Comparing Sentiment Analysis Tools on GitHub Project Discussions

Lucas Guedes Barboza - 43515

Thesis presented to the **Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Bragança** to obtain the Master of Science Degree in **Informatics** within the scope of the double diploma with the **Universidade Tecnológica Federal do Paraná**.

Supervised by:

Prof. Dr. Rui Pedro Lopes (ESTiG-IPB)

Prof. Dr. Ivanilton Polato (UTFPR)

Bragança

2022-2023

Comparing Sentiment Analysis Tools on GitHub Project Discussions

Lucas Guedes Barboza - 43515

Thesis presented to the **Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Bragança** to obtain the Master of Science Degree in **Informatics** within the scope of the double diploma with the **Universidade Tecnológica Federal do Paraná**.

Supervised by:

Prof. Dr. Rui Pedro Lopes (ESTiG-IPB)

Prof. Dr. Ivanilton Polato (UTFPR)

Bragança

2022-2023

Dedication

I dedicate this achievement to everyone whom I have taken away time spent together to fulfill this journey.

Acknowledgment

First and foremost, I would like to express my gratitude to my professors and advisors who guided and shaped my academic journey. To my advisor Rui Pedro Lopes, your expertise and guidance have been indispensable throughout this process. To my co-advisor Ivanilton Polato, I extend my appreciation for the invaluable contributions and availability throughout the entire work, even from a distance. To my parents, Edson e Elaine, who devoted their lives to nurturing the person I have become today. Your constant support and encouragement throughout my life are the foundation of my achievements. A special thank you goes to my girlfriend, Mariana, who accompanied me on this entire journey, offering constant love and support to make the completion of this work possible. I would like to express my gratitude to the entire UTFPR community, which has been tremendously influential in my education and has enabled my participation in the Dual Diplomacy program. I am also grateful to the entire IPB community, which warmly embraced the UTFPR students, providing us with a highly enriching experience both professionally and culturally. I would also like to sincerely thank my psychologist, Hellen, whose professional assistance and valuable insights were of immense importance for the progress of this dissertation. Your guidance has contributed to the development of skills to cope with stress and life's adversities. To my mother-in-law, Ana, whose motivation played a pivotal role in guiding me towards pursuing a career in IT. Thanks to my work colleague, João Gris, for constantly checking up on the progress of this work and providing words of encouragement throughout the entire process. To my friends from the fraternity in Campo Mourão: Angelo, Gabriel, Gustavo, Marcelo, João Biffe, João Marcos, Vinicius, and to my flatmates in Bragança: Marcelo, João Velozo, João Lucas, Thalles,

Felipe, Sthefanny, and Gleidson. Thank you for the cherished moments we shared and the companionship regardless of the circumstances. Finally, I would like to express my gratitude to everyone who, in one way or another, played a role in this achievement.

Without all of you, this accomplishment could not have been realized.

To each and every one of you, my sincerest gratitude.

Abstract

The context of this work is situated in the rapidly evolving sphere of Natural Language Processing (NLP) within the scope of software engineering, focusing on sentiment analysis in software repositories. Sentiment analysis, a subfield of NLP, provides a potent method to parse, understand, and categorize these sentiments expressed in text. By applying sentiment analysis to software repositories, we can decode developers' opinions and sentiments, providing key insights into team dynamics, project health, and potential areas of conflict or collaboration. However, the application of sentiment analysis in software engineering comes with its unique set of challenges. Technical jargon, code-specific ambiguities, and the brevity of software-related communications demand tailored NLP tools for effective analysis.

The study unfolds in two primary phases. In the initial phase, we embarked on a meticulous investigation into the impacts of expanding the training sets of two prominent sentiment analysis tools, namely, SentiCR and SentiSW. The objective was to delineate the correlation between the size of the training set and the resulting tool performance, thereby revealing any potential enhancements in performance.

The subsequent phase of the research encapsulates a practical application of the enhanced tools. We employed these tools to categorize discussions drawn from issue tickets within a varied array of Open-Source projects. These projects span an extensive range, from relatively small repositories to large, well-established repositories, thus providing a rich and diverse sampling ground.

Keywords: natural language processing, sentiment analysis, software engineering, opinion mining.

Resumo

O contexto deste trabalho situa-se na esfera em rápida evolução do Processamento de Linguagem Natural (PLN) no âmbito da engenharia de software, com foco na análise de sentimentos em repositórios de software. A análise de sentimentos, um subcampo do PLN, fornece um método poderoso para analisar, compreender e categorizar os sentimentos expressos em texto. Ao aplicar a análise de sentimentos aos repositórios de software, podemos decifrar as opiniões e sentimentos dos desenvolvedores, fornecendo informações importantes sobre a dinâmica da equipe, a saúde do projeto e áreas potenciais de conflito ou colaboração. No entanto, a aplicação da análise de sentimentos na engenharia de software apresenta desafios únicos. Jargão técnico, ambiguidades específicas do código e a brevidade das comunicações relacionadas ao software exigem ferramentas de PLN personalizadas para uma análise eficaz.

O estudo se desenvolve em duas fases principais. Na fase inicial, embarcamos em uma investigação meticulosa sobre os impactos da expansão dos conjuntos de treinamento de duas ferramentas proeminentes de análise de sentimentos, nomeadamente, SentiCR e SentiSW. O objetivo foi delinear a correlação entre o tamanho do conjunto de treinamento e o desempenho da ferramenta resultante, revelando assim possíveis aprimoramentos no desempenho.

A fase subsequente da pesquisa engloba uma aplicação prática das ferramentas aprimoradas. Utilizamos essas ferramentas para categorizar discussões retiradas de bilhetes de problemas em uma variedade diversificada de projetos de código aberto. Esses projetos abrangem uma ampla gama, desde repositórios relativamente pequenos até repositórios

grandes e bem estabelecidos, fornecendo assim um campo de amostragem rico e diversificado.

Palavras-chave: processamento de linguagem natural, análise de sentimentos, engenharia de software, mineração de opinião.

Contents

1 Introduction **1**

1.1 Problem Statement 2

1.2 Objetives 2

1.3 Structure of the document 3

2 Context and Technologies **4**

2.1 Natural Language Processing 4

2.2 Machine Learning 5

2.2.1 Supervised Learning 6

2.2.2 Unsupervised Learning 7

2.2.3 Semi-Supervised Learning 8

2.2.4 Reinforcement Learning 9

2.3 Mining Software Repositories 10

2.4 Sentiment Analysis 11

2.4.1 Levels of Sentiment Analysis 12

2.4.2 Sentiment Analysis Approaches 14

2.4.3 SentiCR 15

2.4.4 SentiSW 16

2.5 Feature Extraction 16

2.5.1 Bag-of-Words 17

2.5.2 Word2Vec 18

2.5.3 Doc2Vec . . . 19
2.5.4 TF-IDF . . . 21
2.6 Evaluation and Performance metrics . . . 22
2.7 Related Work . . . 26

3 Method **29**
3.1 Research Questions . . . 29
3.2 First Study: Context in Sentiment Analysis . . . 30
3.2.1 Data Augmentation . . . 30
3.2.2 SentiCR+ dataset generation . . . 31
3.2.3 SentiSW+ dataset generation . . . 32
3.2.4 Performance Evaluation . . . 32
3.3 Second Study: Applied Sentiment Analysis in GitHub OSS projects . . . 33
3.3.1 Data collection . . . 33
3.3.2 Repository selection . . . 34
3.3.3 Issue comment metadata extraction . . . 35
3.3.4 Filtered issue population sizes . . . 35
3.3.5 Comment classification . . . 37
3.4 Evaluation/Confirmation . . . 37

4 Discussion and Results **39**
4.1 Analysis . . . 39
4.2 Threats to Validity . . . 47

5 Conclusion **51**
5.1 Future Work . . . 52

List of Tables

3.1 Population sizes by range 35
3.2 Target population issue count 36
3.3 Issues with English corpus 36

4.1 Precision, recall, F1-score metrics for macro-average. 40
4.2 SentiCR precision, recall, F1-score metrics for macro-average. 40
4.3 Precision, recall, F1-score metrics for macro-average. 41
4.4 Cohen Kappa Score between the 4 models 41
4.5 Mean and standard deviations across Python project sizes. 44
4.6 Mean and standard deviations across Java project sizes. 45
4.7 Mean and standard deviations across Javascript project sizes. 46

List of Figures

2.1 Machine Learning categories 6
2.2 Supervised Learning. Source: [15] 7
2.3 Unsupervised Learning. Source: [15] 8
2.4 Reinforcement Learning 9
2.5 SA levels. Source: 12
2.6 Sentiment Analysis techniques. 14
2.7 Bag-of-Words 17
2.8 Word2Vec models: The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word. 18
2.9 Doc2Vec models: Distributed Memory (DM) and Distributed Bag of Words (DBoW). 20
2.10 ROC Curve 25

3.1 SentiCR+ generation process 31
3.2 SentiSW+ generation process 32
3.3 Data collection flowchart 34

4.1 Sentiment Distribution amongst Programming Languages 43
4.2 Python Sentiment Distribution by Category 43
4.3 Java Sentiment Distribution by Category. 44
4.4 Javascript Sentiment Distribution by Category. 45

Acronyms

AI Artificial Intelligence. 9

BoW Bag-of-Words. 12, 15, 17

CBOW Continuous Bag of Words. 18, 19

DBoW Distributed Bag of Words. xvi, 19, 20

DM Distributed Memory. xvi, 19, 20

FN False Negative. 23

FP False Positive. 23, 24

FPR False Positive Rate. 24

GBT Grading Boosting Tree. 16

MDP Markovian Decision Process. 9

MSR Mining Software Repositories. 10

NLP Natural Language Processing. 4, 5, 11, 12, 26, 39

OCR Optical Character Recognition. 4

OSS Open-Source Software. 1, 3, 51

PCA Principal Component Analysis. 7

RL Reinforcement Learning. 9, 10

ROC Receiver Operator Characteristic. 24

SA Sentiment Analysis. 1, 11, 29, 37, 39, 42

SE Software Engineering. 16

SG Skip-Gram. 18, 19

SL Supervised Learning. 6

SSL Semi-Supervised Learning. 8

TF-IDF Term Frequency-Inverse Document Frequency. 15, 16, 21, 22

TN True Negative. 23

TP True Positive. 23, 24

TPR True Positive Rate. 24

UL Unsupervised Learning. 7

Chapter 1

Introduction

Over the past years, an increasingly amount of attention is being drawn towards capturing traces of emotions and sentiments experienced and expressed by software developers in collaborative working enviroments. Such studies explore the social aspects of software engineering and are becoming an important field of research.

However, as pointed by Rishi [1], due to Open-Source Software (OSS) projects often rely on geographically distributed team settings and voluntary contributions, traditional approaches of capturing emotions become a challenging task, as these direct observations tempers the developers from their natural workflow frequently. Thus, a non-intrusive approach such as detecting sentiments from software engineering textual artifacts could be a supplementative alternative to traditional approaches. With this background, we apply Sentiment Analysis (SA) to several software development ecosystems in this study.

Social coding platforms (e.g., GitHub) provide a high level of transparency by displaying a multitude of visible cues on individual and project public profile pages [2] - in particular, project related activities like the number of contributors, issues, issue comments, and code reviews. The availability of such materials facilitates to study the sentiments expressed during development activities.

Ortu et al. [3], highlights the impact of human affectivity in terms of productivity, in which sentiment analysis and form of correspondence - be it polite or not - can be useful to monitor and identify the mood of the team. Thus, allowing team leaders to articulate,

bringing factors that motivate serenity in the community to resolve possible threats to productivity.

1.1 Problem Statement

Sentiment in the workplace plays an important role in collaborative work. With the rising trend of distributed development teams, software engineers are frequently using a variety of social media tools such as mailing lists, forums, software code repositories and issue tracking tools to monitor and manage their collaborations [4].

As emphasized by Dabbish et al. [5], the leveraging of such technologies in the software engineering context support some level of social awareness previous only to collocated teams. As visible information about other developers' actions influenced perceptions of their commitment and general interests

Previous research has indicated that user interactions on the aforementioned communication channels contain notable amounts of emotions. The expression of these sentiments could impact the efficiency of the software development process both positively and negatively [6], [7].

In this study, we put effort towards investigating sentiment distribution in textual artifacts of GitHub projects.

1.2 Objetives

The main objective of this work is to explore the impact on performance caused by the expansion of a train dataset between two sentiment analysis tools. This objective is accomplished mainly by means of experimentation with different train data to derive a variation for the selected models and improve accuracy.

In addition to the main objective, a secondary objective is the retrieval and analysis of issues from source code repositories (GitHub) towards extracting sentiment polarity within open source projects and provide emotional insights about projects. The datasets

will be extracted from a sample of OSS projects hosted on GitHub. The data from all sets will be synthesised to establish if correlation points exist between the research questions.

1.3 Structure of the document

This document is structured into 5 chapters, starting with this introduction, where the context and objectives are laid down. Chapter 2 follows, with a review of concepts and related bibliography. Chapter 3 details the main methodology process developed in this dissertation. Chapter 4 comes next, with an exhibition of the results gathered and discusses research findings. Finally, it ends in Chapter 5, with conclusions, final remarks, additional findings, and future work.

Chapter 2

Context and Technologies

This chapter aims to describe the current most relevant technologies, tools and methods related to the main topics of this project. It begins by laying down the scientific areas of our study: Natural Language Processing, Machine Learning, Sentiment Analysis and Opinion Mining. Then describes the two objects of our research: SentiCR and SentiSW, as well as the Evaluation and Performance metrics used to measure the performance.

