Automatische Übersetzung von natürlicher Sprache zu Computersprache mit maschinellem Lernen

Projektarbeit

im Fachgebiet Informatik

vorgelegt von:	Lyndsey Bonelli
	Micha Merkli
	Stefan Bösch
Studienbereich:	Informatik
Betreuer:	Prof. Dr. Kurt Stockinger
	Dr. Mark Cieliebak

Erklärung betreffend des selbstständigen Verfassen einer Projektarbeit

Mit der Abgabe dieser Projektarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Projektarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

Ort, Datum:	Unterschriften:
.....	.....
	.....

Das Original dieses Formulars ist bei der ZHAW-Version aller abgegebenen Projektarbeiten zu Beginn der Dokumentation nach dem Abstract bzw. dem Management Summary mit Original-Unterschriften und -Datum (keine Kopie) einzufügen.

Lyndsey Bonelli, Micha Merkli und Stefan Bösch

Zusammenfassung

Es existieren mehrere Implementationen, welche für die Übersetzung von natürlicher Sprache zu Computersprache mittels maschinellem Lernen verwendet werden. Ein Schritt, was vor dem Trainieren und der Übersetzung nötig ist, ist das Preprocessing von den Eingabedaten. Da dieser Schritt die Präzision der Übersetzung beeinflusst, ist ein möglichst genaues Preprocessing erstrebenswert. Um die Genauigkeit eines Preprocessing zu bestimmen, wird im Rahmen dieser Projektarbeit ein Konzept entwickelt und exemplarisch für die Bewertung des Entity-Linking umgesetzt. Die Bewertung erfolgt anhand der Entitäten, die im Preprocessing extrahiert wurden, und der erwarteten Entitäten (Ground-Truth). Diese Entitäten werden anhand von verschiedenen Kriterien verglichen, wie die Genauigkeit und die Synonymität der Labels sowie die Genauigkeit des Resultates einer SPARQL-Abfrage, welche auf Basis der extrahierten Entitäten ausgeführt wird. Zudem werden Entitäten erkannt, welche schwierig oder nicht anhand der natürlich-sprachigen Abfrage identifiziert und im Preprocessing extrahiert werden. Schliesslich wird erkannt, welche SPARQL-Abfrage- oder Entitättypen besonders schwierig zu identifizieren und zu extrahieren sind. Eine exemplarische Bewertung wurde anhand der Ergebnisse des Preprocessing der Bachelorarbeit "Automatische Übersetzung von natürlicher Sprache nach SPARQL mit neuronalen Netzen" ermittelt, welche als Grundlage für diese Projektarbeit dient und die Daten der SQA2018-Challenge verwendet. Anhand dieser Bewertung ist erkennbar, dass je mehr Hilfswörter, wie Partikel oder Pronomen, eine Entität enthält, desto schwieriger ist es, sie zu erkennen. Dies betrifft vor allem zum Beispiel Lieder- oder Filmtitel. Ausserdem sind vor allem Entitäten der Triple-Abfragen schwierig zu extrahieren. Dies liegt nicht nur an der Entität selbst, sondern auch daran, dass in solchen Fragen meist mehrere Entitäten verwendet werden. Für eine aussagekräftigere Bewertung von synonymen Entitäten müssten Properties und Ontologies miteinbezogen werden. Zusammen mit Properties und Ontologies muss geklärt werden, ob eine SPARQL-Abfrage gemäss den extrahierten Werten die korrekten Resultate liefert.

Lyndsey Bonelli, Micha Merkli und Stefan Bösch

Vorwort

Wir möchten uns an dieser Stelle bei unseren Betreuern Prof. Dr. Kurt Stockinger, Prof. Dr. Mark Cieliebak und bei Jan Milan Deriu für ihre Unterstützung und Rat bedanken. Zudem möchten wir uns bei Nicolas Hoferer und bei Sebastian Drozdz für ihre Unterstützung bedanken. Des Weiterem bedanken wir uns bei der Zürcher Hochschule für Angewandte Wissenschaften für die Bereitstellung der benötigten Rechenressourcen und bei Remo Maurer für den technischen Support.

Inhaltsverzeichnis

1. Einleitung

SPARQL und andere Abfragesprachen haben ein langes Bestehen in der Informationstechnologie und dienen als essentielles Werkzeug in der Abfrage oder in der Sammlung von Daten aus Speicherstrukturen. Die starke Orientierung an der Speicherstruktur ist auch heutzutage noch der Grund, dass die Abfragesprachen nicht für den Endbenutzer ausgelegt sind.

Seit geraumer Zeit wird mit maschinellen Übersetzungstechnologien daran gearbeitet, Abfragen auch in natürlicher Sprache zu ermöglichen und diese in die gewünschte Abfragesprache zu übersetzen. Dieser Ansatz soll dabei helfen, den Umgang mit strukturierten Daten zu vereinfachen und die Benutzerfreundlichkeit der Abfragesprachen zu steigern.

Das maschinelle Übersetzen basiert darauf, einen Übergangspeicher mit häufig wiederkehrenden Sätzen aufzubauen, der mithilfe von statistischen Methoden die Ähnlichkeit zum Quelltext berechnet. Mit generierten Kombinationen werden Ergebnisse erzielt. Ein Beispiel ist das Seq2Seq-Modell, welches darauf basiert. Dieses Modell generiert von manuell erstellten Templates, welche die Struktur der gewünschten Query widerspiegeln, mittels dem Generatorquery eine neue Query. Die Generatorquery versucht passende Entitäten für das entsprechende Template zu finden. [12]

Die Bachelorarbeit "Automatische Übersetzung von natürlicher Sprache nach SPARQL mit neuronalen Netzen" von Nicolas Hoferer und Sebastian Drozdz [3] basiert ebenfalls auf dem Seq2Seq-Modell. Hauptziel der Arbeit ist ein aufbauendes Preprocessing-Modell zu entwickeln, welches die Eingabedaten passend aufbereitet, um die Genauigkeit der Übersetzung zu steigern. Dabei konnten bereits einige gute Ergebnisse erzielt werden, aber es bleibt Optimierungspotenzial. Aufbauend auf der genannten Bachelorarbeit wird in dieser Arbeit ein Teil des Preprocessing evaluiert und das Modell mit grösseren Datensätzen trainiert, um neue Erkenntnisse für die Weiterentwicklung solcher Systeme zu finden.

1.1. Motivation

Um Ansätze und neue Erkenntnisse für die Weiterentwicklung dieses Systems zu gewinnen, ist ein tiefes Verständnis für die zugrundeliegenden Konzepte und eingesetzten Technologien notwendig. Auf dieser Wissensbasis kann das eingesetzte System punktuell verbessert werden. Im Rahmen dieser Arbeit werden deshalb auch die Technologien beschrieben und mit Beispielen praxisnah aufgearbeitet. Die Autoren möchten ein Fundament auf der nächsten Stufe legen, welches die Grundlage für weitere Optimierungen bieten soll.

Ein Bestandteil in der ursprünglichen Bachelorarbeit untersucht das Preprocessing, welches wie in der Einleitung bereits beschrieben, die Eingabedaten für das NMT-Modell aufbereitet. Das Vorgehen und die Methoden des Preprocessing werden im Kapitel 3 genauer erläutert. Um die Qualität des Preprocessing zu ermitteln, wird in dieser Arbeit ein Konzept vorgestellt, das die Preprocessing Ausgabe mittels selbst entwickelten Metriken evaluiert. Damit soll aufgezeigt werden, wie gut sich das Preprocessing verhält. Das Modell ist stark davon abhängig, wie die Eingabedaten übergeben werden.

In einem letzten Schritt wird untersucht, ob durch das Erhöhen der Trainingsdaten bereits eine Verbesserung der Übersetzung erzielt werden kann. Ursprünglich wurde das Modell mit 5'000 Datensätzen der SQA2018 Challenge trainiert und darauf basierend ausgewertet. In einem weiteren Bestandteil dieser Arbeit, wird der Ansatz weiterverfolgt, mit einer höheren Menge von Datensätzen aus der QALD2017 Challenge Aussagen zum Training und einer möglichen Qualitätssteigerung zu machen.

1.2. Ziel der Arbeit

Ziel der Arbeit ist, die Komponenten der ursprünglichen Bacherlorarbeit[3], welche zur Übersetzung von natürlicher Sprache zu SPARQL-Abfragen verwendet wurden, mit grösseren Datensätzen zu evaluieren und Schlüsse daraus ziehen, ob eine Verbesserung mit mehr Trainingsdaten möglich ist. In einem zweiten Teil wird das selbst entworfene Evaluations-Tool für das Preprocessing vorgestellt, vor allem der Aufbau und die Anwendung. Das Tool soll wertvolle Informationen zur Qualität des Preprocessing liefern, da es ein essentieller Schritt in der gesamten Pipeline ist, um die Abfragen in natürlicher Sprache für die Übersetzung aufzubereiten. Dieser Schritt ist notwendig, da der Erfolg der Übersetzung in die SPARQL-Abfragen stark davon abhängt. Zudem sollen Wissenslücken geschlossen werden, die in der zugrundeliegenden Bachelorarbeit nicht mehr behandelt werden konnten. Die Grundlagen und Konzepte im Zusammenhang mit DBpedia und SPARQL werden detaillierter beschrieben.

1.3. Aufbau der Arbeit

Die Dokumentation ist in mehrere Teile aufgebaut. Kapitel 2 beschreibt die theoretischen Grundlagen, sowie die recherchierten Hintergrundinformationen. Im folgenden Kapitel werden die Komponenten des Preprocessing vertieft. Zudem werden die Daten, welche die Grundlage zur Evaluation der einzelnen Komponenten bilden, beschrieben. Kapitel 4 befasst sich mit den durchgeführten Experimenten auf das Modell der zugrundeliegenden Bachelorarbeit und den Resultaten. Diese dienen zur Analyse der Baseline und des NER-Models mit grösseren Datensätzen und enthalten mehr SPARQL-Abfragen. Das Evaluations-Tool des Preprocessing wird in Kapitel 5 vorgestellt und beschrieben. Kapitel 6 macht einen Ausblick zur Weiterentwicklung des Systems, stellt Ansätze dazu vor, und zeigt auf, was in dieser Arbeit erreicht wurde.

2. Theoretische Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen wie Terminologien und Konzepte beschrieben.

2.1. Linked Data

Dieser Abschnitt erklärt das Thema Linked Data sowie die dafür benötigte Abfragesprache *SPARQL*.

