

Chatbot

DIPLOMARBEIT

verfasst im Rahmen der

Reife- und Diplomprüfung

an der

**Höhere Lehranstalt für Informationstechnologie,
Ausbildungsschwerpunkt Medientechnik**

Eingereicht von:

Felix Dumfarth
Lukas Starka

Betreuer:

Thomas Stütz

Projektpartner:

Leonding, April 2022

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Die Arbeit wurde bisher in gleicher oder ähnlicher Weise keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Die vorliegende Diplomarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

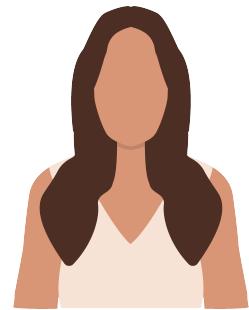
Leonding, April 2022

Felix Dumfarth & Lukas Starka

Zur Verbesserung der Lesbarkeit wurde in diesem Dokument auf eine geschlechtsneutrale Ausdrucksweise verzichtet. Alle verwendeten Formulierungen richten sich jedoch an alle Geschlechter.

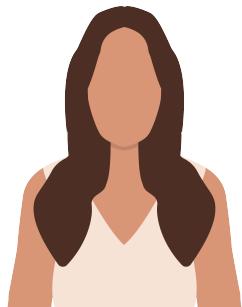
Abstract

This diploma thesis deals with the topic of a chatbot that was created with the help of Rasa and can be found on the HTL Leonding website. This chatbot should be able to provide interested people with answers about the school quickly and easily, saving visitors from having to search around on the website and calling and asking the secretariat.



Zusammenfassung

Die vorliegende Diplomarbeit beschäftigt sich mit dem Thema eines Chatbots, der mithilfe von Rasa erstellt worden ist und auf der Webseite der HTL Leonding zu finden sein soll. Dieser Chatbot soll interessierten Person einfach und schnell Antworten über die Schule liefern können und somit den Besuchern ein herumsuchen auf der Webseite sparen sowie anrufe und nachfragen beim Sekretariat.



Contents

1 Einleitung	1
1.1 Ausgangslage	1
1.2 Istzustand	1
1.3 Problemstellung	1
1.4 Aufgabenstellung	1
2 Technischer Hintergrund	2
2.1 Maschinelles Lernen	2
2.2 Deep Learning	2
2.3 Neuronale Netze	2
2.4 Text Analysis	2
2.5 Natural Language Processing (NLP)	3
3 Toolstack	15
3.1 Programmiersprachen	15
3.2 Technologien	16
3.3 Werkzeuge	16
4 Chatbots am Beispiel von Rasa	18
4.1 Allgemeines	18
4.2 Pipeline	20
4.3 Welche Rolle spielen neuronale Netze in Rasa	31
4.4 Komponenten	32
4.5 Initialisieren	39
4.6 Trainieren	40
4.7 Interagieren	40
5 Implementierung	43
5.1 Systemarchitektur	43

5.2	Backend	43
5.3	Chat Widget	45
5.4	Dashboard	48
5.5	Einbindung in Wordpress	51
5.6	GH Actions	51
5.7	Schwierigkeiten	51
6	Evaluation	52
	Bibliography	VI
	List of Figures	XI
	List of Tables	XII
	Quellcodeverzeichnis	XIII
	Anhang	XIV

1 Einleitung

1.1 Ausgangslage

Zahlreiche Organisationen bauen heutzutage auf eine starke Webpräsenz und wickeln darauf komplexe Produkt- oder Leistungsfunktionalitäten ab. Eine wichtige Rolle spielt dabei nicht nur der Kundenservice sondern auch die Kommunikation, denn die Nutzer erwarten eine möglichst zeitnahe Beratung. Die Bereitstellung von solchen Services ist derzeit mit hohen Kosten verbunden.

1.2 Istanzustand

Es gibt viele Personen die an der HTL Leonding interessiert sind und sich gerne schnell über die Schule informieren wollen.

1.3 Problemstellung

Auf der HTL Leonding Webseite muss jeder Besucher seine Informationen auf sehr vielen verschiedenen Unterseiten suchen oder beim Sekretariat anrufen.

1.4 Aufgabenstellung

Es soll ein Chatbot für die HTL Leonding Webseite erstellt werden, der Besuchern der Webseite einfach und schnell schulspezifische Fragen beantworten soll.

2 Technischer Hintergrund

2.1 Maschinelles Lernen

2.2 Deep Learning

2.3 Neuronale Netze

2.3.1 Convolutional Neural Networks

2.3.2 Recurrent Neural Networks

2.4 Text Analysis

2.4.1 Text analysis vs. Text Mining vs. Text Analytics

Meistens werden die Begriffe Text Analysis und Text Mining im selben Zusammenhang verwendet. Dabei bedeuten sie eigentlich dasselbe, nämlich, dass man den Sinn einer Nachricht extrahiert. Deswegen wird in den folgenden Kapiteln nur von Text Analysis gesprochen.

Es gibt jedoch einen Unterschied zwischen Text Analysis und Text Analytics. Grundsätzlich kann man dabei sagen, dass Text Analysis qualitative Ergebnisse liefert, wohingegen bei Text Analytics die Quantität mehr im Vordergrund steht.[1, 2]

Bei Text Analysis werden also wichtige Informationen aus der Nachricht herausgelesen. Oder anders formuliert geht es darum, dass trotz der Unterschiede der menschlichen Sprache trotzdem die Kernaussage herausgefunden werden kann, mit der dann gearbeitet wird. Es können also dann Informationen herausgefiltert werden, wie beispielsweise, ob etwas positiv oder negativ ist oder was das Hauptthema des Textes ist.[1, 2]

Auf der anderen Seite wird bei Text Analytics werden verschiedene Muster aus einer großen Menge an Nachrichten herausgefiltert, welche dann in Graphen, Tabellen oder Berichten gezeigt werden können. Bei Text Analytics geht es also darum Muster

und Entwicklungen von numerischen Ergebnissen herauszufinden, bei denen dann Informationen herausgefiltert werden können, wie beispielsweise, die Prozentzahl der positiven Bewertungen.[1, 2]

2.4.2 Warum Text Analysis?

Im Gegensatz zum manuellen Aufbereiten von Texten sorgt Machine Learning für eine schnelle Bearbeitung, die außerdem auch kostengünstiger ist, weil viele Stellen wegfallen, die benötigt werden würden, um die Texte selber aufzubereiten. Durch die Verwendung von Text Analysis spart man sich viele Arbeitskräfte, die bei anderen wichtigen Aufgaben innerhalb eines Unternehmens eingesetzt werden können. Außerdem können dadurch Texte rund um die Uhr und zur Echtzeit bearbeitet werden. Durch Algorithmen werden außerdem Fehler reduziert, die bei manuellem Bearbeiten leicht auftreten können und Daten können genauer aufbereitet werden.[1]

2.4.3 Machine Learning mit Text Analysis

Grob kann man behaupten, dass ein Text Analysis Tool aus drei verschiedenen Schritten besteht.

1. Zunächst muss man sich überlegen, welche Daten gesammelt werden sollen, um damit sein Modell zu trainieren und zu testen. Man unterscheidet hierbei zwischen "Internal Data" und "External Data". Unter External Data werden Quellen, wie Zeitungen oder Foren bezeichnet und zu der Internal Data zählen sämtliche Daten, die eine Firma jeden Tag generiert, wie E-Mails, Reports, Chats oder Umfragen.
2. Danach müssen die Daten vorbereitet werden, damit das Programm diese versteht. Dieser Schritt wird meistens als "Data Preprocessing" bezeichnet.
3. Zum Schluss wird dann ein Machine Learning Algorithmus hinzugefügt, welcher sich um die Analyse kümmert. Diesen kann man entweder komplett selber implementieren oder man nimmt sich Libraries zur Hilfe.[2]

2.5 Natural Language Processing (NLP)

Natural Language Processing, oft auch als Akronym **NLP** abgekürzt, sorgt dafür, dass Computer Text auf dieselbe Art und Weise verstehen, wie wir es als Menschen tun.

NLP vereint dabei die Modellierung der menschlichen Sprache mit statistischen Machine Learning und Deep Learning Modellen. Dies ermöglicht der Maschine im Endeffekt, dass diese menschliche Sprache in Form von Text- oder Sprachdaten zu verarbeiten und sozusagen die komplette Bedeutung und Absicht der Benutzerin oder des Benutzers zu verstehen.[3]

2.5.1 Corpus

Unter dem Corpus werden die Daten bezeichnet, die verwendet werden, um das NLP Modell zu trainieren, damit dieses menschliche Sprache versteht und damit arbeiten kann. Der Textkorpus ist also eine Menge von strukturierten Texten, die für den Computer lesbar sind.[4]

2.5.2 Tokenization

Bei der Tokenization repräsentiert jeder Token eine sinnvolle Einheit. Darunter versteht man Wörter, Zeichen oder Sonderzeichen, die alle einen eigenen Token darstellen, lediglich Leerzeichen in einem Satz stellen keine eigenen Token dar.[2, 5]

Die verschiedenen Token, die dabei entstehen, können dabei wie in Listing 1 aussehen:

Listing 1: Beispiel für die Tokenization

```

1 Let us go to the park.
2
3 0: Let
4 1: us
5 2: go
6 3: to
7 4: the
8 5: park
9 6: .

```

Die Tokenization ist dabei besonders für verschiedene Sprachen von Vorteil, da es in diesen immer andere Regeln gibt, wie die Wörter aufgeteilt werden können. In dem Beispiel 1 wirkt es womöglich so, als könnte man mit einer `split()` Methode das selbe Resultat erzielen, aber es gibt auch Besonderheiten einer Sprache, die vom Tokenizer bedacht werden müssen, wie folgendes Beispiel aufzeigt.[2, 5]

Listing 2: Ausnahme für bestimmte Tokens

```

1 "Let's go to the U.K.!"
2
3 0: "
4 1: Let
5 2: 's
6 3: go
7 4: to
8 5: the

```

```

9  6: U.K.
10 7: !
11 8: "

```

2.5.3 Part-of-speech-Tagging (POS-Tagging)

Beim Part-of-speech-Tagging geht es um das Taggen von Wörtern und deren zugehörigen Teil der Sprache. Als Teil einer Sprache, also dem Part-of-speech, werden grundsätzlich Wörter bezeichnet, die ähnliche grammatischen Eigenschaften oder Nutzungen besitzen. Im deutschen sind hierbei also die verschiedenen Wortarten gemeint.

Beim Tagging werden entweder Statistiken angewendet, wie beispielsweise, dass es sehr wahrscheinlich ist, dass nach einem Artikel ein Nomen folgt. Außerdem gibt es sogenannte Rule-Based POS-Taggers, deren Aufgabe es ist, vordefinierte Regeln zu verwenden, um das Tagging zu vollziehen.[2, 5]

Ein Beispiel für POS-Tagging könnte also wie in Listing 3 aussehen:

Listing 3: Beispiel für POS-Tagging

```

1 I am going to the U.K.
2
3 I: Pronoun
4 am: Verb
5 going: Verb
6 to: Part
7 the: Article
8 U.K.: Noun

```

2.5.4 Named-entity recognition (NER)

Named-entity recognition ist eine der wichtigsten Säulen von Natural Language Processing. Entities sind strukturierte Stücke von Informationen, die sich innerhalb der Nachricht einer Benutzerin oder eines Benutzers befinden. Dabei werden Entities aus Texten erkannt und anschließend mit einem Tag der zugehörigen Kategorie versehen.[6]

Beispiele für Entities wären also:

Listing 4: Beispiele für Entities

```

1 PERSON      Personen (inklusive fiktionalen Personen)
2 NORG      Nationalitäten oder religiöse oder politische Gruppen
3 FACILITY   Gebäude, Flughäfen, Brücken, Straßen
4 LOC        Orte
5 PRODUCT     Objekte, Fahrzeuge, Essen
6 LANGUAGE    Sprachen

```

2.5.5 Stemming

Beim Preprocessing von Texten sind außerdem das Stemming und die Lemmatization Schritte, die häufig durchgeführt werden.

Beim Stemming wird der Wortstamm eines Tokens von den Präfixen und Suffixen gelöst. Unter einem Präfix versteht man eine Vorsilbe, die dem Wortstamm vorangestellt wird und bei dem Suffix, handelt es sich im Gegenzug dazu um Worterweiterungen, die dem Stamm folgen. Hierbei gibt es wieder die Möglichkeit selber einen Stemmer zu implementieren oder bereits vorgefertigte Stemmer zu verwenden.[1, 2]

Ein Beispiel für Stemming von deutschen Wörtern wäre also in Listing 5 zu sehen.

Listing 5: Stemming von deutschen Wörtern

```

1
2 verarbeiten ---> ver + arbeit + en
3 ver                  prefix
4 arbeit                base word
5 en                   suffix

```

2.5.6 Lemmatization

Unter der Lemmatization oder auch Lemmatisierung wird der Prozess verstanden, bei dem alle verschiedenen Arten von einem Wort zu ihrem Wortstamm umgeformt werden.

Die Lemmatisierung ist sozusagen eine Weiterentwicklung des Stemming und versucht im Gegensatz zum Stemming den Kontext zu den Wörtern zu bringen. Je nachdem in welchem Kontext die Verben gebraucht werden können sie sich dabei von ihrer Wortart unterscheiden und so eine andere Bedeutung haben. Generell lässt sich sagen, dass Stemmer meist einfacher zu implementieren sind und für schnellere Laufzeiten sorgen, allerdings wird von vielen auch die Lemmatization dem Stemming vorgezogen, aufgrund der Vorteile die das Einbeziehen des Kontextes mit sich bringt.[2, 1, 7]

Beispiele für die Lemmatisierung sind:

Listing 6: Lemmatisierung von deutschen Wörtern

```

1 Gut ---> gut
2 besser ---> gut
3 las ---> lesen
4 gelesen ---> lesen
5 lesend ---> lesen
6 mag ---> m gen

```

2.5.7 Parsing

Parsing könnte man so beschreiben, dass es eine Art ist um einen Satz auseinanderzuteilen, um die Struktur des Satzes besser zu verstehen. Beim Parsing wird also die syntaktische Struktur eines Textes bestimmt. Um dies zu erreichen, nimmt der Parsing Algorithmus die Eigenheiten der Grammatik, von der jeweiligen Sprache, in der der Text geschrieben ist, her.[1]

Es gibt zwei verschiedene Arten von Parsing. Diese sind das sogenannte "Dependency Parsing" und das sogenannte "Constituency Parsing".

Dependency Parsing

Beim Dependency Parsing wird die grammatische Struktur eines Satzes bestimmt. Außerdem kann man mithilfe des Dependency Parsing herausfinden, welche Wörter zusammenhängen und welche Beziehung diese zueinander aufweisen. Dies bedeutet beispielsweise, dass man bei den Wörtern "blauer Ball", "nette Frau" oder ähnlichem erkennen kann, dass sich die Wörter "blau" oder "nett" auf das nachfolgende Wort bezieht. Ein anderes Beispiel könnten die Wörter "black" und "car" sein.

