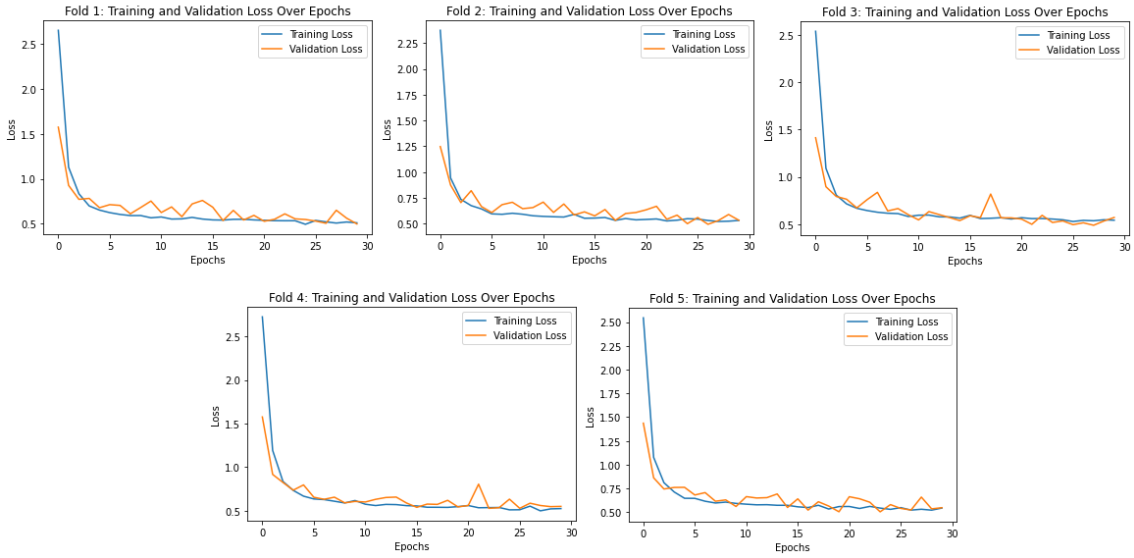

Figure 5.19: Training loss and validation loss over epochs

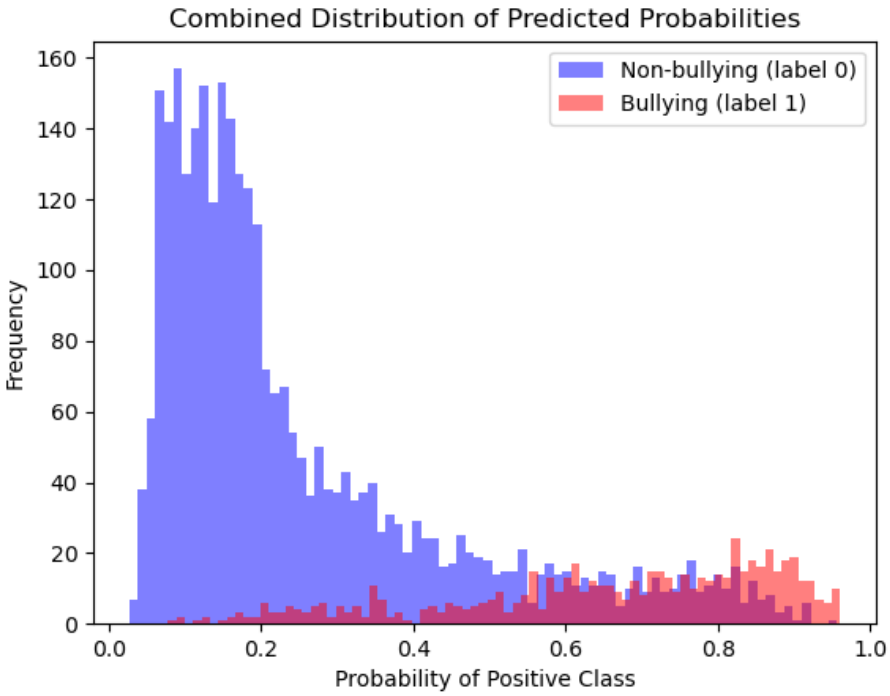

Figure 5.20: Predicted probability distribution CNN (stop words not removed)

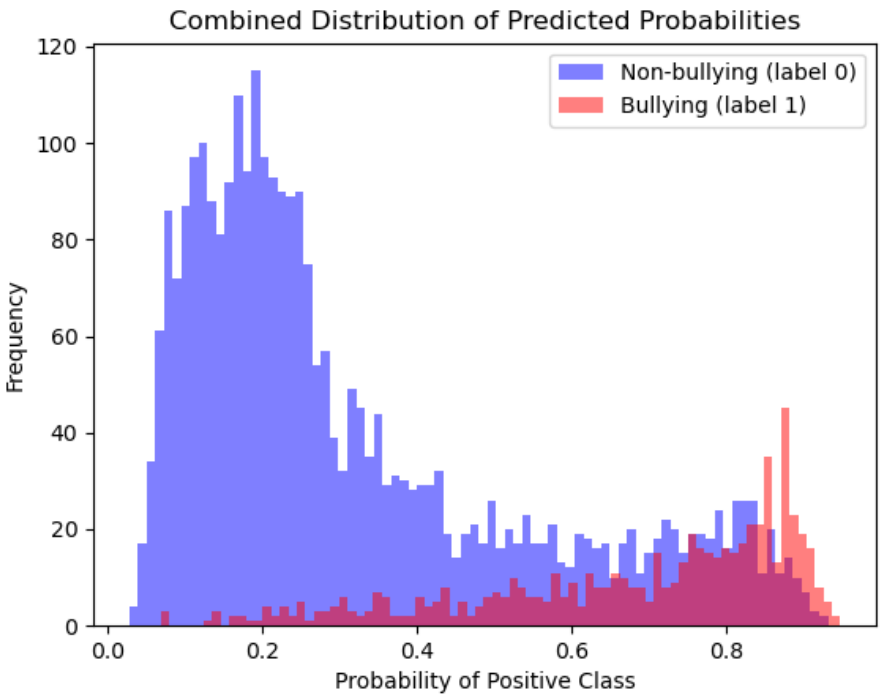

Figure 5.21: Predicted probability distribution CNN (stop words removed)

