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

Marco Maisel
Fakultät Druck & Medien
Hochschule der Medien
Stuttgart, Germany
mm248@hdm-stuttgart.de

***Kurzfassung*— Für die Human-Resources-Abteilung der Firma adorsys GmbH & Co. KG wurde ein Prototyp eines Chatbots für Bewerber entwickelt, dessen Datenverarbeitung in Zukunft ausschließlich auf firmeninternen Servern stattfinden soll. Dazu wurden zuerst mehrere aktuelle Open Source NLU Services betrachtet und verglichen. Als ein wesentliches Vergleichskriterium wurde die Leistungsfähigkeit des Klassifikators herangezogen. Um für diesen speziellen Anwendungsfall aussagekräftige Ergebnisse zu erhalten, wurden domänenspezifische Trainingsdaten gesammelt und zur Evaluation verwendet. Auf Basis der vorangegangenen Evaluation wurde im Anschluss unter Verwendung einer dieser Services ein Chatbot für Bewerber prototypisch konzipiert und realisiert. Der Fokus des Prototyps lag auf dem Szenario der Jobsuche anhand verschiedener Suchkriterien. Um mithilfe einer Konversation Stellenbeschreibungen finden zu können, wurde eine Datenstruktur konzipiert und implementiert, die dies anhand von mehreren Abstraktionsebenen ermöglicht. Weiterhin wurde die Möglichkeit geschaffen, Konversationen zu protokollieren und somit eine iterative Verbesserung des Systems zu schaffen.**

I. EINLEITUNG

Conversational Interfaces ermöglichen die Interaktion mit Computern durch Kommunikation im Stile einer Konversation [1]. 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.

A. *Gründe für Open-Source-Technologien*

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 gewonnen.

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.

II. VERWANDTE ARBEITEN

Das Vorgehen zur Evaluation erfolgt in Anlehnung an Braun et al. [2], die mehrere NLU-Dienste verglichen haben. Der Fokus lag dabei jedoch nicht auf Open-Source-Technologien. Ihr Ansatz beschränkt sich auf die Evaluation der NLU-Komponente und nicht auf ein Chatbot-Gesamtsystem. Sie konstatieren, dass es zum Zeitpunkt ihrer Veröffentlichung Ende 2017 keine weitere systematische Bewertung eines bestimmten NLU-Dienstes oder einen Vergleich mehrerer Dienste, gibt. Außerdem weisen sie auf die Notwendigkeit einer eigenen Evaluation mit domänenspezifischen Trainingsdaten hin. Snips SAS veröffentlichte wäh-

rend des Schreibens dieser Arbeit einige Benchmark-Ergebnisse ihres NLU-Dienstes im Vergleich zu anderen Diensten und umfasst dabei auch Rasa NLU in Kombination mit spaCy [3].

III. Theoretischer Hintergrund

A. Architektur eines Chatbots

Die typische Architektur eines Chatbots besteht aus einem Frontend zur Nutzereingabe und drei Backend-Komponenten:

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.

B. Natural Language Processing und Understanding

Natural Language Processing (NLP) befasst sich mit Techniken zur Analyse für die Verarbeitung von Texten in natürlicher Sprache [4]. 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 Computermodelle und Anwendungen entwickelt, die die Bedeutung natürlicher Sprache verstehen und daraus Schlüsse ziehen können [5]. 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 [6]. Typische Anwendungen sind Spracherkennung, Dialogsysteme, lexikalische Analyse, maschinelle Übersetzung und Systeme zur Fragenbeantwortung sowie zur Stimmungsanalyse [7].

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 [8]. Zu den zukünftigen Möglichkeiten der NLU könnte gehören, die formale Bedeutung der Inhalte natürlicher Sprache zu extrahieren und in maschinell ausführbare Programme zu transformieren [9]. 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 [10]. 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.

IV. Evaluation

Um eine geeignete NLU-Komponente für die protypische Umsetzung des HR-Chatbots zu finden, wurden vier Open Source Services zur Evaluation ausgewählt. Ziel dieser Services ist die Extraktion von strukturierten Informationen aus den unstrukturierten Eingaben in natürlicher Sprache. Vorabkriterium für die Wahl der vier Services war die Möglichkeit Informationen aus deutschsprachigen Texten extrahieren zu können. Die Auswahl für einen der Services für die anschließende Umsetzung des HR-Chatbots geschieht anhand der Leistungsfähigkeit des Klassifikators mit domänenspezifischen Daten.

Die Konzeption und die Durchführung der Evaluation orientierte sich am, von Braun et al., 2017 [2] vorgestellten, Ansatz zur Evaluation von NLU-Services.