2.1 Natural Language Processing

Natural Language Processing (NLP) is an interdisciplinary field with connections to computer science, linguistics, and artificial intelligence. It is interested in the interactions between human (natural) languages and computers and serves as a building block for applications of speech recognition and synthesis, machine translation, Optical Character Recognition (OCR), Sentiment Analysis (SA), question answering, and dialogue systems [8].

The term NLP involves a wide set of techniques to address automated generation, manipulation, and analysis of written and spoken human language [9]. It is a way for computers to analyze, understand, and derive meaning from human language in a smart and useful way [10]. In the literature of [11], the author suggests the following definition:

"Natural Language Processing is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications."

But unlike the traditional ways of humans interacting with computers via highly structured programming languages, the task to convert information stored in natural language to a machine-understandable format becomes challenging [10]. As introduced by the authors, human speech is not precise, it may be ambiguous and the linguistic structure may depend on different complex variables, including regional dialects, slang, and social context. Difficulties arise when longer sentences that are highly ambiguous and have complex grammar are to be processed [12].

To overcome these challenges, there have been ongoing research efforts on developing NLP systems varying in employed techniques and purposes. The processing tasks performed by an NLP consist of several layers ranging from the phonology and morphology of elements to the lexical, syntactic, and semantic aspects of a text, to the discourse and pragmatic properties of natural language text [12].

2.2 Machine Learning

Machine learning is a multidisciplinary field that involves computer science, engineering, and statistics. Its primary goal is to detect meaningful patterns in data and extract useful information. Machine learning is necessary when a programmer cannot provide an explicit and detailed specification of the execution process due to the complexity of patterns to be studied. The logic behind machine learning enables models to learn and improve continuously, making them adaptable to complex problems [13]. The field attempts to replicate humans' ability to recognize patterns using computation, and it is particularly useful when the dataset is large or complex and requires automation for reproducible and time-efficient analysis [14].

The learning algorithms can be categorized into four major types, such as Supervised,

Unsupervised, Semi-Supervised, and Reinforcement Learning (Figure 2.1). In the following, we briefly discuss each type of learning technique with the scope of their applicability to solve real-world problems.

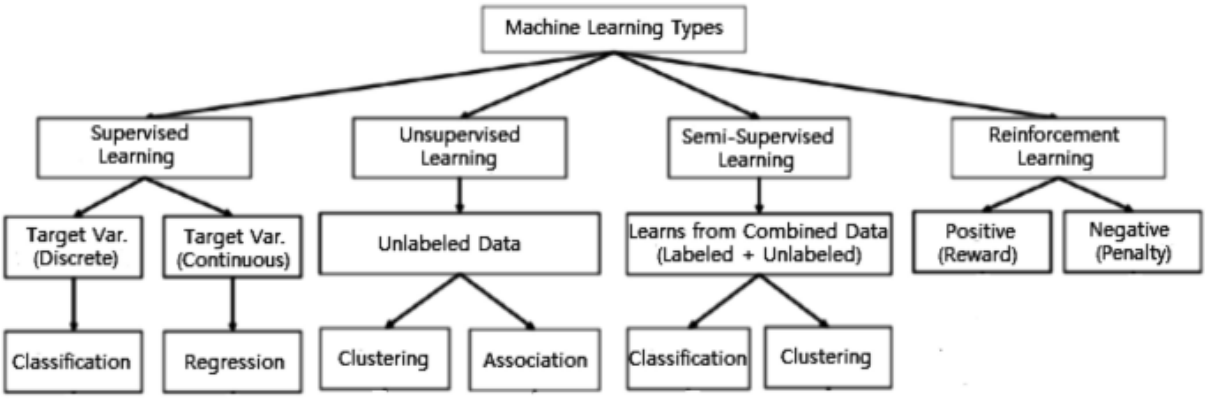

Figure 2.1: Machine Learning categories

2.2.1 Supervised Learning

Supervised Learning (SL) is a branch of machine learning where a model is tutored using a dataset enriched with pre-existing labels, with the aim of enabling the model to generate forecasts on novel, unseen data. This pre-annotated dataset encompasses input data (termed as features) and the corresponding output data (referred to as labels or targets).

The learning process during the model's training phase involves creating an association between the input data and their respective output data. This is accomplished by adopting an iterative approach to minimize the discrepancy between the model's predicted output and the actual output. Once the model is adequately trained, it acquires the capability to provide predictions on unseen input data (Refer to Figure 2.2).

The methodology of SL can be demarcated into two distinct subcategories, namely, classification and regression. The classification paradigm involves training the model to predict a categorical label for each data point in the input set. On the other hand, regression involves training the model to predict a continuous value corresponding to each data point in the input set.

To illustrate, classification problems are exemplified by tasks such as discerning between spam and non-spam emails or categorizing different objects within an image. Conversely, regression problems can be seen in applications like predicting stock market fluctuations or forecasting meteorological changes.

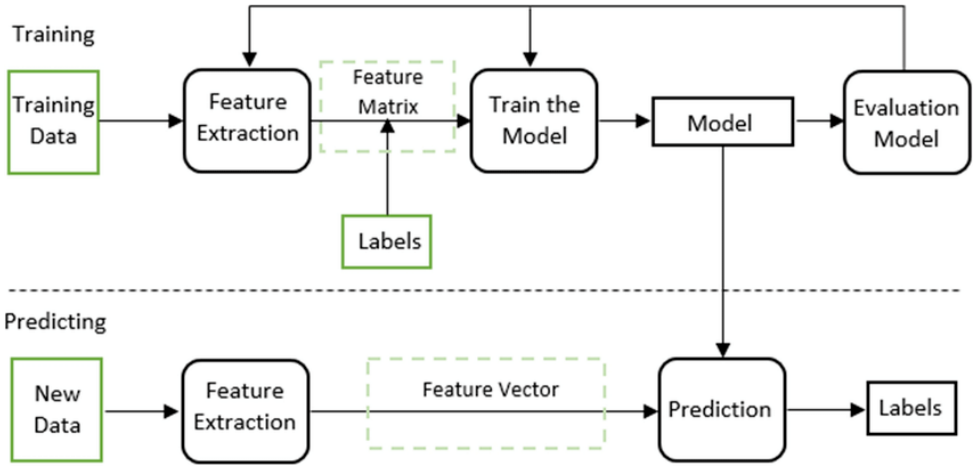

Figure 2.2: Supervised Learning. Source: [15]

2.2.2 Unsupervised Learning

Unsupervised Learning (UL) is a machine learning technique that utilizes unlabeled data to train the algorithm without explicit target outputs or environmental evaluations associated with each input. This technique aims to represent input patterns in a way that reflects the statistical structure of the overall collection of input patterns, based on prior biases as to what aspects of the structure of the input should be captured in the output [16]. By analyzing input data, the algorithm can mine out rules and regulations and detect relevant patterns without the need for labeled information. Unsupervised learning includes clustering and association models, such as the K-means algorithm and Principal Component Analysis (PCA), which identify common aspects in the same group(s) without a specific requirement of labeled information [17]. Overall, unsupervised learning is a powerful tool for discovering meaningful patterns and structures in unlabeled data sets (Figure 2.3).

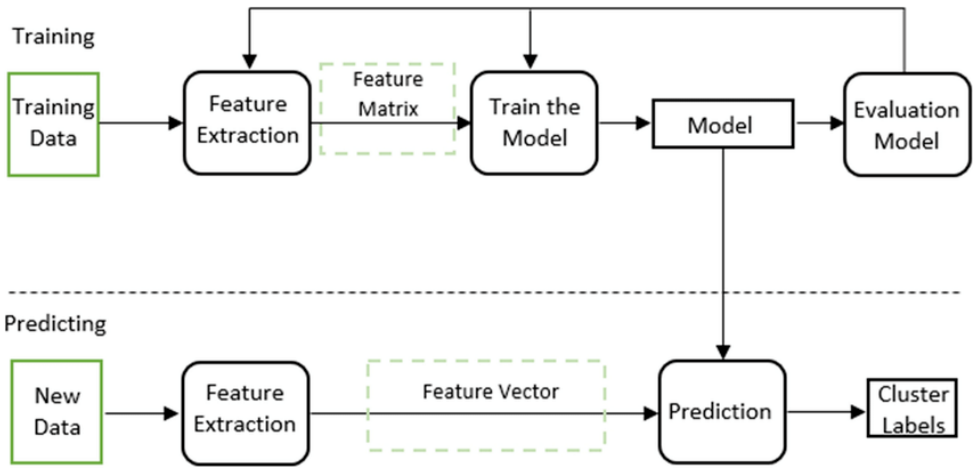

Figure 2.3: Unsupervised Learning. Source: [15]

2.2.3 Semi-Supervised Learning

Semi-Supervised Learning (SSL) is a method that combines supervised and unsupervised learning to avoid their limitations and take advantage of their benefits [18]. While supervised learning requires a substantial amount of labeled training data, which can be expensive and time-consuming to acquire, unsupervised learning can cluster data based on similarity without relying on labeled data. However, this approach is not optimal and it cannot accurately cluster unknown data [19].

To address these challenges, the goal of SSL is to use the labeled data to learn a predictive function for the unlabeled data. To achieve this, assumptions for smoothness, cluster and manifold are made so that label information can be delivered to unlabeled samples [18]. By using only a few labeled patterns as training data, SSL can build a model that can generalize to the rest of the patterns treated as test data. SSL can be further categorized into two types, namely, semi-supervised classification and semi-supervised clustering [19].

This approach is particularly useful in scenarios where it is difficult or expensive to access or measure the target variable for all participants. For instance, longitudinal studies that require several years of follow-up to obtain a reliable disease label can be impractical or prohibitively expensive. SSL can be used in such cases to model existing labeled data,

as well as unlabeled data, in order to maximize the amount of input data and make the most of finite resources and time [20].

2.2.4 Reinforcement Learning

Reinforcement Learning (RL) is a subdomain of Artificial Intelligence (AI) and a distinct approach to machine learning that focuses on goal-oriented algorithms [21], [22]. This methodology involves training an agent, an entity that interacts with its environment, to complete a specific task without requiring separate training and inference phases [22]. RL is built on a trial-and-error strategy, where the agent learns from past interactions with the environment and refines its behavior to achieve desired objectives [23]. RL algorithms fundamentally rely on reward functions, which serve as feedback to the learning algorithm regarding the consequences of the recently executed action [24].

The primary goal of an RL agent is to identify an optimal policy that associates states with appropriate actions, maximizing the rewards obtained from these actions [23]. These rewards, or reinforcements, serve as quantitative assessments of the agent's most recent actions and are represented by positive or negative values [22]. The environment can be modeled as a Markovian Decision Process (MDP) and is typically represented by a 4-tuple <S, A, P, R>, consisting of the state set (S), action set (A), state transition probability matrix (P), and reward function (R) [24]. The Figure 2.4 illustrates the action-reward feedback loop of a generic RL model.

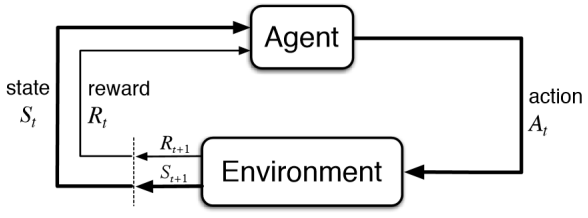

Figure 2.4: Reinforcement Learning

There are two primary approaches to RL problems: model-based and model-free. Model-based approaches involve the agent learning the environment model and refining

its policy to achieve optimality, while model-free approaches focus on policy improvement without acquiring knowledge about the environment model. Model-based methods tend to learn more rapidly than model-free approaches due to their ability to reuse stored information, but they are less prevalent owing to their increased storage requirements and dependency on the initial environment model's accuracy [24].

In recent years, RL applications have experienced growing popularity in diverse domains such as robotics, Internet of Things (IoT), power management, financial trading, and telecommunications. Due to its iterative trial-and-error approach, RL is considered to be very similar to human learning, making it an effective tool for addressing problems in which the operating environment is either unknown or subject to change [22].

2.3 Mining Software Repositories

Mining Software Repositories (MSR) is a field of research that focuses on analyzing data generated throughout the software development process to comprehend valuable and practical insights for software engineering [25]. The information sources analyzed by MSR include source code repositories, version control systems metadata, issue reports, bug tracking systems, and project discussion lists [26].

MSR contributes significantly to understanding the past, present, and predicting the future characteristics of software maintenance [26]. Researchers use historical and valuable information stored in software repositories to monitor complex projects and products without interfering with development activities. For example, the information can assist developers in understanding the rationale for the current structure of the software system [26].

One of the most used sources of information in MSR is version control systems such as Git, as they provide precise information about the source code, its evolution, and the developers of the software [25]. However, extracting information from Git repositories can be difficult due to the encapsulation of all Git features in the available tools. As a result, developers may have to write long and complex implementations to extract even simple

data from a Git repository [25].

Despite the challenges, the rich data in software repositories has enabled data-driven studies of social systems, including those in areas like computational social science, social network analysis, organizational theory, and management science. The commit history of developers allows for the construction of social networks that can proxy collaboration, coordination, or communication structures in software teams. Furthermore, the detailed record of file modifications in the commit log of Git repositories enables more advanced network reconstruction techniques that can provide insights beyond more coarse-grained definitions [27].

2.4 Sentiment Analysis

SA, also known as opinion mining, is a computational approach to studying individuals' sentiments, emotions, evaluations, and attitudes towards various entities such as products, services, organizations, individuals, events, and topics, among others [28]–[30]. The field has grown rapidly with the expansion of social media platforms, which generate an unprecedented volume of opinionated data in digital formats [28], [29]. Sentiment analysis is a prominent research area within NLP and has applications in various disciplines, including management, marketing, finance, political science, communications, health science, and history [28].

