

**Philosophische** Fakultät III

Sprach- , Literatur- und Kulturwissenschaften

Institut für Information und Medien, Sprache und Kultur (I:IMSK)  
Lehrstuhl für [Medieninformatik | Informationswissenschaft]

Seminar: UR-Talking

Modul: INF-M 06.3

SS 2015

Leitung: Dr. Jürgen Reischer, Dr. Hanna Knäusl

UR-Talking – Implementierung des Chatbot-Prototypen „Elise“

Matthias Bräuer, Iris Figalist, Nadine Kaul

1632272, 1638283, 1650374

Medieninformatik, Informationswissenschaft

5./6. Semester B.A.

Marienstraße 8

93053 Regensburg

Tel.: 0176/32273316

Email: matthias.braeuer@stud.uni-regensburg.de

Abgegeben am 30.09.2015

Abbildungen

[Abbildung 1 – MySql „elise“ Datenbank 3](#_Toc284864361)

[Abbildung 2 – Beispieldatensatz aus „buzz\_words“ 3](#_Toc284864362)

[Abbildung 3 – Beispieldatensatz aus „location“ 3](#_Toc284864362)

[Abbildung 4 – Beispieldatensatz aus „unknown“ 3](#_Toc284864362)

[Abbildung 5 – Benutzerfrage 3](#_Toc284864362)

[Abbildung 6 – Stemming Benutzerfrage 3](#_Toc284864362)

[Abbildung 7 – Stemming Vergleichsfrage 3](#_Toc284864362)

[Abbildung 8 – Levenshtein- Distanz Schaubeispiel 3](#_Toc284864362)

# Einleitung

# Problemstellung

Für das Studienfach der Informationswissenschaft soll ein Prototyp für den Chatbot „Elise“ implementiert werden, der Fragen rund um das Thema Informationswissenschaft an der Universität Regensburg beantwortet. Als Kommunikationsoberfläche wird eine Anwendung benötigt, die Fragen eines Nutzers in Textform entgegennimmt, verarbeitet und eine passende Antwort zurückgibt. Da es nicht nur rein textbasierte Antworten gibt, muss der Chatbot in der Lage sein multimediale Inhalte sinnvoll wiederzugeben. Im Folgenden werden allgemeine Überlegungen erläutert, sowie der Aufbau und die Funktionalität des implementierten Prototypen erklärt.

# Lösungsansatz

## Allgemeine Architektur

Die allgemeine Architektur besteht, so wie es für interaktive Webanwendungen üblich ist, aus 2 Komponenten. Dem Client und einem Server. Der Client ist hierbei zuständig für das Anzeigen der graphischen Elemente des Chatbots, um dem Nutzer eine möglichst gute „User-Experience“ zu gewährleisten. Der Server kümmert sich um die Verarbeitung der gestellten Frage und das Auswählen, sowie das Zurückgeben einer passenden Antwort an den Client. Als geeignetes Framework wurden hierbei „Webforms“ von „asp.net“ gewählt.

## Frontend

Das Frontend der „Elise“ ist sehr simpel gehalten und konzentriert sich auf seine wesentliche Aufgabe, das Entgegennehmen einer Frage und das Anzeigen des Chatverlaufs und multimedialer Inhalte.

Um die genannten Voraussetzungen zu erfüllen wurde Chatfenster, inklusive Eingabefeld, implementiert. Wird der „ok“ button gedrückt, wird die eingegebene Frage per „jQuery-Ajax-Post-Request“ an den Server geschickt und als Listenelement an das Chatfenster angebunden. „Ajax“ bedeutet „*Asynchronous JavaScript and XML”* und arbeitet, wie der Name schon sagt, asynchron. Das heißt, dass solange auf eine Antwort vom Server gewartet wird, der Nutzer nicht warten muss, falls die Anfrage einmal länger dauert. Außerdem gibt es eine „success“-Methode, die aufgerufen wird sobald der Server erfolgreich antwortet. Für „Elise“ bedeutet das, dass nun auf der Clientseite eine Antwort auf die gestellte Frage vorhanden ist und diese nun auch als Listenelement an das Chatfenster angehängt werden kann.

Wie bereits zu Beginn des Kapitels erwähnt wurde, wird das Informationsdefizit des Nutzers multimedial behoben, was eine Unterscheidung der Antworten voraussetzt. Um zwischen verschiedenen Datentypen zu differenzieren, wird zusätzlich zur eigentlichen Antwort, dessen Typ mitgeschickt. Dieser besitzt 2 mögliche Ausprägungen, „Text“ und „Link“. Wird der Typ „Text“ detektiert, wird lediglich das Chatfenster aktualisiert. Ist das Ergebnis ein „Link“, wird dieser zum einen in einen „IFrame“ geladen und zum anderen im Chatfenster angehängt. Um späteres Anschauen zu gewährleisten und lästiges kopieren zu vermeiden wird dem „Link“ im Chatfenster zusätzlich ein Button hinzugefügt, der den Link in den „IFrame“ lädt, sobald er gedrückt wird.