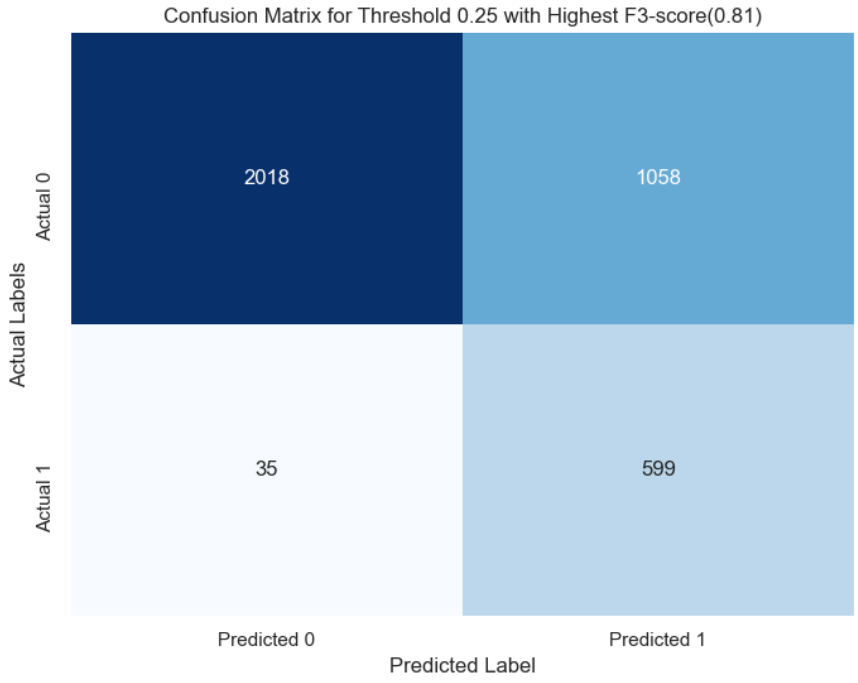

Figure 5.22: Confusion matrix CNN (stop words not removed)

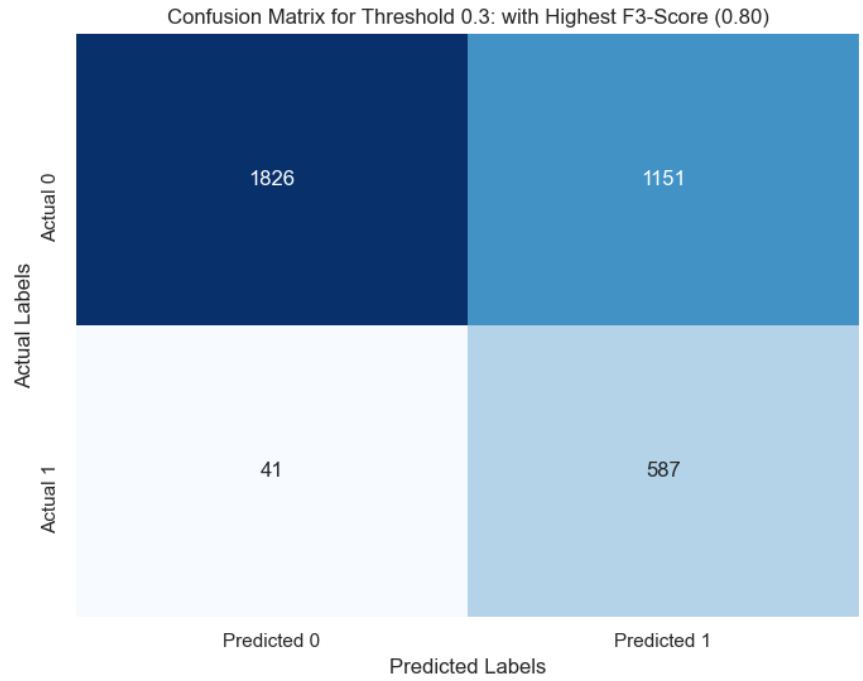

Figure 5.23: Confusion matrix CNN (stop words removed)

With the optimal threshold of 0.25, the model demonstrates a high recall of 0.94, achieving a strong sensitivity to bullying instances, but at the cost of a low precision of 0.36, leading to many false positives. This is reflected in the confusion matrix in Figure 5.22, where out of 634 positive instances, 599 are correctly identified as bullying (TP), but with 1058 non-bullying instances incorrectly classified as bullying (FP). Due to the low threshold set, the non-bullying class naturally has a lower recall of 0.66 and a higher precision of 0.98. Or 2018 out of 3066 correctly identified non-bullying instances (TN), but with 1058 FP, as can be seen in the confusion matrix. The results with the stopwords removed shown in Figure 5.23 shows similar results but slightly weaker performance.

5.7 RAG

Table 5.16 shows that RAG implementation has the highest recall among all models at 0.97, indicating it rarely misses bullying instances. However, its precision is the lowest at 0.29, meaning it generates a significant number of false positives. This model is highly sensitive to bullying but lacks the ability to accurately filter out non-bullying instances, making it less practical despite its high recall.

Class	Precision	Recall	F1-Score	F3-Score	Macro F3
Non-bullying	0.99	0.55	0.71	0.57	0.68
Bullying	0.29	0.97	0.45	**0.79**	

Table 5.16: Results of RAG

The confusion matrix in Figure 5.24 shows that out of 179 positive instances, 174 are correctly identified as bullying, and 5 are missed (FN). Out of 945 non-bullying instances, 519 are correctly identified (TN) and 426 are incorrectly classified as bullying (FP).