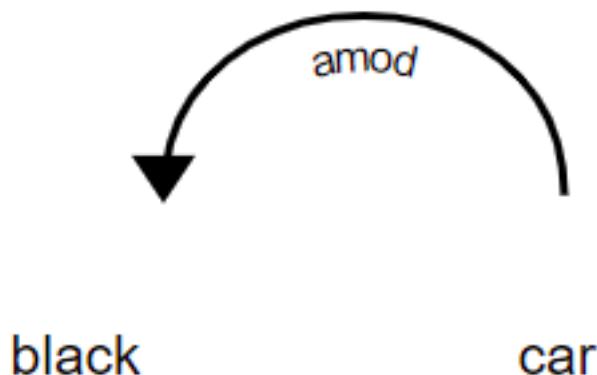


Figure 1: Beispiel für die Beziehung zweier Wörter beim Dependency Parsing [8]

Wie man aus der Grafik 1 entnehmen kann, bezieht sich die Farbe Schwarz in diesem Fall auf das Auto. Im Englischen wird das Wort "car" deswegen auch als "Head" bezeichnet und "black" ist ein "child", also ein Wort, dass abhängig von diesem "Head" ist. Die Beziehung, die zwischen diesen Wörtern vorliegt, wird mit **amod** bezeichnet. Dies ist eine Abkürzung für "Adjectival Modifier", also ein Adjektiv, welches ein Nomen modifiziert und zu der Bedeutung des Nomens beiträgt. In diesem Fall gibt dieses eine genaue Beschreibung der Farbe des Autos.[8]

Diese Beziehungen kann man auch in einem Tree darstellen.

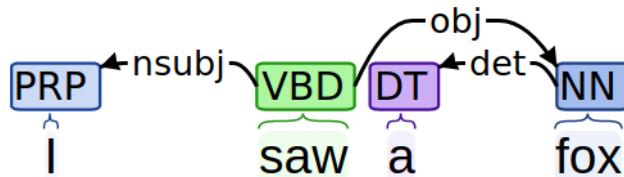


Figure 2: Darstellung des Dependency Parsing [9]

Constituency Parsing

Beim Constituency Parsing wird ein Satz in mehrere Teilphrasen zerteilt, die auch "Constituents" genannt werden. Im Gegensatz zum Dependency Parsing, bei dem man Beziehungen zwischen den Wörtern innerhalb eines Satzes bekommt, kann man hierbei also die Sätze gruppieren. Dies kann helfen, um die Struktur von Sätzen aufzuzeigen. Der Nachteil hierbei ist, dass man nun keinen Context in den Sätzen mehr hat, dafür kann man aber beispielsweise gut überprüfen, ob ein Satz grammatisch korrekt oder inkorrekt ist.[2]

Die berühmtesten Tags um Constituents zu bezeichnen sind:

- VP für Verbalphrasen (Verb Phrases)
- NP für Nominalphrasen (Noun Phrases)

Der Satz "I saw a fox" wird hierbei als Tree des Constituency Parsings dargestellt.

In dem Beispiel 3 kann man sehen, dass der Satz in zwei Teile aufgeteilt wurde. Der Textbaustein "I" wurde als Substantiv (NP) erkannt und "saw a fox" ist hierbei die Phrase des Verbs (VP). Darunter sieht man die Regeln, die dazu führten, dass hierbei erkannt wurde, dass es sich um eine Nominalphrase und eine Verbalphrase handelt. Diese Regeln werden durch die Grammatik der Sprache erstellt und so sagt beispielsweise die Regel " $S \rightarrow NP VP$ " aus, dass ein ganzer Satz (S) durch eine Nominalphrase (NP) und eine Verbalphrase (VP) entsteht.[9]

2.5.8 Stopwords

Unter Stopwords oder Stoppwörtern werden Wörter verstanden, die häufig in einem Satz vorkommen, aber nicht wirklich zum Informationsgehalt des Textes beitragen.

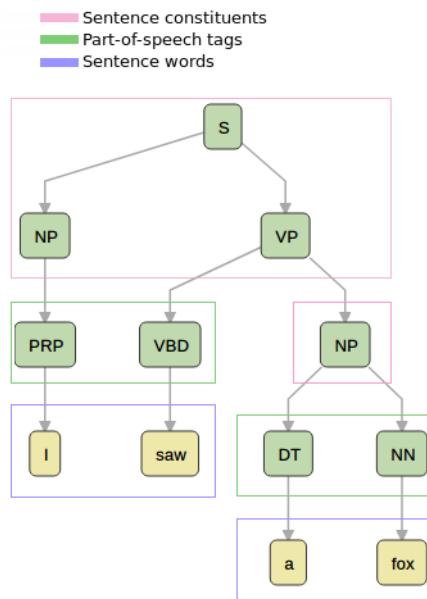


Figure 3: Darstellung des Constituency Parsing [9]

Deswegen werden diese oft nicht beachtet, da sie keine Relevanz für die Bedeutung des Satzes haben.

Bei den Stoppwörtern handelt es sich meistens um die am häufigsten vorkommenden Wörter einer Sprache. Deutsche Stoppwörter sind zum Beispiel:[10]

- was
- aber
- auch
- der
- die
- das
- du
- wir
- ihr
- können
- müssen
- dürfen

2.5.9 Vectorization

Durch Machine Learning können Systeme basierend auf vorherigen Eingaben und Mustern Dinge voraussagen. Diese Systeme benötigen allerdings Trainingsdaten, um ihre Präzision dabei zu steigern. Wenn man nun also ein System trainieren möchte benötigt man eine Darstellungsform, die auch von Maschinen verstanden werden kann. Hierbei kommen nun Vektoren zum Einsatz, weil das System dadurch die wichtigsten Informationen extrahieren und daraus lernen kann.

2.5.10 Skalarprodukt

Beim Natural Language Processing benötigt man sehr oft das sogenannte Skalarprodukt. Dieses kann auch als inneres Produkt oder Punktprodukt bezeichnet werden. Das Skalarprodukt ordnet zwei Vektoren eine Zahl zu. Ein Skalarprodukt berechnet man wie in 4 zu sehen ist.

$$\vec{a} \bullet \vec{b} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \bullet \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

Figure 4: Skalarprodukt von zwei Vektoren [11]

2.5.11 Bag of Words

Eine der einfachsten Techniken um einen Text nummerisch zu repräsentieren, ist mithilfe der sogenannten Bag of Words Vectorization. Dabei wird zunächst jedes Wort, welches im Corpus vorkommt als sogenanntes Vokabular gespeichert. Danach wird in jedem Satz die Anzahl dargestellt, wie oft ein bestimmtes Wort vorkommt.[1]

Listing 7: Bag of Words

```

1 Dies ist ein Bag of Words Beispiel.
2
3 Das      0
4 Dies     1
5 ist      1
6 ein      1
7 eine    0
8 Bag      1
9 of       1
10 Words   1
11 Beispiel 1
12 Example 0

```

Um die Überlappungen von Bag of Words zu berechnen, nutzt man das Skalarprodukt der beiden Vektoren von Sätzen, die man vergleichen will. Bei diesem Skalarprodukt kommt dann immer ein Wert (Skalar) als Ergebnis heraus.

2.5.12 Bag of n-grams

Ein Bag of n-grams ist ziemlich ähnlich zu einem Bag of Words. Allerdings wird bei einem Bag of n-grams ein Text als eine unsortierte Collection von n-grams dargestellt. In dem Bag of n-grams wird dann die Anzahl, wie oft ein n-gram vorkommt gespeichert. Ein n-gram ist eine Sammlung von n aufeinander folgender Wörter. Wenn das $n = 2$ ist, spricht man von einem bigram, wenn $n = 3$ ist von einem trigram und so weiter. Somit würde unser Beispiel von 7 als bigram folgendermaßen aussehen:

Listing 8: Bag of n-grams

```

1 Dies ist ein Bag of Words Beispiel.
2
3 1: <start> Dies
4 2: Dies ist
5 3: ist ein
6 4: ein Bag
7 5: Bag of
8 6: of Words
9 7: Words Beispiel
10 8: Beispiel <end>

```

2.5.13 Term Frequency times Inverse Document Frequency (TF-IDF)

TF-IDF steht für Term Frequency times Inverse Document Frequency und dies ist die meist verwendete Technik um die Vorkommen von Wörtern innerhalb eines Satzes zu zählen. Mithilfe von TF-IDF kann man ermitteln, wie wichtig ein Wort in einem Satz ist. Bei TF-IDF werden zwei verschiedene Metriken miteinander multipliziert.[12]

1. **Term Frequency TF:** Die Term Frequency kann man sich ausrechnen, indem man die Anzahl eines Terms, der in einem Dokument vorkommt, durch die Anzahl der Terme insgesamt rechnet.
2. **Inverse Document Frequency IDF:** Diese gibt an wie selten ein Wort im ganzen Dokument ist. Dabei wird die Anzahl der Dokumente (N) durch die Anzahl des Terms (n) gerechnet und davon wird der Logarithmus berechnet.

$$TF(t, d) = \frac{\text{number of times } t \text{ appears in } d}{\text{total number of terms in } d}$$

$$IDF(t) = \log \frac{N}{1 + df}$$

$$TF - IDF(t, d) = TF(t, d) * IDF(t)$$

Figure 5: Berechnung von TF-IDF [13]

Wenn man diese zwei Werte nun also miteinander multipliziert, erhält man so einen Wert, der indiziert, ob ein Wort relevant für die Bedeutung in dem Dokument ist. Je höher der Wert ist, desto relevanter ist das Wort sozusagen.

Listing 9: TF-IDF Beispiel

```

1 Heute ist ein warmer Tag
2
3 Term      Count
4 Heute     2
5 ist       1
6 ein       1
7 warmer   2
8 Tag       5

```

Bei dem Beispiel 9 wird gezeigt, dass die Wörter, die häufig in einem deutschen Satz vorkommen, wie "ist" oder "ein", einen niedrigen Score erhalten und das Wort "Tag" einen hohen Score aufweist, was bedeutet, dass dieses Wort vermutlich wichtig für die Bedeutung des Satzes ist.

Bei der Text Analysis mit Machine Learning werden solche TF-IDF Algorithmen verwendet, um Daten in Kategorien einzurichten und wichtige Schlüsselwörter zu extrahieren. Auch Suchmaschinen wie Google verwenden diese Technologien, um aus dem Suchbegriff des Users die zentrale Aussage herauszufinden und die Suchergebnisse nach ihrer Relevanz zu sortieren.[12]

2.5.14 Word2Vec

Word2Vec ist das berühmteste Modell für Word Embedding. Unter Word Embedding wird verstanden, dass Texte in denen Wörter dieselbe Bedeutung haben, zusammen repräsentiert werden. Es werden also sozusagen alle Wörter in einem Koordinatensystem dargestellt, wobei Wörter, die zusammen gehören, nah beieinander sind. Word2Vec nimmt dabei einen großen Corpus an Text und produziert daraus einen Vektor mit den einzigartigen Wörtern, die ähnlich zueinander sind. Es ist sehr berühmt dafür, dass demonstriert werden kann, wie Wörter miteinander verbunden sind.

Rasa hat als kleines Nebenprojekt mit "WhatLies" eine Library geschaffen, die versuchen soll, dass man Word Embeddings besser versteht. Hierbei kann man Grafiken erstellen, die die Wörter in einem Koordinatensystem repräsentieren.[14]

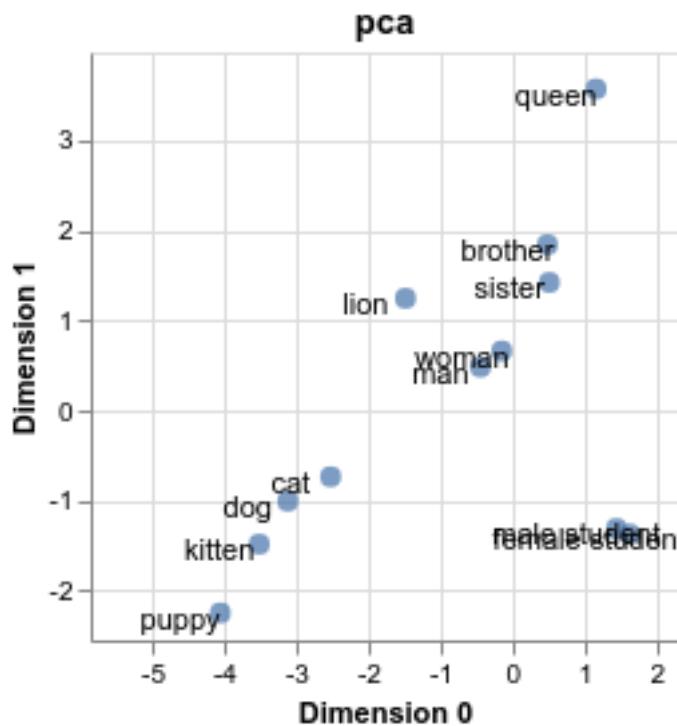


Figure 6: Whatlies Illustration [14]

Aus der Abbildung 6 kann man nun erkennen, dass die Wörter "brother" und "sister" oder "woman" und "man" nahe beieinander im Raum dargestellt werden. Außerdem kann man daraus lesen, dass sich "man" zu "brother" ähnlich wie "woman" zu "sister" oder "puppy" zu "dog" sich ähnlich wie "kitten" zu "cat" verhält. Dies findet man dadurch heraus, dass man ein ähnliches Ergebnis bekommt, wenn man die zwei Unterschiede der Vektoren miteinander vergleicht. Man kann sich außerdem vorstellen, dass man zum Beispiel bei der Rechnung "King - Man + Woman = ?" als Ergebnis die Position des Wortes "Queen" erhält.

2.5.15 NLP und NLU Tools

Ein paar Beispiele für berühmte Tools, die man nützen kann für Natural Language Understanding (NLU) und Natural Language Processing (NLP) sind:

- NLTK: Das Natural Language Toolkit (kurz NLTK) ist wahrscheinlich die berühmteste Python Library für NLP. NLTK besitzt über 50 Ressourcen für das Bearbeiten

von Corpora. Außerdem gibt es viele eingebaute Libraries für das Analysieren eines Textes, wie die Klassifikation, Tokenization, Stemming, Tagging, Parsing und Semantic Reasoning.

- SpaCy: SpaCy ist ebenfalls eine sehr berühmte Bibliothek für Python. SpaCy beschreibt sich selbst als "Industrial-Strength Natural Language Processing" und unterstützt über 64 Sprachen. SpaCy hat nur einen POS-Tagging-Algorithmus und pro Sprache nur einen Named-Entity-Recognizer, was den Vorteil hat, dass es nicht voller unnötiger Features ist und sich nur auf ein paar wichtige Funktionalitäten konzentriert.
- TensorFlow: TensorFlow wird von Google betrieben und ist bei weitem die meist verwendete Library für Deep Learning. Außerdem ist die Plattform Open-Source.
- Keras: Keras ist eine Deep Learning Plattform, die in Python geschrieben ist. Sie wurde entworfen um schnelle Iterationen und schnelles Ausprobieren mit deep neural networks zu ermöglichen. Außerdem bietet Keras ein Interface zu deep neural networks an, welche dann beispielsweise auf TensorFlow, Theano oder anderen Backends laufen können.
- CoreNLP: Das CoreNLP Projekt von Stanford ist ein NLP toolkit, welches zwar in Java geschrieben wurde aber auch in anderen Sprachen verwendet werden kann, die auf der JVM Plattform sind.
- Gensim: Gensim ist eine Open-Source Library für Python, mit der man semantische NLP Modelle trainieren kann. Außerdem ist Gensim Plattform unabhängig und läuft auf jedem Betriebssystem.
- TextBlob: TextBlob ist eine Python Library für das Bearbeiten von Texten.
- FastText: FastText ist eine Open-Source Library, die es Benutzern erlaubt text representations und text classifier zu verwenden.

2.5.16 NLP in Rasa

In Rasa kann man sich die verschiedenen Schritte vom Natural Language Processing ansehen, indem man sich die Pipeline ansieht. Diese befindet sich in der **config.yml** Datei und dort sind alle Komponenten aufgelistet, die Rasa verwendet und man kann diese nach Belieben auch selbst bearbeiten.