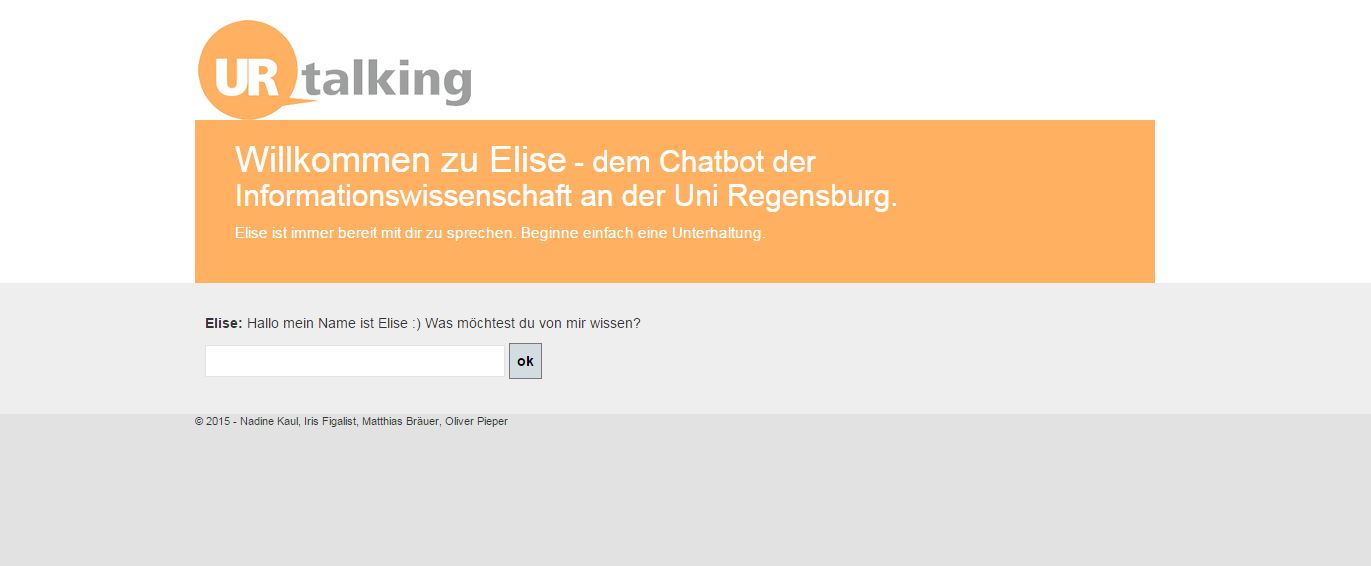
**

Abbildung 1 – „Elise“ Frontend

## Backend

### Überlegung und Probleme

Im Internet findet man viele Chatbot-Anwendungen. Sei es als informatives System, oder schlichtweg als Gesprächspartner für alltägliche Probleme. Jedes dieser Systeme steht seiner ganz eigenen Problematik gegenüber. Ein Thema, das als Basis eines jeden erweiterbaren Chatbots dient, teilen sich jedoch alle: „Natural Language Processing“!

„Natural Language Processing“(Abk. „nlp“) beschreibt die Analyse eines Textes in verschiedenen Art und Weisen. Zu unterscheiden sind dabei 2 gegensätzliche, sich aber nicht ausschließende Ansätze. Der „Semantische“ und der „Syntaktische“. Bei der „semantischen“ Textanalyse wird versucht den Sinn des Textes zu erschließen. So können gleiche Wörter in unterschiedlicher Reihenfolge, auch unterschiedliche Bedeutung haben. Die „syntaktische“ Textanalyse betrachtet allein den eigentlichen Text, als Ausprägung seiner einzelnen Zeichen. So haben gleiche Wörter in der Regel gleiche Bedeutung in unterschiedlichen Kontexten.

Obwohl die „semantische“ Analyse deutlich aufwendiger ist, ist sie jedoch in den meisten Fällen der „syntaktischen“ Analyse vorzuziehen, da sie auf Dauer bessere Ergebnisse liefert.

Um „Elise“ die Fähigkeit der „semantischen“ Analyse zu geben, wurden diverse Bibliotheken implementiert und getestet. Die Voraussetzungen, die diese Bibliothekfen erfüllen mussten sind zum einen die Adaption an „C#“ und zum anderen die Unterstützung von deutsch.

Diese Einschränkungen führten zu 2 Bibliotheken, die in den folgenden 2 Kapiteln erläutert werden.

#### StanfordNLP

„StanfordNLP“ ist eine „Natural-Language-Processing-Bibliothek“ der Universität Stanford, die ursprünglich in Java implementiert wurde, jedoch in diverse andere, unter anderem „C#“, Programmiersprachen übersetzt wurde.

