Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Verbesserung der Stimmungsanalyse basierend auf schriftlicher Kommunikation in Entwicklerteams durch Kombination existierender Ansätze

Bachelorarbeit

im Studiengang Informatik

von

Henrik Holm

Prüfer: Prof. Dr. rer. nat. Kurt Schneider
Zweitprüfer: Dr. rer. nat. Jil Ann-Christin Klünder
Betreuer: M.Sc. Martin Obaidi, M.Sc. Larissa Chazette

Hannover, 12. Mai 2021

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 12. Mai 2021

_____

Kurzfassung

In der modernen Softwareentwicklung ist ein positives Arbeitsklima unter Entwicklern unverzichtbar. Dieses erhöht die Produktivität und die Entwickler erzielen im Schnitt bessere Ergebnisse, wenn sie glücklicher sind. Zu diesem Zweck werden Stimmungsanalysetools verwendet, um die schriftliche Kommunikation innerhalb eines Entwicklerteams bezüglich des Sentiments und der Gefühlslage auszuwerten. Basierend auf den so gewonnenen Daten können eventuelle Probleme frühzeitig erkannt und behoben werden.

Allerdings sind die guten Klassifizerungsergebnisse bisheriger Tools meist nur spezifisch für die Domäne des Trainingsdatensatzes replizierbar. Dies erschwert die Analyse von Testdaten aus gänzlich unbekannten Domänen und führt zu Klassifizierungsgenauigkeiten von meist unter 70%. Eine mögliche Folge dieser niedrigen Genauigkeit könnte die falsche Einschätzung der Gefühlslage innerhalb eines Teams sein.

Ein Ansatz, um diesem Verhalten entgegen zu wirken, ist es, einen sogenannten Votingclassifier zu verwenden, welcher mehrere Klassifizierungstools in einem Ensemble vereint. Hierbei bietet sich auch die Option, jedes Tool auf einem Datensatz einer unterschiedlichen Domäne zu trainieren und so eine breitere Domänenabdeckung zu erzielen.

In dieser Arbeit wird ein selbst entworfener Votingclassifier, bestehend aus drei Tools, betrachtet und bezüglich seiner Performanz auf domänenfremden Testdaten ausgewertet. Ebenfalls wird analysiert, ob das Ensemble in derselben Domäne ein besseres Ergebnis erzielt als jedes der drei Einzeltools.

Abstract

In the field of modern software development a positive working climate is essential. It enhances productivity because a happy developer is prone to delivering better results. Sentiment analysis tools have been developed to analyze and classify the contents of textual communication between developers. Based on this data it is often possible to identify team internal problems early and handle them before they need to be escalated.

Though most of these tools deliver promising results when used with test data from the domain they were also trained in, the results when testing with data from a different domain are often lackluster. While most tools can deliver an accuracy of above 80% when used inside the training domain, the accuracy often falls below 60% when used outside of those domains. This can lead to misinterpretation of situations when using classified data to assess the teams mood.

To counteract this behaviour it is possible to use the approach of a Votingclassifier. This method combines multiple tools in an ensemble to cast a majority vote when doing a classification task. Because it is possible to combine multiple domains when using multiple tools, the coverage of domains will be broader and results wont neccesarily be bound to one specific domain.

In this thesis I will explore a custom Votingclassifier that combines three tools and analyze if we can achieve better results when classifying data from outside the training domain. We will also explore the possibilities to improve results in a specific domain even further by using an ensemble instead of only one specific tool.

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Problemstellung . . . 2
- 1.2 Motivation und Lösungsansatz . . . 2
- 1.3 Struktur . . . 5

2 Grundlagen **6**
- 2.1 Stimmungsanalyse . . . 7
- 2.2 Zusammenhang von Emotionen und Produktivität . . . 8
- 2.3 Klassifizierung . . . 8
- 2.4 Kreuzvalidierung von Ergebnissen . . . 9
- 2.5 Relevante Messwerte . . . 9
  - 2.5.1 Cohen's Kappa . . . 11
- 2.6 Verwendete Tools . . . 12
  - 2.6.1 Senti4SD . . . 12
  - 2.6.2 Klassifizierung von Meyer . . . 13
  - 2.6.3 Vortrainiertes RoBERTa Modell . . . 13
- 2.7 Votingclassifier . . . 14
- 2.8 Python Bibliotheken . . . 15

3 Verwandte Arbeiten **17**
- 3.1 Goldstandard Datensätze . . . 18
- 3.2 Stimmungsanalyse in der Softwareentwicklung . . . 19

4 Konzept **21**
- 4.1 Vortraining der Tools . . . 21
  - 4.1.1 Tool von Meyer und Horstmann . . . 21
  - 4.1.2 *RoBERTa* . . . 22
  - 4.1.3 Senti4SD . . . 22
- 4.2 Auswahl der Datensätze . . . 23
- 4.3 Auswertung der Einzelergebnisse . . . 25
- 4.4 Aufbau des Votingclassifiers . . . 26

4.5 Ergebnisse eines Durchlaufs 28

5 Auswertungen **30**
5.1 Domäneninterne Klassifizierung 30
5.2 Fremde Testdomänen 33
5.3 Vergleich Einzeltools und Votingclassifier 36
5.4 Threats to Validity 39

6 Zusammenfassung und Ausblick **40**
6.1 Zusammenfassung 40
6.2 Verbesserung des Votingclassifiers 41
6.3 Ausblick 42

Kapitel 1

Einleitung

Die Kommunikation unter beteiligten Entwicklern spielt in dem Bereich der Informatik und insbesondere der Softwareentwicklung eine große Rolle [4]. Durch Analyse und Auswertung eben dieser Kommunikation ist es in manchen Fällen möglich, frühzeitig eventuelle Konflikte und Stimmungseinbrüche innerhalb eines Teams zu erkennen [7]. Für den Erfolg eines Softwareprojektes ist ein gutes Arbeitsklima und das schnelle Beseitigen teaminterner Probleme essentiell, da glückliche Entwickler in der Regel auch produktiver und effizienter arbeiten [6] [20].

Die in 2017 von Kuhrmann et al. [11] durchgeführte HELENA Studie ergab, dass zu dem Zeitpunkt der Studie 60% aller Entwicklerteams (n = 1006) in einem globalen Kontext und örtlich verteilt arbeiten [11]. Besonders in quell-offenen Projekten auf kollaborativen Entwicklungsplattformen wo Entwickler über größere Distanzen zusammen arbeiten spielt die Kommunikation eine große Rolle [10].

Bei firmeninternen Projekten findet die Kommunikation meist über geschlossene Kanäle statt, ein Zugriff auf solche Daten ist aufgrund vieler sicherheitsrelevanter Aspekte nur schwer möglich [9]. Deswegen wird in dieser Arbeit darauf verzichtet und es werden Datensätze aus öffentlichen Kanälen verwendet, darunter Github, Stackoverflow[1] und Jira[2]. Weitere Informationen zu den besagten Datensätzen sind in Kapitel 3 zu finden.

[1] https://stackoverflow.com/

[2] https://www.atlassian.com/de/software/jira

1.1 Problemstellung

Bereits existierende Tools für die Stimmungsanalyse von Kommunikation innerhalb Entwicklerteams erreichen zwar schon Klassifizierungsgenauigkeiten über 80% [1], allerdings sind diese Ergebnisse meist spezifisch auf die Domäne des Trainingsdatensatzes beschränkt [16]. Hierbei erreicht das Tool Senti4SD von Calefato et al. [1] mit Training auf dem Stackoverflow Goldstandard Datensatz [18] und anschließender Klassifizierung in der selben Domäne eine Genauigkeit von bis zu 87% [1].

Sobald diese Tools nun aber ohne erneutes Vortraining auf einen Datensatz einer anderen Domäne angewendet werden, zum Beispiel ein Wechsel von Jira auf Github als Testdomäne, nimmt die Genauigkeit der Klassifizierungsergebnisse merklich ab [16]. Für das vorhergehend genannte Beispiel unter Verwendung des Tools Senti4SD [1] ist eine Abnahme des F1-Wertes (Macro-Average) von 26% zu beobachten [16].

1.2 Motivation und Lösungsansatz

Aufgrund des großen Unterschiedes zwischen den verschiedenen Datensätzen, wäre es nun wünschenswert, eine Lösung zu finden, welche auch auf nicht bereits im Training verwendeten Domänen ebenfalls vergleichbar gute Ergebnisse erzielt. Ein eventuelles Anwendungsgebiet einer solch flexiblen Klassifizierung könnte zum Beispiel die teamübergreifende Stimmungsanalyse innerhalb einer Firma sein. So kann es möglich sein, Probleme frühzeitig zu identifizieren und eliminieren, die Softwareentwickler sind entsprechend glücklicher und effizienter [6] [7].

Um die Genauigkeit der Klassifizierung in den in Kapitel 1.1 beschriebenen Szenarien zu verbessern, wird in dieser Arbeit der Ansatz eines selbst entworfenen Votingclassifiers untersucht. Hierbei werden drei unterschiedliche Tools der Stimmungsanalyse in der Softwareentwicklung kombiniert, um eine Mehrheitsentscheidung über die jeweilige Eingabe in Form eines natürlichsprachigen Textes zu treffen. Dabei fiel die Entscheidung auf die folgenden drei Tools:

Das bereits erwähnte Senti4SD[3] von Calefato et al. [1], da es mit einem klassischen Machine Learning Ansatz aussagekräftige Ergebnisse erzielt.

Des Weiteren wird ein vortrainiertes RoBERTa Modell [13] [23] verwendet, welches im Gegensatz zu den anderen beiden Tools den Ansatz eines selbst lernenden Neuronalen Netzes verwendet. Die Entscheidung fiel auf eben dieses Modell, da es im durchgeführten Vergleich von Zhang

[3] https://github.com/collab-uniba/pySenti4SD

et al. [26] die durchschnittlich besten Ergebnisse erzielte im Vergleich zu anderen, vortrainierten Modellen.

Die dritte Komponente bildet ein selbstentwickelter Ansatz von Christian Meyer [15], basierend auf der Masterarbeit von Julian Horstmann [9]. Hierbei wird eine Kombination verschiedener, klassischer Machine Learning Verfahren verwendet, darunter *Support Vector Machines*, *Random Forests* und *Naive Bayes Classifier* [15]. Auf die Verwendung der Erweiterung in Form eines evolutionären Algorithmus von Meyer wird in dieser Arbeit verzichtet, da dieser zu keinem messbaren Anstieg der Performanz geführt hat [15].

Berechnet werden die genannten Werte mit dem Python Modul scikit-learn[4] unter Verwendung der Funktion *classification_report* [5].

Somit ist es nun möglich, im Ensemble drei jeweils unterschiedliche Domänen zu betrachten und sich den Forschungsfragen zu widmen.

Basierend auf diesen Ansätzen sollen im Laufe dieser Arbeit die folgenden drei Forschungsfragen näher erörtert werden:

RQ1 Erzielt ein Ensemble aus Klassifizierungstools höhere Klassifizierungsgenauigkeiten als die verwendeten Einzeltools innerhalb derselben Domäne eines Datensatzes?

RQ2 Wie verhält sich die Klassifizierungsgenauigkeit des Votingclassifiers, wenn sich die Trainingsdomänen der Einzeltools gänzlich von der Testdomäne unterscheiden?

RQ3 Inwiefern unterscheidet sich die Klassifizierungsgenauigkeit des in **RQ2** beschriebenen Votingclassifiers von der im Votingclassifier enthaltenen Einzeltools bei Training dieser in der Testdomäne?

[4]https://scikit-learn.org/stable/index.html

[5]https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

RQ1 betrachtet für alle drei Tools jeweils die Modelle mit der höchsten Klassifizierungsgenauigkeit aus einem Split innerhalb eines Datensatzes, beispielsweise alle drei Tools trainiert auf dem Github Goldstandard. Anschließend folgt eine Klassifizierung und Auswertung auf ungesehenen Github Testdaten.

Dadurch soll der Frage nachgekommen werden, ob ein Votingclassifier in dieser Konstellation genauere Klassifizierungsergebnisse liefern kann als das Beste der drei Einzeltools in dem jeweiligen Durchlauf. Näheres bezüglich dieses Konzeptes wird in Kapitel 4 erläutert.