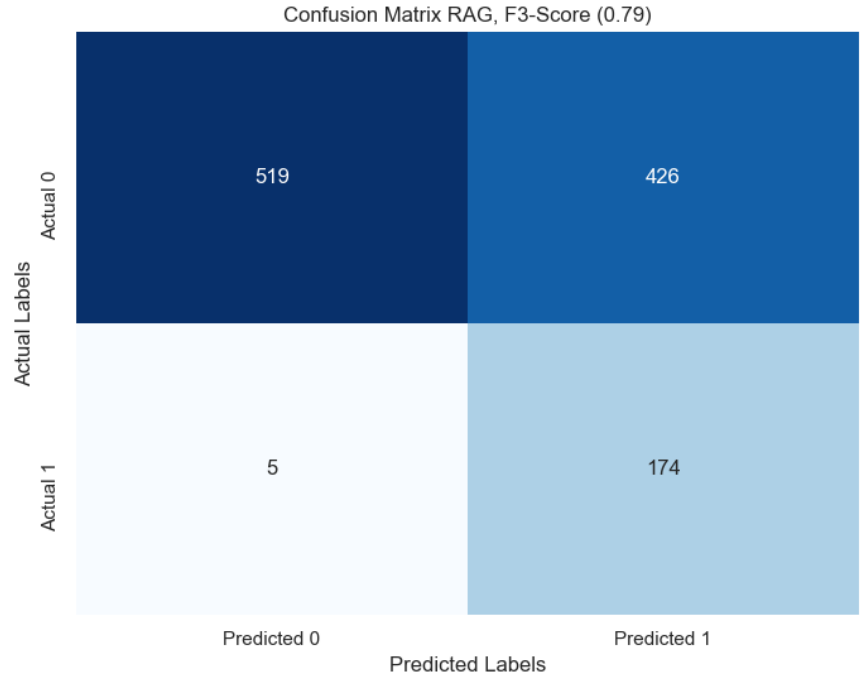

Figure 5.24: Confusion matrix RAG

5.8 GPT-4o

In addition to the prompt used with Llama 3 in 4.5, slight modifications were made to it to observe how the results might differ depending on the prompt. The performance metrics of both prompts are presented below in Table 5.17 and Table 5.18.

```
initial_prompt = Klassificera följande text utifrån om personen som
    skrivit den verkar mobbad i skolmiljön. Om det tyder på
   mobbning, svara endast med '1.0'. Annars, svara endast med
   '0.0'. Klassificeringen gäller enbart om författaren av texten ä
   r mobbad. Innehåll som bara är stötande eller våldsamt innebär
   inte automatiskt att texten ska märkas som '1.0'.

modified_prompt = Klassificera följande text utifrån om personen
   som skrivit den verkar mobbad, rädd, hotad, eller råkar för våld
    i skolmiljön. Om det tyder på mobbning, rädsla eller hot, svara
    endast med '1.0'. Annars, svara endast med '0.0'.
   Klassificeringen gäller enbart om författaren av texten är
   mobbad, känner sig rädd eller hotad. Innehåll som bara är stö
   tande eller upprörande innebär inte automatiskt att texten ska m
   ärkas som '1.0'.
```

GPT-4o achieves the highest overall performance with the modified prompt, with a balanced and high precision of 0.75 and recall of 0.94. This indicates it is very

effective at correctly identifying bullying instances while minimizing false positives. The high F3-score of 0.92 reflects its superior ability to prioritize recall, ensuring that few bullying instances are missed while still maintaining a reasonable precision.

For the first prompt, the confusion matrix in Figure5.25 shows that out of 200 positive instances, 170 are correctly identified as bullying, and 30 are missed (FN). Out of 210 non-bullying instances, 166 are correctly identified (TN), 44 are incorrectly classified as bullying (FP).

For the second modified prompt, the confusion matrix in Figure 5.26 indicates an improved performance. Out of the 200 positive instances, 189 are correctly identified as bullying, and only 11 are missed (FN). However, there is a slight increase in false positives, with 64 non-bullying instances incorrectly classified as bullying out of 210 negative instances. This results in slightly less precision but with higher recall than the first prompt. The results of the second prompt reflect GPT-4o's superior ability to accurately detect bullying instances while gaining a reasonable rate of false positives, making it the most reliable model for this task. However, this model isn't free to use and it can't be prompted with sensitive data.

Class	**Precision**	**Recall**	**F1-Score**	**F3-Score**	**Macro F3**
Non-bullying	0.85	0.79	0.82	0.88	0.82
Bullying	0.79	0.85	0.82	**0.84**	

Table 5.17: Results of GPT-4o with initial prompt

Class	**Precision**	**Recall**	**F1-Score**	**F3-Score**	**Macro F3**
Non-bullying	0.93	0.70	0.80	0.71	0.82
Bullying	0.75	0.94	0.83	**0.92**	