3 Toolstack

3.1 Programmiersprachen

3.1.1 Java



Figure 7: Java Logo[15]

Für das Backend haben wir uns für Java entschieden. Java ist laut dem TIOBE-Index[16] eine der populärsten Programmiersprachen. Java ist eine objektorientierte Programmiersprache und besitzt dadurch Klassen und Vererbung[15].

3.1.2 Python

Python ist eine Programmiersprache die mehrere Arten der Programmierung unterstützt wie zum Beispiel die objektorientierte, die aspektorientierte und die funktionale Programmierung. Python wird oft für Machine Learning verwendet.

3.1.3 Typescript

Typescript ist eine Programmiersprache die eine kompakte und einfache Syntax zur Programmierung von Webseiten und Anwendungen bietet. Typescript baut auf Javascript auf und hilft zum Beispiel beim frühzeitigen Erkennen von Fehlern.

3.2 Technologien

3.2.1 REST Service

REST steht für Representational State Transfer. REST Anfragen sind CRUD – Operationen (Create, Read, Update, Delete). Diese sind GET, POST, PUT und DELETE Requests. Außerdem gibt es noch OPTIONS, PATCH, HEAD, TRACE und CONNECT diese werden aber in der vorliegenden Arbeit nicht benutzt.

- * Die GET Methode ist zum Abfragen da, es soll ein Request geschickt werden und nur Daten zurückgegeben werden, es soll jedoch auf dem Server wohin die Anfrage geschickt worden ist nichts geschehen außer Daten lesen.
- * Bei POST sollen Daten hinzugefügt werden, im Standardfall ist in der Response der URI der neu gespeicherten Daten
- * Bei PUT sollen Daten hinzugefügt werden, falls diese schon existieren werden Sie upgedatet und falls nicht, wird ein neuer Eintrag erstellt.
- * Bei DELETE sollen Daten gelöscht werden.

3.2.2 Quarkus

Quarkus ist ein Framework für Java welches sich auf die REST-Service-API ausrichtet.

3.2.3 Rasa

3.2.4 Angular

Angular ist ein auf TypeScript basierendes Framework für Webseiten. Gedacht ist es für Single Page Applications.

3.3 Werkzeuge

3.3.1 IntelliJ IDEA

IntelliJ ist ein IDE welche eine Vielzahl von Programmiersprachen unterstützt und für diese Arbeit verwendet wird.

3.3.2 GitHub

GitHub ist ein Online-Repository welches die gemeinsame Programmierung von Programmierern und Entwicklern erleichtert.

3.3.3 Docker

Docker ist einer der bekanntesten Container-Manager.

4 Chatbots am Beispiel von Rasa

4.1 Allgemeines

4.1.1 Rasa Produkte

Der Rasa Stack wird in Rasa NLU und Rasa Core aufgeteilt. Diese sind so aufgebaut, dass sie unabhängig voneinander eingesetzt werden können. So besteht die Möglichkeit, nur einen Teil der Architektur auf Rasa aufzubauen und zusätzlich weitere Services mit einzubinden.

4.1.2 Rasa Core

Rasa Core ist verantwortlich für den Conversation Flow, Context-Handling, Bot-Responses und das Session Management. Dabei kann auf der Rasa NLU oder anderen Services aufgebaut werden, die die Intent Recognition und Entity Extraction übernehmen und die Ergebnisse dem Rasa Core zur Verfügung stellen.[17]

Rasa Core bezieht sich dabei auf die Hauptkomponente, die die Nachrichten erhält und darauf antwortet.[17]

Der Rasa Core hält für jede Session, also für jeden User, einen Tracker. Dieser enthält den momentanen Zustand der Konversationen, der jeweiligen User. Bekommt der Bot nun eine Nachricht, wird zuerst der Interpreter durchlaufen, welcher den Originaltext als Eingabe bekommt, und die Eingabe, den Intent und die extrahierten Entities zurückgibt. Zusammen mit dem aktuellen Zustand des Trackers entscheidet die Policy Komponente nun, welche Action, also Antwort des Bots, als nächstes ausgeführt werden soll. Diese Entscheidung wird nicht durch einfache Regeln getroffen, sondern genauso wie Intents oder Entities, auf der Grundlage von einem, mit Machine Learning, trainierten Model.[17, 18]

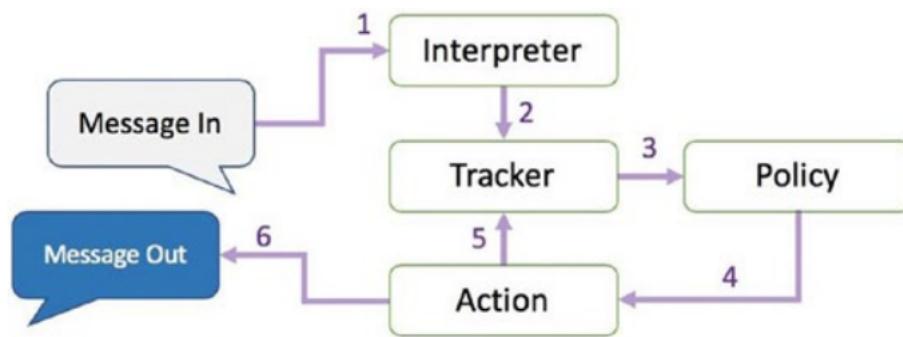


Figure 8: Rasa Core Aufbau [18]

4.1.3 Policies

Policies sind ein Teil von **Rasa Core** und der Assistent benutzt Policies um zu entscheiden, welche Action als nächstes ausgeführt werden soll. Es gibt machine-learning und rule-based policies.[19]

Hierbei kann man die Policies beispielsweise ändern. Dies macht man in der **config.yml** Datei.

Bei den Policies gibt es unterschiedliche Priorities, die dann zum Einsatz kommen, wenn mehrere Policies dieselbe Confidence vorhergesagt haben.[20]

TED Policy

Die TED Policy steht für Transformer Embedding Dialogue Policy und wird meistens standardmäßig verwendet.[21]

Bei jedem Dialog bekommt die TED Policy drei Informationen als Input. Die Nachricht des Users, die vorherige Action die vorhergesagt wurde und Slots und aktive Forms. Dann werden diese in den Dialogue Transformer Encoder gepackt und anschließend werden sogenannte Dense Layer verwendet. Danach wird die Ähnlichkeit zwischen den System Actions und dem Dialogue Embedding berechnet und zum Schluss werden noch CRF Algorithmen verwendet, um Entities zu erkennen.[21]

4.1.4 Rasa NLU

Rasa NLU hat grundsätzlich zwei Hauptaufgaben.

Zum einen wäre da die Intent Recognition und die Entity Recognition.[22, 23]

Die Intent Recognition, ist die Erkennung der Nutzer-Absichten. Dazu muss die NLU mit ausreichend Utterances, also Responses trainiert werden. Dabei gibt die NLU alle zugehörigen Intents geordnet nach dem Confidence Score zurück. Rasa verfügt demnach über ein Multi Intent Matching.[22, 23]

Außerdem gibt es noch die Entity Recognition, die dafür zuständig ist Entities, also wichtige Informationen, aus natürlicher Sprache zu extrahieren.[22]

Der Aufbau der NLU ist vollständig konfigurierbar mithilfe der sogenannten Pipeline.[24]

4.2 Pipeline

Rasa Open Source bietet bei der Initialisierung des Projekts eine Standard-NLU-Konfiguration.[25]

In Rasa Open Source werden eingehende Nachrichten von einer Reihe von Komponenten verarbeitet. Diese Komponenten werden nacheinander in einer sogenannten Processing Pipeline ausgeführt, die im config.yml File definiert ist. Wenn man eine NLU-Pipeline auswählt, kann man allerdings sein Model anpassen und an das Dataset verfeinern.[24]

In der Abbildung 9 werden die Komponenten und ihre Lifecycle abgebildet.

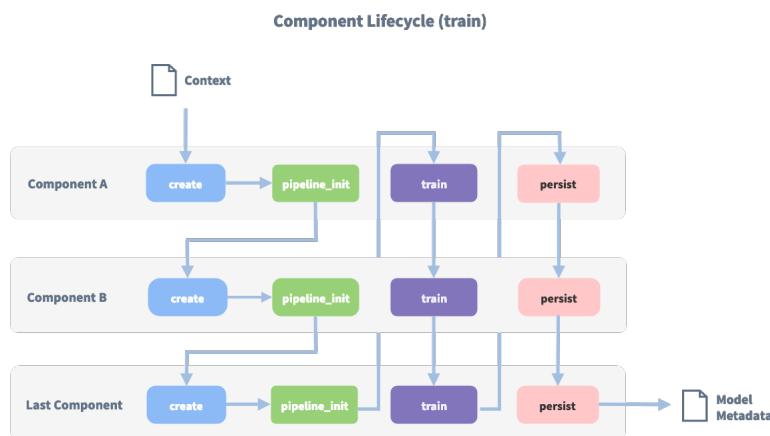


Figure 9: Component Lifecycle [26]

Bevor die erste Komponente mit der Create-Funktion erstellt wird, wird ein sogenannter Kontext erstellt (der nichts anderes als ein Python-dict ist). Dieser Kontext wird verwendet, um Informationen zwischen den Komponenten zu übergeben. Beispielsweise kann eine Komponente Merkmalsvektoren für die Trainingsdaten berechnen, diese im

Kontext speichern und eine andere Komponente kann diese Merkmalsvektoren aus dem Kontext abrufen und eine Intent Klassifikation durchführen.[26, 27]

Zunächst wird der Kontext mit allen Konfigurationswerten gefüllt, die Pfeile im Bild zeigen die Aufrufreihenfolge und visualisieren den Pfad des übergebenen Kontexts. Nachdem alle Komponenten trainiert und beibehalten wurden, wird das finale context dictionary verwendet, um die Metadaten des Models beizubehalten.[26, 27]

Dies ist in der Grafik 10 zu erkennen.

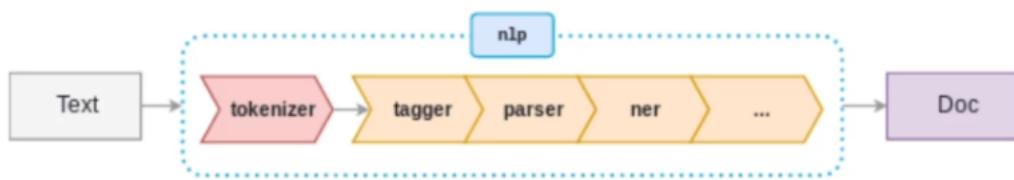


Figure 10: Component Lifecycle [28]

4.2.1 Arten von NLU Pipelines

Es gibt verschiedene bereits konfigurierte Pipelines.[24]

Grundsätzlich ist zwischen Pipelines zu unterscheiden, indem man sich informiert, ob sie Pretrained Word Vectors verwenden oder nicht.

Beispiele für Pipelines mit Pretrained Word Vectors

Der Vorteil von Pipelines, die Pretrained Word Vectors verwendet ist, dass diese bereits aus der jeweiligen Sprache word vectors besitzen. Somit weiß das Model beispielsweise, dass Äpfel und Birnen ähnlich sind ohne, dass dies in den Intents irgendwo spezifiziert werden muss.[29, 30]

Die Vorteile hierbei sind also, dass das Model weniger Trainingsdaten benötigt, um eine gute Performance zu besitzen. Außerdem geht der ganze Trainingsprozess in der Regel schneller und die sogenannten Iteration Times sind kürzer als bei Modellen ohne Pretrained Word Vectors.[29, 23, 30]

Listing 10: SpaCy Sklearn Pipeline

```

1 language: "de"
2
3 pipeline: "spacy_sklearn"

```

Die SpaCy Pipeline verwendet Pretrained Word Vectors von GloVe oder fastText.[31, 32, 30]

Es gibt außerdem auch noch Pipelines von MITIE. Diese verwendet MITIE als Source für die word vectors. Ein Vorteil von MITIE ist, dass man hier auch seine eigenen word vectors trainieren kann, indem man einen Corpus von Wikipedia oder ähnlichen Seiten verwendet. Allerdings wird MITIE meistens nicht empfohlen und es könnte auch sein, dass MITIE demnächst deprecated sein wird. Die MITIE Pipeline kann man wie in Listing 11 definieren.[33, 34]

Listing 11: MITIE Sklearn Pipeline

```
1 language: "de"
2
3 pipeline: "mitie_sklearn"
```

Beispiele für Pipelines ohne Pretrained Word Vectors

Der Vorteil von Pipelines ohne Pretrained Word Vectors ist, dass diese speziell auf den Fachbereich angepasst sind, für den man den Chatbot entwickelt.[23, 35]

Als Beispiel kann man die Wörter "balance" und "symmetry" aus dem Englischen sehen. Diese Wörter sind eng miteinander verwandt. Allerdings kann im Kontext von Banken das Wort "balance" auch mit "cash" verwandt sein. Bei einem vortrainierten Model würden diese Wortvektoren nicht nah aneinander liegen, aber wenn man einen Chatbot hat der nur Intents besitzt, die mit Banken und Rechnungswesen zu tun haben, werden diese zwei Wörter "balance" und "cash" ohne Pretrained Word Vectors als ähnlich erkannt werden.[35]

Außerdem benutzen diese Pipelines kein sprach-spezifisches Model und somit kann man sie in allen Sprachen verwenden, die tokenisiert werden kann. Eine solche Pipeline kann man wie in Listing 12 definieren.[35]

Listing 12: Tensorflow Embedding Pipeline

```
1 language: "de"
2
3 pipeline: "tensorflow_embedding"
```

Der Bag-of-word-vectors Ansatz ist zwar sehr gut aber leider auch nicht perfekt. Ein Problem davon ist, dass er oftmals keine fachspezifischen Begriffe kennt und außerdem können Typos nicht als Wortvektoren gelernt werden. Außerdem ist das Problem bei Pretrained Word Vectors, dass zehntausende Vektoren gespeichert werden, die vermutlich nie verwendet werden.[35, 36]

Mit dem Tensorflow-Embedding macht man im Grunde genommen genau das Gegenteil. Diese Pipeline verwendet keine vortrainierten Vektoren und sollte mit jeder Sprache verwendet werden können. Diese Pipeline lernt Embeddings für die Intents und für die Wörter und die Embeddings werden verwendet um die Ähnlichkeit zwischen dem Input und den Intents zu ermitteln.[35, 36]

Eine weitere Pipeline lautet wie folgt:

Listing 13: Supervised Embedding Pipeline

```

1 language: "de"
2
3 pipeline: "supervised_embedding"

```

Supervised Embeddings vs. Pretrained Embeddings

In der Grafik 5 sieht man ein Flussdiagramm, welches beschreibt, ob man Supervised Embeddings oder Pretrained Embeddings verwenden soll.

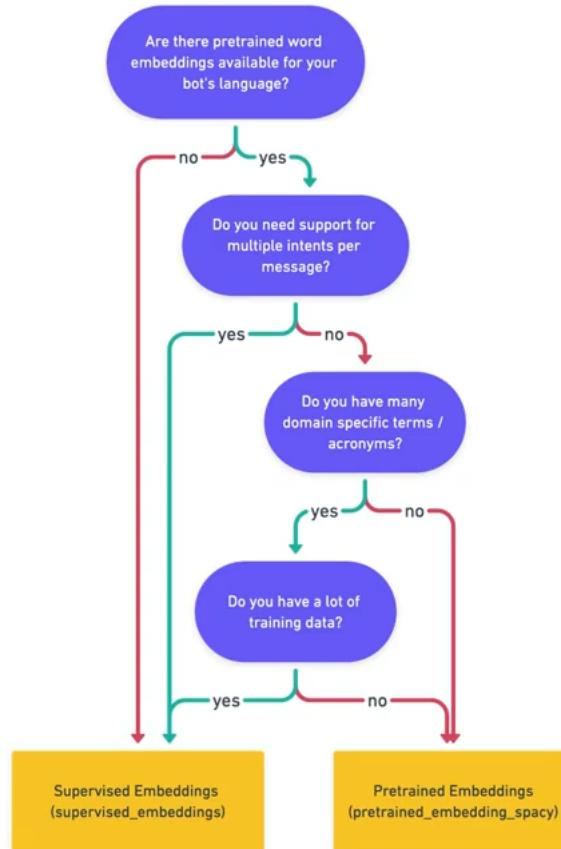


Figure 11: Pretrained oder Supervised Embeddings [23, 30]

4.2.2 Konfigurierte Pipelines

Es gibt also vordefinierte Pipelines von Rasa, die man nutzen kann.