Anschließend wird in **RQ2** betrachtet, ob eine Zusammenstellung an Modellen existiert, für die der Votingclassifier eine höhere Klassifizierungsgenauigkeit auf domänenfremden Testdaten erreichen kann, als es die Einzeltools schaffen würden. Beispielsweise ist hier eine Belegung der Tools mit Modellen aus drei unterschiedlichen Domänen möglich.

Zusätzlich wird im Ensemble ein Datensatz aus gänzlich unbekannter Domäne klassifiziert, ausgewertet und die Performanz der Einzeltools sowie des Votingclassifiers gegenüber gestellt. Dieser Vorgang wird für unterschiedliche Belegungen der Tools mit verschiedenen Modellen wiederholt. Weitere Details zu den Konstellationen und Auswertungsverfahren sind ebenfalls im Konzept in Kapitel 4 zu finden.

Zusammenhängend mit **RQ2** wird in **RQ3** betrachtet, inwiefern die Ergebnisse der Zusammenstellungen des Votingclassifiers sich von den Ergebnissen von domäneninternen Tools unterscheiden. Dies bedeutet einen zusätzlichen Vergleich mit Klassifizierungsgenauigkeiten der im Votingclassifier verwendeten Einzeltools. Der Unterschied hierbei ist, dass die Einzeltools innerhalb der Testdomäne trainiert wurden.

1.3 Struktur

Im folgenden Kapitel 2 werden die Grundlagen der Stimmungsanalyse und der verwendeten Tools erläutert. Dabei wird besonders auf die Stimmungsanalyse innerhalb des Gebietes der Softwareentwicklung und die Klassifizierung von natürlichsprachigen Texten eingegangen. Ebenfalls wird die Funktionsweise des Votingclassifiers sowie die erhobenen Messwerte erläutert.

In Kapitel 3 wird auf verwandte Arbeiten eingegangen. Hierzu gehören jeweils die verwendeten Tools der Klassifizierung, aber auch die Auswahl an Datensätzen, welche für Training und Auswertung verwendet werden. Ebenfalls wird auf Arbeiten eingegangen, welche sich mit den Zusammenhängen von Emotionen in der Softwareentwicklung und der Performanz von Entwicklern beschäftigen.

Anschließend wird in Kapitel 4 das verwendete Konzept des Vortrainings der Tools und deren Zusammenstellung erläutert. Hierbei wird ebenfalls darauf eingegangen, warum die entsprechenden Tools als Bestandteil ausgewählt wurden. Es werden alle verwendeten Konstellationen des Votingclassifiers aufgezählt und für den weiteren Verlauf der Arbeit entsprechend benannt.

In Kapitel 5 werden die gesammelten Ergebnisse und dazugehörigen Metriken ausgewertet und anhand dieser die gewählten Konfigurationen des Votingclassifiers bewertet. Zum Abschluss werden eventuelle Verbesserungen identifiziert und die gesammelten Ergebnisse in Relation zu den Einzelergebnissen der jeweiligen Tools gestellt.

Abschließend werden in Kapitel 6 die Ergebnisse zusammengefasst und in Relation zu anderen Arbeiten gestellt. Ebenfalls wird ein Ausblick gestellt, welche weitergehende Forschung basierend auf den Erkenntnissen dieser Arbeit denkbar wäre.

Kapitel 2

Grundlagen

In der modernen Softwareentwicklung ist die Kommunikation zwischen beteiligten Entwicklern besonders relevant [6]. Dazu werden im Zuge dieser Arbeit Methoden aus den Bereichen der *Stimmungsanalyse*, des *Natural Language Processing* und der *Klassifizierung* dieser Dokumente verwendet. Hierzu wurden Programme entwickelt, welche sich explizit mit dem Bereich der Stimmungsanalyse in der Softwareentwicklung beschäftigen. Diese Tools werden in den kommenden Abschnitten präsentiert und ihre Funktionsweise erläutert.

Um die Güte eines Modells, welches von einem Tool produziert wurde, zu bewerten, ist die Kenntnis bestimmter Messwerte nötig. Die wichtigsten dieser Werte werden in dem kommenden Kapitel mit zugehörigen Formeln und Erklärungen präsentiert. Ebenfalls werden Begriffe erläutert, welche für das Verständnis der Funktionsweise des sogenannten *Votingclassifiers* essentiell sind. Der Votingclassifier bildet das softwareseitige Kernstück zur Betrachtung der Forschungsfragen dieser Arbeit.

Der Votingclassifier wurde in der Programmiersprache Python[1] verfasst, ein Großteil der verwendeten Tools basiert ebenfalls auf Python Code. Deswegen werden in dem kommenden Kapitel ebenfalls die wichtigsten der verwendeten Bibliotheken erläutert.

[1] https://www.Python.org/

2.1 Stimmungsanalyse

Schriftliche Kommunikation in Form von natürlicher Sprache kann unter Umständen einen emotionalen Einfluss des Autors enthalten [7]. Die Praxis der Stimmmungsanalyse befasst sich mit der Extraktion und Interpretation dieser Gefühlslage in natürlichsprachiger Texten [22]. Dabei wird sich meistens auf den Aspekt der subjektiven Emotionen konzentriert, welche der Autor in seinen Text einfließen ließ [22].

In dieser Arbeit wird lediglich die übergeordnete Färbung eines Textes oder Satzes betrachtet. Dabei wird zwischen positiv und negativ gefärbtem Sprachgebrauch differenziert. Sätze, die in keine der beiden vorher genannten Kategorien fallen, werden als neutral gefärbt betrachtet. Durch eine positive oder negative Färbung eines Textes ist es eventuell möglich, Trends zu erkennen und Rückschlüsse auf die Gefühlslage des Autors zu ziehen [7].

Text des Autors	Label
“I LOVE this change.”	Positiv
“Oh.. you custom stuff freaks.. I still hate you.”	Negativ
“Read the commit message.”	Neutral

Tabelle 2.1: Beispielsätze aus dem Github Goldstandard [17].

Im Laufe dieser Arbeit werden natürlichsprachige Sätze gelegentlich auch als *Dokumente* bezeichnet und dessen Einordnung in eine Klasse als *Polarität*, *Label* oder *Sentiment*. Ebenfalls werden einzelne Worte, aus denen ein Dokument besteht, gelegentlich als *Tokens* bezeichnet. Tokens können sowohl Wörter, als auch Links oder Emojis innerhalb eines Satzes sein.

In der Vorverarbeitung von Dokumenten wird häufig die sogenannte *Lemmatisierung* von Tokens durchgeführt. Hierbei werden vereinzelt Wörter in ihre Grundform überführt (z.B. “reading the commit message” zu “read the commit message”, um eine bessere Vereinheitlichung zu gewährleisten [14].

Des Weiteren wird das Konzept der *Stoppwörter* verwendet, hierbei werden Tokens mit geringem bis gar keinem Informationsgehalt aus einem Dokument entfernt [14]. So wird letztendlich aus dem Beispiel “read the commit message” das Dokument “read commit message” nach Entfernen des Tokens *the*.

2.2 Zusammenhang von Emotionen und Produktivität

Graziotin et al. [7] untersuchten, ob ein Zusammenhang zwischen den Emotionen, Stimmungen und Gefühlen von Entwicklern und deren Selbsteinschätzung der Produktivität besteht. Dabei wurden vier professionelle Entwickler und vier Studenten bei einer Programmieraufgabe nach ihrer Einschätzung zur aktuellen Produktivität befragt [7]. Dabei beobachteten Graziotin et al. [7], dass Gefühle wie Dominanz und Wertigkeit positiv mit der Produktivitätseinschätzung korrelieren und es deswegen sinnvoll sein kann, Emotionsanalyse im Bereich der Softwareentwicklung zu betreiben.

Ebenfalls untersuchten Graziotin et. al [6] ob Entwickler, welche ihren aktuellen emotionalen Zustand als *glücklich* betiteln, eine bessere Problemlösungsfähigkeit im Bezug auf die Softwareentwicklung besitzen. Die erhobenen Daten legten dabei einen Zusammenhang zwischen der Gefühlslage der Entwickler und ihren analytischen Fähigkeiten zur Problemlösung nahe [6].

Ortu et al. [20] betrachteten Kommentare innerhalb von Jira Projekten im Bezug auf deren Affekt und die jeweilige Bearbeitungszeit. Unter dem Begriff Affekt wurden dabei die Eigenschaften *Emotion, Sentiment und Höflichkeit* zusammengefasst und es wurde untersucht, ob ein Zusammenhang zu der Bearbeitungszeit des Problems besteht. Sie stellten dabei fest, dass die Existenz bestimmter positiv konnotierter Emotionen mit einer Verringerung der Bearbeitungszeit eines Problems korreliert [20].

2.3 Klassifizierung

Unter der Klassifizierung eines Textes versteht man dessen Einordnung in eine bestimmte Klasse, für unseren Fall eine der drei emotionalen Färbungen, welche in dem Bereich *Stimmungsanalyse* angesprochen wurden [15] . Das Tool von Meyer und Horstmann nimmt hierbei einen natürlichsprachigen Satz und zerlegt ihn in seine Bestandteile (sogenannte *Tokens*) und analysiert diese [9] [15]. Selbiges Tokenisierungsverfahren verwendet auch das Tool Senti4SD [1]. Lediglich das RoBERTa basierte Modell greift auf den Ansatz eines neuronalen Netzes zurück. Weiteres zu den verwendeten Tools ist im Abschnitt 2.6 zu lesen.

Im Bereich der textuellen Kommunikation auf Entwicklerplattformen sind hierbei nicht nur Worte relevant, sondern auch spezielle Bauteile wie Hyperlinks oder Emojis in Form von Satzzeichen [2]. Hierbei können verwendete Emoticons Rückschlüsse auf die Gefühlslage des Autors geben. [2] Wie Meyer in seiner Arbeit bereits beobachtete, wirkt sich das Entfernen von

Hyperlinks aus dem Eingabetext positiv auf die Klassifizierungsgenauigkeit aus, da so eventuell entstandene Emoticons innerhalb eines Links nicht falsch interpretiert werden können [15].

2.4 Kreuzvalidierung von Ergebnissen

Meyer verwendet in seiner Arbeit das Konzept der stratifizierten Kreuzvalidierung in fünffacher Ausführung für das Training und Auswerten seines Tools [15]. Zur besseren Vergleichbarkeit wird dieses Prinzip für die anderen beiden Tools, welche in dieser Arbeit verwendet werden, ebenfalls angewandt. Hierbei wird der betrachtete Datensatz in fünf gleich große und zueinander disjunkte Mengen unterteilt [8]. Anschließend werden vier dieser Mengen zusammen gelegt und als Trainingsinput für das Modell verwendet, auf der verbleibenden Menge wird anschließend getestet und ausgewertet.

Bei der stratifizierten Variante der Kreuzvalidierung wird ebenfalls darauf geachtet, dass das Verhältnis der Label innerhalb eines Splits möglichst dem des Ursprungsdatensatzes entspricht [8].

Ein Vorteil dieses Verfahrens ist es, dass man die tatsächliche Performanz der Modelle besser annähern kann [8]. Die Betrachtung aller Daten in verschiedenen Konstellationen ermöglicht außerdem eine Minimierung von Störfaktoren, welche durch den Datensatz bedingt seien könnten [8].

Nach der Betrachtung aller Splits ist es nun möglich das Modell zu wählen, dessen Messwerte dem gewünschten Klassifizierungsverhalten nahe kommen. In dieser Arbeit wurde dabei hauptsächlich die Klassifizierungsgenauigkeit verwendet, bei Gleichstand wurde auf den F1-Wert ausgewichen.

2.5 Relevante Messwerte

Für die Auswertung der Performanz eines betrachteten Modells ist das Verständnis bestimmter Messwerte im Bezug auf Dokumente und Klassen wichtig.

True Positives (TP): Sätze, die von dem Modell korrekt in die betrachtete Klasse eingeordnet wurden.

False Positives (FP): Sätze, die fälschlicherweise von einer fremden Klasse der aktuell betrachteten Klasse zugeordnet wurden.

True Negatives (TN): Sätze einer fremden Klasse, die auch korrekt in diese fremde Klasse eingeordnet wurden.

