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

Automatische Klassifikation von Aussagen in Meetings von Entwicklungsteams

Automatic Classification of Statements in Meetings of Development Teams

Bachelorarbeit

im Studiengang Informatik

von

Marc Herrmann

Prüfer: Prof. Dr. rer. nat. Kurt Schneider
Zweitprüfer: Dr. rer. nat. Jil Ann-Christin Klünder
Betreuer: Dr. rer. nat. Jil Ann-Christin Klünder

Hannover, 03.03.2021

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig 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 03.03.2021

_____

Marc Herrmann

Zusammenfassung

Gute Kommunikation innerhalb von Entwicklungsteams spielt eine wichtige Rolle für den Erfolg in der Softwareentwicklung und trägt entscheidend dazu bei Probleme bei der Entwicklung frühzeitig zu erkennen und zu beheben. Um die textuelle Kommunikation innerhalb von Entwicklungsteams zu analysieren wurde bereits ein Algorithmus entwickelt welcher, mithilfe von maschinellen Lernverfahren, eine Klassifikation schriftlicher Kommunikation in die Kategorien positiv, negativ und neutral vornimmt. Dieser Algorithmus soll nun genutzt werden, um nicht nur die textuelle, sondern auch die sprachliche Kommunikation in Meetings zu analysieren und dem Projektleiter zum Ende eines Meetings direkt Feedback über dessen Verlauf zu geben. Die Schwierigkeit dabei besteht darin, aufgenommene Sprache ausreichend gut zu transkribieren und weiterzuverarbeiten. Dafür wurde in dieser Arbeit ein Konzept entwickelt und implementiert, welches durch Mikrofonzugriff Ton in Meetings aufzeichnet, mithilfe einer Open Source Sprachengine transkribiert, und in einem Ablauf von Verarbeitungsschritten, welcher mit der Ausgabe der Vorhersagen der Kategorien positiv, negativ und neutral zu den Aussagen endet, weiter verarbeitet. Die Anwendbarkeit des Verfahrens wurde anhand einer Studie mit einem aufgezeichneten Gespräch eines studentischen Entwicklungsteams demonstriert. Es konnte eine moderate Übereinstimmung zu der Klassifikation durch einen menschlichen Zuhörer nachgewiesen werden. Möglichkeiten das Verfahren zu optimieren, um die Ergebnisse weiter zu verbessern, sowie andere Anwendungsfälle werden vorgestellt.

Abstract

Automatic Classification of Statements in Meetings of Development Teams

Good communication within development teams is an important factor for success in software development and makes a decisive contribution to recognizing and fixing problems at an early stage of development. To analyze the textual communication within development teams an algorithm has been created, which relies on machine learning methods, to classify textual communication into positive, negative, and neutral categories. Instead of classifying only textual communication, this algorithm should be used now to also classify conversations in meetings directly to provide feedback about the course of a meeting to the project manager afterward. The difficulty consists in making the transcription of recorded audio good enough to process it further. Therefore, a concept has been created in this thesis, to capture microphone audio directly, transcribe it using an open-source speech engine, and apply further processing steps, to finally output the forecast of the categories positive, negative, and neutral for each statement, in a single sequence of processing steps. The applicability of the concept has been demonstrated within a study using a recorded meeting of a student development team. Moderate agreement between the classifications of the software and a human observer could be demonstrated. Possibilities to improve the concept further and other use cases will be presented.

Inhaltsverzeichnis

Kapitel 1

Einleitung

In den Forschungsgebieten der angewandten Informatik und Computer-Linguistik wird sich schon seit geraumer Zeit damit beschäftigt, zu versuchen, Computern beizubringen die menschliche Sprache zu erkennen und zu verstehen. Ein vielversprechender Ansatz dabei ist neuronale Netze mithilfe von Sprach-Aufnahmen und dazugehörigen Transkripten zu trainieren, und die dadurch gewonnenen Modelle zu nutzen, um von Menschen gesprochene Wörter und Gespräche zu erkennen und zu transkribieren [32, 39, 40]. Das Fachgebiet Software Engineering der Gottfried Wilhelm Leibniz Universität Hannover beschäftigt sich damit Meetings von Entwicklungsteams im Bereich der Softwareentwicklung zu analysieren [41], mit dem Ziel aus gewonnen Resultaten Optimierungen vornehmen zu können. Im Rahmen der Forschung von Klünder et al. [16] wurde daher das Verfahren *Act4teams-SHORT* entwickelt, welches auf dem Verfahren *Act4teams* von Kauffeld [12] beruht. Mit Act4teams-SHORT können Interaktionen in Meetings von Entwicklungsteams in 11 verschiedene Kategorien sortiert und anschließend analysiert werden [14]. Da dieses Verfahren aber aktuell nur von Hand durch einen menschlichen Beobachter, der sich mit im Meeting befindet, angewendet werden kann gibt es die Bestrebung das Verfahren langfristig zu automatisieren [14]. Dafür wurde bereits von Horstmann [11] ein Verfahren entwickelt, welches schriftliche Kommunikation in Entwicklungsteams mithilfe verschiedener Merkmale, Klassifizierungsalgorithmen, und einem evolutionären Algorithmus (EA) analysiert. Dabei wird für einzelne Sätze, anhand von Metriken die durch *Natural Language Processing* (NLP) extrahiert werden, mithilfe von einem durch maschinelle Lernverfahren trainiertem Modell eine Vorhersage über deren Sentiment (positiv, negativ oder neutral) getroffen. Anhand dessen kann anschließend der Verlauf des Sentiments über längere Zeiträume analysiert werden. Somit können Rückschlüsse auf die Stimmung innerhalb des Teams erlangt werden. Dieses Verfahren wurde von Meyer [24] bereits für englische schriftliche Kommunikation optimiert. Der Algorithmus von Horstmann und Meyer [11, 24] wird im folgenden als

Klassifikationsalgorithmus bezeichnet, diese Arbeit baut jedoch vollständig auf der deutschen Version des Algorithmus von Horstmann [11] ohne die Optimierungen von Meyer [24] auf. Der Klassifikationsalgorithmus [11, 24] und die vielversprechenden Ansätze im Bereich der Spracherkennung sollen nun zusammengeführt werden, um es in der ersten Iteration, mithilfe einer Software zu ermöglichen, Aussagen in Meetings vollautomatisiert zu transkribieren und mithilfe des Klassifikationsalgorithmus [11, 24] zunächst in die drei groben Kategorien positiv, negativ und neutral unterteilen zu können. Dieses Vorhaben soll als Grundlage für die langfristige Weiterentwicklung dienen, um es in Zukunft zu ermöglichen eine vollautomatisierte Unterteilung in die 11 verschiedenen Kategorien von Interaktionen aus dem Verfahren Act4teams-SHORT [14, 16] zu ermöglichen.

1.1 Problemstellung

Meetings sind in allen Bereichen der Wirtschaft präsent und die wichtigste Art in Teams die am selben Projekt arbeiten zu kommunizieren [41]. Daher ist es für den Erfolg eines Softwareprojektes entscheidend Meeting Abläufe zu optimieren und mögliche Probleme frühzeitig zu erkennen [41]. Mögliches problematisches Verhalten wäre beispielsweise das stetige Ansprechen von Problemen ohne zugehörige Lösungsvorschläge zu nennen, da dies zu schlechter Stimmung im Team führen kann [16]. Die entsprechenden Grundlagen dafür sind bereits durch die Verfahren Act4teams [12] beziehungsweise Act4teams-SHORT [16] gegeben und wurden außerdem bereits von Horstmann [11] und Meyer [24] teil automatisiert. Aktuell ist dennoch ein menschlicher Beobachter im Meeting nötig der das Gespräch von Hand transkribiert. Neben zusätzlichen (Personal-)Kosten führt dies auch vermehrt zum Gefühl der Beobachtung und beeinflusst möglicherweise die Aussagen einzelner Teilnehmer. Der zeitliche Aufwand ist auch um einiges höher, da die transkribierten Daten erst wieder manuell in den Klassifikationsalgorithmus [11, 24] eingespeist werden müssen. Damit kein menschlicher Beobachter in den Meetings mehr nötig ist, und um Zeit und Kosten bei der Analyse zu sparen soll das Verfahren zunächst so weit automatisiert werden, dass das Gespräch im Meeting durch einen Computer mit Mikrofon aufgezeichnet und automatisch transkribiert werden kann. Dieses Transkript kann dann als Eingabe für den Klassifikationsalgorithmus [11, 24] verwendet und weiterverarbeitet werden. Das Kernproblem dabei ist, dass die Gespräche ausreichend gut transkribiert werden müssen, um eine möglichst geringe Fehlerquote nach der Auswertung zu erhalten. Die Erkennung menschlicher Sprache durch einen Computer ist jedoch ein sehr komplexes Problem. Bei Betrachtung der deutschen Sprache, welche im Rahmen dieser Arbeit erkannt werden soll, als Beispiel, umfasst neben anderen das Deutsche Wörterbuch der Brüder

Jakob und Wilhelm Grimm [8] in etwa 450.000 Stichwörter. Eine aktuelle Auflage des Dudens [5] kommt immerhin noch auf rund 148.000 Stichwörter. Da es um einiges schwieriger ist kontinuierliche Sprache zu erkennen, als diskrete Sprache zu erkennen und dass es viele verschiedenen Stimmen und Akzente gibt, wird schnell klar, dass das Problem alles andere als trivial ist. Aus diesen Überlegungen ergibt sich die folgende zentrale Forschungsfrage:

Zentrale Forschungsfrage *Wie gut kann die zu entwickelnde Software Gespräche in Meetings transkribieren, und die einzelnen Aussagen in die Kategorien positiv, negativ und neutral einstufen, um aussagekräftige Ergebnisse zu erhalten?*

1.2 Lösungsansatz

Um die Problemstellung zu lösen ist zunächst ein theoretisches Konzept nötig, um das weitere Vorgehen zu planen. Dieses Konzept wird in Kapitel 3 ausführlich behandelt. Damit am Ende der Arbeit ausreichend gute Ergebnisse erzielt werden können ist die Qualität der automatisch, von der zu entwickelnden Software, erstellten Transkripte maßgeblich für den Erfolg des Vorhabens. Um dies zu gewährleisten muss auf bereits vorhandene Open Source Software zur Spracherkennung zurückgegriffen werden, da die eigene Entwicklung einer solchen Software den Rahmen dieser Arbeit sprengen würde. Dafür ist es nötig eine Software für die Einbindung zu finden welche beim Transkribieren von Texten eine geringe WER[1] und CER[2] aufweist. Dies muss des Weiteren in deutscher Sprache möglich sein und nicht wie bei den meisten vorhandenen Fertiglösungen auf Englisch oder Mandarin, da die Software hauptsächlich in Entwicklungsteams eingesetzt werden soll, welche auf Deutsch kommunizieren. Dieses Problem lässt sich, mithilfe von neuronalen Netzen auf Deutsch trainierten, Open Source Modellen für die später verwendete Sprachengine lösen. Bei der Auswahl des deutschen Sprachmodells muss dabei wieder auf eine geringe WER und CER geachtet werden, um bei der späteren Anwendung der zu entwickelnden Software verwendbare Ergebnisse zu erzielen. Anschließend müssen die Ergebnisse dieser Transkription dann geeignet formatiert in den Klassifikationsalgorithmus [11, 24] eingegeben werden, um sie auszuwerten. All diese einzelnen Verarbeitungsschritte sollen idealerweise automatisch durch die Ausführung der zu entwickelnden Software erledigt werden, sodass die Benutzung auch für unerfahrene Benutzer möglich ist. Am Ende soll das Ergebnis der Klassifikation durch die Software idealerweise unwesentlich von der manuellen Klassifikation durch einen menschlichen Zuhörer abweichen.

[1] Wortfehlerquote (engl. Word Error Rate)
[2] Buchstabenfehlerquote (engl. Character Error Rate)

1.3 Zielsetzung der Arbeit

Die Eingangs beschriebene vollständig automatisierte Analyse von Interaktionen wie in den Verfahren Act4teams und Act4teams-SHORT wird vom Fachgebiet Software Engineering der Gottfried Wilhelm Leibniz Universität Hannover als eines der Forschungsziele für die nächsten Jahre behandelt. Diese Arbeit soll einen wichtigen ersten Schritt, welcher zur Vervollständigung dieses Zieles notwendig ist, darstellen. Ziel der zu entwickelnden Software ist es, zu ermöglichen in einem einzelnen automatisierten und implementierten Verfahren, die Sprache in Meetings in einzelne Aussagen aufgegliedert zu transkribieren. Anschließend sollen die transkribierten Daten im selben Prozess durch den Klassifikationsalgorithmus [11, 24] evaluiert, und anhand ihrer Polaritäten klassifiziert werden. Die Ergebnisse sollen dem Nutzer direkt in geeigneter Form angezeigt werden, um dem Team nach dem Meeting die Stimmung einzelner Aussagen und des gesamten Gespräches zurückzumelden. In dem Fall, dass unverhältnismäßig viele negative Aussagen vorliegen, kann der Projektleiter dann weitere Maßnahmen ergreifen, um die Stimmung im Team und künftigen Meetings zu verbessern.

1.4 Struktur der Arbeit

In Kapitel 2 wird der Leser zunächst darauf hingewiesen welche grundlegenden Kenntnisse zum Verständnis dieser Arbeit erforderlich sind, bevor weiterführende Grundlagen, die auf den Vorkenntnissen des Lesers basieren, erklärt werden. Dazu gehören die von *Baidu* entwickelte *Deepspeech* Architektur und dessen Implementation von *Mozilla*. Kapitel 3 befasst sich mit dem theoretischen Konzept, welches entwickelt wurde um das in der Einleitung erläuterte Problem zu lösen. Die Implementierung dieses theoretischen Konzeptes wird dem Leser in Kapitel 4 detailliert beschrieben. Dort werden technische Hintergründe der Implementation sowie deren Besonderheiten aufgezeigt. Nach diesem Kapitel sollte dem Leser die Funktionsweise der entwickelten Software klar sein. Anschließend werden in Kapitel 5 die Ergebnisse der Software evaluiert, um daraus in Kapitel 6 die zentrale Forschungsfrage zu beantworten, sowie mögliche Verbesserungen anzusprechen. Im folgenden Kapitel 7 wird die Abgrenzung dieser Arbeit zu verwandten Arbeiten vorgenommen, indem wieder auf die im Hauptteil behandelten Themen eingegangen wird. Zum Schluss werden das Vorhaben und die Resultate dieser Arbeit in Kapitel 8 noch einmal zusammengefasst. Außerdem wird ein Ausblick für die Weiterentwicklung dieser Ergebnisse, für die Forschung am Fachgebiet Software Engineering, und darüber hinaus, gegeben.

Kapitel 2

Grundlagen

Es wird angenommen, dass Leser dieser Arbeit ein grundlegendes Verständnis in Bereichen der angewandten Informatik besitzen. Dazu gehören die grundlegende Funktionsweise von maschinellem Lernen, Deep Learning, neuronalen Netzen und der Anwendung *TensorFlow* [7]. Außerdem sollte der Leser mit der Programmiersprache *Python* [45] vertraut sein, welche ein zentraler Bestandteil im Kapitel der Umsetzung sein wird. Grundlegende Begriffe im Bereich der Spracherkennung wie kontinuierliche und diskrete Sprache werden ebenfalls als bekannt vorausgesetzt und nicht weiter erklärt. Des Weiteren wird angenommen, dass Leser mit der Forschung von Klünder et al. [16] und deren Resultaten vertraut sind, da diese Arbeit darauf aufbaut. Dazu gehören insbesondere das Verfahren Act4teams-SHORT [9, 16], sowie der Klassifikationsalgorithmus [11, 24], welcher in dieser Arbeit wiederverwendet wird. Im Folgenden werden die darüber hinaus gehenden Grundlagen, welche zum weiteren Verständnis nötig sind abgehandelt. Zunächst wird in Abschnitt 2.1 noch einmal auf den Stand der Technik, auf welchem diese Arbeit aufbaut, eingegangen und insbesondere die Arbeit von Horstmann [11] behandelt, bevor in Abschnitt 2.2 auf die Funktionsweise von rekurrenten neuronalen Netzen eingegangen wird, und die Unterschiede und Vorteile zu herkömmlichen neuronalen Netzen erklärt werden. Diese werden auch im darauf folgenden Abschnitt 2.3 der Deepspeech Architektur von Baidu verwendet werden. Dort werden ebenfalls Unterschiede und Vorteile im Vergleich zu traditionellen Ansätzen aufgezeigt. Die Open Source Deepspeech Implementierung von Mozilla, welche eine zentrale Rolle der in dieser Arbeit entwickelten Software spielt, wird im letzten Abschnitt 2.4 der Grundlagen ausführlich erklärt.