Resource Description Framework
Das Resource Description Framework (RDF) ist ein System zur Formulierung von logischen Aussagen über Ressourcen und gilt als wichtiger Bestandteil des Semantic Web. Jede Aussage in einem RDF-Model besteht aus einem Tripel Subjekt, Prädikat, Objekt. Die dafür verwendeten Bezeichner sind global eindeutig und universell, dies wird mit URI's erreicht. [16]

Abbildung 2.1 zeigt ein Beispiel eines RDF-Triples, wo das Subjekt "Barack Obama" über das Prädikat "Geburtsdatum" mit dem Wert (Objekt) "1961-08-04" beschrieben.

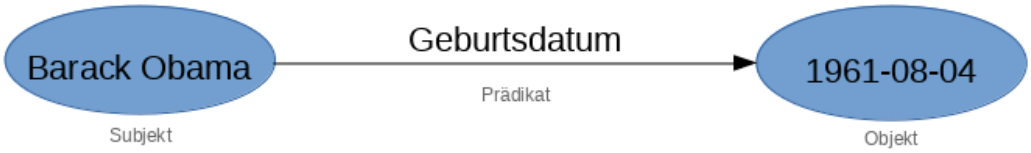
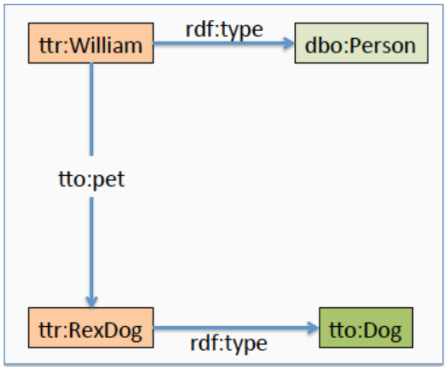


Abbildung 2.1.: Beispiel eines RDF Tripels.

Subjekt: Die Ressource, die beschrieben wird, wird durch eine URI spezifiziert.
Prädikat: Die Eigenschaft der Ressource, welche durch die Aussage beschreibt, wird ebenso durch eine URI spezifiziert.
Objekt: Der Wert der Eigenschaft, muss durch eine URI oder ein Literal. Ein Literal ist ein Text, eine Zahl oder ein Datum.




```
@prefix tto: <http://example.org/tuto/ontology#> .
@prefix ttr: <http://example.org/tuto/resource#> .
@prefix dbo: <http://dbpedia.org/ontology/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#>

# Resources (data)
ttr:William tto:pet ttr:RexDog .
ttr:William rdf:type dbo:Person .
ttr:RexDog rdf:type tto:Dog .
# Classes (ontology)
tto:Dog rdf:type rdfs:Class .
tto:Dog rdfs:label “dog”^^xsd:string .
```

Abbildung 2.2.: Beispiel eines RDF Graphen (links) mit den dazugehörigen Aussagen (rechts)

Mehrere solcher Tripels können, wie in Abbildung 2.2 gezeigt, als Graph dargestellt werden.

SPARQL

SPARQL Protocol And RDF Query Language (SPARQL) ist eine graphische Abfragesprache für RDF. Das World Wide Web Consortium (W3C) trieb die Standardisierung und Entwicklung von SPARQL voran. Es wird benutzt um Abfragen an eine RDF Datenbank zu senden.
Die Abfrage ist, wie der Tripelstore, mit den zuvor behandelten Tripels aufgebaut. Die Variable, die man abfragen will, wird mit einem anführenden ? gekennzeichnet.[14]

Listing 2.1: Beispiel für eine SPARQL-Abfrage, wo das Resultat alle Leader von den United-States ist.

```
1 SELECT DISTINCT ?person WHERE {
2   dbr:United_States dbo:leader ?person.
3 }
```

Listing 2.2: Alle Filme, wo Tom Hanks mitspielt.

```
1 SELECT ?movies WHERE {
2     ?movies rdf:type dbo:Film .
3     ?movies dbo:starring dbr:Tom_Hanks .
4 }
```

2.2. DBpedia

Dieser Abschnitt erklärt die Daten die in dieser Arbeit verwendet wurden.

In dieser Arbeit werden die Daten von DBpedia der Version 2016-04 verwendet. Diese Version umfasst circa 4'233'000 verschiedene Ressourcen. Sie setzt sich in der englischen Sprache aus 754 Klassen und beinahe 60'000 Eigenschaften zusammen.

Entity
Eine *Entity* ist ein bestimmtes und eindeutiges Objekt. Ein anderer Begriff dafür ist "Resource". Die *Entity* wird durch eine Resource-URI spezifiziert. Ein Beispiel für eine *Entity* ist Tom Hanks oder die Stadt Zürich. Jede *Entity* hat mindestens einen *Type*, welcher bestimmt welche *Properties* eine *Entity* besitzen kann. Der *Type* sollte eine *Ontology* sein, jedoch wird bei manchen *Entities* auch eine andere *Entity* als Type verwendet.

Abbildung 2.3 zeigt ein Beispiel der *Entity* Zürich die vom *Type* Stadt ist und drei *Properties* besitzt. Die *Entity* Zürich ist über die URI "http://dbpedia.org/resource/Zürich" auf DBpedia abrufbar.

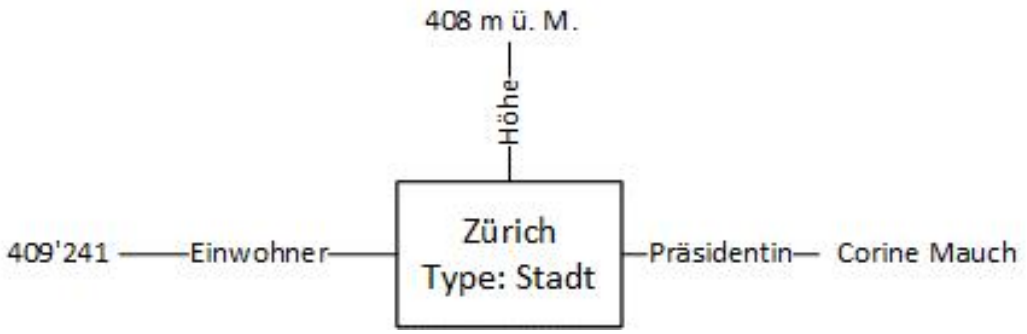


Abbildung 2.3.: Beispiel einer *Entity*.

Ontology

Eine *Ontology* beschreibt den Oberbegriff einer *Entity*. Im Beispiel Zürich ist das "Stadt". Das bedeutet, dass Tom Hanks und Arnold Schwarzenegger beide zur *Ontology* "Schauspieler" gehören. Eine *Ontology* bestimmt welche *Properties* eine *Entity* haben kann. Die *Ontologies* werden als Baum strukturiert, wo jede *Ontology* die *Properties* von ihrer Eltern-*Ontology* erbt. Die Wurzel des Baumes ist *Thing*, wovon jede *Ontology* erbt.

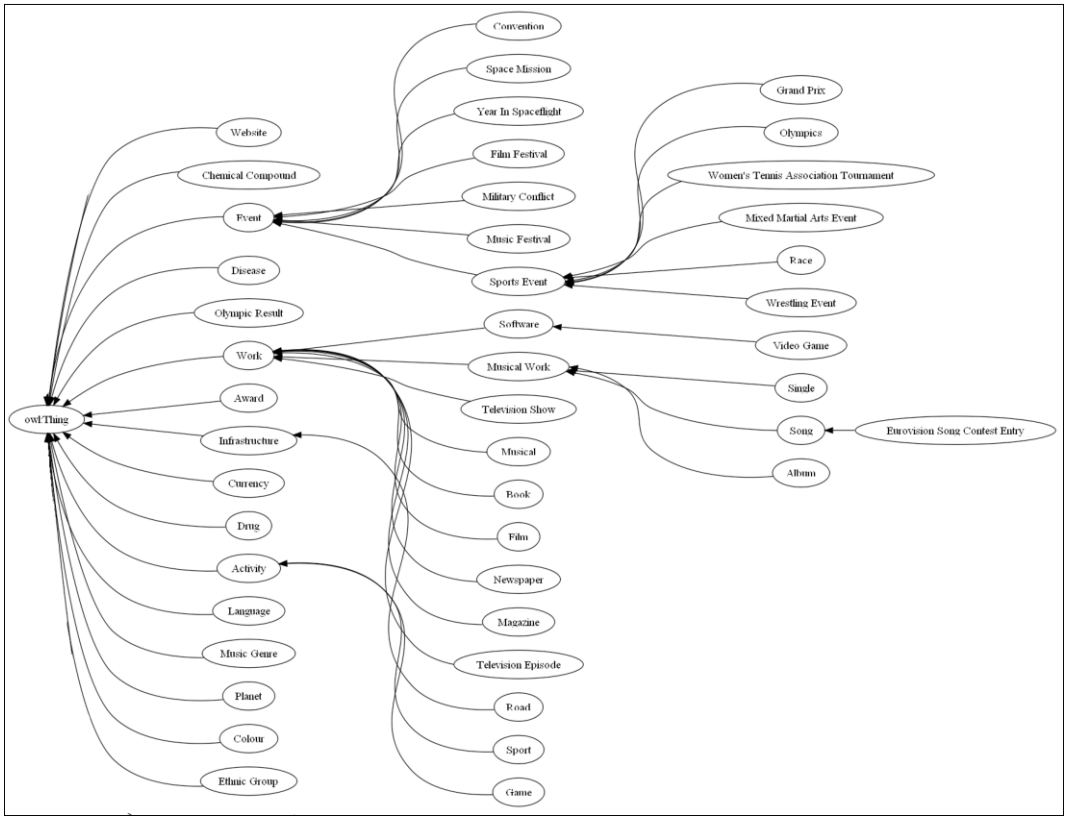


Abbildung 2.4.: Beispiel einer *Ontology*-Struktur.

Abbildung 2.4 zeigt einen Teil der *Ontology*-Struktur von DBpedia.

Property

Die *Property* ist ein Wert der die *Entity* beschreibt, zum Beispiel hat die Stadt Zürich die *Property* Höhe. Ein weiterer Unterschied zwischen *Ontologies* und *Properties* ist, dass der erste Buchstabe bei *Ontologies* nach Konvention gross geschrieben werden und bei *Properties* klein.

2.3. Performanz-Metriken

In diesem Abschnitt werden die verschiedenen Metriken für die Trainingsevaluierung beschrieben. Dafür sind die folgenden Begriffe nötig:

True Positives (TP): Die Anzahl an Antworten vom System, die auch im Ground Truth Resultat vorkommen.
False Positives (FP): Die Anzahl an Antworten vom System, die jedoch nicht im Ground Truth Resultat vorkommen.
True Negatives (TN): Die Anzahl an Negativen Ground Truth Resultaten, die auch negativ vom System geliefert werden. Da es sich bei dieser Arbeit nicht um ein Klassifizierungsproblem handelt, wird dies hier nicht benötigt.
False Negatives (FN): Die Anzahl der Ground Truth Resultate, die nicht vom System geliefert wurden.