Dabei kann man nun wieder entscheiden, ob man Pretrained Word Embeddings will oder nicht. Sollte man sich dafür entscheiden, kann man auf eine Pipeline mit SpaCy zurückgreifen, welche den SpacyFeaturizer verwendet und somit Pretrained Word Embeddings nutzt. Diese ist in Listing 14 zu sehen.[37, 38]

Listing 14: Spacy Startpipeline

```

1 language: "de" # Code f r die Sprache
2
3 pipeline:
4   - name: SpacyNLP
5     model: "de_core_news_lg" # Spezifizierung, welches Modell von SpaCy genommen
        werden soll
6   - name: SpacyTokenizer
7   - name: SpacyFeaturizer
8   - name: RegexFeaturizer
9   - name: LexicalSyntacticFeaturizer
10  - name: CountVectorsFeaturizer
11  - name: CountVectorsFeaturizer
12    analyzer: "char_wb"
13    min_ngram: 1
14    max_ngram: 4
15  - name: DIETClassifier
16    epochs: 100
17  - name: EntitySynonymMapper
18  - name: ResponseSelector
19    epochs: 100

```

Wenn man sich dafür entscheidet keine vortrainierten Word Embeddings zu nutzen und somit sein Modell spezifischer auf den eigenen Anwendungsfall anpassen möchte, verwendet man die Standard Rasa Pipeline. Diese verwendet den CountVectorsFeaturizer, bei dem nur die Trainingsdaten, die zur Verfügung gestellt werden, trainiert werden.[37, 38, 39]

Listing 15: Default Pipeline

```

1 language: "de" # Code f r die Sprache
2
3 pipeline:
4   - name: WhitespaceTokenizer
5   - name: RegexFeaturizer
6   - name: LexicalSyntacticFeaturizer
7   - name: CountVectorsFeaturizer
8   - name: CountVectorsFeaturizer
9     analyzer: "char_wb"
10    min_ngram: 1
11    max_ngram: 4
12  - name: DIETClassifier
13    epochs: 100
14  - name: EntitySynonymMapper
15  - name: ResponseSelector
16    epochs: 100

```

Die von uns verwendete Pipeline aus Listing 16 beruht auf dieser Standard-Pipeline und die jeweiligen Komponenten werden in den folgenden Kapiteln genauer erläutert.

Listing 16: Unsere Pipeline

```

1 language: de
2
3 pipeline:
4   - name: WhitespaceTokenizer
5   - name: RegexFeaturizer
6   - name: LexicalSyntacticFeaturizer
7   - name: CountVectorsFeaturizer
8   - name: CountVectorsFeaturizer
9     analyzer: char_wb
10    min_ngram: 1
11    max_ngram: 4
12   - name: DIETClassifier
13   epochs: 100
14   constrain_similarities: true
15   - name: EntitySynonymMapper
16   - name: ResponseSelector
17   epochs: 100
18   retrieval_intent: faq
19   - name: ResponseSelector
20   epochs: 100
21   retrieval_intent: chitchat
22   - name: FallbackClassifier
23   threshold: 0.3
24   ambiguity_threshold: 0.1

```

4.2.3 WhitespaceTokenizer

Ein WhitespaceTokenizer ist eine sehr einfache Art eines Tokenizers. Hierbei wird, wie der Name bereits vermuten lässt, der Satz aufgeteilt in verschiedene Tokens. Ein Token kann zum Beispiel also ein Wort sein und beim WhitespaceTokenizer wird der Satz pro Whitespace also Leerzeichen aufgeteilt.[40, 41, 42]



Figure 12: Tokens durch einen WhitespaceTokenizer [42]

Es gibt auch komplexere Tokenizer und Tokenizer, die speziell an Sprachen angepasst sind und die besonderen Regeln, die in diesen Sprachen gelten, beachten. Außerdem kann man natürlich auch selber einen Tokenizer schreiben.[40, 41, 42]

Tokenizer sollten generell weit am Anfang einer Pipeline sein, wenn nicht sogar die erste Komponente.

4.2.4 RegexFeaturizer

Einen RegexFeaturizer kann man verwenden, um die EntityExtraction zu erleichtern. Bei diesem werden Regular Expressions und Lookup Tables verwendet und der RegexFeaturizer

gibt an, ob ein Wort von den Regular Expressions oder Lookup Tables erkannt wurde. Der EntityExtractor nimmt dann diese Tokens als Input und produziert Named Entity Recognition Ergebnisse.[43, 42, 44]

Regular Expressions

Regular Expressions können verwendet werden, um Muster eines Strings zu beschreiben und können hier bei dem Beispiel von Chatbots beispielsweise verwendet werden, um Zahlen zu beschreiben.[43, 42, 44]

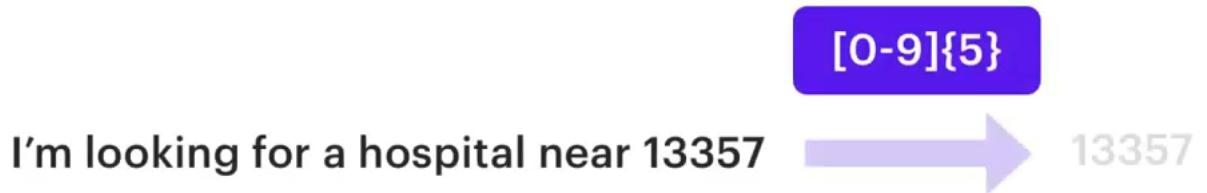


Figure 13: Regular Expression Beispiel [42]

Lookup Tables

Lookup Tables können verwendet werden, wenn eine Entity vordefinierte Werte haben kann. Zum Beispiel kann eine Entity mit dem Namen "country" 194 verschiedene Länder annehmen.[43, 42, 44]



Figure 14: Lookup Table Beispiel [42]

Der RegexFeaturizer muss vor dem EntityExtractor in der Pipeline platziert werden.[43, 42, 44]

4.2.5 CRFEntityExtractor

Eine Möglichkeit um Entities aus der Nachricht des Benutzers zu extrahieren ist über den CRFEntityExtractor.[45]

CRF steht hierbei für Conditional Random Field. Dieses Modell lernt, welche Komponenten eines Satzes Entities sind und welche Entities diese sind.[45, 42, 44] Dies macht der CRFEntityExtractor, indem er die Sequenzen der Tokens beobachtet. Es wird also ein Token aus dem Satz herausgepickt, und anschließend wird geschaut, ob die Wörter danach und davor zum Kontext des Worts beitragen, um zu erkennen, ob es sich hierbei um eine Entity handelt. Dabei schaut der Extractor auf Eigenschaften, wie beispielsweise, ob das Wort groß- oder kleingeschrieben ist, ob es einen Prefix hat, ob es eine Zahl ist oder ob es ein speziell definiertes Wort für die Entities ist. Dies wird auch mit den Wörtern davor und danach gemacht.[45, 42, 44]

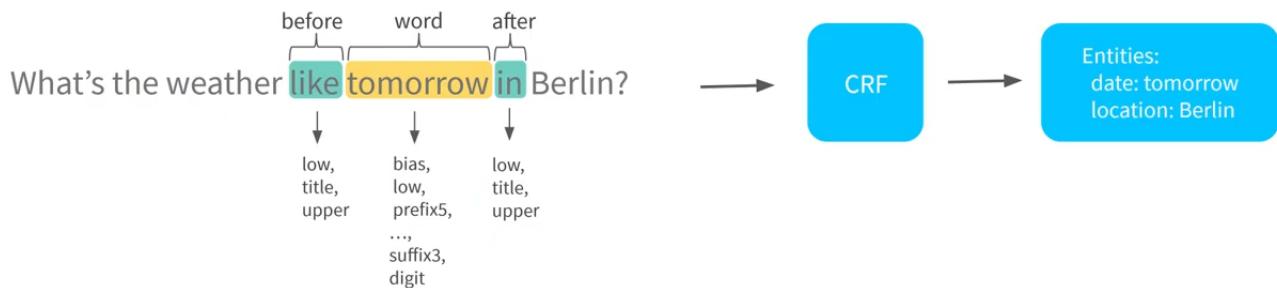


Figure 15: Funktionsweise vom CRFEntityExtractor [42]

Der CRFEntityExtractor produziert anschließend als Output, welche Wörter in einem Satz Entities sind und welche Entities diese sind, also was ihre Labels sind. Außerdem wird noch produziert, wie sicher das Modell war, dass diese Entities korrekt sind, also die sogenannte Confidence wird ausgegeben und welches Modell verwendet wurde.[45, 42, 44]

4.2.6 LexicalSyntacticFeaturizer

Ein Featurizer generell wird verwendet, um Features von den Tokens zu extrahieren. Diese können dann von dem Intent-Klassifikations-Modell genutzt werden, um Muster zu erkennen und anschließend den korrekten Intent vorherzusagen.[46, 42, 47]

Der LexicalSyntacticFeaturizer bekommt als Input einen Token und erstellt dann Features für die Entity Extraktion. Dabei verwendet er ein sogenanntes Sliding Window, welches über jeden Token in der Nachricht des Benutzers verwendet wird.[46, 42, 47]

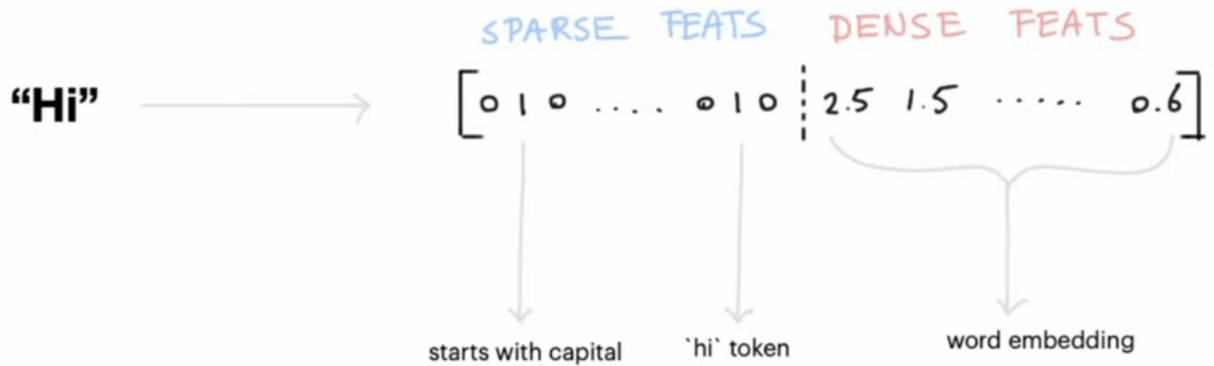


Figure 16: Funktionsweise von einem Featurizer [47]

4.2.7 CountVectorsFeaturizer

Der CountVectorsFeaturizer erstellt Bag-of-Words. Dabei wird gezählt, wie oft ein bestimmtes Wort aus den Trainingsdaten in der Nachricht des Benutzers vorkommt. Dies wird dann als Input dem Intent-Classification-Model übergeben.[48, 47, 42, 49]

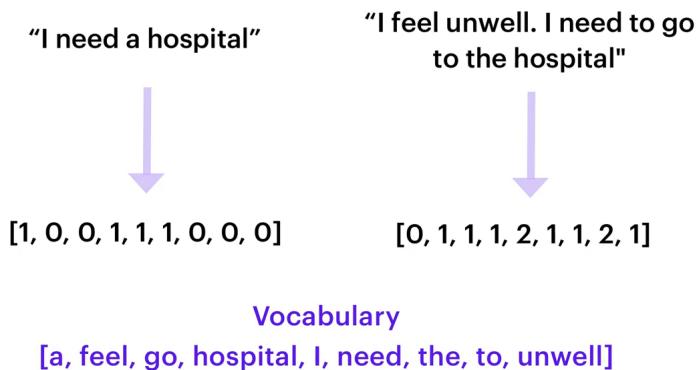


Figure 17: CountVectorsFeaturizer Beispiel [47]

Beim CountVectorsFeaturizer kann man außerdem angeben, dass statt Wörtern sogenannte n-grams verwendet werden sollen. Dies macht das Modell in der Regel robuster gegen Typos allerdings erhöht sich damit auch die Zeit, die das Modell zum Trainieren benötigt.[48, 47, 42, 49]

Listing 17: CountVectorsFeaturizer

```

1 pipeline:
2   - name: CountVectorsFeaturizer
3     analyzer: char_wb
4     min_ngram: 1
5     max_ngram: 4

```

4.2.8 DIETClassifier

4.2.9 EntitySynonymMapper

Der EntitySynonymMapper erwartet als Input einen Extractor von den Entity Extractors und liefert als Ausgabe modifizierte Werte für die Entities, die bereits vorher erkannt wurden. Wenn beim EntitySynonymMapper also eine Entity überliefert wird und in den Trainingsdaten Synonyme für diese Entity vorhanden sind, wird der Wert der Entity auf den Wert, der in den Trainingsdaten angegeben ist, gesetzt. Wenn also beispielsweise in einem Satz des Benutzers New York City oder NYC vorkommt, werden diese beide auf den selben Wert nyc gesetzt.[50]

Listing 18: Entity Synonym Mapper

```

1  [
2    {
3      "text": "I moved to New York City",
4      "intent": "inform_relocation",
5      "entities": [
6        {
7          "value": "nyc",
8          "start": 11,
9          "end": 24,
10         "entity": "city",
11       }],
12     },
13   {
14     "text": "I got a new flat in NYC.",
15     "intent": "inform_relocation",
16     "entities": [
17       {
18         "value": "nyc",
19         "start": 20,
20         "end": 23,
21         "entity": "city",
22       }]
22 ]

```

4.2.10 ResponseSelector

Ein Selector sagt die korrekte Response voraus aus einer Menge von möglichen Responses. Beim ResponseSelector wird als Ausgabe ein Key-Value Paar ausgegeben. Als Key wird hierbei der Retrieval Intent ausgegeben und das Value ist die vorhergesagte Response und die Confidence, mit der diese vorhergesagt wurde.[51]

Listing 19: Response Classifier

```

1  {
2    "response_selector": {
3      "faq": {
4        "response": {
5          "id": 1388783286124361986,
6          "confidence": 0.7,
7          "intent_response_key": "chitchat/ask_weather",
8          "responses": [
9            {
10              "text": "It's sunny in Berlin today",
11              "image": "https://i.imgur.com/nGF1K8f.jpg"
12            },

```

```

13         {
14             "text": "I think it's about to rain."
15         },
16     ],
17     "utter_action": "utter_chitchat/ask_weather"
18 },
19 "ranking": [
20     {
21         "id": 1388783286124361986,
22         "confidence": 0.7,
23         "intent_response_key": "chitchat/ask_weather"
24     },
25     {
26         "id": 1388783286124361986,
27         "confidence": 0.3,
28         "intent_response_key": "chitchat/ask_name"
29     }
30 ],
31 }
32 }
33 }
```

Bei einem Retrieval Intent handelt es sich um einen speziellen Intent, der weiter aufgeteilt werden kann in sogenannte Sub-Intents. Dies kann man zum Beispiel nutzen, wenn man einen Retrieval Intent für FAQ und für Chitchat hat, bei denen dann jede individuelle Frage einen Sub-Intent darstellt.[52] Verwendet wird dies, wenn der Bot zum Beispiel bei FAQ oder Chitchat sowieso immer mit der selben Antwort auf die Frage antworten soll, unabhängig von dem was vorher geschrieben wurde.[53]

Auch wir haben diesen Ansatz in unsere Pipeline gewählt, wie ihn 20 zu sehen ist.