False Negatives (FN): Sätze der aktuell betrachteten Klasse, die fälschlicherweise einer fremden Klasse zugeordnet wurden.

Support: Anzahl von Elementen einer Klasse innerhalb des betrachteten Datensatzes.

Die unteren Metriken sind für unsere Auswertung relevant und sind wie folgt definiert [14]:

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

Precision: Korrekt klassifizierte Elemente innerhalb einer Klasse in Abhängigkeit von der Gesamtsumme der dieser Klasse zugeordneten Elemente.

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

Recall: Korrekt klassifizierte Elemente innerhalb einer Klasse in Abhängigkeit von der Gesamtsumme der in dieser Klasse tatsächlich enthaltenen Elemente.

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

F-Wert: Bei Betrachtung des F-Wertes werden Precision und Recall gewichtet zusammen gefasst.

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

Accuracy: Hierbei handelt es sich um die Klassifizierungsgenauigkeit eines Modells. Der Wert repräsentiert die Gewichtung der korrekt klassifizierten Dokumente innerhalb der Gesamtanzahl der Elemente. Er ist für den weiteren Verlauf dieser Arbeit der Maßstab zur Bewertung der Güte von vorliegenden Modellen.

$$MacroAvg = \frac{\sum Ergebnisse}{|Ergebnisse|}$$

Macro Average: Dieser Wert repräsentiert den Durchschnitt der Ergebnisse eines bestimmten Messwertes. Man berechnet hierbei die Summe aller klassenspezifischen Ergebnisse und teilt sie durch die Anzahl der Klassen.

Diese Werte werden für jedes Klassifizierungsergebnis eines Tools erhoben und übersichtlich in einer Tabelle dargestellt. Berechnet werden die Werte von der Funktion *classification-report* des *Scikit-Learn* Packages[2] basierend auf der Liste der eigentlichen und vorhergesagten Labels.

[2]https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

2.5.1 Cohen's Kappa

Um zu messen, inwiefern die einzelnen Klassifizierungstools, welche innerhalb des Votingclassifiers verwendet werden, in ihren Entscheidungen übereinstimmen, wird als Wert das sogenannte **Cohen's Kappa** [3] verwendet.

Das Kappa repräsentiert ein bestimmtes Level, auf welchem sich zwei Parteien bei einem Klassifizierungsproblem einig sind [3]. Der Wert kann Fließkommazahlen zwischen -1 und 1 annehmen, wobei ein höherer Wert eine höhere Übereinstimmung bedeutet [3]. Innerhalb des Codes wird die Funktion *cohen_kappa_score*[3] für die Berechnung des Wertes verwendet und zusätzlich für jeden Durchlauf des Votingclassifiers abgespeichert. Es werden jeweils drei Vergleiche aufgestellt, da jeweils jedes Einzelergebnis mit den anderen verglichen wird.

Der Wert ist wie folgt definiert [3]:

$$k = \frac{(p_o - p_e)}{(1 - p_e)}$$

Wobei p_o die empirische Wahrscheinlichkeit repräsentiert, dass zwei Klassifizierer sich für ein betrachtetes Dokument auf dasselbe Label einigen [3]. Der Wert p_e bildet die erwartete Wahrscheinlichkeit einer Übereinstimmung, falls beide Klassifizierer zufällige Label setzen würden [3].

[3]https://scikit-learn.org/stable/modules/generated/sklearn.metrics.cohen_kappa_score.html

2.6 Verwendete Tools

2.6.1 Senti4SD

Bei Senti4SD handelt es sich um ein Tool zur Klassifizierung von natürlichsprachigen Texten aus dem Bereich der Softwareentwicklung [1]. Im Umfang dieser Arbeit wird die Adaption des Tools für die Programmiersprache Python verwendet, welches im Jahre 2019 von Calefato et al. auf Github bereitgestellt wurde [1][4].

Es wird eine Klassifizierungsgenauigkeit von 87% über alle Label erreicht bei Training und anschließender Auswertung auf dem Stackoverflow Goldstandard [18] [1]. Dieser Datensatz wurde im Zuge der Entwicklung des Tools Senti4SD erstellt [1]. Weitere Details zu den in dieser Arbeit verwendeten Datensätzen sind in Kapitel 3 zu finden.

Das Tool verwendet bereits existierende Sentimentlexika, um Dokumente anhand bestimmter Eigenschaften wie zum Beispiel der Anzahl an positiver oder negativer Emoticons zu bewerten [1]. Ebenfalls wird auf spezifische Merkmale zur Betonung von Kommunikation geachtet, beispielsweise wie viele Worte komplett groß geschrieben wurden oder ob sich Satzzeichen auffällig oft wiederholen (Beispielsweise die Zeichen *???* oder *!!!*) [1].

Trainiert werden die Modelle hierbei mit einer *Support Vector Machine*, welche die Möglichkeit besitzen auch bei einem hochdimensionalen Merkmalsraum zu lernen und generalisieren [1].

[4] https://github.com/collab-uniba/pySenti4SD

2.6.2 Klassifizierung von Meyer

Im Zuge seiner Bachelorarbeit optimierte Christian Meyer [15] das Klassifizierungstool von Julian Horstmann [9] durch Ergänzungen innerhalb der Merkmalsmenge und die Implementation eines domänenspezifischen Lexikons. Ebenfalls erweiterte er das Tool durch einen evolutionären Algorithmus [15]. In dieser Arbeit wird das Klassifizierungstool ohne Anwendung des evolutionären Algorithmus verwendet, da dieser zu keinem merklichen Anstieg der Klassifizierungsgenauigkeit führte [15].

Bei Training und Auswertung des Tools auf dem Github Goldstandard [17] ist eine Klassifizierungsgenauigkeit von 87% über alle Klassen zu beobachten. Diese Genauigkeit entspricht in etwa der, welche das Tool Senti4SD auf dem Stackoverflow Goldstandard [18] erreicht [1].

In diesem Tool werden verschiedene Methoden des maschinellen Lernens kombiniert und für das Training von Modellen verwendet [15] [9]. Darunter befinden sich ebenfalls *Support Vector Machines*, aber auch ein *Naive Bayes Classifier* sowie *Random Forests*. Ebenfalls wird das Konzept der fünffachen Kreuzvalidierung verwendet um eine möglichst wohl gewichtete Menge an Trainingssplits zu gewährleisten [15] [14].

2.6.3 Vortrainiertes RoBERTa Modell

Bei dem RoBERTa Modell handelt es sich um eine Abwandlung des BERT Modells, welches von Devlin et al. [5] entworfen wurde. Die Abkürzung steht für "Robustly Optimized BERT Pretraining Approach" [13]. Durch Anpassung der Hyperparameter und umfassenderes Vortraining wurde das Modell von Liu et al. [13] optimiert und übertrifft die Klassifizierungsgenauigkeiten des BERT Modells [5] um durchschnittlich zwei Prozentpunkte [26].

Neuronale Netze bieten einen alternativen Ansatz zu klassischen Verfahren des maschinellen Lernens innerhalb der Sentimentsanalyse [19]. Bei dem RoBERTa Modell wurde besonders viel Wert auf die Anpassung der initialen Trainingsdaten gelegt, um eine klare Linie zu dem BERT Modell zu ziehen und eventuelle Verbesserungen zu identifizieren [13]. Hierzu wurde eine größere Trainingsdatenmenge verwendet und es wurde länger, also in mehr Iterationen, trainiert [13]. Ebenfalls wurden längere Dokumente verwendet und es wurde ein dynamisch wechselndes *Masking Pattern* verwendet [13].

Diese Optimierungen haben dafür gesorgt, dass das das RoBERTa Modell fast alle anderen Modelle innerhalb der öffentlichen SQuAD 2.0 Tabelle im Bezug auf die Klassifizierungsgenauigkeit schlägt [13]. Deswegen fiel die Wahl des dritten Tools für diese Arbeit auf RoBERTa.

2.7 Votingclassifier

Eine Möglichkeit, mehrere Klassifizierer für das selbe Testdokument anzuwenden und eine Gemeinschaftsentscheidung zu treffen ist die Verwendung eines sogenannten *Votingclassifiers*. Meyer verwendete ebenfalls einen Votingclassifier zur Kombination klassischer Ansätze des maschinellen Lernens [15].

Dazu verwendete er das Python Package *Scikit-Learn*[5] und dessen Klasse *VotingClassifier*[6]. Da diese Klasse allerdings nur mit *Scikit-Learn* nativen Ansätzen funktioniert und keine eigenen Tools als Eingabe entgegen nehmen kann, wird ein eigener Votingclassifier in der Programmiersprache Python implementiert.

In dieser Arbeit wird ein Votingclassifier bestehend aus drei Komponenten verwendet, eine Mehrheitsentscheidung wird also getroffen, wenn das Klassifizierungsergebnis von mindestens zwei Tools übereinstimmt. Da in dieser Arbeit nur Datensätze mit drei möglichen Label betrachtet werden, ist es somit möglich, dass alle drei Tools unterschiedliche Label vorhersagen.

Sobald dieser Fall eintritt, wird von der Pythonbibliothek *secrets*[7] eine Zufallsentscheidung auf eines der drei möglichen Label getroffen.

An diesem Punkt sind ebenfalls andere Ansätze möglich, um einen Gleichstand aufzulösen. Diese Möglichkeiten werden in Kapitel 6 näher betrachtet und in Aussicht gestellt.

Während des Ablaufs meines Votingclassifiers werden alle drei vorhergehend genannten Tools einzeln ausgeführt, ihre jeweiligen Klassifizierungsergebnisse als *.csv* Datei abgespeichert und ausgewertet. Anschließend werden diese drei Dateien miteinander verglichen und es wird per Mehrheitsvotum eine vierte Datei erstellt, welche letztendlich das Ergebnis des Votingclassifiers bildet.

Ebenfalls erstellt mein Tool die *classification-reports*[8] für alle vier Ergebnisse, ein übersichtlicher Vergleich der Performanz und Accuracy ist somit möglich.

[5] https://scikit-learn.org/
[6] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
[7] https://docs.python.org/3/library/secrets.html
[8] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

2.8 Python Bibliotheken

Gemessen an Pull Requests auf Github im ersten Quartal des Jahres 2021 lag Python auf Platz 2 der am meisten verwendeten Programmiersprachen[9].

Durch die Nähe zur englischen Sprache und die unkomplizierte Verwendung des Python Interpreters war es mir möglich, in kurzer Zeit Skripte zu verfassen, welche besonders bei repetetiven Aufgaben wie dem Durchsuchen von Ordnerstrukturen oder Umstrukturieren von Textdateien eine große Hilfe waren. Eine Erweiterung der Standardsprache durch sogenannte Packages ist mit dem Tool *pip*[10] möglich.

Das Package **Pandas**[11] ermöglicht eine einfache Verwaltung und Manipulation von Daten innerhalb eines Python Skripts. Hierzu werden die gewünschten Daten bespielsweise aus einer .csv oder Exceldatei eingelesen und in einem sogenannten *DataFrame* Objekt gespeichert.

Auf diesen Objekten kann der Nutzer nun mathematische Operationen ausführen, Felder mit eigenen Funktionen manipulieren oder statistische Auswertungen durchführen. Der Hauptanwendungsfall in dieser Arbeit ist das Einlesen von Dokumenten und zugehörigen Labels, das anschließende Klassifizieren mit den gewünschten Tools und letztendlich ein Abspeichern der Ergebnisse in einer .csv Datei.

Mit dem Package **Numpy**[12] ist es möglich, innerhalb eines Pythonskriptes große Matrizen und Zahlenarrays zu verwalten. Meistens werden die Funktionen in Kombination mit *DataFrames* aus Pandas verwendet, um möglichst effizient Operationen auf einer Datenspalte, interpretiert als Array, auszuführen. Ein konkretes Beispiel dieser Arbeit ist die Mehrheitsentscheidung des Votingclassifiers. Hierbei werden alle drei Spalten der Einzelergebnisse miteinander verglichen, um entweder das überwiegende Label in eine neue Spalte zu schreiben oder bei Gleichstand ein entsprechendes Zufallsvotum durchzuführen.