„StanfordNLP“ bietet unter anderem einen „Parser“, einen „POS-Tagger“, sowie einen „NER“, für die deutsche Sprache an.

* + - * 1. POS-Tagger

Der „POS-Tagger“, auch „Part-Of-Speech-Tagger“ genannt, analysiert einen Text und tagged die einzelnen Wörter, unter Berücksichtigung dessen Definition und der im Kontext angrenzenden Wörter.

* + - * 1. Parser

Der Parser bietet 3 verschieden Ausgaben. Einen „POS-tagged“, einen „grammatikalisch strukturierten“ und einen „Typabhängigkeiten-repräsentierenden“ Text.

Bsps:

* „POS-tagged“:

The/DT strongest/JJS rain/NN ever/RB recorded/VBN in/IN India/NNP

shut/VBD down/RP the/DT financial/JJ hub/NN of/IN Mumbai/NNP ,/,

snapped/VBD communication/NN lines/NNS ,/, closed/VBD airports/NNS

and/CC forced/VBD thousands/NNS of/IN people/NNS to/TO sleep/VB in/IN

their/PRP$ offices/NNS or/CC walk/VB home/NN during/IN the/DT night/NN

,/, officials/NNS said/VBD today/NN ./.

* „Grammatikalisch strukturiert“:

(ROOT

(S

(S

(NP

(NP (DT The) (JJS strongest) (NN rain))

(VP

(ADVP (RB ever))

(VBN recorded)

(PP (IN in)

(NP (NNP India)))))

(VP

(VP (VBD shut)

(PRT (RP down))

(NP

(NP (DT the) (JJ financial) (NN hub))

(PP (IN of)

(NP (NNP Mumbai)))))

(, ,)

(VP (VBD snapped)

(NP (NN communication) (NNS lines)))

(, ,)

(VP (VBD closed)

(NP (NNS airports)))

(CC and)

(VP (VBD forced)

(NP

(NP (NNS thousands))

(PP (IN of)

(NP (NNS people))))

(S

(VP (TO to)

(VP

(VP (VB sleep)

(PP (IN in)

(NP (PRP$ their) (NNS offices))))

(CC or)

(VP (VB walk)

(NP (NN home))

(PP (IN during)

(NP (DT the) (NN night))))))))))

(, ,)

(NP (NNS officials))

(VP (VBD said)

(NP-TMP (NN today)))

(. .)))

* „Typabhängigkeiten-represäntierend“:

det(rain-3, The-1)

amod(rain-3, strongest-2)

nsubj(shut-8, rain-3)

nsubj(snapped-16, rain-3)

nsubj(closed-20, rain-3)

nsubj(forced-23, rain-3)

advmod(recorded-5, ever-4)

partmod(rain-3, recorded-5)

prep\_in(recorded-5, India-7)

ccomp(said-40, shut-8)

prt(shut-8, down-9)

det(hub-12, the-10)

amod(hub-12, financial-11)

dobj(shut-8, hub-12)

prep\_of(hub-12, Mumbai-14)

conj\_and(shut-8, snapped-16)

ccomp(said-40, snapped-16)

nn(lines-18, communication-17)

dobj(snapped-16, lines-18)

conj\_and(shut-8, closed-20)

ccomp(said-40, closed-20)

dobj(closed-20, airports-21)

conj\_and(shut-8, forced-23)

ccomp(said-40, forced-23)

dobj(forced-23, thousands-24)

prep\_of(thousands-24, people-26)

aux(sleep-28, to-27)

xcomp(forced-23, sleep-28)

poss(offices-31, their-30)

prep\_in(sleep-28, offices-31)

xcomp(forced-23, walk-33)

conj\_or(sleep-28, walk-33)

dobj(walk-33, home-34)

det(night-37, the-36)

prep\_during(walk-33, night-37)

nsubj(said-40, officials-39)

root(ROOT-0, said-40)

tmod(said-40, today-41)

Diese Ausgaben bieten folgende Möglichkeiten. Z.B. kann man aus der „gramtikalisch-

strukturierten“ Ausgabe alle Nomen und Verben auslesen, die maßgeblich Entscheidend für die Bedeutung des Satzes sind. Die Typabhängigkeiten bieten die Grundlage für eine Gewichtung auftretender Wortpaare und darausfolgender Bestimmung ähnlicher Sätze bzw. passender Antworten auf Fragen. Treten z.B. ein Wort und dessen Abhängigkeit in der, zu der Frage verglichenen Antwort auf, ist die Wahrscheinlichkeit auf eine passende Antwort gestoßen zu sein höher, als wenn zwei unabhängige Wörter in der potentiellen Antwort erkannt wurden.