2.1 Stand der Technik

Horstmann [11] untersuchte im Rahmen seiner Masterarbeit die schriftliche Kommunikation in Gruppenchats von Softwareentwicklungsteams. Dabei entwickelte er ein Verfahren welches im ersten Schritt mittels *Natural Language Processing* (NLP) verschiedene Metriken aus allen einzelnen Aussagen beziehungsweise Textnachrichten eines Textverlaufs extrahiert [11]. Zu den Metriken die dabei betrachtet werden gehören sowohl statistische Metriken wie beispielsweise die Länge und Häufigkeit einzelner Wörter sowie deren Verhältnis zum Rest des Satzes, als auch inhaltliche Metriken, wie die Polarität einzelner Wörter, welche durch die Datensätze von Waltinger [46] und Remus et al. [37] bestimmt wird. Anschließend kann mit maschinellen Lernverfahren ein Modell trainiert werden, indem den einzelnen Textnachrichten manuell die Trainingslabel positiv, negativ und neutral hinzugefügt werden [11]. Der *evolutionäre Algorithmus* (EA) schließt dann im Training einzelne Metriken ein und aus um das beste Modell zu bestimmen [11]. Für die Vorhersage einer Klasse einer Aussage ohne Label wird dann auf das Modell zurückgegriffen [11]. Dazu werden drei einzelne Klassifikatoren (*Random Forests*, *Support Vector Machine* und *Naive Bayes*) herangezogen die unabhängig voneinander eine Vorhersage treffen [11]. Ein *VotingClassifier* bestimmt schlussendlich gewichtet die mehrheitliche Klasse, die dann für die Aussage hervorgesagt wird [11]. Horstmann validierte seine Ergebnisse anhand eines großen Datensatzes von Textnachrichten aus einem dezentralen Entwicklerteam eines privaten Unternehmens [11]. Der EA erreichte dabei im Training eine Genauigkeit von 63%, welche jedoch nur unwesentlich schlechter war als die Genauigkeit der Label die manuell von einer Person ausgewählt wurden [11]. Horstmann nutze sein Verfahren um aus dem Datensatz des Unternehmens eine Trendanalyse zu erstellen, welche den Verlauf der Stimmung im Team über einen längeren Zeitraum durch eine auf und ab schwankende Linie, welche die Datenpunkte der einzelnen Tage verbindet, visualisiert [11].

2.2 Rekurrente neuronale Netze

Ein rekurrentes, oder auch rückgekoppeltes, neuronales Netz (RNN) kennzeichnet sich durch eine Verbindung des Ausgangssignals von Neuronen zu sich selbst (direkte Rückkopplung), zu Neuronen derselben Schicht (seitliche Rückkopplung), oder zu Neuronen einer der vorherigen Schichten (indirekte Rückkopplung) [17]. In der Abbildung 2.1 ist die Kante E_1 ein Beispiel für einen indirekte Rückkopplung, die Kante E_2 ist ein Beispiel für eine seitliche Rückkopplung und die Kante E_3 ist ein Beispiel für eine direkte Rückkopplung. Die unbeschrifteten Kanten sind nicht rekurrent.

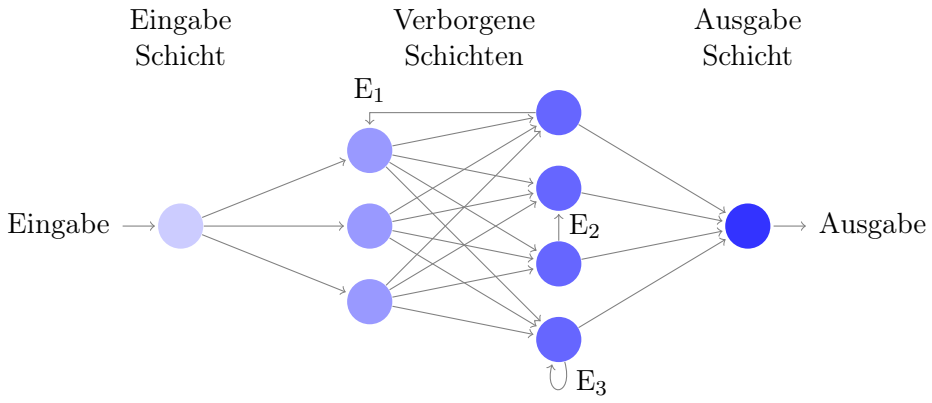

Abbildung 2.1: Beispiel für ein rekurrentes neuronales Netz.

Auf diese Art und Weise unterscheiden sie sich von herkömmlichen Feedforward-Netzen, in welchen die Ausgänge der einzelnen Neuronen nur mit den Eingängen von Neuronen der jeweils nächsten Schicht verbunden sind. Diese, erweiterten, rekurrenten neuronalen Netze bilden ebenfalls näher die biologischen neuronalen Netze im menschlichen Gehirn nach, da es dort ebenfalls entsprechende rückkoppelnde Verbindungen gibt [17]. Rekurrente neuronale Netze finden außerdem häufig in der Spracherkennung Anwendung [39].

2.2.1 Mathematische Hintergründe

Die folgenden Beispiele dienen der Veranschaulichung von (rekurrenten) neuronalen Netzen und sind Kruse et al. [17] entnommen.

Graphentheoretische Grundlagen

Ein (gerichteter) **Graph** ist ein Tupel $G = (V, E)$, bestehend aus einer (endlichen) Menge V von **Knoten** oder **Ecken** und einer (endlichen) Menge $E \subseteq V \times V$ von **Kanten**.

Wir nennen eine Kante $e = (u, v) \in E$ **gerichtet** von Knoten u zu Knoten v.

Sei $G = (V, E)$ ein (gerichteter) **Graph** und $u \in V$ ein Knoten. Dann werden die Knoten der Menge

$$pred(u) = \{v \in V | (v, u) \in E\}$$

die **Vorgänger** des Knotens u und die Knoten der Menge

$$succ(u) = \{v \in V | (u, v) \in E\}$$

die **Nachfolger** des Knotens u genannt.

Allgemeine Definition eines neuronalen Netzes
Ein (künstliches) neuronales Netz ist ein (gerichteter) Graph $G = (U, C)$, dessen Knoten $u \in U$ **Neuronen** oder **Einheiten** und dessen Kanten $c \in C$ **Verbindungen** genannt werden. Die Menge U der Knoten wird partitioniert in

- die Menge U_{in} der **Eingabeneuronen**,
- die Menge U_{out} der **Ausgabeneuronen**, und
- die Menge U_{hidden} der **versteckten Neuronen**.

Es gilt:

$$U = U_{in} \cup U_{out} \cup U_{hidden},$$

$$U_{in} \neq \emptyset,\ U_{out} \neq \emptyset,\ U_{hidden} \cap (U_{in} \cup U_{out}) = \emptyset.$$

Typen (künstlicher) neuronaler Netze

- Falls der Graph eines neuronalen Netzes **azyklisch** ist, wird das Netz **Feed-Forward-Netz** genannt.
- Falls der Graph eines neuronalen Netzes **Zyklen** enthält, (rückwärtige Verbindungen), wird es **rekurrentes neuronales Netz** genannt.

2.3 Baidu Deepspeech Architektur

Baidu Research, die Forschungsabteilung des chinesischen Unternehmens Baidu, bringt viele Top-Talente der Informatik und anderen Ingenieurwissenschaften an den Standorten Seattle, Peking, und im Silicon Valley zusammen um gemeinsam die zukunftsorientierte Forschung in den Bereichen künstliche Intelligenz, maschinelles Lernen, und Deep Learning voranzutreiben und grundlegende neue Erkenntnisse zu erlangen. [3]. Hannun et al. [10] veröffentlichten 2014 im Rahmen der Forschung vom Baidu Research Silicon Valley AI Lab ihren Artikel *Deep Speech: Scaling up end-to-end speech recognition* [10], welcher die Deepspeech Architektur erstmals für die Öffentlichkeit im Detail vorstellte, und es so auch anderen Forschern ermöglichte die Erkenntnisse und Ideen für ihre eigenen Anwendungen zu nutzen. Die Architektur wird von Hannun et al. [10] als signifikant simpler, sowie performanter als herkömmliche Sprachsysteme beschrieben. Auf die detaillierten Unterschiede zu traditionellen Systemen zur Spracherkennung wird im folgenden Abschnitt 2.3.1 ausführlich eingegangen. Laut eigenen Angaben der Autoren erreichte ihr eigenes System unter Verwendung der

Deepspeech Architektur auf dem *Switchboard Hub5'00* Benchmark eine durchschnittliche Fehlerquote von 16,0% über den gesamten Datensatz hinweg [10, 38] (Details in Abschnitt 2.3.3). Des Weiteren geben Hannun et al. [10] an, dass ihr System in herausfordernden Situationen mit geräuschvollen Umgebungen besser performt als kommerzielle *State of the Art* Systeme. Auch dieser Punkt macht Deepspeech zu einer geeigneten und interessanten Lösung für das Vorhaben dieser Arbeit, da in Meetings in Büroräumen oftmals auch eine gewisse Kulisse von Hintergrundgeräuschen präsent sein kann. Diese soll die Spracherkennung der zu entwickelnden Software nicht beeinträchtigen, da die Umgebungsgeräusche meist unvermeidbar sind.

2.3.1 Unterschiede zu traditionellen Ansätzen

Traditionelle Sprachsysteme basieren meistens auf mühsam entwickelten Sprachverarbeitungs-Pipelines, wohingegen Deepspeech gänzlich auf solche, von Hand entworfenen, Komponenten zur Modellierung von Hintergrundgeräuschen, Hall und unterschiedlichen Sprechern verzichtet [10]. Stattdessen lernt Deepspeech direkt mit einer Funktion welche robust gegen diese Effekte ist [10]. Im traditionellen Systemen werden außerdem oft Phonem-Lexika zur Spracherkennung verwendet [10]. Ein Phonem ist die kleinste Einheit eines Wortes, welche ausschlaggebend für dessen Bedeutung ist [19]. Der Austausch eines solchen Phonems gegen ein anderes verändert also die Bedeutung des Wortes, wie folgendes Beispiel veranschaulicht [19].

Phonemaustausch im **Inlaut**:	W**o**che - W**a**che S**o**g - S**ie**g
Phonemaustausch im **Anlaut**:	**F**all - **B**all **D**orf - **T**orf
Phonemaustausch im **Auslaut**:	Ruh**e** - Ruh**m** Hau**s** - Hau**t**

Tabelle 2.1: Beispiele zum Phonemaustausch [19].

Phoneme sind also die kleinsten lautlichen Bestandteile von Wörtern mit der Funktion Bedeutungsunterschiede zu kennzeichnen [19]. Daher ist klar, dass diese in der Spracherkennung eine entscheidende Rolle spielen um die richtige Bedeutung eines Wortes zu erfassen und die Wordfehlerquote (WER) beziehungsweise Buchstabenfehlerquote (CER) zu senken. Im Gegensatz zu den herkömmlichen verfahren braucht Deepspeech weder ein solches Phonem-Lexikon, um Unterscheidungen vorzunehmen, noch überhaupt das Konzept eines Phonems [10]. Wie eingangs erwähnt verwenden traditio-

nelle Systeme oft von Hand entwickelte Komponente wie verschiedene Verarbeitungsebenen, spezialisierte Eingabedaten, akustische Modelle und verdeckte Markowmodelle (HMM[1]) [10]. Die Entwickler der traditionellen Systeme müssen dafür einen großen Aufwand betreiben um von Hand ein Feintuning der einzelnen Sprachmodelle und Parameter des Systems vorzunehmen [10]. Die Einführung von Deep Learning Algorithmen hat zwar dafür gesorgt, dass dieser Aufwand verringert und die Performanz der Systeme deutlich verbessert wurde, aber in den meisten fällen nur durch das automatisierte Feintuning der Sprachmodelle und Parameter [10]. Damit spielt Deep Learning immer noch eine untergeordnete Rolle in diesen Systemen [10]. Als Resultat dessen müssen diese Systeme, damit sie auch in Umgebungen mit lauten Hintergrundgeräuschen funktionieren, mühsam auf Robustheit entwickelt werden, wohingegen Deepspeech Deep Learning mit, den in Abschnitt 2.2 angesprochenen, rekurrenten neuronalen Netzen verwendet [10]. Dabei bedient Deepspeech sich der Vorteile von Deep Learning, um von großen Datensätzen lernen zu können und damit nicht nur die allgemeine Performanz zu steigern, sondern, da es sich bei dem trainieren der neuronalen Netze um einen *End-to-End* Prozess mit dem Ziel Transkripte zu produzieren handelt, mit genug Trainingsdaten und Rechenleistung selber Robustheit gegen Hintergrundgeräusche und unterschiedliche Sprecher zu lernen [10]. Um diese Funktionsweise zu demonstrieren führten Hannun et al. [10] einen Test von Deepspeech gegen etablierte Spracherkennungssysteme von Unternehmen wie unter anderem *Apple*, *Google* und *Bing* durch. Dazu entwickelten sie einen eigenen Benchmark um die Performanz in Umgebungen mit Hintergrundgeräuschen zu testen, welcher aus 100 klaren und 100 mit Hintergrundgeräuschen belasteten Aufnahmen von 10 verschiedenen Sprechern bestand [10]. Dieses Testset wurde dann auf die Aufnahmen reduziert für welche alle Dienste ein nicht leeres Ergebnis lieferten, und ergab die folgenden Ergebnisse.

[1] engl. Hidden Markov Model

System	Klare Aufnahmen (94 gewertet)	Aufnahmen mit Hintergrund-geräuschen (82 gewertet)	Kombiniert (176 gewertet)
Apple Dictation	14.24	43.76	26.73
Bing Speech	11.73	36.12	22.05
Google API	6.64	30.47	16.72
wit.ai	7.94	35.06	19.41
Deepspeech	**6.56**	**19.06**	**11.85**

Tabelle 2.2: Ergebnisse in prozentualer Wortfehlerquote (WER), entnommen aus Hannun et al. [10].

Deepspeech konnte sich wie an den Ergebnissen zu erkennen ist nicht nur mit den Technologien anderer Großkonzerne im Techniksektor mithalten, sondern diese auch, vor allem in den Aufnahmen mit Hintergrundgeräuschen, und damit auch insgesamt, übertreffen. Auf dieselbe Art und Weise konnte mit dem Training mit vielen Grafikprozessoren und vielen tausenden Stunden an Trainingsdaten das Ergebnis von 16,0% Wortfehlerquote über den gesamten Switchboard Hub5'00 Benchmark erreicht werden, ohne dass Komponenten des Systems von Hand feingetunt werden mussten.

2.3.2 Funktionsweise und Besonderheiten der Architektur

Über die in Teilabschnitt 2.3.1 angesprochenen technischen Unterschiede hinaus gibt es noch weitere Besonderheiten der Funktionsweise von Deepspeech. In ihrem Artikel beschreiben Hannun et al. [10], dass sie beim Training der Rekurrenten neuronalen Netze einen Dropout[2] von 5% bis 10% verwendet haben. Allerdings nur auf den Feedforward Schichten und nicht auf den rekurrenten Schichten, damit die trainierten Netze nicht zu sehr auf die Trainingsdaten zugeschnitten sind, und um eine gewisse Varianz zu bieten [10]. Eine weitere Methode um dies beim Testen zu erreichen war es ein ganzes Ensemble an rekurrenten neuronale Netze parallel zu verwenden und eine Mittelwertbildung auf die einzelnen Ausgaben anzuwenden [10]. Alleine durch das Training mit großen Sprachdatensätzen kann das Modell für das rekurrente neuronale Netz bereits ein lesbares Transkript auf einzelner Buchstaben-Ebene produzieren [10]. In den meisten Fällen sind diese vorausgesagten Buchstabenfolgen, auch ohne externe Spracheinschränkungen, bereits die korrekt buchstabierten Wörter, falls die

[2] Ausschalten einer bestimmten Anzahl von Neuronen in jeder Schicht des Netzes

Voraussage falsch ist, was häufig nur bei Wörtern vorkommt welche kaum oder gar nicht im Trainingsset vorhanden sind, dann ist die Buchstaben-Folge zumindest eine phonetisch plausible Darstellung des Wortes [10]. Ein Beispiel aus dem Artikel findet sich in Tabelle 2.3 wieder. Dieses Problem ist in der Praxis schwer zu umgehen, da es schlichtweg unmöglich ist mit genug Daten zu trainieren, damit das Modell alle Wörter der Sprache oder alle Sprachkonstrukte schon einmal gehört hat [10]. Um das Problem dennoch zu umgehen wurde ein N-Gramm[3] Sprachmodell in Deepspeech integriert. Ein solcher N-Gramm Korpus wie Beispielsweise 2006 noch von Google auf CDs veröffentlicht, enthält alle aufeinander auftretenden Buchstabenfolgen beliebiger Länge einer Sprache und deren statistische Häufigkeiten des Auftretens. So ein N-Gramm-Modell wurde bei Deepspeech ebenfalls angewendet um Fehler in den Vorhersagen des rekurrenten neuronalen Netzes zu erkennen und zu korrigieren. Ergebnisse dieser Korrektur finden sich in der folgenden Tabelle 2.3.

Ausgabe des RNN	Korrigiertes Transkript
what is the weather like in bostin right now prime miniter nerenr modi arther n tickets for the game	what is the weather like in boston right now prime minister narendra modi are there any tickets for the game

Tabelle 2.3: Beispiele des Transkriptes, direkt durch das rekurrente neuronale Netz (links), mit Fehlern (markiert) die durch die Anwendung des N-Gramm-Modells behoben wurden (rechts) [10].

Die Ausgabe des korrigierten Transkriptes wurde dabei durch Kreuzvalidierungsverfahren der Vorhersagen des rekurrenten neuronalen Netzes und des N-Gramm-Modells über die wahrscheinlichste Buchstabenfolge berechnet [10]. Die Parameter des Kreuzvalidierungsverfahren wurden durch einen Strahlensuche-Algorithmus welcher von Maas et al. [22] entwickelt wurde optimiert.

