

Untersuchung des Einsatzes von Open Source Natural Language Understanding Services für Conversational Interfaces

Masterarbeit

im Studiengang
Computer Science and Media

vorgelegt von

Marco Maisel

Matr.-Nr.: 33800

am 17.12.2018

an der Hochschule der Medien Stuttgart

Erstprüfer/in:	Prof. Dr.-Ing. Johannes Maucher
Zweitprüfer/in:	Steffen Blümm

Ehrenwörtliche Erklärung

„Hiermit versichere ich, Marco Maisel, ehrenwörtlich, dass ich die vorliegende Masterarbeit mit dem Titel: „Untersuchung des Einsatzes von Open Source Natural Language Understanding Services für Conversational Interfaces“ selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Ich habe die Bedeutung der ehrenwörtlichen Versicherung und die prüfungsrechtlichen Folgen (§ 26 Abs. 2 Bachelor-SPO (6 Semester), § 24 Abs. 2 Bachelor-SPO (7 Semester), § 23 Abs. 2 Master-SPO (3 Semester) bzw. § 19 Abs. 2 Master-SPO (4 Semester und berufsbegleitend) der HdM) einer unrichtigen oder unvollständigen ehrenwörtlichen Versicherung zur Kenntnis genommen.“

Stuttgart, 17.12.2018

Kurzfassung

Conversational Interfaces ermöglichen die Interaktion mit Computern durch Kommunikation im Stile einer Konversation. Die Idee dieser Interaktion existiert bereits seit langem. Die Rechenleistungen heutiger Prozessoren und Fortschritte in der Künstlichen Intelligenz ermöglichen jedoch erst die geforderte Komplexität handzuhaben, die zur derzeitigen Akzeptanz und Popularität dieser Systeme im Consumer-Bereich geführt hat. Neben Sprachassistenten wie Apples Siri und Amazons Alexa gehören auch Chatbots, die eine bidirektionale Echtzeitkommunikation in Textform ermöglichen, zu den Conversational Interfaces.

Ermöglicht wird diese Art der Kommunikation durch das Verarbeiten der Text- oder Audio-Daten mittels Natural Language Understanding. Dies geschieht meist auf Servern des jeweiligen Anbieters des Ökosystems. Da jedoch der Schutz eigener Daten sowie von Nutzerdaten gerade in sicherheitskritischen Bereichen wie dem Finanzsektor oder Human-Resources-Bereich wichtig ist, stellt die externe Speicherung und Verarbeitung dieser Daten ein Problem dar und führt dazu, dass solche Systeme oft kritisch betrachtet werden. Ein Ansatz die Akzeptanz von Conversational Interfaces in diesen Domänen zu erhöhen ist der Einsatz von Open-Source-Lösungen, die ausschließlich auf eigenen Servern betrieben werden.

Vor diesem Hintergrund soll für die Human-Resources-Abteilung der Firma adorsys GmbH & Co. KG ein Chatbot für Bewerber entwickelt werden, dessen Datenverarbeitung in Zukunft ausschließlich auf firmeninternen Servern stattfinden soll. Dazu werden zuerst mehrere aktuelle Open Source NLU Services betrachtet und verglichen. Als ein wesentliches Vergleichskriterium wird die Leistungsfähigkeit des Klassifikators herangezogen. Um für diesen speziellen Anwendungsfall aussagekräftige Ergebnisse zu erhalten, werden domänenspezifische Trainingsdaten gesammelt und zur Evaluation verwendet.

Auf Basis der vorangegangenen Evaluation wird im Anschluss unter Verwendung einer dieser Services ein Chatbot für Bewerber prototypisch konzipiert und realisiert.

Abstract

Conversational Interfaces promise to enable humans to interact with computing devices in form of natural conversations. Although the idea itself has existed for a long time, Conversational Interfaces take advantage of the most recent developments in computing. The tremendous amount of computing power available via cloud computing as well as the advances in the field of Artificial Intelligence are two of them. Besides language assistants like Apple's Siri and Amazon's Alexa, chatbots, which enable bidirectional real-time communication in text form, are a huge share of conversational interfaces.

This type of communication is made possible by processing text or audio data using natural language understanding. This is usually done by services hosted on servers of the respective provider of the ecosystem. However, as privacy and data protection is a severe concern, and in some countries reinforced by law, server-side processing and storage of personal data as dealt with by financial applications or human resource services is under critical discourse. One approach to increase the acceptance of conversational interfaces in these domains is the use of open source solutions, which are operated entirely on proprietary servers.

Therefore, a chatbot for candidates is to be developed for the human resources department of adorsys GmbH & Co. KG, whose data processing will exclusively take place on in-house servers in the future. To be able to design a viable solution several of the current open source natural language understanding services are analysed and evaluated. The performance of the classifier is used as a primary comparison criterion. In order to gain meaningful results for this application, domain-specific training data is collected and used for evaluation.

Based on this evaluation, a chatbot for candidates is then prototypically designed and implemented using one of these services.

Inhaltsverzeichnis

Ehrenwörtliche Erklärung.....	ii
Kurzfassung.....	iii
Abstract.....	iv
Inhaltsverzeichnis	v
Abbildungsverzeichnis.....	viii
Tabellenverzeichnis	ix
Quellcodeverzeichnis	x
Abkürzungsverzeichnis	xi
1 Einleitung	1
1.1 Motivation und Ziele.....	1
1.1.1 Chatbots im Bereich Human Resources	2
1.1.2 Gründe für Open-Source-Technologien	3
1.2 Ziele der Arbeit	4
1.3 Aufbau der Arbeit	5
2 Theoretischer Hintergrund.....	6
2.1 Conversational Interfaces.....	6
2.1.1 Aufbau einer Konversation.....	6
2.1.2 Kategorisierung eines Chatbots	7
2.1.3 Architektur eines Chatbots	8
2.1.4 Historie der Conversational Interfaces	9
2.1.5 Chatbots als Conversational Assistants	10
2.1.6 Begriffe im Kontext des NLU's.....	11
2.2 Natural Language Processing und Understanding	11
2.3 Maschinelle Lernverfahren im Kontext von Chatbots	13
2.3.1 Wort-Embeddings.....	13
2.3.2 Conditional Random Fields	14
2.3.3 Support Vektor Maschinen	14
2.3.4 Neuronale Netze	15
2.3.5 Rekurrente Neuronale Netze.....	17
2.3.6 Long Short-Term Memory Netzwerke	17

3	Evaluation	18
3.1	Übersicht der NLU-Services	18
3.1.1	Rasa NLU	18
3.1.2	Snips NLU	21
3.1.3	Mycroft Padatious.....	21
3.2	Datensammlung und -verarbeitung	22
3.3	Evaluationsverfahren.....	25
3.3.1	K-fache Kreuzvalidierung	26
3.3.2	Leistungsmetriken.....	26
3.4	Evaluationsergebnisse	28
4	Entwurf des Chatbot-Prototyps	33
4.1	Szenario und Umfang des HR-Chatbots	33
4.2	Intent Jobsuche.....	33
4.2.1	Szenario	33
4.2.2	Datenstruktur eines Jobs	35
5	Implementierung des Chatbot-Prototyps.....	37
5.1	Architektur	37
5.2	Umsetzung der Frontend-Komponente	38
5.3	Umsetzung der NLU-Komponente	39
5.3.1	Pipeline	40
5.4	Umsetzung der Dialogmanagement-Komponente	43
5.4.1	Architektur.....	44
5.4.2	Domain	45
5.4.3	Stories	48
5.4.4	Policy	49
5.5	Aktionen.....	50
5.5.1	Implementierung der Datenstruktur eines Jobs	52
5.5.2	Benutzerdefinierte Aktionen zur Jobsuche	53
5.6	Logging	57
5.6.1	Permanente Datenspeicherung.....	58
5.6.2	Visualisierung der Dialoginformationen	58
5.7	Deployment.....	60
5.8	Evaluation des Prototypen.....	62
5.8.1	Evaluation der NLU-Komponente.....	63

5.8.2	Evaluation des Gesamtsystems als Nutzertest	65
6	Schlussbetrachtung.....	68
6.1	Fazit.....	68
6.2	Ausblick	69
Anhang A:	70
A.1	Evaluationsergebnisse der NLU-Komponente	71
A.2	Logs Nutzertest (Auszug).....	74
Quellenverzeichnis		77

Abbildungsverzeichnis

Abbildung 1: Architektur eines Chatbots	8
Abbildung 2: Beispiel für Wort-Embeddings [38]	14
Abbildung 3: Beispiel einer SVM [40]	15
Abbildung 4: Perzeptron	16
Abbildung 5: Beispiel eines mehrschichtigen Neuronalen Netzes	16
Abbildung 6: Beispiel eines Rekurrenten Neuronalen Netzes	17
Abbildung 7: K-fache Kreuzvalidierung	26
Abbildung 8: Intent Jobsuche	34
Abbildung 9: Architektur des Chatbot-Prototyps	37
Abbildung 10: Chatbot-Widget auf der Webseite	39
Abbildung 11: Architektur des Chatbot-Prototyps mit NLU-Komponente	39
Abbildung 12: Rasa Component Lifecycle [61]	40
Abbildung 13: Architektur des Chatbot-Prototyps mit Dialogmanagement- Komponente	44
Abbildung 14: Architektur Rasa Core [47]	44
Abbildung 15: Architektur des Chatbot-Prototyps mit Action-Server	50
Abbildung 16: Datenstruktur eines Jobs	52
Abbildung 17: Beispiel-Dialog Stellensuche 1	54
Abbildung 18: Beispiel-Dialog Stellensuche 2	54
Abbildung 19: Beispiel-Dialog Stellensuche 3	55
Abbildung 20: Beispiel-Dialog Stellensuche 4	56
Abbildung 21: Beispiel-Dialog Stellensuche 5	56
Abbildung 22: Beispiel-Dialog Stellensuche 6	57
Abbildung 23: Architektur des Chatbot-Prototyps mit Logging_Komponente	58
Abbildung 24: Logging-Frontend	59
Abbildung 25: Architektur Logging-Komponente	60
Abbildung 26: OpenShift-Projekt	62
Abbildung 27: Konfusionsmatrix NLU-Komponente	64
Abbildung 28: Erfolgsquote Nutzertest	67

Tabellenverzeichnis

Tabelle 1: Entity-Typen im Evaluations-Datensatz.....	24
Tabelle 2: Evaluation Rasa NLU Pipeline 1	28
Tabelle 3: Evaluation Rasa NLU Pipeline 2	29
Tabelle 4: Evaluation Snips NLU	29
Tabelle 5: Evaluation Mycroft Padatious	30
Tabelle 6: Evaluation der NLU-Komponente des Prototyps	63

Quellcodeverzeichnis

Listing 1: Formatierung der Trainingsdaten für Rasa NLU	25
Listing 2: Rasa NLU Pipeline Konfiguration (<i>config_tensorflow.yml</i>).....	41
Listing 3: Rasa Core Domain: Intents und Entities (<i>domain.yml</i>).....	46
Listing 4: Rasa Core Domain: Slots (<i>domain.yml</i>).....	47
Listing 5: Rasa Core Domain: Actions (<i>domain.yml</i>).....	47
Listing 6: Rasa Core Domain: Templates (<i>domain.yml</i>)	47
Listing 7: Rasa Core Story „Greet“ (<i>stories.md</i>)	48
Listing 8: Definition von Aktionen (<i>actions.py</i>).....	51
Listing 9: Tracker-Objekt im Beispiel-Dialog.....	55
Listing 10: Dockerfile Rasa Core	61

Abkürzungsverzeichnis

AWS	Amazon Web Services
CLU	Conversational Language Understanding
CRF	Conditional Random Field
CUI	Conversational User Interface
EC2	Elastic Compute Cloud
FANN	Fast Artificial Neural Network
GUI	Graphical User Interface
HR	Human Resources
LSTM	Long Short-Term Memory
NLG	Natural Language Generation
NLP	Natural Language Processing
NLU	Natural Language Understanding
ORM	Object Relational Mapping
POS	Part-of-speech
SVM	Support Vektor Maschine
VUI	Voice User Interface
WIMP	Windows, Icons, Menus, Pointer

In der folgenden Arbeit wird aus Gründen der besseren Lesbarkeit ausschließlich die männliche Form verwendet. Dies ist jedoch nicht geschlechtsspezifisch gemeint, sondern geschieht ausschließlich aus Gründen der besseren Lesbarkeit.

1 Einleitung

Bereits im Jahr 2015 war der Zeitpunkt erreicht, an dem Messenger Apps weltweit mehr Nutzer verzeichnen konnten als soziale Netzwerke [1]. Dieser Trend zeigt sich auch in der Business-to-Consumer-Kommunikation. Laut Meekers 2017 Internet Trends Report ging der Kundenservice im letzten Jahr stark weg von herkömmlichen Kanälen wie Telefon oder E-Mail und hin zu chatbasierten Systemen [2].

Die derzeit häufigste Form der Interaktion mit einem Computer ist über eine grafische Benutzerschnittstelle, dem Graphical User Interface (GUI). Jedoch ist die WIMP¹-basierte GUI nicht die intuitivste Art der Interaktion, da zuerst gelernt werden muss, wie sich das Interface bedienen lässt und bei weiteren Interaktionen das Gelernte erneut abgerufen werden muss. Eine Schnittstelle, die den Benutzer nicht zwingt, bestimmte Befehle oder Interaktionsmethoden zu erlernen, erzeugt weniger Schwierigkeiten. [3]

Conversational User Interfaces (CUIs) ermöglichen die Interaktion mit Computern durch Kommunikation im Stile einer Konversation [4]. Die Idee dieser Interaktion existiert bereits seit langem. Die Rechenleistungen heutiger Prozessoren und Fortschritte in der Künstlichen Intelligenz ermöglichen jedoch erst die geforderte Komplexität handzuhaben, die zur derzeitigen Akzeptanz und Popularität dieser Systeme im Consumer-Bereich geführt hat. Google schreibt Conversational Interfaces sogar das Potenzial als die neue Art der Interaktion mit Computern zu [5]. Neben Sprachassistenten wie Apples Siri und Amazons Alexa gehören auch Chatbots, die eine bidirektionale Echtzeitkommunikation in Textform ermöglichen, zu den Conversational Interfaces.

1.1 Motivation und Ziele

Diese Arbeit konzentriert sich auf den Einsatz von textbasierten Chatbots im Unternehmensumfeld. Im Rahmen des Chatbot Market Research Report 2017 der Firma Mindbowser wurde eine Umfrage mit über 300 Teilnehmern verschiedenster Branchen durchgeführt. Diese zeigt, dass 96% der befragten Unternehmen glauben, dass Chatbots kein

¹ WIMP: Windows, Icons, Menus, Pointer

kurzweiliges Phänomen sind, sondern die Art der Interaktion mit Technologien langfristig verändern werden. Weiterhin sind 67% der Teilnehmer davon überzeugt, dass Chatbots mobile Anwendungen in den nächsten 5 Jahren an Leistung übertreffen werden [6]. Ähnlich positive Zukunftsprognosen für Chatbots zeigt eine Umfrage von Oracle. 80% der befragten Vertriebs- und Marketingleiter gaben an, Chatbots bereits zur Verbesserung der Kundenerfahrung zu nutzen oder planen, dies bis zum Jahre 2020 umzusetzen [7]. Auch Credence Research, ein globales Marktforschungs- und Beratungsunternehmen, erwartet ein exponentielles Wachstum des Marktes für Chatbots im Prognosezeitraum bis zum Jahre 2023 [8]. Dass sich Nutzer schnell an eine Interaktion im Stile einer Konversation gewöhnen verkündete Google zur I/O Developer Conference 2017. Demnach fanden bereits beinahe 70% aller Anfragen an Googles Assistent nicht durch die für eine Websuche typischen Schlüsselwörter, sondern in natürlicher, dialogorientierter Sprache statt [9].

1.1.1 Chatbots im Bereich Human Resources

Eine Studie von Randstad US zeigt, dass 95% der befragten Arbeitnehmer der Meinung sind, Technologie sollte als Unterstützung des Rekrutierungsverfahrens eingesetzt werden. Es herrscht jedoch Einigkeit darüber, dass neuartige Technologien nicht als Ersatz, sondern ausschließlich als Unterstützung angewendet werden sollten, da sich der Prozess der Stellensuche dadurch unpersönlicher gestaltet [10]. Ein Einsatz von Chatbots im Bereich Human Resources (HR) muss deshalb gut konzipiert werden, um das Gefühl der Unpersönlichkeit weitestgehend zu vermeiden. Auch eine Studie der Firma ubisend zeigt, dass bereits im Jahr 2016 über 45% der Befragten sagten, sie würden ein Unternehmen lieber über eine Messaging-Anwendung als über E-Mail kontaktieren. Als Grund wurde vor allem genannt, dass Messaging-Anwendungen weniger unpersönlich und kalt wirken. Allerdings sagten knapp 50% der Teilnehmer auch, dass sie ein Unternehmen lieber über Messaging-Anwendungen kontaktieren als über das Telefon, obwohl eine telefonische Kontaktaufnahme noch persönlicher wäre [11].

Ein Blick auf die aktuelle Situation von Bewerbungsprozessen zeigt, dass es deutliches Verbesserungspotential gibt. Die Beratungsfirma Net Federation hat die Bewerberfreundlichkeit der Recruiting-Webseiten von 75 deutschen Konzernen anhand von 157 Kriterien

untersucht. Dabei wurde aufgezeigt, dass häufig Funktionalitäten fehlen, die es den Bewerbern erleichtert sich zu bewerben.

Bewerbungsprozesse und die damit zusammenhängenden Kommunikationsabläufe werden fast ausschließlich unter den Bedingungen der HR-Prozesse praktiziert und berücksichtigen kaum die Bedürfnisse der Bewerber. Diesen Zuständen könnte mit einfachen technischen Erweiterungen Abhilfe geschaffen werden. Scheinbar scheuen sich die HR-Abteilungen der Unternehmen davor, traditionelle Strukturen mit technischen Neuerungen wie Chats, Kennenlernvideos oder One-Click-Bewerbungen aufzubrechen.
[12]

Ein Chatbot im Bewerbungsprozess kann eine Vielzahl an Aufgaben übernehmen und damit sowohl potentielle Bewerber als auch Mitarbeiter im Bereich des Personalwesens unterstützen. Ein Hauptaspekt vieler Chatbots ist die Vereinfachung der Informationssuche eines Nutzers. Anstatt komplette Webseiten oder FAQ-Listen nach Informationen zu durchsuchen oder ein Unternehmen telefonisch oder per Mail zu kontaktieren, kann ein Chatbot Antworten auf konkrete Fragen liefern. Dazu zählen in diesem Bereich unter anderem Informationen zum Unternehmen, dem Bewerbungsprozess samt Bewerbungsgespräch oder zu einer konkreten Stellenanzeige. Wird neben dem Bereitstellen von Informationen auch ein Teil des Bewerbungsprozesses in einem Chatbot abgewickelt, kann den Mitarbeitern im Personalbereich weitere Arbeit abgenommen werden. So können Informationen zum Bewerber sowie deren Motivation oder Fähigkeiten abgefragt und aufgenommen werden. Weiterhin können Chatbots zum Vereinbaren von Terminen wie z.B. Vorstellungsgesprächen verwendet werden. All diese Aufgaben können im Gegensatz zur Abwicklung durch einen Menschen simultan mit beliebig vielen Nutzern abgewickelt werden [13].

1.1.2 Gründe für Open-Source-Technologien

Eine aktuelle Umfrage der Firma PricewaterhouseCoopers (PwC) mit 1000 Teilnehmern im Alter von 18-65 Jahren von Februar 2018 zeigt auf, dass die Hälfte der Befragten, die selbst keine Erfahrung mit Sprachassistenten haben, diese meiden, weil sie Angst um ihre Privatsphäre haben. Die am häufigsten genannten Gründe waren, dass ein Gerät nicht

ununterbrochen mithören solle und dass die Befragten Angst um die Sicherheit ihrer Daten hätten [14].

Der Einsatz von Open-Source-Technologien hat im Gegensatz zum Einsatz kommerzieller Produkte einen großen Vorteil im Bereich des Datenschutzes. Ein wesentlicher Punkt dabei ist, dass keine Daten an Cloud-Services von Drittanbietern übertragen werden müssen. Die Speicherung und Verarbeitung aller Daten kann somit auf internen Servern des Anbieters oder Kunden stattfinden. Je nach gewünschtem Funktionsumfang des Conversational Interfaces ist sogar eine Implementierung auf einem physikalischen Gerät ohne Internetzugriff möglich. Der Einsatz solcher Systeme, die auf Open-Source-Technologien aufgebaut sind, kann somit die Akzeptanz, vor allem in sicherheitskritischen Domänen, erhöhen. Gleichzeitig wird durch deren Einsatz Unabhängigkeit von den großen Anbietern wie Facebook, Google, Microsoft, etc. gewonnen, welche aus unternehmerischer Sicht interessant sein könnte.

Weiterhin bringt die Verwendung von Open-Source-Technologien ein breiteres Spektrum an Anpassungsmöglichkeiten mit sich. So fällt eine Integration in eigene, bestehende Systeme einfacher. Innerhalb eines Chatbots basieren sowohl Komponenten im Bereich des Natural Language Understandings (NLU) als auch im Dialogmanagement oft auf Machine-Learning-Modellen. Kommerzielle Anbieter haben den Vorteil, dass deren Modelle, aufgrund der hohen Anzahl an Nutzerdaten, meist gut trainiert und für eine breitere Menge an Aufgaben nutzbar sind. Die Verwendung eigener Modelle bietet jedoch den Vorteil, dass diese besser an die eigenen Daten angepasst werden können und somit in einer speziellen Domäne bessere Leistung bringen können. Da diese Modelle im Optimalfall kontinuierlich mit gesammelten Daten trainiert und somit verbessert werden, ist es wiederum wichtig, diese Daten selbst zu besitzen.

1.2 Ziele der Arbeit

Vor diesem Hintergrund soll für die Human-Resources-Abteilung der Firma adorsys GmbH & Co. KG ein Prototyp eines Chatbots für Bewerber entwickelt werden, dessen Datenverarbeitung in Zukunft ausschließlich auf firmeninternen Servern stattfinden soll. Dazu werden zuerst mehrere aktuelle Open Source NLU Services betrachtet und vergli-

chen. Als ein wesentliches Vergleichskriterium wird die Leistungsfähigkeit des Klassifikators herangezogen. Um für diesen speziellen Anwendungsfall aussagekräftige Ergebnisse zu erhalten, werden domänenspezifische Trainingsdaten gesammelt und zur Evaluation verwendet.

Auf Basis der vorangegangenen Evaluation soll im Anschluss unter Verwendung einer dieser Services ein Chatbot für Bewerber prototypisch konzipiert und realisiert werden. Dabei soll ein detailliertes Verständnis einer Service-Umgebung geschaffen werden. Der Fokus des Prototyps liegt auf dem Szenario der Jobsuche anhand verschiedener Suchkriterien. Um mithilfe einer Konversation Stellenbeschreibungen finden zu können, soll eine Datenstruktur konzipiert und implementiert werden, die dies anhand von mehreren Abstraktionsebenen ermöglicht. Weiterhin soll die Möglichkeit geschaffen werden, Konversationen zu protokollieren und somit eine iterative Verbesserung des Systems zu ermöglichen.

1.3 Aufbau der Arbeit

Diese Arbeit gliedert sich, nach Einleitung und Motivation, in vier weitere Kapitel.

Kapitel 2 beschäftigt sich zunächst mit den theoretischen Grundlagen und begrifflichen Definitionen von CUIs und Chatbots, auf die im weiteren Verlauf zurückgegriffen wird.

In Kapitel 3 werden mehrere Open Source NLU Services vorgestellt und unter Verwendung domänenspezifischer Trainingsdaten verglichen. Anschließend wird eine Auswahl für die spätere Implementierung anhand der Evaluationsergebnisse getroffen.

Kapitel 4 beschreibt die prototypische Implementierung eines Chatbots für Bewerber zur Unterstützung des HR-Teams.

Kapitel 5 resümiert die Arbeit und gibt einen Ausblick auf mögliche Erweiterungen und anschließende Arbeiten.

2 Theoretischer Hintergrund

2.1 Conversational Interfaces

Chatbots ermöglichen die Interaktion mit Computern durch textbasierte, bidirektionale Kommunikation im Stile einer Konversation in Echtzeit. Die Kommunikation mit einem Chatbot orientiert sich in vielen Fällen an der Interaktion mit einem menschlichen Gegenüber, um die Konversation möglichst natürlich wirken zu lassen. Um ein System zu entwickeln, welches eine Mensch-zu-Mensch-Konversation nachbildet, muss zunächst der Aufbau einer Konversation analysiert werden.

2.1.1 Aufbau einer Konversation

Eine Konversation wird als informelle Art einer Interaktion beschrieben, bei der Neuigkeiten oder eigene Ansichten ausgetauscht werden. Sie zeigt sich als eine intuitive Form zwischenmenschlicher Interaktion. Neben dem Austausch von einfachen Äußerungen werden innerhalb einer Konversation allerdings auch Aktionen ausgeführt. Es werden Fragen gestellt, Komplimente und Versprechen gemacht und auf das Geäußerte des Gegenübers eingegangen [4]. Searle [15] unterscheidet in seiner im Jahr 1969 publizierten *Theory of Speech Acts* zwischen dem, was ein Konversationsteilnehmer mit seiner Äußerung meint und dem, was er bei seinem Gegenüber an Wirkung beabsichtigt. Diese Absicht, welche als Aktion angesehen werden kann, wird vom anderen Konversationsteilnehmer interpretiert und mit einer Äußerung, die darauf Bezug nimmt, beantwortet. Das Erkennen der Intention des Gesprächspartners ist sowohl bei Gesprächen zwischen Menschen als auch bei der Interaktion mit einem Chatbot ein wesentliches Kriterium einer erfolgreichen Kommunikation.