* + - * 1. NER

Ein „NER“, auch „Named-Entity-Recognizer“, analysiert einen Text und tagged Nomen mit der semantischen Ausprägung dessen. Je nach Vollständigkeit des „NERs“, werden die Wörter/Phrasen, „Albert Einstein“, „Microsoft“, „1000 Euro“ wie folgt getagged.

* „Albert Einstein“ : „Person“
* „Microsoft“ : „Organization“
* „1000 Euro“ : „Money“

Damit lässt sich in den meisten Fällen die Anzahl der potentiellen Antworten deutlich verringern, da Fragen immer den Antworttypen determinieren. So lassen sich durch die „Named-Entities“ auch Fragen mit nicht eindeutig definierendem Fragewort(z.B. „was“) einem Antworttypen zuordnen und somit alle Antworten die dem geforderten Typen nicht entsprechen ignorieren.

Bsp.: „Was ist der Euro?”, erwartet eine Antwort vom Typ “Money”. „Was ist Microsoft?“, erwartet eine Antwort vom Typ “Organization”. Antworttyp wird durch das den Typ des Nomens (“Euro”, “Microsoft”) eindeutig bestimmt.

#### SharpNLP

“SharpNLP” ist die, in “C#” übersetzte Bibliothek von OpenNLP, eine, in Java verfasste Bibliothek von Apache. Sie bietet im Grunde die Gleichen Funktionen wie „StanfordNLP“.

#### Probleme

Implementiert und getestet wurden beide Bibliotheken. Leider treten bei beiden einige Probleme auf, die sowohl an der schlechten Unterstützung von deutsch, als auch der mageren Implementierung in „C#“ liegen.

|  |  |  |
| --- | --- | --- |
| Bibliothek | SharpNLP | StanfordNLP |
| Models | Nur englisch vorhanden | Teilweise vorhanden |
| POS-Tagger | Nutzlos ohne deutsche Models | Funktioniert |
| Parser | Nutzlos ohne deutsche Models - | Stürzt in „Webforms“ ab |
| NER | Nutzlos ohne deutsche Models - | Erkennt nur sehr selten „Named-Entities“ |

Model-Files bieten die Grundlage für „NLP“. Sie beinhalten große Datensätze und Regeln, wie der zu analysierende Text zu behandeln ist. „SharpNLP“ erwarten .nbin Dateien und stellt leider nur englische Versionen zur Verfügung. Da die Entwicklung von „SharpNLP“ auch seit ein paar Jahren gleich Null ist, verwundert es nicht, dass der „Model-Converter“, der .bin Models von „OpenNLP“ in .nbin Models konvertieren soll nicht mehr unterstützt wird.

StanfordNLP stellt ein paar deutsche Models bereit, jedoch wurden beim Testen nicht die gewünschten Ergebnisse erzielt.

#### Fazit

Als Fazit kann man sagen, dass die Kombination von „C#“ und „deutsch“ in Bezug auf „NLP“ eher eine schlechte Wahl ist. Die Programmiersprache „Java“ bietet wohl mit das beste Angebot für „NLP-Bibliotheken“. Aus diesem Grund wurde für „Elise“ eine syntaktische Analyse in Verbindung mit einer simulierten semantischen Analyse implementiert.

### Architektur

Die allgemeine Architektur der Serverkomponente beinhaltet folgende Komponenten.

* „ValuesController“
* „SimpleString“-Bibliothek
* „SynonymsDictionary“-Bibliothek
* „AnswerTypeDetector“-Bibliothek
* „Antwortgenerierung“

### „ValuesController“

Der „ValuesController“ implementiert die REST-Schnittstelle zwischen Server und Client. Er nimmt Anfragen des Nutzers entgegen und sendet die ausgewählte Antwort wieder zurück.

### „SimpleString“-Bibliothek

„SimpleString“ ist eine Bibliothek, die aus der statischen Klasse „String“ besteht. Ihre Aufgabe besteht darin, der „SynonymsDictionary-“ und „AnswerTypeDetector“-Bibliothek die Aufgabe für das Überprüfen, ob eine Phrase für sich alleine in einem Satz steht und für das Ersetzen von einer Phrase durch eine andere in einem Satz, abzunehmen.

### „SynonymsDictionary“-Bibliothek

Die „SynonymsDictionary“-Bibliothek ist, wie der Name schon sagt, eine Bibliothek die Synonyme verwaltet. Das Herz der Bibliothek bietet die Klasse „Dictionary.cs“, die entweder einen ganzen Satz „synonymisiert“ und zurückgibt, oder einfach nach einem Synonym zu einem übergebenen Wort sucht und dieses gegebenenfalls zurückgibt. Sie bietet also die Möglichkeit Frage und potentielle Antwort auf einen einheitlichen Wortschatz zu bringen und somit z.B. demographische oder regionale Unterschiede in der verbalen Ausdrucksweise abzufangen bzw. zu mindern, um eine erfolgreichere Anwortbestimmung zu garantieren.