2.3.1. Genauigkeit

Die Genauigkeit zeigt prozentual wie viele SPARQL-Abfragen vom System genau mit der Ground Truth Abfragen übereinstimmen. Wenn zum Beispiel die Hälfte der generierten SPARQL-Abfragen genau mit denen von der Ground Truth übereinstimmen, so ist die Genauigkeit 50%.

2.3.2. SPARQL-Genauigkeit

Die Genauigkeit ist jedoch für SPARQL-Abfragen ungeeignet, weil die Reihenfolge der Triples in der SPARQL-Abfrage vertauscht werden können. Zudem können sich die Namen der Variablen zu denen in der Ground Truth unterscheiden.

Listing 2.3: Beispiel für eine Abfrage, wo die Variabel anders benannt sind.

```
1 SELECT DISTINCT ?uri WHERE {
2   <http://dbpedia.org/resource/Londo_Mollari> {http://dbpedia.org/ontology/series}> ?uri
3 }
4 <=>
5 SELECT DISTINCT ?x WHERE {
6   <http://dbpedia.org/resource/Londo_Mollari> <http://dbpedia.org/ontology/series> ?x
7 }
```

Listing 2.4: Beispiel für eine Abfrage, wo die Reihenfolge der Triples keine Rolle spielt.

```
1 SELECT DISTINCT ?uri WHERE {
2   <http://dbpedia.org/resource/The_Ultimate_Merger> <http://dbpedia.org/ontology/
        executiveProducer> ?uri.
3   <http://dbpedia.org/resource/Trump_Productions> <http://dbpedia.org/ontology/
        keyPerson> ?uri
4 }
5 <=>
6 SELECT DISTINCT ?uri WHERE {
7   <http://dbpedia.org/resource/Trump_Productions> <http://dbpedia.org/ontology/
        keyPerson> ?uri .
8   <http://dbpedia.org/resource/The_Ultimate_Merger> <http://dbpedia.org/ontology/
        executiveProducer> ?uri
9 }
```

2.3.3. F-Score

Der F-Score ist ein Mass, welches Präzision und Ausbeute kombiniert, mithilfe des gewichteten harmonischen Mittel. Dafür benötigen wir die Formeln für Präzision 2.1 und Ausbeute 2.2.

$$Präzision = \frac{TP}{TP + FP} \tag{2.1}$$

$$Ausbeute = \frac{TP}{(TP + FN)} \tag{2.2}$$

Die Formel vom F1-Score 2.3 ist wie folgt definiert.

$$F_1 = 2 * \frac{Präzision * Ausbeute}{Präzision + Ausbeute} \tag{2.3}$$

In dieser Arbeit werden jedoch für die Evaluierung der Mikro-F-Score und der Makro-F-Score verwendet.
Mikro-F-Score: Es werden zuerst die Summe aller True und False Positives und aller True- und False negatives berechnet. Aufgrund dieser wird die Präzision und Ausbeute berechnet, um den F-score zu erhalten.
Makro-F-Score: Es werden Präzision, Ausbeute und F-Score pro Frage berechnet und dann der Durchschnitt verwendet. Wenn es Fragen mit keiner Antwort gibt, erhalten diese einen F-Score von 0. [4]

2.4. Stemming

Stemming ist die Stammformreduktion eines Wortes. Der Zweck von Stemming ist, Wörter auf eine gemeinsame Basis zu reduzieren, um zu erkennen, welche Wörter eine ähnliche Bedeutung haben. Dies ermöglicht die Zuordnung der Wörter zueinander. [11] Grob beschrieben entfernt ein *Stemmer* das Ende eines Wortes. Das passiert aber nur, wenn das Ende ein bestimmtes Muster aufweist, z.B. wenn ein Wort in plural geschrieben ist oder es sich um ein konjugiertes, reguläres Verb handelt. [11]

Es existieren verschiedene Stemmer, die für das Stemming verwendet werden. Der *Porter-Stemmer* [13] Algorithmus, welcher auch im Preprocessing der Bachelorarbeit [3] verwendet wird, ist am meisten verbreitet. Für die Evaluation in Abschnitt 5 wird eine neuere Version des Porter-Stemmer, der *Porter2-Stemmer* oder auch *Snowball (English) Stemmer* [19], verwendet, welcher auch vom Entwickler des Porter-Stemmer als Ablösung des ursprünglichen Porter-Stemmers empfohlen wird. [20]

Beispiele 2.4.1 bis 2.4.5 zeigen Stemming-Resulate von Nomen, Adjektiven und Verben. Beispiel 2.4.6 zeigt das Resultat, wenn Stemming auf alle Wörter eines Satzes angewendet wird. In allen Beispielen wird der Porter2-Stemmer verwendet.

Beispiel 2.4.1
dog, dog's, dogs –> dog

Beispiel 2.4.2
brilliant, brilliantly –> brilliant

Beispiel 2.4.3
suggest, suggestive, suggestively –> suggest

Beispiel 2.4.4
condition, conditional, conditionally –> condit

Beispiel 2.4.5
see –> see
saw –> saw
seen –> seen

Beispiel 2.4.6
The boy's cars are different colors –> The boy car are differ color

In Beispiel 2.4.4 ist ersichtlich, dass für alle drei Wörter der Stamm "condit" ermittelt wird. Zwar wäre der korrekte Stamm "condition", allerdings ist die Funktion des Stemming trotzdem erfüllt, da die Wörter einen gemeinsamen Stamm besitzen und so einander zugeordnet werden können.

In Beispiel 2.4.5 werden für alle drei Varianten des Verbes "see" andere Wortstämme ausgegeben. Vor allem bei irregulären Verben ist ein gemeinsamer Wortstamm nicht immer erzeugbar, da diese Art von Endungen nicht berücksichtigt werden oder ein Entfernen der Endung nicht zwingend auf einen Wortstamm zurückführt.

2.5. Levenshtein-Distanz

Die *Levenshtein-Distanz* ist ein Mass, welches beschreibt, wie viele Editieroperationen mindestens nötig sind, um eine Zeichenkette in eine andere zu überführen. Zu den Editieroperationen gehören das Einfügen, Löschen oder Ersetzen von einzelnen Zeichen. [15]

Beispiel 2.5.1
Maus vs. Haus
Die Levenshtein-Distanz beträgt 1, weil ein Zeichen ersetzt wird. Also bei *Maus* wird das M durch ein H ersetzt, um *Haus* zu erhalten.

Ein verbreiteter Algorithmus verwendet dynamische Programmierung für die Berechnung der Levenshtein-Distanz. Dabei wird eine (m+1)x(n+1)-Matrix erstellt, wobei m der Anzahl Zeichen der ersten Zeichenkette und n der Anzahl Zeichen der zweiten Zeichenkette entspricht. Zu beiden Längen wird eins addiert, weil das leere Wort ϵ ebenfalls berücksichtigt wird. Das leere Wort ϵ steht für eine leere Zeichenkette.

Jede Zelle *(i,j)* der Matrix beschreibt die Levenshtein-Distanz zwischen dem i- und j-Präfix der beiden Zeichenketten. [15] Der Algorithmus für die Berechnung der Zellenwerte und die Ermittlung der Levenshtein-Distanz ist wie folgt definiert [15]:

$$D_{0,0} = 0,\ D_{i,0} = i,\ D_{0,j} = j$$
$$\text{sowie für alle } 1 \leq i \leq m,\ 1 \leq j \leq n$$
$$D_{i,j} = \begin{cases} {}_{i-1,j-1} + 0 \text{ gleiches Zeichen} \\ D_{i-1,j-1} + 1 \text{ Zeichen ersetzen} \\ D_{i-1,j-1} + 0 \text{ Zeichen einfügen} \\ D_{i-1,j-1} + 0 \text{ Zeichen löschen} \end{cases}$$

Abbildung 2.5 zeigt eine solche Matrix anhand der Wörter *Qualität* und *Quantität*. In der Ecke unten rechts ist die minimale Levenshtein-Distanz, in diesem Fall also zwei. Im gekennzeichneten Pfad muss dafür ein Zeichen ersetzt und ein Zeichen eingefügt werden.

	ϵ	Q	u	a	n	t	i	t	ä	t
ϵ	0	1	2	3	4	5	6	7	8	9
Q	1	0	1	2	3	4	5	6	7	8
u	2	1	0	1	2	3	4	5	6	7
a	3	2	1	0	1	2	3	4	5	6
l	4	3	2	1	1	2	3	4	5	6
i	5	4	3	2	2	2	2	3	4	5
t	6	5	4	3	3	2	3	2	3	4
ä	7	6	5	4	4	3	3	3	2	3
t	8	7	6	5	5	4	4	3	3	2

Abbildung 2.5.: Matrix für die Berechnung der Levenshtein-Distanz zwischen zwei Zeichenketten, wobei die minimale Levenshtein-Distanz rot markiert ist.

2.5.1. Word-based Levenshtein-Distanz

Für die Evaluation in Abschnitt 5 wird die Levenshtein-Distanz zwischen Wörtern statt Zeichen ermittelt. Das Vorgehen ist dasselbe wie bei der Levenshtein-Distanz zwischen Zeichenketten. Abbildung 2.6 zeigt exemplarisch wie die Berechnung der Levenshtein-Distanz zwischen Wörtern durchgeführt wird.

	ϵ	Burbank	California
ϵ	0	1	2
Luther	1	1	2
Burbank	2	1	2

Abbildung 2.6.: Matrix für die Berechnung der word-based Levenshtein-Distanz.

2.6. POS-Tagging

Part-Of-Speech-Tagging (kurz: POS-Tagging) ist ein Verfahren, das die Wortarten der Wörter in einem Satz bestimmt. Die Identifizierung der Wortarten unterstützt die Ermittlung von wahrscheinlichen Nachbarwörtern. In der Bachelorarbeit [3] wird der POS-Tagger von der *Stanford CoreNLP* Bibliothek verwendet. [17]
In der Arbeit [5] werden die POS-Tags in 2.1 für die Ontologie Extrahierung verwendet:

Abbildung 2.7 zeigt ein Beispiel POS-Tagging für die Frage “which comic characters are painted by bill finger?”. Die Bedeutung der Tags wird in [1] beschrieben.

Tabelle 2.1.: Übersicht über die verwendeten POS-Tags in [3].