A. Übersicht der NLU-Services

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.

a) Rasa Pipeline mit Backend spaCy und scikit-learn

Der erste Schritt in der Pipeline ist die Tokenisierung des eingegebenen Textes mithilfe der spaCy [11] 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) [12]. Jedes Token wird in einen Vektor abgebildet und anschließend zu einer Repräsentation des gesamten Satzes vereint. Der scikit-learn [13] 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 [14].
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. Die Verwendung dieses Modells bietet den Vorteil, dass 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 hauptsächlich domänenspezifische Anfragen erhält.

b) Rasa Pipeline mit Backend TensorFlow

In der Pipeline mit TensorFlow [15] werden keine vortrainierten Wortvektoren verwendet. Stattdessen werden Embeddings sowohl für die Wörter an sich, als auch für die Intents spezifisch für den bestehenden Datensatz gelernt [16]. 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 [17], 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 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 [16]. 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.

2) Snips NLU

Snips NLU ist ein Open-Source-NLU-Tool zur Intent-Klassifizierung und Entity-Extraktion.
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. 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 gefunden Entities auf bestimmte vordefinierte Entities wie Zeiten, Temperaturen oder Nummer analysiert und können bei Bedarf formatiert werden [3].

3) Mycroft Padatious

Padatious ist ein Open Source Intent-Parser der Firma Mycroft AI. Er basiert auf der *Fast Artificial Neural Network* (FANN) Bibliothek [18].
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 Wahrscheinlich 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* [19] ein zweiter Parser eingesetzt, der mit Hilfe von Regular Expressions gewährleistet, dass genaue Übereinstimmungen mit den Trainingsdaten korrekt erkannt werden [20]–[22]. 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.

B. Datensammlung- und verarbeitung

Zur Erhebung von authentischen Trainings- und Testdaten wurde ein Szenario entworfen, um herauszufinden, wie potenzielle Bewerber mit einem Chatbot interagieren würden. Dabei sollten sich Nutzer in die Rolle eines Bewerbers versetzen und eine Konversation mit einem Chatbot führen. Im Kontext dieses Szenarios wurden auf einer Firmenkontaktmesse und über ein Online-Umfrage-Tool ein Datensatz mit Antworten bzw. Anfragen erstellt. Der dadurch entstandene Datensatz umfasst 272 Utterances in deutscher Sprache, welcher vom Autor manuell gelabelt wurde. Er beinhaltet drei verschiedene Intents und acht Entities:

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*).

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*).

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*).

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 [2].

C. *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 [23]. 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.

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 [24].
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. Das 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.

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

D. *Evaluationsergebnisse*

Alle NLU-Services wurden mit dem vorgestellten Datensatz trainiert und 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 (∑ Intents) und ein Ergebnis für Entities ergab (∑ Entities). Als ausschlaggebendes Kriterium für die Verwendung eines Services wurde der berechnete f1-score herangezogen. Die Ergebnisse der Evaluation sind in Tabelle 2-5 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
∑ Intents	0,938	0,932	0,935
Entities	**precision**	**recall**	**f1-score**
∑ 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
∑ Intents	0,97	0,966	0,968
Entities	**precision**	**recall**	**f1-score**
∑ 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
∑ Intents	0,98	0,945	0,962
Entities	**precision**	**recall**	**f1-score**
∑ 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
∑ Intents	0,825	0,76	0,791
Entities	**precision**	**recall**	**f1-score**
∑ 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 Komponenten 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 [25].

V. Entwurf des Chatbot-Prototyps

A. Szenario

Kernfunktion des Chatbot-Prototyps soll die Suche nach einer Stelle in Form eines Dialoges sein. 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.

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.

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.

B. *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.

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.

VI. Implementierung des Prototyps

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:

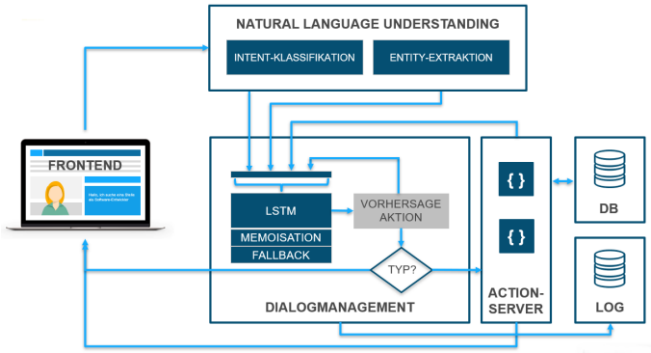

Abbildung 1: Architektur des Chatbot-Prototyps

1. Nachrichten können vom Nutzer über ein Chat-Interface in einem Web-Frontend eingegeben werden.
2. Die NLU-Komponente verarbeitet die unstrukturierten Daten aus der Nachricht des Nutzers und extrahiert daraus strukturierte Daten in Form von Intents und Entities.
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.
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.
5. Die Logging-Komponente bietet eine detaillierte Möglichkeit zur Speicherung von Gesprächsinformationen.