[9] https://madnight.github.io/githut/#/pull_requests/2021/1
[10] https://pypi.org/project/pip/
[11] https://pandas.pydata.org/
[12] https://numpy.org/

Das Package **PyTorch**[13] bildet neben *Tensorflow*[14] die Grundlage für die Anwendung maschineller Lernmethoden innerhalb des Python Frameworks. Es liefert Funktionen wie das Trainieren und Auswerten neuronaler Netze und der grundlegenden Arbeit mit dessen Modellen. Besonders ist hierbei die mögliche Verwendung von einer oder mehreren Grafikkarten zur hardwareseitigen Beschleunigung der zeitintensiven Trainingsschritte.

Anwendung fand dieses Package in dieser Arbeit im Zusammenhang mit dem RoBERTa Modell [13] und dessen Vortraining auf verschiedenen Datensätzen sowie der anschließenden Auswertung innerhalb des Votingclassifiers. Zur Beschleunigung wurden Remotegrafikkarten unter Verwendung von *Google Colab*[15] bereitgestellt. Die verschiedenen Modelle wurden dort trainiert, dann heruntergeladen und können anschließend lokal zur Klassifizierung verwendet werden.

Eine Erweiterung von *PyTorch* um Funktionalitäten des Natural Language Processing bietet das Package **Transformers**[16]. Es ermöglicht das Laden des RoBERTa [13] Basismodelles und die anschließende Konfiguration mit vortrainierten Parametern. Außerdem wird es verwendet, um die Klassifizierung von Dokumenten innerhalb des RoBERTa [13] Tools durchzuführen.

Scikit-Learn findet unter anderem Anwendung innerhalb des Tools von Meyer [15] und stellt grundlegende Funktionen des maschinellen Lernens bereit. *Scikit-Learn*[17] übernimmt dort unter anderem Klassifizierungsaufgaben durch Verwendung von *Support Vector Machines*, *Random Forests* und *Naive Bayes* [15]. Innerhalb des Votingclassifiers dieser Arbeit wird das Package hauptsächlich zur Auswertung und Darstellung von Klassifizierungsergebnissen verwendet. Die Funktion *classification_report* übernimmt hierbei die Berechnung von Precision, Recall, F1-Wert und Accuracy basierend auf ursprünglichen und vorhergesagten Labels.

[13] https://pytorch.org/
[14] https://www.tensorflow.org/
[15] https://colab.research.google.com/
[16] https://huggingface.co/transformers/
[17] https://scikit-learn.org/stable/

Kapitel 3

Verwandte Arbeiten

Als Grundlage für Training und Auswertung von Performanz der verwendeten Tools dieser Arbeit werden drei Datensätze betrachtet. In diesem Kapitel wird die Erhebung und Zusammenstellung dieser Datensätze erläutert. Anschließend wird auf weitere relevante Arbeiten eingegangen, welche sich mit dem Thema der Stimmungsanalyse innerhalb der Softwareentwicklung beschäftigen.

Außerdem werden in dem kommenden Kapitel Arbeiten präsentiert, die den Zusammenhang der Produktivität von Entwicklern und deren Gefühlslage während der Programmierung untersuchen. Ein möglicher Anwendungsfall des in dieser Arbeit entworfenen Votingclassifiers ist zum Beispiel die Analyse der Gefühlslage innerhalb eines Entwicklerteams. Es wird die These aufgestellt, dass basierend auf dieser Gefühlslage ein frühzeitiges Eingreifen in die Situation möglich ist und somit eventuell die Produktivität des Teams gesteigert werden kann.

3.1 Goldstandard Datensätze

Novielli et al. [18] erhoben 2018 im Zuge der Entwicklung des Tools *Senti4SD* einen Datensatz bestehend aus 4800 Fragen, welche von Entwicklern auf der Seite Stackoverflow gestellt wurden. Diese 4800 Sätze wurden entsprechend der vorhandenen Emotionen mit den Labeln *Positiv*, *Negativ* und *Neutral* versehen, dies geschah durch eine Mehrheitsentscheidung verschiedener bewertender Personen [18].

Die erhobenen Kommunikationsdaten decken den Zeitraum von Juli 2008 bis September 2015 ab und wurden entsprechend der Emotionen *Love*, *Joy*, *Anger*, *Sadness*, *Fear* und *Surprise* mit Labeln versehen [18]. Die Abwesenheit von Emotionen innerhalb eines Satzes führte hierbei zur Vergabe des Labels *Neutral* [18].

Ebenfalls von Novielli et al. [17] wurde ein Goldstandard Datensatz auf der kollaborativen Entwicklungswebseite Github erstellt. Dieser Datensatz enthält über 7000 manuell nach Emotionen annotierte Sätze, welche ebenfalls mit den Label *Positiv*, *Negativ* und *Neutral* versehen wurden. Die Sätze stammen aus textuellen Beiträgen zu Pull Requests und Commits, also Änderungen von Entwicklern an öffentlichem Code, in unterschiedlichen Projekten [17].

Ortu et al. [21] erstellten einen Datensatz bestehend aus etwa 6000 Kommentaren, entnommen aus einem öffentlichen Jira Projekt. Enthalten waren dabei vier große, quelloffene Softwareprojekte: Apache[1], Spring[2], JBoss[3] und CodeHaus[4]. Aufgrund einiger Duplikate im Ursprungsdatensatz verwende ich für diese Arbeit 3974 dieser Kommentare, welche ebenfalls mit den drei oben genannten Labeln versehen wurden. Diese Auswahl wurde getroffen, da dieser Datensatz laut Aussage von Ortu et al. "ein feingranulareres Labeling" besitzt [21], als die übrigen circa 2000 verbleibenden Sätze.

Zhang et al. [26] untersuchten die Performanz von vortrainierten Transformermodellen im Bereich der Softwareentwicklung. Hierbei verwendeten sie unterschiedliche Datensätze zur Bewertung ihrer Modelle, darunter auch die sogenannten *Api* und *App* Datensätze [26].

Der Api Datensatz wurde von Uddin et al. [25] erstellt und besteht aus 4522 Sätzen, wobei 1338 Sätzen von der Seite Stack Overflow entnommen wurden [25]. Außerdem wird in dieser Arbeit der App Datensatz verwendet, dieser wurde von Lin et al. [12] erstellt, bestehend aus lediglich 341 Bewertungen von mobilen Applikationen [12].

[1] http://www.apache.org
[2] https://spring.io
[3] http://www.jboss.org
[4] http://www.codehaus.org

3.2 Stimmungsanalyse in der Softwareentwicklung

Calefato et al. [1] entwarfen das Tool *Senti4SD* und ermöglichten somit Training und Klassifizierung von Modellen, welche spezifisch auf die Domäne des *Software Engineerung* zugeschnitten sind. Hierzu wird ein spezielles Wortlexikon in Kombination mit Ansätzen des maschinellen Lernens verwendet, damit das Tool ein Eingabedokument als *Positiv, Negativ oder Neutral* einordnen kann [1]. Dieses Tool wurde entworfen, um Entwickler bei der Klassifizierung ihres Austausches in unterschiedlichen Kommunikationskanälen zu unterstützen [1].

Liu et al. [13] trainierten das RoBERTa Modell als verbesserten Ansatz des BERT Modells [5] und betrachteten dabei besonders den Einfluss der gewählten Hyperparameter auf die Performanz. Durch das Anpassen von Hyperparametern und der Verwendung von größeren Trainingsdatensätzen ist es ihnen gelungen, bessere Ergebnisse zu erzielen als ihre Vorgänger wie z.B. das BERT Modell [13].

Julian Horstmann [9] entwarf im Zuge seiner Masterarbeit ein Programm, welches textuelle Kommunikation innerhalb eines Entwicklerteams klassifiziert. Hierbei erhob er einen eigenen Datensatz basierend auf dem digitalen Schriftverkehr zwischen Entwicklern und hat die Sätze ebenfalls mit den drei Labeln versehen, die in dieser Arbeit auch verwendet werden. Im Vergleich zu der manuellen Vergabe von Labeln erreichte sein Tool allerdings eine 4% schlechtere Genauigkeit, nämlich 63%.

In seiner Bachelorarbeit nahm sich Christian Meyer [15] der Frage an, ob es möglich ist, das Tool von Horstmann mit diversen Erweiterungen bezüglich der Klassifizierungsgenauigkeit zu verbessern. Dazu analysierte er mögliche Fehler, die bei der Klassifizierung mit dem Tool aufgetreten sind und erstellte entsprechende Fehlerklassen. Anschließend implementierte er identifizierte Verbesserungen und erreichte damit eine Verbesserung der Genauigkeit im Vergleich zu dem Basisansatz von Horstmann um 5,1% [15].

Zhang et al. [26] verglichen die Performanz verschiedener vortrainierter Transformermodelle mit jenen Tools, die klassische Ansätze des maschinellen Lernens verwenden. Darunter befanden sich vier Transformermodelle (BERT, RoBERTa, XLNet und ALBERT) sowie fünf *klassische* Tools (Stanford CoreNLP, SentiStrength, SentiStrength-SE, SentiCR sowie Senti4SD) [26]. Dabei überprüfen sie die Klassifizierungsgenauigkeit aller Tools und Modelle auf den vorhergehend genannten Goldstandard Datensätzen.

Sie stellten dabei fest, dass sich die Performanz der Tools abhängig vom Testdatensatz ändert [26]. Außerdem wurde beobachtet, dass das *RoBERTa*

Modell durchschnittlich die höchsten Werte unter den vortrainierten Transformermodellen erzielt. Somit fiel die Entscheidung auf dieses Tool [26].

Novielli et al. [19] verglichen in einer Replikationsstudie die Performanz von *klassischen* Sentimentanalysetools mit der von neuronalen Netzen und Transformermodellen. Hierbei wurde die Zustimmung zwischen den einzelnen Tools im Bezug auf die Labelsetzung betrachtet. Ebenfalls wurde betrachtet, wie weit die Klassifizierung der Tools mit der manuellen Labelsetzung eines Goldstandard Datensatzes übereinstimmt [19].

Als Motivation für diese Arbeit dienten unter anderem die Erkenntnisse von Novielli et al. [16], welche Ergebnisse Sentimentanalysetools erzielen, wenn sie außerhalb der Trainingsdomäne angewendet werden. Dabei verwenden sie dieselben drei Datensätze wie in dieser Arbeit [16]. Sie trainieren dabei vier unterschiedliche Tools, klassifizieren anschließend Dokumente aus einer anderen Testdomäne und untersuchen, inwiefern sich die Ergebnisse von denen unterscheiden, die innerhalb der Trainingsdomäne erzielt werden.

Dabei ziehen sie den Schluss, dass die Performanz von Sentimentanalysetools merklich abnimmt, wenn die Test- sowie Trainingsdatensätze aus unterschiedlichen Domänen stammen [16].

Hierbei wurden aber weder Tools aus dem Bereich der neuronalen Netze verwendet, noch wurde der Ansatz eines Votingclassifiers evaluiert [16]. Deswegen habe ich mich in dieser Arbeit dazu entschieden, diese Möglichkeiten zu untersuchen.

Kapitel 4

Konzept

In diesem Kapitel wird der Trainingsablauf der verwendeten Tools erklärt und dargestellt, wie ihre Ergebnisse interpretiert werden. Basierend auf den gemessenen Werten der Einzeltools werden die Modelle für den Votingclassifier gewählt.

Ebenfalls wird die Häufigkeitsverteilung der Label innerhalb der verwendeten Datensätze und die Zusammenstellung eben dieser betrachtet. Anschließend wird die Funktionsweise des entworfenen Votingclassifiers erläutert und an Codebeispielen veranschaulicht.

4.1 Vortraining der Tools

4.1.1 Tool von Meyer und Horstmann

Das Tool von Christian Meyer [15] und Julian Horstmann [9] basiert auf klassischen Ansätzen des maschinellen Lernens, welche in Kombination mit domänenspezifischen Lexika verwendet werden. Es ist für jeden verwendeten Datensatz notwendig, ein solches Lexikon zu erstellen, welches den jeweiligen Dokumenten gewichtete Vektoren zuordnet [15]. Dieser Vorgang ist aufgrund der Anzahl an Dokumenten innerhalb der Datensätze, sowie der nicht parallelen Arbeitsweise der Funktionen, sehr zeitaufwändig [9]. Da dieses Verfahren ebenfalls bei der anschließenden Klassifizierung für den Testdatensatz wiederholt werden muss, dauert ein Gesamtdurchlauf im Vergleich zu den anderen in dieser Arbeit betrachteten Tools um ein vielfaches länger. Weitere Details zu diesem Verhalten sind in den Auswertungen in Kapitel 5 zu finden.