Tag	Beschreibung	Beispiele
NN	Substantiv (Singular) oder Masse	Person
NNS	Substantiv (Plural)	Persons
DT	Determinative (determiner)	Determinative sind Pronomen, z.B. the, that, a, some
IN	Präposition oder untergeordnete Konjunktion	under, on, beside, after, before, until etc.

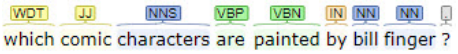


Abbildung 2.7.: Beispiel POS-Tags für eine Frage.

2.7. Named Entity Recognition

Named Entity Recognition (NER) dient dazu, Eigennamen in Texten zu erkennen. Bei Eigennamen handelt es sich um ein oder mehrere Wörter, die eine Entity beschreiben. Es existieren verschiedene Werkzeuge, die für NER verwendet werden. Für die Entities Extrahierung im Preprocessing von [3] wird dafür der Online-Service von DBpedia-Spotlight[1] verwendet. Neben dem Text, wo die Entities identifiziert werden, wird bei der Spotlight-API eine *Confidence Score* angegeben. Je höher die Confidence Score, desto weniger Entities werden zurückgegeben. Das bedeutet, dass nur Entities zurückgegeben werden, die mit hoher Wahrscheinlichkeit den Satzteilen zugeordnet werden können. Wenn die Confidence Score tiefer angesetzt ist, dann werden mehr Entities zurückgegeben.

Für die Arbeit [3] wird eine Confidence Score von 0.1 verwendet. Die Arbeit enthält eine detailliertere Beschreibung von Spotlight und die Begründung der Verwendung der gewählten Confidence Score.

[1] https://www.dbpedia-spotlight.org/api

3. Vorgehen und Methoden

3.1. Preprocessing der Bachelorarbeit

Im Preprocessing werden die Eingabedaten für die Übersetzung aufbereitet. Dieser Schritt ist essenziell, denn das korrekte Extrahieren von Entities, Properties und Ontologies beeinflusst den Erfolg der Übersetzung und ermöglicht das Erkennen von bestimmten Fragearten, so dass SPARQL-Templates verwendet werden können.
In Abschnitt 2.2 befindet sich eine detaillierte Beschreibung von Entities, Properties und Ontologies.

Das Ziel des Preprocessing ist, die Tokens und das Vokabular für das Trainieren des NMT-Modells vorzubereiten. Dazu gehört in den Natural Language (NL)-Fragen möglichst alle Textteile zu erkennen, die Entities, Properties oder Ontologies sind, und diese entsprechend durch die passende DBpedia-URI zu ersetzen. Damit können Teile der NL-Frage Teilen der SPARQL-Query zugeordnet werden.

Das Preprocessing umfasst ein oder mehrere Pipelines, welche wiederum aus mehreren Funktionen bestehen, die nacheinander ausgeführt werden. Das Endresultat des Preprocessing unterscheidet sich je nach den ausgeführten Funktionen. Beispielsweise ist es möglich, im Preprocessing nur die Entity-Extrahierung durchzuführen, oder die Resultate der Entity-Extrahierung, Ontology-Extrahierung und Property-Extrahierung werden zusammengefügt.

Für die unterschiedlichen Pipelines werden je nach ihrer Verantwortlichkeit verschiedene Eingaben benötigt und verschiedene Ausgaben produziert. Die Ausgaben dienen wiederum als Eingaben für das Trainieren des NMT-Modells. Die Eingaben und Ausgaben sind jeweils in Form von Dateien gegeben. Die Dateien werden in der technischen Dokumentation B genauer beschrieben. Eine detaillierte Beschreibung der wichtigsten Pipelines ist in den Abschnitten 3.1.2 bis 3.1.6 gegeben.

Eine weitere Aufgabe des Preprocessing ist, die SQA2018 Daten in Evaluation und Training aufzuteilen. Die Aufteilung ist wie folgt: 90% der Daten sind für das Training, 10% für die Evaluation.

3.1.1. Tokens und Vokabular

Die Hauptaufgabe des Preprocessing ist die Vorbereitung der Tokens und des Vokabulars für das NMT-Modell. Es ist essenziell, dass alle wichtigen Elemente der NL-Frage und SPARQL-Query eindeutige und unterscheidbare Tokens sind. Elemente sind Wörter und auch wichtige, syntaktische Elemente der SPARQL-Queries, die keine Wörter sind, beispielweise die eckigen Klammern { und } um die RDF-Triples. Für solche Elemente wie Klammern, Punkte etc. sind daher spezifische Tokens definiert.

Spezifische Tokens werden auch für Entities, Ontologies und Properties verwendet. Anstatt die URI als Token zu verwenden, wird sie in eine andere Form umgeschrieben. Im Postprocessing (nach einer Übersetzung) werden die Tokens durch die korrekten URI's wieder ersetzt.

Tabelle 3.1 zeigt eine Übersicht spezieller Tokens, die für URI's und sonstige syntaktische Elemente von SPARQL verwendet werden.

Das Vokabular entspricht dem Wortschatz des Modells. Es gibt an, welche Wörter für die Eingabesprache und welche Wörter für die Ausgabesprache verfügbar sind. Das Vokabular ist aus den Tokens gebildet, welche im Preprocessing ermittelt werden, da die Verwendung vom gesamten Vokabular der englischen Sprache und von allen DBpedia-Entities, -Ontologies und -Properties aufgrund von mangelndem Speicher nicht praktikabel ist. [6]

Tabelle 3.1.: Liste der Tokens mit Beschreibung

Token	**Beschreibung**
dbr_Y	Diese Tokens repräsentieren DBpedia-Entities, die im URI-Pfad */resource* liegen. Beim Y handelt es sich um das Label der Entity. **Beispiel 3.1.1** http://dbpedia.org/resource/Bill_Finger –> dbr_Bill_Finger
dbo_Y	Bei dbo_Y handelt es sich um eine DBpedia-Ontologies. Das Y wird jeweils durch das Label der Ontologie ersetzt. **Beispiel 3.1.2** http://dbpedia.org/ontology/primeMinister –> dbo_primeMinister
dbp_Y	dbp_Y Tokens stehen für DBpedia-Properties. Y wird durch das Label der Property ersetzt. **Beispiel 3.1.3** http://dbpedia.org/property/mother –> dbp_mother
sep_dot	Entspricht einem Trennungspunkt .
brack_open	Eine geöffnete geschweifte Klammer {
brack_close	Eine geschlossene geschweifte Klammer }
var_uri	Die gesuchte(n) Antwort(en), ursprünglich gekennzeichnet als ?uri.
var_x	Eine zusätzliche Variable, ursprünglich gekennzeichnet als ?x
attr_open	Entspricht einer geöffneten Klammer (. Wird bei COUNT verwendet.
attr_close	Entspricht einer geschlossenen Klammer). Wird bei COUNT verwendet.
rdf_type	Entspricht dem RDF-Type, also dem Oberbegriff resp. der Ontologie einer Variable.

3.1.2. Baseline

Die Baseline-Pipeline dient zum Zweck, die Tokens und das Vokabular für das Trainieren des Baseline-Modells vorzubereiten. Es werden keine Entities, Properties oder Ontologies aus der NL-Frage extrahiert. Abbildung 3.1 zeigt eine grafische Darstellung der Baseline-Pipeline inklusive Eingabe- und Ausgabedateien.

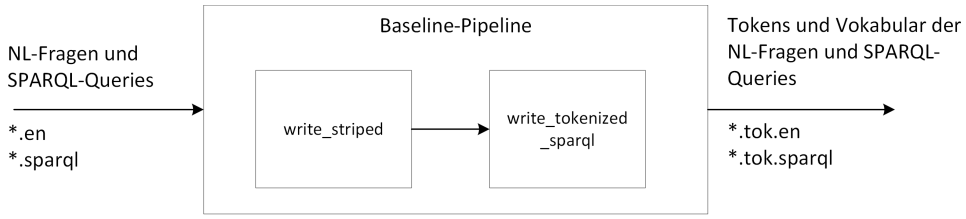


Abbildung 3.1.: Grafische Darstellung der Pipeline für die Vorbereitung der Tokens des Baseline-Model

Innerhalb der Pipeline werden zwei Funktionen *write_striped* und *write_tokenized_sparql* angewendet.

write_striped Die eingegebenen NL-Fragen werden ohne Veränderungen oder Extrahierungen für die Tokens übernommen. Da keine Extrahierungen vorgenommen werden, ist die Ausgabe der NL-Frage gleich wie die Eingabe, und die resultierenden Tokens und das Vokabular entsprechen der ursprünglichen Eingabefrage.

write_tokenized_sparql Aus der SPARQL-Query werden die SPARQL-Tokens vorbereitet. Die DBpedia-URI's oder syntaktische Elemente, wie geschweifte Klammern, werden in klar unterscheidbare Tokens umgewandelt.

Listing 3.1 zeigt anhand eines Exemplars die Resultate der Baseline-Pipeline und wie die Tokens aus der SPARQL-Query erstellt werden.

Listing 3.1: Beispiel Eingabe-Daten (NL-Frage und SPARQL-Query) und der Ausgabe nach Durchlauf der Baseline-Pipeline.

```
1  Eingabe
2  NL-Frage: name the alma mater of michael hahn ?
3  SPARQL: SELECT DISTINCT ?uri WHERE {<http://dbpedia.org/resource/Michael_Hahn
       > <http://dbpedia.org/Ontologie/almaMater> ?uri }
4
5  Ausgabe
6  NL-Frage Tokens: name the alma mater of michael hahn ?
7  SPARQL Tokens: SELECT DISTINCT var_uri WHERE brack_open dbr_Michael_Hahn
       dbo_almaMater var_uri brack_close
```

3.1.3. Entity Extrahierung

Bei der Entity Extrahierung werden die Entities aus der NL-Frage identifiziert und durch die entsprechenden DBpedia-URI's ersetzt. Die URI von Entities enthalten *resource* im Pfad.

Abbildung 3.2 zeigt eine grafische Darstellung der Entity Extrahierung. Links werden die nötigen Eingabedateien für die Pipeline dargestellt, rechts die produzierten Ausgabedateien, die für das Trainieren des NER-Modells nötig sind.