Ein weiterer Erfolgsfaktor einer Konversation ist die Möglichkeit der Kollaboration zwischen Gesprächsteilnehmern. Durch abwechselnde Äußerungen können sie Maßnahmen zum gegenseitigen Verständnis ergreifen, um Missverständnissen vorzubeugen oder gemeinsam an der Lösung von Problemen zu arbeiten [16]. Dieser Punkt spielt auch bei der Entwicklung eines Chatbots eine wichtige Rolle. Dabei muss vor allem bedacht werden,

wie auf Äußerungen reagiert wird, deren Intention nicht eindeutig zuzuordnen ist oder wie auf Nachfragen eines Nutzers zu Antworten des Chatbots reagiert werden kann.

2.1.2 Kategorisierung eines Chatbots

Eine mögliche Kategorisierung von Chatbots kann anhand ihrer Komplexität erfolgen. Quarteroni [17] unterscheidet dabei zwischen reaktiven und aufgabenorientierten Systemen:

Reaktive Systeme versuchen Muster in den Äußerungen eines Nutzers zu erkennen. Auf erkannte Muster wird daraufhin mit vordefinierten Antworten reagiert. Ziel dieser Systeme ist im Allgemeinen die Aufrechterhaltung einer Konversation ohne dabei eine spezifische Aufgabe zu erfüllen.

Aufgabenorientierte Systeme hingegen haben das Ziel einen Nutzer bei der Realisierung einer bestimmten Aufgabe zu unterstützen. Diese Systeme können, abhängig von der Art der Aufgabe, in vier Unterkategorien eingeteilt werden:

1. *Systeme zur Informationssuche* dienen als Quelle von bestimmten Informationen ohne dabei Daten im Hintergrund zu manipulieren.
2. *Transaktionssysteme* hingegen können andere Systeme modifizieren, um beispielsweise Daten zu speichern oder zu verändern. Beispiele hierfür sind Systeme, bei denen Nutzer ihre Kontaktinformationen eingeben können oder Systeme zur Flug- oder Reisebuchung.
3. *Prozedurale Systeme* werden vor allem für komplexere Aufgaben verwendet. Dabei folgt das System einem bestimmten Plan, der entweder im Voraus definiert oder während der Laufzeit berechnet wird. Diese Systeme kommen unter anderem zur technischen Fehlersuche zum Einsatz.
4. *Persuasive Systeme* verfügen über ein komplexes internes Modell, welches es ihnen ermöglicht, ein eigenes Ziel im Gespräch zu erreichen.

Nach dieser Kategorisierung sind die meisten Chatbots, die heutzutage eingesetzt werden, der Kategorie der aufgabenorientierten Systeme zuzuordnen. Dabei kommen vor allem Systeme zur Informationssuche zum Einsatz, welche limitierte Fähigkeiten zu Transaktionen oder prozeduralen Aktionen besitzen [17].

2.1.3 Architektur eines Chatbots

Die typische Architektur eines Chatbots besteht aus einem Frontend zur Nutzereingabe und drei Backend-Komponenten. Abbildung 1 zeigt einen groben Überblick dieser Architektur:

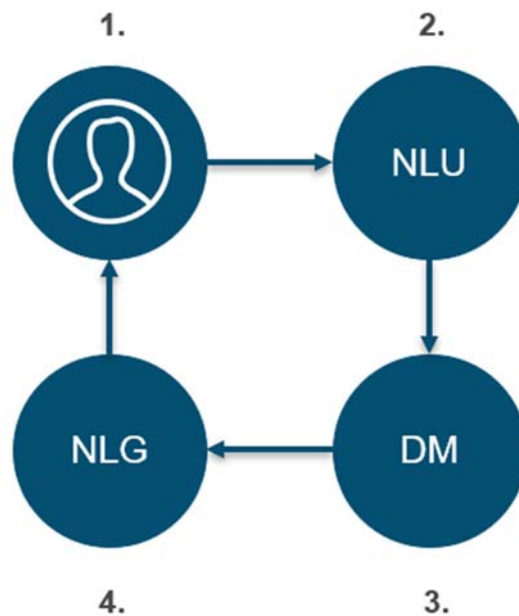


Abbildung 1: Architektur eines Chatbots

1. Interaktion mit dem System: Eine Äußerung des Nutzers in Textform trifft im System ein.
2. Natural Language Understanding: Aus der eingegangenen Äußerung werden Informationen extrahiert.
3. Dialogmanagement: Als Reaktion auf die extrahierten Informationen aus der Nutzereingabe wird eine Aktion ausgewählt, die zurückgegeben werden soll.
4. Natural Language Generation: Für die ausgewählte Aktion wird eine Antwort in natürlicher Sprache erzeugt.

Einzelheiten zu einzelnen Komponenten sowie die erweiterte Architektur des implementierten Chatbots folgen in Kapitel 5.

2.1.4 Historie der Conversational Interfaces

Die Historie der Conversational Interfaces beginnt bereits in den 1950er und 1960er Jahren, wobei erste Tests zur möglichen Intelligenz von Computern und Programme zur Simulation von textbasierten Konversationen entwickelt wurden. Im Folgenden soll ein grober Überblick über die Entwicklung von Conversational Interfaces im Laufe der Zeit anhand einiger ausgewählter Konzepte und Technologien gegeben werden.

Alan Turing entwickelte im Jahr 1950 den Turing-Test. Dieser soll herausfinden, ob ein Computerprogrammes in der Lage ist, einen Menschen so zu täuschen, dass dieser glaubt, er spreche mit einem anderen Menschen. Während eines Testablaufs stellt die Testperson Fragen über eine textbasierte Schnittstelle und erhält daraufhin Antworten. Die Antworten sind entweder von einer Person oder von einem Computerprogramm. Aufgabe der Testperson ist es, herauszufinden, ob seine Konversation mit einem Menschen oder einem Computer stattfindet. Nach Turing könne ein System dann als intelligent bezeichnet werden, wenn Menschen bei einer beliebigen Konversation mit diesem nicht eindeutig unterscheiden können, ob es sich um einen Menschen oder einen Computer handelt [18].

Im Jahr 1966 entwickelte Joseph Weizenbaum *ELIZA*, ein Computerprogramm, welches eine Konversation mit einem Computer in natürlicher Sprache ermöglichte und somit eine der ersten Versionen eines Chatbots darstellte. Eingaben von Nutzern wurden dabei auf bestimmte Schlüsselwörter untersucht. Wurde ein bekannter Begriff gefunden, wurde der eingegebene Satz aufgrund einer, mit dem Schlüsselwort verknüpften, Regel verändert und zurückgegeben. Konnte kein Schlüsselwort gefunden werden, wurde mit einer von mehreren vordefinierten Phrasen geantwortet, die den Nutzer dazu bringen soll, weitere Informationen preiszugeben. Dabei wurde für *ELIZA* ein Skript verwendet, welches die Ausdrucksweise eines Psychotherapeuten imitierte [19].

Shoebox, ein im Jahr 1961 von IBM entwickeltes Voice User Interface (VUI) gilt als eines der ersten Geräte, welches die Steuerung über Sprache ermöglichte. Dieses konnte hauptsächlich zu mathematischen Berechnungen verwendet werden. Eingesprochene, einfache mathematische Formeln, wurden dabei in elektrische Impulse umgewandelt, die mithilfe eines Messkreises klassifiziert und zur Berechnung in eine Rechenmaschine weitergeleitet wurden [20].

Im Jahr 1995 präsentierte Dr. Richard S. Wallace mit der *Artificial Linguistic Internet Computer Entity (A.L.I.C.E.)* ein System, welches den bisherigen überlegen war und eine

menschliche Konversation am ehesten imitieren konnte. Im Gegensatz zu *ELIZA* werden bei *A.L.I.C.E.* Informationen aus der Nutzereingabe mithilfe von Natural Language Processing (NLP) extrahiert und eine Konversation aufgrund von bestimmten Techniken der Mustererkennung aufgebaut [21].

SmarterChild, ein im Jahr 2001 von der Firma ActiveBuddy Inc. entwickelter Chatbot, war der erste Chatbot, der innerhalb von Messaging-Diensten wie AIM, MSN Messenger oder Yahoo Messenger funktionierte und deshalb große Popularität erlangte. Neben einfachen Gesprächen konnte *SmarterChild* auch Informationen wie z.B. Aktienkurse oder Wettervorhersagen liefern [22]. Weiterhin gilt *SmarterChild* als Wegbereiter für Apples *Siri* [23].

Mit der Einführung von Apples *Siri* [24] im Jahr 2011 wurden Conversational Interfaces auch im Consumer-Bereich immer bekannter und vermehrt genutzt. Zu den bekanntesten Vertretern gehören neben *Siri* auch Googles Assistant [25], Microsofts Cortana [26] sowie Amazons Alexa [27].

2.1.5 Chatbots als Conversational Assistants

In der Literatur werden Chatbots oft als *Conversational Agents* bezeichnet. Noessel [28, vgl. S.23] postuliert jedoch, dass diese Bezeichnung für Chatbots häufig nicht zutreffend ist und unterscheidet zwischen den Begriffen Agent und Assistent.

Agenten sind primär Technologien, die autonom agieren können. Im Bereich der Künstlichen Intelligenz wird zwischen einer Vielzahl von intelligenten Agenten unterschieden. So existieren reaktive Agenten, die einen Datenstrom nach bestimmten Auslösern für Aktionen überwachen und darauf reagieren. Weiterhin gibt es beispielsweise planende, zielbasierte oder lernende Agenten [29]. Ziel eines Agenten ist es, dem Nutzer zu helfen, sein eigenes Ziel zu erreichen. Dies geschieht beispielsweise durch das Ausführen von Aktionen und Aufgaben, die dem Nutzer abgenommen werden. Der Agent kann also selbst aktiv arbeiten und zu einem gewissen Maße Verantwortung übernehmen.

Assistenten hingegen sind Technologien, die den Nutzer bei der Ausführung einer Aufgabe unterstützen, ihm jedoch nicht eigenverantwortlich Aufgaben abnehmen.

Nach dieser Definition sind Chatbots eher den Assistenten und nicht den Agenten zuzuordnen. Nach Noessel sollte man sich also die Frage stellen, ob Conversational Interfaces wie Chatbots nicht besser als *Conversational Assistants* bezeichnet werden sollten [28].

2.1.6 Begriffe im Kontext des NLUs

Im Rahmen des NLUs in Chatbots wird bei der Verarbeitung von Texten häufig von Utterances, Intents und Entities gesprochen. Der Begriff Utterance steht dabei für eine einzelne, komplette Anfrage bzw. Äußerung der Nutzer. Intents sind die Absichten, die Nutzer mit dem gesamten Text erreichen wollen. Entities (Slots) repräsentieren hingegen einzelne Begriffe oder Objekte innerhalb der Anfrage, die den Intent genauer spezifizieren [30].

Wenn ein Nutzer im Kontext des HR-Chatbots für Bewerber also die Anfrage „Ich suche eine Stelle als Werkstudent im Bereich Software-Entwicklung“ stellt, dann ist der komplette Eingabetext die Utterance. Der Intent ist die Absicht eine Auflistung von verfügbaren Jobs zu erhalten mit den Entities „Werkstudent“ und „Software-Entwicklung“. Intents und Entities werden dabei im Entwicklungsprozess mit Namen definiert. In diesem Beispiel könnte der Intent *findExistingJob* sowie die Entities *formOfEmployment* (Werkstudent) und *desiredJob* (Software-Entwicklung) heißen.

2.2 Natural Language Processing und Understanding

Natural Language Processing (NLP) befasst sich mit Techniken zur Analyse für die Verarbeitung von Texten in natürlicher Sprache [31]. NLP ist ein interdisziplinäres Gebiet, das Computerlinguistik, Informatik, Kognitionswissenschaft und Künstliche Intelligenz kombiniert. Aus wissenschaftlicher Sicht zielt NLP darauf ab, die kognitiven Mechanismen zu modellieren, die dem Verständnis und der Produktion menschlicher Sprachen zugrunde liegen. Aus technischer Sicht beschäftigt sich NLP damit, wie man Computermodele und Anwendungen entwickelt, die die Bedeutung natürlicher Sprache verstehen und daraus Schlüsse ziehen können [32]. NLP bezieht sich auf alle Systeme, die eine Ende-

zu-Ende-Interaktion zwischen Mensch und Computer in natürlicher Sprache ermöglichen. Dieses Gebiet wird als kritischer Teil jeder Menschen zugewandten Künstlichen Intelligenz gesehen, da das System im Optimalfall gesprochene oder geschriebene Sprache aufnehmen und in seine Bestandteile zerlegen, sowie eine Bedeutung darin erkennen muss, um geeignete Aktionen festzulegen und in verständlicher Sprache zu reagieren [33]. Typische Anwendungen sind Spracherkennung, Dialogsysteme, lexikalische Analyse, maschinelle Übersetzung und Systeme zur Fragenbeantwortung sowie zur Stimmungsanalyse [34].

Die Begriffe Natural Language Understanding (NLU) bzw. Conversational Language Understanding (CLU) können verwendet werden, um die Aufgabe zu bezeichnen, natürliche Sprache innerhalb eines CUI-Systems zu verstehen [35]. Zu den zukünftigen Möglichkeiten der NLU könnte gehören, die formale Bedeutung der Inhalte natürlicher Sprache zu extrahieren und in maschinell ausführbare Programme zu transformieren [36]. Eine explizite Unterscheidung zwischen NLP und NLU/CLU ist in der Literatur nur selten zu finden. Die Begriffe werden daher oft synonym verwendet. In dieser Arbeit wird, wie auch in den evaluierten Services, im Weiteren ausschließlich der Begriff NLU verwendet.

Da sich diese Arbeit auf textbasierte Chatbots konzentriert, soll weiterhin auf die Funktion der NLU-Komponente in diesen eingegangen werden. Das Ziel der NLU-Komponente ist die Extraktion von Domäne, Intent und Entities [37]. Bei der Klassifikation der Domäne muss erkannt werden, in welcher Domäne sich der Nutzer inhaltlich bewegt. Dies spielt im vorliegenden Fall eines HR-Chatbots jedoch eine untergeordnete Rolle, da vorerst nur von domänenspezifischen Anfragen ausgegangen wird. Die zweite Aufgabe ist die Bestimmung des Intents. Dabei soll erkannt werden, welches Ziel der Nutzer mit seiner Nachricht verfolgt. Im Falle des HR-Chatbots könnte dies z.B. das Finden einer vorhandenen Stellenausschreibung oder die Suche nach bestimmten Informationen zum Bewerbungsprozess sein. Als dritte Aufgabe übernimmt die NLU-Komponente die Extraktion von einer oder mehreren relevanten Informationen (Entities) innerhalb des Textes. Dies könnte im vorliegenden Fall z.B. der eigene Name, die Bezeichnung einer Stelle oder Technologie sowie Orte oder Zeiten sein.

NLU-Services werden heutzutage von den führenden Anbietern meist API-basiert angeboten. Diese benötigen nur wenige nutzerspezifische Trainingsdaten, um erste Ergebnisse

zu erzielen. Zur Intent-Erkennung kommen dabei häufig Verfahren wie Logistische Regression, Support Vektor Maschinen oder Neuronale Netze zum Einsatz. Die Extraktion der Entities geschieht in Produktivsystemen oft mithilfe von musterbasierten Ansätzen wie Regular Expressions oder Machine-Learning-Verfahren wie Conditional Random Fields [17]. Aus der Forschung der letzten Jahre zeigt sich, dass sehr gute Ergebnisse mit Wort-Embeddings und Tiefen Neuronalen Netzen wie Convolutional Neural Networks oder Recurrent Neural Networks erzielt werden können [34]. Welche Techniken und Verfahren in den zu evaluierenden NLU-Services zum Einsatz kommen wird in Kapitel 3 näher beschrieben.

2.3 Maschinelle Lernverfahren im Kontext von Chatbots

In diesem Kapitel soll das Konzept bestimmter maschineller Lernverfahren erläutert werden, welche in den zu evaluierenden NLU-Services sowie der Chatbot-Implementierung dieser Arbeit verwendet werden.

2.3.1 Wort-Embeddings

Ein Wort-Embedding ist eine Funktion, die Worte in Vektoren eines n -dimensionalen Raumes abbildet. Diese Vektoren werden als Embeddings (deutsch: Einbettungen) bezeichnet, da Wörter bzw. Sätze dadurch so dargestellt werden, dass sie in einem Vektorraum eingebettet sind. Die Idee hinter dem Konzept der Wort-Embeddings ist, dass Wörter mit ähnlicher Bedeutung in ähnliche Vektoren abgebildet werden [37]. Abbildung 2 zeigt beispielhaft einen zweidimensionalen Vektorraum. Dabei ist zu erkennen, dass ähnliche Wörter an ähnlichen Stellen des Raumes abgebildet werden.

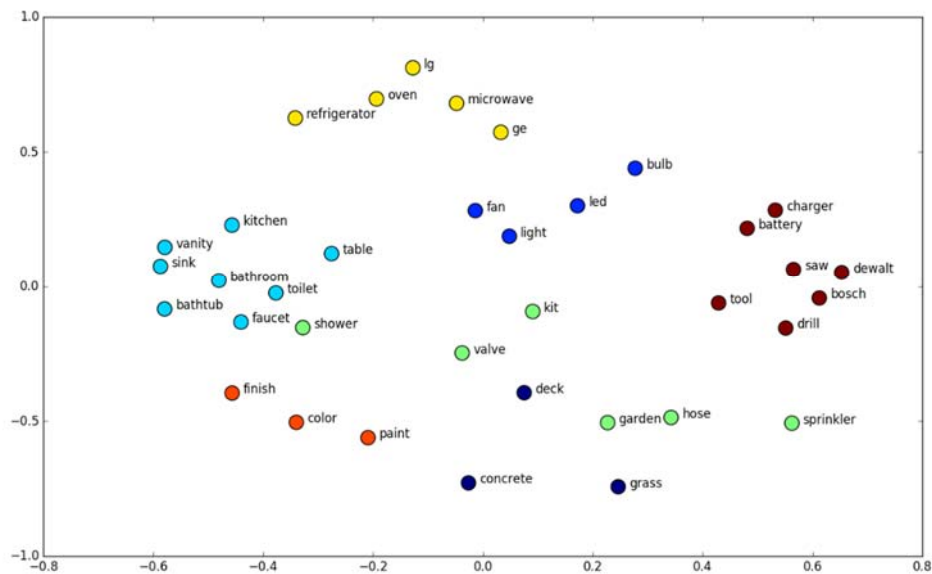


Abbildung 2: Beispiel für Wort-Embeddings [38]

2.3.2 Conditional Random Fields

Conditional Random Fields (CRFs) sind Modelle des maschinellen Lernens, welche die Fähigkeit besitzen, mehrere Variablen vorhersagen zu können, die voneinander abhängig sind. CRFs gehören zu den diskriminativen Klassifikatoren. Dies bedeutet, dass eine Entscheidungsgrenze zwischen verschiedenen Klassen modelliert wird. CRFs werden häufig für die Kennzeichnung oder das Parsen von sequentiellen Daten verwendet, weshalb sie häufig für die Verarbeitung natürlicher Sprache in CUIs zum Einsatz kommen [39].

2.3.3 Support Vektor Maschinen

Support Vektor Maschinen (SVMs) gehören ebenfalls zu den diskriminativen Klassifikatoren. Bei SVMs wird eine Entscheidungsgrenze zwischen verschiedenen Klassen gesucht, welche die Daten so trennt, dass der minimale Abstand eines Trainingselements zur Klassengrenze maximal wird. Diese Grenze ist abhängig von den nächstliegenden Vektoren, welche als Stützvektoren (englisch: *support vectors*) bezeichnet werden. Der dadurch entstehende Entscheidungsbereich impliziert eine bessere Generalisierung als eine einfache Klassengrenze [40]. Abbildung 3 zeigt beispielhaft eine Klassifizierung in

zwei Klassen, deren Instanzen als Punkte und Pluszeichen dargestellt werden. Die dicke Linie stellt die Klassengrenze dar, umkreiste Instanzen sind die Stützvektoren. Die gestrichelten Linien definieren die Ränder des Entscheidungsbereiches, welcher aufgrund der Stützvektoren entsteht.

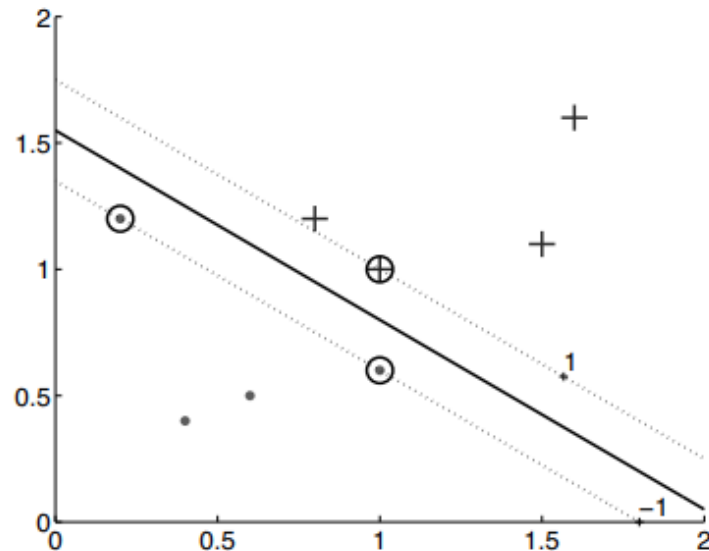


Abbildung 3: Beispiel einer SVM [40]

2.3.4 Neuronale Netze

Künstliche neuronale Netze werden den Strukturen des menschlichen Gehirns nachempfunden. Ein künstliches neuronales Netz kann als mathematisches Modell miteinander verbundener künstlicher Neuronen gesehen werden.

Das grundlegende Verarbeitungselement wird *Perzeptron* genannt. Es nimmt Eingaben $x_j \in \mathbb{R} < 1, \dots, d$ entgegen, die jeweils mit Verbindungsgewichten $w_{i,j} \in \mathbb{R}$ assoziiert sind. Diese Gewichte repräsentieren die Verbindungsstärke zwischen Neuronen. Jedes Neuron berechnet aus den Eingaben sowie den Verbindungsgewichten eine gewichtete Summe, welche anschließend an eine Aktivierungsfunktion $g()$ übergeben wird. Die Ausgabe y_i wird schließlich durch diese Aktivierungsfunktion bestimmt [40], [41]. Abbildung 4 zeigt das mathematische Modell eines einfachen Perzeptrons:

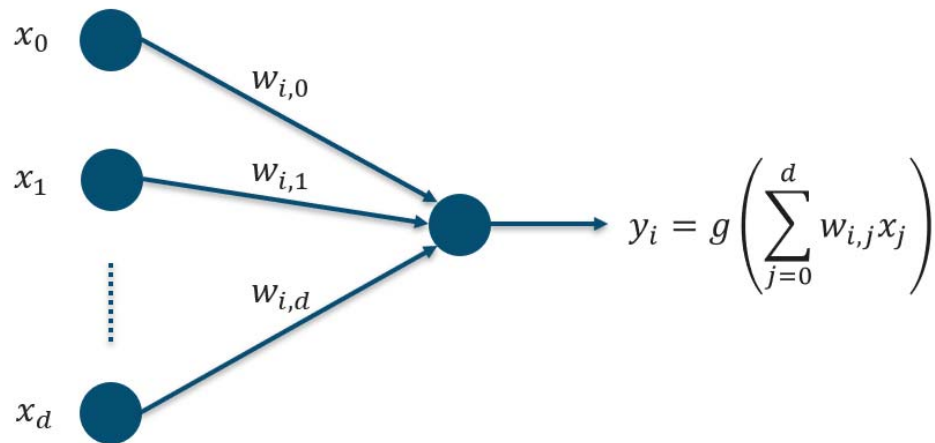


Abbildung 4: Perzeptron

Neuronale Feedforward-Netze sind eine Aneinanderreihung mehrerer Schichten aus Neuronen und besitzen keine Rückkopplung. Einschichtige Feedforward-Netze sind dabei die simpelste Form eines neuronalen Netzes. Diese besitzen lediglich eine Eingabe- und eine Ausgabeschicht. Mehrschichtige Feedforward-Netze besitzen neben diesen zwei Schichten eine Anzahl an Zwischenschichten, welche verdeckte Schichten genannt werden. Jede Schicht verwendet dabei die Ausgabe der vorherigen Schicht als Eingabe. Abbildung 5 zeigt beispielhaft ein Neuronales Netz mit zwei verdeckten Schichten. Dabei befinden sich drei Neuronen in der Eingabe- sowie ein Neuron in der Ausgabeschicht [40].

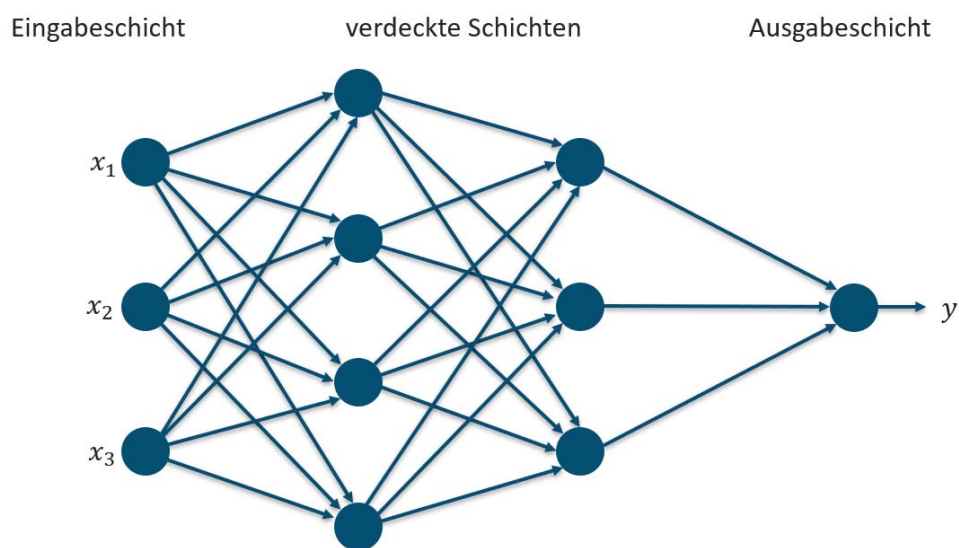


Abbildung 5: Beispiel eines mehrschichtigen Neuronalen Netzes

2.3.5 Rekurrente Neuronale Netze

Im Gegensatz zu Neuronalen Feedforward-Netzen können Ausgaben eines Neurons auf die eigene Eingabe oder die Eingabe einer vorangehenden Schicht gekoppelt werden. Dieses Verhalten bezeichnet man als Rekurrenz. Aufgrund dessen fungiert ein solches Netz als eine Art Kurzzeitgedächtnis, da Zustände aus vorigen Zeitpunkten in die Berechnung der Ausgabe einfließen können [40]. Abbildung 6 zeigt beispielhaft ein Rekurrentes Neuronales Netz, bei dem die Ausgabe der Neuronen der verdeckten Schicht auf die eigene Eingabe gekoppelt wird.