A. *Umsetzung der Frontend-Komponente*

Für die Implementierung des Chatbot-Frontends wurde *rasa-webchat* [26] verwendet. Dieses wurde so angepasst, dass es zum Design der Webseite der adorsys GmbH & Co. KG passt. Über das WebSocket-Protokoll wird eine bidirektionale Client-Server-Kommunikation ermöglicht. Im Backend wird dazu ein WebSocket bereitgestellt, mit welchem sich das Chatbot-Frontend verbinden kann.

B. *Umsetzung der NLU-Komponente*

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.

Für die Implementierung dieses Chatbot-Prototyps wurde eine Pipeline umgesetzt, welche hauptsächlich auf den, mit TensorFlow implementieren, Embeddings zur Intent-Klassifikation aufbaut.

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 [27].
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 [28]. 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.

C. *Umsetzung der Dialogmanagement-Komponente*

Das Dialogmanagements 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. Abbildung 2 zeigt den Prozess des Dialogmanagements vom Eingang einer Nachricht über die NLU-Komponente bis zur Ausgabe einer Antwort.

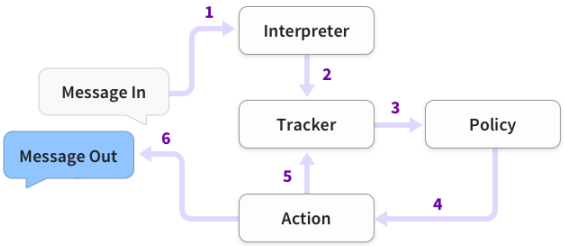

Abbildung 2: Architektur Rasa Core

1. Eine vom Nutzer eingegebene Nachricht wird vom System empfangen und an einen NLU-Service weitergeleitet, der den natürlichsprachlichen Text 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. Für jede Session gibt es jeweils ein neues Tracker-Objekt.
3. Die Policy erhält den aktuellen Zustand des Trackers-Objekts. Im Rahmen dieses Prototyps wurden drei Policies implementiert: *Memoization Policy, Keras Policy* und *Fallback Policy*.
4. In jeder Iteration des Dialogs treffen die Policies eine Voraussage über die nächste auszuführende Aktion. Eine Aktion kann eine Antwort an den Nutzer in Form einer Nachricht sein oder die Ausführung einer beliebigen Funktion. Die Policy, deren Voraussage die höchste Wahrscheinlichkeit aufweist, wird 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 Trainings-

daten 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) [29]. 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 [30]. 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. 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 [14].

1) Benutzerdefinierte Aktionen zur Jobsuche

Wie 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 ausführt, die die Eingabe und die bisher bekannten Informationen in Form von gesetzten Slots mit einer Datenbank abgleichen. Die dafür im Backend-Server implementierte Klasse *ActionMatchJobSlots* 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 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.

D. Logging-Komponente

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. Rasa Core speichert 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. 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.

E. 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 Evaluationsschritt 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.

1) Evaluation der NLU-Komponente

Der 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
∑ Intents	0,92	0,87	0,89
Entities	**precision**	**recall**	**f1-score**
∑ Entities	0,97	0,97	0,97

Tabelle 6: NLU-Evaluation 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). 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.
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 dargestellt:

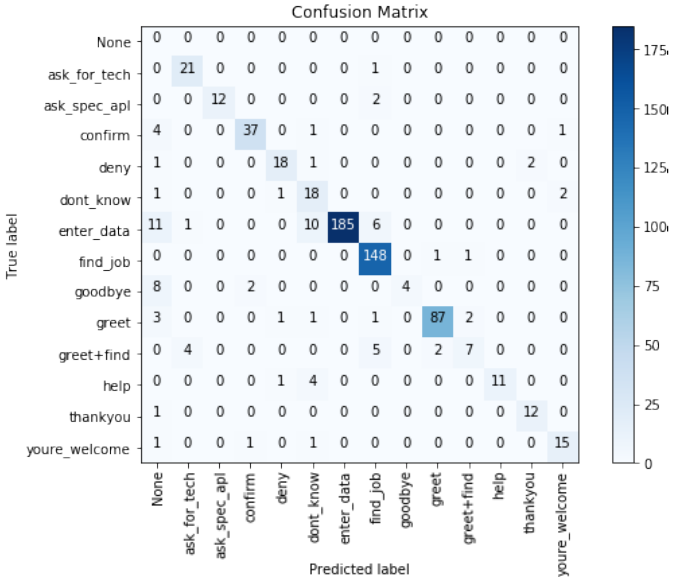

Abbildung 3: Konfusionsmatrix NLU-Komponente

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.