2.3.3 Performanz im Switchboard Hub5'00 Benchmark

Der Switchboard Hub5'00 Benchmark gilt als einer der populärsten seiner Art in dem Bereich der Spracherkennung. Zudem ist er der größte Sprachkorpus für normale Konversationen, welche zudem auch kontinuierliche Sprache sind, und nicht wie eine Vielzahl Sprachkorpora diskrete Sprache. Das Korpus umfasst 2000 Stunden, welche sich in je 5 Minuten Gespräche zweier Fremder über ein vorher festgelegtes Thema aufgliedern [38]. Das Korpus ist des Weiteren ein repräsentatives Beispiel

[3] Zerlegung eines Wortes in Teilstücke der Länge n

für amerikanisches Englisch, da er in den Aspekten der Verteilung der Geschlechter, Herkunft und Migrationshintergrund, der einzelnen Sprecher und Sprecherinnen ausgeglichen ist [38]. Die Schwierigkeit ein gutes Ergebnis mit einer Spracherkennungssoftware in dem Benchmark zu erzielen liegt darin, dass es wie in echten Konversationen Versprecher, Wiederholungen und andere Beeinträchtigungen im Vergleich zu klar aufgenommenen einzelnen Wörtern gibt [38]. Da das Ziel dieser Arbeit ist, Gespräche in Meetings ausreichend gut zu transkribieren sind die Ergebnisse der zugrunde liegenden Architektur in diesem Benchmark also nicht zu vernachlässigen. Das Korpus wird bereits seit 25 Jahren studiert, unter anderem von Forschungsabteilungen von Firmen wie *AT&T* und *IBM* und Universitäten wie der Rheinisch-Westfälischen Technischen Hochschule Aachen, dem Karlsruher Institut für Technologie, und der University of Cambridge [38]. Abbildung 2.2 zeigt den Fortschritt im Switchboard Hub5'00 Benchmark seit 1993.

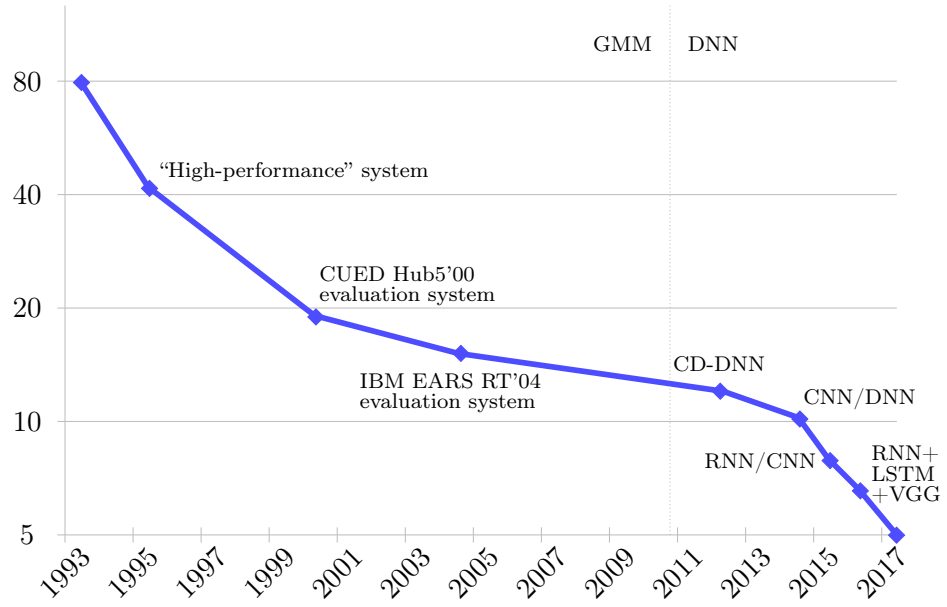

Abbildung 2.2: Fortschritt im Switchboard Hub5'00 Benchmark [38]. Angaben in Wortfehlerquote in Prozent.

Wie der Abbildung zu entnehmen ist, lag die beste Fehlerquote im Benchmark 1993 noch bei 80% während sie 2014, im Erscheinungsjahr von Baidus Deepspeech, nur noch bei knapp über 10% lag. Außerdem findet Ende 2010 die Transition von Mischverteilungsmodellen (GMM[4]) zu tiefen neuronalen

[4] engl. Gaussian Mixture Model

Netzen (DNN[5]) statt, wodurch die Fortschritte, wie sichtbar ist, wieder deutlich beschleunigt wurden.

2.3.4 Bedeutung der Ergebnisse

Damit liegt Baidus Deepspeech Architektur zum öffentlichen Erscheinungsdatum 2014 im Ergebnis des weithin anerkannten Switchboard Hub5'00 Benchmarks nur wenige Prozentpunkte über den bis dato besten Ergebnissen überhaupt. Dabei darf nicht vergessen werden, dass die anderen Projekte im Benchmark oft kommerziell sind und ihre Architektur nicht wie Baidu offen legen, sowie, dass seitdem Weiterentwicklungen in Deepspeech Implementierungen anderer Forscher und Entwickler stattgefunden haben (mehr dazu in Abschnitt 2.4). Angesichts dessen können Hannun et al. [10] in ihrem Artikel zurecht von *State of the Art* Spracherkennung sprechen. Das Zwischenfazit dieses Abschnitts ist, dass diese Architektur eine geeignete Basis für das Vorhaben dieser Arbeit bildet, und aufgrund ihrer Fähigkeiten und Besonderheiten zwischen vielen anderen Systemen besonders geeignet scheint um die zugrunde liegende Problemstellung zu lösen. Der folgende Abschnitt widmet sich dessen, eine geeignete Open Source Implementierung dieser Architektur für die zu entwickelnde Software zu finden.

2.4 Mozilla Common Voice und Deepspeech

Bei *Mozilla Research*, der Forschungsabteilung von Mozilla, wird sich ebenfalls darum bemüht, Spracherkennung voranzutreiben um Nutzern die Operation ihrer Geräte nur durch ihre Stimme zu ermöglichen und Nutzer die visuell oder physisch benachteiligt sind zu unterstützen [30]. Um nicht nur die eigenen Bemühungen in diesem Sektor voranzutreiben, sondern zum internationalen Fortschritt beizutragen möchte Mozilla Research zum einen frei verfügbare Sprachdatensätze zum Training von neuen Applikationen, sowie eine Open Source Sprachengine zur Spracherkennung und Sprachsynthese anbieten [30]. Letzteres hat Mozilla Research mit der eigenen Open Source Sprachengine Deepspeech erreicht welche in Teilabschnitt 2.4.3 genauer behandelt wird. Zunächst wird in den beiden Teilabschnitt 2.4.1 und 2.4.2 das Common Voice Projekt für freie Sprachdatensätze und der zugehörige deutsche Sprachdatensatz ausführlich behandelt.

2.4.1 Mozilla Common Voice Projekt

In vielen Sprachen außer Englisch und Mandarin ist es immer noch sehr schwer freie und gute Datensätze zum Training zur Spracherkennung für neuronale Netze, oder andere Sprachsysteme zu finden. Aus diesem Grund

[5] engl. Deep Neural Network

hat Mozilla im Jahr 2017 das Projekt *Common Voice* [26] ins Leben gerufen. Auf der Website des Projektes [26] hat jeder Internetbenutzer die Möglichkeit kurze vorgegebene Sätze aus einer Datenbank in jeder Sprache seiner Wahl über sein Mikrofon am Computer einzusprechen, oder Aufnahmen anderer Nutzer und zugehöriger Texte, ebenfalls in verschiedenen Sprachen, zu validieren. Zeichnet ein Nutzer eine solche kurze Sprachaufnahme eines Satzes aus der Datenbank auf, wird diese Sprachaufnahme in eine Warteschlange eingereiht, in welcher sie zunächst verbleibt. Validieren zwei verschiedene Benutzer, dass diese Sprachaufnahme mit dem angezeigten Satz aus der Datenbank übereinstimmt, so wird das Set aus Sprachaufnahme und Text in den Common-Voice-Datensatz der entsprechenden Sprache eingefügt. Wird die Sprachaufnahme hingegen von zwei verschiedenen Nutzern als nicht mit dem Text übereinstimmend markiert, wird die Sprachaufnahme aus der Warteschlange genommen und nicht in den Common-Voice-Datensatz der entsprechenden Sprache eingefügt [26]. In allen anderen Fällen verbleibt die Aufnahme in der Warteschlange zur Evaluation durch andere Benutzer. Durch dieses einsenden von Aufnahmen für den Datensatz mit Einverständnis der Nutzer und der Validierung anderer Aufnahmen ebenfalls durch die Nutzer selbst, ermöglicht das Common Voice Projekt in vielen Sprachen frei zugängliche und ausreichend große Datensätze anzubieten. Der Aufwand der Forscher bei Mozilla Research ist dabei minimal, da sie nicht wie bei herkömmlichen Verfahren Sprecher hunderte Stunden Texte einsprechen und die validieren müssen. Auf diese Art und Weise sind bereits von Nutzern aus aller Welt 9100 Stunden Sprachaufnahmen aufgezeichnet und 7200 Stunden Sprachaufnahmen in 54 verschiedenen Sprachen validiert worden[6]. Darauf entfallen auf die deutsche Sprache 823 Stunden Sprachaufnahmen von denen 765 Stunden validiert sind[6] [26]. Durch die Aufnahmen einzelner Nutzer gibt es außerdem eine hohe Diversität, im deutschen Common-Voice-Datensatz sind es 11.731 verschiedene Sprecher[6] was im Training eines neuronalen Netzes den Vorteil bringen kann, dass dieses nicht zu sehr auf die Stimmen einzelner Sprecher angelernt wird (Machine Learning Bias), und somit die von neuronalen Netzen gewünschte Generalisierung bringt. Ein ähnlicher Effekt tritt dadurch auf, dass die Nutzer alle ihre eigenen Mikrofone benutzen und die Aufnahmen alle ein unterschiedliches Level an Klarheit haben, und auch durchaus Hintergrund- oder Störgeräusche enthalten können. Das ist, auch wenn es sich zunächst wie ein Nachteil anhört, für die Generalisierung des zu trainierenden Netzes tatsächlich ein Vorteil. Die in dieser Arbeit zu entwickelnde Software soll langfristig auch auf verschiedenen Systemen mit unterschiedlicher Hardware eingesetzt werden, von daher ist ein neuronales Netz welches mit Aufnahmen von möglichst vielen verschiedenen Mikrofonen, Sprechern und Umgebungen trainiert wurde für unsere Zwecke ideal.

[6] Abfragen getätigt am 07.12.2020

2.4.2 Deutscher Common-Voice-Datensatz

Mit den in Teilabschnitt 2.4.1 angesprochenen 765 validierten Stunden Sprachaufnahmen von 11.731 verschiedenen Sprechern und den dazugehörigen Transkripten gehört der deutsche Common-Voice-Datensatz zu den größten deutschen Sprachkorpora für Audio-Dateien. Viele der Zusammengehörigen Audio- und Transkript-Dateien enthalten zudem von Mozilla veröffentlichte demografische Informationen zu den Sprechern wie Alter, Geschlecht und Akzent. Für den gesamten deutschen Datensatz gibt Mozilla selbst beispielsweise die folgenden, in den Abbildungen 2.3, 2.4 und 2.5 gezeigten, Unterteilungen an [26, 27]. Abbildung 2.3 zeigt, dass der Großteil des Datensatzes aus Aufnahmen in akzentfreiem Deutsch besteht und der Anteil an Schweizerdeutsch und österreichischem Deutsch vernachlässigbar ist. Die in Abbildung 2.4 veranschaulichte Verteilung der Altersgruppen der Sprecher ist hingegen ungefähr ausgeglichen zwischen den Altersspannen 19 bis 29, 30 bis 39, 40 bis 49 und 50 bis 59, nur die Altersspannen der unter 19-Jährigen und 60 bis 69-Jährigen sind unterrepräsentiert, aber untereinander wieder ausgeglichen. Die Geschlechterverteilung der Sprecher in Abbildung 2.5 ist wiederum sehr unausgeglichen, da der Großteil der Sprecher männlich ist. Dies ist allerdings zu vernachlässigen, da unsere Software in Meetings von Softwareentwicklungsteams eingesetzt werden soll, und dort eine ähnliche Geschlechterverteilung wie in Abbildung 2.5 zu erwarten ist [31]. Insgesamt entsprechen die demografischen Daten des Datensatzes denen, die auch für das Umfeld der zu entwickelnden Software angenommen werden können. Dies macht den deutschen Common-Voice-Datensatz geeignet für die zu entwickelnde Software, umso mehr, wenn der Mangel an alternativen, kostenfreien, großen, deutschen Sprachdatensätzen beachtet wird.

Abbildung 2.3: Unterteilung der Akzente der Sprecher im deutschen Common-Voice-Datensatz in Prozent [27].

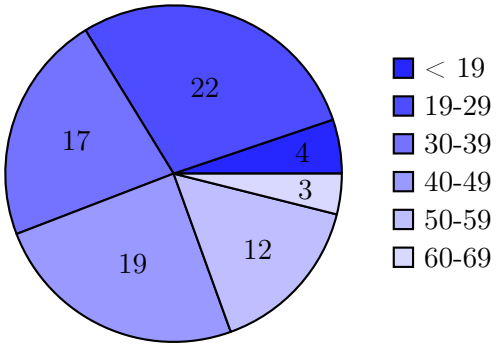

Abbildung 2.4: Aufteilung der Altersgruppen der Sprecher im deutschen Common-Voice-Datensatz in Prozent [27].

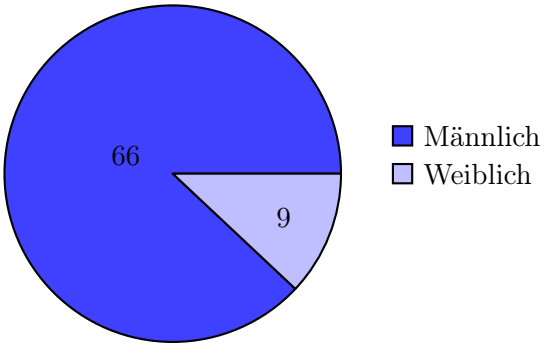

Abbildung 2.5: Geschlechterverteilung der Sprecher im deutschen Common-Voice-Datensatz in Prozent [27].

2.4.3 Mozilla Deepspeech Projekt

Die Forscher bei Mozilla Research machten sich die von Hannun et al. [10] veröffentliche Deepspeech Architektur, welche in Abschnitt 2.3 ausführlich behandelt wurde zunutze, um diese als Grundstruktur für ihre eigene Open Source Sprachengine Deepspeech zu nutzen. Für die Implementierung der Architektur wurde das vom *Google Brain Team* 2017 entwickelte und ebenfalls unter Open-Source-Lizenz veröffentliche TensorFlow [7] Framework, welches häufig Anwendung im Bereich des maschinellen Lernens findet genutzt. TensorFlow eignet sich auch besonders um die von Hannun et al. [10] beschriebenen rekurrenten neuronalen Netze der Deepspeech Architektur zu implementieren, was genauso bei Mozilla Research gemacht wurde. Das Deepspeech Projekt veröffentlichten die Forscher 2018 auf GitHub [28], wo es seitdem ständig weiterentwickelt wird. Zu dem eigentlichen Quellcode werden neben der Dokumentation, wie die Sprachengine angesteuert wird, unter den Releases auf GitHub [28] auch bereits vortrainierte Sprachmodelle für Deepspeech auf Englisch und Mandarin angeboten. Für das englische Sprachmodell gibt Mozilla selbst dabei für Release 0.8.2 eine Wortfehlerquote (WER) von 5,97% auf dem 1000 Stunden umfassenden *LibriSpeech Clean Test Corpus* [34] an [28]. Für die menschliche Wortfehlerquote werden oft 4% aus einem Artikel von Lippmann [21] zitiert, welche also um ungefähr 2 Prozentpunkte niedriger als die von Deepspeech wäre. In einem Artikel von Stolcke und Droppo [42] wird jedoch angemerkt, dass dabei lediglich personelle Kommunikation ohne weitere Angaben referenziert wird. Stolcke und Droppo [42] messen in ihrem Ausführlich beschrieben Verfahren selbst eine Wortfehlerquote von ungefähr 5,9% für menschliche Hörer und liegen damit nur um 0,07 Prozentpunkte unter der von Mozilla angegebenen Wortfehlerquote von 5,97% für Deepspeech. Dies demonstriert die Mächtigkeit der Deepspeech Sprachengine. Neben den auf der GitHub Seite des Deepspeech Projektes [28] von Mozilla angebotenen Sprachmodellen, können Nutzer auch selber aus eigenen Sprachdatensätzen, wie denen in Teilabschnitt 2.4.1 und 2.4.2 beschriebenen Common-Voice-Datensätzen, Sprachmodelle für Deepspeech generieren und trainieren. Wie dies genau funktioniert wird in der Dokumentation von Deepspeech welche auf der GitHub Seite des Projektes [28] verlinkt ist ausführlich beschrieben. Dabei können ausreichend trainierte Sprachmodelle wie die von Mozilla selbst veröffentlichten Modelle im .pbmm Format exportiert werden, welches so direkt von Deepspeech genutzt werden kann, indem der Pfad zu der .pbmm Datei beim Aufruf von Deepspeech auf der Kommandozeile mit angegeben wird. Alternativ besteht auch die Möglichkeit das Sprachmodell als .tflite Datei zu exportieren, welche konvertiert wird, um TensorFlow Lite zu nutzen, um so auch auf Systemen mit wenigen Ressourcen gute Ergebnisse zu erzielen. Neben dem Austausch der vortrainierten Sprachmodelle ist es auch möglich den sogenannten Scorer auszutauschen der ebenfalls aus einer

einzigen .scorer Datei besteht, welche bei der Nutzung von Deepspeech als optionales Kommandozeilen Argument mit angegeben werden kann. Dieser Scorer enthält das im Teilabschnitt 2.3.2 beschriebene N-Gramm-Modell der Architektur, und wird genutzt um die generierten Transkripte durch eine zusätzliche Vorhersage von wahrscheinlichen Buchstabenfolgen der Sprache zu verbessern. Die Nutzung des Scorers ist zwar optional, verbessert die Genauigkeit der Ergebnisse aber immens.