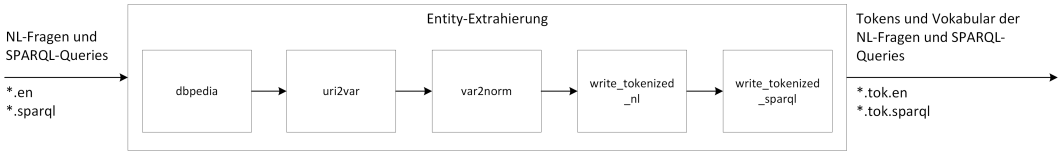


Abbildung 3.2.: Grafische Darstellung der Pipeline für die Vorbereitung der Tokens des Entitiesmodells

Für die Entity Extrahierung werden die folgenden Funktionen ausgeführt:

dbpedia Die Entities werden in der NL-Frage mit Hilfe von DBpedia-Spotlight identifiziert. Jede NL-Frage wird an die Spotlight-API gesendet. Die Spotlight-API liefert wiederum eine Liste der erkannten Entities in der Frage zurück. In der NL-Frage werden die entsprechenden Wörter durch die treffenden Entities-URIs aus der Antwort von Spotlight ersetzt. Zusätzlich wird vor der URI der Typ der Entity, gefolgt von einem @, angegeben. Der Typ *NotFound* bedeutet, dass von Spotlight für diese Entity kein Typ angegeben wird.

Beispiel 3.1.4
In: which comic characters are painted by bill finger?
Out: which comic characters are
<NotFound@http://dbpedia.org/resource/Painting> by
<Person@http://dbpedia.org/resource/Bill_Finger>?

uri2var Die URIs der Entities werden in der NL-Frage und in der SPARQL-Query durch Variablen ersetzt. Dabei entspricht der Name der Variable dem Typ der Entity und einer Zahl, die für die Reihenfolge der Entities verwendet wird. Aufgrund der verwendeten Confidence Score für die Spotlight-API werden mehr Entities identifiziert, als in der SPARQL-Query eigentlich verwendet werden. Diese Entities werden nicht in Variablen umgewandelt, da sie keiner Entity in der SPARQL-Query zugeordnet werden können. Im nachfolgenden Beispiel

3.1.5 ist die Entity *<NotFound@http://dbpedia.org/resource/Painting>* nicht in der SPARQL-Query und wird daher in der NL-Frage nicht durch eine Variable ersetzt.

Beispiel 3.1.5
In: which comic characters are
<NotFound@http://dbpedia.org/resource/Painting> by
<Person@http://dbpedia.org/resource/Bill_Finger>?
Out: which comic characters are
<NotFound@http://dbpedia.org/resource/Painting> by <Person1>?

var2norm In diesem Schritt erfolgt die Normalisierung der Variablen in Tokens, d.h. alle Variablennamen in der NL-Frage und der SPARQL-Query werden zu *varX* verändert, wobei X eine Zahl ist und für die Reihenfolge der Entities steht.

Beispiel 3.1.6
In: which comic characters are
<NotFound@http://dbpedia.org/resource/Painting> by <Person1>?
Out: which comic characters are
<NotFound@http://dbpedia.org/resource/Painting> by <var1>?

write_tokenized_nl Diese Funktion erstellt die Tokens für die NL-Frage. Die Ergebnisse aus *var2norm* werden ohne weitere Anpassungen als Tokens für die NL-Frage übernommen.

write_tokenized_sparql Aus der SPARQL-Query werden die Tokens erstellt. Diese Funktion ist identisch zu der *write_tokenized_sparql* Funktion in der Baseline-Pipeline.

In der nachfolgenden Listing 3.2 sind exemplarisch Eingabe- und Ausgabedaten, die die resultierenden NL-Frage und SPARQL-Query nach der Entity Extrahierung aufzeigen.

Listing 3.2: Beispiel Eingabedaten (NL-Frage und SPARQL-Query) und die Ausgabe nach der Entity Extrahierung.

```
1  Eingabe
2  NL−Frage: which comic characters are painted by bill finger?
3  SPARQL: SELECT DISTINCT ?uri WHERE {?uri <http://dbpedia.org/Ontologie/creator>
       <http://dbpedia.org/resource/Bill_Finger> . ?uri <http://www.w3.org/1999/02/22−
       rdf−syntax−ns#type> <http://dbpedia.org/Ontologie/ComicsCharacter>}
4
5  Ausgabe
6  NL−Frage Tokens: which comic characters are <NotFound@http://dbpedia.org/resource/
       Painting> by <var1>?
7  SPARQL Tokens: SELECT DISTINCT var_uri WHERE brack_open var_uri dbo_creator
       <var1> sep_dot var_uri rdf_type dbo_ComicsCharacter brack_close
```

3.1.4. Ontology Extrahierung

Je nach Formulierung der NL-Frage können Ontologies extrahiert werden, die für die Bildung eines SPARQL-Query nötig sind. Allerdings kommen nicht zwingend in allen Fragen Ontologies vor. Der *RDF-Type* einer Variable ist ebenfalls mit einer Ontologie beschrieben. Die Angabe eines RDF-Type einer Variable sagt aus, dass der RDF-Type respektive die Ontologie dem Oberbegriff der Variable entspricht.

Beispiel 3.1.7
?x <http://www.w3.org/1999/02/22-rdf-syntax-nstype>
<http://dbpedia.org/Ontologie/MilitaryConflict>
Die Variable ?x ist vom RDF-type MilitaryConflict, respektive MilitaryConflict ist der Oberbegriff von ?x.

Für die Ontology Extrahierung ist eine Pipeline von Funktionen definiert, deren Ablauf in Abbildung 3.3 aufgezeigt ist.

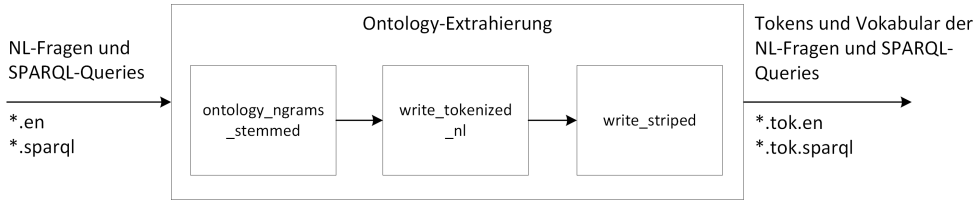


Abbildung 3.3.: Grafische Darstellung der Pipeline für die Vorbereitung der Tokens des Ontology Modells

Die Funktionen erfüllen die folgenden Aufgaben:

ontology_ngrams_stemmed Die Ontologies aus der NL-Frage werden mittels eines Lookup der DBpedia-Ontologies erkannt. Diese DBpedia-Ontologies stammen aus der DBpedia Version 2016-04, die in Abschnitt 3.2 beschrieben ist. Die Ontologies im Lookup sind *stemmed*, d.h. es werden jeweils die Wortstämme verwendet. Die Daten für die DBpedia-Ontologies sind in Anhang B weiter beschrieben.

Pro Wort in der NL-Frage ermittelt der POS-Tagger einen POS-Tag. Für das POS-Tagging wird Stanford CoreNLP [17] verwendet.

Mittels dem Porter-Stemmer der NLTK-Bibliothek [18] wird für jedes Wort der NL-Frage der Wortstamm ermittelt. Der Wortstamm ist nötig, um Lookups der DBpedia-Ontologies durchzuführen. Für die Lookups werden ngrams aus Wörtern der NL-Frage gebildet. Die ngrams bestehen jeweils aus ein bis drei Wörtern und werden in absteigender Länge geprüft, d.h. es werden zuerst *Trigramme*, dann *Bigramme* und zuletzt *Monogramme* gebildet und für die Lookups verwendet.

Nach dem Stemming erfolgt eine Überprüfung der POS-Tags. Die Tags dienen dazu, gewisse Ontologies auszuschliessen. Denn bei bestimmten Kombinationen von POS-Tags handelt es sich mit grosser Wahrscheinlichkeit nicht um eine Ontology [5]. Wenn ein Wort zusammen mit den Nachbarwörtern, die vorher und/oder nachher sind, eine bestimmte Kombination bildet, wird der Lookup der DBpedia-Ontology übersprungen und es wird mit dem nächsten Wort fortgefahren.

Nach der Überprüfung der POS-Tags erfolgt ein Lookup der DBpedia-Ontology mittels dem gebildeten ngram. Zusätzlich werden Synonyme für die Wörter der ngrams verwendet und ein weiterer Lookup durchgeführt. Es wird eine Liste mit 13 Synonymen verwendet.

Wenn ein Lookup mittels ngram und/oder Synonym erfolgreich ist, wird der passende ngram in der Frage durch die mittels dem Lookup gefundenen Ontology-URI ersetzt. Wenn der Lookup mit ngram und Synonym fehlschlägt, wird mit dem nächsten ngram der Frage fortgefahren.

Beispiel 3.1.8
In: which comic characters are painted by bill finger?
Out: which <http://dbpedia.org/ontology/ComicsCharacter> are painted by bill finger?
Die Ontology wird mittels dem Bigramm comic charac identifiziert. **charac** entspricht dem ermittelten Wortstamm von **character**.

Beispiel 3.1.9

In: how many people have headed organizations headquartered at niederkirchnerstrasse?

Out: how many <http://dbpedia.org/ontology/Person> have headed <http://dbpedia.org/ontology/Organ> headquartered at niederkirchnerstrasse?

Die Ontology http://dbpedia.org/ontology/Person wird erkannt, da für people ein Synonym person verwendet wird.

Die Ontology <http://dbpedia.org/ontology/Organ> wird eingesetzt, weil für organization der Wortstamm Organ ermittelt wird. Korrekt wäre aber die Ontology <http://dbpedia.org/ontology/organization>.

write_tokenized_nl In diesem Schritt werden die Tokens für die NL-Frage vorbereitet. Dabei werden alle ermittelten Ontology-URI's durch entsprechende Tokens ersetzt, die syntaktisch gleich zu den Tokens der SPARQL-Query sind. Es ist allerdings möglich, dass je nach Länge und Formulierung der NL-Frage mehr und/ oder falsche Ontologies extrahiert werden, verglichen mit der SPARQL-Query.

Beispiel 3.1.10

In: which <http://dbpedia.org/Ontologie/ComicsCharacter> are painted by bill finger?

Out: which <dbo_ComicsCharacter> are painted by bill finger?

write_striped Diese Funktion bereitet die Tokens der SPARQL-Query vor. Die SPARQL-Tokens werden bereits in der Baseline-Pipeline vorbereitet und für die Ontologies ohne Veränderungen übernommen. Die SPARQL-Tokens der Ontology Extrahierung entsprechen also den SPARQL-Tokens der Baseline-Pipeline.

Listing 3.3 zeigt ein Beispiel für eine Ausgabe nach der Ontology Extrahierung.