Nach dem Erstellen des Lexikons werden aus dem Eingabedatensatz fünf gleich große Splits erstellt, bestehend aus Trainingsdaten (80%) und

Testdaten (20%). Bei dieser Methode handelt es sich um eine stratifizierte, fünffache Kreuzvalidierung, Details zu dem Ablauf sind in Abschnitt 2.4 zu finden. Die Verhältnisse der Klassenverteilungen innerhalb eines Splits sind also möglichst identisch zu denen des Eingabedatensatzes [8].

Anschließend wird für alle fünf Splits jeweils das Vortraining und die Klassifizierung durchgeführt. Details zu den internen Abläufen hierbei sind der Arbeit von Meyer [15] und Horstmann [9] zu entnehmen. Aus jedem dieser fünf Durchläufe resultiert ein Modell für das Tool, welches ich jeweils zusammen mit den *classification-reports* für die spätere Verwendung und Auswertung abspeichere.

4.1.2 *RoBERTa*

Bei dem Training des RoBERTa Modells [13] wurde sich an dem verwendeten Code von Zhang et al. [26] orientiert. Der Originalcode ist in folgendem Github-Repository[1] zu finden. Es wurden Anpassungen vorgenommen, um eine fünffache Kreuzvalidierung zu ermöglichen und die Modelle ebenfalls für spätere Verwendung abzuspeichern. Auch bei den Hyperparametern und der Anzahl an Epochen habe ich mich an die verwendeten Werte von Zhang et al. gehalten, da diese gute Ergebnisse erzielt haben [26]. Hierbei ist allerdings anzumerken, dass Zhang et al. in ihrer Arbeit keine Kreuzvalidierung verwendet haben [26].

Das Training eines neuronalen Netzes dieses Ausmaßes ist sehr rechenintensiv [13]. Zur Reduzierung der zeitlichen Belastung wurde das Training auf die Onlineplattform *Google Colab*[2] ausgelagert. Dort ist es möglich, unter Verwendung einer Grafikkarte in der Cloud, Trainingszeiten von Modellen im Vergleich zu einem herkömmlichen Heimcomputer um ein Vielfaches zu verringern.

Training und Auswertung wurden hier ebenfalls in fünffacher Kreuzvalidierung auf den identischen Splits durchgeführt, welche auch für die anderen Tools verwendet wurden. Modelle und Statistiken wurden in meiner Google Drive gespeichert und anschließend auf das Testsystem heruntergeladen.

4.1.3 Senti4SD

Auch für dieses Tool wurden die selben Splits verwendet, somit erhalten wir hier ebenfalls fünf Modelle inklusive Statistiken pro Datensatz. Das Tool Senti4SD basiert, wie das Tool von Horstmann [9] und Meyer [15], auf Ansätzen

[1] https://github.com/soarsmu/SA4SE
[2] https://colab.research.google.com/

des maschinellen Lernens in Kombination mit einem domänenspezifischen Lexikon [1]. Details zu den inneren Abläufen des Tools können der Arbeit von Calefato et al. [1] entnommen werden.

In meiner Arbeit verwendete ich die Python-Variante[3], da somit eine simplere Implementation des Tools in das Framework des Votingclassifiers möglich war.

4.2 Auswahl der Datensätze

Die Entscheidung, welche Datensätze für das Training der Tools dieser Arbeit verwendet werden sollen, wurde unter Berücksichtigung bestimmter Kriterien getroffen. Um der Gefahr der ungenauen oder falschen Labelvergabe entgegen zu wirken, wurde darauf geachtet, dass es sich bei allen drei Datensätzen um solche handelt, die von unterschiedlichen Personen per Hand mit Hilfe eines Emotionsmodells gelabelt wurden.

Für die Erstellung eines solchen Datensatzes gelten bestimmte Regeln. Diese Regeln wurden von Novielli et al. [18] für das Labeling der drei zum Training benutzten Datensätze angewandt. Sie sind auf der Githubseite des Forschungsteams einsehbar[4].

Bei der Erstellung dieser Datensätze werden die einzelnen Dokumente von unterschiedlichen Personen auf das Vorhandensein von Emotionen überprüft und die Entscheidungen begründet festgehalten [18]. Hierbei wird als Label die sogenannte Basisemotion verwendet, welche mit dem Vorhandensein der untergeordneten Emotionen des zweiten und dritten Levels begründet wird [18].

Diese Emotionslevel wurden im Jahre 1987 von Shaver et al. [24] erarbeitet. Ein Beispiel für die Emotion *Sadness* ist in der folgenden Tabelle zu sehen. Die gesamte Tabelle ist ebenfalls auf der Githubseite von Novielli zu finden.

Basic Emotion	2nd Level	3rd Level
Sadness	Shame	Guilt, Regret, Remorse

Tabelle 4.1: Levelbeispiel der Basisemotion *Sadness*.

[3]https://github.com/collab-uniba/pySenti4SD
[4]https://github.com/collab-uniba/EmotionDatasetMSR18

Nachdem jedes Dokument innerhalb eines Datensatzes basierend auf den gefundenen Emotionen gelabelt wurde, ist es möglich, dieses Labeling in die drei Polaritätsklassen *Positiv*, *Neutral* und *Negativ* zu überführen. Dazu werden Dokumente, welche die Emotionen *Love* und *Joy* enthalten, der Klasse *Positiv* zugeordnet. Die Emotionen *Anger, Sadness* und *Fear* führen zu der Vergabe des Labels *Negativ*.

Lediglich Sätze der Emotion *Surprise*, sowie solche mit Abwesenheit jeglicher Emotionen, werden der Klasse *Neutral* zugeordnet.

Für die Datensätze aus der Domäne Github und Stackoverflow wurde dieses Mapping bereits von Novielli et al. durchgeführt [17] [18]. Für den Jira Datensatz [21] habe ich das Emotionsmapping selber mit Hilfe eines Skriptes durchgeführt.

Zusätzlich zu den drei eingehend benannten Datensätzen werden noch zwei weitere Datensätze verwendet. Diese sind besonders für die Betrachtung der **RQ2** und **RQ3** interessant, da somit eine Betrachtung des Klassifizierungsverhaltens des Votingclassifiers gänzlich außerhalb der Trainingsdomäne möglich ist.

Datensatz	# Docs	# Positiv (%)	# Neutral (%)	# Negativ (%)
Github	7122	2013 (28)	3022 (43)	2087 (29)
Stackoverflow	4423	1527 (35)	1694 (38)	1202 (27)
Jira	3974	626 (15)	3058 (76)	290 (8)
Api	4522	890 (20)	3136 (69)	496 (11)
App	341	186 (55)	25 (7)	130 (38)

Tabelle 4.2: Häufigkeitsverteilung der einzelnen Label innerhalb eines Datensatzes.

Besonders zu beachten sind hierbei die ungleichmäßigen Verteilungen der Labels innerhalb des Jira und App Datensatzes. Ebenfalls enthält der App Datensatz zusätzlich noch eine wesentlich geringere Menge an Dokumenten als die restlichen Datensätze.

Allerdings unterscheidet sich die Domäne des App Datensatzes von der klassischen Softwareentwicklung, da hier Applikationen von Menschen bewertet wurden, die wahrscheinlich selber keine Softwareentwickler waren [26].

4.3 Auswertung der Einzelergebnisse

Es wurden jeweils pro Tool, Datensatz und Split ein Modell, sowie der zugehörige *classification-report* erstellt. Dieser Report enthält wichtige Metriken wie die Accuracy des jeweiligen Modells. Ein Beispiel dazu ist in der folgenden Tabelle dargestellt:

	Precision	Recall	F1	Support
Negative	0.91	0.89	0.90	418
Neutral	0.93	0.91	0.92	605
Positive	0.90	0.96	0.93	403
Accuracy			0.92	1426
Macro Avg.	0.92	0.92	0.92	1426
Weighted Avg.	0.92	0.92	0.92	1426

Tabelle 4.3: Beispielhafter Klassifizerungsreport des RoBERTa Modells auf einem Split des Github Datensatzes.

Bei der Bewertung eines Modells wird als erster relevanter Messwert die Accuracy betrachtet, da dieser die gesamte Klassifizierungsgenauigkeit eines Modells repräsentiert [14]. Falls es hier innerhalb eines Splits und Datensatzes zu Gleichstand kam, bin auf den F1-Wert der jeweiligen Klassen ausgewichen, da dieser Wert sowohl die Precision als auch den Recall aller Klassen betrachtet. Details zu den Berechnungen der Messwerte sind in dem Abschnitt 2.5 zu finden.

Hierbei wurde darauf geachtet, dass die Differenz zwischen den F1-Werten der Klassen *Positiv, Negativ und Neutral* möglichst gering bleibt. So soll ein konsistentes Klassifizierungsverhalten des Modells angenähert werden.

Die so ermittelten stärksten Modelle innerhalb eines Splits wurden auf dem Testsystem abgelegt, da später innerhalb des Votingclassifiers pro Datensatz nur das Beste der fünf Modelle verwendet wird.

4.4 Aufbau des Votingclassifiers

Die Funktionsweise eines Votingclassifiers wird in den Grundlagen in Abschnitt 4.4 erläutert. Um eigenständige Tools zu kombinieren war der Entwurf einer neuen Codestruktur nötig, da ein solches Tool am Markt noch nicht existiert.

Mein Votingclassifier ist in der Programmiersprache Python geschrieben und vereint drei Klassifizierungstools.

Hierbei ist es nötig, jedem Tool in einer Konfigurationsdatei ein unterschiedliches Modell zuzuweisen und die gewählte Konfiguration auf einem Datensatz testen. Eine beispielhafte Konfigurationsdatei sieht wie folgt aus:

```
{
  "modelConfigs": {
    "content": [
      {
        "roberta": "bestJira/pretrained_jira_5",
        "christian": "bestJira/jira_gold_model_2.sav",
        "senti": "bestJira/jira1.model"
      }
    ]
  },
  "testFiles": {
    "content": ["api_dataset.csv", "app_dataset.csv"]
  }
}
```

Listing 1: Beispielhafte Konfigurationsdatei des Votingclassifiers.

Hierbei ist es besonders nützlich, sowohl die Modellzusammenstellungen als auch die Testdatensätze als Liste anzugeben, um einen iterativen Ablauf aller benötigten Testszenarien zu ermöglichen.

Als *Durchlauf* bezeichne ich die Klassifizierung und Auswertung einer bestimmten Zusammenstellung des Votingclassifiers auf genau einer Testdatei. Hierbei wird für jeden Durchlauf initial eine Textdatei und ein Ordner erstellt, welche Informationen über die Zusammenstellung der Modelle sowie den gewählten Testdatensatz enthält.

In diesem Ordner werden im weiteren Verlauf ebenfalls die Ergebnisse der Klassifizierungen der Einzeltools und des Votingclassifiers in Tabellenform

abgelegt. Ebenfalls werden die *classification_reports* als Textdatei abgespeichert, um ein nachträgliches Vergleichen der Ergebnisse zu ermöglichen.

Das weitere Vorgehen innerhalb eines Durchlaufes lässt sich gut mit folgendem Python Code aus dem Programmablauf Votingclassifiers veranschaulichen:

```
# run classifications to predict labels
doRobertaClassification()
doSentiClassification()
doChristianClassification()

# merge results and do majority vote
merge = readResults()
voteFromMergedData(merge)

# move result files to timestamped folder
curTime = datetime.datetime.now().strftime[...]]
os.system(f"mkdir resultDump/{curTime}")
os.system(f"mv predictions/* resultDump/{curTime}")
```

Hierbei werden erst die Klassifizierungen pro Tool durchgeführt und als Ergebnis entsprechende Dateien erstellt, welche jeweils die Dokumente, eigentliche Label und die vom Tool vorhergesagten Label enthalten. Anschließend werden diese Dateien zeilenweise basierend auf dem Index der Dokumente zusammengeführt.