Despite its importance, challenges in sentiment analysis arise from the informal text generated by users on social media platforms, including spelling errors, new slang, or abbreviations, and incorrect grammar usage. Other challenges include the scarcity of resources, such as large annotated datasets or resources in languages other than English, the evolving dictionary of web slang, and the difficulty in detecting sarcasm and irony. Furthermore, identifying multiple sentiments in a single sentence and determining polarity from comparative sentences also present significant challenges [29].

2.4.1 Levels of Sentiment Analysis

Sentiment analysis has been studied extensively across multiple levels. However, identifying sentiments and opinions is primarily done at three levels of granularity: document, sentence, or aspect level [28]. The levels of sentiment analysis are illustrated in Figure 2.5, with the first two levels being particularly interesting and highly challenging. The third level is even more difficult as it involves a detailed investigation [31].

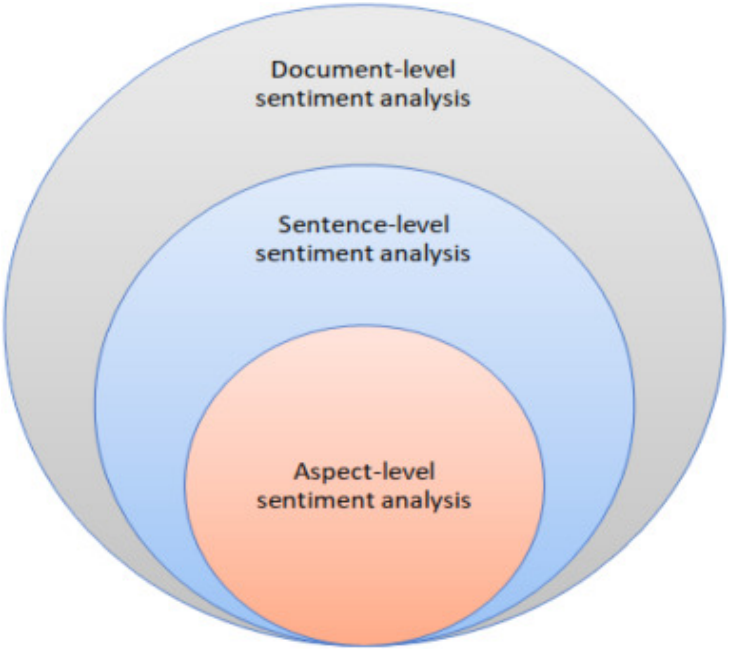

Figure 2.5: SA levels. Source:

A brief presentation of each level is as follows.

Document-Level

Document level sentiment classification involves determining the overall sentiment conveyed by an entire document or record, which can be a binary classification task (positive or negative), a regression task (e.g., inferring a rating score), or a multi-class classification task. The Bag-of-Words (BoW) model is traditionally employed to generate text representations in NLP and text mining, transforming a document into a numeric feature vector with a fixed length [28].

This level requires a deep comprehension of the intricate relationships between words and phrases and the full context of semantic information [29]. This classification works best when a document is authored by a single person and focuses on a single entity. However, conflicting sentiments within the document can impact the final decision [30].

Sentence-Level

At this level, the objective is to categorize individual sentences within a document based on whether they express positive, negative, or neutral opinions [28], [30]. To accomplish this, a preliminary step of subjectivity classification may be performed to determine whether a sentence is opinionated or not. Neural networks are commonly used to generate sentence representations for classification, and additional syntactic and semantic information may be considered to enhance accuracy [28]. Essentially, sentence-level sentiment analysis dissects documents or paragraphs into individual sentences to determine the polarity of each sentence [29]. To do this, sentences are classified as either objective, expressing factual information, or subjective, expressing opinions and views [30].

Aspect-Level

Aspect-level sentiment analysis, also known as aspect-based sentiment analysis, is a detailed approach that focuses on extracting and summarizing opinions expressed about specific aspects or features of entities, referred to as targets. This process involves several subtasks, including aspect extraction, entity extraction, and aspect sentiment classification [28].

In contrast to document and sentence level sentiment classification, aspect level classification considers both sentiment and target information [28]. This level of sentiment analysis identifies opinions about specific features of entities, which can be explicit or implicit, rather than the overall sentiment of sentences or paragraphs [29], [30].

This task is challenging due to the complexity of modeling the semantic relationships between the target and its surrounding context words [28]. Thus, making this level more

challenging than the other two. Neural network-based learning models are utilized to capture the semantic connections between the target word and the context words [28].

2.4.2 Sentiment Analysis Approaches

The existing approaches for sentiment analysis can be categorized based on various points of view (e.g. a view of the text, level of detail of text analysis). However, most literature usually divides sentiment analysis approaches into three categories: Machine Learning approaches, Lexicon-Based approaches, and Hybrid approaches. Machine learning is the most widely-used approach. It relies on machine learning algorithms and linguistic features to perform sentiment classification. The lexicon-based approach uses a sentiment lexicon which represents a list of words and phrases that are commonly used to express positive or negative sentiments. On the other hand, hybrid approaches combine machine learning and lexicon-based approaches to improve sentiment analysis performance. Figure 2.6 provides the outline of the sentiment analysis approaches.

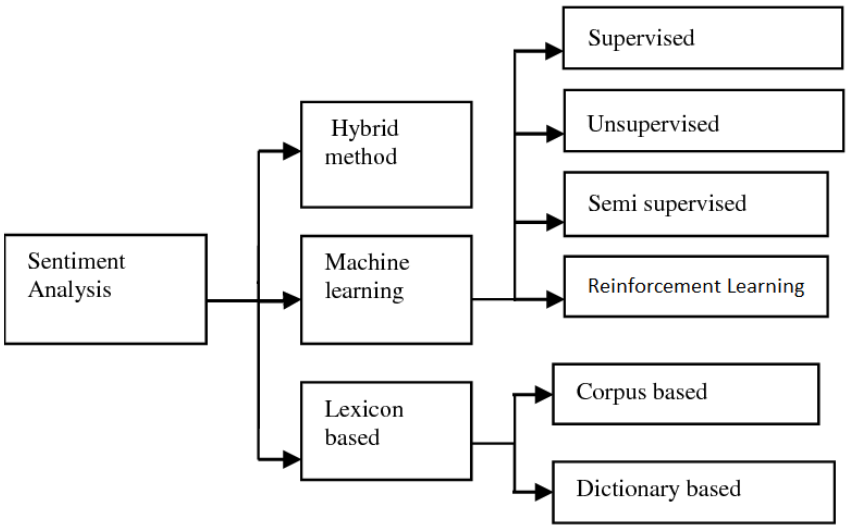

Figure 2.6: Sentiment Analysis techniques.

Lexicon-based

This approach utilizes a sentiment lexicon, which is a list of words and phrases commonly used to express positive or negative sentiments. The lexicon-based approach is practical for sentence and feature-level sentiment analysis and does not require training data. However, it is challenged by domain dependency and dropped performance compared to machine learning when provided with large training datasets. Two major techniques for creating and annotating sentiment lexicons include the dictionary-based approach and the corpus-based approach.

Machine-Learning based

This approach involves classifying sentiment polarity using training and testing datasets. Machine learning can be further divided into supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. Although machine learning can provide better classification results, it often requires large training datasets to perform well.

Hybrid based

This approach combines the benefits of both machine learning and lexicon-based approaches to achieve higher accuracy and stability. By using sentiment lexicons as input features for the sentiment classifier, the hybrid approach can inherit the strengths of both methods.

2.4.3 SentiCR

SentiCR is a supervised sentiment analysis tool designed particularly for code review [32]. It uses a sentiment oracle that leverages a feature vector generated by computing Term Frequency-Inverse Document Frequency (TF-IDF) for a BoW extracted from the input text. SentiCR provides a suite of data preprocessing steps, these include a two-stage negation preprocessing approach, expansion of contractions, special handling of emoticons,

word stemming, and removal of stop-words, code snippets and URLs. The training process performs a 10-fold cross-validation to validate eight supervised algorithms. In which, the distributed version implements Grading Boosting Tree (GBT) as its default algorithm as it displayed the highest precision, recall and accuracy amongst all.

The tool is a binary classifier, where given a text in input, SentiCR issues either -1 for sentences predicted as negative, or 0 for sentences predicted as Non-Negative. As opposed to SentiSW, SentiCR does not distinguish between positive and neutral sentiment.

2.4.4 SentiSW

SentiSW is an entity-level sentiment analysis tool specific for Software Engineering (SE) domain created by [33]. It aims at classifying issue comments into three categories negative, positive, and neutral. The tool also returns the target subject along with the sentiment, which helps in identifying the entities that invoke a particular sentiment. The authors have experimented with two distinct feature extraction methods and 6 supervised classification algorithms. Identically, the experimental evaluation results yielded a configuration analogous to SentiCR, the combination of TF-IDF and GBT performed best among all others.

The models were trained on a manually labeled dataset containing 3,000 issue comments and the authors reported a 68% mean precision, 63% mean recall, and 77% accuracy, which outperforms other existing tools.

2.5 Feature Extraction

An important step in the machine learning is effective feature extraction. In order to translate natural language processing problems into machine learning problems, the first step is to convert natural language into information such as vectors that computers can recognize.

2.5.1 Bag-of-Words

The BoW model is a widely used and effective method for document representation, transforming variable-length input data into fixed-size representations by quantizing it through a codebook [34]. The simplicity and efficiency of the BoW model have contributed to its popularity, as it has the capability to achieve high accuracy in various applications, such as sentiment classification [35]. The BoW model maps a document into a vector as v = [x1, x2,...,xn], with xi denoting the occurrence of the ith word in basic terms, which are usually the top n highest-frequency words in the dataset [36].

However, the BoW model has several limitations. Firstly, it only considers the presence or absence of words, ignoring the influence of other words in the document and disregarding contextual information [36]. This issue leads to difficulties in effectively capturing the semantics of documents, as semantically similar documents with different word usages might be mapped to divergent vectors under the BoW model [37]. Moreover, the BoW model overlooks long-range word ordering, causing sentences with different meanings but using the same words to have similar representations [35].

The model also struggles with handling negation, as it fails to recognize the intended negative sentiment in sentences where positive words are negated [35]. Furthermore, the BoW model has a limited understanding of word semantics, as words are treated as atomic units, resulting in sparse "one-hot" vectors and no notion of similarity between words [35].

Despite these limitations, the BoW model has been widely adopted in various applications, and it serves as a fundamental starting point for more advanced text representation methods (Figure 2.7).

		add	but	efficient	great	improve	implementation	missing	more	names	nice	please	refactoring	unit	tests	variable
Great refactoring, add more tests.	->	1	0	0	1	0	0	0	1	0	0	0	1	0	1	0
Efficient implementation, lacks unit tests.	->	0	0	1	0	0	1	0	0	0	0	0	0	1	1	0
Nice refactoring, but improve variable names.	->	0	1	0	0	1	1	0	0	1	1	0	1	0	0	1
Missing unit tests, please add.	->	1	0	0	0	0	0	1	0	0	0	1	0	1	1	0

Figure 2.7: Bag-of-Words

2.5.2 Word2Vec

Word2Vec, developed by [38], is a neural network-based algorithm developed for generating word embeddings, which are low-dimensional vector representations of words that encode their semantic features in an unsupervised manner using large text corpora [39]. This method maps each word to a fixed-length vector in an n-dimensional vector space, where similar words are positioned close to each other [39]. It is designed to capture the contextual relationships between words in a sentence or document without human intervention, enabling the efficient processing of text data for various natural language processing and machine learning tasks [40].

The Word2Vec algorithm comprises two primary approaches: Continuous Bag of Words (CBOW) and Skip-Gram (SG), which differ in their input-output configurations [41], [42]. Figure 2.8 illustrates the architecture of both CBOW and SG algorithms, highlighting their unique designs and processes.

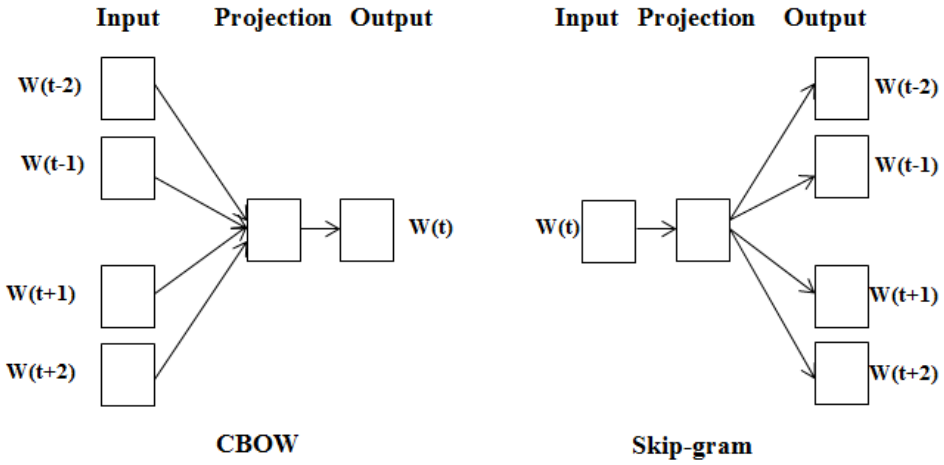

Figure 2.8: Word2Vec models: The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

In the CBOW approach, the algorithm takes a set of word inputs and attempts to predict the context in which these words appear [41]. This method focuses on understanding the contextual relationships between words, with the neural network learning to generate word embeddings by predicting the surrounding words based on the given input words.