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.

Ein Nutzertest begann jeweils mit der Begrüßung des Chatbots und 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 anonym und unüberwacht 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 90,7% der Konversationen im Nutzertest als erfolgreich angesehen werden.

VII. 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. [2] 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.

QUELLENVERZEICHNIS

[1] M. McTear, Z. Callejas, und D. Griol, *The Conversational Interface: Talking to Smart Devices*, 1st ed. 2016. New York, NY: Springer, 2016.

[2] 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.

[3] A. Coucke *u. a.*, „Snips Voice Platform: an embedded Spoken Language Understanding system for private-by-design voice interfaces“, *ArXiv180510190 Cs*, Mai 2018.

[4] E. D. Liddy, „Natural Language Processing“, in *Encyclopedia of Library and Information Science*, 2nd Ed., NY: Marcel Decker, Inc, 2001.

[5] L. Deng und Y. Liu, *Deep Learning in Natural Language Processing*, 1st Aufl. Springer Publishing Company, Incorporated, 2018.

[6] 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].

[7] T. Young, D. Hazarika, S. Poria, und E. Cambria, „Recent Trends in Deep Learning Based Natural Language Processing“, *ArXiv170802709 Cs*, Aug. 2017.

[8] 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.

[9] „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].

[10] D. Jurafsky und J. H. Martin, *Speech and Language Processing*, 3rd edition draft. Upper Saddle River, NJ, 2017.

[11] „spaCy 101: Everything you need to know · spaCy Usage Documentation“. [Online]. Verfügbar unter: https://spacy.io/usage/spacy-101. [Zugegriffen: 20-Juni-2018].

[12] „Linguistic Features · spaCy Usage Documentation“. [Online]. Verfügbar unter: https://spacy.io/usage/linguistic-features. [Zugegriffen: 20-Juni-2018].

[13] F. Pedregosa *u. a.*, „Scikit-learn: Machine Learning in Python“, *J. Mach. Learn. Res.*, Bd. 12, S. 2825–2830, 2011.

[14] T. Bocklisch, J. Faulkner, N. Pawlowski, und A. Nichol, „Rasa: Open Source Language Understanding and Dialogue Management“, *ArXiv171205181 Cs*, Dez. 2017.

[15] Martín Abadi *u. a.*, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015.

[16] 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].

[17] L. Wu, A. Fisch, S. Chopra, K. Adams, A. Bordes, und J. Weston, „StarSpace: Embed All The Things!“, *CoRR*, Bd. abs/1709.03856, 2017.

[18] *Fast Artificial Neural Network Library (FANN): libfann/fann*. Fast Artificial Neural Network Library (FANN), 2018.

[19] M. D. Scholefield, *A rigid, lightweight, dead-simple intent parser. Contribute to MatthewScholefield/padaos development by creating an account on GitHub*. 2018.

[20] „Padatious documentation“. [Online]. Verfügbar unter: https://padatious.readthedocs.io/en/latest/. [Zugegriffen: 25-Sep-2018].

[21] „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].

[22] „Mycroft Github Repository“, *GitHub*. [Online]. Verfügbar unter: https://github.com/MycroftAI. [Zugegriffen: 25-Sep-2018].

[23] E. Alpaydin, *Maschinelles Lernen*, 1. Aufl. München: Oldenbourg Wissenschaftsverlag, 2008.

[24] S. Raschka, *Python Machine Learning*, 1. Aufl. Packt Publishing, 2015.

[25] „Choosing a Rasa NLU Pipeline“. [Online]. Verfügbar unter: https://rasa.com/docs/nlu/choosing_pipeline. [Zugegriffen: 22-Sep-2018].

[26] „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].

[27] T. Hughes, „Entity extraction with the new lookup table feature in Rasa NLU“, *Medium*, 13-Sep-2018. .

[28] „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].

[29] S. Hochreiter und J. Schmidhuber, „Long Short-Term Memory“, *Neural Comput*, Bd. 9, Nr. 8, S. 1735–1780, Nov. 1997.

[30] J. D. Williams und G. Zweig, „End-to-end LSTM-based dialog control optimized with supervised and reinforcement learning“, *ArXiv160601269 Cs*, Juni 2016.