2.4.4 Deutsche Sprachmodelle und Scorer

Durch die einfache Austauschbarkeit der Sprachmodelle lässt Deepspeech sich schnell auf verschiedene Bedürfnisse der Nutzer anpassen, und kann so in verschiedenen Bereichen und Sprachen eingesetzt werden. Daher finden sich auf Open Source Plattformen wie GitHub und GitLab viele weitere Projekte in denen die Deepspeech Sprachengine genutzt wird um andere Sprachen als Englisch und Mandarin zu erkennen. So veröffentlichten auch Agarwal und Zesch [2] einen Artikel, in welchem sie ihr vorgehen beschreiben, aus deutschen Sprachdatensätzen die für Deepspeech nötigen Sprachmodelle und Scorer zu trainieren. Ihre finalen Ergebnisse des Artikels veröffentlichten sie ebenfalls als Open-Source-Projekt auf GitHub [1, 2]. Der Artikel wird später in Kapitel 7, Abschnitt 7.1 noch genauer betrachtet. Ein weiteres großes Open-Source-Projekt ist unter dem Namen *Deepspeech-Polyglot* auf GitLab [33] zu finden. Dieses Projekt beschäftigt sich damit Sprachmodelle und Scorer für Deepspeech in anderen Sprachen zur Verfügung zu stellen und umfasst momentan die Sprachen Deutsch, Spanisch, Französisch, Italienisch und Polnisch. Die deutschen Datensätze des Modells haben mit einer Wortfehlerquote von 12,8% die niedrigste WER für ein deutsches Deepspeech Modell, und untertreffen damit sogar die Ergebnisse von Agarwal und Zesch [2]. Das Deepspeech-Polyglot Projekt wird ebenfalls in Kapitel 7, Abschnitt 7.1 noch ausführlicher behandelt. Daher werden für diese Arbeit ebenfalls die Sprachmodelle und Scorer des Deepspeech-Polyglot Projektes genutzt, da in diese bereits unzählige Stunden an Training investiert wurden, was im Rahmen dieser Arbeit nicht möglich gewesen wäre.

Kapitel 3

Konzept

Dieses Konzept dient dazu, es zu ermöglichen in Meetings von Entwicklungsteams, Aussagen automatisch zu transkribieren und anschließend mithilfe des Klassifikationsalgorithmus [11, 24] Vorhersagen über die Polarität (positiv, negativ oder neutral), der einzelnen Aussagen zu treffen. Dieser Vorgang soll idealerweise ohne Unterbrechung, also in einem einzelnen Programmablauf geschehen, sowie möglichst simpel für den Anwender gestaltet sein. In diesem Kapitel wird der entwickelte konzeptionelle Ablauf von Verarbeitungsschritten, welche dafür nötig sind, detailliert vorgestellt. Dabei werden sowohl Resultate, als auch Quellcode von anderen Abschlussarbeiten am Fachgebiet Software Engineering der Gottfried Wilhelm Leibniz Universität Hannover verwendet. Dazu gehört insbesondere der Klassifikationsalgorithmus von Horstmann [11] und Meyer [24] sowie die Resultate von Klünder et al. [13, 14, 16]. Die folgende Abbildung 3.1 gibt einen Überblick über die einzelnen Schritte, welche die zu entwickelnde Software für die gewünschten Resultate durchführen muss. Da jeder Teilschritt zudem in einem einzelnen Abschnitt nochmal detailliert erläutert wird, dient die Abbildung zudem als Übersicht für dieses Kapitel.

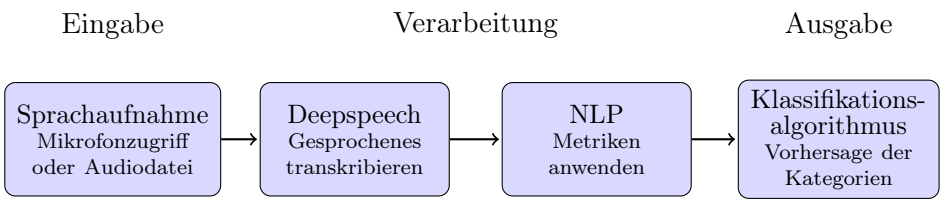

Abbildung 3.1: In einzelne Verarbeitungsschritte aufgegliederter konzeptioneller Ablauf der zu entwickelnden Software.

3.1 Idee

Als Eingabe der zu entwickelnden Software wird die Sprache, beziehungsweise der Ton aus den Meetings von Entwicklungsteams verwendet. Dieser soll in Echtzeit als Eingabe für die Software dienen, sprich es soll dem Nutzer später möglich sein die Software, beispielsweise auf einem Laptop, der sich in der Mitte des Konferenzraums befindet, zu starten und dann das Meeting wie gewohnt durchzuführen. Anschließend soll die Aufzeichnung über eine Tastenkombination abgebrochen werden können, womit das aufgezeichnete Transkript gespeichert wird und die Weiterverarbeitung durch *Natural Language Processing* (NLP) beginnt. Wurden die aufgezeichneten Aussagen verarbeitet, so sollen sie dem Klassifikationsalgorithmus als Eingabe dienen, welcher dann für die einzelnen Aussagen eine der drei Kategorien positiv, negativ oder neutral vorhersagt. In den folgenden Abschnitten 3.2 bis 3.5 werden die einzelnen Schritte ausführlicher, aber dennoch konzeptionell beschrieben. Die konkrete technische Implementierung der einzelnen Schritte wird im Anschluss in Kapitel 4 ausformuliert.

3.2 Sprachaufnahme

Um den Ton, welcher als Eingangssignal vom Mikrofon des Rechners, auf dem die zu entwickelnde Software ausgeführt wird, übertragen wird, zu verarbeiten muss dieser zunächst, in ein für einen Computer verständliches Format, konvertiert werden. Dies geschieht, indem das Eingangssignal, also die Audiowellen, welche einen zeitlichen Verlauf einer Frequenz der Schwingung repräsentieren, periodisch abgetastet werden um eine zeitdiskrete Information abspeichern zu können. Dieses periodische Abtasten der Werte wird auch *Sampling* genannt. Die Samplingrate gibt dabei die Frequenz, in welcher das Eingangssignal abgetastet wird, an. Bei jedem Abtastvorgang wird der Wert des Eingangssignals an die nächste ganze Zahl angenähert und in Binärdarstellung gespeichert. So entsteht eine digitale Repräsentation des Tons, die anschließend weiterverarbeitet werden kann. Abbildung 3.2 stell ein Beispiel für ein zeitkontinuierliches Eingangssignal dar.

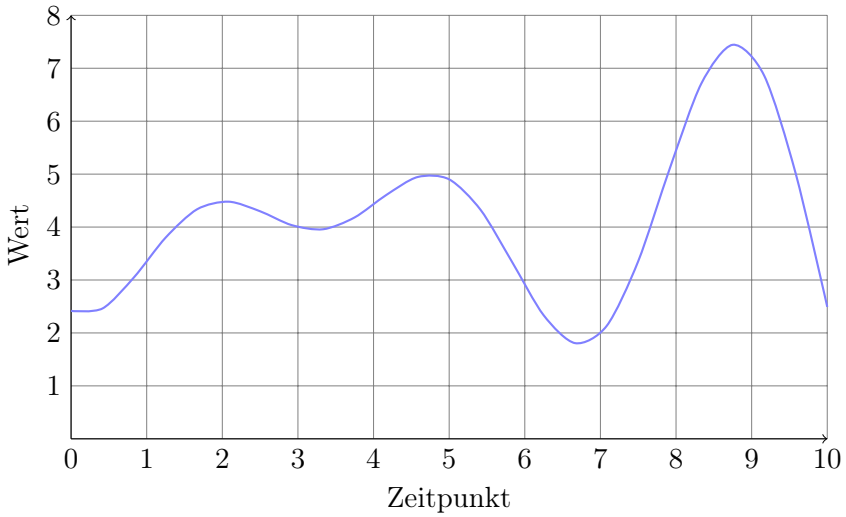

Abbildung 3.2: Beispielhafter Verlauf eines zeitkontinuierlichen Eingangssignals.

Wie zu sehen ist, kann eine solche Wellenform nicht trivial formatiert und für die Weiterverarbeitung abgespeichert werden. Dazu bedarf es einem Sampling wie in Abbildung 3.3 gezeigt. Die dunkelblaue Linie zeigt die angenäherte und vereinfachte Darstellung des Eingangssignals (hellblau). Tabelle 3.1 enthält die angenäherten ganzzahligen Werte zu jedem Zeitpunkt, sowie die Binärcodierung in der sie abgespeichert wurden.

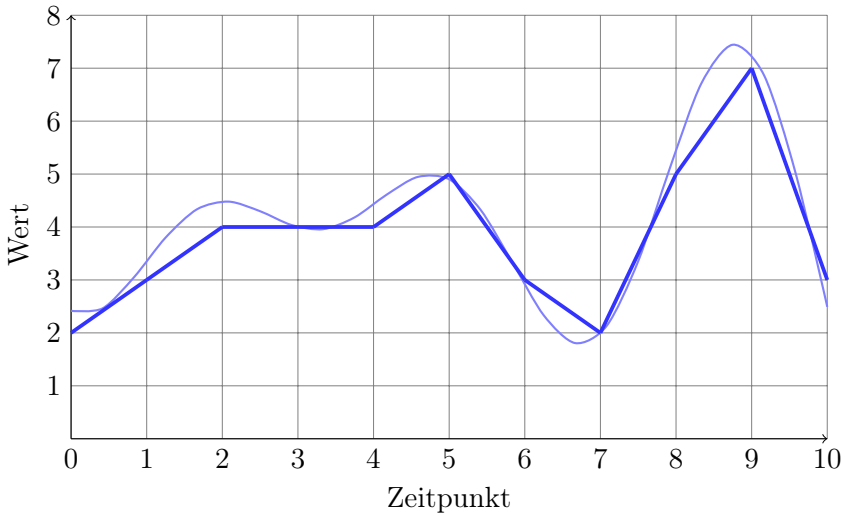

Abbildung 3.3: Sampling des Signals aus Abbildung 3.2.

Zeitpunkt	1	2	3	4	5	6	7	8	9	10
Annäherung	3	4	4	4	5	3	2	5	7	3
Binär	011	100	100	100	101	011	010	101	111	011

Tabelle 3.1: Angenäherten ganzzahlige Werte und Binär Codierung zu Abbildung 3.3.

Bei der Rekonstruktion des Tons werden die gesampleten Werte einfach mit den Werten des nächsten Zeitpunktes linear verbunden, weil keine Aussage über die Werte für die dazwischen liegenden Zeitpunkte getroffen werden kann. Daher entsteht der kantige Verlauf des gesampleten Signals in Abbildung 3.3. Wie außerdem zu sehen ist, beträgt die Samplingrate in dem Beispiel 1 da pro Zeiteinheit einmal abgetastet wird. Je höher die Abtastrate ist, desto niedriger ist der Datenverlust beim Sampling. Für Deepspeech wird Audio mit einer Abtastrate von 16.000 Hz benötigt. Alternativ können auch direkt bereits abgespeicherte Audiodateien im Waveform (WAV) Format welche mit 16.000 Hz codiert wurden als Eingabe für die Sprachengine genutzt werden. Dieses Feature wird in der zu entwickelnden Software aber nur optional genutzt werden, da das Hauptaugenmerk auf der Verarbeitung der Daten in Echtzeit über das Mikrofon liegt.

3.3 Transkribieren

Um diese, im ersten Schritt durch das Mikrofon des Nutzers aufgenommenen und für den Computer formatierten, Audiosignale in Text umzuwandeln soll die in Kapitel 2, Teilabschnitt 2.4.3 ausführlich behandelte Sprachengine Deepspeech von Mozilla zum Einsatz kommen. Die Gründe dafür wurden bereits offengelegt. Die Umwandlung des Eingangssignals in formatierte Waveform-Dateien soll zwischenzeitlich nach einer bestimmen Zeitspanne geschehen. Dazu soll nach jedem Ende dieser Zeitspanne ein Frame erstellt werden, der alle Informationen des Eingangssignals ab dem Zeitpunkt des letzten gespeicherten Frames enthält. Um festzulegen, wann eine solche Zeitspanne endet, sollen Pausen beim Sprechen erkannt werden. Ein Frame soll also jeweils den Beginn des Sprechens bis zur nächsten Redepause abdecken. Auf diese Art und Weise wird das Gespräch im Idealfall automatisch in Sätze beziehungsweise Aussagen aufgegliedert. Wurde ein Frame erkannt soll dieser wie im vorherigen Abschnitt 3.2 beschrieben in das Waveform Format mit einer Abtastrate von 16.000 Hz zwischengespeichert, und als Eingabe für Deepspeech genutzt werden. Dadurch kann der von Deepspeech erkannte Text eines Frames in der zu entwickelnden Software zwischengespeichert werden. Wenn der Nutzer die Sprachaufnahme über das Mikrofon durch eine Tastenkombination beendet hat, sollen die einzelnen Frames, durch

Zeilenumbrüche getrennt, in einer Textdatei gespeichert werden. Diese Textdatei repräsentiert das Transkript des Meetings. Die Abspeicherung erfolgt bewusst vor den folgenden Verarbeitungsschritten, welche sowohl Rechen-, als auch Zeitintensiver sind, um in jedem Fall ein Backup des Transkriptes zu generieren. Die einzelnen Frames bleiben jedoch für die Weiterverarbeitung noch in einer geeigneten Datenstruktur gespeichert, und werden nicht neu aus der Textdatei geladen.

3.4 Natural Language Processing

Zu diesem Zeitpunkt liegen die einzelnen transkribierten Aussagen bereits in einer geeigneten Datenstruktur gespeichert vor, und es kann direkt auf diese zugegriffen werden. Selbige können nun als Eingabe für das Natural Language Processing (NLP) genutzt werden. NLP wird hier nicht weiter erklärt, stattdessen wird auf Horstmann [11], Abschnitt 2.3 verwiesen. Für die Gewinnung der inhaltlichen und statischen Metriken wird, wie für den Klassifikationsalgorithmus [11, 24] ebenfalls auf die Erkenntnisse und den bereits vorhanden Quellcode von Horstmann [11] zurückgegriffen. Da jedoch nur gesprochener Text analysiert werden soll, und keine Textnachrichten, werden die Emoticons aus den statistischen Metriken entfernt, da diese für das Vorhaben überflüssig sind. Andere statistische Metriken wie die Wortlänge und Häufigkeit einzelner Wörter, die auch in dem transkribierten Text einfach ermittelt werden können, sollen weiterhin erfasst und ausgewertet werden können. Die inhaltlichen Metriken werden weiterhin verwendet, um Beziehungen zwischen Wörtern herzustellen und ein tieferes Verständnis über einzelne Aussagen zu erlangen. Anschließend sollen die Aussagen zusammen mit den extrahierten Metriken in einem geeigneten Format abgespeichert werden. Um den Evolutionären Algorithmus (EA) [11] anlernen zu können soll es die Möglichkeit geben den einzelnen Aussagen manuell Trainingslabel hinzuzufügen, dies ist nötig damit der EA ein Modell lernen kann, welches später für die automatische Klassifikation genutzt werden kann. Falls der Nutzer sich dazu entscheidet, sollen die eingegebenen Trainingslabel ebenfalls zusammen mit den Aussagen und Metriken in einem geeigneten Format abgespeichert werden.

3.5 Klassifikationsalgorithmus

Wie Eingangs erwähnt spielt der Klassifikationsalgorithmus [11, 24] eine zentrale Rolle für das Vorhaben dieser Arbeit. Entscheidet sich der Nutzer im vorherigen Schritt dazu, Trainingslabel zu den Daten hinzuzufügen, so wird der Evolutionäre Algorithmus (EA) [11] diese zuerst zum Lernen und Erstellen eines Modells nutzen. Die genauen Lernverfahren des EA werden in dieser Arbeit nicht behandelt und sind Horstmann [11] zu entnehmen. Das

trainierte Modell kann dann später zusammen als Eingabe mit Aussagen, welche nur die extrahierten Metriken und keine manuell eingetragenen Label besitzen, genutzt werden, um für diese Aussagen aufgrund des erlernten Modells eine Vorhersage zu treffen. Der Klassifikationsalgorithmus kann des Weiteren die Güte der gesamten Konversation mittels Durchschnitt und Standardabweichung bestimmen. Es ist zu erwarten, dass die Ergebnisse je akkurater sind, desto mehr Trainingsdaten dem EA vorliegen. Um gute Ergebnisse zu erzielen ist es also Eingangs nötig einen hohen manuellen Aufwand zu betreiben, um einen großen Trainingsdatensatz mit Labeln zu versehen. Dazu folgt in Kapitel 5 ein ausführlicheres Beispiel anhand eines echten Meetings eines Entwicklungsteams, inklusive dem Hinzufügen von Trainingslabeln und anlernen des EA.

Kapitel 4

Umsetzung

Dieses Kapitel behandelt ähnliche Unterpunkte wie bereits im Konzept in Kapitel 3 angesprochen, geht jedoch wesentlich tiefer in die Tatsächliche technische Implementation der einzelnen Komponenten der entwickelten Software ein, und behandelt diese nicht nur auf konzeptioneller Ebene. Der Leser soll in diesem Kapitel ein Verständnis über die Funktionsweise einzelner Arbeitsschritte der entwickelten Software erlangen. Dafür empfiehlt sich Erfahrung im Bereich der Datenverarbeitung von Sprache mit der Programmiersprache Python, Wissen über die Funktionsweise des Klassifikationsalgorithmus [11, 24], und insbesondere dem Evolutionären Algorithmus (EA) von Horstmann [11]. Da die Funktionsweise der Deepspeech Architektur und der zugehörigen Sprachengine von Mozilla eingangs zu genüge erklärt wurde, wird dieses Wissen ebenfalls im Folgenden vorausgesetzt. Zunächst werden in Abschnitt 4.1 die verwendete Programmiersprache und die verwendeten Programmbibliotheken erläutert und warum diese ausgewählt wurden, bevor in Abschnitt 4.2 der Vorgang des Mirkofonzugriff und transkribieren beschrieben wird. In Abschnitt 4.3 wird das Extrahieren der benötigten Metriken per Natural Language Processing (NLP) erläutert bevor zum Abschluss in Abschnitt 4.4 die Anwendung des Klassifikationsalgorithmus [11, 24] auf die Daten beschrieben wird.