On the other hand, the SG approach reverses the CBOW process, taking context inputs and predicting the words that fit within that context [41]. This method is particularly effective in capturing the semantic relationships between words, as the neural network learns to generate word embeddings by predicting target words based on their surrounding context. Both CBOW and SG utilize backpropagation and stochastic gradient descent methods to learn word vectors, featuring a single hidden layer in their neural network architectures [40].

The algorithm maintains the contextual integrity of sentences and the semantic similarity between words, where related words are represented near each other in the vector space [39], [42]. By evaluating the cosine similarity between word vectors, the model is capable of discerning semantic relationships, with similar words having similar vectors and dissimilar words having diverse vectors. Word2Vec is advantageous due to its simplicity, scalability, and speed, providing good results for both small and large datasets [40].

2.5.3 Doc2Vec

Doc2Vec, an extension of the Word2Vec model, was introduced by [43] to improve the learning of embeddings from word-to-word sequences and is applicable to word n-grams, sentences, paragraphs, or documents [44]. The model aims to represent documents as fixed-length, low-dimensional vectors and is structured as a three-layer neural network, consisting of an input layer, a hidden layer, and an output layer [44], [45]. In contrast to Word2Vec, Doc2Vec requires documents to be labeled, as it associates words with their corresponding labels, rather than focusing on relationships between words [46].

Doc2Vec features two distinct algorithms: DM and DBoW, which correspond to the and SG algorithms in Word2Vec [44]. The DBoW approach operates similarly to SG, with the input being substituted by a unique token representing the document, while word order within the document is disregarded [45]. On the other hand, the DM algorithm functions in a manner akin to DBoW, introducing an extra document token as input in addition to multiple target words. However, unlike DBoW, these vectors are concatenated rather

than summed [45].

Figure 2.9 shows the two architectures, the first is DM, and it will add a paragraph id to be trained with word vectors. This paragraph id contains information that is missing from the current word. The latter, shows ways to input into the DBoW model is a paragraph id, predicting randomly sampled words in this document.

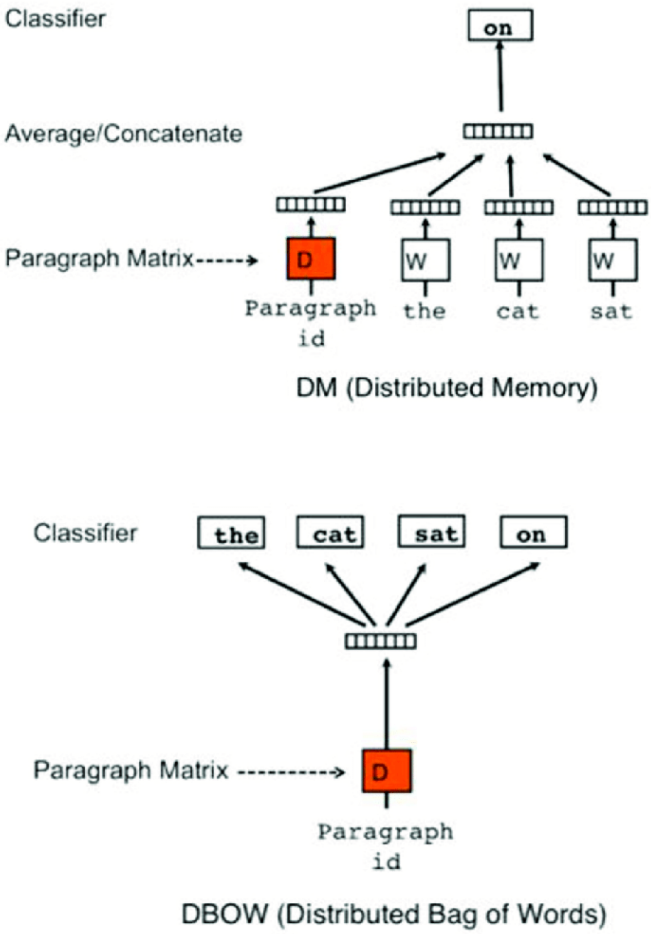

Figure 2.9: Doc2Vec models: DM and DBoW.

Through the training process, Doc2Vec generates document vectors for each document, with each document vector corresponding to a column in the document vector matrix D [47]. Additionally, each word vector is represented by a column in the word vector matrix W [47]. Once adequately trained, the distributed representation of each document in the corpus can be obtained, which can be utilized for subsequent text mining tasks such as

clustering or classification [48].

2.5.4 TF-IDF

TF-IDF is a widely used method in information retrieval text mining, and natural language processing to quantify the importance of words within a textual corpus [49], [50]. It is a statistical measure that reflects the significance of a word or phrase to a document or category in a collection of documents or corpora, with the primary idea being that words or phrases with a high frequency in one document and low frequency in others should be considered more important as they are more useful for classification purposes [50], [51]. TF-IDF is based on the bag-of-words scheme, which represents a document as a collection of words used within it [48]. This approach is often employed in information retrieval and text mining to evaluate the importance of a linguistic term, such as a unigram or bigram [52].

The TF-IDF algorithm comprises two components: Term Frequency (TF) and Inverse Document Frequency (IDF) [48], [50]. Term Frequency represents the frequency of a word's occurrence in a document, calculated as the number of times the word appears in a document divided by the total number of words in the document [50], [52].

Calculate Term Frequency (TF) TF is computed as the ratio of the frequency of a specific term in a document to the total number of terms in the document. The formula for TF is: $TF(t,d) = \frac{f(t,d)}{n(d)}$

Where $f(t,d)$ is the frequency of term t in document d and $n(d)$ is the total number of terms in document d.

In contrast, inverse document frequency measures the general importance of a term across an entire corpus [50], [51]. The importance of a term increases with its frequency in the text, but is offset by the frequency of the term in the domain of interest, thereby scaling down frequent words like "the" or "for" [52]. It is computed by taking the logarithm of the inverse of the proportion of documents in the corpus containing the term, where N

is the total number of documents and $df(t)$ is the number of documents containing the word i [50].

Calculate Inverse Document Frequency (IDF) IDF is calculated as the logarithm of the ratio of the total number of documents in the corpus to the number of documents containing the term 't'. The formula for IDF is: $IDF(t) = log(\frac{N}{df(t)})$

Where N is the total number of documents in the corpus and $df(t)$ is the number of documents containing the term t.

The TF-IDF weight, a product of TF and IDF values, is used to evaluate the importance of a linguistic term in a studied corpus [50], [52]. Higher weights signify that a word carries greater significance within the corpus, which is particularly useful for document classification [48], [50].

Compute the TF-IDF weight The TF-IDF weight for a term 't' in a document 'd' is calculated by multiplying the term frequency and inverse document frequency values. The formula for the TF-IDF weight is: $TF-IDF(t,d) = TF(t,d) \times IDF(t)$

Where $TF(t,d)$ is the term frequency of term t in document d and $IDF(t)$ is the inverse document frequency of term t.

Statistical methods like machine learning and deep learning require text data to be transformed into numerical values before analysis can be performed [50]. TF-IDF serves this purpose by converting words into a numeric format that can represent the data as a whole [53]. The outcome of the TF-IDF process is a matrix composed of rows and columns, with data represented in rows and features in columns [53]. This matrix can then be utilized as input for various machine learning and deep learning algorithms to perform tasks such as document classification and sentiment analysis [50], [52].

2.6 Evaluation and Performance metrics

To evaluate the performance or quality of a Machine Learning model, different metrics of performance can be applied in order to determine how well the model generalizes on

new data. As such, the following metrics were used to quantify the performance of the tools:accuracy, precision, recall/sensitivity, specificity, f1-score, Cohen Kappa Score, ROC curve.

In the case of binary classification, a given set of data is categorized into 2 groups based on a rule. For each testing subject, 4 scenarios are possible, and the basis of precision, recall, and F1-Score comes from the concepts of True Positive, True Negative, False Positive, False Negative. The following definition explores more about this concept:

True Positive (TP) refers to the positive instances that were correctly predicted by the classifier.

True Negative (TN) refers to the negative instances that were correctly predicted by the classifier.

False Positive (FP) represent the positive instances that were incorrectly labeled as negative.

False Negative (FN) represent the positive instances that were incorrectly labeled as negative.

Accuracy

Accuracy is a metric that informs the ratio between the number of correctly classified predictions made by the model divided by the total number predictions. In other words, it measures the performance of a machine learning model by determining the proportion of correct predictions made by the model out of the total predictions on a given dataset. The accuracy is defined by the formula: Acc = $\frac{TP+TN}{FP+FN+TP+TN}$.

Precision

Given a set of results out of a processed document, Precision is the percentage value indicating how many of those results are correct (correct being based on the expectations of a certain application). It is given by the formula: P = $\frac{TP}{TP+FP}$

Recall/Sensitivity

Shows the proportion of correct positive predictions out of all positives a model could have made. Unlike precision, recall provides an indication of missed positive predictions. $R = \frac{TP}{TP+FN}$

F1-Score

The F1 score can be interpreted as a harmonic mean of the precision and recall values of a system., where an F1 score reaches its best value at 1 and worst score at 0. It is a metric that becomes especially valuable when working on an imbalanced data set. $F1 = 2 * \frac{Precision*Recall}{Precision+Recall}$

Specificity

Specificity is the probability that the test will give a negative result. That is, the test measures the proportion of results without a given characteristic, indicating the power of the test to identify individuals who do not have the trait.

ROC curve

Receiver Operator Characteristic (ROC) curves were introduced in machine learning as a powerful tool for the evaluation of classification models, illustrating how well the classifier algorithm works by summarizing the tradeoff between the True Positive Rate (TPR) and the False Positive Rate (FPR) for a predictive model using different probability thresholds. A ranking model can be represented by a point in ROC space and, to obtain a point in this space, a classification model and the rate of true and false positives are calculated.

The x-axis, or independent variable, is represented by the FPR and the y-axis, dependent variable, by the TPR. Each point in ROC space is a TP/FP data pair. In Figure 2.10, the diagonal line connecting the dot (0, 0) and the dot (1, 1) denotes the performance of the random classifier. The ROC curves in the area with the upper left corner (0, 1) indicate good levels of performance, while the ROC curves in the area with the

lower right corner (1,0) indicate poor levels of performance. In this way, a point of perfect performance would be the point (0, 1) indicating 0% false positives and 100% true positives.

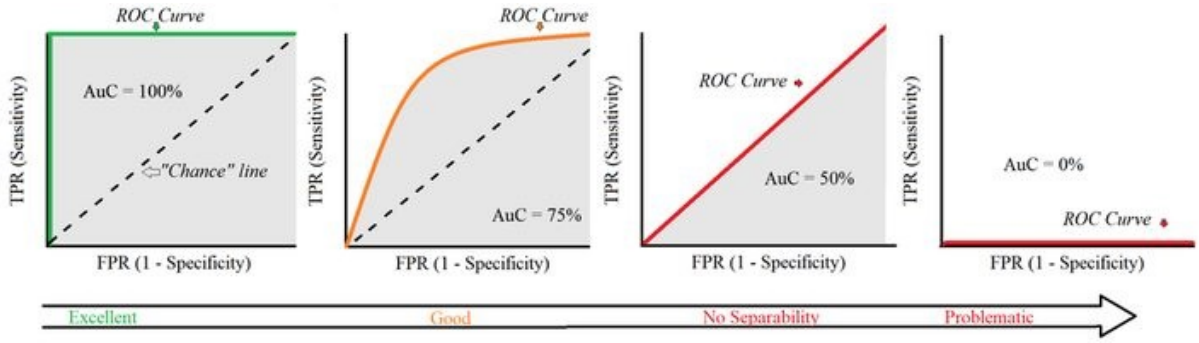

Figure 2.10: ROC Curve

Cohen Kappa Score

Cohen's Kappa is a statistical measure that is used to assess the agreement between two or more raters, who are categorizing the same items into two or more classes. This measure is commonly used in inter-rater reliability studies, where the goal is to assess how well the raters agree on a given classification task.

Cohen's Kappa is computed as the difference between the observed agreement between the raters and the expected agreement if they were randomly guessing, divided by the maximum possible agreement that could be achieved. The resulting score ranges from -1 to 1, with negative values indicating no agreement and positive values indicating some degree of agreement beyond chance.

Cohen's Kappa can be used to assess the agreement between two raters or multiple raters, and it can be applied to nominal or ordinal data. It has several advantages over simpler measures of agreement, such as percentage agreement, because it takes into account the possibility of chance agreement.

In addition to its use in inter-rater reliability studies, Cohen's Kappa is also used in machine learning and natural language processing to evaluate the performance of classifiers. In this context, Cohen's Kappa is used to assess the agreement between the predicted

labels and the true labels for a given set of examples.

2.7 Related Work

Research in the field of sentiment analysis in software engineering context has received growing attention over the last years [54]. The poor performance of off-the-shelf tools in the SE context has led to an increase in the interest for domain-specific SA tools, few attempts have been made by researchers publishing their works regarding this topics [55].

The use of NLP methods in order to analyze the underlying sentiments contained in texts written by developers as introduced in [56] has been verified by their work by investigating the expression of emotions of developers in GitHub issues. The technique has also been employed in other aspects of SE such as analyzing the association between developers' sentiment in commit messages and build breakage in continuous integration servers [57]. They found that negative sentiment both affects and is affected by the result of the build process.

A case study in the Gentoo community [55] revealed emotional preconditions preceding a collaborator's departure and thus, offered a Bayesian classifier capable of monitoring discussions and predicting the likelyhood of contributors leaving the project.