Die nötigen Informationen zur „Synonymisierung“ erhält das Wörterbuch von einer geladenen Wörterbuchdatei(.bin). Vorteil dieser Vorgehensweise gegenüber einer „hart-gecodeten“ Version ist, dass individuelle Wörterbücher geladen werden können.

Die Klasse „ModelHandler.cs“ bietet sogar die Möglichkeit selbst ein .bin Modelfile aus einem .json File zu erstellen, das wie folgt aussieht.

[

{„Synonyms“: [„hi“, ,hey,“, „servus“], „word“: „Hallo“},

{„Synonyms“: [„bye“, ,ciao,“, „tschau“], „word“: „Tschüß“},

]

Dieser “JSON-String” wird beim einlesen in die „ModelHandler.cs“-Klasse in eine Liste von „SynonymObject.cs“ umgewandelt, die wiederum zu einem .bin File serialisiert wird.

### „AnswerTypeDetector“-Bibliothek

Die Bibliothek „AnswerTypeDetector“ versucht einer gegebenen Frage, einen Antworttypen zuzuweisen und somit zum einen die Anzahl der potentiell richtigen Antworten zu verringern und zum anderen eine erfolgreichere Antwortbestimmung zu gatantieren.

Die Bibliothek bietet die Möglichkeit bis zu 5 verschiedene Modeltypen zu laden.

* „Person“
* „Location“
* „Number“
* „Date“
* „Period“

„AnswerTypeDetector“ bietet wie „SynonymsDictionary“, die Möglichkeit über ein „JSON-File“, das wie folgt aussieht, ein eigenes Model(.dat) zu erzeugen.

[“Wann”, “Termin”, “Datum”, “Zeitpunkt”]

Die Klasse “AnswerTypeDetect.cs” implementiert die Methode „List<AnswerType> GetAllAnswerTyps(string question)“ und ermöglicht somit das analysieren der Frage auf einen Antworttypen. Zusätzlich lassen sich die Modeldaten der „AnswerTypeDetector“-Bibliothek durch das im Konstruktor übergebene „SynonymsDictionary“ „synonymisieren“ um möglicherweise auftretende Fehler durch das „SynonymDictionary“ abzufangen.

Bsp: Die gestellte Frage ist „Wo ist die Mensa?“. Würde das „SynonymsDictionary“ das Wort „Wo“ durch das Wort „Standort“ ersetzen, würde „AnswerTypeDetector“ den Antworttyp „Location“ nicht erkennen, falls das Model nur die Daten [„Wo“, „Ort“, „Woher“] enthalten würde. Wir aber auch „AnswerTypeDetector“ mit dem gleichen „SynonymsDictionary“ „synonymisiert“, werden die Modeldate zu [„Standort“, „Ort“, „Woher“] und der Antworttyp „Location“ würde wieder erkannt werden.

### Fazit

Die implementierten Bibliotheken „SimpleAnswerDetector“, „SynonymsDictionary“ und “SimpleString” versuchen im Zusammenspiel das Defizit der nicht vorhandenen semantischen Analyse aufzufangen und die bestmöglichen Grundvorraussetzungen für die syntaktische Antwortanalyse zu schaffen.

### Antwortgenerierung

#### Einleitung

Im Projektteil „Antwortgenerierung“ des Projektes "EliseChatbot", wird sich mit der Aufgabe des Findens einer Antwort in einer MySql Datenbank befasst.

Im Fokus steht zunächst, eine korrekte Antwort auf eine konkrete Frage zu finden.

Um dies zu erreichen, wird im Rahmen des Projektes, ein Datenbank Entwurf angefertigt, der es zulässt Fragen in Tabellen zu untergliedern.

Zudem wird das Stammformreduktions- Verfahren, eine einfache Sortierung des Frage- und Antwortsatzes nach dem Alphabet und die Levenshtein-Distanz verwendet.

#### Datenbank/Datenbankabfrage

Zunächst besteht das Ziel darin, den Suchbereich so gering wie möglich zu halten.

Hierzu wird eine Datenbankstruktur erzeugt, in welcher Fragen nach ihrer Semantik untergliedert werden können.

Dadurch wird die Suche in spezifischen Bereichen ermöglicht und somit eine Einschränkung des Suchbereiches erzeugt.

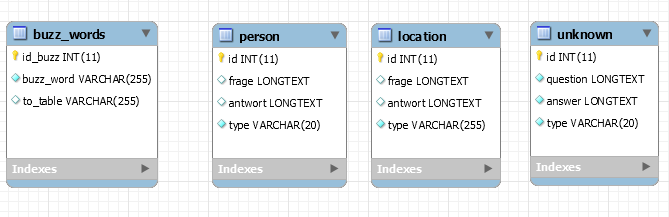


Abbildung 1 – MySql „elise“ Datenbank

#### Tabelle „buzz\_words“

Die Tabelle „buzz\_words“ diente anfänglich der Kategorisierung der Frage. Die Spalte „buzz\_word“ beinhaltet Wörter, welche in der Frage enthalten sein könnten. Aus der zugeordneten Spalte „to\_table“ kann nun ausgelesen werden, um welchen Bereich es sich handeln kann. Beispieltabellen für Bereiche sind hier „person“, „location“ und „unknown“.