4.1 Programmiersprache

4.1.1 Python

Für die entwickelte Software wurde die Programmiersprache Python in der Version 3.8.5 verwendet. Python wird für die Verwendung von Deepspeech in einer Version ab 3.8 benötigt, damit Deepspeech ordnungsgemäß funktioniert. Die Nutzung von Python erleichterte ebenfalls die Anbindung des Klassifikationsalgorithmus [11, 24], da dieser in Python Version 3.7.3 verfasst wurde, und sich damit direkt für die entwickelte Software anpassen und in sie einbinden ließ. Auch das von Mozilla Research entwickelte Beispielprogramm

Microphone VAD Streaming [29] auf welchem die entwickelte Software basiert, wurde in Python verfasst.

4.1.2 Packages

Um die Funktionsweise der entwickelten Software zu ermöglichen sind einige Python-Packages notwendig. Diese lassen sich wie in der *README.md* Datei im Quellverzeichnis beschrieben über den Python Package-Installer *pip* installieren. Im Folgenden sind alle Packages aufgelistet, die für die Inbetriebnahme der Software notwendig sind, mit einer kurzen Erläuterung der Funktionen beziehungsweise der Notwendigkeit für die entwickelte Software zu jedem einzelnen Package.

- **deepspeech**: Wird für den Zugriff auf die Sprachengine und zum transkribieren benötigt.
- **PyAudio**: Bietet Zugriff auf die PortAudio Bibliothek zur Audioaufnahme auf verschiedenen Plattformen.
- **webrtcvad**: Klassifiziert ob beim Mikrofonzugriff gerade gesprochen wird oder nicht, basiert auf dem Google *WebRTC* Projekt.
- **scipy**: Wird zur Abtastratenkonvertierung[1] verwendet.
- **numpy**: Bietet eine einfache Handhabung von Vektoren, Matrizen und Arrays in Python.
- **pandas**: Verarbeitet große Datenmengen in sogenannten DataFrames, also Tabellen ähnlichen Datenstrukturen.
- **tabulate**: Ermöglicht die Ausgabe von formatierten Tabellen aus Pandas DateFrames.
- **pyspellchecker**: Kontrolliert die Rechtschreibung und unterstützt die deutsche Sprache.
- **stop_words**: Liefert eine Liste geläufiger Stopwörter in verschiedenen Sprachen.
- **spacy**: Führt NLP und insbesondere Tokenisierung[2] in verschiedenen Sprachen, unter anderem auch Deutsch durch.
- **nltk**: Steht für *Natural Language Toolkit* und führt ebenfalls NLP durch.

[1] engl. Resampling
[2] Segmentierung eines Textes in einzelne Wort-Tokens

- **scikit_learn**: Bietet eine Vielzahl an Funktionen für maschinelles Lernen in Python, insbesondere verschiedene Klassifikationsalgorithmen.
- **matplotlib**: Ermöglicht Graphen zu zeichnen, um beim trainieren des EA den Fortschritt zwischen den Generationen anzuzeigen.
- **halo**: Bietet die Möglichkeit dem Nutzer über sogenannte Spinner in der Kommandozeile anzuzeigen, dass das Programm gerade arbeitet.
- **termcolor**: Ermöglicht es auf einfache Art und Weise farblichen Text auf der Kommandozeile auszugeben.

4.2 Mikrofonzugriff und Transkription

Um die im Konzept in Kapitel 3, Abschnitt 3.2 beschriebene Funktionalität zu erhalten ist zunächst ein Zugriff auf das Mikrofon über Python, der auf verschiedenen Plattformen funktioniert, sowie eine Sprechpausenerkennung (VAD[3]) nötig. Mozilla Research stellt neben Deepspeech auf einer eigenen GitHub Seite namens *Deepspeech Examples* Beispiele für die Anwendung von Deepspeech für verschiedene Zwecke und in verschiedenen Umgebungen zur Verfügung. Eines dieser Beispiele, welches die gewünschte Funktionalität liefert, ist *Microphone VAD Streaming* [29]. Das Programm greift beim Start automatisch auf das Mikrofon des Nutzers zu und gibt gesprochenen Text nach einer Redepause auf der Kommandozeile aus. Ermöglicht wird dieser Zugriff auf das Mikrofon über das Package *PyAudio* welches eine Python Implementation des Open-Source-Projektes *PortAudio* bietet. PortAudio ermöglicht den Zugriff auf Audioaufnahme und Wiedergabe auf einer Vielzahl von verschiedenen Plattformen, wie Windows und Linux. PyAudio liefert dann einen Audiostream vom Mikrofon des Anwenders in einem Format, dass sich in Python verarbeiten lässt. Die Aktivitätserkennung beim Sprechen wird durch das Package *webrtcvad*, welches auf dem Google *WebRTC* Projekt basiert gewährleistet. Diese Aktivitätserkennung findet oft Anwendung in der Spracherkennung und Googles VAD gilt als die beste frei verfügbare Aktivitätserkennung. Die Funktionen die dieses Beispiel von Mozilla bietet eignen sich damit ideal für das Projekt. Daher diente Microphone VAD Streaming auch als Grundlage der Entwicklung, mit dem Ziel das Beispiel den Vorgaben dieser Arbeit entsprechend abzuwandeln und den Klassifikationsalgorithmus [11, 24] an Microphone VAD Streaming anzubinden. Zur Abwandlung wurden zunächst die in Kapitel 2, Abschnitt 2.4.4 behandelten deutschen Sprachmodelle und Scorer für Deepspeech aus dem Deepspeech-Polyglot Projekt verwendet, da Microphone VAD Streaming standardmäßig nur auf Englisch funktioniert. Nach dieser Anpassung ist bereits eine deutsche Spracherkennung mit Ausgabe der erkannten Texte auf

[3] engl. Voice Activity Detection

der Kommandozeile möglich. Um die Nutzerfreundlichkeit weiter zu steigern wurde das Programm so abgeändert, dass die Sprachmodelle und Scorer über ein festes Unterverzeichnis und nicht über die Kommandozeile aufgerufen werden. Außerdem wurde die komplette Ausgabe auf der Kommandozeile minimiert, auf Deutsch übersetzt und farblich hervorgehoben. Da das Programm standardmäßig abstürzte, wenn kein Mikrofon angeschlossen war, wurde zudem eine Aussagekräftige Fehlermeldung mit dem Beenden des Programmes eingefügt. Bei der VAD wurde des Weiteren eingefügt, dass erkannte Aussagen nicht nur auf der Kommandozeile ausgegeben werden, sondern auch in Echtzeit in einem Array sowie einer Textdatei im Programmverzeichnis abgespeichert werden, um die Daten vor möglichen Abstürzen des Systems zu schützen. Die Aufnahme wird vom Nutzer, wie auch standardmäßig, über eine Tastenkombination beendet, nur dass sich das Programm dann nicht mehr beendet, sondern die in den folgenden Abschnitten 4.3 und 4.4 beschriebenen Verarbeitungsschritte eingeleitet werden.

4.3 Extrahieren von Metriken

Um die transkribierten Daten weiterzuverarbeiten werden diese zunächst aus dem gespeicherten Array in einen Pandas DataFrame geladen. Von dort aus werden die Daten durch eine abgewandelte Version des *DataPreProcessor* von Horstmann [11] verarbeitet. Die Abwandlungen beinhalten zum Beispiel die Entfernung der Extraktion von Merkmalen über Emojis, da diese selbstverständlich nur in der von Horstmann untersuchten schriftlichen Kommunikation im Arbeitsumfeld auftauchen, aber nicht in gesprochener Sprache. Alle anderen Metriken werden jedoch weiterhin aus den Daten extrahiert und den einzelnen Aussagen in dem DataFrame hinzugefügt. Für die genaue Funktionsweise dieses Prozesses wird auf Horstmann [11], Abschnitt 5.3 und folgende verwiesen. Anschließend werden der DataFrame mitsamt der extrahierten Metriken als CSV-Datei und die zugehörigen TF-IDF Daten abgespeichert. Diese Dateien werden dann im nächsten Abschnitt 4.4 aufgerufen, um die Vorhersagen über das Sentiment der einzelnen Aussagen vorzunehmen.

4.4 Anwendung des Klassifikationsalgorithmus

Im Finalen Schritt des Programms wird der EA von Horstmann [11] aufgerufen. Dazu wurde die *Evaluation* Datei von Horstmann abgewandelt, da diese ursprünglich Daten mit Zeitstempel erwartet hat, um daraus ein gesamt Sentiment für jeden Arbeitstag innerhalb der Firma zu bestimmen. Die Datei wurde so weit umgewandelt, dass sie in stattdessen auch in *Prediction* umbenannt wurde, da sie jetzt genutzt wird um die Vorhersagen

für einzelne Aussagen und nicht die Gesamtheit zu treffen. Dafür werden die Daten mit dem besten Hyperparameterset aus dem Training zusammen als Eingabe für den EA verwendet. Der Quellcode von Horstmann [11] wurde dabei so erweitert, dass das beste Hyperparameterset beim Training des EA automatisch als binäres Array abgespeichert, und von der Prediction Datei geladen wird bevor der *CACDTClassifier* aufgerufen wird. Dafür muss im entsprechenden Verzeichnis im Pfad des Programms ein trainiertes Modell für den EA vorliegen, welches durch Eingabe von Trainingsdaten mit manuell versehenen Labeln generiert werden kann (mehr dazu in Kapitel 5, Abschnitt 5.2). Sind diese Voraussetzungen gegeben, so gibt der CACDT-Classifier eine Liste mit den zugehörigen Vorhersagen zu den eingegebenen Aussagen zurück. Das Ergebnis wird dann als neuer DataFrame welcher nur die Aussagen und zugehörigen Vorhersagen beinhaltet gespeichert, der als CSV- und Text-Datei im Ausgabe Verzeichnis abgespeichert wird. So kann der Nutzer später die Vorhersagen des Sentiments zu einzelnen Aussagen betrachten und die CSV-Datei in Zukunft auch wieder weiterverarbeiten, da sie sich einfach zurück in einen Pandas DataFrame laden lässt. Des Weiteren wird der DataFrame mit dem Package *tabulate* als formatierte Tabelle auf der Kommandozeile ausgegeben, wofür jedoch für sehr lange Aussagen nur der Beginn angezeigt wird. Im Anschluss wird noch eine Zählung der positiv, negativ und neutral klassifizierten Aussagen, sowohl absolut als auch relativ zu der Gesamtzahl der Aussagen vorgenommen, welche ebenfalls als CSV-Datei abgespeichert, und als formatierte Tabelle auf der Kommandozeile ausgegeben wird. So kann der Nutzer direkt im Anschluss an ein Gespräch eine erste Vorhersage über dessen Verlauf erhalten, indem er die Anzahl und Anteile der positiven, negativen, und neutralen Aussagen vergleicht.

Kapitel 5

Evaluation

In diesem Kapitel wird die entwickelte Software anhand von echten Datensätzen aus Meetings von Entwicklungsteams evaluiert. Zunächst werden in Abschnitt 5.1 die Quelle der Daten sowie der Vorgang von deren Erhebung beschrieben, bevor in Abschnitt 5.2 der Prozess des Trainings des Evolutionären Algorithmus (EA) [11] mit den Datensätzen beschrieben wird. Anschließend wird in Abschnitt 5.3 die Aufarbeitung der Testdaten, beschrieben, welche nötig ist, damit die Spracherkennung möglichst präzise darauf arbeiten kann. Zum Schluss werden die Ergebnisse der Evaluation in Abschnitt 5.4 betrachtet, wobei versucht wird erste Schlüsse aus den Ergebnissen zu ziehen.

5.1 Verwendete Trainings- und Testdatensätze

Das Fachgebiet Software Engineering der Gottfried Wilhelm Leibniz Universität Hannover führt jährlich im Wintersemester das sogenannte *Softwareprojekt* (SWP) durch [15]. In dieser Veranstaltung haben Studierende die Möglichkeit in kleinen Entwicklungsteams für einen echten Kunden Software zu entwickeln um Erfahrung mit den Abläufen innerhalb des Softwareentwicklungsprozesses für das spätere Berufsleben zu erlangen [15]. Im Rahmen dieser Veranstaltung wurden von einem solchen Entwicklungsteam aus Studierenden der Gottfried Wilhelm Leibniz Universität Hannover ein internes wöchentliches Meeting Gespräch zu ihrem Projekt aufgezeichnet. Dafür haben alle Teilnehmer des Gesprächs eine Einverständniserklärung zur Teilnahme im Rahmen einer wissenschaftlichen Studie unterschrieben, um eine ordnungsgemäße Verwendung der Daten sicherzustellen. Da das Meeting aufgrund der Sars-CoV2 Pandemie nur digital über den online Kommunikationsdienst *Discord* abgehalten werden konnte, wurde dieses Meeting nicht über ein Mikrofon, sondern über einen Bot auf dem Discord Server des Teams aufgezeichnet. Dies hatte jedoch immerhin den Vorteil, dass zu jedem Teammitglied eine eigene Tonspur

aufgezeichnet wurde, was sich in der späteren Aufarbeitung der Testdaten für die Sprachengine in Abschnitt 5.3 als wertvoll herausstellte. Dieses eigens für diese Arbeit aufgezeichnete Gespräch hatte eine Dauer von circa 33 Minuten. Zum Ende des Gespräches wurden die Teilnehmer zu ihrer Empfindung des Sentiments der Kommunikation im gesamten Gespräch befragt. Des Weiteren wurden zwei weitere Gespräche von studentischen Entwicklungsteams aus dem SWP im Wintersemester 2015 mit jeweils einer Länge von 39 beziehungsweise 47 Minuten vom Fachgebiet Software Engineering für diese Arbeit zur Verfügung gestellt. Diese wiesen jedoch eine deutlich niedrigere Audioqualität auf, sodass sich selbst die manuelle Transkription als schwierig herausstellte. Daher wurden nur aus einem der beiden Gespräche die verständlichen Aussagen in den Trainingsdatensatz aufgenommen. Alle Aussagen wurden von Hand transkribiert und nach eigener Empfindung mit einem Sentiment Label *$positiv*, *$negativ* oder *$neutral* am Ende jeder Zeile beziehungsweise Aussage versehen. Die nachfolgende Tabelle 5.1 veranschaulicht zu jeder der drei Sentiment Klassen eine Beispielaussage aus dem Trainingsdatensatz. Die positive Aussage wurde der Klasse hinzugefügt, weil sie Lob enthält, und die negative Aussage wurde der Klasse hinzugefügt, da sie Kritik enthält. Für Aussagen an denen weder etwas positives noch negatives festgemacht werden konnte wurde die neutrale Klasse gewählt.

Klasse	Beispielaussage
positiv	das ist wirklich ein super gutes feature
neutral	das koennen wir jetzt eigentlich auch schon machen langsam
negativ	das ist aber nicht sehr benutzerfreundlich muss ich sagen

Tabelle 5.1: Beispiele zu den einzelnen Klassen aus dem Trainingsdatensatz.

Der gesamte Inhalt wurde anschließend für die einfache Weiterverarbeitung in einer Textdatei im Verzeichnis der entwickelten Software abgespeichert. Der Trainingsdatensatz besteht aus insgesamt 712 Aussagen, welche die Folgende in Abbildung 5.1 veranschaulichte Verteilung der einzelnen Klassen positiv, negativ und neutral aufweisen. Es ist zu sehen, dass die Klasse neutral mit einem Anteil von 77,5% deutlich überrepräsentiert ist, was aber auch von Aussagen in einem professionellen Entwicklungsumfeld zu erwarten ist. Die Klassen positiv und negativ sind mit 10,8% beziehungsweise 11,7% ungefähr ausgeglichen. Insgesamt sind 712 Aussagen jedoch verhältnismäßig wenig. Der Trainingsdatensatz den Horstmann [11] selbst nutzte umfasste Beispielsweise 3778 Nachrichten. Die Audiodateien manuell zu transkribieren und mit Labeln zu versehen ist allerdings auch deutlich aufwendiger, als

einen Datensatz von Textnachrichten manuell mit Labeln zu versehen.

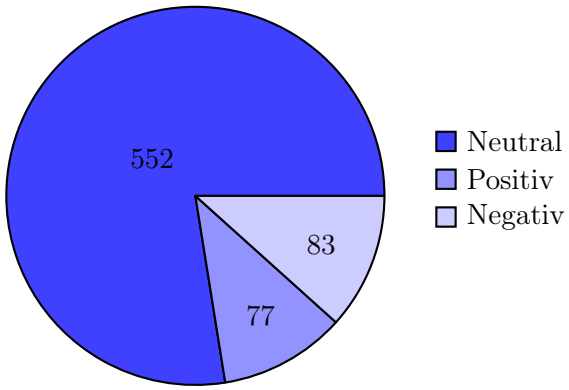

Abbildung 5.1: Aufteilung der Sentimentklassen der insgesamt 712 Aussagen im Trainingsdatensatz.