Mit Hilfe dieser Gesamtdatei, welche alle vorhergesagten Label der Eingabetools enthält, trifft der Votingclassifier nun eine Gemeinschaftsentscheidung oder notiert ein *tie*, also Gleichstand, falls keine eindeutige Mehrheit gefunden werden kann. Dies geschieht unter Verwendung der Python Bibliothek *NumPy*.

```
# do MajorityVote and note down ties
mode = r.mode(axis=1)
r["maj"] = np.where(mode.isna().any(1), mode[0], "tie")
```

Bei jedem Dokument, für welches das Label *tie* gesetzt wurde, wird anschließend randomisiert eines der drei möglichen Label gesetzt. Für zukünftige Betrachtungen sind an dieser Stelle alternative Ansätze denkbar. Auf potentielle Möglichkeiten wird in Kapitel 6 eingegangen.

Anschließend an das Mehrheitsvotum wird die Klassifizierungsgenauigkeit des Ensembles berechnet und als *classification-report* innerhalb des Verzeichnisses gespeichert. Dieser Vorgang wird entsprechend für alle in der Konfigurationsdatei gewünschten Konstellationen wiederholt und pro

Durchlauf werden die Ergebnisse in einem neuen Ordner abgelegt.

Sobald alle Durchläufe beendet wurden, ist eine Auswertung der Gesamtstatistiken über ein Hilfsskript möglich.

4.5 Ergebnisse eines Durchlaufs

Die von dem Votingclassifier erstellten Auswertungsdateien wurden nach den gewünschten Durchläufen aggregiert und in einer Exceldatei dargestellt. Dabei wurden jeweils die verwendeten Modelle pro Tool, der betrachtete Testdatensatz sowie Einzelergebnisse der Tools erfasst.

Ebenfalls wurde die Accuracy des Votingclassifiers aufgenommen und die einzelnen Tools wurden bezüglich des Cohen-Kappa Wertes verglichen, um zu schauen, inwiefern sie sich bei der Labelvergabe einig waren. Hierzu wurde die Funktion *cohen_kappa_score* der scikit-learn Bibliothek in Python verwendet[5].

Insgesamt wurden in dieser Arbeit 21 Durchläufe des Votingclassifiers durchgeführt und ausgewertet. Darunter fallen 3 Durchläufe im Bezug auf **RQ1**, wobei alle Tools innerhalb der Testdomäne trainiert wurden. Außerdem fallen jeweils 9 Durchläufe pro Datensatz bei der Betrachtung der **RQ2** beziehungsweise **RQ2** an, da hier unterschiedliche Domänen innerhalb einer Konfiguration verwendet wurden.

[5]https://scikit-learn.org/stable/modules/generated/sklearn.metrics.cohen_kappa_score.html

Näheres zu den einzelnen Zusammenstellungen ist der unteren Tabelle zu entnehmen. Hier werden die verwendeten Konfigurationen mit einer ID versehen, welche im folgenden Kapitel 5 der Auswertungen referenziert werden.

ID	Senti4SD	Roberta	Meyer
1	Github	Stackoverflow	Jira
2	Github	Jira	Stackoverflow
3	Stackoverflow	Github	Jira
4	Stackoverflow	Jira	Github
5	Jira	Github	Stackoverflow
6	Jira	Stackoverflow	Github
7	Jira	Jira	Jira
8	Stackoverflow	Stackoverflow	Stackoverflow
9	Github	Github	Github

Tabelle 4.4: Verwendete Zusammenstellung des Votingclassifiers im Bezug auf RQ 2 und RQ 3.

Die Entscheidung des Vorgehens fiel auf einen quantitativen Ansatz, um eine möglichst große Abdeckung an Kombinationen zu gewährleisten. Eine Analyse zu den jeweiligen Ergebnissen der Zusammenstellungen folgt in Kapitel 5.

Kapitel 5

Auswertungen

Im folgenden Kapitel werden die Ergebnisse der insgesamt 21 Durchläufe des Votingclassifiers dargestellt und in Relation zu den Einzelergebnissen der verwendeten Tools gestellt. Dabei gehe ich ebenfalls auf die eingehend definierten Forschungsfragen ein.

Innerhalb eines Durchlaufes werden lediglich die Accuracy Werte der Einzeltools, sowie des Ensembles, betrachtet. Für bestimmte Konstellationen stelle ich zusätzlich tiefer greifende Messwerte dar. Dies soll einen eventuellen Einblick in das Verhalten der Klassifizierungstools ermöglichen.

Abschließend werden *Threats to Validity* erläutert, welche unter Umständen die Validität der Ergebnisse dieser Arbeit gefährden.

5.1 Domäneninterne Klassifizierung

Die eingehend gestellte **RQ1** lautet: "Erzielt ein Ensemble aus Klassifizierungstools höhere Klassifizierungsgenauigkeiten als die jeweiligen Einzeltools innerhalb der Domäne eines Datensatzes?".

Dazu betrachte ich jeweils ein Ensemble aus Tools, wobei alle Einzeltools innerhalb derselben Domäne trainiert werden. Anschließend erfolgt eine Klassifizierung und Auswertung auf einem Testdatensatz derselben Domäne.

Für die Datensätze Jira und Github erreicht der Votingclassifier eine um einen Prozentpunkt höhere Accuracy als das jeweils beste Einzeltool. Auf dem Stackoverflow Datensatz ist die Accuracy mit 90% zwischen dem Votingclassifier und dem *RoBERTa* Modell identisch. Für diesen Datensatz werden deswegen im Folgenden noch weitere Messwerte betrachtet.

Die Klassifizierungsgenauigkeiten auf allen drei Goldstandard Datensätzen innerhalb des Votingclassifiers sind in der folgenden Tabelle 5.1 dargestellt:

Domäne	Votingclassifier	Senti4SD	RoBERTa	Meyer
Jira	**0.89**	0.85	0.87	0.88
Stackoverflow	**0.90**	0.87	**0.90**	0.86
Github	**0.93**	0.92	0.92	0.88

Tabelle 5.1: Klassifizierung innerhalb derselben Domäne unter Verwendung des Votingclassifiers.

Da der Votingclassifier und eines der verwendeten Tools unter Betrachtung des Stackoverflow Datensatzes identische Accuracy Werte erreichen, ist es nun interessant, diesen Fall näher zu untersuchen. In der folgenden Tabelle sind die klassenspezifischen F1-Werte der Konstellation des Votingclassifiers auf dem Stackoverflow Datensatz zu sehen.

Tool	Positiv	Neutral	Negativ	Macro Avg.
Votingclassifier	0.94	**0.88**	**0.88**	**0.90**
Senti4SD	0.92	0.84	0.84	0.87
RoBERTa	**0.95**	**0.88**	0.85	**0.90**
Meyer	0.92	0.84	0.83	0.86

Tabelle 5.2: Klassenspezifische F1-Werte innerhalb des Stackoverflow Durchlaufes. Die spaltenweise besten Ergebnisse wurden fett markiert.

Beim Vergleich der Werte des Votingclassifiers und des *RoBERTa* Modells ist nun zu erkennen, dass der Macro Avg. und die Accuracy identisch sind. Allerdings beläuft sich die Spanne bei *RoBERTa* mit Werten von 85% - 95% auf 10% Differenz, bei dem Votingclassifier hingegen sind es nur 6% Differenz (88%-94%). Somit ist die Differenz zwischen dem niedrigsten und höchsten Wert der jeweils schlechtesten und besten Klasse des Votingclassifiers um 4% geringer.

Dieses Verhalten kann ein Indikator dafür sein, dass ein Votingclassifier unter Umständen ein ausgewogeneres Klassifizierungsverhalten produziert als einzelne Tools.

Zwar nimmt man in dem Falle des Stackoverflow Beispiels eine Abnahme des F1-Wertes innerhalb der *Positiv*-Klasse um 1% in Kauf, verringert dafür aber die klasseninternen Differenzen.

Betrachten wir nun das Cohen Kappa (Erklärung siehe Kapitel 2.5.1 für die jeweiligen Toolkombinationen, um uns ein Bild davon zu machen, inwiefern sich die einzelnen Tools innerhalb eines Datensatzes einig waren bei der Klassifizierungsaufgabe. Hierbei wird ebenfalls gewichtet, wie groß die Übereinstimmung sein würde, wenn alle Label zufällig vergeben werden würden.

Näheres zu der Berechnung dieses Wertes ist in dem Kapitel der Grundlagen unter Abschnitt 2.5 zu finden.

Es folgt die Tabelle der Messwerte zwischen den jeweiligen Toolpaaren:

Domäne	RoBERTa - Meyer	RoBERTa - Senti4SD	Meyer - Senti4SD
Jira	0.72	0.62	0.69
Stackoverflow	0.78	0.78	0.78
Github	0.82	0.85	0.81

Tabelle 5.3: Cohen Kappa der jeweiligen Toolkombinationen innerhalb eines Datensatzes.

Da diese Werte durchgehend näher an einer 1 sind als an 0 kann man davon ausgehen, dass sich die Tools untereinander tatsächlich größtenteils einig waren und nicht nur aus Zufall die gleichen Label vergeben haben. In der Dokumentation der verwendeten Bibliothek werden Werte über 0.8 als gute Übereinstimmung genannt, bei Werten gegen und unter 0 spricht man in der Regel von keiner bzw. zufälliger Übereinstimmung zwischen Klassifizierern[1].

In Betrachtung der eingehend gestellten **RQ1** treffe ich nun die Aussage, dass es möglich ist, mit einem Votingclassifier höhere Klassifizierungsgenauigkeiten zu erzielen, als unter Verwendung der jeweiligen Einzeltools. Für zwei der drei betrachteten Datensätze erreicht der Votingclassifier eine um 1% höhere Accuracy, für den anderen Datensatz bleibt die Accuracy identisch.

Fazit Die Verwendung eines Votingclassifiers kann bei domäneninterner Klassifizierung zu einer Erhöhung der Accuracy führen. Ebenfalls gibt es Anzeichen dafür, dass ein Votingclassifier für eine geringere Differenz zwischen den klassenspezifischen F1-Werten sorgen kann.

[1] https://scikit-learn.org/stable/modules/model_evaluation.html#cohen-kappa

5.2 Fremde Testdomänen

RQ2 wurde eingehend wie folgt definiert: "Wie verhält sich die Klassifizierungsgenauigkeit des Votingclassifiers, wenn sich die Trainingsdomänen der Einzeltools gänzlich von der Testdomäne unterscheiden?"

Innerhalb dieses Abschnittes werden jeweils neun Durchläufe des Votingclassifiers auf zwei unterschiedlichen Datensätzen untersucht, dessen Domänen den verwendeten Tools gänzlich unbekannt sind. Zu Beginn präsentiere ich die Ergebnisse auf dem Api Datensatz von Zhang et al. [26]:

ID	Votingclassifier	Senti4SD	RoBERTa	Meyer
1	**0.72**	**0.72**	**0.72**	0.70
2	0.73	0.72	0.70	**0.74**
3	**0.72**	**0.72**	0.70	0.70
4	**0.73**	0.72	0.70	**0.73**
5	0.73	0.70	0.70	**0.74**
6	0.72	0.70	0.72	**0.73**
7	**0.71**	0.70	0.70	0.70
8	0.73	0.72	0.72	**0.74**
9	**0.73**	0.72	0.70	**0.73**

Tabelle 5.4: Klassifizierungsgenauigkeiten der unterschiedlichen Ensembles auf dem Api Datensatz.

Die jeweils durch die ID 1-9 repräsentierten Konfigurationen des Votingclassifiers sind in Kapitel 4.5 erläutert. Es ist zu erkennen, dass die Klassifizierungsgenauigkeiten der Einzeltools, sowie des Votingclassifiers, sich in jeder Konstellation nur um wenige Prozentpunkte unterscheiden. Innerhalb keines Durchlaufes ist die Differenz zwischen dem besten und schlechtesten Tool größer als 4%.

Es existiert lediglich eine Konstellation des Votingclassifiers, bei dem die Accuracy des Ensembles strikt größer ist, als die jedes Einzeltools. Dies ist

Durchlauf 7, bei dem als verwendete Modelle alle Trainingsdomänen aus dem Jira Datensatz stammen. Hierbei ist ein Zuwachs der Klassifizierungsgenauigkeit um einen Prozentpunkt zu beobachten.