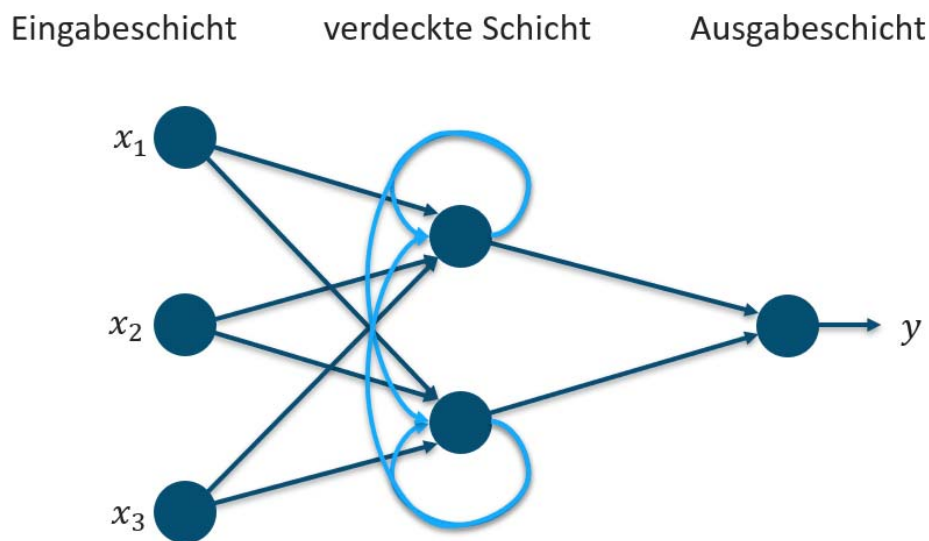


Abbildung 6: Beispiel eines Rekurrenten Neuronalen Netzes

2.3.6 Long Short-Term Memory Netzwerke

Das Problem von Feedforward-Netzen bzw. Rekurrenten Neuronalen Netzen besteht darin, dass Abhängigkeiten zwischen Instanzen nur über einen bestimmten Zeitraum gelernt werden können. Long Short-Term Memory (LSTM) -Netzwerke sind eine spezifische Art Rekurrenter Neuronaler Netze. Diese sind so konzipiert, dass sie den Kontext von Abhängigkeiten auch über einen längeren Zeitraum erfassen können. Sie besitzen interne Mechanismen, die Gates genannt werden, welche den Informationsfluss innerhalb des Netzwerks regeln. Diese Gates lernen, welche Informationen in einer Sequenz abhängiger Daten wichtig sind und somit weitergereicht werden und welche Informationen als unwichtig erachtet und deshalb verworfen werden können [42].

3 Evaluation

Um eine geeignete NLU-Komponente für die prototypische Umsetzung des HR-Chatbots zu finden, wurden vier Open Source Services zur Evaluation ausgewählt: *Snips NLU*, *Rasa NLU* mit Backend *spaCy* und *scikit-learn*, *Rasa NLU* mit Backend *TensorFlow* sowie *Mycroft Padatious*. Ziel dieser Services ist die Extraktion von strukturierten Informationen (Intents und Entities) aus den unstrukturierten Eingaben in natürlicher Sprache. Vorabkriterium für die Wahl der vier Services war die Möglichkeit Informationen aus deutschsprachigen Texten extrahieren zu können, da die allgemeine Kommunikation sowie Bewerbungsverfahren im Unternehmen hauptsächlich in deutscher Sprache stattfinden. Die Auswahl für einen der Services für die anschließende Umsetzung des HR-Chatbots geschieht anhand der Leistungsfähigkeit des Klassifikators mit domänenspezifischen Daten, welche in Kapitel 3.2 näher beschrieben werden.

Die Konzeption und die Durchführung der Evaluation orientierte sich am, von Braun et al., 2017 [43] vorgestellten Ansatz zur Evaluation von NLU-Services, welche auf die Notwendigkeit einer Evaluation mit domänenspezifischen Trainingsdaten hinweisen.

3.1 Übersicht der NLU-Services

3.1.1 Rasa NLU

Rasa NLU ist ein Open-Source-NLU-Tool zur Intent-Klassifizierung und Entity-Extraktion. Es umfasst mehrere NLP- und Machine-Learning-Bibliotheken, die miteinander kombiniert werden können. Dafür gibt es vordefinierte Pipelines mit Voreinstellungen, die für die meisten Anwendungsfälle geeignet sind. Eine Pipeline definiert, wie Benutzereingaben analysiert und tokenisiert, sowie Features extrahiert werden.

3.1.1.1 Rasa Pipeline mit Backend *spaCy* und *scikit-learn*

spaCy ist eine Open Source Python-Bibliothek, welche fortgeschrittene Natural Language Processing Funktionalitäten anbietet. Anwendung findet *spaCy* unter anderem, um

Informationsextraktions- oder NLU-Systeme zu entwickeln. Weiterhin kann spaCy zur Vorverarbeitung von Texten für Deep Learning Verfahren eingesetzt werden [44].

scikit-learn ist eine Open Source Machine Learning Bibliothek zu Zwecken des Data-Minings und der Datenanalyse. Sie stellt eine Reihe von überwachten und unüberwachten Lernalgorithmen zur Verfügung [45].

Der erste Schritt in der Pipeline ist die Tokenisierung des eingegebenen Textes mithilfe der spaCy NLP Bibliothek. Dabei wird dieser an allen Leerzeichen aufgeteilt und anschließend von links nach rechts verarbeitet. Für jede erhaltene Zeichenfolge wird dabei geprüft, ob es Ausnahmeregelungen in der entsprechenden Sprache gibt und ob Präfixe oder Suffixe abgespalten werden können. Im nächsten Schritt werden den einzelnen Token mithilfe eines statistischen Modells Wortarten zugeordnet (Part-of-speech (POS) Tagging) [46]. Jedes Token wird in einen Vektor abgebildet und anschließend zu einer Repräsentation des gesamten Satzes vereint. Der scikit-learn Klassifikator trainiert auf dieser Repräsentation aufbauend eine Support Vektor Maschine als Klassifikator für den Datensatz. Zur Entity-Extraktion kommt ein Conditional Random Field zum Einsatz, welches auf Basis der Token und der zugeordneten Wortarten trainiert wird [47].

Da die Evaluation aufgrund der Trainingsdaten in deutscher Sprache stattfinden soll, wird im Rahmen dieser Arbeit das von spaCy bereitgestellte deutsche Sprachmodell *de_core_news_sm* verwendet. Dieses Modell ist ein Convolutional Neural Network, das auf den *TIGER*- und *WikiNER*-Korpussen trainiert wurde [48].

Die Verwendung dieses Modells bietet den Vorteil, dass bereits ohne eigene Trainingsdaten auf eine große Anzahl an Vektoren zugegriffen werden kann. Gleichzeitig bedeutet dies jedoch auch, dass der Großteil dieser Daten nie Verwendung findet, wenn der NLU-Service, wie im Falle eines HR-Chatbots für Bewerber, hauptsächlich domänenspezifische Anfragen erhält.

3.1.1.2 Rasa Pipeline mit Backend TensorFlow

Mit der Veröffentlichung von Rasa NLU in der Version 0.12 wurde kurz nach Beginn dieser Arbeit eine neue Pipeline integriert, die auf *TensorFlow* basiert.

TensorFlow [49] ist eine Open Source Machine Learning Bibliothek für numerische Berechnungen mit Hilfe von Datenflussgraphen, welche ursprünglich vom Google Brains-Team entwickelt wurde. Die Knoten eines Graphen stellen dabei mathematische Operationen dar, während die Graph-Kanten mehrdimensionale Arrays aus Daten (Tensoren) darstellen, die zwischen ihnen fließen [50].

In der Pipeline mit TensorFlow werden im Gegensatz zur Pipeline aus Kapitel 3.1.1.1 keine vortrainierten Wortvektoren verwendet. Stattdessen werden Embeddings sowohl für die Wörter an sich, als auch für die Intents spezifisch für den bestehenden Datensatz gelernt [51]. Dabei wird jede Nutzereingabe und jeder Intent als Vektor einer reellen Zahl dargestellt. Die Embeddings werden daraufhin genutzt, um die Ähnlichkeit zwischen einer eingegebenen Utterance mit allen Intents zu vergleichen und darauf basierend eine Rangfolge zu erstellen, welcher Intent der Nutzereingabe im Vektorraum am ähnlichsten ist. Die Umsetzung dieser Technik in Rasa NLU basiert auf dem Konzept von *StarSpace* [52], einem von Facebook Research veröffentlichten neuronalen Modell, das auf eine Reihe von maschinellen Lernaufgaben angewendet werden kann. Ein Einsatzgebiet besteht in der Verarbeitung von Texten. Zur Entity-Extraktion kommt wie in der Pipeline mit Backend spaCy und scikit-learn ein Conditional Random Field zum Einsatz.

Ein Vorteil dieser Pipeline ist, dass nur Embeddings für Wörter und Intents aus den eigenen Trainingsdaten gelernt werden und somit der Speicherbedarf deutlich geringer ausfällt als bei der in Kapitel 3.1.1.1 gezeigten Pipeline mit vortrainierten Wortvektoren. Ein weiterer Vorteil dieses Konzeptes ist, dass dadurch ein Modell trainiert wird, das Ähnlichkeiten zwischen einzelnen Intents erkennen kann, da Embeddings nicht nur für Wörter oder Sätze, sondern auch für die Intents selbst gelernt werden [51]. Als Nachteil kann dabei gesehen werden, dass das Modell nur das Vokabular besitzt, welches in den Trainingsdaten vorkommt und dadurch die Gefahr besteht, dass unbekannte Wörter in den Eingaben der Nutzer vorkommen.

Bei der zum Zeitpunkt der Evaluation aktuellen Version 0.12.3 von Rasa NLU funktionierte die Pipeline mit Backend TensorFlow nur in Kombination mit spaCy. Da diese Abhängigkeit in der nächsten Version abgeschafft wurde, wurde für die Evaluation eine experimentelle Rasa Pipeline mit Backend TensorFlow implementiert, die bereits unabhängig von spaCy lief [53].

3.1.2 Snips NLU

Snips NLU ist ein Open-Source-NLU-Tool zur Intent-Klassifizierung und Entity-Extraktion des französischen Start-up-Unternehmens Snips SAS.

Snips NLU Pipeline beinhaltet zwei aufeinanderfolgende Intent-Parser: Einen deterministischen und einen probabilistischen Parser. Der deterministische Parser basiert auf dem Einsatz von Regular Expressions und stellt sicher, dass Nutzereingaben, die ohne Veränderung in den Trainingsdaten vorkommen, immer einen f1-score von 1 erreichen (siehe Kapitel 3.3.2). Wird ein Intent durch den deterministischen Parser erkannt, werden gleichzeitig die Entities daraus gezogen. Kann jedoch kein Intent zugeordnet werden, kommt der probabilistische Parser zum Einsatz, der wiederum in zwei aufeinanderfolgenden Schritten erst Intents und dann Entities erkennt. Die Intent-Erkennung erfolgt mithilfe logistischer Regression, zur Erkennung der Entities kommt wie bei Rasa NLU ein Conditional Random Field zum Einsatz. Unabhängig vom genutzten Intent-Parser werden in einem letzten Schritt die gefundenen Entities auf bestimmte vordefinierte Entities wie Zeiten, Temperaturen oder Nummer analysiert und können bei Bedarf formatiert werden [54].

3.1.3 Mycroft Padatious

Padatious ist ein Open Source Intent-Parser der Firma Mycroft AI. Er basiert auf der *Fast Artificial Neural Network* (FANN) Bibliothek [55] und läuft zur Zeit ausschließlich auf dem Betriebssystem Linux.

Zur Intent- und Entity-Erkennung kommen einfache Feedforward-Netze mit jeweils einer verdeckten Schicht zum Einsatz. Für jeden einzelnen Intent wird ein eigenes Modell trainiert, welches eine Wahrscheinlichkeit ausgibt, wie sehr ein eingegebener Satz dem Intent entspricht. Die Intent-Trainingsdaten werden dabei als kompletter Satz trainiert. Zur Entity-Extraktion kommen drei Feedforward-Netze zum Einsatz. Das erste Netz wird auf die Position der Entity innerhalb einer Nutzeräußerung trainiert, ein zweites Netz trainiert den Satz als Ganzes und ein drittes Netz wird darauf trainiert, wie gut der extrahierte Inhalt mit den vorher in einer separaten Datei definierten Entities übereinstimmt. Bei einer Nutzereingabe werden alle Intents gegen die Eingabe getestet und der Intent mit der höchsten Wahrscheinlichkeit sowie alle erkannten Entities zurückgegeben. Gibt es keinen

Intent, dessen Wahrscheinlichkeit über einem bestimmten Schwellenwert liegt, wird ausgegeben, dass kein Intent gefunden werden konnte. Neben dem Einsatz von neuronalen Netzen wird mit *padaos* [56] ein zweiter Parser eingesetzt, der mit Hilfe von Regular Expressions gewährleistet, dass genaue Übereinstimmungen mit den Trainingsdaten korrekt erkannt werden [57]–[59]. Im Gegensatz zu den anderen betrachteten Services müssen Intents und Entities für das Training getrennt voneinander in separaten Dateien gehalten werden. Die Beziehung zueinander wird in den Intent-Dateien über Platzhalter geschaffen. Ein Vorteil dabei ist, dass Intents mit passenden Entities im Training beliebig kombiniert werden und somit insgesamt mehr verschiedene Trainings-Utterances geschaffen werden. Nachteil dieses Ansatzes ist jedoch, dass gesammelte Trainingsdaten manuell in Intents und Entities aufgeteilt und die Beziehung zueinander dokumentiert werden muss.

3.2 Datensammlung und -verarbeitung

Zur Erhebung von authentischen Trainings- und Testdaten wurde ein Szenario entworfen, um herauszufinden, wie potenzielle Bewerber mit einem Chatbot interagieren würden.

Szenario: „Du bist gerade auf Jobsuche. Durch Internetrecherche bist du auf der Webseite eines potenziellen Arbeitgebers gelandet. Anstatt dich durch verschiedene Stellenanzeigen zu klicken und dich per E-Mail zu bewerben hast du die Möglichkeit den Bewerbungsprozess durch eine Konversation mit einem Chatbot durchzuführen.“

Im Kontext dieses Szenarios wurden auf einer Firmenkontaktmesse und über ein Online-Umfrage-Tool ein Datensatz mit Antworten bzw. Anfragen erstellt. Da die Interaktion mit einem Chatbot anders verlaufen kann als die verbale Kommunikation mit einem Menschen, erfolgten die Aussagen der Teilnehmer stets in schriftlicher Form über ein Formularfeld, um die Interaktion mit einem Chatbot nachzustellen. Der dadurch entstandene Datensatz umfasst 272 Utterances in deutscher Sprache, welcher vom Autor manuell gelabelt wurde. Er beinhaltet drei verschiedene Intents und acht Entities, welche im Folgenden genauer beschrieben werden.

Intent 1 (*introduction*) bezeichnet die Absicht der Nutzer sich beim Chatbot bzw. der Firma vorzustellen und erste Informationen zur Person zu geben. In den Antworten enthaltene Entities waren der Name des Bewerbers (*nameApplicant*), Studiengang (*universityCourse*) und Hochschule (*university*) sowie die aktuelle Arbeitsstelle (*currentJob*).

Aufgabe: „*Wie fängst du ein Gespräch mit dem Chatbot an, um dich kurz vorzustellen?*“

Beispiel-Antwort: „*Hallo, ich bin Marco und studiere Medieninformatik an der Hochschule der Medien Stuttgart.*“

Intent 2 (*findExistingJob*) bezeichnet die Absicht der Nutzer Informationen zu offenen Stellen zu erhalten. Enthaltene Entities waren der gewünschte Job (*desiredJob*), entweder spezifisch oder in einem bestimmten Bereich, sowie das Beschäftigungsverhältnis (*formOfEmployment*).

Aufgabe: „*Frage, ob du für eine von dir ausgewählte Tätigkeit in einem bestimmten Bereich in der Firma arbeiten kannst*“

Beispiel-Antwort: „*Gibt es offene Stellen im Frontend-Bereich als Werkstudent?*“

Intent 3 (*importanceOfExperience*) bezeichnet die Absicht der Nutzer herauszufinden, wie wichtig eine bestimmte geforderte Voraussetzung für eine spezifische Stelle ist, oder ob ihre eigene Erfahrung ausreicht. Enthaltene Entities waren die Menge an Erfahrung (*amountOfExp*), eine bestimmte Technologie (*technology*) sowie der gewünschte Job (*desiredJob*).

Aufgabe: „*Du hast eine konkrete Stelle gefunden, auf die du dich gerne bewerben möchtest. Leider erfüllst du nicht alle geforderten Voraussetzungen dafür. Wie sprichst du dies an?*“

Beispiel-Antwort: „*Ich habe nur wenig Erfahrung in Java. Kann ich mich trotzdem als Mobile Entwickler bewerben?*“

Jeder der 272 Utterances wurde nach Abschluss der Datensammlung ein spezifischer Intent zugeordnet. Die Entities wurden bei allen Services durch die Position innerhalb der

jeweiligen Utterance bestimmt. 101 Utterances des Datensatzes gehören zu Intent 1 (*introduction*), 109 zu Intent 2 (*findExistingJob*) und 62 zu Intent 3 (*importanceOfExperience*). Insgesamt befinden sich 426 Entity-Werte im Evaluations-Datensatz, aufgeteilt auf die vorherig genannten acht Entity-Typen. Tabelle 1 zeigt die Verteilung der Entity-Typen.

Entity	Anzahl im Datensatz
amountOfExp	33
currentJob	11
desiredJob	83
formOfEmployment	64
nameApplicant	95
technology	52
university	31
universityCourse	57

Tabelle 1: Entity-Typen im Evaluations-Datensatz

Dabei ist zu erkennen, dass die Entity-Typen nicht gleichmäßig verteilt sind. Die Entities *nameApplicant* und *desiredJob* kommen überdurchschnittlich häufig vor, während Entities wie *university*, *amountOfExp* oder *currentJob* deutlich seltener zu finden sind. Die ungleichmäßige Verteilung der Daten wurde zur Evaluation absichtlich gewählt. Braun et al. schlagen diese Vorgehensweise vor, um bewerten zu können, ob einige Services sehr häufige oder sehr seltene Entity-Typen besser verarbeiten als andere [43].

Da die Daten bei allen Services unterschiedlich formatiert sein müssen, wurden Intents und Entities für jeden Service einzeln gelabelt bevor mit der Evaluation begonnen werden konnte. Listing 1 zeigt beispielhaft die Formatierung einer Utterance für das Training des NLU-Service von Rasa NLU im JSON-Format.

```

{
  "text": "Hallo, ich bin Marco und studiere Medieninformatik an der
    Hochschule der Medien.",
  "intent": "introduction",
  "entities": [
    {
      "start": 15,
      "end": 20,
      "value": "Marco",
      "entity": "nameApplicant"
    },
    {
      "start": 34,
      "end": 50,
      "value": "Medieninformatik",
      "entity": "universityCourse"
    },
    {
      "start": 58,
      "end": 79,
      "value": "Hochschule der Medien",
      "entity": "university"
    }
  ]
}

```

Listing 1: Formatierung der Trainingsdaten für Rasa NLU

3.3 Evaluationsverfahren

Um die Leistung der einzelnen NLU-Services beurteilen zu können muss der Datensatz in zwei Teile geteilt werden: Einen Trainingsdatensatz, mit dem die Services trainiert werden und einen Testdatensatz, der sich vom Trainingsdatensatz unterscheidet. Da die Menge der verfügbaren Daten, wie im Falle dieser Arbeit, meist klein ist, ist eine Unterteilung in Teilmengen ausreichender Größe nicht möglich. Aus diesem Grund werden dieselben Daten wiederholt mit unterschiedlichen Aufspaltungen verwendet. Dieses Verfahren bezeichnet man als Kreuzvalidierung [40].

3.3.1 K-fache Kreuzvalidierung

Bei der K-fachen Kreuzvalidierung wird der komplette Datensatz X nach dem Zufallsprinzip in K gleichgroße Teile, $X_i, i = 1, \dots, K$, zerlegt. Einer der K Teile wird als Testdatensatz behalten, die restlichen $K - 1$ Teile werden kombiniert als Trainingsdatensatz verwendet. Dieses Vorgehen wird K Mal wiederholt, wobei jedes Mal ein anderer der K Teile als Testdatensatz verwendet wird. Die Leistung wird anhand des Durchschnitts aller Iterationen berechnet. [40] Im Falle dieser Evaluation wurde eine 5-fache Kreuzvalidierung angewendet. Der Datensatz wurde also in fünf gleichgroße Teile zerlegt und die Leistung wurde über fünf Iterationen getestet (vgl. Abbildung 7).

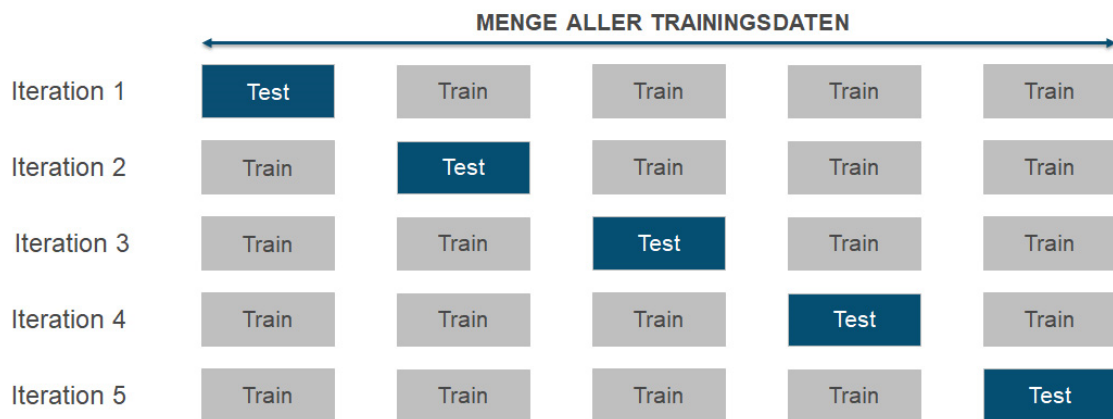


Abbildung 7: K-fache Kreuzvalidierung

3.3.2 Leistungsmetriken

Die Leistungsfähigkeit der Klassifikatoren der einzelnen NLU-Services wird nicht anhand der Anzahl der korrekt klassifizierten Daten (accuracy), sondern anhand der Metriken *precision*, *recall* und *f1-score* bestimmt.

Um precision, recall und f1-score bestimmen zu können, wird die Anzahl der richtig positiven (true positives = TP), bzw. falsch positiven (false positives = FP) und richtig negativen (true negatives = TN) bzw. falsch negativen Vorhersagen (false negatives = FN) der jeweiligen NLU-Services benötigt [60].

Um diese Werte zu erhalten, werden bei einem Klassifikationsproblem, bei dem die Ausgabe eine von mehreren Klassen sein kann, die tatsächliche Klasse mit der vorhergesagten Klasse verglichen. Im Kontext dieser Evaluation wird die Klassifikation von Intents und Entities voneinander getrennt betrachtet. Dabei gilt jeder Intent bzw. jede Entity als eine eigene Klasse.

Dies soll beispielhaft an der Klassifikation eines Intents mit der Bezeichnung *Intent A* erläutert werden:

- **TP:** Ein Intent A, der korrekterweise als Intent A klassifiziert wurde.
- **TN:** Alle anderen Intents, die korrekterweise nicht als Intent A klassifiziert wurden.
- **FP:** Andere Intents, die fälschlicherweise als Intent A klassifiziert wurden.
- **FN:** Ein Intent A, der fälschlicherweise nicht als Intent A klassifiziert wurde.

Die Kalkulation dieser Werte wird für jede Klasse wiederholt. Im Fall dieser Klassifikation werden die Werte also für jeden Intent bzw. für jede Entity definiert.

Der Precision-Wert gibt an, wie viele der erkannten Klasse-*i*-Instanzen tatsächlich Klasse-*i*-Instanzen sind.

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

Der recall-Wert gibt an, wie viele der Klasse-*i*-Eingaben als Klasse-*i*-Instanzen erkannt werden.

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

Der f1-score ist das harmonische Mittel aus precision und recall, wobei der bestmögliche Wert bei 1 und der schlechteste Wert bei 0 liegt.

$$f1 = 2 \frac{precision \times recall}{precision + recall}$$

3.4 Evaluationsergebnisse

Alle NLU-Services wurden mit dem in Kapitel 3.2 vorgestellten Datensatz trainiert und wie in Kapitel 3.3.1 und Kapitel 3.3.2 beschrieben evaluiert. Die Metriken wurden für jede Klasse einzeln ermittelt. Anschließend wurde der Durchschnitt über alle Klassen gebildet, so dass sich pro Service jeweils ein Endergebnis für Intents (\sum Intents) und ein Ergebnis für Entities ergab (\sum Entities). Als ausschlaggebendes Kriterium für die Verwendung eines Services für die folgende prototypische Umsetzung wurde der berechnete f1-score herangezogen. Die Ergebnisse der Evaluation sind in Tabelle 2 bis Tabelle 5 dargestellt.