For their part, the authors of [58] compared three *SE* domain-specific classifiers against a baseline approach represented by an off-the-shelf tool in four publicly available datasets to assess their performance and reliability. *SentiCR* was a subject of this research, and their findings supports the experimentation of our first study, suggesting that the customization of sentiment analysis tools to the software development domain does improve the classification accuracy.

In contrast, the "negative results" paper of [59], the authors shared their experience with building a software library recommender that leverages sentiment analysis in developers' opinions mined from Stack Overflow. The researchers carried out an investigation of the accuracy of three sentiment analysis tools to classify the polarity of SE related texts. The study shows that none of the experimented tools are capable of producing

reliable results when discriminating between positive/negative and neutral sentences, and further warns the research community about the strong limitations of current sentiment analysis tools.

Chapter 3

Method

In this section we present our research questions and method, the data gathered and our ground definitions.

3.1 Research Questions

Based on what has been exposed, this study will therefore address the following research questions:

RQ1 To what extent does the amplification of train dataset affect the performance of the tools studied?

Available literature has established a direct relation between the number of training samples provided with the increase on performance of a given mode. Other research states that the quality of the data. For this particular reason, we test if this hypothesis holds true for both SA tools studied in this work.

RQ2 Does sentiment distribution differ among programming languages?

Github hosts a vast amount of projects that are continuously maintained by millions of developers scattered across the world. The possibility of remote collaboration and communication about the code, underlying project design, and arising issues provides an

infrastructure to develop multi-layered socio-technical communities. A common categorization of sub-communities is by programming languages, which we utilized to investigate sentiment distribution between them.

RQ3 Does sentiment distribution differ among project size?

Given the contrast between OSS and Corporate Culture practices during development [60], we explore whether corporate-run projects are less exposed to negative discussions than non-corporate projects.

3.2 First Study: Context in Sentiment Analysis

The purpose of our first study is to evaluate how effectively the expansion of training datasets enhances the performance of the sentiment analyzers discussed in this paper. We conducted an experiment by permutating their training sets and compared their predictions on a ground-truth "gold standard" dataset from Calefato et. al. [61], containing 4423 posts from Stack Overflow in the form of questions, answers, and comments. The Senti4SD dataset demonstrates a balanced distribution, with 35% of the posts expressing positive emotions, 27% convey negative emotions, and the remaining 38% being labeled as neutral polarity since no specific emotions were observed in those posts. Each post was reviewed and annotated with sentiment polarites (Positive, Negative, Neutral) by three distinct human raters. The annotations are based on a majority agreement.

3.2.1 Data Augmentation

We detected a few inconsistencies that were corrected after a thorough inspection of SentiSW's training CSV file. Out of the 3000 entries, we have detected 7 *NaN* text inputs and 27 rows containing Chinese text. The presence of such characters in the training set could impact the performance of the tool and they were removed to better accommodate the English language domain.

Oftentimes, when working on specific complex tasks such as classifying sentiments out of written text, it is very hard to acquire large amounts of data required to train the models. One way to deal with the problem of limited data is to apply different transformations on the available data to synthesize new data. This approach of synthesizing new data from the available data is referred to as "Data Augmentation". With this in mind, we merged our classifier's dataset and performed two operations on the resulting set, namely, transform to binary and extend from binary.

3.2.2 SentiCR+ dataset generation

For the generation of the SentiCR+ dataset, a transformation was applied to the sentiment polarity labels originally present in the SentiSW dataset. This transformation was carried out in order to align with the binary classification approach utilized in SentiCR. We transform the 3 label sentiment polarity (Positive, Neutral, Negative) from SentiSW to accommodate SentiCR's binary classification (Non-Negative, Negative). According to 3.2, this strategy considers negative sentiments as is and maps positive and neutral sentiments as non-negative.

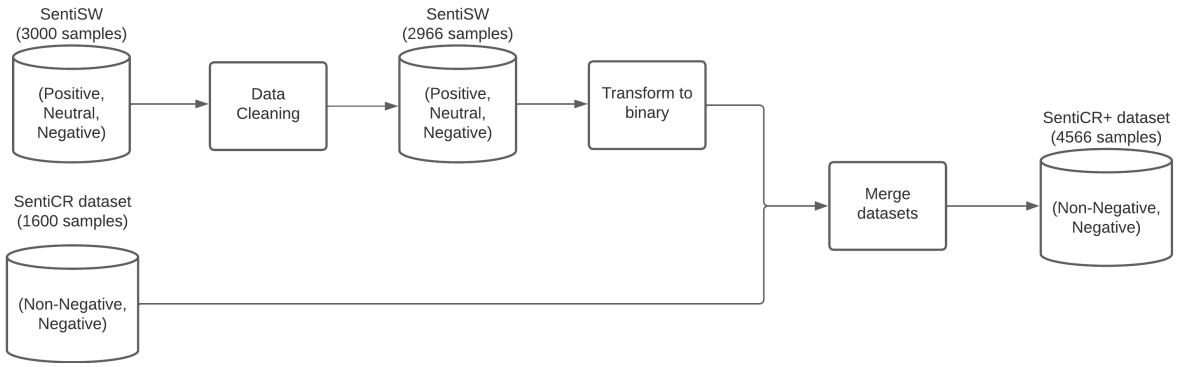

Figure 3.1: SentiCR+ generation process

3.2.3 SentiSW+ dataset generation

For the SentiSW+ dataset generation, a strategy was employed to convert the binary classification of SentiCR into the three-domain class labels of SentiSW (Negative, Neutral, Positive). This conversion process required manual classification of each binary label (Negative, Non-Negative) from SentiCR, as depicted in Figure 3.2.

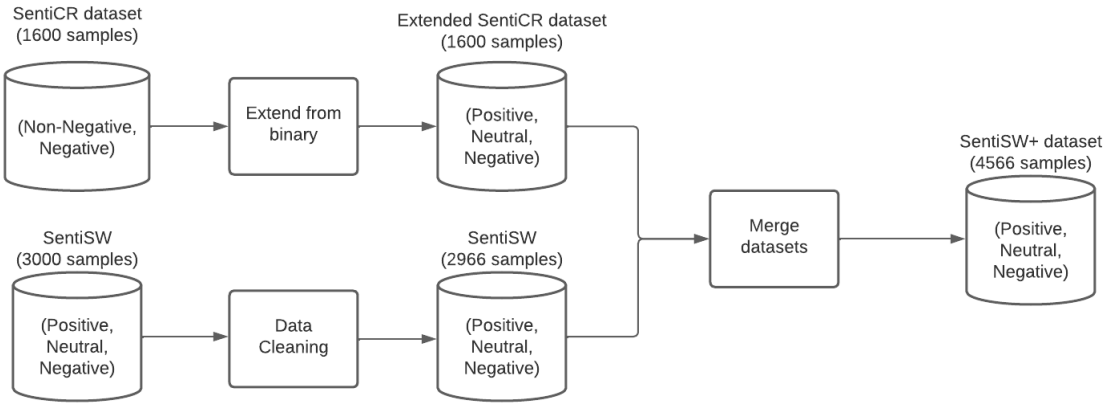

Figure 3.2: SentiSW+ generation process

3.2.4 Performance Evaluation

The final step of the first study involved evaluating the performance of the original tools and their respective variations in classifying Senti4SD's dataset. The generated predictions were compared to the manual annotations of Senti4SD's dataset. Evaluation metrics such as precision, recall, and f1-score were computed using the macro-average approach. Additionally, the Cohen Kappa metric was utilized to measure the agreement rate among the analyzed tools.

3.3 Second Study: Applied Sentiment Analysis in GitHub OSS projects

In an effort to answer our research questions, we analyzed the sentiment expressed by developers during discussions on issues and pull requests of several software development projects. To achieve this, we extracted all issues of a sampled population, ranging from the smallest repository to well-known projects from GitHub and performed Sentiment Analysis on each comment. In this section we thoroughly expose the whole process.

3.3.1 Data collection

The GHTorrent project [62] provides a data dump of GitHub's public event timeline, which is updated every month. At the time of data collection (Jan 2020), MongoDB stores around 18TB of JSON data (compressed), while MySQL more than 6.5 billion rows of extracted metadata. A large part of the activity of 2012-2019 has been retrieved.

The GHTorrent database dump comes in CSV files, one per database table. To restore the database as a whole, the steps in the work of Gousios et al. [63], were conducted. In the current work, only *projects*, *commits*, *issues* and *pull requests* tables were taken into account.

Although GHTorrent stores metadata of OSS repositories, some crucial information about edited files, title, body and comments of issues and pull requests are not available. These additional fields of metadata can be publicly found, however, by making REST API calls on GitHub.

To complement our dataset, an extraction tool (GitCrawler) was developed. GitCrawler asynchronously requests API calls and stores the response in a CSV formatted file. The code repository for our tool can be found in `https://github.com/utfpr/reposa.git`. Figure 3.3 provides an overall view of the steps involved.

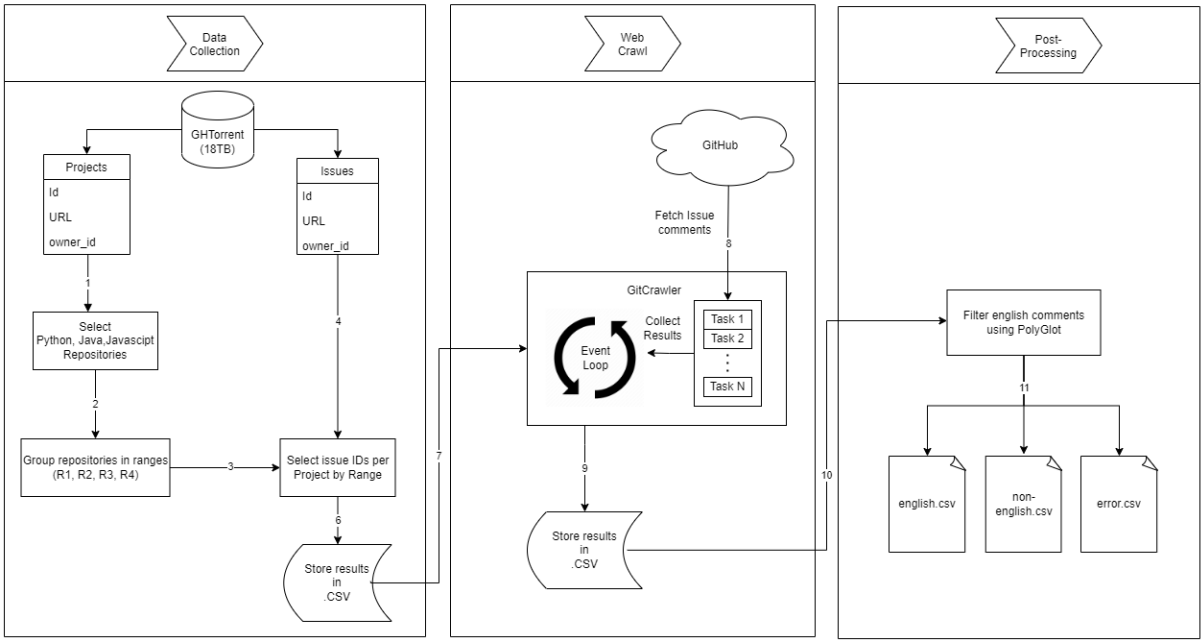

Figure 3.3: Data collection flowchart

3.3.2 Repository selection

In our second study we selected non-deleted Open Source Software projects (forks included) hosted on Github from the three most popular programming languages (Python, Java, and Javascript). To better visualize our data, we defined and grouped projects by ranges of issue comment quantity and noticed that most of the repositories in our dataset is composed of personal projects, with little to no issue/pull requests. The absence of human interaction in these tickets pose no significant value for our investigation and thus, we discarded projects with a maximum of 10 issues. The remaining set of ranges that were considered are described as $R1 \in [10, 100]$, $R2 \in [100, 1000]$, $R3 \in [1000, 10000]$, $R4 \in [10000, 1000000]$.

A simple homogeneous sampling was used to draw samples from each resulting population. The confidence level was set to 95% with a $\pm 5\%$ confidence interval, plugging these values into an online calculator yields the following target populations described in Table 3.1.

Table 3.1: Population sizes by range

Language	***Range***	***Nº Projects***	***Sample 80/20***
Python	10-100	57606	245
	100-1000	8434	239
	1000-10000	606	176
	10000-100000	23	23
Java	10-100	53948	245
	100-1000	7344	238
	1000-10000	613	176
	10000-100000	9	9
Javascript	10-100	120727	246
	100-1000	15562	243
	1000-10000	1000	198
	10000-100000	25	23

3.3.3 Issue comment metadata extraction

For every project in our sample, we acquired a list of all issue identifiers by executing a MySQL query from the installed GHTorrent snapshot. The list of id values was used as input for GitCrawler, which iterates over the identifiers list, receiving JSON responses from GitHub's API and performs an issue comment verification check. For every response that has an accumulated comment count greater than zero, the issue metadata is stored as a row in a CSV file. The issues that do not fit in this criteria are discarded. Table 3.2 summarizes the number of issues, issues with comments and total comment count for their respective segment and language.

3.3.4 Filtered issue population sizes

Since comments from both sources are generated by the community, pre-processing and polishing data was a mandatory step before supplying it to the Sentiment Analysis algorithms. Our analyzers are bound to the English vocabulary and therefore are unable to accurately classify inputs from other languages.