Bei allen anderen Konstellationen ist der Votingclassifier entweder schlechter oder maximal genau so gut, wie das beste der verwendeten Einzeltools innerhalb des Ensembles. Vergleicht man die Einzelergebnisse des Votingclassifiers nun innerhalb aller neun Durchläufe, stellt man fest, dass er niemals einen Wert über 73% erreicht, das Tool von Meyer allerdings mehrmals eine Klassifizierungsgenauigkeit von 74% trifft.

Ähnlich wie im vorherigen Abschnitt, untersuche ich nun wieder die klassenspezifischen F1-Werte eines Durchlaufes. Hierbei wähle ich den Durchlauf mit der ID 7, da dort der Votingclassifier eine höhere Accuracy erzielte als alle Einzeltools. Allerdings ist bei diesem Durchlauf anzumerken, dass alle Einzeltools auf dem Jira Datensatz trainiert wurden. Hierbei ist zu erkennen, dass die Tools bei Vortraining auf diesem Datensatz schlechtere Accuracy Werte erzielen, als auf den anderen beiden. Die Abweichungen liegen zwischen 0% - 4%.

Tool	Positiv	Neutral	Negativ	Macro Avg.
Votingclassifier	0.05	0.82	0.19	0.36
Senti4SD	0.06	0.82	0.27	0.39
RoBERTa	0.08	0.82	0.23	0.38
Meyer	0.04	0.82	0.14	0.33

Tabelle 5.5: Klassenspezifische F1-Werte in Durchlauf ID 7 auf dem Api Datensatz.

In diesem Durchlauf stellen wir kein ausgewogeneres Klassifizierungsverhalten wie in Tabelle 5.2 fest. Auch bei weiterer Betrachtung der F1-Werte anderer Durchläufe ist keine großartige Abweichung innerhalb der Klassen zwischen Votingclassifier und Einzeltools zu beobachten.

In der folgenden Tabelle sind die Accuracy Werte der Konstellationen auf dem App Datensatz zu sehen, welcher ebenfalls von Zhang et al. [26] erstellt wurde.

ID	Votingclassifier	Senti4SD	RoBERTa	Meyer
1	0.55	0.57	**0.70**	0.25
2	0.61	0.57	0.50	**0.62**
3	0.60	0.59	**0.77**	0.25
4	**0.62**	0.59	0.50	0.61
5	0.52	0.35	**0.77**	0.62
6	0.61	0.35	**0.70**	0.61
7	0.34	0.35	**0.50**	0.25
8	0.65	0.59	**0.70**	0.62
9	0.64	0.57	**0.77**	0.61

Tabelle 5.6: Klassifizierungsgenauigkeiten der unterschiedlichen Ensembles auf dem App-Review Datensatz.

Bei diesem Datensatz ist anzumerken, dass die Domäne sehr von der der Softwareentwicklung abweicht, da es sich um Nutzerbewertungen von Applikationen handelt. Diese werden in der Regel von Menschen geschrieben, die nicht zwangsläufig Softwareentwickler sind [26]. Ebenfalls umfasst der Datensatz mit 341 Dokumenten deutlich weniger Einträge, als die zum Training verwendeten Goldstandard Datensätze.

Die Durchläufe 7-9 verwenden hierbei jeweils denselben Datensatz für alle drei Tools. Dabei ist zu erkennen, dass der Jira Datensatz unter ID 7 wesentlich schlechtere Accuracy Werte erzielt, als die anderen beiden Datensätze. Die Abweichungen zwischen dem besten und dem schlechtesten Ergebnis liegen bei *RoBERTa* beispielsweise bei 20% - 27%.

Ebenfalls ist zu erkennen, dass das RoBERTa Modell fast durchgehend die höchsten Accuracy Werte erzielt. Lediglich in einer Konstellation, Nummer

4, ist das Ensemble als Ganzes genauer als die drei verwendeten Einzeltools.

Vergleicht man dieses Gesamtergebnis in Höhe von 62% nun aber wieder mit dem besten Einzelergebnis aller Durchläufe, stellt man fest, dass auch hier der Votingclassifier kein besseres Ergebnis erzielt. Die höchste Klassifizierungsgenauigkeit des Ensembles liegt bei 65%, das RoBERTa Modell erzielt hingegen im besten Durchlauf eine Genauigkeit von 77%. Dieses Ergebnis erreichte das Modell unter Verwendung des Github Datensatzes im Training.

Fazit

Die Verwendung eines Votingclassifiers führt bei der Klassifizierung in fremder Domäne nicht zu einem Anstieg der Accuracy. Falls ein Modell vorliegt, welches in der selben Domäne trainiert wurde, in der auch klassifiziert werden soll, ist dieses wahrscheinlich einem Votingclassifier vorzuziehen.

Ich sehe keine Anzeichen dafür, dass ein Votingclassifier zu einem ausgewogeneren Klassifizierungsergebnis im Bezug auf die einzelnen Klassen führt.

5.3 Vergleich Einzeltools und Votingclassifier

Zur Beantwortung der **RQ3** vergleiche ich nun die Ergebnisse aus **RQ2** mit den Einzeltools, allerdings bei Training dieser innerhalb der Testdomäne. Dies bedeutet beispielsweise das Training, sowie eine anschließende Klassifizierung des *RoBERTa*-Modells auf dem Api-Datensatz.

In dem Abschnitt 5.2 habe ich bereits dargestellt, dass es keine Konstellation des Votingclassifiers gibt, welche die höchste Klassifizierungsgenauigkeit des besten Einzeltools übertrifft. Für den Api Datensatz ist dies das Tool von Meyer mit 74%, wobei der Votingclassifier maximal 73% erreichte (siehe Tabelle 5.4).

Da somit keine domänenfremde Konstellation gefunden wurde, welche die Accuracy eindeutig verbesserte, ist es nun auch zu erwarten, dass der Votingclassifier den domäneninternen Einzeltools ebenfalls unterlegen sein wird.

Diese Auswertungen habe ich nicht selber durchgeführt, sie liegen bereits als Ergebnisse der von Zhang et al. [26] durchgeführten Untersuchungen vor.

Allerdings haben Zhang et al. [26] lediglich das *RoBERTa*-Modell erneut auf den spezifischen Datensatz trainiert, bei der Verwendung von Senti4SD wurde das mitgelieferte Stackoverflow Modell von Calefato et al. [1] verwendet [26]. Da die Forschungsgruppe das Tool von Meyer nicht verwendete, ist somit nur ein direkter Vergleich mit den Ergebnissen des *RoBERTa*-Modells möglich.

In der folgenden Tabelle sind die erreichten Werte dieses Modells von Zhang et al. [26] bei Training auf dem Api Datensatz, sowie auf dem App Datensatz dargestellt:

Tool	Positiv	Neutral	Negativ	Macro Avg.
RoBERTa	**0.78**	**0.93**	**0.71**	**0.81**
Votingclassifier #4	0.32	0.83	0.30	0.48

Tabelle 5.7: Vergleich der klassenspezifischen F1-Werte von Zhang et al. [26] mit dem Votingclassifier #4 auf dem Api Datensatz.

Hierbei entschied ich mich für die Konstellation Nummer 4 des Votingclassifiers, da diese im Vergleich zu den anderen Zusammenstellungen die höchsten Werte erzielte.

Es ist zu erkennen, dass die Differenzen zwischen den Ergebnissen der positiven und negativen Klasse wesentlich größer sind, als die der neutralen Klasse. Somit kann auch hier der Votingclassifier das Ergebnis des Einzeltools nicht übertreffen.

Betrachtet wird nun der App Datensatz, klassifiziert mit dem *RoBERTa*-Modell, ebenfalls durchgeführt von Zhang et al. [26]. Für diesen Datensatz war Konstellation 9 die performanteste, weswegen ich im Folgenden diesen Durchlauf für den Vergleich heran ziehe:

Tool	Positiv	Neutral	Negativ	Macro Avg.
RoBERTa	**0.93**	0.00	**0.91**	**0.61**
Votingclassifier #9	0.81	**0.17**	0.52	0.50

Tabelle 5.8: Vergleich der klassenspezifischen F1-Werte von Zhang et al. mit dem Votingclassifier #9 auf dem App Datensatz.

In diesem Datensatz scheint es sich anders als in dem Api Datensatz zu verhalten. Besonders interessant sind die äußerst niedrigen Werte innerhalb der neutralen Klasse. Bei Vortraining innerhalb der Domäne klassifizierte das *RoBERTa*-Modell keines der Dokumente dieser Klasse korrekt [26]. Der Unterschied des Macro Avg. ist hier nicht so gravierend wie innerhalb des Api Datensatzes, der Votingclassifier unterliegt dem Tool um lediglich 10%.

Zhang et al. nennen für diese beobachtete, vergleichsweise schlechte Performanz als möglichen Grund das Ungleichgewicht der Labelverteilung [26].

Rückblickend auf **RQ3** treffe ich die Aussage, dass die Verwendung eines Votingclassifiers die Klassifizierungsgenauigkeiten im Vergleich zu den verwendeten Einzeltools verringert. Hierbei ist allerdings anzumerken, dass sich die Domänen des Trainings- und Testdatensatzes zwischen den Modellen unterscheiden. Falls also für eine bestimmte Aufgabe ein Modell existiert, welches in derselben Domäne trainiert wurde wie die zu klassifizierenden Daten, ist es ratsam, dieses einem Votingclassifier vorzuziehen.

Besonders innerhalb des App Datensatzes scheint es sich so zu verhalten, dass bei Existenz eines vergleichsweise schlechten Ergebnisses innerhalb des Ensembles das Gesamtergebnis des Votingclassifiers darunter leidet. Weitere Forschung zur Untersuchung dieses Verhaltens ist denkbar.

Fazit

Auch in einem Vergleich mit Einzeltools, die in der Testdomäne trainiert wurden, erreicht der Votingclassifier keine höhere Accuracy. Allerdings gibt es auch hier Anzeichen dafür, dass ein Votingclassifier auf manchen Datensätzen für eine ausgewogenere Klassifizierung sorgen kann (Vergleich Tabelle 5.8).

5.4 Threats to Validity

Innerhalb eines Ablaufes des Votingclassifiers werden Gleichstände mit einer Zufallsbelegung des jeweiligen Labels gelöst. Die kann unter Umständen bei selber Ausführung und identischen Eingabemodellen zu unterschiedlichen Ergebnissen führen. Bei den von mir betrachteten Durchläufen lag der prozentuale Anteil dieser Zufallsentscheidungen zwischen 0.1% bis 0.8%. Hier wäre ein alternativer Ansatz möglich, um die Zufallskomponente komplett zu eliminieren. Bis dahin stellt dieses Verhalten einen *Threat to Conclusion Validity* dar.

Unter Umständen ist es möglich, dass die Ergebnisse dieser Arbeit spezifisch auf die verwendeten Datensätze sind. Durch die Verwendung der drei Datensätze von Novielli et al. [17] [18] [21] versuche ich, einen *Threat to Conclusion Validity* zu minimieren. Allerdings erfolgt die Betrachtung der **RQ2** und **RQ3** auf Datensätzen von Zhang et al. [26], welche nicht basierend auf einem Emotionsmodell erstellt wurden und auch nicht von mehreren, unabhängigen Personen gelabelt wurden.

Somit besteht im Bezug auf diese beiden Datensätze ein *Threat to External Validity*, da die Vergabe der Label eventuell nicht repräsentativ ist.

Ebenfalls ist es möglich, dass bei der Programmierung des Votingclassifiers oder bei der Auswertung der jeweiligen Messwerte Fehler unterlaufen sind. Dies stellt einen *Threat to Internal Validity* dar. Zur Replikation der Durchläufe und Tests ist auf Anfrage eine Herausgabe des Codes möglich.

Einen *Threat to Construct Validity* stellt meine Vorgehensweise der Auswertung basierend auf der Accuracy eines Modells dar. Hier wäre eine weitere Analyse basierend auf den F1-Werten der einzelnen Klassen denkbar. Ebenfalls könnte man auch Precision und Recall gegenüber stellen. Dies habe ich für einige Durchläufe getan, aber eben nur bei diesen, die ich für besonders betrachtenswert innerhalb eines Datensatzes erachtet habe.