5.2 Training des Klassifikationsalgorithmus

Um den händisch erstellten Datensatz auf den EA [11] anzuwenden müssen zuallererst einige Verarbeitungsschritte durchgeführt werden. Der Datensatz wurde zunächst, um spätere Komplikationen bei der Verarbeitung der Daten in Python zu vermeiden, und bessere Ergebnisse in Anbetracht des Trainings für später von Deepspeech erstellte Transkripte zu erhalten, angepasst. Dafür wurde in der Textdatei zum einen jeder Großbuchstabe durch seinen äquivalenten Kleinbuchstaben ersetzt, sowie die folgenden Ersetzungen deutscher Umlaute und Sonderzeichen vorgenommen.

- Ä → **Ae**, ä → **ae**
- Ü → **Ue**, ü → **ue**
- Ö → **Oe**, ö → **oe**
- ß → **ss**

Dies entspricht zum einen ebenfalls der Art und Weise wie Deepspeech später deutsche Transkripte für die Weiterverarbeitung durch den EA [11] erzeugt, und verhindert zum anderen mögliche Konvertierungs- oder Codec-Fehler in Python. Auf Satzzeichen wurde schon beim transkribieren gänzlich verzichtet, auch da diese durch den EA [11] sowieso nicht beachtet werden. Entspricht die Textdatei mit den Trainingsdaten allen oben genannten Kriterien und ist entsprechend formatiert, dass in jeder Zeile eine Aussage und ein manuelles Trainingslabel getrennt durch das Sonderzeichen „$“ vorliegen, kann sie durch das ebenfalls im Rahmen dieser Arbeit angefertigte

Python-Skript *Training* weiterverarbeitet werden. Dieses Skript liest dabei die Textdatei mit den Trainingsdaten aus dem entsprechenden Verzeichnis ein und geht die Datei Zeile für Zeile durch. Dabei wird jede Zeile am Sonderzeichen „$“ getrennt, und der vordere Teil in eine Liste von Aussagen gespeichert. Der hintere Teil wird von überschüssigen Leerzeichen und dem Zeilenumbruch bereinigt und in einer Liste der zugehörigen Label gespeichert. Die Liste der Aussagen wird anschließen als Spalte in einen neuen Pandas DataFrame eingefügt und durch den *DataPreProcessor* von Horstmann [11] verarbeitet. Dadurch werden dem DataFrame wie bereits in Kapitel 4 Abschnitt 4.3 beschrieben, die zugehörigen Metriken die aus jeder Aussage extrahiert werden angefügt. Anschließend wird dem DataFrame eine letzte Spalte mit den zugehörigen Trainingslabeln zu jeder Aussagen angefügt, dann wird der DataFrame als CSV-Datei abgespeichert. Diese Datei dient anschließend als Eingabe für die Trainingsfunktion des EA von Horstmann [11]. An der Trainingsfunktion wurden nur kleinere Anpassungen vorgenommen, wie das automatische Abspeichern des besten Hyperparametersets aus dem Training und eine nutzerfreundlichere Ausgabe auf der Konsole. Der EA versucht dann durch die Aus- und Abwahl einzelner Metriken und dem Testen von Zusammenhängen zwischen diesen und den Trainingslabeln, mithilfe der Scikit-learn Klassifikationsalgorithmen, ein geeignetes Modell zu erlernen. Dabei wird immer eine zufällige Teilmenge der Trainingsdaten als Testdaten und der Rest als Trainingsdaten verwendet. Über mehrere Generationen kann sich so die sogenannte Fitness, also die Akkuratheit der Vorhersagen verbessern. Für eine detailreichere Beschreibung des Vorgangs wird auf Horstmann [11], Abschnitt 6.4 verwiesen. Am Ende des Trainingsvorgangs wird das beste Modell ausgewählt und im entsprechenden Pfad abgespeichert. Dieses wird dann später für die Vorhersage zu eingesprochenen Aussagen verwendet.

5.3 Aufbereitung der Testdaten

Als Testdatei wurde ein Teil aus dem in Abschnitt 5.1 beschriebenen Meeting welches digital über die Plattform Discord abgehalten wurde ausgewählt, da es die beste Qualität aufwies, aufgrund dessen, dass jeder Teilnehmer mit einem eigenen Mikrofon aufgezeichnet wurde. Des Weiteren bot die digitale Aufzeichnungsform den Vorteil, dass zu jedem Teilnehmer eine eigene Audiospur aufgezeichnet werden konnte, auf welcher nur jener Teilnehmer zu hören war. Dadurch wurde eine Aufbereitung ermöglicht. Die einzelnen Spuren wurden zunächst mithilfe der freien Audiosoftware *Audacity* [23] in ein mehrspuriges Projekt importiert. Anschließend wurden zunächst Störgeräusche beseitigt. Dazu gehörten Beispielsweise Interferenzen und andere Störgeräusche im Hintergrund der Teilnehmer. Außerdem wurde auch in Momenten wo zwei Teilnehmer gleichzeitig anfingen zu reden, der Ton des

Teilnehmers der sich unterbrechen ließ entfernt, sodass zu jedem Zeitpunkt nur ein Teilnehmer zu hören war. Zum Schluss wurde der Ausschnitt auf eine Länge von 10 Minuten zurechtgeschnitten da dies eine Vorgabe für die Testdaten dieser Arbeit war. Die Spuren der einzelnen Teilnehmer wurde mit einem Kompressor Effekt versehen, um den Dynamikumfang des Signals zu Beschränken und anschließend normalisiert. Dadurch hatten die Teilnehmer alle eine vergleichbare Lautstärke. Das Projekt wurde im Waveform (WAV) Format mit einer Abtastrate von 16.000 Hz und als Mono-Audio (Einspurig) abgespeichert, um den Vorgaben von Deepspeech zu entsprechen.

5.4 Ergebnisse

Anhand der Verteilung des Trainingsdatensatzes wie in Abbildung 5.1 gezeigt, wäre eine Genauigkeit von 77,5% alleine dadurch möglich, jede Aussage als neutral zu klassifizieren. Der Klassifikationsalgorithmus [11, 24] erreichte auf dem 712 Aussagen umfassenden Trainingsdatensatz jedoch eine Genauigkeit von 81,8%. Damit wurde ein Lerneffekt für das Modell festgestellt. Die im Rahmen dieser Arbeit entwickelte Software extrahierte aus der 10 Minuten Test-Audio-Datei insgesamt 140 verschiedene Aussagen. Die Aufteilung der dazugehörigen Vorhersagen ist in der folgenden Abbildung 5.2 dargestellt. Es wurden 15 Aussagen als positiv klassifiziert, was 10.7% entspricht. 124 Aussagen wurden als neutral klassifiziert was 88,6% entspricht und lediglich eine Aussage als negativ und damit nur ein Anteil von 0,7%.

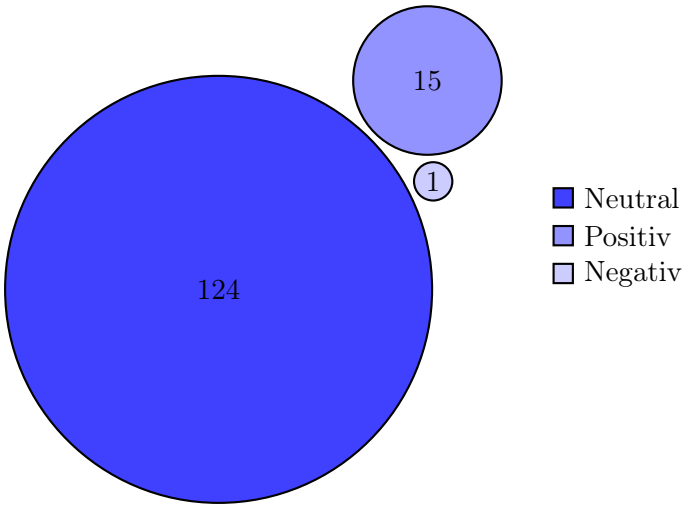

Abbildung 5.2: Aufteilung der Sentimentklassen der 140 durch die Software transkribierten und klassifizierten Aussagen.

Damit weicht die Verteilung der einzelnen Klassen von den Trainingsdaten deutlich ab. Die Klasse neutral ist stärker überrepräsentiert als im Trainingsdatensatz und die Klassen positiv und negativ sind nicht mehr ausgeglichen, sondern sehr unausgeglichen. Der Anteil der Klasse positiv ist um 10 Prozentpunkte höher als derer der Klasse negativ. Dieses Ergebnis stimmt jedoch mit der Einstufung der Teilnehmer des Gespräches überein. Diese stuften das Gesprächsklima nach Ende des Gespräches als neutral bis positiv ein. Die Spracherkennung hatte mit dem schnellen Redetempo und der teilweise undeutlichen Aussprache einiger Teilnehmer Schwierigkeiten. In der folgenden Tabelle 5.2 ist der Beginn des Gespräches jeweils mit den eigentlich Aussagen der Teilnehmer und den transkribierten Aussagen der Software, sowie die Abweichungen beider veranschaulicht. Es ist zu sehen, dass es teilweise zu Abweichungen kommt durch transkribierte Wörter, die einen ähnlichen Klang wie das ursprünglich Gesagte haben. Bei zu schneller oder undeutlicher Sprache weicht das Transkript aber auch teilweise so stark von der ursprünglichen Aussage ab, dass kein inhaltlicher Zusammenhang mehr besteht.

Ursprüngliche Aussage	Transkribierte Aussage
Kommt drauf an, wenn du einstellen möchtest wie sie sich nur bei dir bewegen, ja dann würde das gehen.	kommt drauf an wenn du einstellen moechtest wie sie sich nur bei dir bewegen ja dann wuerde das gehen
Achja.	ach ja
Und wenn ich möchte dass die sich bei allen so bewegen?	und wenn ich moechte dass sie sich beeilen zu bewegen
Dann müssen wir das in den Code schreiben.	dann muessen wir das in den code schreiben
Das heißt aber dann wär's nicht variabel.	zwei hundert erstaunlich variabel

Tabelle 5.2: Vergleich zwischen ursprünglichen und transkribierten Aussagen, fehlerhafte Stellen wurden markiert.

Um die Güte der Ergebnisse gegenzuprüfen wurden 50 von der Software transkribierte Aussagen noch einmal manuell in die Klassen positiv, negativ und neutral eingeordnet, die manuellen Label wurden anschließend mit denen des Klassifikationsalgorithmus [11, 24] verglichen. Die Aussagen wurden dabei nicht zufällig ausgewählt, stattdessen wurden aus dem Transkript der Software jene Aussagen ausgewählt, die am meisten mit dem ursprünglich gesagten übereinstimmten, damit eine sinnvolle manuelle Klassifizierung der

Aussagen vorgenommen werden konnte. Die Anteile der von der Software ausgewählten Klassen und den manuell ausgewählten Klassen wird in der Tabelle 5.3 verglichen.

Bewertungen	**Positiv**	**Neutral**	**Negativ**
Software	10 (20%)	39 (78%)	1 (2%)
Manuell	5 (10%)	45 (90%)	0 (0%)

Tabelle 5.3: Vergleich der Klassifizierung der 50 ausgewählten Aussagen durch die Software sowie manuell.

Die ausgewählten Aussagen weisen für die Software einen niedrigeren Anteil der Klasse neutral, sowie höhere Anteile der Klassen positiv und negativ auf, als die 140 insgesamt klassifizierten Aussagen (Abbildung 5.2). Manuell wurden jedoch nur halb so viele Aussagen als positiv und keine Aussage als negativ klassifiziert. Für die statistische Auswertung wurde das Fleiss' Kappa Verfahren [6] angewendet. Aufgrund dessen, dass viele übereinstimmende Klassifikationen in der Klasse neutral vorlagen, ergab sich ein P_e-Wert von 0,7282. Dies entspricht einer Wahrscheinlichkeit zufälliger Übereinstimmung von 72,8% was sehr hoch ist. Insgesamt ergab sich ein κ-Wert von gerundet 0,56. Dieser κ-Wert liegt nach Landis und Koch [18] im oberen Bereich der moderaten Übereinstimmung, wie Tabelle 5.4 veranschaulicht. Ab einem κ-Wert von 0,61 kann schon von einer erheblichen Übereinstimmung gesprochen werden [18].

κ-Wert	Maß der Übereinstimmung
$< 0{,}00$	Schlecht
0,00 - 0,20	Niedrig
0,21 - 0,40	Einigermaßen
0,41 - 0,60	**Moderat**
0,61 - 0,80	Erheblich
0,81 - 1,00	Fast perfekt

Tabelle 5.4: Maß der Übereinstimmung zu zugehörigen κ-Werten aus dem Fleiss' Kappa Verfahren [6], entnommen aus Landis und Koch [18].

Kapitel 6

Diskussion

Für diese Arbeit wurde ein Verfahren entwickelt mit welchem sich Aussagen aus Audiodateien oder über das Mikrofon in Echtzeit aufgezeichnetem Ton von Meeting Gesprächen, mithilfe des Klassifikationsalgorithmus [11, 24], automatisiert in die Klassen positiv, neutral und negativ klassifizieren lassen. Die Anwendbarkeit des Verfahrens wurde durch die entwickelte Software, welche an einem echten Gespräch aus dem Meeting eines studentischen Entwicklungsteams getestet wurde, demonstriert. Um die Güte der Software und des Klassifikationsalgorithmus [11, 24] für diesen Anwendungsfall zu bewerten wurde zu Beginn dieser Arbeit die folgende zentrale Forschungsfrage formuliert.

Zentrale Forschungsfrage *Wie gut kann die zu entwickelnde Software Gespräche in Meetings transkribieren, und die einzelnen Aussagen in die Kategorien positiv, negativ und neutral einstufen, um aussagekräftige Ergebnisse zu erhalten?*

Aufgrund der im vorangegangenen Abschnitt 5.4 präsentierten Ergebnisse und Berechnungen durch das Fleiss' Kappa Verfahren [6], sowie deren Bedeutung, welche durch Landis und Koch [18] gegeben wurde, lässt sich die zentrale Forschungsfrage wie folgt beantworten.

Beantwortung der zentralen Forschungsfrage *Es wurde gezeigt, dass die entwickelte Software die Aussagen aus den Meetings eines Entwicklungsteams im Vergleich zu einer manuellen Klassifikation mit einer moderaten Übereinstimmung automatisiert klassifizieren kann.*

6.1 Mögliche Einflussfaktoren der Ergebnisse

Bei dem gewählten Fleiss' Kappa Verfahren [6] wird der κ-Wert umso kleiner, desto größer der P_e-Wert, also die Wahrscheinlichkeit zufälliger Übereinstimmung ist. Dadurch dass in den Testdaten ein Großteil der

Aussagen sowohl von der entwickelten Software als auch manuell als neutral klassifiziert wurde, ergab sich eine Wahrscheinlichkeit zufälliger Übereinstimmung von 72,8%, dies führte wiederum zu einem κ-Wert von 0,56 trotz dessen, dass 44 der insgesamt 50 Aussagen von der Software identisch klassifiziert wurden wie manuell. Dies entspricht einer Genauigkeit von 88%.

Bereits Horstmann [11] stellte in seiner Arbeit fest, dass selbst die manuelle Klassifizierung von Aussagen für einen einzelnen Menschen eine Herausforderung darstellt, da jeder ein eigenes empfinden über das Sentiment einer Aussage hat. Auch die im Rahmen dieser Arbeit klassifizierten Test- und Trainingsdatensätze wurden lediglich von einer einzelnen Person klassifiziert. Die Größe des Trainingsdatensatzes für den evolutionären Algorithmus (EA) [11] wurde durch den Zeitaufwand für das manuelle transkribieren und klassifizieren der Aussagen für eine einzelne Person limitiert. Ebenso wurde beim Klassifizieren der Aussagen für die Trainingsdaten chronologisch im Gesprächsverlauf vorgegangen, sodass Klassifikationen anhand des Kontextes vorheriger Aussagen getroffen wurden, der EA besitzt jedoch kein Konzept eines Kontextes, es werden immer nur einzelne Aussagen betrachtet [11].

Das Verfahren ist noch nicht Domänenspezifisch, beziehungsweise ist die Domäne momentan nur durch die Trainingsdaten gegeben, da diese aus Meetings von Entwicklungsteams entnommen wurden. Das Verfahren von Horstmann wurde bereits von Meyer [24] optimiert, diese Optimierungen konnten jedoch im Rahmen dieser Arbeit nicht verwendet werden, da sie auf englischer Sprache basierten, und Tools wie *Senti4SD* [4] nutzen, mit denen sich das Sentiment einer Aussage speziell im Kontext der Softwareentwicklung bestimmen ließ. Da solche Tools im Deutschen noch nicht existieren wurde auf die allgemeinen Sentimentlexika von Waltinger [46] und Remus et al. [37] zurückgegriffen welche auch von Horstmann [11] für seine Arbeit verwendet worden waren.

Da Meetings aufgrund der aktuellen Sars-CoV2 Pandemie nur digital abgehalten werden können, hatte dies auch Einfluss auf diese Arbeit. So entstand das aufgezeichnete Gespräch welches sowohl zum Training als auch zum anschließenden Test der Software genutzt wurde auf der Kommunikationsplattform Discord. Dies hatte zwar den Vorteil, dass jeder Teilnehmer ein eigenes Mikrofon besaß, jedoch auch einige Nachteile wie, die schlechte Audioqualität einzelner Teilnehmer und die auftretenden Störgeräusche entweder im Hintergrund einzelner Teilnehmer, oder durch Interferenzen der Mikrofone mit anderen Geräten. Durch die Verzögerungen kam es auch oft dazu, dass Teilnehmer gleichzeitig begannen zu sprechen oder sich gegenseitig zu unterbrechen.