Abbildung 2 – Beispieldatensatz aus „buzz\_words“

Wird also die Frage „Wo befindet sich die Mensa?“ gestellt, kann das Wort „Wo“ aus dem Fragesatz extrahiert werden. Nun wird in der Spalte „buzz\_word“ das Wort „wo“ gefunden und die zugehörige Zuweisung in der Spalte „to\_table“ die Zuweisung „location“. Somit werden mögliche Antworten nur noch in der Tabelle „location“ gesucht.

Wird kein „buzz\_word“ gefunden, werden alle Tabellen nach einer möglichen Antwort durchsucht.

Im Späteren Verlauf, wurde die Tabelle durch Modeltypen ersetzt, die als Basis den NLP Algorithmus verwenden. Die Restliche Verarbeitung bleibt bestehen.

#### Kategorien - Tabellen am Beispiel „location“

Die Tabelle „location“ repräsentiert hier eine der möglichen Kategorien.

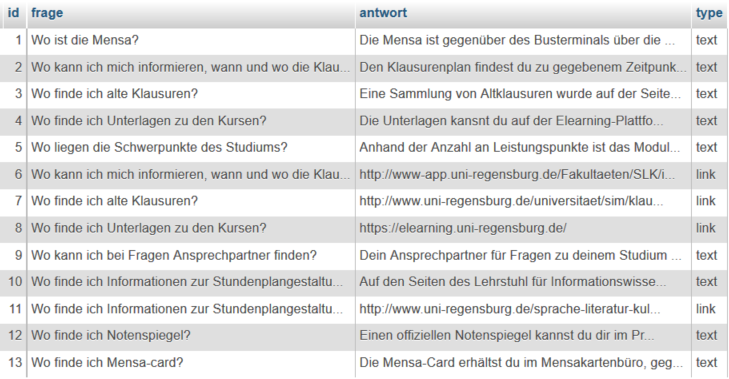


Abbildung 3 – Beispieldatensatz aus „location“

Sie beinhaltet die Spalten „id“, die als Primärschlüssel fungiert, „frage“, welche für den Abgleich der gestellten Frage dient, „antwort“, diese liefert die Antwort zur gefundenen Frage und die Spalte „type“, welche Aufschluss über die Art der Antwort gibt.

#### Typen Definition

Die Typen Definition in den Kategorien - Tabellen geben Aufschluss über die Art der Antwort. Implementierte Typen sind zum derzeitigen Entwicklungsstand, „link“ und „text“. Daraus ergibt sich, dass es für eine Frage zwei typabhängige Antworten geben kann.

Aus Abb. 3 geht hervor, dass für die Frage „Wo finde ich alte Klausuren?“ sowohl für die Antwort mit der „id = 3, type=‘text‘“ zutrifft, als auch für die Antwort mit der „id = 7, type=‘link‘“. Sie unterscheiden sich im Typ der Antwort.

#### Gesonderte Kategorien - Tabelle „unknown“

In der Tabelle „unknown“ werden alle unsortierten oder nicht klar zuordenbaren Fragen gespeichert. In dieser wird gesucht sobald keine eindeutige Kategorisierung durch einen Modeltyp festgelegt werden kann.

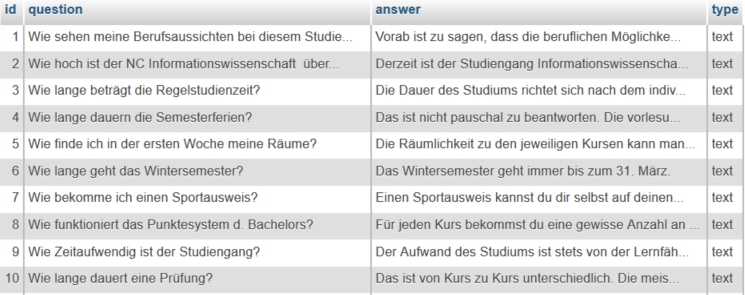


Abbildung 4 – Beispieldatensatz aus „unknown“

Jene besitzt die gleiche Funktionalität wie alle anderen Kategorien - Tabellen.

(Fragen von: „Gruppe 6: Erstellung Facebook/whatsapp- email Korpus“ erhalten).

#### Antwortgenerierung

Nachdem die Typisierung der Frage erläutert wurde, und so mit einer hohen Wahrscheinlichkeit die korrekte Tabelle, wird anschließend eine mögliche Antwort innerhalb dieser Tabelle gesucht.