Rasa Pipeline mit Backend spaCy und scikit-learn

Intent	precision	recall	f1-score
introduction	0,988	0,938	0,96
findExistingJob	0,896	0,962	0,926
importanceOfExperience	0,936	0,866	0,896
\sum Intents	0,938	0,932	0,935
Entities	precision	recall	f1-score
\sum Entities	0,922	0,936	0,929

Tabelle 2: Evaluation Rasa NLU Pipeline 1

Rasa Pipeline mit Backend TensorFlow

Intent	precision	recall	f1-score
introduction	0,99	0,99	0,99
findExistingJob	0,938	0,99	0,962
importanceOfExperience	0,988	0,884	0,93
Σ Intents	0,97	0,966	0,968
Entities	precision	recall	f1-score
Σ Entities	0,928	0,934	0,931

Tabelle 3: Evaluation Rasa NLU Pipeline 2

Snips NLU

Intent	precision	recall	f1-score
introduction	1	0,95	0,97
findExistingJob	0,99	0,936	0,962
importanceOfExperience	0,936	0,952	0,944
Σ Intents	0,98	0,945	0,962
Entities	precision	recall	f1-score
Σ Entities	0,885	0,949	0,916

Tabelle 4: Evaluation Snips NLU

Mycroft Padatious

Intent	precision	recall	f1-score
introduction	0,95	0,95	0,95
findExistingJob	0,667	0,9	0,766
importanceOfExperience	0,857	0,429	0,572
Σ Intents	0,825	0,76	0,791
Entities	precision	recall	f1-score
Σ Entities	0,429	0,386	0,406

Tabelle 5: Evaluation Mycroft Padatious

Aus den Ergebnissen wird ersichtlich, dass Rasas NLU-Service mit beiden Pipelines trotz einer geringen Menge an Trainingsdaten gute Ergebnisse erzielt. Der schlechteste f1-score liegt demnach für den Intent *importanceOfExperience* bei 0,896. Weiterhin zeigt sich, dass die Pipeline mit Backend TensorFlow bei gleichen Bedingungen für alle Intents höhere Werte erzielen konnte.

Die Evaluation von Snips NLU zeigt in den durchschnittlichen f1-scores nur geringe Unterschiede zu den Werten von Rasas Pipeline mit Backend TensorFlow. Der größte Unterschied konnte beim Intent *importanceOfExperience* gefunden werden. Während bei Rasa ein höherer Precision-Wert von 0,988 und niedrigerer Recall-Wert von 0,884 erzielt wurde, zeigt sich bei Snips ein Precision-Wert von 0,936 und ein Recall-Wert von 0,952. Dies bedeutet, dass bei Rasa nahezu alle als Intent *importanceOfExperience* erkannten Intents tatsächlich diesem Intent entsprachen, jedoch insgesamt weniger jener Intents gefunden wurden. Bei Snips NLU wurden insgesamt mehr tatsächlich vorhandene Intents *importanceOfExperience* erkannt, dafür allerdings auch mehr Intents als solche klassifiziert, die eigentlich andere Intents waren.

Bei den durchschnittlichen f1-scores der Entities konnten mit den Werten 0,929, 0,931 und 0,916 bei den bisher genannten drei Services nur geringe Unterschiede festgestellt werden.

Andere Ergebnisse zeigten sich bei der Evaluation von Mycrofts NLU-Service Padatious. Während der f1-score beim Intent *introduction* mit einem Wert von 0,95 sehr gut erscheint, konnten bei den anderen Intents nur f1-scores von 0,766 und 0,572 erzielt werden, was einem durchschnittlichen f1-score über alle Intents von 0,791 entspricht. Auch ein durchschnittlicher f1-score von 0,406 über alle Entities zeigt einen deutlichen Leistungsunterschied zu allen anderen evaluierten Services. Da die Leistungsfähigkeit des Klassifikators als Hauptkriterium für die Auswahl eines Services dient, wurde Mycroft Padatious in der weiteren Analyse und Evaluation nicht weiter betrachtet.

Da die durchschnittlichen f1-scores bei Rasas NLU-Service mit Backend TensorFlow sowohl für Intents als auch für Entities am höchsten ausfielen, fiel die Entscheidung für ebendiesen Service. Weitere Kriterien, die für eine Implementierung des Chatbot-Prototyps mithilfe von Rasa NLU sprachen, waren höhere Precision-Werte bei allen Entities, der modulare Aufbau des Services, sowie die Möglichkeit des Erkennens mehrerer Intents in einer Nutzereingabe. Dies soll im Folgenden näher erläutert werden:

1. Höhere Precision-Werte bei allen Entities: Zwar wurde bei den Ergebnissen von Snips NLU durchschnittlich ein minimal höherer Precision-Wert für alle Intents erreicht, doch die Werte für alle Entities lagen bei Rasa NLU merklich höher. Für die Implementierung des HR-Chatbots ist es wichtig, dass erkannte Entities wie Namen, Technologien oder Jobbezeichnungen auch wirklich dem entsprechen, was der Nutzer meint. Der HR-Bot sollte den Nutzer lieber nicht mit seinem Namen ansprechen als mit einem falschen Namen. Eine falsche Klassifikation einer Technologie führt bei der Stellensuche mit hoher Wahrscheinlichkeit zu einer unpassenden Stelle, die dem potenziellen Bewerber angeboten wird. Ein Nicht-Erkennen einer Entity, welches zu einer Rückfrage des Chatbots führt, erscheint in dieser Situation als eine bessere Alternative.

2. Modularität: Da Rasa NLU mehrere Machine-Learning-Bibliotheken umfasst, die miteinander kombiniert werden können, bietet dieser Service eine höhere Flexibilität und Anpassungsfähigkeit für den weiteren Verlauf dieser Thesis sowie für mögliche nachfolgende Projekte, die auf dieser Arbeit aufbauen. Ein Austauschen einzelner Kom-

ponenten innerhalb der NLU-Pipeline ist genauso möglich wie die komplette Umstellung des Backends auf eine andere Pipeline wie spaCy und scikit-learn oder in Zukunft integrierte Komponenten.

3. Erkennen mehrerer Intents in einer Nutzereingabe: Die auf TensorFlow basierende Pipeline von Rasa NLU ermöglicht es, Modelle zu trainieren, die einer einzigen Eingabe mehrere Intents zuweisen können [61]. Snips NLU plant ebenfalls eine Integration dieser Funktion, zum Zeitpunkt der Evaluation besteht diese Möglichkeit jedoch nicht [62].

4 Entwurf des Chatbot-Prototyps

4.1 Szenario und Umfang des HR-Chatbots

Kernfunktion des Chatbot-Prototyps soll die Suche nach einer Stelle in Form eines Dialoges sein. So soll zum einen direkt nach einem vorhandenen Job gefragt werden können, wenn der potenzielle Bewerber bereits eine konkrete Vorstellung einer möglichen Stelle hat. Dabei hat er die Möglichkeit direkt nach einem konkreten Jobtitel zu fragen. Gibt es eine offene Stellenanzeige, die der Anfrage anhand der gegebenen Informationen eindeutig zugeordnet werden kann, soll diese dem Nutzer direkt vermittelt werden und ihm die Möglichkeit gegeben werden, weitere Informationen dazu zu erhalten. Alternativ kann der potenzielle Bewerber nach Stellen suchen, ohne initial eine konkrete Stelle vor Augen zu haben. Ziel dabei ist es, mit möglichst wenigen Fragen alle benötigten Informationen zu erhalten und überflüssiges Nachfragen zu Wünschen und Vorstellungen des Bewerbers zu vermeiden. Die Umsetzung dieser Funktionalität wird in Kapitel 4.2 erläutert.

4.2 Intent Jobsuche

4.2.1 Szenario

Das Kernszenario des Chatbot-Prototyps beinhaltet die Unterstützung eines potenziellen Bewerbers bei der Jobsuche. Dabei wird prinzipiell zwischen zwei Arten des Suchens unterschieden. Besitzt der Bewerber eine konkrete Vorstellung seines Jobs, kann er direkt nach diesem anhand dessen Jobbezeichnung fragen und erhält, falls eine passende Stellenanzeige in der Datenbank vorhanden ist, Informationen zur gewünschten Stelle als Antwort angezeigt.

Beispiel: „*Ich bin auf der Suche nach einem Job als DevOps Spezialist Senior*“

Erkennt der Chatbot keine eindeutige Jobbezeichnung in der Nachricht, wird versucht einzelne Informationen in Form von Entities aus der Nachricht zu extrahieren, die weiterverwendet werden können. Anschließend soll der Nutzer die Möglichkeit angeboten bekommen, nach ähnlichen Stellen zu suchen. Alternativ kann die Stellensuche auch ohne initiale Angaben zu einem Job gestartet werden.

Beispiel: „Welche offenen Stellen gibt es in ihrem Unternehmen?“

Ziel des Chatbots ist es anschließend, mit Hilfe der bereits erkannten Entities oder ohne vorherige Informationen, mit möglichst wenigen Fragen alle weiteren Informationen, die zum Anzeigen einer geeigneten Stelle nötig sind, vom Nutzer zu erhalten. Anschließend folgt die Frage nach der Vertragsform. Gibt es für die angegebene Vertragsform eine Stelle, wird diese daraufhin angezeigt. Konnte kein passender Job gefunden werden, wird auf weitere Möglichkeiten zu Bewerbungen hingewiesen.

Abbildung 8 zeigt beispielhaft die Konzeption möglicher Routen im Gesprächsverlauf beim Aufrufen des Intents *find_job*:

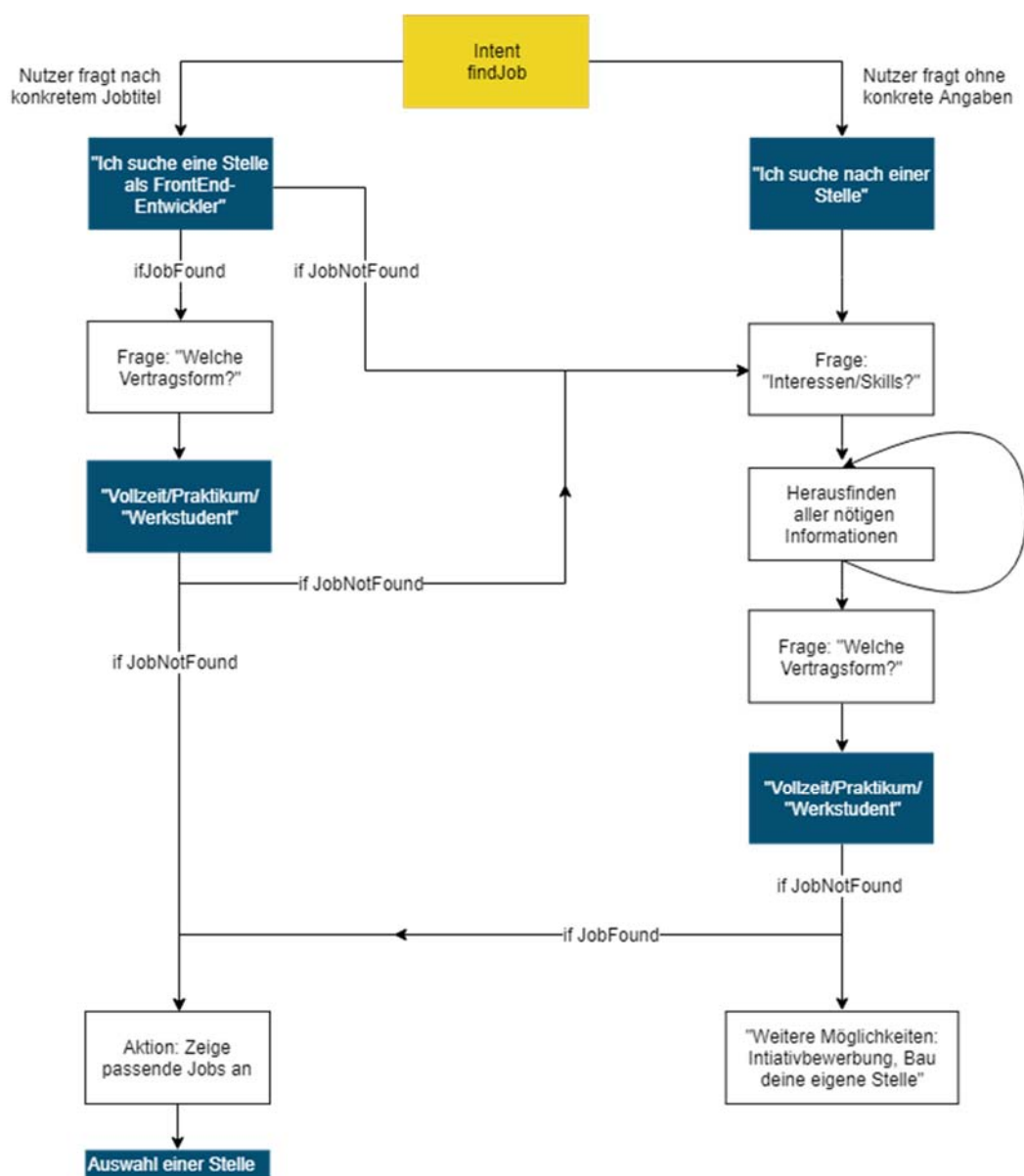


Abbildung 8: Intent Jobsuche

4.2.2 Datenstruktur eines Jobs

Um geeignete Stellen für einen Bewerber als Antwort liefern zu können, wurde eine Datenstruktur für Jobs erarbeitet. Dazu wurde ein Konzept entwickelt, wie anhand von Konversation Stellenbeschreibungen gefunden und zugeordnet werden können. Ein Job wird in diesem Prototyp immer von den Attributen *Task*, *Domain* und *Technology* charakterisiert und kann zusätzlich durch weitere Attribute wie z.B. *formOfEmployment* (Beschäftigungsverhältnis) erweitert werden. Die Attribute wurden anhand der vorhandenen Stellenbeschreibungen und Tätigkeitsbereiche der adorsys GmbH & Co. KG definiert.

Ein *Task* steht für die Haupttätigkeit, die bei einem Job voraussichtlich ausgeführt wird und wird als Verb definiert. Mögliche Tasks sind in diesem Szenario „entwickeln“, „designen“, „administrieren“, „coachen“ und „analysieren“.

Domains bezeichnen einzelne Bereiche innerhalb eines Unternehmens, die sich voneinander abgrenzen lassen und können mindestens einem Task zugeordnet werden. Domains sind unter anderem „Frontend“, „Backend“, „Machine Learning“ oder „Finanzen“, wobei insgesamt zwischen 20 Domains unterschieden wird.

Technologies sind Technologien, die in bestimmten Tasks oder Domains eingesetzt werden. Technology steht dabei nicht nur für Technisches wie Programmiersprachen und Programme, sondern beinhaltet auch bestimmte Methoden oder Herangehensweisen, die in einem Job angewandt oder ausgeführt werden. Beispiele hierfür sind z.B. „Java“, „Python“, „Photoshop“, „Systemadministration“ oder „Scrum“. Für diesen Prototyp wurden 151 verschiedene Technologien herausgesucht und implementiert.

Im Gespräch mit dem Chatbot sollte es nicht zwingend notwendig sein die Informationen in einer bestimmten Reihenfolge zu nennen. Eine Konversation kann also beispielsweise auch beginnen, indem ein Bewerber schreibt, dass er gerne mit einer bestimmten Technologie wie „Java“ arbeiten würde. Bei weiteren Fragen des Bots sollte dieser auf die Antwort des Nutzers Bezug nehmen und bereits bestimmte Tasks und Domains ausschließen können. Um diese Funktionalität zu erreichen müssen die Beziehungen der einzelnen Teile der Datenstruktur eines Jobs zueinander definiert werden. Diese Attribute einer Stellenanzeige können als Abstraktionsebenen gesehen werden, um Assoziationen aufzubauen.

Vor diesem Hintergrund wurden alle aktuellen sowie ausgewählte alte Stellenanzeigen der adorsys GmbH & Co. KG anhand dieser Kriterien untersucht und für die Implementierung formatiert. Dazu wurde eine Vorlage angefertigt, mit welcher eine Zuordnung aller Abstraktionsebenen stattfinden konnte (siehe Anhang A.1). Dies geschah in Zusammenarbeit mit Mitarbeitern der adorsys GmbH & Co. KG. Eine Auflistung aller Stellenanzeigen mit allen zugeordneten Abstraktionsebenen befindet sich in der Datei *actions.py*, welche auf der CD im Ordner *hr-bot/app/actions* zu finden ist.

5 Implementierung des Chatbot-Prototyps

Dieses Kapitel beschreibt die prototypische Umsetzung eines Chatbots für Bewerber zur Unterstützung des HR-Teams. Dazu wurde aufgrund der Evaluationsergebnisse aus Kapitel 3.4 ein NLU-Service auf Basis von Rasa NLU implementiert, der in Kombination mit Rasa Core als Dialogmanagement-Komponente den Kern des Chatbots bildet.

5.1 Architektur

In diesem Kapitel soll eine Übersicht über die Architektur des Chatbot-Prototypen sowie dessen Umsetzung erläutert werden. Details zu einzelnen Komponenten folgen in weiteren Kapiteln. Prinzipiell kann man sechs Bausteine herausarbeiten, die für die Abarbeitung einer Nachricht zusammenspielen:

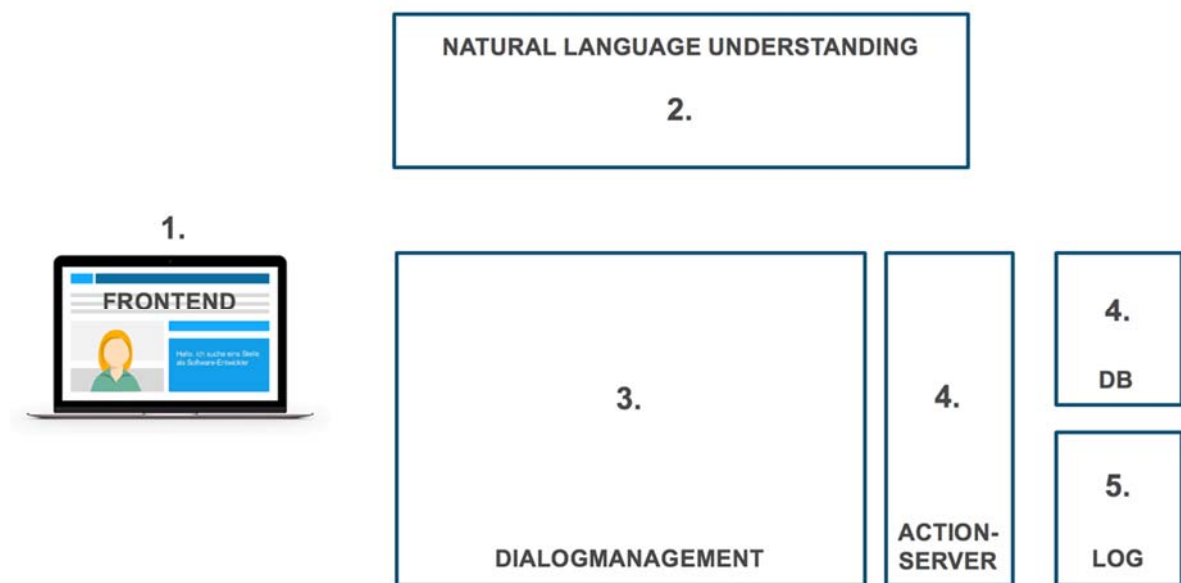


Abbildung 9: Architektur des Chatbot-Prototyps

1. Nachrichten können vom Nutzer über ein Chat-Interface in einem Web-Frontend eingegeben werden. Kapitel 5.2 beschreibt die Umsetzung der Frontend-Komponente.

2. Die NLU-Komponente verarbeitet die unstrukturierten Daten aus der Nachricht des Nutzers und extrahiert daraus strukturierte Daten in Form von Intents und Entities. Eine Beschreibung der NLU-Komponente folgt in Kapitel 5.3.
3. In der Dialogmanagement-Komponente wird aus den erkannten Intents und Entities der aktuellen Eingabe, sowie aus Daten der letzten Aktion, die vom Chatbot ausgeführt wurde und allen bisher gespeicherten konstanten Variablen (Slots), die beispielweise vom Server gesetzt wurden, eine Vektordarstellung des Gesprächszustandes geformt. Dieser Vektor wird daraufhin als Eingabe für eine oder mehrere Policies verwendet, welche für die Voraussage der nächsten Aktion verantwortlich sind. Erklärungen und Details zur Implementierung der Dialogmanagement-Komponente folgen in Kapitel 5.4.
4. Je nach Art der vorhergesagten Aktion wird entweder eine Antwort an den Nutzer gesendet oder weitere Aktionen auf einem Server ausgeführt, bevor eine Nachricht an das Frontend zurückgegeben wird. Eine Beschreibung des Action-Servers erfolgt in Kapitel 5.5
5. Die Logging-Komponente bietet eine detaillierte Möglichkeit zur Speicherung von Gesprächsinformationen und wird in Kapitel 5.6 erläutert.

5.2 Umsetzung der Frontend-Komponente

Für die Implementierung des Chatbot-Frontends wurde *rasa-webchat* [63], ein Chat-Widget, welches mit der JavaScript-Softwarebibliothek *React* [64] umgesetzt ist, verwendet. Dieses wurde so angepasst, dass es zum Design der Webseite der adorsys GmbH & Co. KG passt (siehe Abbildung 10). Über das WebSocket-Protokoll wird mithilfe der JavaScript-Softwarebibliothek *Socket.IO* [65] eine bidirektionale Client-Server-Kommunikation ermöglicht. Im Backend wird dazu ein WebSocket bereitgestellt, mit welchem sich das Chatbot-Frontend verbinden kann.



Abbildung 10: Chatbot-Widget auf der Webseite

5.3 Umsetzung der NLU-Komponente

Diese Sektion beschreibt die Implementierung der NLU-Komponente im Chatbot-Prototyp. Abbildung 11 zeigt die Einordnung dieser Komponente in die Chatbot-Architektur:

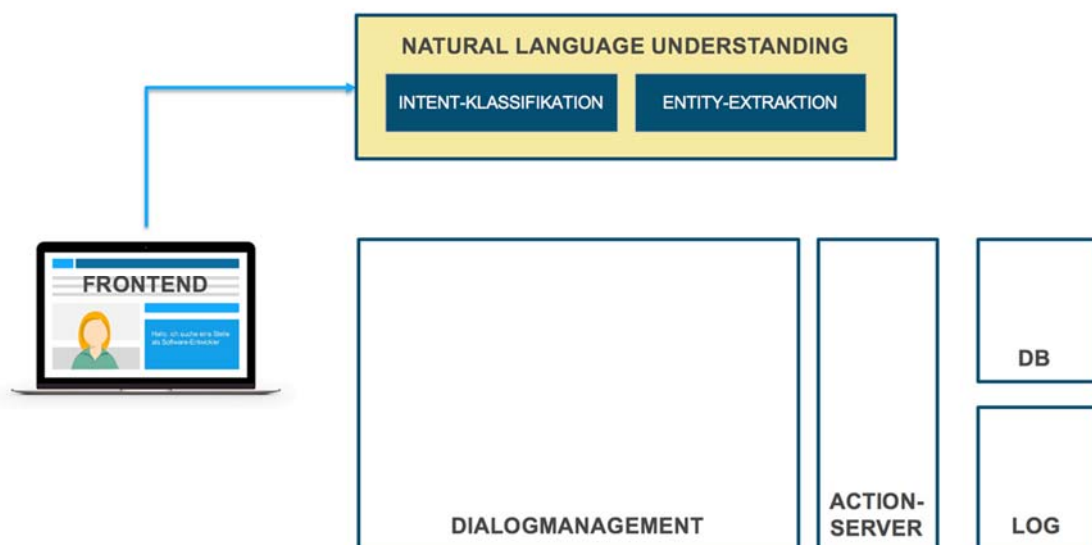


Abbildung 11: Architektur des Chatbot-Prototyps mit NLU-Komponente

5.3.1 Pipeline

Da Rasa NLU die Möglichkeit bietet mehrere NLP- und Machine-Learning-Bibliotheken miteinander zu kombinieren, ist es erforderlich eine Pipeline mit allen Komponenten, die verwendet werden sollen, zu definieren. Dabei können sowohl von Rasa bereitgestellte als auch eigene Komponenten integriert werden, die sequentiell eingehende Nachrichten verarbeiten, strukturierte Informationen daraus ziehen und ihre Ausgabe an die nächste Komponente übergeben. Abbildung 12 zeigt, in welcher Reihenfolge die einzelnen Bestandteile der Komponenten aufgerufen werden:

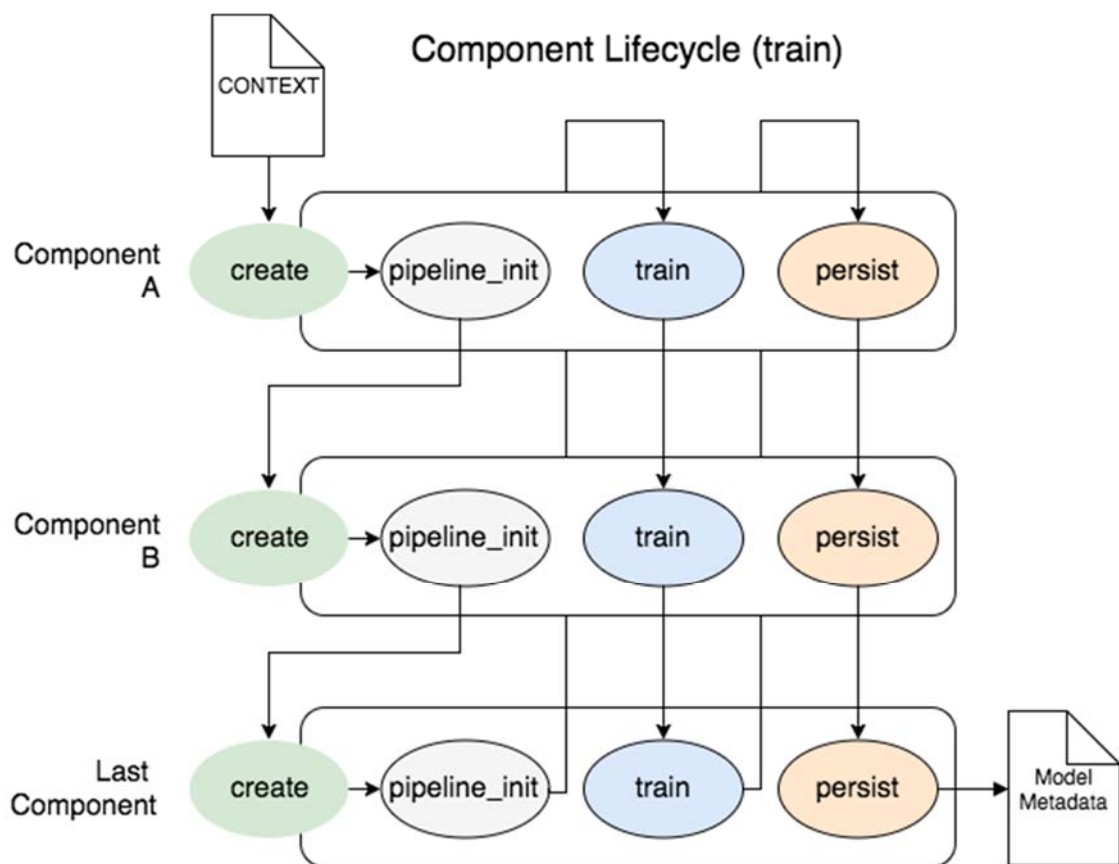


Abbildung 12: Rasa Component Lifecycle [61]

Wie gezeigt, wird initial der Kontext erzeugt. Der Kontext ist ein Python-Dictionary, welches genutzt wird, um Daten zwischen den einzelnen Komponenten zu übertragen. Anschließend werden alle Komponenten erzeugt und mit Konfigurationswerten initialisiert. Diese Werte werden entweder aus der Standardkonfiguration entnommen oder können in der Pipeline-Konfiguration definiert werden. Wenn alle Komponenten initialisiert sind,

folgen die Trainings- und Persistenzphase. Die Trainingsdaten müssen dabei, wie in Listing 1 gezeigt, formatiert sein. Nachdem alle Komponenten durchlaufen wurden wird der Kontext dazu genutzt, um die Metadaten des Modells zu speichern [60].