Table 5.18: Results of GPT-4o with the modified prompt

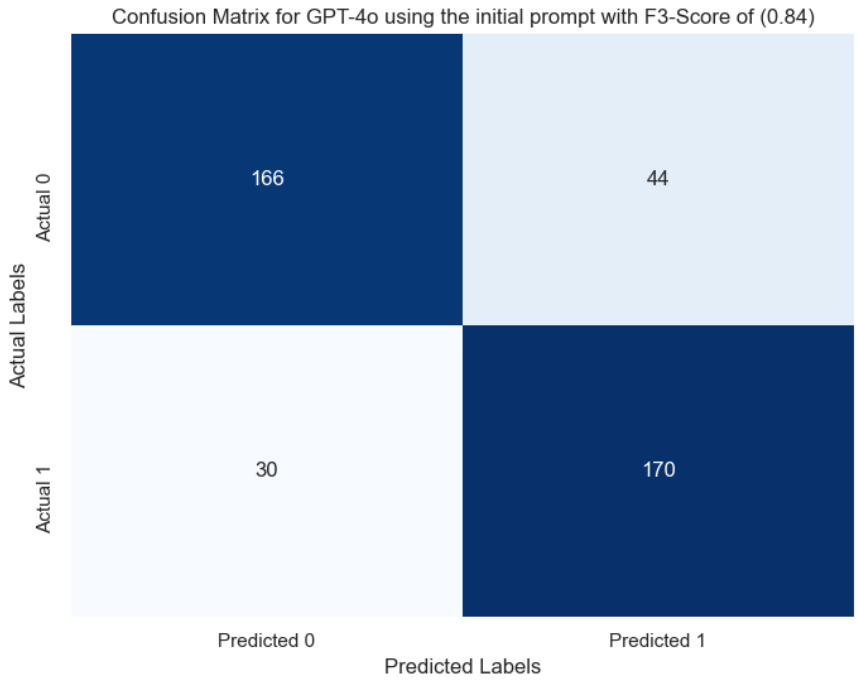

Figure 5.25: Confusion matrix GPT-4o with initial prompt

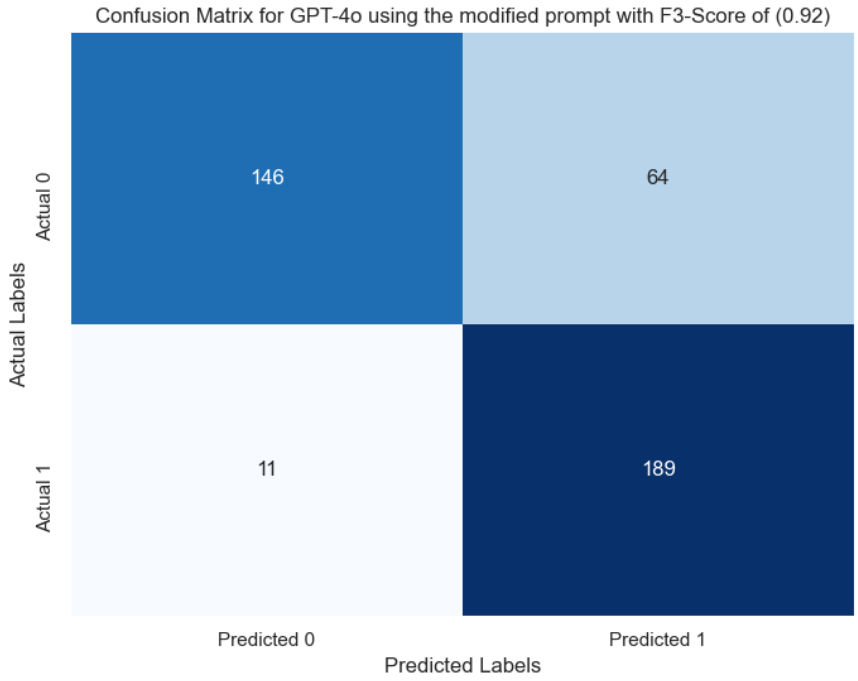

Figure 5.26: Confusion matrix GPT-4o with modified prompt

5.9 Summarizing results

In Table 5.19 below, the bold font is used to highlight the highest score achieved for each evaluation metric across the different models. The best scores are highlighted amongst the supervised models (Logistic Regression, Naive Bayes, SVM, and CNN) and then amongst the two techniques involving LLMs (RAG and GPT-4o).

Model	Class	Precision	Recall	F1-Score	F3-Score	Macro F3
Log Reg	Non-bullying	0.98	0.71	0.82	0.73	0.78
	Bullying	0.40	0.93	0.56	0.82	
Naive Bayes	Non-bullying	0.96	0.68	0.79	0.70	0.72
	Bullying	0.35	0.85	0.50	0.74	
SVM	Non-bullying	0.98	0.75	**0.85**	0.77	**0.80**
	Bullying	**0.43**	0.92	0.59	**0.83**	
CNN	Non-bullying	0.98	0.66	0.79	0.68	0.75
	Bullying	0.36	**0.94**	0.52	0.81	
RAG	Non-bullying	0.99	0.55	0.71	0.57	0.68
	Bullying	0.29	0.97	0.45	0.79	
GPT-4o	Non-bullying	0.93	0.70	0.80	0.71	**0.82**
	Bullying	**0.75**	**0.94**	**0.83**	**0.92**	

Table 5.19: Comparative results across different models (stop words not removed)

In Figure 5.27, the result for the bullying class is presented in an alternative way. The bar for the F3-score is highlighted, since this is the metric we finally compare the models on.

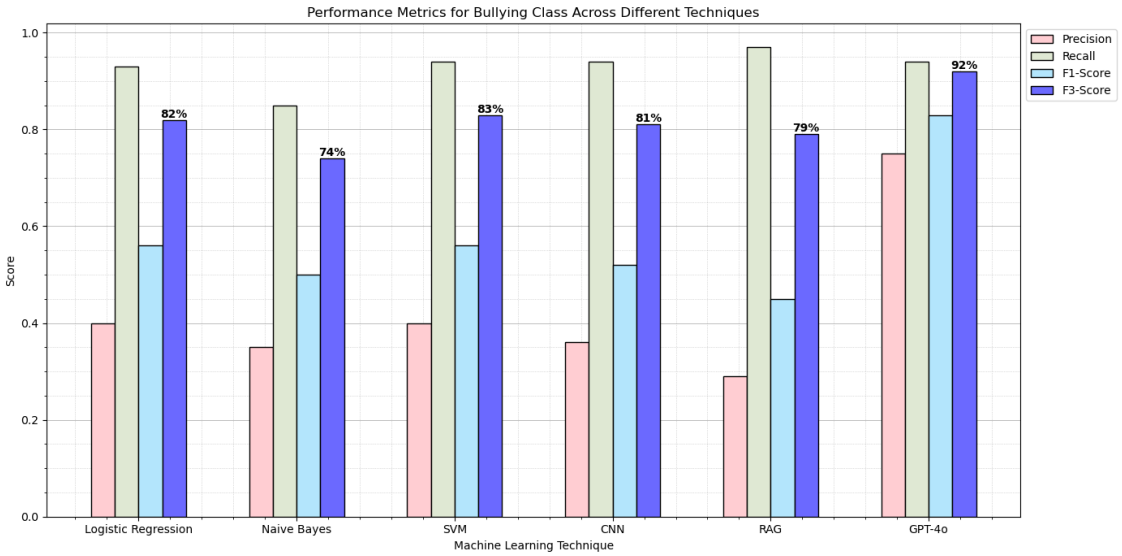

Figure 5.27: Performance Metrics for Bullying Class Across Different Techniques

6 Discussion

This chapter discusses the findings of this thesis and evaluates whether the research questions have been answered. The project aimed to identify the most effective supervised machine learning model for classifying bullying in textual data and compare it with a locally implemented RAG model.