Durch Verwendung der Tools Senti4SD [1], *RoBERTa* [13] und der Klassifizierung von Meyer [15] übernehme ich innerhalb meiner Arbeit ebenfalls die jeweils relevanten *Threats to Validity* dieser Tools.

Kapitel 6

Zusammenfassung und Ausblick

Dieses Kapitel gibt einen abschließenden Überblick über die Ergebnisse dieser Arbeit und die gestellten Forschungsfragen. Dabei werden die wichtigsten Werte präsentiert und Erkenntnisse bei der Verwendung eines Votingclassifiers dargestellt.

Ebenfalls wird ein Ausblick gestellt, welche zukünftige Arbeiten innerhalb des Feldes der Stimmungsanalyse in der Softwareentwicklung denkbar wären. Zusätzlich dazu werden mögliche Verbesserungen des Votingclassifiers in Aussicht gestellt.

6.1 Zusammenfassung

Diese Arbeit befasst sich mit der Stimmungsanalyse von natürlichsprachigen Texten innerhalb der Domäne der Softwareentwicklung. Zur Klassifizierung solcher Texte wurde ein Votingclassifier entworfen, welcher die Accuracy in von den Trainingsdomänen abweichenden Testdomänen verbessern soll. Dieser Votingclassifier besteht aus drei Einzeltools, darunter Senti4SD von Calefato et al., dem *RoBERTa*-Modell sowie einem Ansatz von Julian Horstmann und Christian Meyer.

Die Betrachtung dieses Bereiches der Stimmungsanalyse ist interessant, da Novielli et al. in ihrer Arbeit erkannt haben, dass die Klassifizierungsgenauigkeit von aktuellen Sentimentanalysetools merklich abnimmt, wenn sich die Domäne des Testdatensatzes von der des Trainingsdatensatz unterscheidet.

Die erste Forschungsfrage **RQ1** beschäftigt sich mit der domäneninternen Klassifizierung und ob ein Votingclassifier hier die Accuracy verbessern kann. Die Auswertungen haben gezeigt, dass ein Votingclassifier die Accuracy in zwei Fällen um 1% verbessern konnte. Ebenfalls gibt es Anzeichen, dass ein

Votingclassifier die Ausgewogenheit der klasseninternen Ergebnisse fördert.

Bei der Beantwortung der zweiten Forschungsfrage **RQ2** untersuchte ich das Verhalten des Votingclassifiers bei unterschiedlicher Trainings- und Testdomäne. Hierbei war zu beobachten, dass die Verwendung des Votingclassifiers nicht zu einem Anstieg der Accuracy im Vergleich mit den Einzeltools geführt hat. Zwar gab es einzelne Konstellationen, in denen der Votingclassifier besser war als die Einzeltools innerhalb dieser Konstellation, im übergreifenden Vergleich mit allen Durchläufen aber konnte der Votingclassifier sich im Bezug auf die Accuracy nicht beweisen.

Innerhalb des Api Datensatzes lag der Unterschied allerdings nur bei 1%, das Tool von Meyer und Horstmann erreichte eine Accuracy von maximal 74%, der Votingclassifier maximal 73%.

Die Unterschiede der Accuracy Werte sind bei dem App Datensatz etwas eindeutiger, hier erreicht der Votingclassifier maximal 62%, das *RoBERTa* Modell hingegen 77% unter Verwendung des Github Datensatzes für das Vortraining.

Anschließend habe ich noch verglichen, wie die Ergebnisse der **RQ2** sich von den Accuracy Werten unterscheiden, wenn die Tools innerhalb der selben Domäne trainiert wurden. Diese Frage wurde unter **RQ3** gestellt.

Auch hier unterlag der Votingclassifier dem Einzeltool, allerdings gab es erneut Anzeichen dafür, dass der Votingclassifier eventuell zu ausgewogeneren Klassifizierungsergebnissen führen kann. So lag der F1-Wert der Dokumentenklasse *Neutral* für *RoBERTa* bei 0%, es wurden also keine Dokumente dieser Klasse korrekt gelabelt. Der Votingclassifier hingegen erreichte einen Wert von 17%.

6.2 Verbesserung des Votingclassifiers

Durch das Auflösen von Gleichständen bei der Klassifizierung per Zufallsentscheidung ist der Votingclassifier nicht zwangsläufig deterministisch. Das bedeutet, dass dieselbe Eingabe unter Umständen zu unterschiedlichen Ergebnissen führen kann. Wie bereits in Abschnitt 5.4 erwähnt, lag der Anteil an Zufallsentscheidungen bei meinen Untersuchungen zwischen 0.1% und 0.8%. Zur Optimierung dieses Verhaltens wäre deswegen ein alternativer Ansatz denkbar.

So könnte man zum Beispiel einfach immer das Label des Einzeltools übernehmen, welches historisch die höchste Accuracy hat. Diesen Wert könnte man aus den Modelldetails im Setup einlesen und entsprechend setzen. Alternativ könnte man die zu setzenden Label iterieren. Dieses Verhalten wäre zwar immer noch willkürlich, würde dafür aber die Zufallskomponente

eliminieren.

Ebenfalls nimmt die gesamte Klassifizierung mit dem Votingclassifier einiges an Zeit in Anspruch. Für den Api Datensatz mit 4522 Dokumenten dauerte die Klassifizierung mit dem Tool von Meyer 4697 Sekunden, *RoBERTa* benötigte 3033 Sekunden und Senti4SD lediglich 133 Sekunden. Somit ergibt sich eine Gesamtlaufzeit für den Votingclassifier auf diesen Datensatz von etwa 131 Minuten. Zur Optimierung der Laufzeiten wäre das Verwenden einer dedizierten Grafikkarte sowie die Parallelisierung der drei verwendeten Tools denkbar.

6.3 Ausblick

Bei der Betrachtung der Messwerte habe ich festgestellt, dass es Anzeichen dafür gibt, dass die Verwendung eines Votingclassifiers zu ausgewogeneren Ergebnissen zwischen den Dokumentenklassen führen kann. Deswegen bin ich der Ansicht, dass es interessant sein kann, dieses Verhalten näher zu untersuchen. Dazu könnte man beispielsweise die F1-Werte aller Klassen für jede Konstellation ermitteln und schauen, ob sich ein entsprechendes Muster wiederholt.

Ebenfalls war zu beobachten, dass alle Einzeltools bei Vortraining auf dem Jira Datensatz schlechter abschneiden im Vergleich zu einem Vortraining auf einem der anderen beiden Datensätzen. Dies könnte darauf hindeuten, dass der Jira Datensatz eventuell nicht so gut für Klassifizierungsaufgaben geeignet ist, wie etwa der Github Datensatz. Somit ist eine Betrachtung und Auswertung weiterer Datensätze ein möglicher Ansatz, um eventuell die gesamte Accuracy des Votingclassifiers zu verbessern.

Auch das Verwenden anderer Tools ist ein möglicher Ansatz. Es ist zum Beispiel denkbar, eines der beiden Tools, welches nicht auf einem neuronalen Netz basiert, gegen ein solches auszutauschen. Besonders auf dem App Datensatz war das *RoBERTa* Modell deutlich besser als die anderen beiden Tools. Somit könnte das Verhalten untersucht werden, wenn man das Ensemble durch weitere Modelle ergänzt. Zhang et al. untersuchten in ihrer Arbeit unter anderem die neuronalen Netze *BERT, RoBERTa, XLNet und ALBERT*. Es könnte durchaus interessant sein, ein Ensemble nur aus Tools, welche auf auf neuronalen Netzen basieren, zu erstellen und dessen Ergebnisse mit dem Votingclassifier dieser Arbeit zu vergleichen.

Des Weiteren kann es auch interessant sein, die Anzahl der Tools innerhalb des Votingclassifiers zu erweitern, um somit eine noch größere Domänenabdeckung zu ermöglichen. Man könnte somit einen weiteren Datensatz innerhalb des Trainings in Betracht ziehen.

Literaturverzeichnis

[1] F. Calefato, F. Lanubile, F. Maiorano, and N. Novielli. Sentiment polarity detection for software development. *Empir. Softw. Eng.*, 23(3):1352–1382, 2018.

[2] Z. Chen, Y. Cao, X. Lu, Q. Mei, and X. Liu. Sentimoji: an emoji-powered learning approach for sentiment analysis in software engineering. In M. Dumas, D. Pfahl, S. Apel, and A. Russo, editors, *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, pages 841–852. ACM, 2019.

[3] J. Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960.

[4] J. F. DeFranco and P. A. Laplante. Review and analysis of software development team communication research. *IEEE Transactions on Professional Communication*, 60(2):165–182, 2017.

[5] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[6] D. Graziotin, X. Wang, and P. Abrahamsson. Happy software developers solve problems better: psychological measurements in empirical software engineering. *PeerJ*, 2:e289, 2014.

[7] D. Graziotin, X. Wang, and P. Abrahamsson. Do feelings matter? on the correlation of affects and the self-assessed productivity in software

engineering. *Journal of Software: Evolution and Process*, 27(7):467–487, 2015.

[8] T. Hastie, R. Tibshirani, and J. H. Friedman. *The elements of statistical learning: Data mining, inference, and prediction.* Springer series in statistics. Springer, New York, second edition edition, 2009.

[9] J. Horstmann. Computer-gestützte analyse des kommunikationsverhaltens in entwicklerteams unter berücksichtigung digitaler medien., Master's thesis, Gottfried Wilhelm Leibniz University of Hannover, Germany, 2019.

[10] R. Jolak, A. Wortmann, M. Chaudron, and B. Rumpe. Does distance still matter? revisiting collaborative distributed software design. *IEEE Software*, 35(6):40–47, 2018.

[11] M. Kuhrmann, P. Tell, J. Klünder, R. Hebig, S. Licorish, and S. MacDonell. Helena stage 2 results, https://doi.org/10.13140/RG.2.2.14807.52649, 2018.

[12] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, M. Lanza, and R. Oliveto. Sentiment analysis for software engineering. In M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, editors, *Proceedings of the 40th International Conference on Software Engineering*, pages 94–104, New York, NY, USA, 2018. ACM.

[13] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

[14] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval.* Cambridge Univ. Press, Cambridge, reprinted. edition, 2009.

[15] C. Meyer. Verbesserung von evolutionären algorithmen zur klassifkation von schriftlicher kommunikation in entwicklungsteams, Bachelor's thesis, Gottfried Wilhelm Leibniz University of Hannover, Germany, 2020.

[16] N. Novielli, F. Calefato, D. Dongiovanni, D. Girardi, and F. Lanubile. Can we use se-specific sentiment analysis tools in a cross-platform setting? 37:158–168, 2020.

[17] N. Novielli, F. Calefato, D. Dongiovanni, D. Girardi, and F. Lanubile. A gold standard for polarity of emotions of software developers in github, Mar 2020.

[18] N. Novielli, F. Calefato, and F. Lanubile. A gold standard for emotions annotation in stack overflow. In *Proc. of 15th Int'l Conf. on Mining Software Repositories*, MSR 2018, pages 14–17, 2018.

[19] N. Novielli, F. Calefato, F. Lanubile, and A. Serebrenik. Assessment of se-specific sentiment analysis tools: An extended replication study. *CoRR*, abs/2010.10172, 2020.

[20] 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. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 303–313. IEEE, 2015.

[21] M. Ortu, A. Murgia, G. Destefanis, P. Tourani, R. Tonelli, M. Marchesi, and B. Adams. The emotional side of software developers in jira. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, page 480–483, New York, NY, USA, 2016. Association for Computing Machinery.

[22] B. Pang and L. Lee. Opinion mining and sentiment analysis. *Found. Trends Inf. Retr.*, 2(1–2):1–135, Jan. 2008.

[23] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, pages 1–26, 2020.

[24] P. Shaver, J. Schwartz, D. Kirson, and C. O'Connor. Emotion knowledge: Further exploration of a prototype approach. *Journal of Personality and Social Psychology*, 52(6):1061–1086, 1987.

[25] G. Uddin and F. Khomh. Automatic mining of opinions expressed about apis in stack overflow. *IEEE Transactions on Software Engineering*, 47(3):522–559, 2021.

[26] T. Zhang, B. Xu, F. Thung, S. A. Haryono, D. Lo, and L. Jiang. Sentiment analysis for software engineering: How far can pre-trained transformer models go? In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 70–80. IEEE, 2020.