Listing 3.3: Beispiel Eingabedaten (NL-Frage und SPARQL-Query) und die Ausgabe nach Ontologies Extrahierung.

```
1  Eingabe
2  NL-Frage: which comic characters are painted by bill finger?
3  SPARQL: SELECT DISTINCT ?uri WHERE {?uri <http://dbpedia.org/ontology/creator>
       <http://dbpedia.org/resource/Bill_Finger> . ?uri <http://www.w3.org/1999/02/22-
       rdf-syntax-ns#type> <http://dbpedia.org/ontology/ComicsCharacter>}
4
5  Ausgabe
6  NL-Frage Tokens: which <dbo_ComicsCharacter> are painted by bill finger?
7  SPARQL Tokens: SELECT DISTINCT var_uri WHERE brack_open var_uri dbo_creator
       dbr_Bill_Finger sep_dot var_uri rdf_type dbo_ComicsCharacter brack_close
```

3.1.5. Properties Extrahierung

Wie Ontologies kommen Properties nicht in allen NL-Fragen und SPARQL-Queries vor. Die Property Extrahierung ist ähnlich aufgebaut wie die Ontology Extrahierung. Für die Lookups werden DBpedia-Properties verwendet, und im Gegensatz zu der Ontology Extrahierung wird keine Überprüfung der POS-Tags durchgeführt. In Abbildung 3.4 sind die aufgerufenen Funktionen und deren Reihenfolge dargestellt.

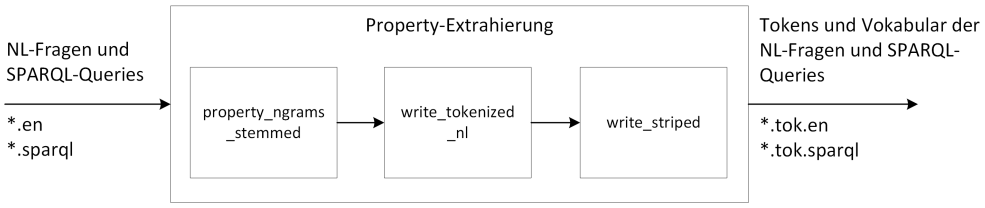


Abbildung 3.4.: Grafische Darstellung der Pipeline für die Vorbereitung der Tokens des Propertiesmodells

Die Funktionen sind:

property_ngrams_stemmed Für die Erkennung von Properties existieren analog zu den Ontologies DBpedia-Properties, welche aus den DBpedia Daten 2016-04 stammen. Mittels Lookup ist es möglich, die passende DBpedia-URI's für eine entsprechende Property zu finden.

Analog zu *ontology_ngrams_stemmed* ermittelt der Porter-Stemmer der NLTK-Bibliothek [18] für jedes Wort der NL-Frage den Wortstamm. Der Wortstamm ist nötig, um Lookups in den DBpedia-Properties durchzuführen, da diese

stemmed sind. Die Lookups erfolgen jeweils mit ngrams aus der NL-Frage - Zuerst *Trigramme*, dann *Bigramme* und zuletzt *Monogramme*.

Ein gebildeter ngram wird für den Lookup der DBpedia-Property verwendet. Zusätzlich werden Wörter des ngram mit einem Synonym ersetzt, sofern dieser in der Liste der Synonyme enthalten ist, welche 13 Synonyme enthält. Sollte der Lookup erfolgreich sein, mit oder ohne Synonym, wird das entsprechende ngram in der NL-Frage durch die DBpedia-Property-URI ersetzt. Bei keiner Übereinstimmung wird mit dem nächsten ngram fortgefahren.

Beispiel 3.1.11
In: was reza amrollahi born in iran?
Out: was reza amrollahi <http://dbpedia.org/property/bornIn> iran?
Die Property wird mittels dem Bigramm born in gefunden.

write_tokenized_nl Die Tokens der NL-Frage werden vorbereitet, so dass die Property-URI's durch passende Tokens ersetzt werden, die gleichnamig wie die Tokens der SPARQL-Query sind. Wie bei den Ontologies können auch hier je nach Länge und Formulierung der NL-Frage mehr und/oder falsche Properties extrahiert werden, verglichen mit den Properties in der SPARQL-Query.

Beispiel 3.1.12
In: was reza amrollahi <http://dbpedia.org/property/bornIn> iran?
Out: was reza amrollahi <dbp_bornIn> iran?

write_tokenized_sparql Für die SPARQL-Query werden die Tokens erstellt. Das Resultat ist identisch zu dem der gleichnamigen Funktion in der Baseline-Pipeline.

Listing 3.4 zeigt eine Beispielausgabe der Property Extrahierung. In diesem Beispiel befindet sich die Property *<dbp_bornIn>* in der NL-Frage, die nicht in der SPARQL-query vorkommt. Ausserdem wird die Property *<dbp_birthplace>* der SPARQL-Query in der NL-Frage nicht erkannt.

Listing 3.4: Beispiel Eingabedaten (NL-Frage und SPARQL-Query) und die Ausgabe nach Properties Extrahierung.

```
1  Eingabe
2  NL-Frage: was reza amrollahi born in iran?
3  SPARQL: ASK WHERE {<http://dbpedia.org/resource/Reza_Amrollahi> <http://dbpedia
       .org/property/birthplace> <http://dbpedia.org/resource/Iran> }
4
5  Ausgabe
6  NL-Frage Tokens: was reza amrollahi <dbp_bornIn> iran?
7  SPARQL Tokens: ASK WHERE brack_open dbr_Reza_Amrollahi dbp_birthplace dbr_Iran
       brack_close
```

3.1.6. Alle Extrahierungen zusammen

Die Resultate der Pipelines bzw. Extrahierungen beschrieben in 3.1.3 bis 3.1.5 werden zusammengefügt. Die Ontologies basieren auf den Resultaten des beschriebenen Extrahierungsprozesses in 3.1.4. Die Entities und die Properties basieren allerdings auf den "Best Case"-Varianten [7] und [8] und nicht auf die in Abschnitt 3.1.3 und 3.1.5 beschriebenen Varianten. Bei den "Best Cases" werden die URI's nach dem gleichen Vorgehen, wie in den vorhergehenden Abschnitten beschrieben, ermittelt. Zusätzlich werden die ermittelten URI's mit den URI's in der SPARQL-Query verglichen und nur dann in der NL-Frage eingesetzt, wenn eine Übereinstimmung in der SPARQL-Query existiert. Dieses Vorgehen bewirkt, dass z.B. überflüssige Entities nicht berücksichtigt werden.

Bevor alle Ergebnisse zusammengefügt werden können, ist es nötig, zuerst die Migration der Resultate der Entity und Property Extrahierungen durchzuführen. Dies wird mit einer weiteren, eigenen Pipeline durchgeführt (vgl. [9]).
Für das Zusammenfügen wird eine Funktion in der Pipeline aufgerufen:

merge_files Diese Funktion fügt die Resultate aus der Entity-Property-Migration und der Ontology Extrahierung zusammen. Falls sich eine Property oder Entity mit einer Ontology überschneidet, wird die Ontology bevorzugt. Überschneidungen kommen vor allem zwischen Properties und Ontologies vor.

Beispiel 3.1.13
NL-Frage: where was the battle fought where 2nd foreign infantry regiment participated?
Entity-Property: where was the <dbo_battle> fought where <var0> participated?
Ontology: where was the <dbo_MilitaryConflict> fought where 2nd foreign infantry regiment participated?

Out: where was the <dbo_MilitaryConflict> fought where <var0> participated?
Die Ontology <dbo_MilitaryConflict> wird statt der Property <dbo_battle> gewählt.

Listing 3.5 zeigt die Ausgabe wenn alle Extrahierungen zusammengefügt werden.

Listing 3.5: Beispiel Eingabedaten (NL-Frage und SPARQL-Query) und die Ausgabe nach Migration der Entities, Properties und Ontologies.

```
1  Eingabe:
2  NL-Frage der Entity-Property-Migration: what are the airlines whose <dbp_hubs> is <
       var0>?
3  NL-Frage der Ontology Extrahierung: what are the <dbo_Airline> whose hub is san
       francisco international airport?
4  SPARQL-Query der Entity-Property-Migration: SELECT DISTINCT var_uri WHERE
       brack_open var_uri dbp_hubs <var0> sep_dot var_uri rdf_type dbo_Airline
       brack_close
5  SPARQL-Query der Ontology Extrahierung: SELECT DISTINCT var_uri WHERE
       brack_open var_uri dbp_hubs dbr_San_Francisco_International_Airport sep_dot
       var_uri rdf_type dbo_Airline brack_close
6
7  Ausgabe:
8  NL-Frage nach Entity-Property-Ontology-Migration: what are the <dbo_Airline> whose
       <dbp_hubs> is <var0>?
9  SPARQL-Query nach Entity-Property-Ontology-Migration: SELECT DISTINCT var_uri
        WHERE brack_open var_uri dbp_hubs <var0> sep_dot var_uri rdf_type
       dbo_Airline brack_close
```

3.1.7. Vokabular

Für das Trainieren der Modelle ist für die NL-Frage und für die SPARQL-Query je ein Vokabular notwendig. Das Vokabular besteht aus den Tokens, die, wie in den vorhergehenden Abschnitten 3.1.2 - 3.1.6 beschrieben, vorbereitet werden. Die Erstellung des Vokabulars ist der letzte Schritt in den Pipelines.

Listing 3.6: Beispiel Eingabedaten (NL-Frage und SPARQL-Query) und die Ausgabe nach Erstellung des Vokabulars.

```
1 NL-Frage: which <dbo_ComicsCharacter> are painted by <var0>?
2 NL-Vokabular: which, <dbo_ComicsCharacter>, are, painted, by, <var0>
3 SPARQL: SELECT DISTINCT var_uri WHERE brack_open var_uri dbo_creator <var0>
      sep_dot var_uri rdf_type dbo_ComicsCharacter brack_close
4 SPARQL-Vokabular: SELECT, DISTINCT, var_uri, WHERE, brack_open, dbo_creator, <
      var0>, sep_dot, rdf_type, dbo_ComicsCharacter, brack_close
```

3.2. Daten

3.2.1. SQA2018

Der Scalable Question Answering 2018 (SQA2018) Datensatz wurde schon bei der Entwicklung dieser Anwendung verwendet, um das Modell zu trainieren und zu evaluieren. Dieser Datensatz stammt von der project-hobbit Organisation und wurde während der SQA2018 Challenge veröffentlicht.

3.2.2. DBNQA

Der DBpedia Neural Question Answering (DBNQA) Datensatz wurde von der Leipzig University of Applied Sciences (HTWK) im Rahmen der QALD2017 Challenge veröffentlicht. Der Datensatz beinhaltet 894'499 Fragenpaare und wurde aus 5217 Vorlagen generiert.