6.2 Optimierungsmöglichkeiten

Um ein noch aussagekräftigeres Ergebnis und einen möglicherweise höheren κ-Wert zu erhalten müssten die Klassen der Aussagen in den Testdaten ein ausgeglicheneres Verhältnis der drei Klassen positiv, negativ und neutral besitzen. Wären diese gleichmäßig in drittel aufgeteilt würde die Wahrscheinlichkeit zufälliger Übereinstimmung nur noch bei rund 33,3% liegen, anstatt bei 72,8%. Damit wäre es dann bei einem guten Ergebnis möglich einen deutlich höheren κ-Wert zu erreichen, welcher dann auch nach Landis und Koch [18] eine erhebliche oder sogar fast perfekte Übereinstimmung ergeben könnte.

Ein größerer Trainingsdatensatz für den EA wäre essenziell um ein weniger von den einzelnen Aussagen der Trainingsdaten beeinflusstes Modell für den Klassifikationsalgorithmus [11, 24] zu erhalten (Machine Learning Bias). Dadurch könnte potenziell die Anzahl fehlerhafter Klassifikationen auf unbekannten Daten gesenkt werden. Ebenso sollten sowohl Trainings- als auch Testdaten von mehr als einer Person manuell klassifiziert werden, dann könnte jeweils die mehrheitliche Klasse gewählt werden, um den Einfluss eigener Empfindungen einzelner Personen zu senken. Eine Erweiterung des Klassifikationsalgorithmus [11, 24] um das Konzept eines Kontextes von Aussagen würde sehr wahrscheinlich zu einer Verbesserung der Vorhersagen führen, da die Emotionalität einer Nachricht oftmals erst durch den Kontext gegeben ist. Im Verlauf des Gesprächs folgten positive oder negative Aussagen auch oftmals aufeinander anstatt einzeln aufzutreten. Dazu müsste der EA nicht nur die Aussagen im Einzelnen, sondern auch die vorangegangenen Aussagen betrachten.

Um die Domänenspezifität der Software zu verbessern, müsste, bei dessen erscheinen, ein deutsches Tool oder Sentimentlexikon welches speziell für die Softwareentwicklung entwickelt wurde eingebunden werden, beziehungsweise ein solches Tool oder Sentimentlexikon selbst entwickelt werden, was aber mit erheblichem Aufwand verbunden wäre. Wenn anschließen ähnliche Metriken wie die durch Senti4SD [4] bei Meyer [24] gegebenen vorliegen könnte versucht werden die Optimierungen des Klassifikationsalgorithmus [11, 24] von Meyer [24] auch für die deutsche Sprache anzuwenden. Damit wäre die Software dann enger auf die Domäne der Entwicklungsteams zugeschnitten als bisher.

Damit die Spracherkennung der Software bestmöglich funktioniert ist es nötig, dass immer nur ein Teilnehmer gleichzeitig spricht, dies ist in einem echten Meeting von Angesicht zu Angesicht in der Regel gegeben. Es wäre außerdem ideal, wenn jeder Teilnehmer über ein eigenes Großmembran-Kondensatormikrofon verfügen würde, um der Software eine bestmögli-

che Audioqualität zur Verfügung zu stellen. Die verwendete Sprachengine Deepspeech befindet sich zudem in ständiger Weiterentwicklung, sodass mit zukünftigen Versionen die Qualität der Transkripte weiter verbessert werden kann. Auch der Austausch der verwendeten deutschen Sprachmodelle oder das Training von eigenen Sprachmodellen für Deepspech kann zu Verbesserungen führen, da für diese Arbeit nur die zum Zeitpunkt besten frei verfügbaren deutschen Sprachmodelle verwendet wurden.

6.3 Threats to Validity

Die zuvor in Abschnitt 6.1 beschriebenen Einflussfaktoren legen den Schluss nahe, dass es Einschränkungen für die Validität und Generalisierbarkeit der Ereignisse der durchgeführten Studie gibt. Diese werden im Folgenden den einzelnen *Threats to Validity* wie *Conclusion Validity*, *Construct Validity*, *Internal Validity* und *External Validity* zugeordnet.

6.3.1 Conclusion Validity

Da das im Rahmen der Studie aufgezeichneten Meeting aufgrund der Sars-CoV2 Pandemie über die Plattform Discord abgehalten werden musste, verfügte nicht jeder der Teilnehmer während des Gespräches über eine ruhige Umgebung wie sie in einem Besprechungsraum gegeben ist. Die Ablenkung der Teilnehmer durch Hintergrundgeräusche in Ihrer Umgebung oder der anderer Teilnehmer, sowie durch andere Störgeräusche die durch die digitale Kommunikation auftraten, ist daher nicht ausschließbar.

Der hohe manuelle Aufwand die Gespräche manuell zu transkribieren und mit Trainingslabeln zu versehen resultierte in einem verhältnismäßig kleinen Trainingsdatensatz von 722 Aussagen. Auch die Testdaten wurden auf eine 10 minütige Audiodatei aus dem Gespräch beschränkt.

6.3.2 Construct Validity

Der Aspekt *mono-operation-bias* trifft auf die Studie möglicherweise zu, da nur ein einzelner 10 minütiger Ausschnitt aus dem Meetinggespräch als Testaudio für die Analyse und Berechnung der Ergebnisse genutzt wurde. Dem Bias wird jedoch durch die zusätzliche manuelle Auswertung der Aussagen im Rahmen des Fleiss' Kappa Verfahren [6] entgegengewirkt. Auch wurde die entwickelte Software nur anhand eines trainierten Modells aus dem Trainingsdatensatz und nur im finalen Entwicklungsstand evaluiert.

Dadurch, dass die Teilnehmer ihr Einverständnis zur Aufzeichnung des Meetings zwecks der Studie abgeben mussten, und die Aufnahme im Gespräch auf Discord deutlich gekennzeichnet war, waren sich die Teilnehmer

zu jedem Zeitpunkt darüber bewusst, dass sie aufgenommen wurden. Dadurch änderte sich wahrscheinlich das Verhalten der meisten Teilnehmer, beziehungsweise es wurde womöglich zusätzlicher Druck auf die Teilnehmer ausgeübt. Ein Teilnehmer sprach dies sogar direkt im aufgezeichneten Gespräch an.

6.3.3 Internal Validity

Die Testdaten wurden aus einem Meeting eines einzelnen studentischen Entwicklungsteam entnommen, welches aus sechs Mitgliedern bestand. Vier der Teammitglieder kannten sich bereits vor Beginn des Projektes und befanden sich bereits in einem freundschaftlichen Verhältnis untereinander, während das bei den anderen zwei Teammitgliedern nicht der Fall war. Auch der Erfahrungsstand der Teilnehmer Unterschied sich stark, so hatten einzelne Teilnehmer bereits berufliche Erfahrung im Bereich der professionellen Softwareentwicklung, andere jedoch nicht. Dies muss bei der Betrachtung der Ergebnisse beachtet werden.

6.3.4 External Validity

Die Anwendbarkeit des Verfahrens wurde anhand der entwickelten Software, welche anhand eines online Meetings eines studentischen Entwicklungsteams getestet wurde, demonstriert. Die Ergebnisse können jedoch nicht generell auch für andere Meetings angenommen werden. Außerdem soll das Verfahren später auch Anwendung bei Softwareentwicklern und Projektleitern von Entwicklungsteams mit deutlich mehr Erfahrung finden. Für den späteren Einsatz des Verfahrens sollten die Vorkenntnisse jedoch keine wichtige Bedeutung haben, zumal die Anwender die Software immer mit eigenen Daten aus ihren Projekten trainieren können.

Kapitel 7

Verwandte Arbeiten

In diesem Kapitel werden wissenschaftliche Arbeiten und Publikationen welche dieselben oder sehr ähnliche Unterpunkte wie diese Arbeit behandeln aufgezählt. Dazu folgt am Ende eines jeden Abschnittes eine kurze Erläuterung welche die Unterschiede der vorgestellten Arbeit oder Publikation zu dieser Arbeit aufzeigt. Der erste Abschnitt 7.1 behandelt dabei die deutsche Spracherkennung mithilfe von Mozilla Deepspeech, während der folgende Abschnitt 7.2 sich mit anderen Arbeiten und Publikationen beschäftigt die ebenfalls eine Analyse von Interaktionen in der Umgebung von Softwareentwicklungsteams behandeln. Im Anschluss wird zu Ende dieses Kapitels in Abschnitt 7.3 noch einmal auf die Abgrenzung dieser Arbeit als gesamtes eingegangen.

7.1 Deutsche Spracherkennung unter Anwendung von Deepspeech

Agarwal und Zesch [2] vom Language Technology Lab der Universität Duisburg-Essen haben sich ebenfalls damit befasst, sich Mozillas Deepspeech zunutze zu machen, um deutsche Sprache zu erkennen. Dabei gingen sie darauf ein, dass es für andere Sprachen als Englisch kaum frei zugängliche kostenlose Lösungen zur Spracherkennung gibt. Das Einzige vortrainierte deutsche Sprachmodell ist das von Milde und Köhn [25]. Ihre Hauptmotivation war, dass für die Sprachengine Deepspeech noch kein vortrainiertes deutsches Sprachmodell existiert [2]. Dies stellt ihrer Meinung nach ein ernsthaftes Hindernis für die angewandte Forschung auf deutschen Sprachdaten dar, da vor allem Dienste von Google, Microsoft oder Amazon Datenschutz bedenken mit sich bringen [2]. Zum Trainieren ihres eigenen Modells nutzen sie öffentlich zugängliche Sprachkorpora. Neben dem deutschen Common Voice Korpus von Mozilla [27] nutzten sie dafür auch den TUDA-De [25, 36, 44] Korpus der und den ebenfalls deutschen Sprachkorpus Voxforge [25, 36, 43], der Technischen Universität

Darmstadt. Ihr bestes Modell erreichte dabei laut eigenen Angaben eine WER von 15,1%. Ihre Ergebnisse veröffentlichten sie als Open-Source-Projekt auf GitHub [1, 2]. Auf dem Artikel von Agarwal und Zesch [2] basiert auch ein weiteres Open-Source-Projekt welches unter dem Namen *Deepspeech-Polyglot* auf GitLab [33] zu finden ist, dieses stellt fertig trainierte Modelle für Deepspeech in verschiedenen Sprachen zur Verfügung, für das deutsche Modell wurden dabei weitaus mehr Trainingsdaten verwendet, da verschiedene Quellen wie zum Beispiel YouTube-Videos von öffentlich rechtlichen Sendern mit einbezogen wurden. Die trainierten Modelle des Deepspeech-Polyglot Projektes wurden auch für diese Arbeit verwendet, da sie mit einer aktuellen WER von 12,8% die niedrigste WER für trainierte deutsche Open-Source-Deepspeech-Modelle zur Spracherkennung bieten. Die vorgestellten Projekte behandeln den Prozess ein Modell für die Sprachenginge Deepspeech mit geeigneten deutschen Trainingsdaten zu trainieren, motiviert durch den Mangel an Alternativen. Durch die Resultate dieser Projekte konnte ein entscheidender Teil dieser Arbeit, nämlich die deutsche Spracherkennung realisiert werden. Ein Training eines eigenen Sprachmodells im Rahmen dieser Arbeit wäre nicht möglich gewesen, da beispielsweise Agarwal und Zesch [2] selbst auf einem sehr leistungsstarken Server immer noch eine hohe Laufzeit für jedes Training hatten. Darüber hinaus stellt die deutsche Spracherkennung nur einen von mehreren Verarbeitungsschritten der im Rahmen dieser Arbeit entwickelten Software dar.

7.2 Interaktionsanalyse in Entwicklungsteams

Klünder et al. [16] entwickelten im Rahmen der Forschung am Fachgebiet Software Engineering der Gottfried Wilhelm Leibniz Universität Hannover das Verfahren Act4teams-SHORT [16], welches auf dem Verfahren Act4teams von Kauffeld [12] basiert. Diese Verfahren wurden entwickelt um Probleme in Meetings von Entwicklungsteams analytisch aufzudecken und somit die Effektivität der Meetings zukünftig zu erhöhen, beziehungsweise das Klima im Team zu verbessern. Dafür wurde bereits von Haak [9] im Rahmen ihrer Masterarbeit ein Tutorial Konzept für die Anlernung der händischen Durchführung von Act4teams-SHORT [9, 16] durch einen menschlichen Beobachter im Meeting entwickelt. Das langfristige Bestreben dieses Verfahren in Zukunft voll automatisiert ablaufen zu lassen war die Motivation für diese Arbeit. Diese Arbeit ist aber insgesamt nur als ein Teilschritt zum Erreichen eben dieses langfristigen Bestrebens zu sehen, und stellt keine vollständige Lösung des Problems dar. Die im Rahmen dieser Arbeit entwickelte Software soll stattdessen für die zukünftige Forschung in dem Bereich den Grundstein eines vollautomatisierten Kontrollflusses von Verarbeitungsschritten für die direkte Analyse und Auswertung von

Interaktionen beziehungsweise Aussagen in Meetings von Entwicklungsteams bieten, auf dem weiter aufgebaut werden kann.

Horstmann [11] entwickelte im Rahmen seiner Masterarbeit eine Software die Textnachrichten, mithilfe eines evolutionären Algorithmus (EA) und maschinellen Lernverfahren, in eine der drei Kategorien positiv, negativ und neutral einordnet. Dies stellt eine sehr grobe erste Einordnung für das langfristige Ziel des Fachgebiets Software Engineering der Gottfried Wilhelm Leibniz Universität Hannover eine automatische Einordnung in die Kategorien aus dem Verfahren Act4teams-SHORT [16] zu erreichen dar, und diente als Grundlage für das Bestreben dieser Arbeit den Prozess weiter zu automatisieren. Seine Resultate testete Horstmann [11] außerdem an Textnachrichten aus Gruppenchats eines privaten Softwareunternehmens und stellte so auch die Funktionalität seiner Software in diesem Umfeld unter Beweis. Es mussten zwar einige Änderungen vorgenommen werden, da aus den Textnachrichten beispielsweise auch Merkmale wie Emoticons extrahiert wurden welche bei Transkripten von Aussagen natürlich nie vorkommen, aber dennoch stellt der Klassifikationsalgorithmus [11, 24] den finalen und entscheidendsten Verarbeitungsschritt der im Rahmen dieser Arbeit entwickelten Software dar.

Die von Horstmann [11] entwickelte Software beziehungsweise der EA wurde von Meyer [24] im Rahmen seiner Bachelorarbeit optimiert. Dazu wurde die Software unter anderem umgeschrieben, um die englische Sprache zu erkennen und die Polarität anhand des *Senti4SD* Klassifikators von Calefato et al. [4] zu bestimmen. Senti4SD [4] wurde speziell für die Polaritätsbestimmung von Nachrichten in Kommunikationskanälen von Softwareentwicklern trainiert. So werden z.B. bestimmte Wörter wie „kill" nicht als negativ klassifiziert, da diese in der Softwareentwicklung eine andere Bedeutung haben. Damit ist die optimierte Version der Software von Meyer [24] viel domänenspezifischer als die von Horstmann, da für die deutsche Sprache nur allgemeine Sentimentlexika existieren, und nicht spezifische für die Softwareentwicklung. Durch diese und andere angewendete Verbesserungen wurde, wenn die durch die alleinige Umstellung auf Englisch erreichte Verbesserung vernachlässigt wird, eine Verbesserung der Genauigkeit von Horstmanns Ergebnissen [11] um 5,1 Prozentpunkte erreicht. Da die Optimierungen nur für die englische Sprache entwickelt wurden ließen sie sich für diese Arbeit leider nicht anwenden, da die Metriken wie Senti4SD nur Englisch erkennen, und es zum jetzigen Zeitpunkt keine deutschen Alternativen gibt.

Powelske [35] analysierte in seiner Masterarbeit die Stimmung in Open-Source-Projekten. Als Quelle nutze er dafür Kommentare der Plattform GitHub, und entwickelte und implementierte ein Verfahren welches ähnlich

wie das von Horstmann [11] entwickelte Verfahren ebenfalls die Stimmung innerhalb des Projektes anhand der Textkommunikation verfolgt. Da er dafür auch englische Daten nutze, griff er, ebenfalls wie Meyer [24], auf Senti4SD [4] zurück. Die Arbeit von Powelske lässt sich damit ebenfalls in die Interaktionsanalyse in Meetings von Softwareentwicklungsteams einordnen, ähnlich wie bei Meyer [24] konnten Powelskes Resultate aber, aufgrund der Nutzung von Tools die nur für die englische Sprache funktionieren, nicht für diese Arbeit genutzt werden, da die im Rahmen dieser Arbeit entwickelte Software deutsche Sprache transkribieren und analysieren sollte.

Calefato et al. [4] befassten sich ebenfalls mit der Sentiment Analyse von Emotionen von Softwareentwicklern. Sie grenzten das ganze jedoch auf Daten die von Beteiligten an Open-Source-Projekten auf sozialen Software Engineering Plattformen wie GitHub hinterlassen wurden ein [4]. Wer an einem Open-Source-Projekt mitwirkt, hat dort die Möglichkeit zu seinen Änderungen Kommentare mitzuteilen, oder mögliche Probleme des Codes aufzuzeigen. Die Daten sammelten Calefato et al. [4] mithilfe von Data-Mining [4]. Sie stellten jedoch fest, dass Fertiglösungen zur Sentiment Analyse allesamt für nicht-technische Umgebungen trainiert wurden, und häufig zu falschen Klassifikationen von technischen Begriffen führten [4]. Daher stellten sie in ihrem Paper [4] Senti4SD, einen Sentiment-Klassifikator der speziell für die Domäne der Softwareentwicklung trainiert wurde, vor. In ihren Ergebnissen präsentierten sie eine deutliche Reduzierung von falschen Klassifikationen positiver und neutraler Aussagen zu negativ [4]. Obwohl die Ergebnisse der Arbeit von Calefato et al. [4] sehr beeindruckend sind lassen sie sich leider nur für die englischsprachige Sentiment Analyse in Entwicklungsteams wie bei den Arbeiten von Meyer [24] und Powelske [35] anwenden, und funktionieren nicht für die deutsche Sprache, wie die im Rahmen dieser Arbeit entwickelte Software.