6.1 Answering the research questions

- Which classification algorithm, among Logistic Regression, Naive Bayes, SVM, and CNN, performs best for classifying bullying in textual data?
- How does the performance of a locally implemented RAG model using a vectorized database compare to the best-performing supervised machine learning model in detecting instances of bullying in textual data?

Based on the analysis, the SVM showed a balance between precision and recall, performing slightly better than Logistic Regression, CNN, and Multinomial Naive Bayes in terms of precision while maintaining a high recall. This makes SVM the most effective model among the traditional supervised machine learning algorithms tested. SVM managed to correctly identify 586 out of 634 bullying instances with a recall of 0.92 with a precision of 0.43, resulting in the highest F3-score of 0.83. This means it had fewer false positives compared to Logistic Regression, Naive Bayes, and CNN.

The RAG model with Llama 3 achieved the highest recall among all models at 0.97, indicating it rarely misses bullying instances. However, it had the lowest precision at 0.29, generating a significant number of false positives. This suggests that while the RAG model is highly sensitive and effective at identifying bullying instances, its low precision limits its practical applicability due to the high rate of false positives. In comparison to the best traditional model, the SVM model with a recall of 0.92 and precision of 0.43, offers a more balanced performance. The SVM model's higher precision and F3-score of 0.83 make it more reliable for practical use, as it better manages the trade-off between detecting bullying instances and minimizing false positives.

During the project, a significant effort was made to source Swedish bullying data. However, due to the limited availability of Swedish data, no relevant datasets were found. A large cyberbullying dataset in English was then acquired from [68]. This dataset is typical of internet contexts and includes a variety of online interactions. Transformer models were utilized to translate $100'000$ instances into Swedish. This translated dataset was intended to serve as a training set, with the Friends dataset utilized for testing. Unfortunately, the results were underwhelming. The disappointing performance could likely be due to the inaccuracies inherent in machine translation and the mismatch in contexts between bullying in a school environment and cyberbullying online, along with differences in language use and sentiment, which likely contributed to the ineffectiveness of detection models.

As an additional step at the end of the project, OpenAI's newest and most advanced model GPT-4o was tested. It achieved a balanced and high precision of 0.75, recall of 0.94, and highest F3-score of 0.92, outperforming all other models, including the RAG model.

6.2 Findings and Comparison with Related Work

The findings of this study align with previous research in text categorization. The second related work which involved evaluating the effectiveness of various supervised machine learning algorithms for text classification found that SVM scored the highest accuracy of 96.86%, demonstrating its robustness and effectiveness in handling text data. In this study, SVM was the best performer among the traditional models, corroborating the findings that SVM is well-suited for text classification tasks [6].

Additionally, another study focused on implementing SVM for analyzing English text documents found that SVM achieved the best performance with smaller feature sets, reinforcing the suitability of SVM for text classification tasks [69]. While the dataset used in this study and context differed, the consistent performance of SVM across various studies underscores its versatility and reliability.

6.3 Contextual Challenges

Several challenges were noted during this degree project. The Friends dataset comprises survey responses from schoolchildren, but the exact questions are unknown, making it difficult to understand the context fully. Additionally, the labeling criteria used by Friends aren't explicitly clear. There are examples in the dataset labeled as non-bullying, although they show signs of bullying, which could contribute to the high number of false positives observed in some models. For instance, here are some anonymized examples that are labeled as non-bullying, despite indications of bullying behavior:

- `folk kan hoppa på mig och bråka med mig utan anledning`
- `kan bli mobbad för kroppen`
- `För vissa brukar jaga mig`

Furthermore, the dataset is small and imbalanced, highlighting the need for more data to train more robust models.

Moreover, removing stop words resulted in slightly worse performance, suggesting that retaining these common words helps maintain context which is crucial for accurately detecting bullying. Initially, our research into text classification with supervised machine learning indicated that removing stop words was a common and necessary preprocessing step. However, this isn't always the case. According to [70], the relevance of stop words can vary depending on the task. In problems like bullying classification or sentiment analysis, the removal of stop words can negatively impact performance due to the higher sensitivity of these tasks to contextual

nuances. For instance, a task like document classification may not be as affected by stop words removal.

Another significant challenge was the need for powerful computational resources. Access to such resources would have allowed for downloading and running larger, more advanced models locally. This limitation constrained our ability to explore more complex models and potentially achieve better performance.

7 Conclusion

This chapter concludes the report by summarizing the findings, discussing their relevance, answering the hypothesis stated in the introduction, and suggesting areas for future work.

This degree project aimed to identify the most effective supervised machine learning model for classifying bullying in textual data by evaluating a selected subset of models: Logistic Regression, Multinomial Naive Bayes, SVM, and CNN. The most effective model was then compared with a locally implemented RAG model.

Based on the analysis, the SVM emerged as the best-performing traditional model, achieving a precision of 0.43, a recall of 0.92, and the highest F3-score of 0.83. This indicates that SVM effectively balances the classification of bullying instances with minimizing false positives, making it the most reliable model among the traditional algorithms tested.

The RAG implementation with Llama 3, while achieving the highest recall at 0.97, had the lowest precision at 0.29, resulting in a significant number of false positives, and a lower F3-score of 0.79. This highlights its limitations in practical applications despite its sensitivity. The findings indicate that while RAG models can be highly effective in detecting bullying, they need further refinement to improve precision and reduce false positives. These results are probably a reflection of the limited "intelligence" of smaller LLMs at this point in time. Llama 3 is also not specifically trained to solely focus on the Swedish language, which naturally limits its performance in classifying Swedish text.