Hierzu werden die vom jeweiligen Benutzer gestellten Fragen (i.F. Benutzerfrage) mit den in der Tabelle gespeicherten Vergleichsfragen mittels der Levenshtein-Distanz verglichen.

Das Ergebnis liefert die beste Übereinstimmung von Benutzerfrage zur Vergleichsfrage innerhalb der Tabelle. Die zugehörige Antwort der Vergleichsfrage liefert somit die Antwort.

#### „Stemming“ und „Tokenizing“

Zu Beginn des „Stemming-“ und „Tokenizing-“ Prozesses wird der Text in ihre einzelnen Wörter zerlegt und als Array gespeichert. Anschließend wird von Lucene.net eine Liste mit deutschen Stoppwörtern geladen. Das Array der Eingabe wird mit dieser Liste verglichen und es wird eine gefilterte Liste der Eingabe zurückgeliefert. Im nächsten Schritt wird die Eingabe bereinigt. Alle Wörter werden auf Kleinschreibung umgestellt und sämtliche Satzzeichen werden entfernt. Im letzten Schritt werden alle Wörter des Arrays gestemmt und schließlich wird die verarbeitete Eingabe als String zurückgeliefert. Als Stemmer wird der „GermanStemmer“ von Lucene.net verwendet.

#### Stammformreduktion

Da die Levenshtein- Distanz dem Vergleich der Benutzerfrage zu Vergleichsfrage dient, ist es notwendig, Benutzer- und Vergleichsfrage einander anzunähern. Dies kann unter anderem durch die Stammformreduktion erreicht werden.

Wie in Abb. 5 ersichtlich ist, gibt der Benutzer die Frage „wo ist die mensa?“ ein.

In Abb. 6/7 ist die Entfernung der Stoppwörter in der Benutzerfrage, als auch der Vergleichsfrage durch die Funktion „StemmerAndTokenizer.stemAndTokenize(sentence)“ zu sehen.

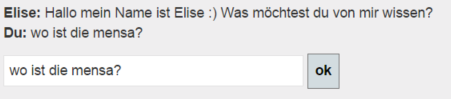


Abbildung 5 - Benutzerfrage

C:\Users\naddl1337\AppData\Local\Microsoft\Windows\INetCache\Content.Word\stemmer_userfrage_wo_ist_die_mensa.png

Abbildung 6- Stemming Benutzerfrage

C:\Users\naddl1337\AppData\Local\Microsoft\Windows\INetCache\Content.Word\stemmer_vergeleichsfrage_wo_ist_die_mensa.png

Abbildung 7- Stemming Vergleichsfrage

#### Sortierung

Da der Benutzer keiner vorherigen Einschränkung in der Fragestellung unterliegt, ist es möglich, dass dieselben Wörter bei der Benutzerfrage verwendet werden, die auch in der Vergleichsfrage zu finden sind, aber die Satzstellung in der Benutzerfrage verändert wird. Am Beispielsatz „Wo ist die Mensa?“, ist eine weitere Möglichkeit die Frage zu stellen , gegeben mit „Die Mensa ist wo?“.

Der Vergleich der Levenshtein- Distanz bringt hier ein schlechteres Ergebnis als zuvor die Frage „Wo ist die Mensa?“ die als Ergebnis die völlige Übereinstimmung als Ergebnis liefert.

Um die Veränderung der Satzstellung aus dem Vergleich der Levenshtein- Distanz auszuschließen, wird eine einfache Sortierung der Benutzerfrage, sowie der Vergleichsfrage nach dem Alphabet vorgenommen.

## NLP ReplaceSynonyms

Eine weitere Möglichkeit der Annäherung der Benutzer- und Vergleichsfrage, ergibt sich aus der Entfernung und Gleichsetzung der vorhandenen Synonyme.

Der Erfolg der Funktionen „nlp.ReplaceBySynonyms(question\_user);“ und „nlp.ReplaceBySynonyms(question\_user);“ hängt von der dahinterliegenden Sammlung an Synonymen ab, welche beim Programmstart aufgerufen wird. Am Beispielsatz der Benutzerfrage „Wie hoch ist der Notenschnitt bei Informationswissenschaften“ und der Vergleichsfrage „Wie hoch ist der NC bei Informationswissenschaften“ sind gefundene Synonyme „Notenschnitt“ und „NC“.

#### Levenshtein-Algorithmus

Nachdem die Benutzerfrage mit der Stammformreduktion, der Sortierung und dem NLP Replacesynonym- algorithm angenähert wurde, wird im Folgenden der Levenshtein- Algorithmus verwendet.

Dieser wird auch als „Editierdistanz“ bezeichnet, da die minimale Anzahl an Einfüge- , Lösch- und Ersetz-Operationen berechnet wird, die notwendig sind, um zwei Zeichenketten anzugleichen.

Abb. 8 zeigt Beispielhaft die Berechnung der Distanz der zwei Zeichenketten „meilenstein“ und „levensthein“.