Lin et al. [20] stellten ebenfalls fest, dass bestehende Tools zur Sentiment Analyse unzuverlässig Ergebnisse für Software Engineering Datensätze lieferten. Daher extrahierten sie, ähnlich wie Calefato et al. [4], 40.000 Kommentare von der Plattform *Stack Overflow*, auf der Softwareentwickler Fragen stellen und beantworten können [20]. Diesen Datensatz haben sie manuell mit Trainingslabeln versehen und nutzen ihn um eines der bestehenden Tools zur Sentiment Analyse damit zu trainieren. Trotz des hohen manuellen Aufwands und viel Zeit, die in das Projekt investiert wurde, berichteten Lin et al. [20] von negativen Ergebnisse. Im Folgenden änderten sie ihren Fokus, um eine Untersuchung der Genauigkeit von häufig verwendeten Tools zur Sentiment Analyse auf Software Engineering Datensätzen durchzuführen, sowie den Einfluss verschiedener Datensätze auf die Performanz der Tools. [20]. Ihre Ergebnisse sahen sie als eine Warnung an die Forschungsgemeinschaft über die aktuell starken Limitierungen

von Sentiment Analyse Tools [20]. Die Arbeit von Lin et al. [20] zeigt Limitierungen des Standes der Technik auf. Da im Rahmen dieser Arbeit deutsche Sprache untersucht werden sollte, konnte sowieso keine Domänenspezifische Sentiment Analyse für den Bereich des Software Engineering durchgeführt werden. Die verwendete Sentimentanalyse von Horstmann [11] basiert auf den Datensätzen von Waltinger [46] und Remus et al. [37] die lediglich allgemeine deutsche Begriffe mit einem Sentiment versehen.

7.3 Abgrenzung der Arbeit

Obwohl sich diese Arbeit in einzelnen Punkten zu den hier aufgezeigten Arbeiten zuordnen lässt, stellt sie eher einer Verknüpfung der verschiedenen Themenbereiche der Spracherkennung und Interaktionsanalyse in Entwicklungsteams dar, die es so noch nicht gibt. Diese Arbeit wurde jedoch durch die vorgestellten Arbeiten beeinflusst, beruht auf deren Resultaten, und wurde sogar erst durch sie motiviert. Die Zielsetzung ist jedoch im Fall dieser Arbeit eine komplett neue, da sie sich nicht mehr nur mit der Analyse von Textdaten befasst, sondern ein automatisiertes Konzept für Spracherkennung und Sentimentanalyse entwickelt und implementiert.

Kapitel 8

Zusammenfassung und Ausblick

Diese Kapitel gibt einen abschließenden Überblick über die wichtigsten Ergebnisse und Inhalte dieser Arbeit. Angefangen in Abschnitt 8.1 mit einer Wiederholung der eigentlichen Problemstellung und der erarbeiteten Lösung bis zu Abschnitt 8.2, indem unter anderem Möglichkeiten auf dieser Arbeit aufzubauen aufgezeigt werden.

8.1 Zusammenfassung

Zu Beginn dieser Arbeit gab es den Klassifikationsalgorithmus [11, 24] und das Bestreben, im Rahmen der Forschung des Fachgebiets Software Engineering der Gottfried Wilhelm Leibniz Universität Hannover, das Verfahren weiter zu automatisieren. Daraus ergab sich das Ziel dieser Arbeit eine Software zu entwickeln welche in einem einzelnen Ablauf von Verarbeitungsschritten Aussagen aus Gesprächen in Meetings von Entwicklungsteams transkribiert, und mithilfe des Klassifikationsalgorithmus [11, 24] auswertet. Hierfür wurde zunächst ein Konzept von Verarbeitungsvorgängen für den Ablauf der notwendigen Schritte von der Eingabe von Audiodaten bis zur Ausgabe der Vorhersagen erstellt. Für die Implementierung des Konzeptes wurden verschiedene Open Source Lösungen zur Spracherkennung betrachtet und die geeignetste, auf dem aktuellen Entwicklungsstand der Spracherkennung ausgewählt. Resultate aus der Forschung für deutsche Spracherkennung im Rahmen der ausgewählten Lösung wurden eingearbeitet und benutzt um eine möglichst hohe Qualität der automatisiert erstellten Transkripte zu erhalten. Dafür wurde auch auf eine Sprechpausenerkennung (VAD) gesetzt um den Audiostream in einzelne Frames von Aussagen aufzuteilen. Durch die Resultate von Horstmann [11] konnten anschließend die nötigen Metriken aus den Aussagen mittels Natural Language Processing (NLP) extrahiert werden. Der Klassifikationsalgorithmus [11, 24] diente für die anschließende

Bestimmung der Vorhersagen der Klassen positiv, negativ und neutral für die einzelnen Aussagen aufgrund ihrer zugehörigen Metriken. Die entwickelte Software, und damit auch das Verfahren, wurde mit Audiodaten aus Meetings von studentischen Softwareprojekten, welche manuell transkribiert und gelabelt wurden trainiert, und getestet. Es ergab sich dabei eine hohe Übereinstimmung der vergebenen Klassen durch die Software und manuell durch eine Person von 88%. Aufgrund einer Vielzahl von übereinstimmenden Klassifikationen in der Klasse neutral ergab sich jedoch auch eine ebenfalls hohe Wahrscheinlichkeit zufälliger Übereinstimmung von 72,8%, welcher bei der Anwendung des Fleiss' Kappa Verfahren [6] einen κ-Wert von 0,56 ergab, was einer moderaten Übereinstimmung zwischen der Software und der manuellen Klassifikation entspricht [18]. Damit konnte die Anwendbarkeit des Verfahrens bewiesen werden, es gibt jedoch auch weiteres Potenzial für Verbesserungen, insbesondere in den Bereichen der Spracherkennung und des Klassifikationsalgorithmus [11, 24].

8.2 Ausblick

Zu den Problemen die im Rahmen dieser Arbeit nicht gelöst werden konnten gehören wie in der Zusammenfassung bereits erwähnt wurde hauptsächlich die Spracherkennung und der Klassifikationsalgorithmus [11, 24]. Erstere bietet mit dem aktuelle genutzten Modell eine Wortfehlerquote (WER) von 12,8%, dieses ist zwar zum Zeitpunkt der Verfassung dieser Arbeit das beste verfügbare deutsche Modell gewesen, aber immer noch mehr als doppelt so schlecht wie die menschliche WER von ungefähr 6% [42]. Sollten in der Zukunft bessere trainierte Modelle für die deutsche Sprache veröffentlicht werden, können diese einfach mit denen in der Software verwendeten ausgetauscht werden, um bessere Ergebnisse beim transkribieren zu erhalten. Auch die verwendete Sprachengine Deepspeech, wird stetig weiterentwickelt, so kann dort auch ein neuer Release schon zu einer verbesserten Performanz der Software führen. Der Ansatz selbst ein Sprachmodell zu trainieren ist ebenso denkbar, jedoch durch die hohe benötigte Rechenleistung beziehungsweise den hohen Zeitaufwand sehr aufwendig. Die Vorhersagen des Klassifikationsalgorithmus [11, 24] bieten ebenfalls viele Verbesserungsmöglichkeiten, zunächst durch eine Erweiterung des Trainingsdatensatzes wie in Abschnitt 6.2 angesprochen, oder durch Änderungen am Klassifikationsalgorithmus [11, 24] wie der Implementierung eines Konzepts eines Kontextes einer Aussage. In Anbetracht der Forschung des Fachgebiets Software Engineering der Gottfried Wilhelm Leibniz Universität Hannover bleibt noch die feingranulare Klassifizierung der Aussagen in die Klassen des Verfahrens Act4Teams-SHORT, anstatt nur in die Klassen positiv, negativ und neutral, zu entwickeln. Eine solche Änderung am Klassifikationsalgorithmus [11, 24] ließe sich auch auf die entwickelte Software

übertragen. Falls in Zukunft deutsche Sentimentanalyse Tools, die speziell für den Einsatz in Entwicklungsteams entwickelt wurden, erscheinen können diese ebenfalls integriert werden, so wie Meyer [24] das englischsprachige Senti4SD [4] genutzt hat, um das Konzept von Horstmann [11] zu verbessern. Eine allgemeine domänenunabhängige Klassifikation bietet jedoch den Vorteil, dass die Software in mehr Bereichen eingesetzt werden kann, de facto in Meetings von Teams aus allen Wirtschaftsbereichen. Aufgrund der andauernden Sars-CoV2 Pandemie, und der Folge, dass Meetings immer noch auf unbestimmte Zeit Online abgehalten werden, wäre auch eine Anbindung der Software an eine Open Source Kommunikationsplattform wie *BigBlueButton* denkbar, um dort eine direkte Auswertung der abgehaltenen Meetings zu erhalten.

Literaturverzeichnis

[1] A. Agarwal and T. Zesch. Automatic Speech Recognition (ASR) - DeepSpeech German. `https://github.com/AASHISHAG/deepspeech-german`. letzer Zugriff: November 2020.

[2] A. Agarwal and T. Zesch. German end-to-end speech recognition based on deepspeech. In *Preliminary proceedings of the 15th Conference on Natural Language Processing (KONVENS 2019): Long Papers*, pages 111–119, Erlangen, Germany, 2019. German Society for Computational Linguistics & Language Technology.

[3] Baidu Research. Homepage. `http://research.baidu.com/`. letzer Zugriff: November 2020.

[4] F. Calefato, F. Lanubile, F. Maiorano, and N. Novielli. Sentiment polarity detection for software development. *Empirical Software Engineering*, 23:1352–1382, 05 2018.

[5] Dudenredaktion. *Duden - Die deutsche Rechtschreibung (28. Auflage)*. Dudenverlag (Bibliographisches Institut GmbH), Berlin, 2020.

[6] J. Fleiss. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5):378—382, November 1971.

[7] Google Brain. TensorFlow. `https://www.tensorflow.org/`. letzer Zugriff: November 2020.

[8] J. Grimm and W. Grimm. *Deutsches Wörterbuch*. S. Hirzel Verlag, Leipzig, 1852.

[9] A.-C. Haak. Improving meeting analysis in development teams using tutorials. Master's thesis, Gottfried Wilhelm Leibniz Universität Hannover, June 2020.

[10] A. Y. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng. Deep speech: Scaling up end-to-end speech recognition. *CoRR*, abs/1412.5567, 2014.

[11] J. Horstmann. Tool-supported analysis of communication behavior in development teams considering digital media. Master's thesis, Gottfried Wilhelm Leibniz Universität Hannover, August 2019.

[12] S. Kauffeld. *Kompetenzen messen, bewerten, entwickeln.* Schäffer-Poeschel, Stuttgart, 2006. 978-3-7910-2508-7.

[13] J. Kluender, C. Unger-Windeler, F. Kortum, and K. Schneider. Team meetings and their relevance for the software development process over time. In *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 313–320. IEEE, 2017.

[14] J. Klünder. *Analyse der Zusammenarbeit in Softwareprojekten mittels Informationsflüssen und Interaktionen in Meetings.* PhD thesis, Gottfried Wilhelm Leibniz Universität Hannover, 2019. Logos Verlag Berlin, 978-3-8325-4898-8.

[15] J. Klünder, L. Handke, T. Gfesser, K. Schneider, and S. Kauffeld. Soziale Aspekte traditioneller und hybrider Softwareentwicklung. In *Proceedings of the 15th Workshop on „Software Engineering im Unterricht der Hochschulen"(SEUH 2017)*, 02 2017.

[16] J. Klünder, N. Prenner, A.-K. Windmann, M. Stess, M. Nolting, F. Kortum, L. Handke, K. Schneider, and S. Kauffeld. Do you just discuss or do you solve? Meeting analysis in a software project at early stages. In *Proceedings of 42nd International Conference on Software Engineering Workshops*, Seoul, 10 2020. IEEE/ACM.

[17] R. Kruse, C. Borgelt, C. Braune, F. Klawonn, C. Moewes, and M. Steinbrecher. *Computational Intelligence: Eine methodische Einführung in Künstliche Neuronale Netze, Evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze. Zweite Auflage.* Springer-Vieweg, Wiesbaden, 2015. `https://www.computational-intelligence.eu/?page_id=174`.

[18] J. Landis and G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33 1:159–74, 1977.

[19] Lernhelfer. Lexikon, Deutsch, 3 Grammatik und Rechtschreibung, 3.1 Grundlagen und Voraussetzungen, 3.1.3 Phonem. `https://www.lernhelfer.de/schuelerlexikon/deutsch/artikel/phonem`. letzer Zugriff: Dezember 2020.

[20] 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)*, pages 94–104, 2018.

[21] R. Lippmann. Speech recognition by machines and humans. *Speech Communication*, 22:1–15, 07 1997.

[22] A. L. Maas, A. Y. Hannun, D. Jurafsky, and A. Y. Ng. First-pass large vocabulary continuous speech recognition using bi-directional recurrent dnns. *CoRR*, abs/1408.2873, 2014.

[23] D. Mazzoni. Audacity. `https://www.audacityteam.org/`. letzer Zugriff: Februar 2021.

[24] C. Meyer. Improving evolutionary algorithms for classifying text-based communication in development teams. Bachelor's Thesis, Gottfried Wilhelm Leibniz Universität Hannover, November 2020.

[25] B. Milde and A. Köhn. Open source automatic speech recognition for german. In *Proceedings of ITG 2018*, 2018.

[26] Mozilla Research. Common Voice Projekt. `https://commonvoice.mozilla.org/de`. letzer Zugriff: November 2020.

[27] Mozilla Research. Common Voice Projekt - Deutscher Datensatz. `https://commonvoice.mozilla.org/de/datasets`. letzer Zugriff: November 2020.

[28] Mozilla Research. Deepspeech - GitHub. `https://github.com/mozilla/DeepSpeech`. letzer Zugriff: November 2020.

[29] Mozilla Research. Microphone VAD Streaming - DeepSpeech 0.9.x Examples - GitHub. `https://github.com/mozilla/DeepSpeech-examples/tree/r0.9/mic_vad_streaming`. letzer Zugriff: Januar 2021.

[30] Mozilla Research. Speech & Machine Learning. `https://research.mozilla.org/machine-learning/`. letzer Zugriff: Dezember 2020.

[31] A. Murphy, B. Kelly, K. Bergmann, K. Khaletskyy, R. O'Connor, and P. Clarke. *Examining Unequal Gender Distribution in Software Engineering*, pages 659–671. 08 2019.

[32] A. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan. Speech recognition using deep neural networks: A systematic review. *IEEE Access*, PP:1–1, 02 2019.

[33] Open Source. DeepSpeech-Polyglot. `https://gitlab.com/Jaco-Assistant/deepspeech-polyglot`. letzer Zugriff: November 2020.

[34] D. Povey. LibriSpeech ASR corpus. `http://www.openslr.org/12`. letzer Zugriff: Dezember 2020.

[35] T. Powelske. Sentiment analysis in open source projects using text-based communication. Master's thesis, Gottfried Wilhelm Leibniz Universität Hannover, November 2020.

[36] S. Radeck-Arneth, B. Milde, A. Lange, E. Gouvea, S. Radomski, M. Mühlhäuser, and C. Biemann. Open Source German Distant Speech Recognition: Corpus and Acoustic Model. In *Proceedings Text, Speech and Dialogue (TSD)*, pages 480–488, Pilsen, Czech Republic, 2015.

[37] R. Remus, U. Quasthoff, and G. Heyer. SentiWS - a publicly available German-language resource for sentiment analysis. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta, May 2010. European Language Resources Association (ELRA).

[38] G. Saon. Man vs. machine in conversational speech recognition. *2017 IEEE Automatic Speech Recognition and Understanding Workshop* December 16-20, 2017, Okinawa, Japan, 2017.

[39] G. Saon and M. Picheny. Recent advances in conversational speech recognition using convolutional and recurrent neural networks. *IBM Journal of Research and Development*, 61:1:1–1:10, 07 2017.

[40] M. Sarma. Speech recognition using deep neural network - recent trends. *International Journal of Intelligent Systems Design and Computing*, 1:71, 01 2017.

[41] K. Schneider, J. Klünder, F. Kortum, L. Handke, J. Straube, and S. Kauffeld. Positive affect through interactions in meetings: The role of proactive and supportive statements. *Journal of Systems and Software*, 143, 05 2018.

[42] A. Stolcke and J. Droppo. Comparing human and machine errors in conversational speech transcription. *CoRR*, abs/1708.08615, 2017.

[43] TU Darmstadt. VoxForge - Open Speech Data Corpus for German. `http://www.voxforge.org/home/forums/other-languages/german/open-speech-data-corpus-for-german`. letzer Zugriff: November 2020.

[44] Universität Hamburg, Fachbereich Informatik, Language Technology Group. Open Source Acoustic Models for German Distant Speech Recognition. `https://www.inf.uni-hamburg.de/en/inst/ab/lt/resources/data/acoustic-models.html`. letzer Zugriff: November 2020.

[45] G. Van Rossum. Python. `https://www.python.org/`. letzer Zugriff: November 2020.

[46] U. Waltinger. Sentiment analysis reloaded - a comparative study on sentiment polarity identification combining machine learning and subjectivity features. In *WEBIST 2010, Proceedings of the 6th International Conference on Web Information Systems and Technologies, Volume 1, Valencia, Spain, April 7-10, 2010*, pages 203–210. INSTICC Press, 2010.

Abbildungsverzeichnis

Tabellenverzeichnis