Für die Implementierung dieses Chatbot-Prototyps wurde eine Pipeline umgesetzt, welche hauptsächlich auf den in Kapitel 3.1.1.2 erläuterten Embeddings zur Intent-Klassifikation aufbaut. Listing 2 zeigt dessen Konfiguration.

```
pipeline:
- name: tokenizer_whitespace
- name: intent_entity_featurizer_regex
- name: ner_crf
- name: ner_synonyms
- name: intent_featurizer_count_vectors
- name: intent_classifier_tensorflow_embedding
  intent_tokenization_flag: true
  intent_split_symbol: "+"
```

Listing 2: Rasa NLU Pipeline Konfiguration (*config_tensorflow.yml*)

Der erste Schritt in der Pipeline ist ein Whitespace-Tokenizer, der Tokens für jede, durch ein Leerzeichen getrennte, Zeichenfolge erstellt, welche als Grundlage zur Weiterverarbeitung dienen.

Der *intent_entity_featurizer_regex* erstellt eine Liste von Regular Expressions. Für jede, in den Trainingsdaten definierte, Regular Expression wird dabei untersucht, ob diese in der Nutzereingabe gefunden werden kann. Regular Expressions werden innerhalb dieses Prototyps ausschließlich in Form sogenannter *Lookup Tables* verwendet. Ein Lookup Table enthält alle bekannten Werte, die von einer Entity erwartet werden können. So existiert ein Lookup Table für Technologien, welcher die Namen aller Technologien, die erkannt werden sollen, enthält. Bei der Ausführung der Komponente werden die, im ersten Schritt entstandenen, Token untersucht, ob eine Übereinstimmung mit einem der Werte innerhalb eines Lookup Tables vorliegt und dementsprechend markiert. Der Einsatz dieser Tables bietet den Vorteil, dass Entities besser erkannt werden können, auch wenn sie in den Trainingsdaten nicht gelernt wurden [66].

Anschließend kommt mit der *ner_crf*-Komponente ein Conditional Random Field zur Entity-Extraktion zum Einsatz. Dabei wird aus den, durch die vorherigen Komponenten vorverarbeiteten, Trainingsdaten ein Modell trainiert, welches zur Erkennung der Entities in den Eingaben der Nutzer verwendet wird. Die *ner_synonyms*-Komponente ordnet synonymen Entities dieselben Werte zu. Dazu werden innerhalb der Trainingsdaten mehrere Arrays mit Begriffen definiert, die synonym behandelt werden sollen. Die, durch das Conditional Random Field erkannten, Entities werden daraufhin auf diese Synonyme untersucht und im positiven Fall durch entsprechende Werte ersetzt. Dies spielt vor allem im Szenario der Stellensuche eine wichtige Rolle, da aufgrund der erkannten Entities die Datenstruktur eines Jobs aufgebaut wird, um damit nach passenden Stellen in der Datenbank zu suchen. Einzelheiten zur Datenstruktur und dessen Umsetzung im Prototypen folgt in Kapitel 5.5.

Im folgenden Schritt kommt der *intent_featurizer_count_vectors* zum Einsatz. Ein Featurizer wandelt die einzelnen Token in Features um, die von maschinellen Lernalgorithmen verwendet werden können. Welche Token als Feature erkannt und welche ausgeschlossen werden sollen, kann beim Initialisieren der Komponente definiert werden. Unter Verwendung der *CountVectorizer*-Funktion von scikit-learn wird die Anzahl der Vorkommen jedes Tokens innerhalb des Vokabulars analysiert und eine Bag-of-Words-Repräsentation von Features erstellt [67]. Ein einzelner Intent wird demnach als Vektor anhand der Häufigkeit der darin enthaltenen Features dargestellt.

Als Intent-Klassifikator kommt anschließend der *intent_classifier_tensorflow_embedding* zum Einsatz. Dabei werden Embeddings sowohl für die im vorherigen Schritt extrahierten Features als auch für die Intents als Summe aller enthaltenen Features für den bestehenden Datensatz gelernt. Intents, die ähnlich erscheinen, besitzen im Vektorraum eine geringere Distanz voneinander als Intents, die weniger ähnlich erscheinen. Die Embeddings werden genutzt, um die Ähnlichkeit zwischen einer eingegebenen Utterance mit allen im Training gelernten Intents zu vergleichen und darauf basierend eine Rangfolge zu erstellen, welcher Intent der Nutzereingabe im Vektorraum am ähnlichsten scheint. Die Nutzung der TensorFlow Embeddings als Intent-Klassifikator bringt einen weiteren Vorteil mit sich: Mehrere Intents können innerhalb einer Nachricht erkannt werden. Vor allem bei der Begrüßung könnte ein weiterer Intent, wie beispielsweise der Grund für die

Kontaktaufnahme, angehängt werden. Durch die in Listing 2 gezeigten Parameter *intent_tokenization_flag* und *intent_split_symbol* wird diese Funktion aktiviert und bestimmt, wie entsprechende Utterances als Multi-Intents in den Trainingsdaten definiert werden.

5.4 Umsetzung der Dialogmanagement-Komponente

Die folgende Sektion beschreibt die Umsetzung des Dialogmanagements im Chatbot. Dieses wurde mit dem Open-Source-Tool *Rasa Core* umgesetzt. Rasa Core nutzt keinen regelbasierten Ansatz, der Antworten nur aufgrund fest definierter Regeln erzeugt, sondern basiert auf maschinellen Lernverfahren. Die Dialogmanagement-Komponente wird hauptsächlich in den Dateien *domain.yml*, *policy_config.yml* und *stories.md* konfiguriert, die in der folgenden Sektion näher erläutert werden.

Während der Implementierungsphase wurde Rasa Core in der Version 0.11 und 0.12 veröffentlicht, welche erhebliche Änderungen in der Struktur der Applikation sowie auf API-Ebene mit sich brachte und damit keine Kompatibilität mit Applikationen älterer Versionen zulässt. Um eine Weiterentwicklung auch nach dieser Arbeit gewährleisten zu können, wurden alle Änderungen in das bereits bestehende System eingearbeitet. Die Implementierung sowie alle folgenden Erklärungen und Dokumentationen beziehen sich somit auf Rasa Core ab der Version 0.12.0.

Abbildung 13 zeigt die Einordnung der Dialogmanagement-Komponente in die Chatbot-Architektur:

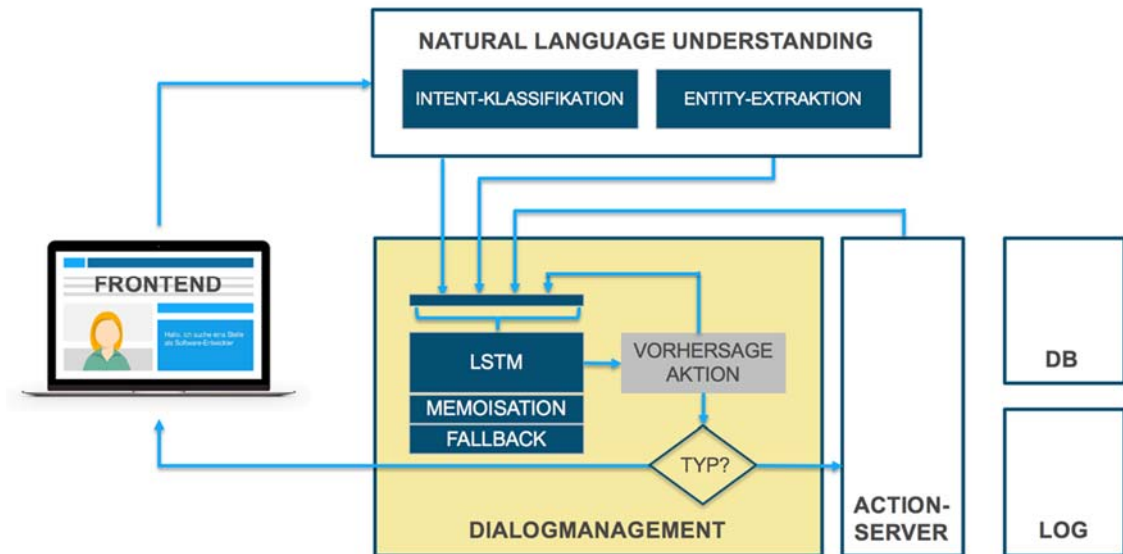


Abbildung 13: Architektur des Chatbot-Prototyps mit Dialogmanagement-Komponente

5.4.1 Architektur

Im Folgenden wird die Architektur einer Rasa-Core-Applikation anhand Abbildung 14 erläutert. Die Abbildung zeigt dabei den Prozess vom Eingang einer Nachricht über die NLU-Komponente bis zur Ausgabe einer Antwort.

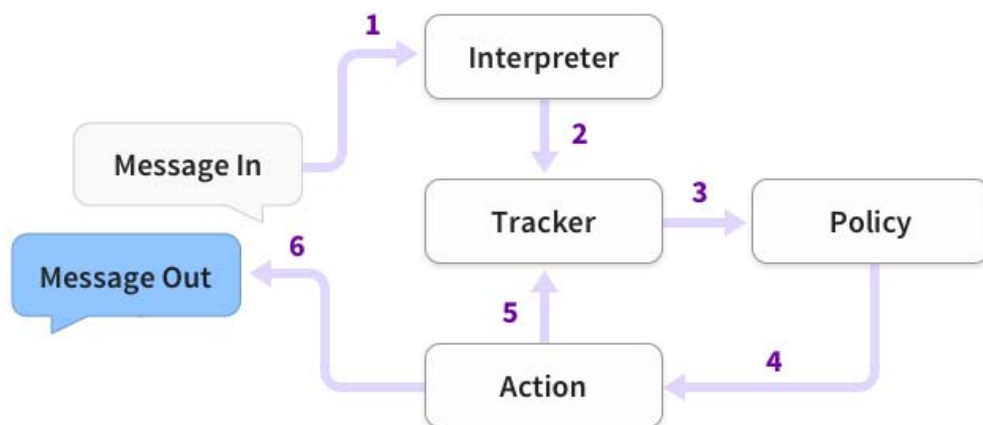


Abbildung 14: Architektur Rasa Core [47]

1. Eine vom Nutzer eingegebene Nachricht wird vom System empfangen und an einen NLU-Service weitergeleitet, der den natürlichsprachlichen Text, wie in Kapitel 3.1 beschrieben, verarbeitet.
2. Die vom NLU-Service extrahierten, strukturierten Informationen werden an den Tracker weitergegeben. Der Tracker ist ein Objekt, das den Gesprächszustand verwaltet und die Information erhält, dass eine neue Nachricht eingegangen ist. Das Tracker-Objekt ist die einzige zustandsorientierte Komponente innerhalb des Dialogmanagement-Systems. Für jede Session gibt es jeweils ein neues Tracker-Objekt.
3. Die Policy erhält den aktuellen Zustand des Trackers-Objekts.
4. Die Policy entscheidet, welche Aktion aufgrund des Zustands des Tracker-Objekts als nächstes ausgeführt werden soll. Eine Aktion kann eine Antwort an den Nutzer in Form einer Nachricht sein oder die Ausführung einer beliebigen Funktion wie z.B. der Abruf von Daten aus einer Datenbank. Für die Ausführung von benutzerdefinierten Aktionen wurde ein Action-Server implementiert, der in Kapitel 5.5 erläutert wird.
5. Die gewählte Aktion wird anschließend vom Tracker protokolliert. In folgenden Iterationen des Dialogs können Aktionen demnach alle relevanten Informationen wie frühere Äußerungen und die Ergebnisse früherer Aktionen nutzen.
6. Auf Basis der getroffenen Entscheidung, welche Aktion ausgeführt werden soll, wird eine Antwort an den Nutzer gesendet. Je nachdem, welche Aktion ausgeführt wurde, wird anschließend entweder auf eine neue Eingabe des Nutzers gewartet (Schritt 1) oder zurück zu Schritt 3 gesprungen. [47]

In den folgenden Kapiteln sollen die wichtigsten Komponenten und deren Rolle im Gesamtsystem erläutert sowie eine Dokumentation zur Implementierung im Chatbot-Prototypen gegeben werden.

5.4.2 Domain

Die Domain fungiert in Rasa Core als übergeordnete Komponente, in der alle verwendeten Bestandteile des Systems spezifiziert werden müssen [68]. Im Folgenden werden die

einzelnen enthaltenen Komponenten und deren Definition im YAML-Format kurz veranschaulicht. Bei allen Veranschaulichungen (Listings) handelt es sich um einfache Beispiele, die der eigentlichen Implementierung entnommen wurden. Eine ausführliche Definition aller Komponenten in der Domain des HR-Chatbots befindet sich in der Datei *domain.yml*, welche auf der CD im Ordner *hr-bot/app* zu finden ist.

Listing 3 zeigt beispielhaft die Definition von Intents und Entities. Dabei müssen in der Domain alle Intents und Entities gespeichert werden, die in den Trainingsdaten vorhanden sind.

```
intents:
  - enter_data
  - greet
  - find_job

entities:
  - nameApplicant
  - jobTask
```

Listing 3: Rasa Core Domain: Intents und Entities (*domain.yml*)

Listing 4 zeigt die Definition von Slots in der Domain. Slots sind Schlüssel-Wert-Paare und dienen zur internen Speicherung von Informationen. Dabei können extrahierte Entities sowie jegliche Informationen, die durch Aktionen des Chatbots generiert oder aus externen Quellen importiert wurden, als Slots gespeichert werden. Slots benötigen, neben einem Titel, einen Typ und können optional einen Startwert erhalten. In diesem Fall wird dem Namen des Nutzers *initial* ein leerer String zugewiesen, um zu vermeiden, dass der Chatbot Antworten gibt, bei denen der Bewerbername als *None* zurückgegeben wird. Für diesen Prototypen wurden hauptsächlich Slots zur Jobsuche implementiert. Zusätzlich wurde ein Slot mit dem Namen des Nutzers definiert, um diesen persönlich ansprechen zu können, wenn er seinen Namen nennt. Eine genauere Beschreibung der Slots und wie diese im Kontext der Jobsuche verwendet werden folgt in Kapitel 5.5.

```
slots:
  nameApplicant:
    type: text
    initial_value: ""
```

Listing 4: Rasa Core Domain: Slots (*domain.yml*)

Listing 5 zeigt die Definition von Aktionen (*actions*), welche als Antwort auf eine Nutzereingabe ausgeführt werden. Außer dem Namen der Aktion müssen der Domain keine weiteren Parameter bekannt sein. Kapitel 5.5 beschäftigt sich mit diesem Thema genauer und geht auf die Realisierung von Aktionen im HR-Prototypen ein.

```
actions:
- action_find_job
- action_match_job_slots
- utter_greet
```

Listing 5: Rasa Core Domain: Actions (*domain.yml*)

Listing 6 zeigt die Definition eines Templates innerhalb der Domain. Templates sind Textnachrichten vom Datentyp String, die der Chatbot als Antwort auf eine Eingabe an den Nutzer sendet. Weiterhin können Variablen in der Nachricht verwendet werden, deren Werte aus vorher definierten Slots genommen werden, wenn diese bereits mit einem Wert gefüllt wurden. Hat ein verwendeter Slot keinen Wert, wird er standardmäßig als *None* zurückgegeben.

```
templates:
  utter_greet:
    - text: "Hallo {nameApplicant}! Wie kann ich dir behilflich sein?"
```

Listing 6: Rasa Core Domain: Templates (*domain.yml*)

5.4.3 Stories

Trainingsdaten werden bei Rasa Core als *Stories* bezeichnet [47]. Eine Story ist dabei eine beispielhafte Konversation oder ein Abschnitt einer längeren Konversation. Mithilfe der Daten, die aus den Trainings-Stories gewonnen werden, wird ein Machine-Learning-Modell trainiert, das zur Entscheidung, welche Aktion als nächstes ausgeführt werden soll, verwendet wird. Einzelheiten zur technischen Ausführung des Trainings folgen in Kapitel 5.4.4.

Listing 7 zeigt beispielhaft die Umsetzung einer kurzen Story im Markdown-Format:

```
1  ## greet
2  * greet
3      - utter_greet
4
5  * enter_data{"jobTask": "designen"}
6      - slot{"jobTask": "designen"}
7      - utter_askDomain
```

Listing 7: Rasa Core Story „Greet“ (*stories.md*)

Eine Story wird mit einem Titel begonnen. Dieser ist obligatorisch, der Inhalt ist jedoch für das Training irrelevant und fördert ausschließlich die Übersichtlichkeit des Codes. Zeile 2 zeigt den Intent *greet*, der Nutzer hat also die Intention den Chatbot zu begrüßen. Neben Intents können damit, wie in Zeile 5 gezeigt, auch Entities definiert werden. Der Intent *enter_data* beschreibt hierbei die Absicht des Nutzers Daten zur Jobsuche zu nennen. In diesem Fall handelt es sich um die Entity *jobTask* mit dem Wert *designen*, welche als Schlüssel-Wert-Paar angegeben wird. Zeile 6 zeigt, dass für diese eingehende Entity *jobTask* ein Slot gesetzt werden soll. Der Slot muss vorher mit demselben Namen in der Domain definiert worden sein. Auf den Intent folgend wird in Zeile 3 bzw. Zeile 7 eine Antwort des Bots in Form einer Aktion definiert, die aufgrund des Intents erwartet wird. In diesem Fall handelt es sich dabei um simple Textnachrichten, die in der Domain als Template angelegt wurden. Eine Auflistung aller zum Training verwendeten Stories befindet sich in der Datei *stories.md*, welche auf der CD im Ordner *hr-bot/app/data/core* zu finden ist.

5.4.4 Policy

Die Policy-Klasse entscheidet, welche Aktion auf der Basis des aktuellen Zustands des Tracker-Objekts als nächstes ausgeführt werden soll. Um eine Aktion vorhersagen zu können, muss mit der Policy ein Featurizer instanziiert werden. Dieser erzeugt aufgrund der Parameter im Tracker-Objekt eine Vektordarstellung des Gesprächszustandes, welche daraufhin von verschiedenen Machine-Learning-Modellen verwendet werden kann. Die verwendeten Parameter beinhalten die letzte Aktion, die vom Chatbot ausgeführt wurde, alle bisher gesetzten Slots, sowie die erkannten Intents und Entities der aktuellen Eingabe des Nutzers [47]. Im Rahmen dieses Prototyps wurden drei Policies implementiert: *Memoization Policy*, *Keras Policy* und *Fallback Policy*. In jeder Iteration des Dialogs treffen die Policies eine Voraussage über die nächste auszuführende Aktion. Die Policy, deren Voraussage die höchste Wahrscheinlichkeit aufweist, wird daraufhin ausgewählt und deren Aktion ausgeführt.

Die *Memoization Policy* stellt sicher, dass eine Konversation, die genauso in den Trainingsdaten vorhanden ist, eindeutig erkannt wird. Ist dies der Fall, wird eine Wahrscheinlichkeit der folgenden Aktion mit 1,0 ausgegeben. Kann keine exakte Übereinstimmung der Konversation mit den Trainingsdaten gefunden werden, werden alle Aktionen mit einer Wahrscheinlichkeit von 0,0 vorhergesagt, was die Notwendigkeit einer weiteren Policy hervorbringt.

Die *Keras Policy* trifft eine Vorhersage der nächsten Aktion aufgrund eines mit Keras implementierten Neuronalen Netzes. Dabei handelt es sich um ein Long Short-Term Memory-Netz (LSTM) [42]. Der Feature-Vektor, der auf der Basis des Tracker-Objekts erzeugt wurde, dient als Eingabe für das LSTM. Die Ausgabe des Netzes ist eine Wahrscheinlichkeitsverteilung über alle möglichen Aktionen. Die Aktion mit der höchsten Wahrscheinlichkeit wird daraufhin ausgeführt und wird als Eingabe in das LSTM in der nächsten Iteration zurückgeführt [69].

Die *Fallback Policy* wird ausgeführt, wenn die NLU-Komponente keinen Intent mit einer Wahrscheinlichkeit, welche höher als ein bestimmter Schwellenwert ist, findet. Weiterhin wird sie auch aufgerufen, wenn die beiden vorherigen Policies keine Aktion mit einer Wahrscheinlichkeit über einem bestimmten Schwellenwert vorhersagen konnte.

5.5 Aktionen

Diese Sektion beschreibt die Implementierung von benutzerdefinierten Aktionen im Chatbot-Prototyp. Dafür wurde ein Backend-Server implementiert, der die Aktionen ausführt. Abbildung 15 zeigt die Einordnung dieser Komponente in die Chatbot-Architektur:

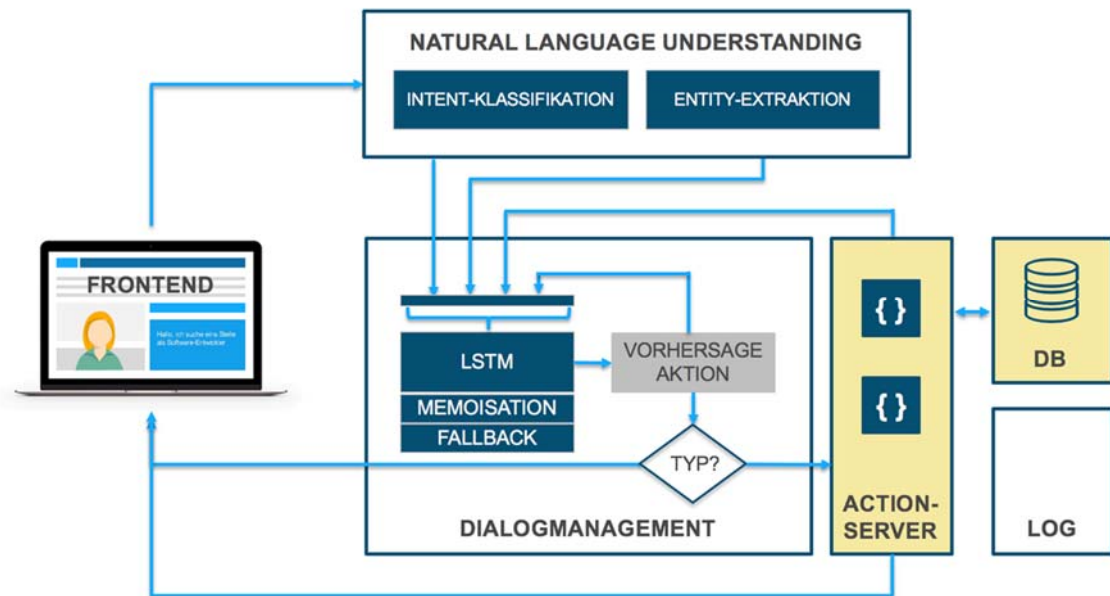


Abbildung 15: Architektur des Chatbot-Prototyps mit Action-Server

Für jede eingehende Nachricht eines Nutzers wird mindestens eine Aktion ausgeführt. Man unterscheidet bei Rasa Core zwischen verschiedenen Arten von Aktionen. Es gibt vordefinierte Standard-Aktionen wie *action_listen* und *action_restart*, welche vorgeben, dass auf eine neue Nachricht des Nutzers gewartet werden bzw. der Zustand der Konversation zurückgesetzt werden soll. Weiterhin gibt es Utterance-Aktionen, die simple Textnachrichten an den Nutzer senden. Dies geschieht mithilfe der in Kapitel 5.4.2 vorgestellten Templates (siehe Listing 6) innerhalb der definierten Domain. Als letzte und für dieses Szenario wichtigste Art von Aktionen stehen benutzerdefinierte Aktionen, welche beliebige Funktionen ausführen können. Dafür wurde ein Webserver implementiert, welcher bei der Auswahl einer benutzerdefinierten Aktion angesprochen wird.