Listing 20: Response Selector für Chitchat und FAQ

```

34 - name: ResponseSelector
35   epochs: 100
36   retrieval_intent: faq
37 - name: ResponseSelector
38   epochs: 100
39   retrieval_intent: chitchat
```

4.2.11 FallbackClassifier

Der FallbackClassifier ist standardmäßig nicht in der Rasa Pipeline enthalten und wurde von uns hinzugefügt.[37]

Diesen verwendet man, um mit Nachrichten umzugehen, bei denen nur eine sehr niedrige Confidence vorhergesagt wurde. In diesem Fall wird dann ein Intent mit dem Namen "nlu_fallback" vorhergesagt, welchen man dann behandeln kann, indem man beispielsweise als Antwort immer definiert, dass der User seine Nachricht neu formulieren soll. Die Confidence wird hierbei auf den Wert gesetzt welcher im Threshold angegeben wird.[54, 55] Man kann außerdem einen sogenannten "ambiguity_threshold" angeben. Bei diesem wird der "nlu_fallback" Intent vorhergesagt, wenn die zwei als

wahrscheinlichst empfundenen Intents sich um weniger, als den ambiguity_threshold in der Confidence unterscheiden.[54]

Listing 21: Fallback Classifier

```

1 pipeline:
2 - name: FallbackClassifier
3   threshold: 0.7
4   ambiguity_threshold

```

4.3 Welche Rolle spielen neuronale Netze in Rasa

Rasa selbst definiert 5 Stufen von AI, wie in Abbildung 18 zu sehen ist..[56]

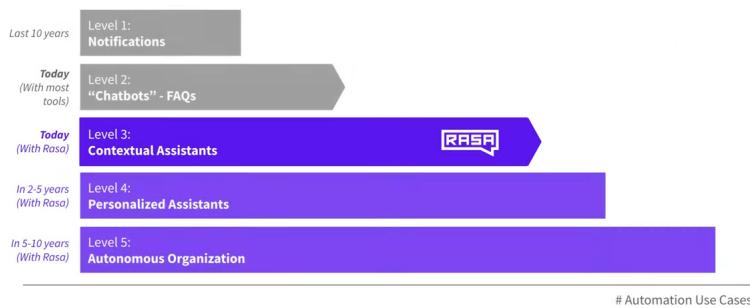


Figure 18: 5 Stufen von AI [57]

Level 1: Notifications

Auf Level 1 geht es darum simple Aufgaben zu erfüllen, wie man sie möglicherweise von seinem Smartphone kennt. Darunter fällt zum Beispiel das Anlegen von Terminen und anderen Benachrichtigungen, die man dann zu der eingestellten Uhrzeit bekommt.[58, 56, 57]

Level 2: "Chatbots" & FAQs

Auf Level 2 geht es darum, dass der Benutzer einfache Fragen stellen kann und anschließend der Chatbot auf diese antwortet. Diese Stufe findet man sehr oft, allerdings sind diese sehr fehleranfällig, weil hier nur eine Menge an Regeln angelegt wird, an die sich der Chatbot hält. Dies wird sehr oft in Form von FAQs genutzt und auch ein paar kleine Follow-up Fragen können hierbei schon definiert sein.[58, 56, 57]

Level 3: Contextual AI Assistants

Dieses Level wird derzeit von Rasa unterstützt. Zusätzlich zu normalen FAQs wird hierbei auch der Kontext beachtet. Es macht nämlich einen Unterschied, wann, wie und in welcher Situation ein Benutzer etwas gesagt hat. Bei Contextual Assistants wird

der Kontext, also was bereits zuvor gesagt wurde, auch beachtet. Außerdem lenken sie Konversationen in die gewünschte Richtung und werden mit der Zeit immer besser, wenn man sie trainiert.[58, 56, 57]

4.4 Komponenten

In der sogenannten Domain werden alle Intents, Entities, Slots, Responses, Forms und Actions angegeben, die der Bot kennt. Diese ganzen Informationen befinden sich in der **domain.yml** Datei.[59]

4.4.1 Intents

Intents sind die Absichten hinter der Nachricht des Benutzers. Als Intents werden also alle möglichen Beispielsätze definiert, die ein Benutzer sagen könnte, um eine bestimmte Absicht auszudrücken.[60]

Intents werden in dem **nlu.yml** File wie folgt angegeben:

Listing 22: Intents Beispiel

```

1  ## intent:<name des intents>
2  - <phrase 1>
3  - <phrase 2>
4  - <phrase 3>
```

4.4.2 Responses

Responses sind die Antworten, die vom Bot gegeben werden, wenn ein bestimmter Intent erkannt wurde.[61]

Responses fügt man in seinem **domain.yml** File wie folgt ein:

Listing 23: Responses Beispiel

```

1  responses:
2    utter_<name der response>:
3      - text: "<text>"
4      - image: "<img link>"
5      - buttons:
6        - title: "<button title>"
7        - payload: "<payload>"
8      ...
9
10 z.B.
11
12 utter_chitchat/what_are_you_doing:
13   - text: Ach nichts besonderes, nur ein wenig chatten!
14   - text: Ich entspanne gerade ein wenig aber durchl cher mich ruhig mit Fragen
         ich muss sowieso wieder an die Arbeit.
15   - text: Frag mich ruhig etwas ich brauche sowieso eine Ablenkung von all dem
         was ich eigentlich machen sollte!
```

```

16     buttons:
17         - title: "Wie geht es dir?"
18             payload: "Wie geht es dir?"
19         - title: "Willst du mich heiraten?"
20             payload: "Willst du mich heiraten?"
21         - title: "Welche Technologien wurden f r dich verwendet?"
22             payload: "Welche Technologien wurden f r dich verwendet?"
23         - title: "Wie sp t ist es?"
24             payload: "Wie sp t ist es?"

```

4.4.3 Stories

Stories werden als Trainingsdaten verwendet, die zum Trainieren des Models des Bots verwendet werden. Stories können dabei genutzt werden, um Models zu trainieren, bei denen auch unvorhersehbare Konversationspfade behandelt werden und unterscheiden sich in dieser Hinsicht von den Rules.[62]

Bei einer Story wird also die Unterhaltung zwischen einem Benutzer und dem Bot dargestellt. Dabei wird die Eingabe des Benutzers als Intent angegeben und die Antwort, mit der der Bot antworten soll, als Name der Action.[62]

Stories können wie folgt aussehen und sind im **stories.yml** File anzugeben:

Listing 24: Stories Beispiel

```

1 stories:
2   - story: name der story
3     steps:
4       - intent: <name des intents>
5       - action: <name der action>
6
7 z.B.
8 stories:
9   - story: sad path 1
10    steps:
11      - intent: greet
12      - action: utter_greet
13      - intent: mood_unhappy
14      - action: utter_cheer_up
15      - action: utter_did_that_help
16      - intent: affirm
17      - action: utter_happy

```

Checkpoints und OR-Statements

Man kann seine Stories außerdem mit Checkpoints und OR-Statements versehen. Bei diesen sollte man aber grundsätzlich aufpassen und sie nur bedacht verwenden, weil in den meisten Fällen die gewünschten Resultate besser mit **Rules** oder einem **ResponseSelector** zu erzielen sind.[63]

Checkpoints können genutzt werden, um seine Trainingsdaten zu vereinfachen, indem man einen Checkpoint in einer Story setzt und auf diesen in einer anderen Story wieder ansetzt. Von diesen sollte man allerdings nicht zu viele machen, weil sonst die Stories

sehr leicht schwer zu lesen und unübersichtlich sind und außerdem die Trainingszeit dadurch erhöht wird.[64]

Man definiert einen Checkpoint am Ende seiner Story, wenn man diesen Teil der Story auch wieder als Voraussetzung für eine weitere Story setzt. In der nächsten Story beginnt man dann mit seinem Checkpoint, also dem Punkt auf den man anknüpfen möchte und die Teile der Konversation, die für diese Story ebenfalls vorausgesetzt werden sollen.

Im folgenden Beispiel werden Checkpoints von Stories verwendet, um an anderen Stories anzuknüpfen[64]:

Listing 25: Checkpoints Beispiel

```

1 stories:
2 - story: beginning_of_flow
3   steps:
4     - intent: greet
5     - action: action_ask_user_question
6     - checkpoint: check_asked_question
7
8   - story: handle_user_affirm
9     steps:
10    - checkpoint: check_asked_question
11    - intent: affirm
12    - action: action_handle_affirmation
13    - checkpoint: check_flow_finished
14
15   - story: handle_user_deny
16     steps:
17     - checkpoint: check_asked_question
18     - intent: deny
19     - action: action_handle_denial
20     - checkpoint: check_flow_finished
21
22 - story: finish_flow
23   steps:
24   - checkpoint: check_flow_finished
25   - intent: goodbye
26   - action: utter_goodbye

```

OR-Statements können dafür verwendet werden, wenn man auf mehrere Intents innerhalb einer Story gleich reagieren möchte.[65]

Man schreibt also anstelle von einem Intent in der Story ein **or** und gibt darunter alle Intents an, von denen einer eintreffen muss, damit die Story zutrifft.

Listing 26: OR Beispiel

```

1 stories:
2 - story:
3   steps:
4   # ... vorherige Schritte
5   - action: utter_ask_confirm
6   - or:
7     - intent: affirm
8     - intent: thankyou
9     - action: action_handle_affirmation

```

4.4.4 Rules

Rules werden angegeben, um kleine Teile von Unterhaltungen anzugeben, die immer wieder gleich behandelt werden sollen. Diese sollten allerdings nicht allzu häufig verwendet werden, weil man nie alle Konversationen vorhersagen kann. Um Rules verwenden zu können, muss man die **RulePolicy** in der Policy Konfiguration eintragen.[66]

Um eine Rule zu verwenden, schreibt man folgendes in sein **rules.yml** File:

Listing 27: Rules Beispiel

```

1  rules:
2
3  - rule: Say 'hello' whenever the user sends a message with intent 'greet'
4    steps:
5      - intent: greet
6      - action: utter_greet

```

4.4.5 Slots

Slots sind sozusagen das Gedächtnis des Bots. Diese sind als key-value Paare dargestellt und können dazu verwendet werden, damit Information, die der Benutzer bereitstellt, gespeichert werden können, ähnlich zu Entities. Diese Informationen können beispielsweise der Name des Benutzers sein oder Informationen, die für den generellen Kontext des Gesprächs wichtig sind.[67]

Listing 28: Slots Beispiel

```

1  slots:
2    slot_name: <slot name>
3      type: <type>
4
5 z.B.
6
7 slots:
8   name:
9     type: rasa.shared.core.slots.TextSlot
10    initial_value: null
11    auto_fill: true
12    influence_conversation: false

```

4.4.6 Entities

Entities sind strukturierte Stücke von Informationen, die sich innerhalb der Nachricht eines Benutzers befinden. Solche Entities können beispielsweise ein Ort, ein Beruf oder ein Name sein.[68]

Um Entities zu erstellen schreibt man folgendes in sein **domain.yml** File:

Listing 29: Entities in der Domain

```

1  entities:
2    - <entity name>
3    - <entity name>
4
5 z.B.
6
7 entities:
8  - name
9  - branch
10 - teacher
11 - class
12 - grade
13 - subject

```

Diese Entities müssen dann noch in den Intents angegeben werden, in denen sie vorkommen sollen. Dies macht man, indem man folgende Syntax bei den Trainingssätzen im **nlu.yml** File verwendet und ergänzt:

Listing 30: Entities Beispiel

```

1 Hallo mein Name ist [Lukas](name).
2 Ich h tte gerne eine [kleine](size) [Pizza](meal)

```

Entity Roles

Entity Roles können sinnvoll in manchen Szenarien sein. Zum Beispiel bei folgendem Satz:

Listing 31: Entity Roles Beispiel

```
1 Buche einen Flug von [Linz](city) nach [London](city).
```

In diesem Fall sind sowohl Linz als auch London zwar richtig gekennzeichnet als city Entity, allerdings reicht diese Information noch nicht aus, damit der Chatbot richtig reagieren kann. Hierbei wäre es praktisch, wenn man noch angibt, welche dieser zwei Städte das Ziel und welche der Abflugsort ist. Dies macht man mit Entity Roles.[69]

Entity Groups

Entity Groups können genutzt werden, wenn man Entities miteinander gruppieren möchte.[69]

Dies kann zum Beispiel hier sinnvoll sein:

Listing 32: Entity Groups Beispiel

```
1 Ich h tte gerne eine kleine [Pizza](meal) mit [Pilzen](topping) und eine
[Salami](topping) [Pizza](meal).
```

Bei der Gruppe muss hier erkannt werden, welche zwei Entities zusammen gehören[69]:

Listing 33: Entity Groups Beispiel 2

```

1 Ich h tte gerne eine kleine [Pizza](meal) mit [Pilzen](topping) und eine
   [Salami](topping) [Pizza](meal).
2 Group 1: [Pizza](meal) [Pilzen](topping)
3 Group 2: [Salami](topping) [Pizza](meal)

```

Nutzung von Entity Roles und Entity Groups

4.4.7 Actions

Es gibt 2 verschiedene Arten von Messages:

1. **Static Messages:** Diese sind unabhängig vom User Input und benötigen keinen Action Server[70]
2. **Dynamic Messages:** Diese sind abhängig vom User Input und benötigen einen Action Server[70]

Der Rasa Action Server führt sogenannte Custom Actions für einen Rasa Open Source Conversation assistent aus.

Wenn der Assistant eine gewisse Custom Action vorhersagt, sendet der Rasa Server einen POST request an den Actionserver mit einer JSON Payload mit dem Namen der vorhergesagten Action, der Conversation ID, den Inhalten des Trackers und den Inhalten der Domain.[71]

4.4.8 Forms

Um mehrere Informationen von einem Benutzer zu bekommen, eignen sich Forms. Um Forms zu verwenden, muss die **RulePolicy** in der Policy Konfiguration eingetragen sein.[72]

Wenn man ein Formular hinzuzufügen will, muss man dies in der forms Section in dem **domain.yml** File angeben.

Listing 34: Forms Beispiel

```

1 forms:
2   restaurant_form:
3     required_slots:
4       cuisine:
5         - type: from_entity
6           entity: cuisine
7     num_people:
8       - type: from_entity
9         entity: number

```

4.4.9 Synonyms

Mithilfe von Synonymen kann man extrahierten Entities einen anderen Wert geben, als sie eigentlich vorher hatten, wenn diese in der Bedeutung gleich sind. Wenn man also mit verschiedenen Wörtern dasselbe meint, kann man sich Synonyms zur Hilfe nehmen.[73]

Ein Beispiel dafür wäre folgendes im **nlu.yml** File:

Listing 35: Synonym Beispiel

```

1 - synonym: Medientechnik
2   examples: |
3     - IT-Medientechnik
4     - IT Medientechnik
5     - Medientechnologie

```

4.4.10 Custom Actions

Neben normalen Responses kann man auch Custom Actions definieren. Diese sind in Python zu schreiben und dabei kann man seinen eigenen Code für Berechnungen oder dergleichen verwenden. Hierbei muss man eine Klasse implementieren, die Kind der Klasse **Action** ist und eine Methode **name** und **run** besitzt. Eine Custom Action wird bei unserem Chatbot beispielsweise für die Ausgabe des aktuellen Datums genutzt, wie bei Listing 36 zu sehen ist.