Additionally, the advanced GPT-4o model was tested on a smaller, anonymized subset of the original dataset, which achieved impressive results with a precision of 0.75 and a recall of 0.94. However, due to its cost, closed-source nature, and inability to handle sensitive data, GPT-4o isn't a viable option for this particular application. The use of only parts of the original data set for its evaluation is also not a completely fair comparison in a controlled experiment. The exploration of GPT-4o, though, underscores the potential of advanced language models in this domain and suggests that with accessible and secure alternatives, similar models could be highly effective. Yet, the success of such models heavily depends on the prompt, which plays a crucial role in guiding it to provide good results.

The hypothesis that LLMs would outperform traditional supervised machine learning models was not conclusively supported by the results. While the advanced GPT-4o model showed promising performance, the locally implemented RAG model with Llama 3 did not surpass the best-performing traditional model, SVM, in terms of the F3-score. This suggests that the effectiveness of LLMs in detecting bullying in text surveys is dependent on factors such as model size, training data, and prompt design.

The results of this study are relevant to both the academic community and anti-bullying initiatives, including organizations like Friends and educational institutions such as schools. They illustrate how machine learning, particularly models like SVM, can be leveraged to address critical social issues such as bullying. The study also

emphasizes the importance of a balanced approach, where both precision and recall are crucial to developing reliable and practical solutions.

7.1 Future work

Future work could include increasing the dataset size with more labeled examples to help train more robust models and reduce the impact of data imbalance. Exploring advanced preprocessing techniques tailored to the Swedish language and the specific context of bullying could also further improve model performance.

Additionally, testing a wider variety of machine learning models, such as Recurrent Neural Networks (RNNs), could provide further insights and possible improvements.

If more powerful computers are available, running advanced local language models, like Llama 3 70B, could boost performance to be closer to that of the GPT-4o results, while keeping it private and local.

Further work could involve gaining better insight into the specific questions that were asked for the respective answers in the Friends dataset, giving the language models more context to base their answers on.

Another area could be to evaluate additional hyperparameters and layers in the CNN model. This was a very time-consuming activity in this project even with the relatively small parameter grid and few layers. More time and better computers could allow for further exploration into this area.

References

[1] M. Loftsson, P. R. Vergara, Y. Christiansson, *et al.*, *Mobbningens förekomst: Tre barn utsatta i varje klass*, Friends. [Online]. Available: https://friends.se/uploads/2022/05/Mobbningens_forekomst-compressed.pdf.

[2] friends, *Friends - united against bullying!*, friends. [Online]. Available: https://friends.se/en/what-friends-does/.

[3] D. Khurana, A. Koli, K. Khatter, and S. Singh, *Natural language processing: State of the art, current trends and challenges*, Multimedia Tools and Applications, Jul. 2022. [Online]. Available: https://link.springer.com/article/10.1007/s11042-022-13428-4.

[4] *Tietoevry - empowering humanity, society and businesses with tietoevry's purposeful tech solutions*, tietoevry AB. [Online]. Available: https://www.tietoevry.com/en/meet-our-people/2023/11/mattias-and-elias-using-ai-and-machine-learning-to-combat-bullying/.

[5] C. Eid and O. Enström, *Bullying detection through graph machine learning: Applying neo4j's unsupervised graph learning techniques to the friends dataset*, Sep. 2023. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:1797968/FULLTEXT01.pdf (visited on 04/01/2024).

[6] I. Dawar, N. Kumar, S. Negi, S. Pathan, and S. Layek, "Text categorization using supervised machine learning techniques", in *2023 Sixth International Conference of Women in Data Science at Prince Sultan University (WiDS PSU)*, 2023, pp. 185–190. DOI: 10.1109/WiDS-PSU57071.2023.00046.

[7] D. Jurafsky and J. H. Martin, *Speech and Language Processing : an Introduction to Natural Language processing, Computational linguistics, and Speech Recognition*, Third Edition draft. Pearson Education, Feb. 2024. [Online]. Available: https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf (visited on 04/16/2024).

[8] A. Mann and O. Höft, *Categorization of swedish e-mails using supervised machine learning*, Jun. 2021. [Online]. Available: https://kth.diva-portal.org/smash/record.jsf?pid=diva2%3A1562606&dswid=-1455 (visited on 05/06/2024).

[9] A. Géron, *Hands-on Machine Learning with Scikit-Learn and TensorFlow : concepts, tools, and Techniques to Build Intelligent Systems*. O'reilly Media, 2017.

[10] *Api reference*, OpenAI. [Online]. Available: https://platform.openai.com/docs/api-reference (visited on 05/16/2024).

[11] L. Borduas-Souvertjis, *How the negative effects of bullying can impact both teens' mental and physical health*, Free Your Mind Initiative, 2024. [Online]. Available: https://freeyourmindinitiative.com/2024/05/17/how-the-negative-effects-of-bullying-can-impact-both-teens-mental-and-physical-health/ (visited on 06/24/2024).

[12] *A complete guide to natural language processing*, DeepLearning.AI, Jan. 2023. [Online]. Available: https://www.deeplearning.ai/resources/natural-language-processing/ (visited on 04/20/2024).

[13] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Cambridge University Press, 2008. [Online]. Available: https://nlp.stanford.edu/IR-book/ (visited on 04/16/2024).

[14] M. G. Singh, *To use or lose: Stop words in nlp*, Medium, Aug. 2023. [Online]. Available: https://medium.com/@gelsonm/to-use-or-lose-stop-words-in-nlp-de946edaa468#:~:text=Stopwords%2C%20while%20it%20looks%20simple (visited on 06/04/2024).

[15] N. Indurkhya and F. J. Damerau, *Handbook of natural language processing*, 2nd Edition. Taylor Francis, 2010.