We decided to exclude rows of data that didn't possess an English corpus. For this aim,

Language	*Range*	*Total issues*	*Issues with comments*	Issue comments
Python	R1	7160	3299	9946
	R2	36441	32254	128554
	R3	375474	175751	750282
	R4	404806	30440	163593
Java	R1	6876	2280	6098
	R2	58935	33754	118495
	R3	420153	152876	590602
	R4	270454	22368	109524
Javascript	R1	7235	2523	7214
	R2	60294	32254	116203
	R3	40089	168167	702541
	R4	373790	23802	128481

Table 3.2: Target population issue count

a python script was written to receive the issue comments file generated in our previous step and filter out every non-english comment text utilizing polyglot - an open-source python library that is used to perform different NLP operations. The script produces three files as output, one containing english-only issue comments and the other containing non-english text and language classification error results. After execution, we obtained the results as shown in Table 3.3.

Language	*Range*	*Issue comments*	*en-US Issues*	en-US issue comments
Python	R1	3299	3178	9472
	R2	32254	35280	124682
	R3	175751	170536	723506
	R4	30440	28843	148248
Java	R1	2280	2133	5651
	R2	33754	31339	108770
	R3	152876	147557	562898
	R4	22368	22319	106123
Javascript	R1	2523	2290	6458
	R2	32254	30254	108254
	R3	168167	160472	659227
	R4	23802	23603	125191

Table 3.3: Issues with English corpus

3.3.5 Comment classification

We ran our 4 different variations of sentiment analyzers against the resulting set of issue and pull request comments. For each R segment, we've acquired the prediction label and an array of sentiment probabilities. Additionally, for the 2 variations of SentiSW, we've obtained their respective detected entity when the predicted labels were different than "Neutral".

3.4 Evaluation/Confirmation

To further investigate the benefit of incorporating more data into SA tools during the training phase, we evaluate their performance by comparing their scores against a test dataset of manually classified truth labels by the author. The test dataset consists of a sample of randomly selected issue comment texts individually extracted from all R ranges across all programming languages. We employed the identical sample calculator values provided before to acquire the target population. For each R range and for every programming language, we collected 250 text artifacts, totalling to 3000 issue comments.

We computed accuracy metrics, ROC curves and Cohen Kappa's correlation score to give us a preliminary picture of each SA tools.

Chapter 4

Discussion and Results

In conformity the first study of this dissertation, which is to determine how much train data expansion helps improving NLP classification accuracy, this chapter starts by defining the three experimental approaches for evaluating the proposed models accuracy. The first and second experiment takes statistical measures of performance of the tools before and after being subjected to more train samples. In the third experiment we draw sample of the discordance between the tools and apply the before metioned statistical metrics in order to identify the change in behaviour of the outputs.

4.1 Analysis

Together the previous results provide insights to consider when interpreting the discoveries of the research questions described in the previous chapter. The second study moves on to discuss the obtained classification results using both SA tools and their altered models. Additional findings are reported in the end of this chapter.

Pursuing RQ1, we expect to assess to what extent does the amplification of train dataset affect the performance of the tools studied.

To quantify the benefit of incorporating more data into SA tools during the training phase, we conducted the steps described in section 3.2 to measure their performance by comparing their outputs against a test dataset (*Senti4SD*) of manually classified truth

labels. Table 4.1 shows the acquired values for precision, recall and f1-score metrics:

Sentiment Analysis Tool	**Precision**	**Recall**	**F1-score**
SentiCR	0.76811	0.88854	0.82395
SentiCR+	0.77071	0.93294	0.84410
SentiSW	0.58449	0.89020	0.70566
SentiSW+	0.59878	0.86954	0.70919

Table 4.1: Precision, recall, F1-score metrics for macro-average.

The first experiment already provides an interesting result: when using the expanded model of SentiSW, the act of increasing the train data size did not have a considerable influence in the output predictions. In our view, this is due to the fact that the expansion of train data did not incorporate any information on past behavior. Both with and without SentiCR's train examples, this model scored a slight difference of accuracy of 0.3%.

In contrast, SentiCR expressed more change in behavior. After increasing the amount of train data, the tool also increased it's precision. This suggests that SentiCR's training dataset could be revised or amplified in order to improve the performance of the model.

To account for the previous evidence, we further investigate the benefit of expanding the model's train dataset as documented in Evaluation/Confirmation. For this purpose, we measured the accuracy metrics for each individual (Table 4.2).

		SentiCR			SentiCR+		
		Precision	Recall	F1-score	Precision	Recall	F1-score
Python	R1	0.698919	0.80958	0.73422	0.72706	0.77198	0.74645
	R2	0.60801	0.73443	0.63176	0.66759	0.71767	0.68808
	R3	0.70185	0.84135	0.74118	0.81666	0.87929	0.84397
	R4	0.80511	0.88511	0.83775	0.84022	0.73052	0.77073
Java	R1	0.6983	0.8087	0.73619	0.70954	0.68043	0.69354
	R2	0.60626	0.63577	0.61789	0.75347	0.68919	0.7152
	R3	0.59301	0.71287	0.61388	0.71325	0.84295	0.75828
	R4	0.58433	0.74868	0.61564	0.5375	0.55291	0.5433
Javascript	R1	0.66907	0.72621	0.69054	0.76094	0.76939	0.76507
	R2	0.71273	0.74694	0.72767	0.79215	0.76749	0.77901
	R3	0.64179	0.78688	0.67117	0.72328	0.78529	0.7489
	R4	0.67011	0.73265	0.69181	0.83637	0.78082	0.80496

Table 4.2: SentiCR precision, recall, F1-score metrics for macro-average.

According to the results , SentiCR+ performed on average 7,4% better precision, -1,6% Recall, and 4,5% F1-Score than the original. Overall, this tool has benefited the most. Indicating the viability of improving the model by means of increasing its train dataset.

SentiSW+ got 0,6% precision, 3,9% Recall and 1,9% F1-score (Table 4.3). The tool suffered a slight increase for the neutral classification and a moderate amount for the negative class in both R1 and R2 Java. And for R1, R2 Python and R4 Java, SentiSW+ performed slightly worse than its original version.

		SentiSW			SentiSW+		
		Precision	Recall	F1-score	Precision	Recall	F1-score
Python	R1	0.8184	0.81142	0.81482	0.77441	0.79648	0.78483
	R2	0.7102	0.77459	0.73822	0.69815	0.77124	0.72941
	R3	0.70798	0.73086	0.71545	0.68662	0.77567	0.72162
	R4	0.6475	0.65878	0.62755	0.67912	0.73393	0.69014
Java	R1	0.65133	0.74686	0.68853	0.65977	0.81892	0.7154
	R2	0.62024	0.64771	0.62968	0.65651	0.67247	0.65313
	R3	0.71329	0.7181	0.71566	0.71413	0.79107	0.74438
	R4	0.67529	0.7732	0.71286	0.6223	0.75523	0.66172
Javascript	R1	0.77552	0.77742	0.77412	0.80883	0.8154	0.80628
	R2	0.75421	0.76001	0.75614	0.78729	0.80351	0.79399
	R3	0.61931	0.7375	0.65951	0.66505	0.8149	0.71789
	R4	0.76464	0.7805	0.75321	0.78302	0.84119	0.80127

Table 4.3: Precision, recall, F1-score metrics for macro-average.

We compute the Cohen Kappa Score between SentiCR, SentiCR+, SentiSW and SentiSW+ and their respective score against a ground truth column *y_true* annotated by the author (Table 4.4).

	SentiCR+	SentiSW	SentiSW+	y_true
SentiCR	0.51433	0.19518	0.34056	0.40821
SentiCR+		0.33691	0.45029	0.51114
SentiSW			0.82201	0.6327
SentiSW+				0.64828

Table 4.4: Cohen Kappa Score between the 4 models

We can observe that the original models of SentiCR and SentiSW didn't agree with

each other much, with a Cohen Kappa Score below 0.20 which is categorized as none to slight agreement. When exclusively exposing SentiCR to SentiSW's train dataset, and vice-versa, both comparisons scored a fair agreement of 0.33 and 0.34 respectively. When simultaneously exposed, the concordance between outputs improved to a moderate level of 0.45. As a final remark, when compared to the y_true column, both models expressed different improvements.

This reinforces that even when exposed to the train dataset of SentiCR, SentiSW retains a substantial agreement (0.82) of its original version. Despite SentiSW output is more inclined to the manually classified truth labels, the incorporation of SentiCR's train dataset increased the Cohen Kappa's score by a factor of 0.01558.

Whereas SentiCR, suffered a significant change in output label predictions and although it scored a moderate agreement of 0.51, Interestingly, this change was beneficial, increasing the models concordance with the provided test set by a factor of 0.10293.

In relation to RQ2, the approach to check if sentiment distribution differ among programming languages will be checked. After performing the classifications over the collected samples described in Comment Classification section of our Method. A set of stacked bar graphs grouped by language were built to better visualize this situation.

SentiCR and SentiCR+ ranked Python as the most Negative, Java as the least Negative, and Javascript as the most Non-Negative (Figure 4.1). When evaluated by SentiSW and SentiSW+, Javascript ranked as the most Negative, Java as the most Positive, and Python/Javascript as the most Neutral. All four SA tools unanimously ranked Java as having the least amount of Negative sentiment expression on issue discussions.

Finally, the approach to RQ3 implies checking the sentiment distribution among project size. It is important to highlight that in this study we consider the definition of project size as the total number of issue tickets.

In Python projects, the ratio of Non-Negative sentiments for binary classification tools SentiCR and SentiCR+ is considerably higher than Negative sentiments across all project sizes (Figure 4.2). It is apparent that as the project size increases, the overall quantity of Negative sentiments increases, while the quantity of Non-Negative sentiments reduces.

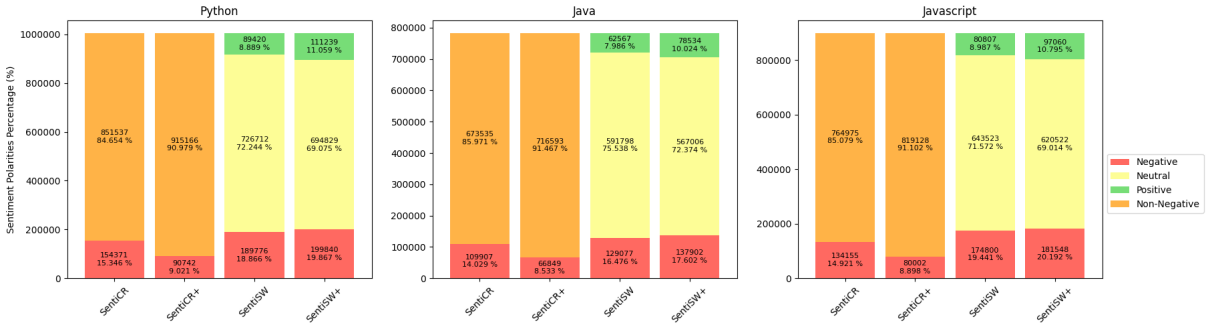

Figure 4.1: Sentiment Distribution amongst Programming Languages

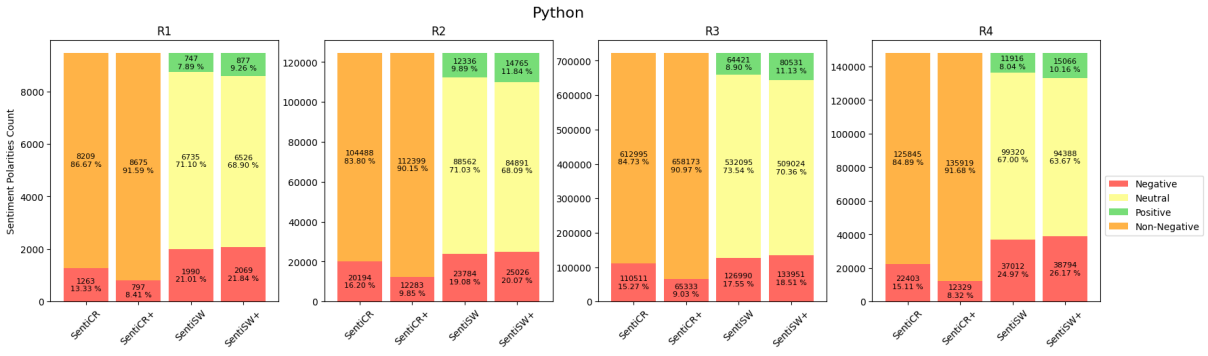

Figure 4.2: Python Sentiment Distribution by Category

The average sentiment proportions and their standard deviations provide insight into the stability and consistency of these tools' classifications across different project sizes (Table 4.5). When looking at SentiSW and SentiSW+, which classify sentiments into Negative, Neutral, and Positive, there is a clear predominance of Neutral sentiments across all project sizes. The Positive sentiment count, though lower than Neutral, increases with the project size, with a noticeable increase in R2 and R3 projects. In R4, the tools expressed a slight decline in positive sentiment but the value still remains higher when compared to R1.

	Negative		Non-Negative		Neutral		Positive	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
SentiCR	0.1498	0.01196	0.8222	0.0590	-	-	-	-
SentiCR+	0.08903	0.007	0.911	0.0071	-	-	-	-
SentiSW	0.2065	0.03205	-	-	0.7067	0.0271	0.08681	0.0092
SentiSW+	0.2165	0.0331	-	-	0.6775	0.0288	0.106	0.0113

Table 4.5: Mean and standard deviations across Python project sizes.

In Java projects, the patterns observed are similar to Python (Figure 4.3). SentiCR and SentiCR+ again show higher counts of Non-Negative sentiments compared to Negative sentiments across all project sizes. Larger projects saw an increase in the classification of Negative sentiments, which is consistent with our findings from Python projects.