Listing 36: Custom Action für die Ausgabe des aktuellen Datums

```

1 class ActionWhatDateIsIt(Action):
2     def name(self) -> Text:
3         return "action_what_date_is_it"
4
5     def run(self, dispatcher: CollectingDispatcher,
6             tracker: Tracker,
7             domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
8
9         now = datetime.now()
10        weekday = datetime.today().weekday()
11        weekday_string = ("Montag", "Dienstag", "Mittwoch", "Donnerstag",
12                          "Freitag", "Samstag", "Sonntag")[weekday]
13
14        current_date = now.strftime("%d.%m.%Y")
15        print("Current Date =", current_date)
16
17        dispatcher.utter_message(text=f"Heute ist {weekday_string} der
18                                  {current_date}!")
19
20    return []

```

Nun muss man die Custom Actions in seiner **domain.yml** Datei definieren, wie in 37.

Listing 37: Custom Actions in Domain definiert

```

1 actions:
2   - action_what_date_is_it

```

4.5 Initialisieren

4.5.1 Rasa Init

Einen neuen Rasa Assistant kann man mithilfe des Init-Befehls in der Konsole anlegen. Der Befehl dafür lautet wie in Listing 38 zu sehen ist.

Listing 38: Befehl fürs Initialisieren

```
1 rasa init
```

Bei diesem Befehl wird man gefragt, ob man direkt ein voreingestelltes Model trainieren möchte. Dieses Modell wird basierend auf den Demodaten, die von jedem neu erstellten Rasa Projekt zur Verfügung gestellt werden, trainiert.

```
(rasa_env) lukas@lukas-IdeaPad-5-15ALC05:~/Documents/Schule$ rasa init
Welcome to Rasa! 🤖

To get started quickly, an initial project will be created.
If you need some help, check out the documentation at https://rasa.com/docs/rasa.
Now let's start! 🚀

? Please enter a path where the project will be created [default: current directory] chatbot-demo
? Path 'chatbot-demo' does not exist 🛡️. Create path? Yes
Created project directory at '/home/lukas/Documents/Schule/chatbot-demo'.
Finished creating project structure.
? Do you want to train an initial model? 💪 (Y/n)
```

Figure 19: Rasa Init Konsolenausgabe

Nachdem dieser Befehl ausgeführt wurde, bekommt man folgende Files, die für einen Chatbot benötigt werden.

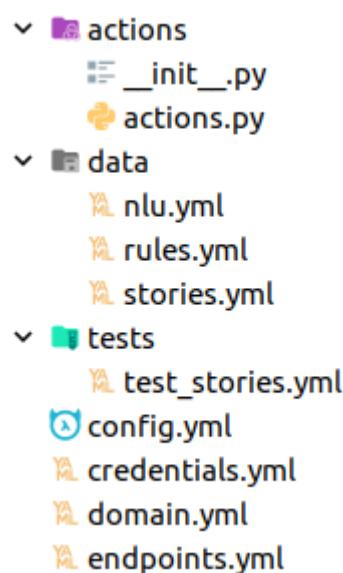


Figure 20: File Tree nach dem Initialisieren

4.6 Trainieren

4.6.1 Rasa Train

Um das Modell nun zu trainieren, nützt man den `Rasa train` Befehl. Dabei werden nur die Komponenten (Core oder NLU) trainiert, die sich auch geändert haben im Vergleich zu dem letzten Modell. Man kann aber auch selbstständig angeben, dass man nur `core` oder `nlu` trainieren möchte. Nach dem Trainieren wird das Modell in den Pfad gespeichert, der beim `-out` definiert wurde und standardmäßig `/models` ist. Dabei wird als Modellname der aktuelle Zeitstempel gewählt, außer man verwendet die `-fixed-model-name` Flag. Wenn man andere Pipelines oder Policies verwenden möchte, kann man auch ein anderes config-File, in dem diese gespeichert sind, angeben, indem man `-c` schreibt. Ab Rasa 2.2 hat man außerdem die Möglichkeit eine Feinabstimmung an einem bereits bestehenden Modell vorzunehmen. Dies macht man, wenn man neue Trainingsdaten bei bereits bestehenden Intents hinzugefügt hat. Hierbei ergibt sich eine viel kürzere Trainingszeit, als wenn das Modell nochmal von vorne trainiert würde und man gibt dabei den Pfad zum Modell an, wenn man nicht, wie standardmäßig definiert, das neueste trainieren möchte.[74]

Listing 39: Rasa Train Befehle

```

1 rasa train
2 rasa train -c config.yml #Das verwendete Config File (default: config.yml)
3 rasa train --out trained-models/ #Der Pfad, in dem das Modell gespeichert wird
  (default: models/)
4 rasa train --fixed-model-name leobot-model #Der Name des Modells (default:
  <timestamp>.tar.gz)
5 rasa train core #Rasa Core mit den Stories trainieren
6 rasa train nlu #Rasa NLU mit den NLU-Daten trainieren
7 rasa train --finetune models/20220211-094939.tar.gz #Finetune eines bestehenden
  Modells

```

4.7 Interagieren

4.7.1 Rasa Shell

Wenn man sich nun mit seinem Bot unterhalten möchte, kann man eine Chat Session über die Konsole mit dem `rasa train` Befehl. Normalerweise wird dabei das aktuellste Modell genommen. Sollte man dies allerdings nicht wollen, kann man mit `-model` auch den Pfad zum richtigen Modell angeben.

Listing 40: Rasa Shell Befehle

```
1 rasa shell
```

```

2 rasa shell --model models/20220211-094939.tar.gz #Das Modell, das verwendet werden
    soll
3 rasa shell --debug #Debug-Modus aktivieren

```

Für detailliertere Ausgaben kann man außerdem das `--debug` Flag verwenden. Die Ausgabe dabei sieht wie in Abbildung 21 aus:

```

2022-03-30 17:49:47 DEBUG  rasa.core.policies.ensemble - Predicted next action using policy_0_MemoizationPolicy.
2022-03-30 17:49:47 DEBUG  rasa.core.processor - Predicted next action 'utter_htl_age' with confidence 1.00.
2022-03-30 17:49:47 DEBUG  rasa.core.processor - Policy prediction ended with events '[<rasa.shared.core.events.DefinePr
evUserUtteredFeaturization object at 0x7f89ade8ed30>]'.
2022-03-30 17:49:47 DEBUG  rasa.core.processor - Action 'utter_htl_age' ended with events '[BotUttered('Die HTL Leonding
wurde 1984 gegründet und der Zweig für EDV und Organisation entstand 1985.'), {"elements": null, "quick_replies": null, "bu
ttons": [{"title": "Wie schwer ist die HTL?", "payload": "/ask_htl_difficulty"}, {"title": "Wie lange dauert die HTL?", "pa
yload": "/htl_duration"}, {"title": "Wie komme ich zur HTL?", "payload": "/public_transport_connection"}, {"title": "Zeig mir
ein Bild der HTL", "payload": "/show_school_picture"}], "attachment": null, "image": null, "custom": null}, {"utter_acti
on": "utter_htl_age"}, 1648655387.435862)]'.

```

Figure 21: Rasa Shell Ausgabe mit Debug Flag

4.7.2 Rasa Interactive

Um eine interaktive Session mit dem Bot zu starten, kann man den Befehl `rasa interactive` nutzen. Dabei wird zunächst ein Modell trainiert und anschließend startet eine interaktive Shell Session. Dabei kann man dann alle vorhergesagten Aktionen vom Assistant korrigieren. Wenn man nur das Core Model testen möchte, kann man dies spezifizieren.

Listing 41: Interaktive Trainingssession starten

```

1 rasa interactive
2 rasa interactive --model models/20220211-094939.tar.gz #Das Modell, das verwendet
    werden soll
3 rasa interactive core #Interaktive Session f r das Core Modell starten

```

Bei dieser Trainingssession hat man die Möglichkeit jedes Mal anzugeben, ob der vorhergesagte Intent und die daraus resultierende Antwort die richtige sind. Außerdem wird die gesamte Chat History visualisiert und ausgegeben, wie in Abbildung 22 zu sehen ist.

```
? The bot wants to run 'utter_greet', correct? Yes
-----
Chat History

#      Bot                                     You
1      action_listen

2                                     Gibt es Mädchen an der HTL?
                                         intent: greet 1.00
                                         ...
                                         0.00
                                         action_listen 0.99
                                         ...
                                         0.96
                                         Hey! How are you?

5      utter_greet 0.96
                                         Hey! How are you?
```

Figure 22: Ausgabe beim Interactive Befehl

5 Implementierung

5.1 Systemarchitektur

Es gibt 3 große Services, das Chat Widget auf der Schulhompage, das Dashboard und das Backend. Unser System sieht folgendermaßen aus:

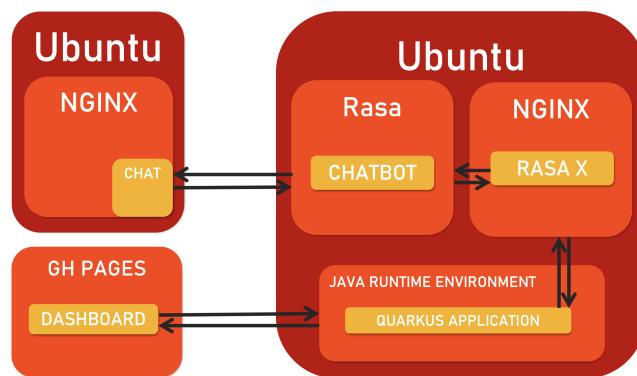


Figure 23: Systemarchitektur

Wir haben 2 Ubuntu VM's, eine für den Chat und eine für Rasa und das Backend, nachdem der Chat auf die HTML Leonding Seite kommt, wird dadurch nur noch eine VM benötigt.

Unser Chat wird gerade mit NGINX gehostet, dieser kommuniziert mit Rasa direkt über REST, wenn der Benutzer eine Nachricht sendet, wird diese an Rasa gesendet und Rasa antwortet mit der passenden Antwort. Das Dashboard wird auf GH Pages gehostet, dieses kommuniziert mit dem Backend über REST, das Backend authentisiert sich bei Rasa X und holt sich alle Konversationen und bereitet diese auf und sendet sie an das Dashboard.

5.2 Backend

Unser Quarkus3.2.2 Backend hat die Aufgabe mit Rasa X?? zu kommunizieren und die Konversationsdaten für das Dashboard aufzubereiten und zu senden.

Unser Backend besitzt folgende Endpoints:

- GET /api/conversations
- GET /api/conversations/id
- POST /api/feedback
- GET /api/feedback
- GET /api/file/filename
- PUT /api/file/filename

Bei den beiden “conversations” endpoints muss man sich aber zuerst authentisieren und sich einen Bearer Token von Rasa x holen damit man auf die anderen Rasa X endpoints zugreifen kann .

5.2.1 GET /api/conversations

Der Endpoint ruft zuerst die getAuth() funktion auf, um den Bearer Token zu erhalten.

Danach wird der Rasa X GET Endpoint /api/conversations aufgerufen mit dem Bearer Token im Header. Von diesem Endpoint wird ein JSON Objekt zurückgegeben, das die Konversationen enthält, aber zusätzlich noch viele andere unrelevante Daten, deshalb holt sich das Backend wirklich nur ID des Senders, die Zeit und die Anzahl von Nachrichten der Unterhaltungen.

5.2.2 GET /api/conversations/id

Der Endpoint /api/conversations/id ruft zuerst die getAuth() funktion auf, um den Bearer Token zu erhalten.

Um nur von einer gezielten Unterhaltung die Nachrichten zu erhalten wird der Rasa X Endpint /api/conversations/id/messages aufgerufen mit dem Bearer Token im Header. Die Response wird auch in dieser Form schon zurückgegeben, da die Response keine unwichtigen Daten enthält.

5.2.3 POST /api/feedback

Der Endpoint /api/feedback erhält ein JSON Objekt mit den Daten des Feedback Formulars und speichert diese in die Datenbank.

5.2.4 GET /api/feedback

Der Endpoint /api/feedback liest alle Feedbacks aus der Datenbank und gibt diese zurück.

5.2.5 GET /api/file/filename

Der Endpoint /api/file/filename liest die im URL angegeben Datei aus dem Filesystem aus und gibt diese zurück. Die möglichen Dateinamen sind:

- nlu.yml
- rules.yml
- stories.yml
- config.yml
- domain.yml

5.2.6 PUT /api/file/filename

Der Endpoint /api/file/filename erhält den Inhalt aus dem File welches auch im URL angegeben wurde und überschreibt dieses dann im Filesystem.

5.3 Chat Widget

Der Chatbot der HTL Leonding sollte auf der Schulhompage als Chatblase angezeigt werden, und verschiedene Elemente wie Buttons und Links unterstützen.

5.3.1 Konzept

Während den Anfängen der vorliegenden Arbeit wurde ein Konzept erstellt, um das mögliche Aussehen festzulegen. Lange Zeit wurde der Chatbot unter den Namen Leon geführt. Dies wurde jedoch im späteren Verlauf geändert und Leon wurde Teil des langjährigen Leonie Projektes der HTL Leonding. Ursprünglich war das Symbol des Chatbots, wie man am Konzept sehen kann, ein Bot. Durch den Wechsel in die Leonie Familie wurde dieses Logo jedoch gegen Leonie getauscht.