[16] G. Miner, D. Delen, J. Elder, A. Fast, T. Hill, and R. A. Nisbet, *Practical text mining and statistical analysis for non-structured text data applications*. Academic Press, 2012.

[17] J. Murel and E. Kavlakoglu, *What are stemming and lemmatization? | ibm*, IBM, Dec. 2023. [Online]. Available: https://www.ibm.com/topics/stemming-lemmatization#f03 (visited on 04/16/2024).

[18] *Tfidftransformer*, scikit-learn.org. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html.

[19] M. Chaudhary, *Tf-idf vectorizer scikit-learn*, Medium, Sep. 2020. [Online]. Available: https://medium.com/@cmukesh8688/tf-idf-vectorizer-scikit-learn-dbc0244a911a.

[20] C. Albon, *Machine learning with Python cookbook practical solutions from preprocessing to deep learning*. Cambridge O'reilly, 2018.

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python", *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[22] C. Goyal, *Improve naive bayes text classifier using laplace smoothing*, Analytics Vidhya, Nov. 2023. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/04/improve-naive-bayes-text-classifier-using-laplace-smoothing/ (visited on 05/11/2024).

[23] *Logisticregression*, Scikit-learn.org, 2014. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html (visited on 05/13/2024).

[24] *What is support vector machine? | ibm*, IBM, Dec. 2023. [Online]. Available: https://www.ibm.com/topics/support-vector-machine (visited on 04/23/2024).

[25] S. Soni, S. S. Chouhan, and S. S. Rathore, "Textconvonet: A convolutional neural network based architecture for text classification", *Applied Intelligence*, Oct. 2022. DOI: 10.1007/s10489-022-04221-9. (visited on 11/16/2022).

[26] D. Mwiti, *Nlp essential guide: Convolutional neural network for sentence classification | cnvrg.io*, cnvrg.io, Jun. 2021. [Online]. Available: https://cnvrg.io/cnn-sentence-classification/.

[27] L. Voita, *Convolutional models for text*, lena-voita.github.io, Nov. 2023. [Online]. Available: https://lena-voita.github.io/nlp_course/models/convolutional.html#parameters (visited on 05/22/2024).

[28] P. A. Flach, *Machine Learning : The Art and Science of Algorithms That Make Sense of Data*. Cambridge University Press, 2012.

[29] H. Allamy, "Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study)", Dec. 2014. [Online]. Available: https://www.researchgate.net/publication/295198699_METHODS_TO_AVOID_OVER-FITTING_AND_UNDER-FITTING_IN_SUPERVISED_MACHINE_LEARNING_COMPARATIVE_STUDY (visited on 05/10/2024).

[30] J. Lever, M. Krzywinski, and N. Altman, "Model selection and overfitting", *Nature Methods*, vol. 13, pp. 703–704, Aug. 2016. DOI: 10.1038/nmeth.3968. (visited on 04/08/2020).

[31] A. Bhande, *What is underfitting and overfitting in machine learning and how to deal with it.* Medium, Mar. 2018. [Online]. Available: https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76 (visited on 07/11/2020).

[32] *What is overfitting?*, IBM. [Online]. Available: https://www.ibm.com/topics/overfitting (visited on 05/10/2024).

[33] S. Mutasa, S. Sun, and R. Ha, "Understanding artificial intelligence based radiology studies: What is overfitting?", *Clinical Imaging*, vol. 65, pp. 96–99, 2020. DOI: https://doi.org/10.1016/j.clinimag.2020.04.025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0899707120301376.

[34] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning : Data Mining, Inference, and Prediction, Second Edition*. Springer, 2009, pp. 261–294. [Online]. Available: https://hastie.su.domains/Papers/ESLII.pdf (visited on 05/07/2024).

[35] *Auto classifier node (spss modeler)*, IBM, Jan. 2024. [Online]. Available: https://www.ibm.com/docs/en/cloud-paks/cp-data/4.8.x?topic=modeling-auto-classifier-node (visited on 05/07/2024).

[36] M. Hossin and S. M.N, "A review on evaluation metrics for data classification evaluations", *International Journal of Data Mining Knowledge Management Process*, vol. 5, pp. 01–11, Mar. 2015. DOI: 10.5121/ijdkp.2015.5201. [Online]. Available: https://www.nature.com/articles/nmeth.3945.

[37] D. Chicco and G. Jurman, "The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation", *BMC Genomics*, vol. 21, Jan. 2020. DOI: 10.1186/s12864-019-6413-7. [Online]. Available: https://link.springer.com/article/10.1186/s12864-019-6413-7#citeas (visited on 04/24/2024).

[38] M. Hossin and S. M.N, "A review on evaluation metrics for data classification evaluations", *International Journal of Data Mining Knowledge Management Process*, vol. 5, pp. 01–11, Mar. 2015. DOI: 10.5121/ijdkp.2015.5201. [Online]. Available: https://www.researchgate.net/publication/275224157_A_Review_on_Evaluation_Metrics_for_Data_Classification_Evaluations.

[39] P. A. Flach, *Machine learning : the art and science of algorithms that make sense of data*. Cambridge University Press, 2012, pp. 49–80.

[40] D. Olson and D. Delen, *Advanced Data Mining Techniques*. Jan. 2008, pp. 137–147, ISBN: 978-3-540-76916-3. DOI: 10.1007/978-3-540-76917-0. [Online]. Available: https://www.researchgate.net/publication/220695151_Advanced_Data_Mining_Techniques.

[41] R. Muhamedyev, K. Yakunin, S. Iskakov, S. Sainova, A. Abdilmanova, and Y. Kuchin, "Comparative analysis of classification algorithms", in *2015 9th International Conference on Application of Information and Communication Technologies (AICT)*, 2015, pp. 96–101. DOI: 10.1109/ICAICT.2015.7338525. [Online]. Available: https://ieeexplore.ieee.org/document/7338525.

[42] B. Carterette, "Precision and recall", in *Encyclopedia of Database Systems*, L. LIU and M. T. ÖZSU, Eds. Boston, MA: Springer US, 2009, pp. 2126–2127, ISBN: 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_5050. [Online]. Available: https://link.springer.com/referenceworkentry/10.1007/978-0-387-39940-9_5050.