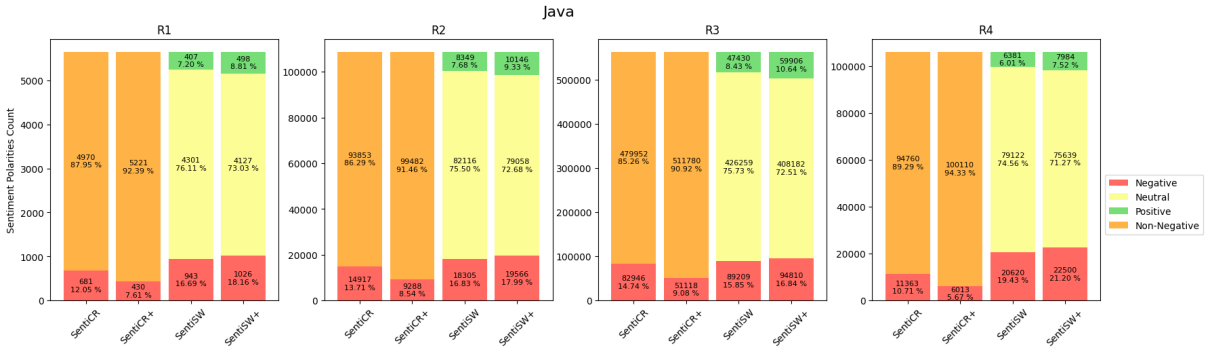

Figure 4.3: Java Sentiment Distribution by Category.

In the case of SentiSW and SentiSW+ tools, the mean values presented in Table 4.6 for Neutral sentiments are the highest. The mean values for Negative sentiments are

significantly higher than Positive sentiments, and the Negative sentiments in SentiSW are particularly high. The standard deviation values for all sentiment categories and tools are relatively low, suggesting consistent sentiment classification across different Java projects.

	Negative		Non-Negative		Neutral		Positive	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
SentiCR	0.128	0.0179	0.87198	0.0178	-	-	-	-
SentiCR+	0.0772	0.015	0.9228	0.015	-	-	-	-
SentiSW	0.172	0.0155	-	-	0.7547	0.0066	0.0733	0.0101
SentiSW+	0.1854	0.0186	-	-	0.7237	0.0076	0.0908	0.013

Table 4.6: Mean and standard deviations across Java project sizes.

In Javascript projects, similar patterns can be observed as in Python and Java projects. Both SentiCR and SentiCR+ tools show lower mean values for Negative sentiments compared to SentiSW and SentiSW+ tools. The SentiCR+ tool shows a significant decrease in the mean value of Negative sentiments compared to SentiCR (Figure 4.4).

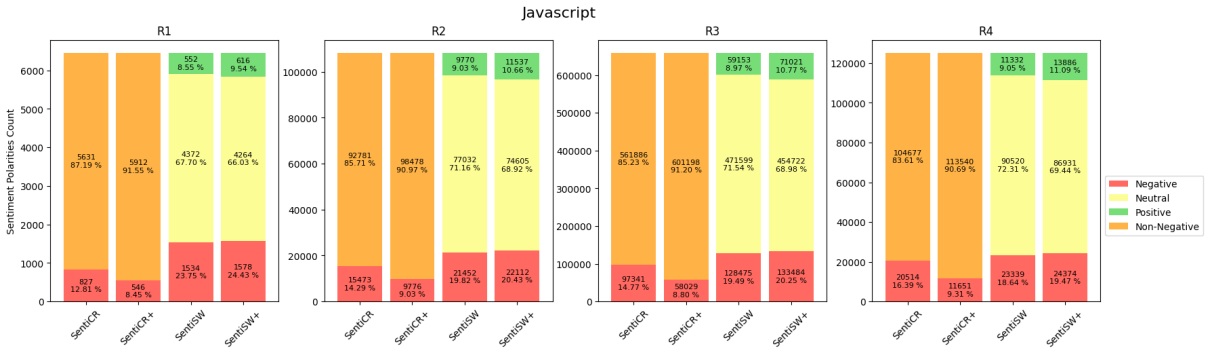

Figure 4.4: Javascript Sentiment Distribution by Category.

In Table 4.7, the narrative changes however, with SentiSW and SentiSW+ showing a progressive decrease in Negative sentiments as the project size grows, followed by an increase of Neutral and Positive sentiments. It's possible that very large projects have established communication norms that limit the amount of negative sentiment expressed in public issue discussions or that more experienced developers work on these projects and express criticism more constructively.

To summarize our findings, the count of Neutral sentiments was the highest among

	Negative		Non-Negative		Neutral		Positive	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
SentiCR	0.1456	0.0147	0.8543	0.0147	-	-	-	-
SentiCR+	0.089	0.0036	0.911	0.0036	-	-	-	-
SentiSW	0.2042	0.0227	-	-	0.7068	0.0204	0.089	0.0024
SentiSW+	0.2114	0.0223	-	-	0.6834	0.0156	0.1052	0.0068

Table 4.7: Mean and standard deviations across Javascript project sizes.

all sentiment labels across all project sizes, indicating that the bulk of issue discussions are factual and objective, rather than being emotionally charged. It is also worth noting that the number of Positive sentiments also increased with project size. However, the rise was not as dramatic as with Negative sentiments.

Regarding the influence of Programming Language on Sentiment, while the overall trends are consistent across Python, Java, and Javascript projects, there are subtle differences in the sentiment distributions. This might be influenced by the language-specific context and nuances that can alter the sentiment of the discussions. However, more detailed analysis would be needed to confirm this.

Considering the relatively low standard deviations in sentiment percentages, we can say that the sentiment distributions are quite stable across different project sizes. These results indicate that both the size of the project and the sentiment analysis tool version are significant variables that influence the sentiment distribution. However, to formally quantify the significance of these findings, hypothesis testing (like Chi-square or ANOVA) could be conducted. As with any analysis, it is crucial to consider potential confounding factors. For example, the type and nature of the projects, the community around each language, the temporal aspect (the point in time when the analysis is performed), among others.

Finally, the obtained results only provide a snapshot of sentiment analysis across different project sizes and languages. There is a need for future work to investigate into the contextual factors affecting sentiment in open-source project discussions, including a thorough evaluation of sentiment analysis tools across diverse datasets.

4.2 Threats to Validity

The performance of the sentiment analysis models was measured using precision, recall, and F1-score metrics, which are standard measures in the field of natural language processing. However, these measures might not capture all aspects of the model's performance. For instance, they might not account for the model's ability to correctly classify complex or ambiguous sentiment expressions. Furthermore, these metrics assume that all errors are equally costly, which might not be the case in practical applications. The use of a single, manually classified dataset (Senti4SD) for evaluation might also limit the validity of our results, as the model's performance might vary with different datasets.

Moreover, our study relied on the use of existing sentiment analysis tools (SentiCR and SentiSW) and their "expanded" versions (SentiCR+ and SentiSW+). However, the operational definition of "expanded" - that is, the addition of more training data - might not reflect all the ways these tools could be improved or modified. Other factors, such as changes in model architecture or training strategies, could also play a significant role in the model's performance and were not considered in this study.

Finally, it is worth noting that the performance of these models was tested solely in the context of programming languages (Python, Java, JavaScript), which may limit the generalization of our findings to other domains or languages.

Regarding the internal validity, our study relies on machine learning models trained on pre-existing datasets. Therefore, any biases or errors inherent in the data could potentially influence our findings. The randomness of data split (into training, validation, and test datasets) can also impact the models' performance. Furthermore, the performance of the models may have been affected by the default configurations we used during the training process, including the learning rate, batch size, and number of training epochs values.

Another threat to the internal validity could be the annotation bias. While the sentiment of developers' comments was manually annotated to create the Senti4SD dataset, the process is inherently subjective and may introduce biases. The researchers who annotated the sentiments might have interpreted some comments differently than others, and

this could affect the model's training and evaluation.

Lastly, the replication of SentiCR and SentiSW's strategies in creating expanded versions might not be perfect, leading to potential inaccuracies. The process of obtaining additional training data for these models, as well as the selection of data, may not perfectly mimic the strategies used in the original studies, which might affect the comparison between the original and expanded versions of the models.

In terms of external validity, our study is primarily limited by the specificity of the data we used. The Senti4SD dataset was created using comments from Stack Overflow related to specific programming languages. Consequently, the trained models might not perform as well when applied to different programming languages, other domains outside of software development, or even different types of text within software development (e.g., code comments, bug reports, etc.).

Also, while we used two widely-used sentiment analysis tools, the findings might not be generalized to all sentiment analysis tools. The generalization of our findings might also be limited by the scope of our data, as the sentiment of developers could be influenced by a variety of factors that were not captured in our dataset.

Statistical conclusion validity is concerned with the extent to which conclusions about the relationship among variables based on the data are correct or "reasonable". In our study, we have used standard statistical measures, including precision, recall, F1-score for comparing the performances of standard and expanded versions of the models. Although these measures are widely accepted in the field, they each have inherent limitations. For instance, precision, recall, and F1-score are threshold-dependent measures, meaning that their values can change significantly depending on the chosen threshold for classification. This could potentially lead to different conclusions about the models' performances and reliability of our statistical conclusions.

Moreover, while the performance improvements observed in the expanded versions of the models were statistically significant, we must be cautious about overinterpreting these results. The practical significance of these improvements, especially in terms of their impact on real-world applications, requires further investigation.

Finally, researcher bias may pose a potential threat to the validity of the study, as it can subtly influence numerous aspects of the research process, from the collection and interpretation of data to the presentation and interpretation of the findings.

In the context of our study, researcher bias could primarily manifest in the manual labeling of both the Senti4SD dataset, and the additional 3000 GitHub issue comments sample dataset created in section 3.4. Given that these datasets were manually labeled by the dataset's creators and also by us, there may be inherent subjectivity and bias in the classification of sentiments. Our interpretations of sentiments could have been influenced by our personal experiences, beliefs, and knowledge, potentially leading to inconsistent or skewed labeling. Even our opinions on interpreting the developers' comments might differ.

The risk of confirmation bias also exists, where we might unintentionally highlight results that align with our hypotheses and underplay or disregard results that counter them. This can influence the overall conclusion of the study and its perceived effectiveness.

In summary, while our study presents a comparison between standard and expanded versions of sentiment analysis tools within the context of software development, the threats to validity outlined above should be considered when interpreting our findings. Further research is needed to validate and extend our results.

Chapter 5

Conclusion

As supported by current literature, the introduction of new train data is often a good strategy to increase a model's classification performance until it reaches a theoretical accuracy limit. In the present work, we show the presence of this phenomena by comparing two sentiment analysis tools against variations of themselves subjected to additional train data of the other.

Evidence indicates that by increasing SentiCR's train dataset, the classifier has improved its precision leading to the hypothesis that SentiCR's algorithm is still susceptible to improvement by expanding its train dataset. Whereas, SentiSW didn't express such a significant benefit, and thus, other areas of improvement such as fine-tuning the training process by parameter weights modification, or algorithm model implementation could be investigated.

On our second part of the study we obtained sentiment analysis data labels from issue texts from OSS hosted on Github. Results show that as the OSS project's size increases, the more migrations are happening from Negative to Non-Negative classes. Across the 3 most popular programming languages, Python was the most negative and second most neutral. In contrast, Java was the least Negative and the most Positive. Lastly, Javascript ranked most neutral of the three.

5.1 Future Work

The study undertaken provides significant insights into the use of sentiment analysis tools, showcasing their potential utility and outlining their current limitations. It demonstrates the potential benefits of incorporating larger datasets into the training phase of these tools. This promising avenue of investigation, however, opens up a plethora of further research opportunities.

While this study has achieved its set objectives, it is essential to acknowledge certain limitations. Specifically, the potential for researcher bias and threats to external validity could potentially impact the study's generalization.

There is room for further exploration in several areas. An extension of the current study could involve the application of the sentiment analysis tool to a wider range of data sources, such as pull requests or commits. This could provide a more comprehensive understanding of the tool's efficacy in other software engineering contexts.

Methodologically, future work could focus on refining the manual labeling process to mitigate potential researcher bias, perhaps through the use of a more diverse group of annotators. Further research could also consider developing and applying more robust methods to increase the statistical conclusion validity, including the use of advanced statistical models or machine learning algorithms.

Bibliography

[1] Rishi, Deepak, "Affective sentiment and emotional analysis of pull request comments on github," M.S. thesis, 2017. [Online]. Available: http://hdl.handle.net/10012/12728.

[2] H. S. Qiu, Y. L. Li, S. Padala, A. Sarma, and B. Vasilescu, "The signals that potential contributors look for when choosing open-source projects," in *ACM Conference on Computer-Supported Cooperative Work and Social Computing*, ser. CSCW, ACM, 2019. DOI: https://doi.org/10.1145/3359224.

[3] M. Ortu, B. Adams, G. Destefanis, P. Tourani, M. Marchesi, and R. Tonelli, "Are bullies more productive? empirical study of affectiveness vs. issue fixing time," May 2015. DOI: 10.1109/MSR.2015.35.

[4] M.-A. Storey, C. Treude, A. van Deursen, and L.-T. Cheng, "The impact of social media on software engineering practices and tools," in *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, ser. FoSER '10, Santa Fe, New Mexico, USA: Association for Computing Machinery, 2010, pp. 359–364, ISBN: 9781450304276. DOI: 10.1145/1882362.1882435. [Online]. Available: https://doi.org/10.1145/1882362.1882435.

[5] L. Dabbish, H. Stuart, J. Tsay, and J. Herbsleb, "Social coding in github: Transparency and collaboration in an open software repository," Feb. 2012, pp. 1277–1286. DOI: 10.1145/2145204.2145396.