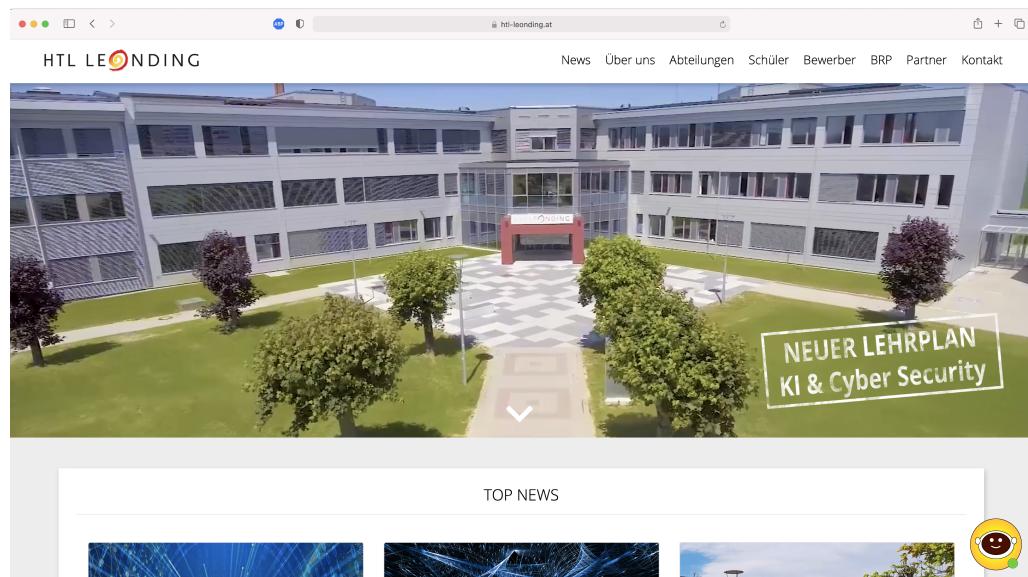


Figure 24: Konzept Chatbot geschlossen

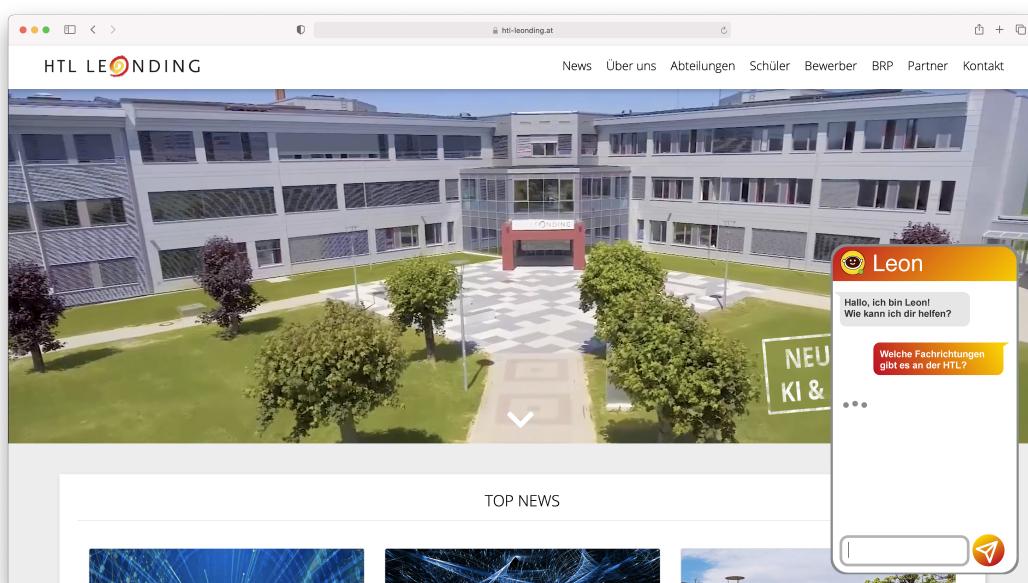


Figure 25: Konzept Chatbot geöffnet

5.3.2 Umsetzung

Umgesetzt wurde das Frontend mithilfe von Angular. Die Chatblase ist eine eigene Komponente, die durch CSS immer rechts unten fixiert ist. Die Farben des Chatbots sollten natürlich an die HTL Leonding erinnern, deshalb wurde ein Farbverlauf aus Farben des HTL Logos erstellt.

Jedoch begann der Chatbot sehr anders, zu beginn wurde der Bot zuerst als ganze Seite entwickelt und nicht nur als Chatblase.

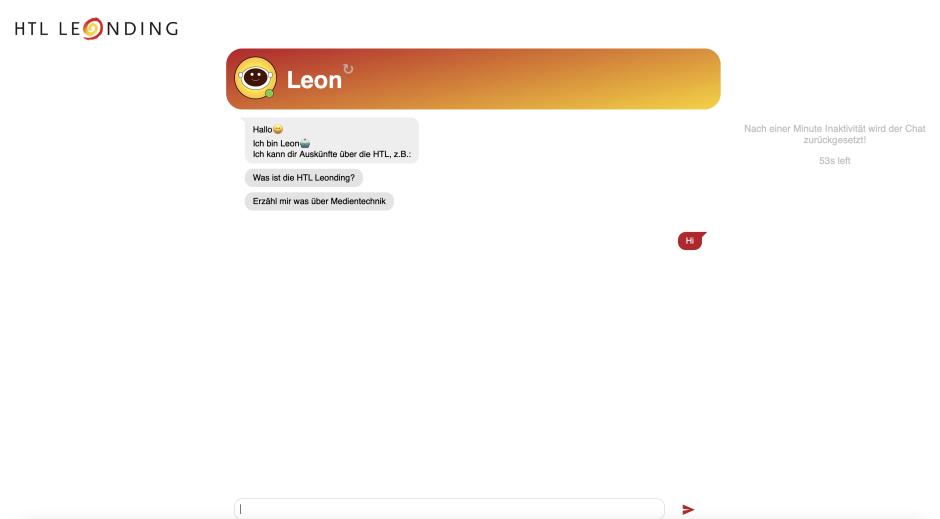


Figure 26: Chatbot auf einer ganzen Seite

Natürlich war dies nicht das endgültige Ziel so wurde der Bot schnell zur Chatblase umgewandelt. Zum Testen war die HTL Leonding Seite mithilfe eines IFRAMES eingebunden und zusätzlich Chatblase.

Um das Gespräch in eine Richtung zu lenken wurden Buttons eingeführt, die nach fast jeder Antwort mögliche folge Fragen vorschlagen.

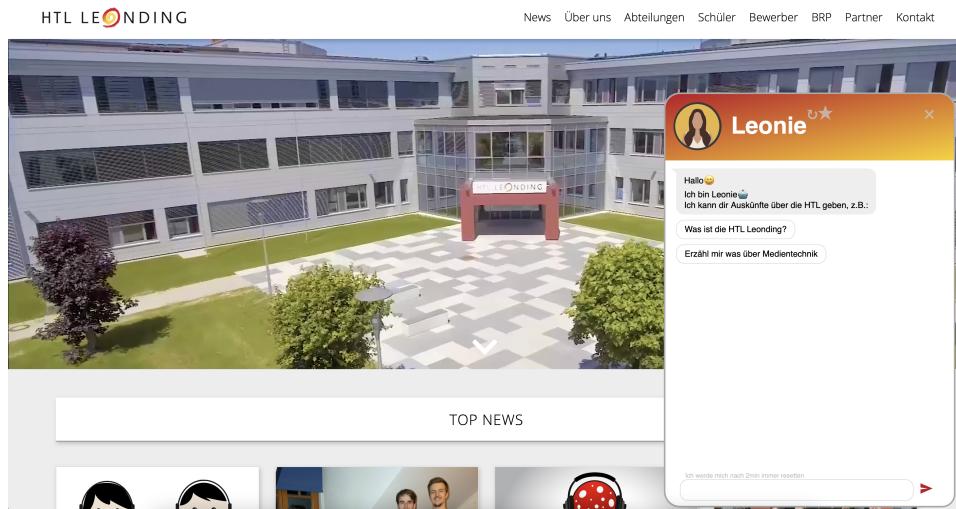


Figure 27: Chatbot

Um Bewertungen von echten Benutzern zu holen wurde außerdem eine Feedback Seite eingeführt, in dieser kann der Benutzer eine 1 bis 5 Sterne bewertung und einen Text absenden.

5.4 Dashboard

Um alle Gespräche und die Bewertungen der Benutzer anzuzeigen wurde ein Dashboard eingeführt, wo nur dies möglich war. Im Laufe der Arbeit wurde dieses dann erweitert, um für den Leobot Conversation Cycle als Seite zu dienen.

5.4.1 Leobot Conversation Cycle



Figure 28: Leobot Conversation Cycle

Der Leobot Conversation Cycle ist ein Workflow der dazu dient, den Leobot zu überprüfen und zu erweitern.

Der Workflow besteht aus folgenden Schritten:

- Unterhalten
- Überprüfen
- Verbessern
- Trainieren

Diese 4 Schritte werden immer wieder durchgeführt.

Unterhalten

Unterhalten ist der erste Schritt des Leobot Conversation Cycles, in diesem Schritt werden dem Bot Fragen gestellt, die er beantworten muss, dies passiert natürlich während Unterhaltungen im Chat auf der Schulhompage. Die Unterhaltungen werden dann alle gespeichert.

Überprüfen

Überprüfen ist der zweite Schritt des Leobot Conversation Cycles, in diesem Schritt wird der Bot überprüft, ob er die Fragen richtig beantwortet und erkannt hat. Dieser Schritt wird von Verantwortlichen für den Leobot durchgeführt.

Verbessern

Verbessern ist der dritte Schritt des Leobot Conversation Cycles, in diesem Schritt wird der Bot verbessert, das heißt die Fehler werden versucht zu beheben. Je nachdem was der Fehler des Bots war muss hier anders gearbeitet werden. Wenn er einen Intent den er eigentlich kennt aber die Formulierung des Benutzers so war das er diesen nicht erkannt hat, wird die Eingabe des Benutzers zu den Trainingsdaten hinzugefügt.

Falls jemand aber eine Frage stellt zu der es keinen Intent gibt und sich die Verantwortlichen denken es wäre ein guter Intent so wird der ganze Intent zu den Trainingsdaten hinzugefügt.

Trainieren

Trainieren ist der vierte Schritt des Leobot Conversation Cycles, in diesem Schritt werden die Trainingsdaten an den Bot übergeben und dieser Trainiert und wechselt auf das neu trainierte Model.

Und nun beginnt der Leobot Conversation Cycle wieder von vorne.

5.4.2 Umsetzung

Einerseits kann man sich die vergangenen Unterhaltungen ansehen, so wie die nlu.yml, stories.yml, rules.yml, domain.yml direkt im Monaco Editor bearbeiten und speichern.

Im Editor wurden auch Code Vorschläge benutzt um etwas an Tipparbeit sparen zu können.

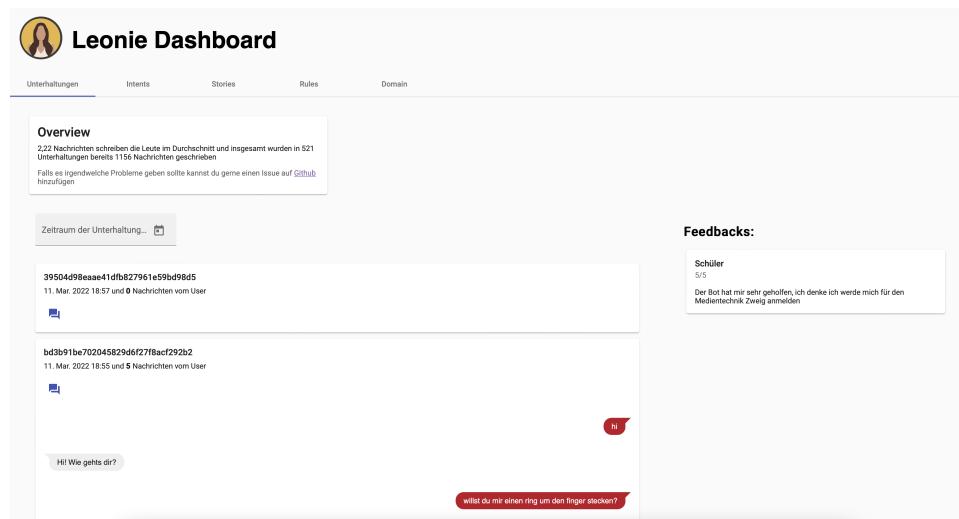


Figure 29: Dashboard

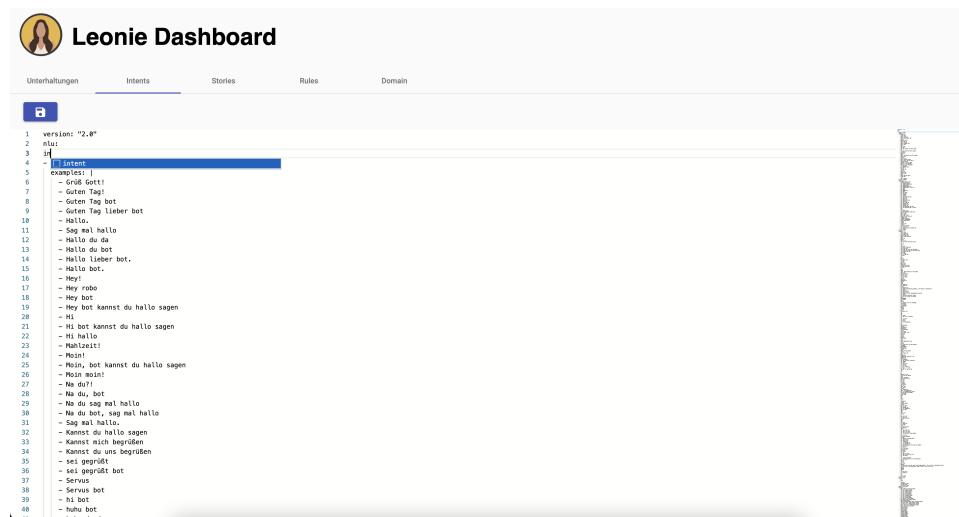


Figure 30: Vorschläge

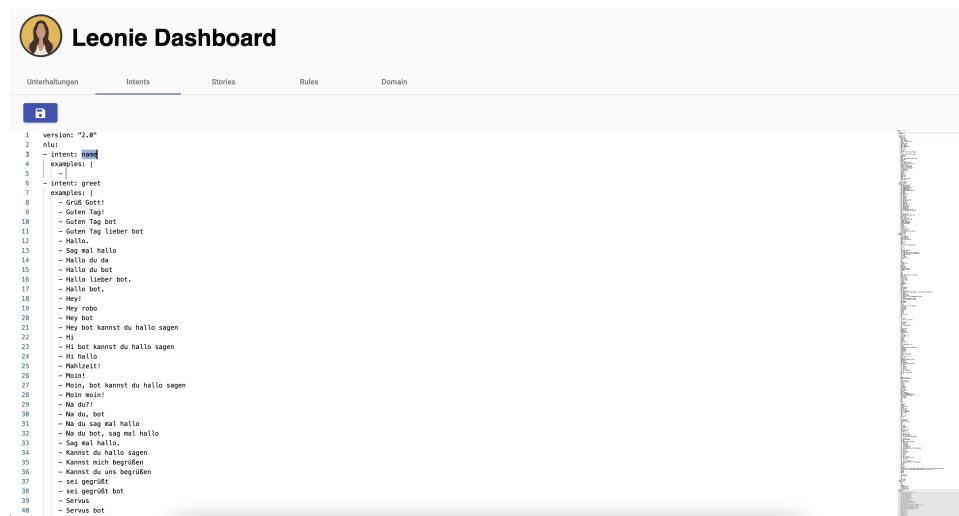


Figure 31: Vorschlag benutzt

5.5 Einbindung in Wordpress

5.5.1 Angular Elements

”Webkomponenten sind eine Gruppe von Web-Technologien, die es ermöglichen, benutzerdefinierte, wiederverwendbare HTML Elemente zu erstellen, deren Funktionalität gekapselt ist und damit vollständig getrennt von anderem Code.”[75]

Um unsre Chatkomponente auf die Wordpress Seite zu bringen, entschieden wir uns dazu die Komponente mithilfe von Angular Elements als WebKomponente zu exportieren um somit eine einfache einbindung auf die Schulhompage zu ermöglichen.

5.6 GH Actions

Die Praxische Arbeit besteht aus sehr vielen einzelnen Projekten, mit verschiedenen Funktionen, doch bei vielen der Github Projekte haben wir mithilfe von Github Actions unser Deployment automatisiert.

Backend

Unser Backend wird mithilfe von Github Actions automatisiert gebaut und das .jar file wird über SSH auf unsre VM geladen.

5.7 Schwierigkeiten

5.7.1 Backend

Beim Backend war eine sehr große Schwierigkeit das die Rasa X API nicht sehr gut dokumentiert ist und deshalb sehr vieles nur durch probieren und sehr tiefe Recherchen ermittelt werden konnte.

5.7.2 Frontend

Beim Frontend war die größte Schwierigkeit der export der Chatkomponente da Angular Materials nicht mit exportiert werden konnten, diese jedoch für die Feedback ansicht benutzt worden sind.