[43] J. Murel and E. Kavlakoglu, *What is a confusion matrix? | ibm*, IBM, Jan. 2024. [Online]. Available: https://www.ibm.com/topics/confusion-matrix (visited on 04/24/2024).

[44] D. Steen, *Beyond the f-1 score: A look at the f-beta score*, Medium, Oct. 2020. [Online]. Available: https://medium.com/@douglaspsteen/beyond-the-f-1-score-a-look-at-the-f-beta-score-3743ac2ef6e3.

[45] K. Bondarenko, *Precision and recall in recommender systems. and some metrics stuff.* Medium, Feb. 2019. [Online]. Available: https://bond-kirill-alexandrovich.medium.com/precision-and-recall-in-recommender-systems-and-some-metrics-stuff-ca2ad385c5f8.

[46] *Introducing meta llama 3: The most capable openly available llm to date*, ai.meta.com, Apr. 2024. [Online]. Available: https://ai.meta.com/blog/meta-llama-3/.

[47] *Hello gpt-4o*, OpenAI, May 2024. [Online]. Available: https://openai.com/index/hello-gpt-4o/ (visited on 05/16/2024).

[48] F. M. Aguilera, *Temperature in the context of a large language model*, Medium, May 2024. [Online]. Available: https://ai.plainenglish.io/temperature-in-the-context-of-a-large-language-model-7966938c35a0 (visited on 05/22/2024).

[49] *What is rag? - retrieval-augmented generation explained - aws*, Amazon Web Services, Inc. [Online]. Available: https://aws.amazon.com/what-is/retrieval-augmented-generation/#:~:text=RAG%20allows%20developers%20to%20provide.

[50] H. Face, *Hugging face - ai community buidling the future*, huggingface.co. [Online]. Available: https://huggingface.co/.

[51] T. pandas development team, *Pandas-dev/pandas: Pandas*, version latest, Feb. 2020. DOI: 10.5281/zenodo.3509134. [Online]. Available: https://doi.org/10.5281/zenodo.3509134.

[52] P. S. Foundation, *Re — regular expression operations — python 3.12.3 documentation*, Python.org, 2009. [Online]. Available: https://docs.python.org/3/library/re.html.

[53] *Lemmy, v2.1.0*, Python Package Index (PyPI), 2019. [Online]. Available: https://pypi.org/project/lemmy/ (visited on 04/25/2024).

[54] Explosion, *Spacy, v3.0*, ExplosionAI GmbH, Berlin, 2021. [Online]. Available: https://spacy.io/ (visited on 04/25/2024).

[55] J. Bolton, C. D. Manning, P. Qi, Y. Zhang, and Y. Zhang, "Stanza: A python natural language processing toolkit for many human languages", in *Proceedings of the Association for Computational Linguistics (ACL) System Demonstrations*, 2020. [Online]. Available: https://arxiv.org/abs/2003.07082 (visited on 04/25/2024).

[56] S. N. Group, *Ner models*, Stanza, 2020. [Online]. Available: https://stanfordnlp.github.io/stanza/ner_models.html (visited on 04/25/2024).

[57] Språkbanken, *Suc 3.0 | språkbanken text*, spraakbanken.gu.se, Mar. 2024. [Online]. Available: https://spraakbanken.gu.se/en/resources/suc3 (visited on 04/25/2024).

[58] H. Dedhia, *Stop words in 28 languages*, www.kaggle.com, Sep. 2020. [Online]. Available: https://www.kaggle.com/datasets/heeraldedhia/stop-words-in-28-languages (visited on 04/25/2024).

[59] G. Diaz, *Stopwords-iso/stopwords-sv*, GitHub, Jan. 2023. [Online]. Available: https://github.com/stopwords-iso/stopwords-sv?tab=readme-ov-file (visited on 05/10/2024).

[60] NVIDIA, *Cuda toolkit*, NVIDIA Developer, Jul. 2013. [Online]. Available: https://developer.nvidia.com/cuda-toolkit (visited on 05/02/2024).

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

[62] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning", *Journal of Machine Learning Research*, vol. 18, no. 17, pp. 1–5, 2017. [Online]. Available: http://jmlr.org/papers/v18/16-365.html.

[63] F. Chollet *et al.*, *Keras*, https://keras.io, 2015.

[64] *Gensim: Topic modelling for humans*, radimrehurek.com. [Online]. Available: https://radimrehurek.com/gensim/.

[65] *The ai-native open-source embedding database*, www.trychroma.com. [Online]. Available: https://www.trychroma.com/.

[66] *lm studio - discover and run local llms*, lmstudio.ai. [Online]. Available: https://lmstudio.ai/.

[67] *Openai python library*, OpenAI. [Online]. Available: https://platform.openai.com/docs/libraries/python-library (visited on 05/15/2024).

[68] N. Ejaz, S. Choudhury, and F. Razi, "A comprehensive dataset for automated cyberbullying detection", *data.mendeley.com*, vol. 2, Jan. 2024. [Online]. Available: https://data.mendeley.com/datasets/wmx9jj2htd/2 (visited on 05/21/2024).

[69] J. Philip, B. VeerasekharReddy, M. Harshini, I. Haritha, S. Patil, and S. Shareef, "A comparative study of text classification using selective machine learning algorithms", in *2023 7th International Conference on Intelligent Computing and Control Systems (ICICCS)*, 2023, pp. 482–484. [Online]. Available: https://ieeexplore.ieee.org/document/10142474 (visited on 05/18/2024).

[70] Wilame, *Why is removing stop words not always a good idea*, Medium, Mar. 2023. [Online]. Available: https://medium.com/@wila.me/why-is-removing-stop-words-not-always-a-good-idea-c8d35bd77214 (visited on 05/19/2024).

A Appendix 1

The source code for this degree project is available in the GitHub repository at `https://github.com/sw0rdd/BullyingDetection`.