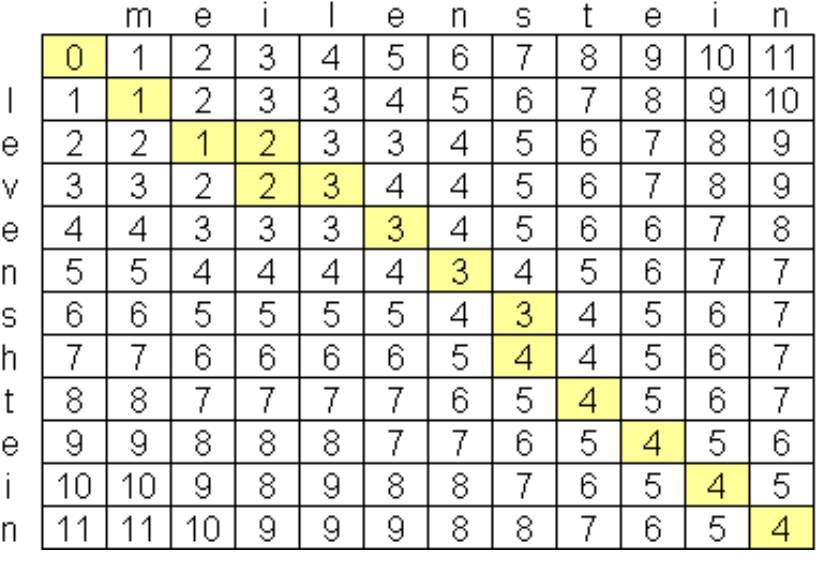


Abbildung 8- Levenshtein- Distanz Schaubeispiel

Die rechte untere Ecke liefert das Ergebnis „4“ als Anzahl der benötigten Operationen um die beiden Zeichenketten anzugleichen.

Die in dem Projekt „EliseChatbot“ verwendete API des Levenshtein- Algorithmus

verwendet wie in Abb 8 eine Matrix und gibt für die totale Gleichheit der Benutzer- und Vergleichsfrage einen Wert von „0“ zurück. In diesem Fall ist die Antwort auf die Frage gefunden.

Die größte mögliche Anzahl an Operationen und somit Veränderungen um die beiden Fragen einander anzunähern, liefert einen Rückgabewert von „100“.

Ergebnisse zwischen 0 und 40 sind bei der Anzahl von 65 erfassten Fragen immer korrekt.

Bei Ergebnissen, welche größer als 40 sind, wird die zuvor gefundene Tabelle verworfen und anschließend über alle Tabellen und deren Antworten verglichen, da es sich um eine Falschantwort handeln kann.

Für Ergebnisse die größer 65 sind, wird der Algorithmus abgebrochen, da keine valide Antwort zu finden ist.

# Fazit

Nach einigen Testreihen für bestimmte Benutzerfragen hat sich das Problem ergeben, dass der Levensthein Algorithmus bei einer sehr umfangreichen Frage nicht mehr die gewünschten Ergebnisse erzielt. Sind die Benutzerfragen oder auch die Vergleichsfragen mit Adverben verdeutlicht, so werden diese nicht herausgefiltert und verfälschen das Ergebnis insofern, dass das Ergebnis einen Wert zwischen 60 und 80 annimmt und somit die gefundene Antwort wieder in den Bereich der falschen Antworten fällt.

Literaturverzeichnis

* Levenshtein- Algorithmus API.

Retrieved from <http://www.codeproject.com/script/Articles/ViewDownloads.aspx?aid=13525>. [26.03.2012]

* Abbildung 8. Retrieved from <http://www.levenshtein.de/>
* http://www.nlpado.de/~sebastian/software/ner\_german.shtml
* https://sergeytihon.wordpress.com/tag/stanford-nlp/ aufgerufen

[01.09.2015]

* https://www.lt.informatik.tu-darmstadt.de/de/data/german-named-entity-recognition/

[02.09.2015]

* https://sites.google.com/site/germeval2014ner/

[02.09.2015]

* http://www.uni-weimar.de/medien/webis/teaching/theses/beyer\_2010.pdf

[02.09.2015]

* http://www.rhyous.com/2014/10/20/splitting-sentences-in-c-using-stanford-nlp/

[06.09.2015]

* http://code-bude.net/2013/07/22/csharp-api-fuer-den-wortschatz-leipzig-thesaurus-webservic/

[07.09.2015]

* http://seanglover.com/blog/2012/08/extracting-noun-phrases-with-contextual-relevance-in-net-using-opennlp/

[13.09.2015]

* http://nlp.stanford.edu/software/lex-parser.shtml

[29.09.2915]

* http://nlp.stanford.edu/software/index.shtml
* [29.09.2915]
* https://de.wikipedia.org/wiki/Ajax\_(Programmierung)

[29.09.2915]