Jede Aktion wird als eigene Python-Klasse definiert und ist eine abgeleitete Klasse der in Rasa Core implementierten Klasse *Action*, von welcher die Methoden *name* und *run* geerbt werden (siehe Listing 8).

```
class ActionFindExistingJob(Action):  
  
    def name(self):  
        return "action_find_job"  
  
    def run(self, dispatcher, tracker, domain):  
        [...]
```

Listing 8: Definition von Aktionen (*actions.py*)

In der Methode *name* wird der Name der Aktion definiert, der innerhalb der Stories verwendet wird.

Die Methode *run* erwartet die Eingabeparameter *dispatcher*, *tracker* und *domain*, welche im Folgenden erläutert werden sollen:

- Tracker: Die Tracker-Klasse erlaubt den Zugriff auf alle Daten, die im bisherigen Gesprächsverlauf gesammelt wurden.
- Dispatcher: Über die Dispatcher-Klasse können Nachrichten an den Nutzer gesendet werden.
- Domain: Die Domäne des Bots (siehe Kapitel 5.4.2)

Nach dem Ausführen einer entsprechenden Funktion im Server werden gesammelte oder verarbeitete Daten, sowie eine Reihe an möglichen Events an die Dialogmanagement-Komponente zurückliefert. Diese Events sind nötig, um den Tracker zu aktualisieren und können beispielsweise Slots Werte zuweisen oder zurücksetzen [47].

Die in diesem Prototyp implementierten benutzerdefinierten Aktionen sind hauptsächlich Funktionen, die eingegebene Informationen im Szenario der Jobsuche mit Daten in einer Datenbank vergleichen und entsprechende Werte zurückgeben. Um diese Funktionen erläutern zu können, soll im Folgenden zuerst auf die Datenstruktur eines Jobs in diesem Prototyp eingegangen werden.

5.5.1 Implementierung der Datenstruktur eines Jobs

Wie in Kapitel 4 beschrieben sollte der Chatbot anhand der aufgebauten Datenstruktur eines Jobs auf die Antworten des Nutzers Bezug nehmen können. Aus technischer Sicht wurde dafür eine SQLite -Datenbank [70] aufgesetzt, welche die Beziehung der drei Abstraktionsebenen in Form von Tabellen zueinander abbildet. Die Speicherung der Daten in der SQLite-Datenbank geschieht mithilfe von *Pony ORM*. Pony ORM ist ein *Object Relational Mapping* (ORM) Tool, welches es ermöglicht mit dem Inhalt einer Datenbank in Form von Objekten zu arbeiten

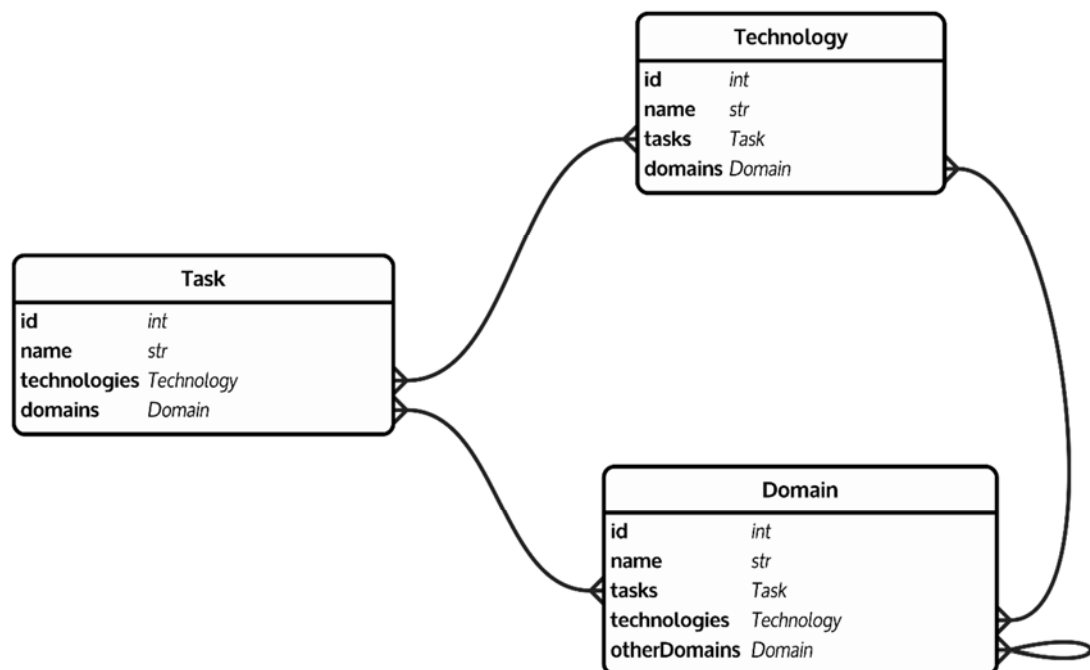


Abbildung 16: Datenstruktur eines Jobs

Jede der Tabelle besitzt neben einer ID und einem Namen eine m:n-Beziehung zu den zwei weiteren Tabellen. Weiterhin besitzt die Domain eine selbstreferenzierende Beziehung zu weiteren Domains. Der Grund dafür ist, dass eine Domain wie z.B. „Fullstack“ andere Domains wie „Frontend“ oder „Backend“ beinhalten kann. Diese Selbstreferenz ist jedoch nur Teil der Konzeption und wurde bei der prototypischen Implementation nicht weiter berücksichtigt. Alle Domains wurden in der Datenbank so definiert, dass diese ohne weitere Beziehung bestehen können.

5.5.2 Benutzerdefinierte Aktionen zur Jobsuche

Wie in Kapitel 4 beschrieben, werden dem Nutzer Stellenanzeigen aufgrund von Informationen zu den Abstraktionsebenen Task, Domain und Technology angezeigt. Um Nachfragen des Chatbots möglichst einfach zu halten und unsinnige Nachfragen zu vermeiden, werden zu jedem Zeitpunkt, wenn eine Eingabe zu mindestens einer der Abstraktionsebenen von der NLU-Komponente erkannt wurden, Funktionen ausgeführt, die die Eingabe und die bisher bekannten Informationen in Form von gesetzten Slots mit einer Datenbank abgleichen. Die dafür im Backend-Server implementierte Klasse *Action-MatchJobSlots* erhält als Eingabeparameter alle bisherigen Daten des aktuellen Gesprächszustandes in Form des Tracker-Objektes.

Zuerst wird dabei untersucht, ob aufgrund der Nutzeräußerung direkt eine passende Stelle gefunden werden kann. Dazu wird die letzte Nachricht aus dem Tracker-Objekt extrahiert und mit den Titeln der vorhandenen Jobs verglichen. Kann mindestens ein Stellentitel als Teil-String in der Äußerung des Nutzers gefunden werden, wird dieser sofort als Slot gesetzt und an den Nutzer zurückgegeben. Die Stellensuche ist damit erfolgreich abgeschlossen. Kann kein Job gefunden werden, wird versucht durch wenige Fragen des Chatbots die Datenstruktur einer Stelle abzubilden. Ausgehend von den bisher erkannten Informationen zu den drei Abstraktionsebenen Task, Domain und Technology, welche im Tracker-Objekt als Slots gesetzt wurden, werden alle verbleibenden Möglichkeiten zu den noch nicht gesetzten Slots mithilfe einer Abfrage an die in Kapitel 5.5.1 vorgestellte Datenbank herausgesucht und wiederum als Slots mit den Namen *possibleTasks*, *possibleDomains* oder *possibleTechnologies* im Tracker-Objekt gespeichert. Auf Basis der nun gesetzten Slots wird im Anschluss als Antwort eine Nachricht an den Nutzer gesendet. Diese Antwort kann entweder eine einfache Textnachricht mit einer weiteren Nachfrage zur Stellensuche sein oder dem Nutzer weitere Auswahlmöglichkeiten zu einer der Abstraktionsebenen in Form von Buttons bereitstellen. Die Anzeige von Buttons an manchen Stellen innerhalb des Chatbots wurde gewählt, da dem Nutzer die Datenstruktur in Form von verschiedenen Abstraktionsebenen nicht bekannt ist und auch nicht bekannt sein muss. Um Nachfragen möglichst einfach zu halten, wird dem Nutzer die Möglichkeit gegeben zwischen mehreren vorgeschlagenen Begriffen wie z.B. Domains in Form von Buttons zu wählen. Dieses Konzept der Aktionen im Backend-Server soll im Folgenden an einem einfachen Beispiel einer Konversation mit dem Chatbot verdeutlicht werden. Ein potentieller Bewerber könnte den Dialog mit folgender Utterance beginnen:

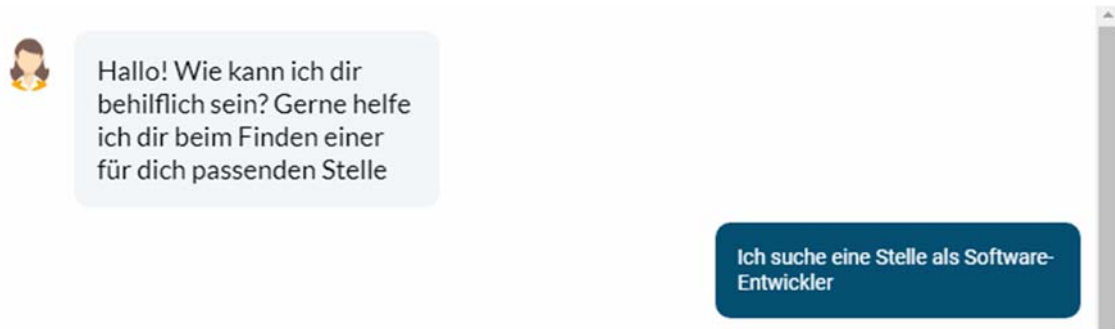


Abbildung 17: Beispiel-Dialog Stellensuche 1

Die NLU-Komponente erkennt daraufhin die Intention des Nutzers, eine Stelle zu finden und extrahiert die Task-Entity mit dem Wert „entwickeln“. Dies ist möglich, da die implementierte *ner_synonyms*-Komponente synonymen Entities dieselben Werte zuordnet. Weiterhin wird der Wert der Task-Entity als Slot im Tracker-Objekt gespeichert. Wie erläutert, wird aufgrund des Erkennens einer der drei Abstraktionsebenen anschließend eine Aktion ausgeführt, welche alle verbleibenden Möglichkeiten zu den noch nicht gesetzten Slots als Rückgabe besitzt. Neben dem erkannten Slot `jobTask: ['entwickeln']` werden damit die Slots `possibleDomains` und `possibleTechnologies` mit allen noch möglichen Werten versehen. Kann für einen oder beide der Slots nur eine einzige Möglichkeit gefunden werden, wird diese automatisch als Slot Domain oder Technology gesetzt und macht eine weitere Nachfrage nach dieser Abstraktionsebene damit obsolet. Eine weitere Möglichkeit besteht darin, dass bei bestimmten Angaben für eine Abstraktionsebene keine Werte in der Datenbank gefunden werden. Tritt dieser Fall ein, wird der Prozess der Jobsuche in diesem Prototypen abgebrochen und es wird auf die Möglichkeit einer Initiativbewerbung hingewiesen. Da im geschilderten Szenario jedoch beide Slots noch eine Vielzahl an möglichen Werten zulassen, fährt der Chatbot mit weiteren Nachfragen fort:

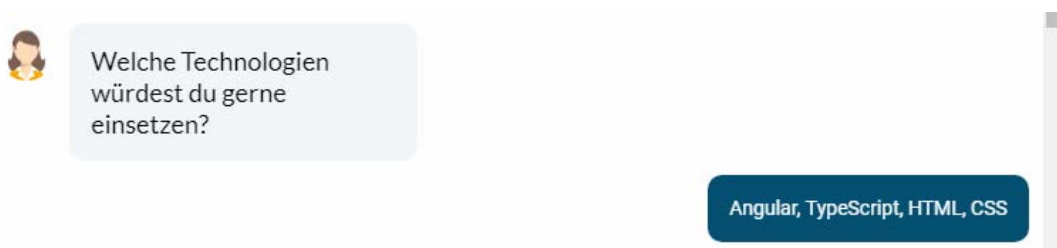


Abbildung 18: Beispiel-Dialog Stellensuche 2

Die NLU-Komponente erkennt aus der Antwort des Nutzers daraufhin vier Werte der Technology-Entity, welche ebenfalls automatisch als Slot im Tracker-Objekt gespeichert werden. Da nun zwei der drei Abstraktionsebenen eines Jobs erkannt wurden, wird erneut eine Aktion ausgeführt, die alle Domains zurückgibt, die aufgrund der bisherigen Angaben noch möglich erscheinen. Für den bisherigen Konversationsverlauf sähe ein Ausschnitt des Tracker-Objektes nun wie folgt aus:

```
[...]
jobTask: ['entwickeln'],
technology: ['Angular', 'Typescript', 'HTML', 'CSS'],
domains: [],
possibleDomains: ['fullstack', 'backend', 'web', 'frontend', 'mobile']
[...]
```

Listing 9: Tracker-Objekt im Beispiel-Dialog

Damit potenzielle Stellen noch besser eingegrenzt werden können, wird nach der gewünschten Domain gefragt. Um es für den Nutzer möglichst einfach zu gestalten, werden die noch verbleibenden Möglichkeiten als Buttons innerhalb des Chat-Fensters angezeigt:

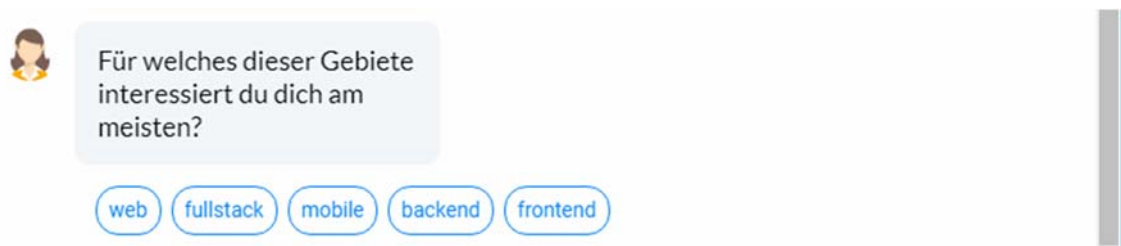


Abbildung 19: Beispiel-Dialog Stellensuche 3

Klickt der Nutzer auf einen der Buttons, wird dessen Wert als Antwort an den Chatbot gesendet. In diesem Szenario wählt der Nutzer die Domain „Web“. Da nun Daten für alle drei Abstraktionsebenen erkannt wurden und damit alle benötigten Slots im Tracker-Objekt gespeichert sind, können anschließend ergänzende Fragen zur Jobsuche gestellt werden, wie z.B. Fragen nach dem gewünschten Beschäftigungsverhältnis, die in diesem Prototypen ebenfalls in Form von Buttons dargestellt werden.



Abbildung 20: Beispiel-Dialog Stellensuche 4

Sind alle notwendigen Informationen gesammelt kommt anschließend eine weitere benutzerdefinierte Aktion zum Einsatz. Die ebenfalls im Backend-Server implementierte Klasse *ActionFindExistingJob* erhält als Eingabe alle gespeicherten Slots des Tracker-Objektes und vergleicht die Eingaben mit den im Backend gespeicherten Jobs. Wird mindestens eine Stellenanzeige gefunden, welche wenigstens eine Übereinstimmung für jeden Slot besitzt, wird diese vom Chatbot an den Nutzer als Antwort zurückgegeben.

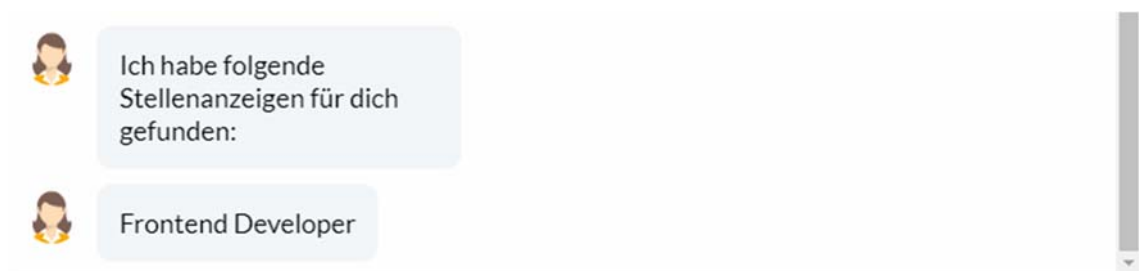


Abbildung 21: Beispiel-Dialog Stellensuche 5

Obwohl versucht wurde, die Nachfragen nach den einzelnen Abstraktionsebenen möglichst einfach und intuitiv zu gestalten, stellte sich in der Konzeption heraus, dass die Möglichkeit geschaffen werden sollte, Gegenfragen stellen zu können, wenn dem Nutzer nicht klar erkenntlich ist, was von ihm verlangt wird. Dies ist vor allem bei der Frage nach Technologien, die man gerne einsetzen würde, der Fall. Ein Nutzer könnte darauf beispielsweise nachfragen, welche Technologien es denn überhaupt gibt oder wie der Begriff Technologie für das Unternehmen des Chatbots definiert ist. Um dem Nutzer eine möglichst einfache und auf seine bisherigen Angaben angepasste Antwort liefern zu kön-

nen, wurde dafür eine weitere benutzerdefinierte Aktion implementiert. In der Klasse *ActionShowTechnologies* wird eine Auswahl aller Technologien, die mit den bisherigen Eingaben noch möglich erscheinen, gesammelt und an den Nutzer zurückgesendet. Aufgrund dessen erhält er eine Vorstellung, was unter einer Technologie verstanden werden kann sowie eine Antwort, die möglicherweise bereits für ihn passende Technologien enthält.

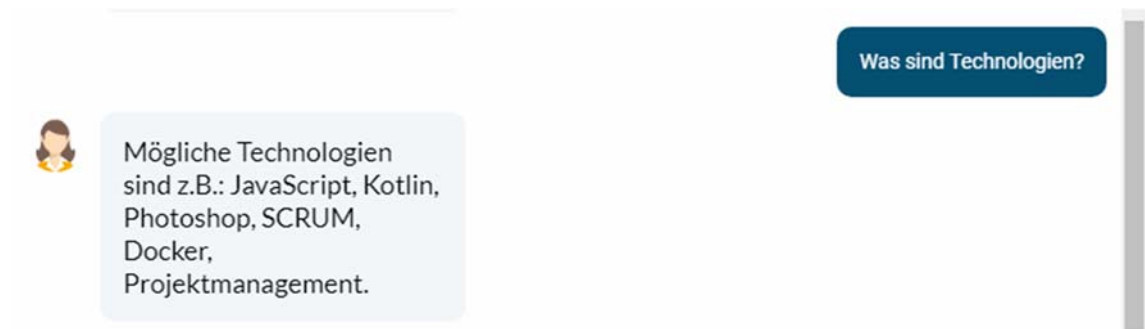


Abbildung 22: Beispiel-Dialog Stellensuche 6

5.6 Logging

Um eine Evaluation des Prototypen als Gesamtsystem durchführen zu können, ist es notwendig möglichst viele Informationen über die Dialoge von Nutzern mit dem Chatbot zu erhalten. Weiterhin sollen alte Gespräche als Grundlage für weiteres Training des Chatbots dienen. Dazu wurden Komponenten entwickelt, die eine detaillierte Möglichkeit zur Speicherung von Gesprächsinformationen bieten. Abbildung 23 zeigt die Einordnung dieser Komponente in die Chatbot-Architektur:

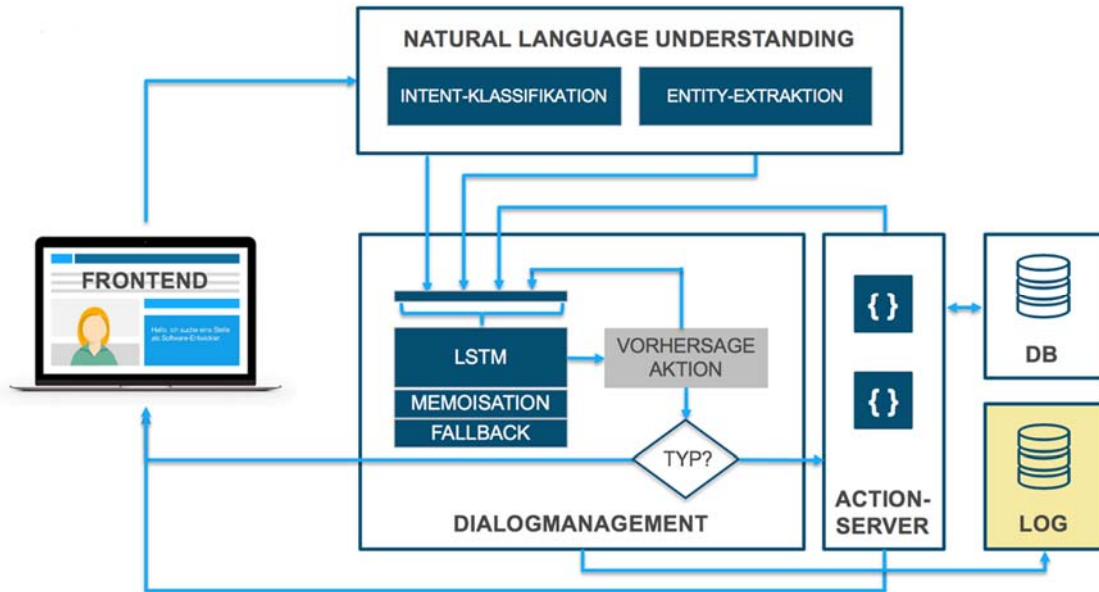


Abbildung 23: Architektur des Chatbot-Prototyps mit Logging_Komponente

5.6.1 Permanente Datenspeicherung

Rasa Core speichert, wie in Kapitel 5.4.1 und 5.4.4 beschrieben, den aktuellen Gesprächszustand einer Konversation im Tracker-Objekt. Standardmäßig wird dafür die *InMemoryTrackerStore*-Klasse verwendet, welche alle Daten im lokalen Speicher ablegt. Dies genügt für den produktiven Einsatz eines Chatbots, reicht für ein detailliertes Logging jedoch nicht aus, da die Daten nur gespeichert bleiben, solange das System läuft. Aufgrund dessen wurde eine TrackerStore-Klasse implementiert, die den Gesprächszustand in jedem Schritt in eine Datenbank speichert.

5.6.2 Visualisierung der Dialoginformationen

Ergänzend zur Speicherung aller Gesprächsdaten in einer Datenbank wurde ein Frontend in Form einer Webseite entwickelt, um die Daten gefiltert, formatiert und strukturiert darbieten zu können. Für jeden stattgefundenen Dialog können somit alle Eingaben der Nutzer sowie die daraufhin erkannten Aktionen und Antworten des Chatbots nachvollzogen und für weitere Trainingsiterationen der NLU- und Dialogmanagement-Komponente verwendet werden. Auf die einzelnen Bestandteile der Logging-Komponente wird im Folgenden eingegangen:

- *MongoDB* [71] ist eine dokumentenorientierte Open Source NoSQL-Datenbank, in welcher JSON-ähnliche Dokumente verwaltet werden können. Eine MongoDB-Datenbank kann mehrere Kollektionen enthalten, in denen wiederum mehrere Dokumente gespeichert werden können. Für den Prototypen wurde eine Datenbank angelegt, die eine Kollektion mit allen bisher geführten Konversationen als Dokumente enthält.
- *Node.js* [72] ist eine Open-Source JavaScript-Laufzeitumgebung, die auf Chromes V8 JavaScript-Engine basiert. Sie wird hauptsächlich zur Entwicklung von Serveranwendungen verwendet. *Express* [73] ist ein minimalistisches Node.js Framework, welches jenes um einige nützliche Features für die Entwicklung von Web-Anwendungen erweitert. Express wird in dieser Komponente zum Aufbau der API-Endpunkte des Backends in Form einer REST-Schnittstelle verwendet. Mithilfe des Express Routers wurden Funktionen zur Datenbank-Abfrage definiert, die durch HTTP-Anfragen an die entsprechende Route ausgeführt werden sollen. Weiterhin wird Express verwendet, um den Backend-Server zu starten.
- *Angular* [74] ist ein Framework zur Entwicklung von Frontend-Webapplikationen, welches auf TypeScript basiert. Im Prototypen wurde innerhalb des Frontends ein Angular Service implementiert, der HTTP-Anfragen an die jeweiligen Endpunkte des Backends sendet und eine Liste von Konversationen im JSON-Format als Antwort erhält. Die Liste wird daraufhin nach bestimmten Ereignissen innerhalb der Konversation und weiteren Variablen gefiltert und als strukturierte Liste auf der Webseite angezeigt. Abbildung 24 zeigt beispielhaft die Ausgabe eines kurzen Gesprächsanfangs auf der Webseite.

```

Bot did: action_listen

User said: hi

Bot did: utter_greet

Bot said: Hallo ! Wie kann ich dir behilflich sein? Gerne helfe ich dir beim Finden einer für dich passenden Stelle

Bot did: action_listen

```

Abbildung 24: Logging-Frontend

Durch die auf der Webseite enthaltenen Informationen, die in Aussagen und Aktionen des Nutzers bzw. des Chatbots unterteilt wurden, besteht die Möglichkeit, schnell und unkompliziert Fehlverhalten im Chatbot zu untersuchen sowie Nutzeraussagen als neue Trainings-Utterances zu verwenden. Eine Aktualisierung der Gesprächsverläufe im Logging-Frontend kann durch die verwendete Architektur ohne Verzögerung stattfinden, da die Daten im Prototypen nicht weiter zwischengespeichert werden, sondern direkt aus dem aktuellen Tracker-Objekt des Chatbots geladen werden. Abbildung 25 zeigt die Architektur der vorgestellten Logging-Komponente sowie deren Beziehung zum Chatbot-System.