Die Daten sind bereits Preprocessed und müssen deshalb in ihr Ursprungsformat konvertiert werden, so dass sie vom in dieser Arbeit verwendetem Preprocessing verwendet werden können. Zudem werden alle DESCRIBE-SPARQL-Abfragen entfernt, da dieser Abfragetyp im SQA2018 Datensatz nicht vorkommt. Schliesslich werden 669'737 Datenpaare verwendet.[2]

4. Experimente und Resultate

In diesem Kapitel werden die Ergebnisse vom NMT-Training der Daten, die vorher das Preprocessing durchlaufen haben, gezeigt. Es werden die Ergebnisse vom Baseline mit denen vom Named Entity Recognition (NER) verglichen. Zudem wird der Unterschied der Datenmenge betrachtet. Für die Ergebnisse der Datenmenge von 5000 werden die SQA2018 Daten verwendet für alle übrigen ein jeweils zufällig gewählter Teil der DBNQA Daten. Für das Baseline Training wird keine Attention und ein Dropout von 20% verwendet. Beim NER wird die Bahdanau Attention verwendet, sowie auch ein Dropout von 20%.

Die Metriken werden im Abschnitt 2.3 Performanz-Metriken erklärt.

Die genaue Durchführung und Erklärung der Experimente ist in der Bachelorarbeit[3] beschrieben. Die Baseline basiert dabei auf dem Standardmodell von

SPARQL as a Foreign Language, welches die Daten ohne Preprocessing entgegen nimmt und die Queries generiert. Beim Named Entity Recognition (NER) hingegen werden die Daten im Preprocessing in einem ersten Schritt für das Modell aufgearbeitet. Das Preprocessing nimmt dem Modell den Schritt der eigentlichen Entitäten Extrahierung aus den Datensätzen ab und ersetzt diese mit Variablen in den Queries. Dadurch werden die Abfragen aus den Daten generalisiert und zu Abfragen-Templates zusammengebaut. Das neuronale Netz muss somit keine Transformation von Ressourcen vornehmen und diese auch nicht im Vokabular aufnehmen. Dieser Schritt erfolgt mittels Spotlight.

4.1. Berechnung der Scores

Untersuchungen des Logs von der Evaluation und dem Skript, welches die Scores berechnet, haben gezeigt, dass sich die Scores nicht durch den Durchschnitt der einzelnen Scores ermittelt werden. Vielmehr wird für jede Metrik der Checkpoint gesucht, der den höchsten Wert hat und von dem wird der durchschnittliche F-Score berechnet mittels der Formel. $(Mikro - F - Score + Makro - F - Score)/2$

4.2. Resultate Baseline

Tabelle 4.1.: Resultate der Baseline abhängig von der Anzahl Datensätze

	Genauigkeit	**SPARQL-Genauigkeit**	**Mikro-F-Score**	**Makro-F-Score**
5'000	0.0	0.0405	0.0473	0.0369
50'000	0.0179	0.1321	0.1445	0.1445
100'000	0.1701	0.1701	0.1734	0.1701

Aus den Resultaten in der Tabelle 4.1 wird direkt ersichtlich, dass sich die Scores bei steigender Anzahl Datensätze verbessern. Auch wenn die Zunahme der einzelnen Scores bei steigender Anzahl Datensätze Schrittweise immer kleiner wird, gibt es eine steigende Tendenz nach oben. Daraus lässt sich schliessen, dass die Erhöhung der Datensätze einen positiven Effekt auf das Training hat.

Ein weiterer Punkt was aus den Resultaten ersichtlich wird, ist dass die Resultate im Vergleich zu den Versuchen aus der Bachelorarbeit [3] schlechter ausfallen. Da die Experimente auf der gleichen DBpedia-Version durchgeführt wurden, welcher bei der Bachelorarbeit ebenfalls verwendet wurde, für die Speicherung der Daten jedoch anstelle von Graph-DB Virtuoso verwendet wurde. Dies führt vermutlich dazu, dass einige Datensätze nicht in den eigentlichen Score einfliessen und somit zu einer Verschlechterung der Resultate führen.

4.3. Resultate NER

Tabelle 4.2.: SPARQL-Genauigkeit von der Baseline und dem NER

	Baseline	**NER**
5'000	0.0	3.0385
50'000	3.0711	N/A
100'000	5.3459	48.3792

Da die eigentliche Evaluation des NER fehlschlug, und somit keine Scores ermittelt werden, wurden die Scores händisch anhand vom Log der Evaluation ermittelt. Dies ist jedoch nur für die SPARQL-Genauigkeit möglich. Damit ein Vergleich mit der Baseline möglich ist, zeigt die Tabelle 4.2 den Vergleich der SPARQl-Genauigkeit zwischen der Baseline und dem NER mittels den händisch ermittelten SPARQL-Genauigkeit, welche in diesem Falle auch für die Baseline durchgeführt wird. Aus den Resultaten wird ersichtlich, dass die SPARQL-Genauigkeit bei beiden Varianten bei steigender Anzahl Datensätze zunimmt. Ebenfalls wird ersichtlich, das der

zusätzliche Schritt im Preprocessing für das NER eine Steigerung der SPARQL-Genauigkeit um das zehnfache erzeugt bei einer Anzahl von 100'000 Datensätzen. Um eine genauere Auswertung durchzuführen, müssen die Fehler in der Evaluation gefunden werden und in einem weiteren Schritt das NER nochmals evaluieren. Dieses Beispiel sollte jedoch aufzeigen, dass sich mittels der zusätzlichen Named Entity Recognition (NER) im Preprocessing die SPARQL-Genauigkeit steigern lässt.

5. Evaluation

In diesem Kapitel befindet sich eine detaillierte Beschreibung der NERP-Evaluation, die im Rahmen dieser Arbeit konzipiert und umgesetzt wird. Abschnitt 5.1 beschreibt den allgemeinen Zweck einer Evaluation des Preprocessing. In Abschnitt 5.2 ist das Konzept für eine Evaluation der Entities Extrahierung beschrieben. Das Konzept wird wie beschrieben umgesetzt und deren Ergebnisse sind in Abschnitt 5.3 dokumentiert.

5.1. Zweck

Das Preprocessing von Daten als Vorbereitung für das Trainieren der NMT-Modelle ist ein wichtiger Schritt, welcher die Resultate und Performanz der Modelle stark beeinflusst. Es ist daher erstrebenswert, dass das Preprocessing möglichst die korrekten Entities, Ontologies und Properties erkennt und extrahiert. Durch das Vergleichen der Resultate der NMT-Modelle, die auf Basis von verschiedenen Preprocessing-Pipelines trainiert werden, ist eine Bewertung der Qualität des Preprocessing teilweise möglich (vgl. [10]). Allerdings ist anhand dieser Resultate nur erkennbar, inwiefern die Pipelines einzeln oder kombiniert die Ergebnisse des NMT-Modells beeinflussen. Es ist auch essenziell, die für das Preprocessing verwendeten Methoden und Werkzeuge im Detail zu evaluieren und zu bewerten. Wenn beispielsweise die Effizienz und Genauigkeit von Werkzeugen für NER (wie Spotlight) bewertet werden sollen, ist es nötig zu erkennen, ob die korrekten Entities im Preprocessing extrahiert werden.

Im Rahmen dieser Arbeit wird eine Software (die *NERP-Evalution*) konzipiert und umgesetzt, welche die Entity Extrahierung des Preprocessing bewertet. Die Software analysiert die Ausgaben (respektive die Tokens für Entities) des Preprocessing. Die Analyse erzeugt ein Resultat, welches das Preprocessing anhand von verschiedenen Kriterien bewertet.

5.2. Konzept

5.2.1. Ein- und Ausgabe

Die NERP-Evaluation benötigt für die Durchführung Eingabedaten und liefert die entsprechende Bewertung als Ausgabe zurück.
Als Eingabe sind die folgenden Daten nötig:

NL-Frage Die ursprüngliche NL-Frage, welche im Preprocessing verarbeitet wurde.

Beispiel 5.2.1
how many teams have la kings players in them currently?

GT-Entities Eine Liste (Array) mit allen Entities, die in der Ground Truth (GT) Query verwendet werden. Die GT-Query ist die SPARQL-Query, die zu der NL-Frage gehört und für das Trainieren des NMT-Modells nötig ist. Die vollständige DBpedia-URI der GT-Entity muss mitgegeben werden.

Beispiel 5.2.2

```
['http://dbpedia.org/resource/Los_Angeles_Kings']
```

Preprocessing-Entities Eine Liste (Array) mit allen Entities, die im Preprocessing extrahiert werden. Die vollständige DBpedia-URI der Preprocessing-Entity muss mitgegeben werden.

Beispiel 5.2.3

```
['http://dbpedia.org/resource/Team','http://dbpedia.org/resource/
    Los_Angeles_Kings']
```

Die Ausgabe erfolgt im JSON-Format. Sie enthält die Bewertungen der Kriterien pro GT-Entity, die Zuordnung von Preprocessing-Entities zu GT-Entities und eine Liste von GT-Entities, die aus der NL-Frage nicht abgeleitet werden können. Im nächsten Abschnitt sind diese Resultate detaillierter beschrieben.

Abbildung 5.1 zeigt eine Übersicht der Ein- und Ausgabedaten.

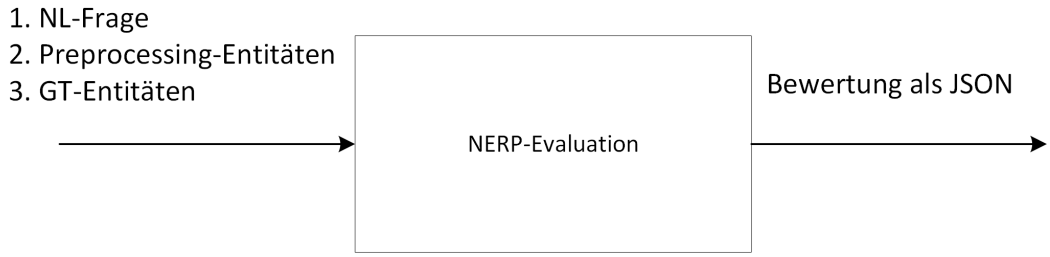


Abbildung 5.1.: Ein- und Ausgabedaten für die NERP-Evaluation.

5.2.2. Struktur der NERP-Evaluation

Die Bewertung der NERP-Evaluation erfolgt nach der *syntaktischen* und der *semantischen* Ähnlichkeit der GT-Entities und der Preprocessing-Entities. Da anhand der Eingabe nicht erkennbar ist, welche Preprocessing-Entity welcher GT-Entity zuzuordnen ist, wird in der Evaluation jede Preprocessing-Entity mit jeder GT-Entity nach jedem Kriterium verglichen. Mittels der Resultate ist es möglich festzustellen, welche Preprocessing-Entity welche GT-Entity repräsentiert. Die Bewertung erfolgt mit Werten zwischen 0.0 (0% Ähnlichkeitsgrad) und 1.0 (100% Ähnlichkeitsgrad, also identisch).