[6] S. C. Müller and T. Fritz, "Stuck and frustrated or in flow and happy: Sensing developers' emotions and progress," in *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ser. ICSE '15, Florence, Italy: IEEE Press, 2015, pp. 688–699, ISBN: 9781479919345.

[7] N. Raman, M. Cao, Y. Tsvetkov, C. Kästner, and B. Vasilescu, "Stress and burnout in open source: Toward finding, understanding, and mitigating unhealthy interactions," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, 2020, pp. 57–60.

[8] K. Darwish, N. Habash, M. Abbas, *et al.*, "A panoramic survey of natural language processing in the arab world," *Commun. ACM*, vol. 64, no. 4, pp. 72–81, Mar. 2021, ISSN: 0001-0782. DOI: 10.1145/3447735. [Online]. Available: https://doi.org/10.1145/3447735.

[9] D. Meurers, "Natural language processing and language learning," *Encyclopedia of applied linguistics*, pp. 4193–4205, 2012.

[10] M. M. Lopez and J. Kalita, *Deep learning applied to nlp*, 2017. DOI: 10.48550/ARXIV.1703.03091. [Online]. Available: https://arxiv.org/abs/1703.03091.

[11] E. D. Liddy, "Natural language processing," in *Encyclopedia of Library and Information Science, 2nd Ed. NY. Marcel Decker, Inc*, 2001, pp. 7–13.

[12] F. Hogenboom, F. Frasincar, and U. Kaymak, "An overview of approaches to extract information from natural language corpora," Jan. 2010.

[13] M. Leo, S. Sharma, and K. Maddulety, "Machine learning in banking risk management: A literature review," *Risks*, vol. 7, p. 29, Mar. 2019. DOI: 10.3390/risks7010029.

[14] J. G. Greener, S. M. Kandathil, L. Moffat, and D. T. Jones, "A guide to machine learning for biologists," *Nature Reviews Molecular Cell Biology*, vol. 23, no. 1, pp. 40–55, 2022.

[15] S. Ghareeb, A. Hussain, D. Al-Jumeily Obe, *et al.*, "Evaluating student levelling based on machine learning model's performance," *Discover Internet of Things*, vol. 2, May 2022. DOI: 10.1007/s43926-022-00023-0.

[16] Y. Reddy, V. Pulabaigari, and E. B, "Semi-supervised learning: A brief review," *International Journal of Engineering & Technology*, vol. 7, p. 81, Feb. 2018. DOI: 10.14419/ijet.v7i1.8.9977.

[17] K. R. Dalal, "Analysing the role of supervised and unsupervised machine learning in iot," in *2020 international conference on electronics and sustainable communication systems (ICESC)*, IEEE, 2020, pp. 75–79.

[18] Y. Chong, Y. Ding, Q. Yan, and S. Pan, "Graph-based semi-supervised learning: A review," *Neurocomputing*, vol. 408, pp. 216–230, 2020, ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2019.12.130. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231220304938.

[19] Y. Reddy, V. Pulabaigari, and E. B, "Semi-supervised learning: A brief review," *International Journal of Engineering & Technology*, vol. 7, p. 81, Feb. 2018. DOI: 10.14419/ijet.v7i1.8.9977.

[20] S. Vieira, W. Lopez Pinaya, and A. Mechelli, "Introduction to machine learning," English, in *Machine Learning*. Netherlands: Elsevier, Jan. 2019, pp. 1–20, ISBN: 9780128157398. DOI: 10.1016/B978-0-12-815739-8.00001-8.

[21] K. Mak, K. Lee, and C. Park, "Applications of machine learning in addiction studies: A systematic review," *Psychiatry Research*, vol. 275, Mar. 2019. DOI: 10.1016/j.psychres.2019.03.001.

[22] S. Spanò, G. C. Cardarilli, L. Di Nunzio, *et al.*, "An efficient hardware implementation of reinforcement learning: The q-learning algorithm," *IEEE Access*, vol. 7, pp. 186 340–186 351, 2019. DOI: 10.1109/ACCESS.2019.2961174.

[23] A. Khan and A. Lapkin, "Searching for optimal process routes: A reinforcement learning approach," *Computers & Chemical Engineering*, vol. 141, p. 107 027, 2020, ISSN: 0098-1354. DOI: https://doi.org/10.1016/j.compchemeng.2020.107027. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0098135420303999.

[24] Z. Mammeri, "Reinforcement learning based routing in networks: Review and classification of approaches," *IEEE Access*, vol. 7, pp. 55 916–55 950, 2019. DOI: 10.1109/ACCESS.2019.2913776.

[25] D. Spadini, M. Aniche, and A. Bacchelli, "Pydriller: Python framework for mining software repositories," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2018, Lake Buena Vista, FL, USA: Association for Computing Machinery, 2018, pp. 908–911, ISBN: 9781450355735. DOI: 10.1145/3236024.3264598. [Online]. Available: https://doi.org/10.1145/3236024.3264598.

[26] T. Siddiqui and A. Ahmad, "Data mining tools and techniques for mining software repositories: A systematic review," in Jan. 2018, pp. 717–726, ISBN: 978-981-10-6619-1. DOI: 10.1007/978-981-10-6620-7_70.

[27] C. Gote, I. Scholtes, and F. Schweitzer, "Git2net: Mining time-stamped co-editing networks from large git repositories," in *Proceedings of the 16th International Conference on Mining Software Repositories*, IEEE Press, 2019, pp. 433–444.

[28] L. Zhang, S. Wang, and B. Liu, "Deep learning for sentiment analysis: A survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 4, e1253, 2018.

[29] P. Nandwani and R. Verma, "A review on sentiment analysis and emotion detection from text," *Social Network Analysis and Mining*, vol. 11, 2021.

[30] M. Birjali, M. Kasri, and A. Beni-Hssane, "A comprehensive survey on sentiment analysis: Approaches, challenges and trends," *Knowledge-Based Systems*, vol. 226, p. 107 134, 2021, ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2021.107134. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S095070512100397X.

[31] M. Birjali, M. Kasri, and A. Beni-Hssane, "A comprehensive survey on sentiment analysis: Approaches, challenges and trends," *Knowledge-Based Systems*, vol. 226, p. 107 134, 2021, ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2021.107134. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S095070512100397X.

[32] T. Ahmed, A. Bosu, A. Iqbal, and S. Rahimi, "Senticr: A customized sentiment analysis tool for code review interactions," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE, 2017, pp. 106–111.

[33] J. Ding, H. Sun, X. Wang, and X. Liu, "Entity-level sentiment analysis of issue comments," in *Proceedings of the 3rd International Workshop on Emotion Awareness in Software Engineering*, 2018, pp. 7–13.

[34] B. Zieliński, M. Lipiński, M. Juda, M. Zeppelzauer, and P. Dłotko, "Persistence bag-of-words for topological data analysis," *arXiv preprint arXiv:1812.09245*, 2018.

[35] J. Barry, "Sentiment analysis of online reviews using bag-of-words and lstm approaches.," in *AICS*, 2017, pp. 272–274.

[36] D. Yan, K. Li, S. Gu, and L. Yang, "Network-based bag-of-words model for text classification," *IEEE Access*, vol. 8, pp. 82 641–82 652, 2020.

[37] R. Zhao and K. Mao, "Fuzzy bag-of-words model for document representation," *IEEE transactions on fuzzy systems*, vol. 26, no. 2, pp. 794–804, 2017.

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

[39] A. H. Ombabi, O. Lazzez, W. Ouarda, and A. M. Alimi, "Deep learning framework based on word2vec and cnnfor users interests classification," in *2017 Sudan Conference on Computer Science and Information Technology (SCCSIT)*, 2017, pp. 1–7. DOI: 10.1109/SCCSIT.2017.8293054.

[40] S. Sivakumar, L. S. Videla, T. Rajesh Kumar, J. Nagaraj, S. Itnal, and D. Haritha, "Review on word2vec word embedding neural net," in *2020 International Conference on Smart Electronics and Communication (ICOSEC)*, 2020, pp. 282–290. DOI: 10.1109/ICOSEC49089.2020.9215319.

[41] M. Bilgin and H. Köktaş, "Sentiment analysis with term weighting and word vectors," *International Arab Journal of Information Technology*, vol. 5, pp. 953–959, Sep. 2019.

[42] J. H. Lau and T. Baldwin, "An empirical evaluation of doc2vec with practical insights into document embedding generation," *CoRR*, vol. abs/1607.05368, 2016. arXiv: 1607.05368. [Online]. Available: http://arxiv.org/abs/1607.05368.

[43] Q. V. Le and T. Mikolov, *Distributed representations of sentences and documents*, 2014. arXiv: 1405.4053 [cs.CL].

[44] M. Avinash and E. Sivasankar, "A study of feature extraction techniques for sentiment analysis," in *Emerging Technologies in Data Mining and Information Security*, A. Abraham, P. Dutta, J. K. Mandal, A. Bhattacharya, and S. Dutta, Eds., Singapore: Springer Singapore, 2019, pp. 475–486, ISBN: 978-981-13-1501-5.

[45] J. H. Lau and T. Baldwin, *An empirical evaluation of doc2vec with practical insights into document embedding generation*, 2016. arXiv: 1607.05368 [cs.CL].

[46] M. Bilgin and H. Köktaş, "Sentiment analysis with term weighting and word vectors," *International Arab Journal of Information Technology*, vol. 5, pp. 953–959, Sep. 2019.

[47] S. Lee, X. Jin, and W. Kim, "Sentiment classification for unlabeled dataset using doc2vec with jst," Aug. 2016, pp. 1–5. DOI: 10.1145/2971603.2971631.

[48] D. Kim, D. Seo, S. Cho, and P. Kang, "Multi-co-training for document classification using various document representations: Tf–idf, lda, and doc2vec," *Information Sciences*, vol. 477, pp. 7–11, 2019, ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2018.10.006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025518308028.

[49] A. Madasu and S. Elango, "A study of feature extraction techniques for sentiment analysis," *CoRR*, vol. abs/1906.01573, 2019. arXiv: 1906.01573. [Online]. Available: http://arxiv.org/abs/1906.01573.

[50] M. Chiny, M. Chihab, O. Bencharef, and Y. Chihab, "Lstm, vader and tf-idf based hybrid sentiment analysis model," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 7, 2021. DOI: 10.14569/IJACSA.2021.0120730. [Online]. Available: http://dx.doi.org/10.14569/IJACSA.2021.0120730.

[51] C.-z. Liu, Y.-x. Sheng, Z.-q. Wei, and Y.-Q. Yang, "Research of text classification based on improved tf-idf algorithm," in *2018 IEEE International Conference of Intelligent Robotic and Control Engineering (IRCE)*, 2018, pp. 218–222. DOI: 10.1109/IRCE.2018.8492945.

[52] I. Yahav, O. Shehory, and D. Schwartz, "Comments mining with tf-idf: The inherent bias and its removal," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 3, pp. 437–450, 2019. DOI: 10.1109/TKDE.2018.2840127.

[53] A. F. Yusrifa Deta Kirana, "Sentiment analysis of beauty product reviews using the k-nearest neighbor (knn) and tf-idf methods with chi-square feature selection," *Journal of Data Science & Its Application*, vol. 4, no. 1, 2021. DOI: 10.34818/jdsa.2021.4.71. [Online]. Available: https://doi.org/10.34818/jdsa.2021.4.71.

[54] M. Sánchez-Gordón and R. Colomo-Palacios, "Taking the emotional pulse of software engineering — a systematic literature review of empirical studies," *Information and Software Technology*, vol. 115, Aug. 2019. DOI: 10.1016/j.infsof.2019.08.002.

[55] D. Garcia, M. Zanetti, and F. Schweitzer, "The role of emotions in contributors activity: A case study on the gentoo community," *International Conference on Social Computing and Its Applications*, Jun. 2013. DOI: 10.1109/CGC.2013.71.

[56] F. Jurado and P. Rodriguez, "Sentiment analysis in monitoring software development processes: An exploratory case study on github's project issues," *Journal of Systems and Software*, vol. 104, pp. 82–89, 2015, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2015.02.055. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121215000485.

[57] R. Souza and B. Silva, "Sentiment analysis of travis ci builds," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, IEEE, 2017, pp. 459–462.

[58] N. Novielli, D. Girardi, and F. Lanubile, "A benchmark study on sentiment analysis for software engineering research," Mar. 2018. DOI: 10.1145/3196398.3196403.

[59] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, M. Lanza, and R. Oliveto, "Sentiment analysis for software engineering: How far can we go?" In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, 2018, pp. 94–104. DOI: 10.1145/3180155.3180195.

[60] W. M. Theunissen, A. Boake, and D. G. Kourie, "In search of the sweet spot: Agile open collaborative corporate software development," in *ACM International Conference Proceeding Series*, vol. 150, 2005, pp. 268–277.

[61] F. Calefato, F. Lanubile, F. Maiorano, and N. Novielli, "Sentiment polarity detection for software development," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1352–1382, Sep. 2017. DOI: 10.1007/s10664-017-9546-9. [Online]. Available: https://doi.org/10.1007%2Fs10664-017-9546-9.

[62] G. Gousios, "The ghtorent dataset and tool suite," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR '13, San Francisco, CA, USA: IEEE Press, 2013, pp. 233–236, ISBN: 9781467329361.

[63] G. Gousios, P. Rani, D. Spinellis, and Srikanth, *Github mirror readme*, `https://github.com/gousiosg/github-mirror/blob/master/sql/README.md`, 2019.