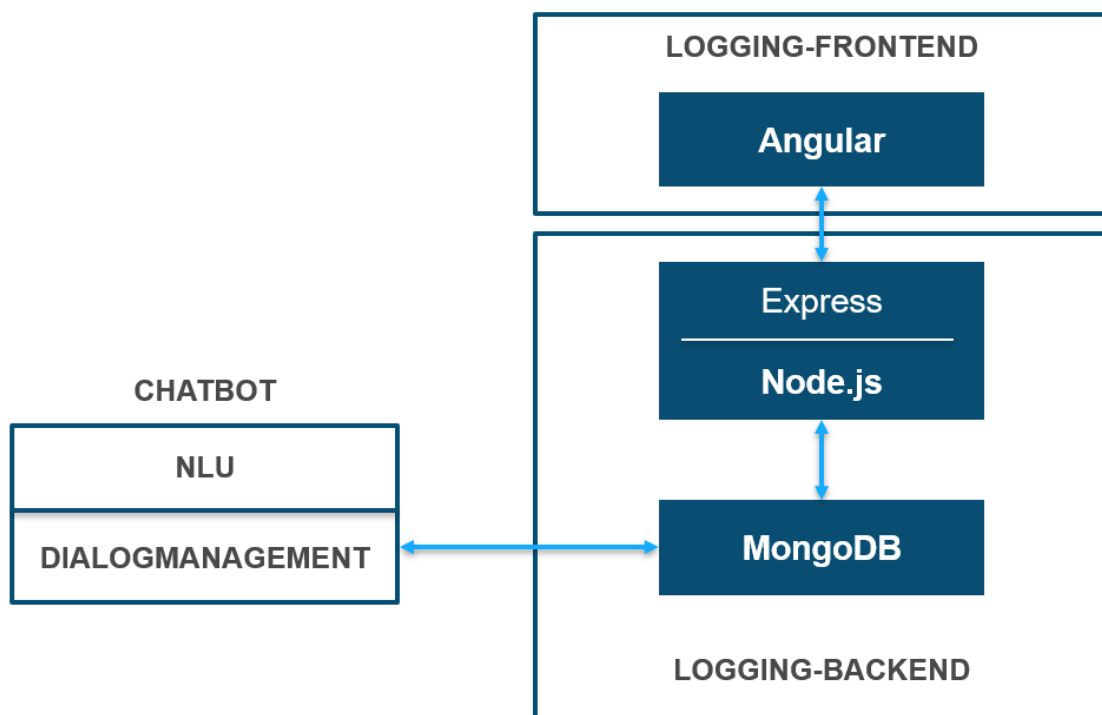


Abbildung 25: Architektur Logging-Komponente

5.7 Deployment

Alle Komponenten des Prototypen, mit Ausnahme der Logging-Komponente, wurden mithilfe von *Docker* in unabhängigen Containern bereitgestellt und sind im Web über die Plattformen *OpenShift* bzw. *Amazon Web Services (AWS)* erreichbar. Die Struktur und die dabei verwendeten Technologien werden im Folgenden kurz erläutert.

Docker ist eine Open-Source-Implementierung der Containerisierung-Technologie, welche die Erstellung und den Betrieb von verteilten Applikationen ermöglicht. Ein solcher Container enthält alles, was zum Ausführen der Software erforderlich ist, einschließlich aller nötigen Bibliotheken, Systemtools sowie dem Code, der ausgeführt wird. Über eine Konfigurationsdatei, die Dockerfile genannt wird, kann ein Abbild eines Containers definiert und gestartet werden [75]. Listing 10 zeigt beispielhaft eine vereinfachte Darstellung des Dockerfiles für die Rasa-Core-Komponente:

```
FROM python:3.6-slim

RUN groupadd -g 999 appuser && useradd -r -u 999 -g appuser appuser

RUN apt-get update -qq && apt-get install [...]

[...]

COPY . .
VOLUME ["/app/model", "/app/config", "/app/project"]

EXPOSE 5005

[...]

CMD ["start", "--core", "app/models/dialogue", "--endpoints", "app/config/endpoints.yml", "-u", "default/current", "--credentials", "app/config/credentials.yml"]
```

Listing 10: Dockerfile Rasa Core

Als Basis-Container wird *python:3.6-slim* verwendet, ein Container, welcher auf einem Linux-Betriebssystem aufsetzt und Python in der Version 3.6 sowie alle dafür nötigen Bibliotheken vorinstalliert hat. Der RUN-Befehl stattet einen Nutzer-Account mit Rechten aus, die nötig sind, um den Container auf der OpenShift-Plattform lauffähig zu machen. Weiterhin werden alle nötigen Bibliotheken und Updates installiert. Im Anschluss wird der Programmcode der Dialogmanagement-Komponente in die Ordnerstruktur des Containers kopiert und der Port 5005 freigegeben. Zum Schluss wird der Befehl definiert, mit welchem der Container gestartet werden soll. Dabei werden alle von Rasa benötigten Konfigurationsdateien als Parameter übergeben.

OpenShift ist eine von Red Hat entwickelte Open-Source-Plattform für die Bereitstellung und Verwaltung von Containern. Ein OpenShift-Projekt kann dabei mehrere Docker-Container enthalten, die verwaltet werden und untereinander kommunizieren können. Weiterhin können Docker-Container über eine Schnittstelle ins Web erreichbar gemacht werden [76]. In diesem Projekt werden die NLU- und Dialogmanagement-Komponente, der Chatbot-Backend-Server sowie das Chat-Frontend in Form einer Webseite als Container auf OpenShift bereitgestellt. Abbildung 26 zeigt einen Ausschnitt der Weboberfläche des OpenShift-Projektes.

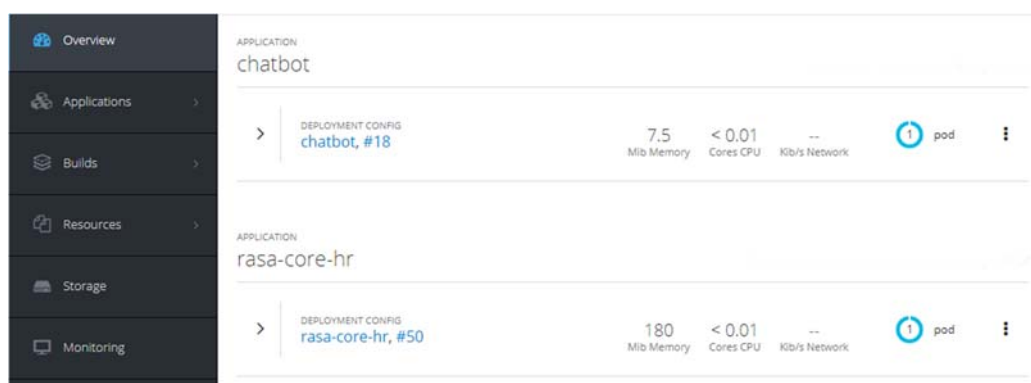


Abbildung 26: OpenShift-Projekt

AWS ist ein von Amazon angebotener Cloud-Computing-Dienst. Der darin enthaltene Web-Service Elastic Compute Cloud (EC2) stellt skalierbare Rechenkapazitäten in Form von Server-Instanzen bereit [77]. Ein EC2-Server wird in diesem Projekt zur Bereitstellung der MongoDB verwendet, welche die Daten der TrackerStore-Klasse der Dialogmanagement-Komponente enthält und für die Logging-Komponente persistiert.

5.8 Evaluation des Prototypen

Bei der Evaluation des HR-Chatbots für Bewerber lag der Fokus auf zwei Aspekten des Prototypen. Zum einen sollte erneut die Leistungsfähigkeit des Klassifikators der NLU-Komponente mit den neu gewonnen Trainingsdaten geprüft werden. Als zweiter Evalua-

tionsschritt sollte weiterhin die Dialogführung des Chatbots in Kombination mit der konzipierten Datenstruktur untersucht werden. Dabei lag der Fokus darauf, herauszufinden, inwiefern eine Jobsuche anhand der drei konzipierten Abstraktionsebenen möglich ist.

5.8.1 Evaluation der NLU-Komponente

Die Evaluation der NLU-Komponente geschah erneut anhand der Metriken precision, recall und f1-score, die in Kapitel 3.3.2 erläutert wurden. Dabei wurden alle Trainingsdaten des Prototypen unter Anwendung einer 5-fachen Kreuzvalidierung verwendet. Dieser Datensatz umfasste zum Zeitpunkt der Evaluation 676 Utterances in deutscher Sprache und beinhaltete 13 verschiedene Intents und fünf Entities sowie den Wert *None*, wenn jeweils kein Intent oder keine Entity erkannt werden konnte. Tabelle 6 zeigt die durchschnittlichen Evaluationsergebnisse:

Intent	precision	recall	f1-score
\sum Intents	0,92	0,87	0,89
Entities	precision	recall	f1-score
\sum Entities	0,97	0,97	0,97

Tabelle 6: Evaluation der NLU-Komponente des Prototyps

Bei der Erkennung der Entities wurden fast durchgängig sehr gute Werte erzielt. Ausnahmen zeigten sich in den Recall-Werten bei der Erkennung der Domains (recall = 0,460) sowie vereinzelt in den Recall-Werten bei der Erkennung der Tasks (recall = 0,779). Eine komplette Auflistung der Ergebnisse aller fünf Iterationen befindet sich im Anhang A.2. Dies bedeutet, dass insgesamt weniger dieser Entities, die in den Trainingsdaten vorhanden waren, erkannt wurden. Der Grund dafür ist unter anderem, dass, im Gegensatz zu den anderen Entities, sehr wenige Trainingsdaten verwendet wurden, da nur eine begrenzte Anzahl an Antworten möglich ist. Ob sich diese Ergebnisse in der Nutzung des Chatbot-Prototypen widerspiegeln, wird in Kapitel 5.8.2 erläutert.

Zur besseren Veranschaulichung wurde für die Evaluation der Intents eine Konfusionsmatrix erstellt. Dabei wird die Anzahl der richtig positiven und richtig negativen bzw. falsch positiven und falsch negativen Vorhersagen, die auch für die Berechnung von precision, recall und f1-score herangezogen wurden, dargestellt:

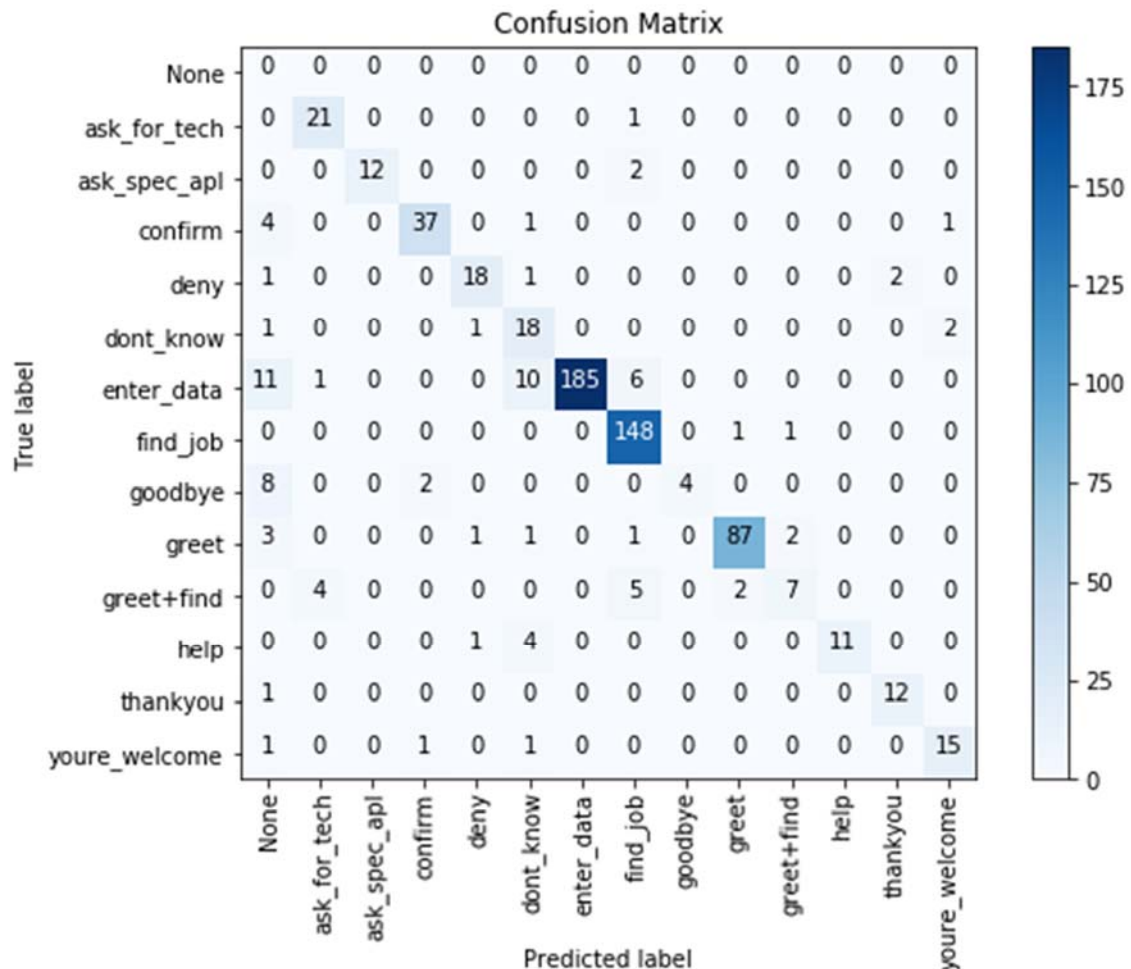


Abbildung 27: Konfusionsmatrix NLU-Komponente

Auffällig ist, dass Intents des Typs *goodbye* sehr schlecht erkannt wurden. Grund hierfür ist, dass Werte dieses Intents meist einzelne Wörter sind, die teilweise sehr unähnlich erscheinen. Da insgesamt wenige Trainingsdaten dieses Intents vorhanden waren und diese gleichmäßig über alle Teile der Trainingsdaten bei der Kreuzvalidierung verteilt waren, kam es zu häufigeren Fehlklassifikationen bei der Evaluation.

Die häufigsten Fehlklassifikationen konnten beim Intent *enter_data* gefunden werden, welcher als Eingabe für die Entities Task, Domain, Technology sowie dem Beschäftigungsverhältnis dient. Hierbei kann ebenfalls angenommen werden, dass dieses Verhalten aufgrund der vielen Trainingsdaten auftritt, die nur aus einem einzigen Wort bestehen und wenig ähnlich erscheinen.

In 5 von 18 Fällen wurde statt dem kombinierten Intent *greet+find_job* nur der Intent *find_job* gefunden, in 2 von 18 Fällen nur der Intent *greet*. Da die vom Chatbot vorhergesagte Aktion auf die Intents *greet+find_job* bzw. *find_job* identisch sein sollte, spielt dies für den Prototypen keine große Rolle. Für weitere Trainingsiterationen sollten dafür jedoch mehr Daten gesammelt werden, um die Klassifikationsleistung zu verbessern.

Der Intent, welcher am häufigsten vorhergesagt wurde, obwohl ein anderer erkannt werden sollte, ist der *None*-Intent. Es konnte also kein Intent gefunden werden. Die Fehlklassifikation als *None* stellt für den Produktiveinsatz das geringste Problem aller falsch klassifizierten Intents dar, da daraufhin die Fallback-Policy der Dialogmanagement-Komponente greift und den Nutzer bittet, seine Formulierung noch einmal anders zu wiederholen. Eine Klassifikation als anderer Intent würde wiederum zu falschen Folgerungen des Chatbots führen.

Insgesamt zeigt sich die Leistungsfähigkeit des Klassifikators in Form der Open-Source-Lösung von Rasa NLU als vielversprechend und lässt sich durch weitere gezielte Trainingsdaten noch verbessern.

5.8.2 Evaluation des Gesamtsystems als Nutzertest

Neben der Evaluation der Leistungsfähigkeit der NLU-Komponente wurde das Gesamtsystem evaluiert. Dies bedeutet, dass eine Nutzer-Interaktion mit dem Chatbot-Frontend über die NLU-Komponente hin zur Dialogführung des Chatbots in Kombination mit der konzipierten Datenstruktur im Backend untersucht werden sollte. Dabei lag der Fokus darauf, herauszufinden, inwiefern eine Jobsuche anhand der drei konzipierten Abstraktionsebenen möglich ist.

Als Szenario für den Nutzertest wurde gewählt, dass man aus der Sicht eines potenziellen Bewerbers nach seiner eigenen Stelle oder der Stelle eines Kollegen im Unternehmen

suchen sollte. Bei Testpersonen, die nicht für die adorsys GmbH & Co. KG arbeiteten, wurde darauf hingewiesen, dass hauptsächlich Technologien und Jobs implementiert wurden, welche im Unternehmen zum Einsatz kommen. Eine bestimmte Stelle, die gesucht werden sollte, wurde nicht vorgegeben.

Ein Nutzertest begann jeweils mit der Begrüßung des Chatbots *„Hallo! Wie kann ich dir behilflich sein? Gerne helfe ich dir beim Finden einer für dich passenden Stelle“*. Ein Test galt als erfolgreich abgeschlossen, wenn durch die Konversation eine passende Stelle gefunden und angezeigt wurde bzw. wenn die Möglichkeit einer Initiativbewerbung angeboten wurde. Die Teilnahme erfolgte unüberwacht und anonym über die bereitgestellte Webseite mit dem Chatbot-Frontend. Insgesamt nahmen 43 Testpersonen teil und führten eine Konversation mit dem Chatbot.

Von den 43 Teilnehmern kamen 39 zu dem Punkt, an dem eine Stelle bzw. die Möglichkeit einer Initiativbewerbung angezeigt wurde. Diese Konversationen können hiermit aufgrund der definierten Kriterien als erfolgreich angesehen werden.

Zwei Konversationen erreichten zwar das Ziel, Technologien wurden von der NLU-Komponente jedoch falsch erkannt, worauf vom Dialogmanagement falsche Aktionen vorhergesagt wurden. Aus diesem Grund werden die Konversationen der zwei Teilnehmer nicht als erfolgreich angesehen.

Zwei Konversationen erreichten das Ziel nicht. Grund hierfür war, dass eine gewünschte Technologie nicht bekannt war und auf erneute Rückfrage nicht mehr geantwortet wurde. Die bei der Evaluation der NLU-Komponente aufgetretenen Schwierigkeiten bei der Erkennung von Tasks oder Domains konnte im Nutzertest nicht festgestellt werden. Dies liegt vor allem daran, dass mögliche Domains oder Tasks aufgrund des Konversationsverlaufes vom Chatbot angezeigt werden und eine Auswahl über Buttons möglich ist.

Insgesamt können also, wie Abbildung 28 zeigt, 90,7% der Konversationen im Nutzertest als erfolgreich angesehen werden.

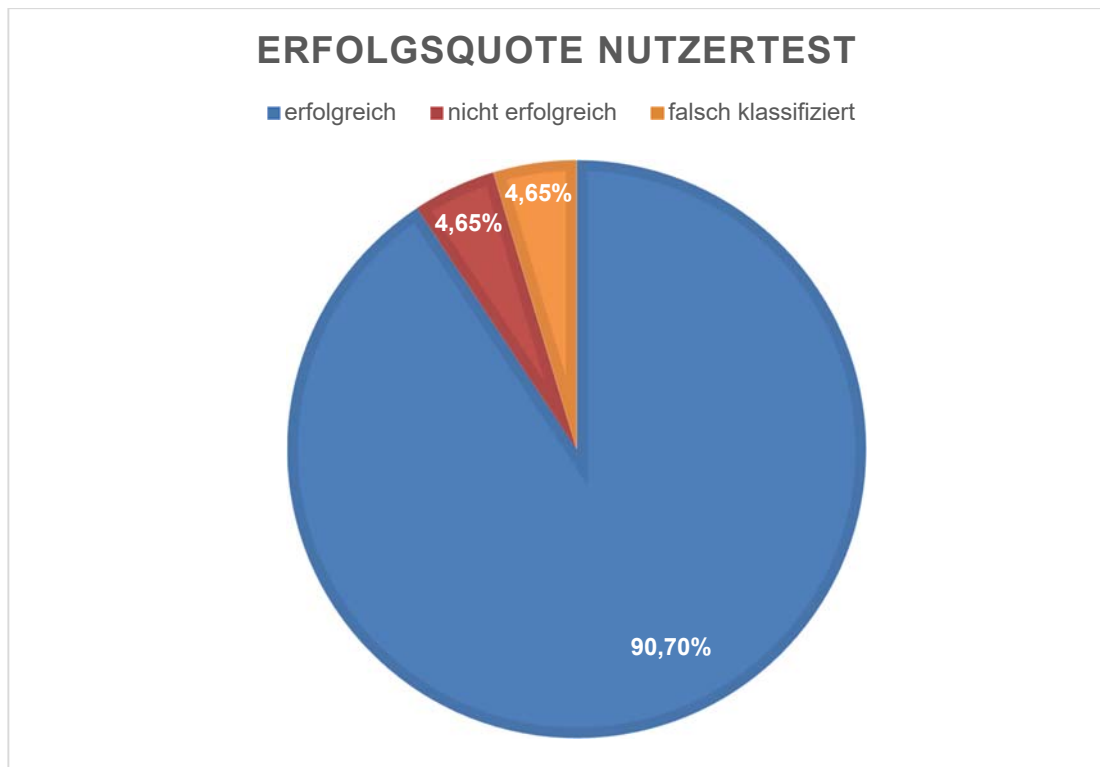


Abbildung 28: Erfolgsquote Nutzertest

Die Logs der Nutzertests befinden sich auszugsweise in Anhang A.3, sowie vollständig auf der CD im Ordner *nutzertests*.

6 Schlussbetrachtung

6.1 Fazit

Diese Arbeit gibt eine Einführung in einige dezidierte Konzepte und Techniken zur Umsetzung von Conversational Interfaces. Ziel war es, einen Überblick über aktuelle Open-Source-NLU-Services zu geben und diese zu evaluieren. Um für die folgende Implementierung aussagekräftige Evaluationsergebnisse zu erhalten, wurden domänenspezifische Trainingsdaten erhoben. Als Resultat der Evaluation hat sich Rasa NLU in Kombination mit Rasa Core als die erfolgversprechendste Option herausgestellt. Dafür wurde ein detailliertes Verständnis der Service-Umgebung vermittelt. Anhand der prototypischen Implementierung in Form eines Chatbots werden Möglichkeiten zur Umsetzung domänenspezifischer Anforderungen im Kontext der Jobsuche aufgezeigt.

Um mithilfe einer Konversation Stellenbeschreibungen finden zu können, wurde eine Datenstruktur für Jobs konzipiert und implementiert, die eine Suche anhand von mehreren Abstraktionsebenen ermöglicht. Wie der Nutzertest zeigte, konnte mit der Datenstruktur eine hohe Erfolgsquote bei Konversationen zur Jobsuche erreicht werden und bietet eine gute Basis für Erweiterungen in zukünftigen Arbeiten.

Weiterhin wurde mit der Logging-Komponente eine Möglichkeit geschaffen, Konversationen zu protokollieren und somit eine iterative Verbesserung des Systems zu ermöglichen. Diese Komponente kann als Grundlage für die effiziente Entwicklung weiterer Chatbots dienen, mithilfe einer Kombination aus Datenbank und Administratoroberfläche.

Wie auch Braun et al. [43] konstatieren, kann die Evaluation der NLU-Services nur als eine Momentaufnahme des aktuellen Zustands der verglichenen Dienste dienen. Für die Auswahl eines Services in anderen Projekten ist demnach eine eigene Evaluation mit aktuellen Versionen der Dienste und domänenspezifischen Trainings- und Testdaten angeraten. Es hat sich gezeigt, dass die Evaluation eines Conversational Interfaces anhand der Klassifikationsleistung der NLU-Komponente eine gute Basis darstellt. Für eine umfassende Einschätzung des Gesamtsystems ist dies jedoch nicht ausreichend und erfordert weitere Maßnahmen wie den, in dieser Arbeit ausgeführten, Nutzertest. In Kombination mit der erarbeiteten Logging-Komponente können Probleme im Konversationsfluss

schnell aufgedeckt werden. Zur Verbesserung eines solchen Systems bietet sich vor allem ein iterativer Ansatz an, bei dem Modelle mit den Daten trainiert werden, die im produktiven Umfeld oder in einem authentischen Testszenario gesammelt werden. Um den Prototypen auf ein einsatzfähiges Niveau im Unternehmen zu bringen, ist es zwingend notwendig, mehrere Iterationen aus Tests und Anpassungen vorzunehmen.

6.2 Ausblick

Der Funktionsumfang des implementierten Prototyps deckt erste Aufgaben eines HR-Bots für Bewerber ab. Basierend auf dieser Arbeit sind eine Vielzahl von Verbesserungen und weiteren Ergänzungen der Implementierung möglich.

Ein wesentlicher Punkt liegt dabei in der Erkennung von Wörtern, die nicht in den Trainingsdaten vorhanden sind. Aufgrund der Implementierung der NLU-Komponente auf Basis von Machine-Learning-Algorithmen wird ermöglicht, dass auch Wörter als Entities erkannt werden, die nicht in den Trainingsdaten vorhanden sind. Im vorliegenden Szenario, bei dem Technologien erkannt werden sollen, stellt dies jedoch ein Problem dar. Wenn in einer Nutzereingabe Technologien erkannt werden, die im Backend nicht hinterlegt sind, schlagen die Aktionen zur Zuordnung der verschiedenen Abstraktionsebenen fehl. Zukünftige Arbeiten könnten sich mit diesem Problem auseinandersetzen.