6 Evaluation

Bibliography

- [1] MonkeyLearn Inc., „What is Text Analysis? A Beginner’s Guide,” o.D., letzter Zugriff am 16.03.2022. Online verfügbar: <https://monkeylearn.com/text-analysis/>
- [2] Yulia Gavrilova, „Machine Learning Text Analysis,” 2020, letzter Zugriff am 16.03.2022. Online verfügbar: <https://serokell.io/blog/machine-learning-text-analysis>
- [3] IBM Cloud Education, „Natural Language Processing (NLP),” 2020, letzter Zugriff am 17.03.2022. Online verfügbar: <https://www.ibm.com/cloud/learn/natural-language-processing>
- [4] tutorialspoint, „NLP - Linguistic Resources,” 2020, letzter Zugriff am 17.03.2022. Online verfügbar: https://www.tutorialspoint.com/natural_language_processing/natural_language_processing_linguistic_resources.htm
- [5] Christopher Kipp, „Einstieg in Natural Language Processing – Teil 2: Preprocessing von Rohtext mit Python,” 2018, letzter Zugriff am 17.03.2022. Online verfügbar: <https://data-science-blog.com/blog/2018/10/18/einstieg-in-natural-language-processing-teil-2-preprocessing-von-rohtext-mit-python>
- [6] Susan Li, „Named Entity Recognition with NLTK and SpaCy,” 2018, letzter Zugriff am 24.03.2022. Online verfügbar: <https://towardsdatascience.com/named-entity-recognition-with-nltk-and-spacy-8c4a7d88e7da>
- [7] ICHI.PRO, „Einführung in NLP - Teil 2: Unterschied zwischen Lemmatisierung und Stemming,” 2018, letzter Zugriff am 24.03.2022. Online verfügbar: <https://ichi.pro/de/einfuhrung-in-nlp-teil-2-unterschied-zwischen-lemmatisierung-und-stemming-61064739575228>
- [8] Shivanee Jaiswal, „Natural Language Processing — Dependency Parsing,” 2021, letzter Zugriff am 26.03.2022. Online verfügbar: <https://towardsdatascience.com/natural-language-processing-dependency-parsing-cf094bbbe3f7>
- [9] Francesco Elia, „Constituency Parsing vs Dependency Parsing,” 2020, letzter Zugriff am 26.03.2022. Online verfügbar: <https://www.baeldung.com/cs/constituency-vs-dependency-parsing>
- [10] Ranks.nl, „German Stopwords,” o.D., letzter Zugriff am 26.03.2022. Online verfügbar: <https://www.ranks.nl/stopwords/german>
- [11] Dennis Rudolph, „Skalarprodukt berechnen: Vektoren, Formel und Definition,” 2020, letzter Zugriff am 26.03.2022. Online verfügbar: <https://www.gut-erklaert.de/mathematik/skalarprodukt-berechnen-vektoren-formel-definition.html>
- [12] Bruno Stecanella, „Understanding TF-IDF: A Simple Introduction,” 2019, letzter Zugriff am 27.03.2022. Online verfügbar: <https://monkeylearn.com/blog/what-is-tf-idf/>

- [13] Kinder Chen, „Introduction to Natural Language Processing — TF-IDF,” 2021, letzter Zugriff am 27.03.2022. Online verfügbar: <https://kinder-chen.medium.com/introduction-to-natural-language-processing-tf-idf-1507e907c19>
- [14] Koaning, „WhatLies,” o.D., letzter Zugriff am 27.03.2022. Online verfügbar: <https://koaning.github.io/whatlies/>
- [15] Wikipedia, „Java (Programmiersprache),” 2022, letzter Zugriff am 20.02.2022. Online verfügbar: [https://de.wikipedia.org/wiki/Java_\(Programmiersprache\)](https://de.wikipedia.org/wiki/Java_(Programmiersprache))
- [16] Tiobe, „Tiobe Index,” 2022, letzter Zugriff am 26.03.2022. Online verfügbar: <https://www.tiobe.com/tiobe-index/>
- [17] Daniel Rösch, „Rasa Core,” 2019, letzter Zugriff am 03.03.2022. Online verfügbar: <https://botfriends.de/blog/botwiki/rasa-core/>
- [18] S. S. Abhishek Singh, Karthik Ramasubramanian, *Introduction to Microsoft Bot, RASA, and Google Dialogflow*, 1. Aufl. United States of America: Rasa Technologies Inc, 2019.
- [19] Rasa Technologies GmbH, „Rasa Policies,” 2022, letzter Zugriff am 03.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/policies/>
- [20] ——, „Policy Priority,” 2022, letzter Zugriff am 03.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/policies#policy-priority>
- [21] ——, „TED Policy,” 2022, letzter Zugriff am 03.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/policies#ted-policy>
- [22] Daniel Rösch, „Rasa NLU,” 2019, letzter Zugriff am 03.03.2022. Online verfügbar: <https://botfriends.de/blog/botwiki/rasa-nlu/>
- [23] Tobias Wochinger, „Rasa NLU in Depth: Part 1 – Intent Classification,” 2019, letzter Zugriff am 03.03.2022. Online verfügbar: <https://rasa.com/blog/rasa-nlu-in-depth-part-1-intent-classification/>
- [24] Rasa Technologies GmbH, „How to choose a pipeline,” 2022, letzter Zugriff am 03.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/tuning-your-model/#how-to-choose-a-pipeline>
- [25] ——, „Tuning Your Model,” 2022, letzter Zugriff am 03.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/tuning-your-model/>
- [26] ——, „Component Lifecycle,” 2022, letzter Zugriff am 03.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/tuning-your-model/#component-lifecycle>
- [27] ——, „Component Lifecycle,” 2022, letzter Zugriff am 03.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/tuning-your-model/#component-lifecycle>
- [28] Bhavan Ravi, „NLP Behind Chatbots — Demystifying RasaNLU — 1 — Training,” 2018, letzter Zugriff am 03.03.2022. Online verfügbar: <https://medium.com/bhavaniravi/demystifying-rasa-nlu-1-training-91a08429c9fb>
- [29] Sociopath auf Stackoverflow, „Having a combination of pre trained and supervised embeddings in rasa nlu pipeline,” 2019, letzter Zugriff am 03.03.2022. Online verfügbar: <https://stackoverflow.com/questions/63683812/having-a-combination-of-pre-trained-and-supervised-embeddings-in-rasa-nlu-pipeli>

- [30] R. T. I. Justine Petraityte, *The Rasa Masterclass Ebook Pre Configured Pipelines*, 1. Aufl. United States of America: Rasa Technologies Inc, 2019.
- [31] Anran Jiao, „Components of Spacy Sklearn Pipeline,” 2020, letzter Zugriff am 07.03.2022. Online verfügbar: https://www.researchgate.net/figure/The-list-of-components-which-are-equal-to-pipeline-spacy-sklearn_fig2_340534832
- [32] Rasa Technologies GmbH, „SpacyNLP,” 2022, letzter Zugriff am 07.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/components#spacy-nlp>
- [33] ——, „MiteNLP,” 2022, letzter Zugriff am 07.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/components#mite-nlp>
- [34] Intellify Solutions, „Open Source Conversational AI Rasa,” o.D., letzter Zugriff am 07.03.2022. Online verfügbar: <https://intellifysolutions.com/blog/know-open-source-conversational-ai-rasa/>
- [35] Alan Nichol, „Supervised Word Vectors from Scratch in Rasa NLU,” 2018, letzter Zugriff am 03.03.2022. Online verfügbar: <https://medium.com/rasa-blog/supervised-word-vectors-from-scratch-in-rasa-nlu-6daf794efcd8>
- [36] Rasa Technologies GmbH, „How to Choose a Pipeline,” 2022, letzter Zugriff am 03.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/tuning-your-model/#how-to-choose-a-pipeline>
- [37] ——, „Sensible Starting Pipelines,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/tuning-your-model/#sensible-starting-pipelines>
- [38] ——, „Components,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/components/>
- [39] ——, „Rasa NLU Examples,” 2020, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasahq.github.io/rasa-nlu-examples/>
- [40] ——, „WhitespaceTokenizer,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/components/#whitespacetokenizer>
- [41] R. T. I. Justina Petraityte, *The Rasa Masterclass Ebook WhitespaceTokenizer*, 1. Aufl. United States of America: Rasa Technologies Inc, 2019.
- [42] J. P. Rasa. (Ep 4 - Rasa Masterclass) Training the NLU models: understanding pipeline components | Rasa 1.8.0. Youtube. Online verfügbar: <https://www.youtube.com/watch?v=ET1k9OrsfYQ&list=PL75e0qA87dlHQny7z43NduZHPo6qd-cRc&index=6>
- [43] R. T. I. Justina Petraityte, *The Rasa Masterclass Ebook RegexFeaturizer*, 1. Aufl. United States of America: Rasa Technologies Inc, 2019.
- [44] Rasa Technologies GmbH, „RegexFeaturizer,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/components/#regexfeaturizer>
- [45] ——, „CRFEntityExtractor,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/components/#crfentityextractor>
- [46] ——, „LexicalSyntacticFeaturizer,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/components/#lexical-syntactic-featurizer>

- [47] Rasa, Justina Petraityte, „(Ep 4 - Rasa Masterclass) Training the NLU models: understanding pipeline components | Rasa 1.8.0,” 2019, letzter Zugriff am 03.03.2022. Online verfügbar: <https://www.youtube.com/watch?v=YxMzz6NF6Zw&list=PL75e0qA87dlEjGAc9j9v3a5h1mxI2Z9fi&index=9>
- [48] Rasa Technologies GmbH, „CountVectorsFeaturizer,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/components/#countvectorsfeaturizer>
- [49] R. T. I. Justina Petraityte, *The Rasa Masterclass Ebook Count Vectors Featurizer*, 1. Aufl. United States of America: Rasa Technologies Inc, 2019.
- [50] Rasa Technologies GmbH, „EntitySynonymMapper,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/components/#entitysynonymmapper>
- [51] ——, „ResponseSelector,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/components/#responseselector>
- [52] ——, „Retrieval Intent,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/glossary#retrieval-intent>
- [53] ——, „Chitchat and FAQs,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/chitchat-faqs>
- [54] ——, „FallbackClassifier,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/components/#fallbackclassifier>
- [55] ——, „NLU Fallback,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/fallback-handoff/#nlu-fallback>
- [56] Alex Weidauer, „Conversational AI: Your Guide to Five Levels of AI Assistants in Enterprise,” 2018, letzter Zugriff am 03.03.2022. Online verfügbar: <https://rasa.com/blog/conversational-ai-your-guide-to-five-levels-of-ai-assistants-in-enterprise/>
- [57] Rasa, „(Ep 1 - Rasa Masterclass) Intro to conversational AI and Rasa | Rasa 1.8.0,” 2019, letzter Zugriff am 03.03.2022. Online verfügbar: <https://www.youtube.com/watch?v=-F6h43DRpcU&list=PL75e0qA87dlHQny7z43NduZHPo6qd-cRc&index=3&t=69>
- [58] R. T. I. Justine Petraityte, *The Rasa Masterclass Ebook*, 1. Aufl. United States of America: Rasa Technologies Inc, 2019.
- [59] Rasa Technologies GmbH, „Domain,” 2022, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/domain/>
- [60] ——, „Intents,” 2022, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/domain/#intents>
- [61] ——, „Responses,” 2022, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/domain/#responses>
- [62] ——, „Stories,” 2022, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/stories/>

- [63] ——, „Checkpoints und OR-Statements,” 2022, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/stories/#checkpoints-and-or-statements>
- [64] ——, „Checkpoints,” 2022, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/stories/#checkpoints>
- [65] ——, „Or Statements,” 2022, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/stories/#or-statements>
- [66] ——, „Rules,” 2022, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/rules>
- [67] ——, „Slots,” 2022, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/domain#slots>
- [68] ——, „Entities,” 2022, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/domain#entities>
- [69] Mady Mantha, „Entity Roles and Groups,” 2020, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/blog/introducing-entity-roles-and-groups/>
- [70] Rasa, „(Ep 8 - Rasa Masterclass)Implementing custom actions, forms and fallback | Rasa 1.8.0,” 2019, letzter Zugriff am 02.03.2022. Online verfügbar: https://www.youtube.com/watch?v=9POI7LiKH_8
- [71] Rasa Technologies GmbH, „Actions,” 2022, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/domain#actions>
- [72] ——, „Forms,” 2022, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/domain#forms>
- [73] ——, „Synonyms,” 2022, letzter Zugriff am 02.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/nlu-training-data/#synonyms>
- [74] ——, „Rasa Train,” 2022, letzter Zugriff am 30.03.2022. Online verfügbar: <https://rasa.com/docs/rasa/command-line-interface/#rasa-train>
- [75] Mozilla, „Web Components,” 2022, letzter Zugriff am 13.03.2022. Online verfügbar: https://developer.mozilla.org/de/docs/Web/Web_Components

List of Figures

1	Beispiel für die Beziehung zweier Wörter beim Dependency Parsing [8]	7
2	Darstellung des Dependency Parsing [9]	8
3	Darstellung des Constituency Parsing [9]	9
4	Skalarprodukt von zwei Vektoren [11]	10
5	Berechnung von TF-IDF [13]	12
6	Whatlies Illustration [14]	13
7	Java Logo[15]	15
8	Rasa Core Aufbau [18]	19
9	Component Lifecycle [26]	20
10	Component Lifecycle [28]	21
11	Pretrained oder Supervised Embeddings [23, 30]	23
12	Tokens durch einen WhitespaceTokenizer [42]	25
13	Regular Expression Beispiel [42]	26
14	Lookup Table Beispiel [42]	26
15	Funktionsweise vom CRFEntityExtractor [42]	27
16	Funktionsweise von einem Featurizer [47]	28
17	CountVectorsFeaturizer Beispiel [47]	28
18	5 Stufen von AI [57]	31
19	Rasa Init Konsolenausgabe	39
20	File Tree nach dem Initialisieren	39
21	Rasa Shell Ausgabe mit Debug Flag	41
22	Ausgabe beim Interactive Befehl	42
23	Systemarchitektur	43
24	Konzept Chatbot geschlossen	46
25	Konzept Chatbot geöffnet	46
26	Chatbot auf einer ganzen Seite	47
27	Chatbot	47
28	Leobot Conversation Cycle	48
29	Dashboard	50
30	Vorschläge	50
31	Vorschlag benutzt	50

List of Tables

Quellcodeverzeichnis

1	Beispiel für die Tokenization	4
2	Ausnahme für bestimmte Tokens	4
3	Beispiel für POS-Tagging	5
4	Beispiele für Entites	5
5	Stemming von deutschen Wörtern	6
6	Lemmatisierung von deutschen Wörtern	6
7	Bag of Words	10
8	Bag of n-grams	11
9	TF-IDF Beispiel	12
10	SpaCy Sklearn Pipeline	21
11	MITIE Sklearn Pipeline	22
12	Tensorflow Embedding Pipeline	22
13	Supervised Embedding Pipeline	23
14	Spacy Startpipeline	24
15	Default Pipeline	24
16	Unsere Pipeline	24
17	CountVectorsFeaturizer	28
18	Entity Synonym Mapper	29
19	Response Classifier	29
21	Fallback Classifier	31
22	Intents Beispiel	32
23	Responses Beispiel	32
24	Stories Beispiel	33
25	Checkpoints Beispiel	34
26	OR Beispiel	34
27	Rules Beispiel	35
28	Slots Beispiel	35
29	Entities in der Domain	35
30	Entities Beispiel	36
31	Entity Roles Beispiel	36
32	Entity Groups Beispiel	36
33	Entity Groups Beispiel 2	37
34	Forms Beispiel	37
35	Synonym Beispiel	38
36	Custom Action für die Ausgabe des aktuellen Datums	38
37	Custom Actions in Domain definiert	38
38	Befehl fürs Initialisieren	39
39	Rasa Train Befehle	40
40	Rasa Shell Befehle	40
41	Interaktive Trainingssession starten	41

Anhang