Eine weitere Verbesserung der NLU-Komponente könnte der Umgang mit zusammengesetzten Wörtern wie z.B. „Frontend-Entwickler“ sein, die vor allem in der deutschen Sprache auftreten. Die vorliegende Implementierung erkennt solche Wörter nur als Ganzes und dementsprechend auch nur als eine Entity. Da eine Integration dieser Funktion in Rasa NLU zum aktuellen Zeitpunkt nicht geplant scheint, bietet sich dies als Möglichkeit für eine zukünftige Erweiterung an.

Aufgrund der raschen Entwicklung im Bereich von Open-Source-Technologien für Conversational Interfaces bietet der Markt auch in Zukunft viel Potential für weitere Projekte und Evaluationen.

Anhang A:

A.1 Zuordnung der Stellenbeschreibungen

Jobtitel: _____

Task:

entwickeln		designen		administrieren		coachen		analysieren	

Domain:

web		frontend		security		agile		marketing	
mobile		fullstack		personal		testing			
backend		machinelearning		business		recht			
devops		datascience		finanzen		consulting			

Technologie/Herangehensweise/Methode:

angular		sql		word	
java		nosql		excel	
spring		mysql		powerpoint	
kotlin		sqlite		prozessdesign	
android		postgresql		prozessanalyse	
swift		mongodb		businessanalyse	
ios		couchdb		geschäftsmodell	
typescript		nofs		strategieberatung	
javascript		verschlüsselung		bewerbermanagement	
html		docker		weiterbildungsmanagement	
css		openshift		personalverwaltung	
sass		continuousintegration		projektplan	
less		continuousdeployment		gantt	
react		releasemanagement		requirementsengineering	
vue		konfigurationsmanagement		projektmanagement	
jquery		systemadministration		controlling	
nodejs		systemintegration		buchführung	
npm		software_testing		buchhaltung	
python		teststrategie		arbeitsrecht	
c		testautomatisierung		steuerrecht	
buildtools		cucumber		sozialversicherungsrecht	
c#		mocha		workshop	
nlu		chai		schulung	
nlp		supertest		scrum	
tensorflow		unittest		kanban	
scikit-learn		integrationstest			
rasa		end-to-endtest			
cui		accessibility			
dialogflow		usability			
alexa		designthinking			
snips		photoshop			
clustering		illustrator			
watson		indesign			
azure		sketch			
softwarearchitektur		invision			
arc		office			

A.2 Evaluationsergebnisse der NLU-Komponente

Intents:

	precision	recall	f1-score	support
ask_for_technologies	1.00	1.00	1.00	8
ask_speculative_application	1.00	0.67	0.80	3
confirm	0.91	0.91	0.91	11
deny	1.00	0.83	0.91	6
dontknow	0.80	1.00	0.89	4
enter_data	1.00	0.85	0.92	33
find_job	0.92	0.97	0.94	35
goodbye	1.00	0.50	0.67	2
greet	0.91	0.91	0.91	22
greet+find_job	1.00	0.50	0.67	2
help	1.00	1.00	1.00	2
thankyou	1.00	1.00	1.00	1
you_are_welcome	1.00	1.00	1.00	4
avg / total	0.95	0.90	0.92	133

	precision	recall	f1-score	support
ask_for_technologies	0.88	1.00	0.93	7
confirm	1.00	0.88	0.93	8
deny	1.00	1.00	1.00	2
dontknow	0.38	0.75	0.50	4
enter_data	1.00	0.83	0.91	53
find_job	0.90	0.96	0.93	28
goodbye	1.00	0.50	0.67	4
greet	0.96	1.00	0.98	22
greet+find_job	0.00	0.00	0.00	2
help	1.00	1.00	1.00	2
thankyou	1.00	0.50	0.67	2
you_are_welcome	0.00	0.00	0.00	1
avg / total	0.92	0.87	0.89	135

	precision	recall	f1-score	support
ask_for_technologies	1.00	0.67	0.80	3
ask_speculative_application	1.00	1.00	1.00	2
confirm	0.75	0.60	0.67	5
deny	1.00	1.00	1.00	6
dontknow	0.50	0.83	0.62	6
enter_data	1.00	0.90	0.95	41
find_job	0.79	1.00	0.89	31
goodbye	0.00	0.00	0.00	3
greet	1.00	0.75	0.86	16
greet+find_job	0.86	0.60	0.71	10
help	1.00	0.50	0.67	2
thankyou	1.00	1.00	1.00	4
you_are_welcome	0.80	0.80	0.80	5
avg / total	0.88	0.84	0.85	134

	precision	recall	f1-score	support
ask_for_technologies	1.00	1.00	1.00	1
ask_speculative_application	1.00	1.00	1.00	5
confirm	1.00	0.83	0.91	12
deny	0.75	0.50	0.60	6
dontknow	0.50	0.60	0.55	5
enter_data	1.00	0.94	0.97	32
find_job	0.90	1.00	0.95	27
goodbye	0.00	0.00	0.00	2
greet	0.91	0.95	0.93	22
greet+find_job	0.83	0.50	0.62	10
help	1.00	0.40	0.57	5
thankyou	0.50	1.00	0.67	2
you_are_welcome	0.83	1.00	0.91	5
avg / total	0.89	0.85	0.86	134

	precision	recall	f1-score	support
ask_for_technologies	1.00	1.00	1.00	3
ask_speculative_application	1.00	0.75	0.86	4
confirm	0.88	1.00	0.93	7
deny	0.50	1.00	0.67	2
dontknow	0.43	1.00	0.60	3
enter_data	1.00	0.85	0.92	54
find_job	0.97	1.00	0.98	29
goodbye	1.00	0.33	0.50	3
greet	0.92	0.92	0.92	13
greet+find_job	1.00	0.75	0.86	4
help	1.00	0.80	0.89	5
thankyou	1.00	1.00	1.00	4
you_are_welcome	1.00	0.67	0.80	3
avg / total	0.96	0.89	0.91	134

Entities:

	precision	recall	f1-score	support
domain	1.00	0.36	0.53	11
formOfEmployment	1.00	1.00	1.00	18
jobTask	1.00	0.50	0.67	6
nameApplicant	1.00	1.00	1.00	16
no_entity	0.97	1.00	0.98	512
technology	1.00	0.89	0.94	57
avg / total	0.97	0.97	0.97	620

	precision	recall	f1-score	support
domain	1.00	0.73	0.85	15
formOfEmployment	1.00	0.88	0.94	17
jobTask	1.00	0.67	0.80	9
nameApplicant	1.00	0.94	0.97	17
no_entity	0.97	1.00	0.99	462
technology	1.00	0.95	0.97	77
avg / total	0.98	0.98	0.98	597

	precision	recall	f1-score	support
domain	1.00	0.67	0.80	9
formOfEmployment	0.91	0.95	0.93	22
jobTask	1.00	0.80	0.89	5
nameApplicant	1.00	0.81	0.89	21
no_entity	0.98	1.00	0.99	586
technology	0.99	0.99	0.99	67
avg / total	0.98	0.98	0.98	710

	precision	recall	f1-score	support
domain	1.00	0.50	0.67	12
formOfEmployment	1.00	0.76	0.87	17
jobTask	1.00	0.67	0.80	9
nameApplicant	0.96	0.92	0.94	25
no_entity	0.97	1.00	0.98	649
technology	0.90	0.88	0.89	60
avg / total	0.97	0.97	0.97	772

	precision	recall	f1-score	support
domain	1.00	0.47	0.64	15
formOfEmployment	1.00	0.93	0.97	15
jobTask	1.00	0.75	0.86	4
nameApplicant	1.00	1.00	1.00	13
no_entity	0.96	1.00	0.98	466
technology	0.99	0.88	0.93	84
avg / total	0.97	0.97	0.96	597

A.3 Logs Nutzertest (Auszug)

CONVERSATION #4

Bot did: action_listen

User said: hi, ich suche nach entwicklerstellen

Bot did: utter_askTechnology

Bot said: Welche Technologien würdest du gerne einsetzen?

Bot did: action_listen

User said: angular

Bot did: action_match_job_slots

Bot said: Für welches dieser Gebiete interessierst du dich am meisten?

Bot did: action_listen

User said: /enter_data{"domain":"frontend"}

Bot did: utter_askFormOfEmployment

Bot said: In welchem Beschäftigungsverhältnis würdest du gerne bei uns arbeiten?

Bot did: action_listen

User said: /enter_data{"formOfEmployment": "vollzeit"}

Bot did: action_find_job

Bot said: Ich habe folgende Stellenanzeigen für dich gefunden:

Bot said: Frontend Developer, Web Entwickler Java Ee

Bot said: Danke für das Testen dieses Prototypen! :)

Bot did: action_listen

CONVERSATION #5

Bot did: action_listen

User said: ich will programmieren

Bot did: action_match_job_slots

Bot did: utter_askTechnology

Bot said: Welche Technologien würdest du gerne einsetzen?

Bot did: action_listen

User said: vue.js

Bot did: action_match_job_slots

Bot said: Für welches dieser Gebiete interessiert du dich am meisten?

Bot did: action_listen

User said: /enter_data{"domain":"fullstack"}

Bot did: utter_askFormOfEmployment

Bot said: In welchem Beschäftigungsverhältnis würdest du gerne bei uns arbeiten?

Bot did: action_listen

User said: /enter_data{"formOfEmployment": "vollzeit"}

Bot did: action_find_job

Bot did: utter_askSpeculativeApplication

Bot said: Leider kann ich für deine bisherigen Angaben keine offene Stelle finden. Möchtest du dich stattdessen initiativ bewerben?

Bot did: action_listen

User said: nein

Bot did: utter_bye

Bot said: Bis bald

Bot did: action_listen

CONVERSATION #39

Bot did: action_listen

User said: Und los

Bot did: action_match_job_slots

Bot did: utter_askTechnology

Bot said: Welche Technologien würdest du gerne einsetzen?

Bot did: action_listen

User said: PHP

Bot did: action_default_fallback

Bot said: Tut mir Leid, das muss ich noch lernen. Kannst du das bitte noch einmal anders formulieren?

Bot did: action_listen

Quellenverzeichnis

- [1] M. Ballve, „Messaging apps have finally caught up to social networks in user numbers and now dominate mobile“, *Business Insider*. [Online]. Verfügbar unter: <http://uk.businessinsider.com/messaging-apps-have-completely-overtaken-social-networks-to-become-the-dominant-platforms-on-phones-2015-4>. [Zugegriffen: 03-Apr-2018].
- [2] M. Meeker, „2017 Internet Trends Report“. [Online]. Verfügbar unter: <http://www.kpcb.com/internet-trends>. [Zugegriffen: 29-März-2018].
- [3] N. Babich, „How Conversational User Interfaces Will Change Our Lives“, *Adobe Blog*, 26-Okt-2017. [Online]. Verfügbar unter: <https://theblog.adobe.com/how-conversational-user-interfaces-will-change-our-lives/>. [Zugegriffen: 30-März-2018].
- [4] M. McTear, Z. Callejas, und D. Griol, *The Conversational Interface: Talking to Smart Devices*, 1st ed. 2016. New York, NY: Springer, 2016.
- [5] „The Conversational UI and Why It Matters | Actions on Google“, *Google Developers*. [Online]. Verfügbar unter: <https://developers.google.com/actions/design/>. [Zugegriffen: 29-März-2018].
- [6] A. Jain, „Chatbots Survey 2017 [Chatbot Market Research Report]“. [Online]. Verfügbar unter: <https://www.slideshare.net/Mobileappszen/chatbots-survey-2017-chatbot-market-research-report>. [Zugegriffen: 03-Apr-2018].
- [7] Oracle, „Can Virtual Experiences Replace Reality?“ [Online]. Verfügbar unter: <https://go.oracle.com/LP=43079>. [Zugegriffen: 01-Mai-2018].
- [8] „Chatbots Market Size, Share, Trend, Growth And Forecast To 2022“. [Online]. Verfügbar unter: <http://www.credenceresearch.com/report/chatbots-market>. [Zugegriffen: 03-Apr-2018].
- [9] S. Huffman und R. Chandra, „Your Google Assistant is getting better across devices, from Google Home to your phone“, *Google*, 17-Mai-2017. [Online]. Verfügbar unter: <https://www.blog.google/products/assistant/your-assistant-getting-better-on-google-home-and-your-phone/>. [Zugegriffen: 01-Mai-2018].
- [10] „an over-automated recruitment process leaves candidates frustrated and missing personal connections, finds randstad US study.“ [Online]. Verfügbar unter: <http://www.randstadusa.com/about/news/an-over-automated-recruitment-process-leaves-candidates-frustrated-and-missing-personal-connections-finds-randstad-us-study/>. [Zugegriffen: 20-Apr-2018].
- [11] D. Alex, „3 stats that show chatbots are here to stay“, *VentureBeat*, 26-Aug-2016. [Online]. Verfügbar unter: <https://venturebeat.com/2016/08/26/3-stats-that-show-chatbots-are-here-to-stay/>. [Zugegriffen: 20-Apr-2018].
- [12] NetFederation GmbH, „NetFed HR Benchmark“. [Online]. Verfügbar unter: <http://www.hr-benchmark.de/human-resources-benchmark-2018/benchmark>. [Zugegriffen: 04-Apr-2018].
- [13] „A How-To Guide For Using A Recruitment Chatbot | Ideal“. [Online]. Verfügbar unter: <https://ideal.com/recruitment-chatbot/>. [Zugegriffen: 15-Nov-2018].

- [14] PricewaterhouseCoopers, „Consumer Intelligence Series: Prepare for the voice revolution“, *PwC*. [Online]. Verfügbar unter: <https://www.pwc.com/us/en/services/consulting/library/consumer-intelligence-series/voice-assistants.html>. [Zugegriffen: 17-Juli-2018].
- [15] J. R. Searle, *Speech Acts: An Essay in the Philosophy of Language*, New Ed. Cambridge: Cambridge University Press, 1969.
- [16] H. H. Clark, *Using Language*. Cambridge University Press, 1996.
- [17] S. Quarteroni, „Natural Language Processing for Industry: ELCA’s experience“, *Inform.-Spektrum*, März 2018.
- [18] A. M. Turing, „Computing Machinery and Intelligence“, *Mind*, Bd. 59, Nr. 236, S. 433–460, 1950.
- [19] J. Weizenbaum, „ELIZA - Computer Program for the Study of Natural Language Communication Between Man and Machine“, *Commun ACM*, Bd. 9, Nr. 1, S. 36–45, Jan. 1966.
- [20] „IBM Archives: IBM Shoebox“, 23-Jan-2003. [Online]. Verfügbar unter: http://sysrun.haifa.il.ibm.com/ibm/history/exhibits/specialprod1/specialprod1_7.html. [Zugegriffen: 22-Apr-2018].
- [21] R. S. Wallace, „The anatomy of A.L.I.C.E.“, in *Parsing the Turing Test*, 2009, S. 181–210.
- [22] „SmarterChild“. [Online]. Verfügbar unter: <https://botwiki.org/bot/smarterchild>. [Zugegriffen: 29-Okt-2018].
- [23] N. Perlroth, „Siri Was Born A Man And Other Things You Don’t Know About Apple’s New Personal Assistant“, *Forbes*. [Online]. Verfügbar unter: <https://www.forbes.com/sites/nicoleperlroth/2011/10/12/siri-was-born-a-man-and-other-things-you-dont-know-about-apples-new-personal-assistant/>. [Zugegriffen: 29-Okt-2018].
- [24] „Siri“, *Apple (Deutschland)*. [Online]. Verfügbar unter: <https://www.apple.com/de/siri/>. [Zugegriffen: 29-Okt-2018].
- [25] „Google Assistant – dein persönlicher Helfer“, *Google Assistant – dein persönlicher Helfer*. [Online]. Verfügbar unter: <https://assistant.google.com/>. [Zugegriffen: 29-Okt-2018].
- [26] „Cortana | Die persönliche digitale Assistentin | Windows“. [Online]. Verfügbar unter: www.microsoft.com/de-de/windows/cortana. [Zugegriffen: 29-Okt-2018].
- [27] „Amazon Alexa“. [Online]. Verfügbar unter: <https://developer.amazon.com/de/alexa>. [Zugegriffen: 29-Okt-2018].
- [28] C. Noessel, *Designing Agentive Technology: AI That Works for People*, 1st edition. Brooklyn, NY: Rosenfeld Media, 2017.
- [29] S. J. Russell und P. Norvig, *Künstliche Intelligenz. Ein moderner Ansatz*, 2., überarb. A. Pearson Studium, 2004.
- [30] „Planning your intents and entities“. [Online]. Verfügbar unter: <https://console.bluemix.net/docs/services/conversation/intents-entities.html#planning-your-intents>. [Zugegriffen: 28-Juni-2018].

- [31] E. D. Liddy, „Natural Language Processing“, in *Encyclopedia of Library and Information Science*, 2nd Ed., NY: Marcel Decker, Inc, 2001.
- [32] L. Deng und Y. Liu, *Deep Learning in Natural Language Processing*, 1st Aufl. Springer Publishing Company, Incorporated, 2018.
- [33] B. Healey, „NLP vs. NLU: What’s the Difference?“, *Medium*, 05-Okt-2016. [Online]. Verfügbar unter: <https://medium.com/@lolatravel/nlp-vs-nlu-whats-the-difference-d91c06780992>. [Zugegriffen: 06-Mai-2018].
- [34] T. Young, D. Hazarika, S. Poria, und E. Cambria, „Recent Trends in Deep Learning Based Natural Language Processing“, *ArXiv170802709 Cs*, Aug. 2017.
- [35] G. Tur, A. Celikyilmaz, X. He, D. Hakkani-Tur, und L. Deng, „Deep Learning in Conversational Language Understanding“, in *Deep Learning in Natural Language Processing*. (eds. Li Deng and Yang Liu), Springer, 2017.
- [36] „Natural Language Understanding (NLU)“, *Microsoft Research*. [Online]. Verfügbar unter: <https://www.microsoft.com/en-us/research/project/natural-language-understanding-nlu/>. [Zugegriffen: 06-Mai-2018].
- [37] D. Jurafsky und J. H. Martin, *Speech and Language Processing*, 3rd edition draft. Upper Saddle River, NJ, 2017.
- [38] S. Ramamoorthy, „Chatbots with Seq2Seq“. [Online]. Verfügbar unter: <http://suriyadeepan.github.io/2016-06-28-easy-seq2seq/>. [Zugegriffen: 28-Nov-2018].
- [39] C. Sutton und A. McCallum, „An Introduction to Conditional Random Fields“, *Found Trends Mach Learn*, Bd. 4, Nr. 4, S. 267–373, Apr. 2012.
- [40] E. Alpaydin, *Maschinelles Lernen*, 1. Aufl. München: Oldenbourg Wissenschaftsverlag, 2008.
- [41] T. M. Mitchell, *Machine Learning*, 1. Aufl. New York, NY, USA: McGraw-Hill, Inc., 1997.
- [42] S. Hochreiter und J. Schmidhuber, „Long Short-Term Memory“, *Neural Comput*, Bd. 9, Nr. 8, S. 1735–1780, Nov. 1997.
- [43] D. Braun, A. Hernandez-Mendez, F. Matthes, und M. Langen, „Evaluating Natural Language Understanding Services for Conversational Question Answering Systems“, in *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, 2017, S. 174–185.
- [44] „spaCy 101: Everything you need to know · spaCy Usage Documentation“. [Online]. Verfügbar unter: <https://spacy.io/usage/spacy-101>. [Zugegriffen: 20-Juni-2018].
- [45] F. Pedregosa u. a., „Scikit-learn: Machine Learning in Python“, *J. Mach. Learn. Res.*, Bd. 12, S. 2825–2830, 2011.
- [46] „Linguistic Features · spaCy Usage Documentation“. [Online]. Verfügbar unter: <https://spacy.io/usage/linguistic-features>. [Zugegriffen: 20-Juni-2018].
- [47] T. Bocklisch, J. Faulkner, N. Pawlowski, und A. Nichol, „Rasa: Open Source Language Understanding and Dialogue Management“, *ArXiv171205181 Cs*, Dez. 2017.

- [48] „German · spaCy Models Documentation“. [Online]. Verfügbar unter: <https://spacy.io/models/de>. [Zugegriffen: 30-Juni-2018].
- [49] Martín Abadi u. a., *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015.
- [50] „tensorflow: Computation using data flow graphs for scalable machine learning“, 30-Juni-2018. [Online]. Verfügbar unter: <https://github.com/tensorflow/tensorflow>. [Zugegriffen: 30-Juni-2018].
- [51] A. Nichol, „Supervised Word Vectors from Scratch in Rasa NLU“, *Rasa Blog*, 18-Apr-2018. [Online]. Verfügbar unter: <http://blog.rasa.com/supervised-word-vectors-from-scratch-in-rasa-nlu/>. [Zugegriffen: 06-Mai-2018].
- [52] L. Wu, A. Fisch, S. Chopra, K. Adams, A. Bordes, und J. Weston, „StarSpace: Embed All The Things!“, *CoRR*, Bd. abs/1709.03856, 2017.
- [53] „Pull-Request: Fast language-agnostic NER CRF without spaCy“, 23-Juli-2018. [Online]. Verfügbar unter: https://github.com/RasaHQ/rasa_nlu/pull/1095. [Zugegriffen: 23-Juli-2018].
- [54] A. Coucke u. a., „Snips Voice Platform: an embedded Spoken Language Understanding system for private-by-design voice interfaces“, *ArXiv180510190 Cs*, Mai 2018.
- [55] S. Nissen, „Large Scale Reinforcement Learning using Q-SARSA (λ) and Cascading Neural Networks“, 2007.
- [56] M. D. Scholefield, *A rigid, lightweight, dead-simple intent parser. Contribute to MatthewScholefield/padaos development by creating an account on GitHub*. 2018.
- [57] „Padatious documentation“. [Online]. Verfügbar unter: <https://padatious.readthedocs.io/en/latest/>. [Zugegriffen: 25-Sep-2018].
- [58] „Padatious Intent Parser“, *Mycroft Community Forum*, 25-Sep-2018. [Online]. Verfügbar unter: <https://community.mycroft.ai/t/padatious-intent-parser/4647/3>. [Zugegriffen: 26-Sep-2018].
- [59] „Mycroft Github Repository“, *GitHub*. [Online]. Verfügbar unter: <https://github.com/MycroftAI>. [Zugegriffen: 25-Sep-2018].
- [60] S. Raschka, *Python Machine Learning*, 1. Aufl. Packt Publishing, 2015.
- [61] „Choosing a Rasa NLU Pipeline“. [Online]. Verfügbar unter: https://rasa.com/docs/nlu/choosing_pipeline. [Zugegriffen: 22-Sep-2018].
- [62] „Many intents · Issue #69 · snipsco/snips-nlu-rs“, *GitHub*. [Online]. Verfügbar unter: <https://github.com/snipsco/snips-nlu-rs/issues/69>. [Zugegriffen: 22-Sep-2018].
- [63] „A chat widget easy to connect to chatbot platforms such as Rasa Core: mrbot-ai/rasa-webchat“, 16-Nov-2018. [Online]. Verfügbar unter: <https://github.com/mrbot-ai/rasa-webchat>. [Zugegriffen: 18-Nov-2018].
- [64] „React – A JavaScript library for building user interfaces“. [Online]. Verfügbar unter: <https://reactjs.org/>. [Zugegriffen: 12-Dez-2018].

- [65] „Socket.IO“, *Socket.IO*. [Online]. Verfügbar unter: <https://socket.io/index.html>. [Zugegriffen: 18-Nov-2018].
- [66] T. Hughes, „Entity extraction with the new lookup table feature in Rasa NLU“, *Medium*, 13-Sep-2018. .
- [67] „4.2. Feature extraction — scikit-learn 0.20.0 documentation“. [Online]. Verfügbar unter: http://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction. [Zugegriffen: 03-Okt-2018].
- [68] „Domain Format“. [Online]. Verfügbar unter: <https://rasa.com/docs/core/domains>. [Zugegriffen: 27-Sep-2018].
- [69] J. D. Williams und G. Zweig, „End-to-end LSTM-based dialog control optimized with supervised and reinforcement learning“, *ArXiv160601269 Cs*, Juni 2016.
- [70] „SQLite Home Page“. [Online]. Verfügbar unter: <https://www.sqlite.org/index.html>. [Zugegriffen: 08-Okt-2018].
- [71] „MongoDB for GIANT Ideas“, *MongoDB*. [Online]. Verfügbar unter: <https://www.mongodb.com/index>. [Zugegriffen: 04-Nov-2018].
- [72] „Node.js“, *Node.js*. [Online]. Verfügbar unter: <https://nodejs.org/en/>. [Zugegriffen: 04-Nov-2018].
- [73] „Express - Node.js web application framework“. [Online]. Verfügbar unter: <http://expressjs.com/>. [Zugegriffen: 04-Nov-2018].
- [74] „Angular“. [Online]. Verfügbar unter: <https://angular.io/>. [Zugegriffen: 04-Nov-2018].
- [75] „Docker“. [Online]. Verfügbar unter: <https://www.docker.com/index.html>. [Zugegriffen: 29-Nov-2018].
- [76] „OpenShift: Container Application Platform by Red Hat, Built on Docker and Kubernetes“. [Online]. Verfügbar unter: <https://www.openshift.com>. [Zugegriffen: 29-Nov-2018].
- [77] „AWS | Amazon Elastic Compute Cloud (EC2) – Cloud Server“, *Amazon Web Services, Inc.* [Online]. Verfügbar unter: <https://aws.amazon.com/de/ec2/>. [Zugegriffen: 29-Nov-2018].