Nachdem die Bewertung durchgeführt wird, werden anhand der NL-Frage und der GT-Entities *Magic Entities* identifiziert und ausgegeben.

Das Ziel ist, dass die zurückgelieferte Ausgabe den Ähnlichkeitsgrad zwischen Entities ausdrückt. Die Bewertung der NERP-Evaluation wird verwendet, um allfällige Optimierungen der Entity Extrahierung des Preprocessing zu erkennen.

Die einzelnen Prozesse sind in den folgenden Abschnitten detaillierter beschrieben.

Semantische Ähnlichkeit von Entities (Entity Similarity Evaluation)
Die semantische Ähnlichkeit der Entities wird bewertet. Die Bewertung erfolgt mittels Word Embeddings. Dafür wird das Tool Magnitude verwendet, mit dem Modell "English Wikipedia 2017 + subword 16B Heavy".

Beispiel 5.2.4
GT-Entity: http://dbpedia.org/resource/Los_Angeles_Kings
Preprocessing-Entity: http://dbpedia.org/resource/Team
Word Embeddings von "Los_Angeles_Kings" und "Team". Der Score beträgt 0.319.

Semantische Ähnlichkeit von Entity-Typen (Type Similarity Evaluation)
Die Typen der Entities entsprechen den Oberbegriffen der Entities. Die semantische Prüfung der Typen hat zum Zweck, zu erkennen, ob zwei Entities Ähnlichkeiten

aufgrund ihrer Klassifizierung aufweisen, obwohl die semantische Ähnlichkeit im ersten Schritt ggf. tief bewertet wurde. Hierfür wird ebenfalls Magnitude verwendet, mit dem Modell "English Wikipedia 2017 + subword 16B Heavy".

Beispiel 5.2.5
GT-Entity: http://dbpedia.org/resource/Los_Angeles_Kings. Typ: hockey team
Preprocessing-Entity: http://dbpedia.org/resource/Team. Typ: band
Word Embeddings von "hockey team" und "band". Der Score beträgt 0.262.

Syntaktische Ähnlichkeit von Entities (Syntax Evaluation)
Zwei Entities sind syntaktisch ähnlich, wenn ein Teil der Wörter im Namen übereinstimmen. Eine komplette Übereinstimmung entspricht einem Exact Match. Die syntaktische Bewertung wird anhand einer *word-based Levenshtein-Distanz* durchgeführt (in Abschnitt 2.5 beschrieben). Vor der Bewertung der Levenshtein-Distanz werden die Satzzeichen in den Entities entfernt.

Bei der Berechnung der word-based Levenshtein-Distanz würden standardmässig Wörter als komplett unterschiedlich betrachtet werden, sobald sie sich um ein Zeichen unterscheiden. Für Fälle wie beispielsweise "House" und "Mouse" ist das optimal, da diese Wörter trotz dem kleinen Unterschied komplett verschiedene Bedeutungen haben. Bei Fällen wie "Hero" und "The_Marvel_Super_Heroes" handelt es sich aber bei "Heroes" und "Hero" um das gleiche Wort in einer anderen Form. Aus diesem Grund wird bei zwei Wörtern, die nicht übereinstimmen, zusätzlich die Wortstämme ermittelt und verglichen. Die Wortstämme werden mittels dem Porter2-Stemmer [19] der NLTK-Bibliothek [18] ermittelt. Falls die Wortstämme übereinstimmen, wird in diesem Fall eine kleinere Distanz verwendet, wie in 5.2.6 beschrieben.

Beispiel 5.2.6
GT-Entity: http://dbpedia.org/resource/The_Marvel_Super_Heroes
Preprocessing-Entity: http://dbpedia.org/resource/Hero
Die Levenshtein-Distanz beträgt in diesem Fall 3.8. Für das Einfügen der Wörter "The Marvel Super" beträgt die Distanz drei, und für das Editieren des Wortes "Hero" nach "Heroes" beträgt die Distanz statt eins 0.8, da die Wörter den gleichen Wortstamm besitzen.

Der syntaktische Score wird anhand der folgenden Formel 5.1 berechnet:

$$syntax\_score = (num\_unique\_words - levenshtein\_distance)/num\_unique\_words \tag{5.1}$$

Dabei entspricht *num_unique_words* der Anzahl einmaliger Wörter in beiden Entities. Falls in beiden Entities alle Wörter unterschiedlich sind und die Levenshtein-Distanz maximal ist (d.h. die Levenshtein-Distanz entspricht der Länge der Entity mit den meisten Wörtern), ist der Score 0.0.

Beispiele 5.2.7 und 5.2.8 zeigen diese beiden Fälle.

Beispiel 5.2.7
GT-Entity: http://dbpedia.org/resource/Burbank,_California
Preprocessing-Entity: http://dbpedia.org/resource/Luther_Burbank
num_unique_words: 3 (Luther, Burbank, California)
levenshtein_distance: 2
Damit entspricht der Score (3 - 2) / 3 = 0.33.

Beispiel 5.2.8
GT-Entity: http://dbpedia.org/resource/Burbank,_California
Preprocessing-Entity: http://dbpedia.org/resource/Distributor
num_unique_words: 3 (Distributor, Burbank, California)
levenshtein_distance: 2
Da keine Wörter übereinstimmen und die Levenshtein-Distanz maximal ist, wird der Score 0.0 zurückgegeben.

Best Matches bestimmen Anhand der Scores, die in den vorhergehend beschriebenen Bewertungen ermittelt wurden, wird nun pro GT-Entity die Preprocessing-Entity bestimmt, die am ähnlichsten ist, respektive anhand der ermittelten Bewertungen am besten passt. Dafür wird der "Best Score" basierend auf die Bewertungen der Preprocessing-Entitäten nach den beschriebenen Kriterien ermittelt. Falls die Evaluierungen einzeln durchgeführt werden, wird die Preprocessing-Entität gewählt, die den höchsten Score bei der nach dem betroffenen Kriterium erzielt hat. Bei der kombinierten Anwendung wird eine "Average Score" berechnet. Die Average Score pro Preprocessing-Entity entspricht dem Durchschnitt der Scores gemäss 5.2:

$$best\_score = (entity\_similarty\_score + type\_similarity\_score + syntax\_score)/3 \tag{5.2}$$

Bei einem "Exact Match" entsprechen alle drei Scores 1.0, damit ist der Durchschnitt ebenfalls 1.0. Pro GT-Entity wird nun die Preprocessing-Entity gewählt, die den höchsten Score (Best Score) bezüglich der GT-Entity hat. Wenn eine Preprocessing-Entity bei zwei GT-Entities der "Best Match" ist, d.h. bei beiden höhere Scores gegenüber den anderen Preprocessing-Entiäten erzielt hat, so wird sie der GT-Entity zugeordnet, wo der Best Score höher ist.

Zusätzlich ist eine Untergrenze definiert, die festlegt, welcher Best Score mindestens erreicht werden muss, damit eine Preprocessing-Entity überhaupt für die Zuordnung an eine GT-Entity berücksichtigt wird. Wenn diese Grenze unterschritten wird, wird die entsprechende Preprocessing-Entity als möglicher Match komplett ausgeschlossen, auch wenn es keine anderen Preprocessing-Entities gibt, die eine höhere Bewertung haben. Dieses Ausschliessen bewirkt, dass ggf. alle Preprocessing-Entities nicht in Frage kommen und es GT-Entities geben kann, die keinen Match haben.

Beispiel 5.2.9
GT-Entity: http://dbpedia.org/resource/Burbank,_California
Preprocessing-Entities und ihre Average Scores bei kombinierter Anwendung:
http://dbpedia.org/resource/Luther_Burbank: 0.39,
http://dbpedia.org/resource/California: 0.51
Der "Best Match" für "Burbank, California" ist "California", da diese Entity den höheren Score hat.

Magic Entities identifizieren
Magic Entities sind Entities, die nicht direkt aus der NL-Frage extrahiert werden können, da weder sie noch Synonyme oder Oberbegriffe in der Frage vorkommen. Ein Beispiel: Die Antwort auf die NL-Frage "Which river flows through Interlaken?" ist "Aare". Nun stehen die Entities "Aare" und "Interlaken" in keiner Beziehung zueinander, d.h. es kann kein RDF-Triple aus den beiden Entities gebildet werden, um die Frage zu beantworten. Aber die Aare steht in Verbindung zu "Bern", was wiederum in Verbindung zu "Interlaken" steht. D.h. es sind zwei RDF-Triples in der Form in Listing 5.1 nötig, um die Frage zu beantworten:

Listing 5.1: Beispiel Triples mit einer Magic Entity.

```
1  SELECT DISTINCT ?river WHERE {
2      dbr:Interlaken  dbo:canton dbr:Bern .
3      dbr:Bern dbo:lowestPlace ?river  .
4  }
```

Jedoch kommt weder der Begriff "Bern" noch ein Synonym oder Oberbegriff in der NL-Frage vor, weshalb die Entity auf eine andere Art im Preprocessing identifiziert werden muss. Da die Identifikation solcher Entities nicht mit den typischen Prozessen möglich ist, werden sie als *Magic Entities* bezeichnet.

Um zu erkennen, ob es sich bei einer GT-Entity um eine Magic Entity handelt, wird eine Übereinstimmung in der NL-Frage anhand der GT-Entity gesucht. Dabei wird die GT-Entity auf verschiedene Arten leicht modifiziert und pro Anpassung wird eine Übereinstimmung in der NL-Frage gesucht. Die Überprüfungen und Anpassungen sind wie folgt:

1. Gibt es eine exakte Übereinstimmung?
2. Gibt es eine Übereinstimmung, wenn Begriffe in Klammern entfernt werden? Begriffe in Klammern sind oftmals nur informativ und nicht zwingend nötig, um die Entity aus der Frage zu identifizieren.
3. Gibt es eine Übereinstimmung ohne Satzzeichen? Satzzeichen in der GT-Entity werden ganz entfernt oder durch Leerzeichen ersetzt.
4. Gibt es eine Übereinstimmung ohne Stopwords? Stopwords werden aus der NL-Frage und aus der GT-Entity entfernt
5. Gibt es in der NL-Frage eine Abkürzung, was für die Wörter in der GT-Entity stehen könnte? Aus den ersten Buchstaben der Wörtern der GT-Entity wird eine Abkürzung gebildet, die dann in der NL-Frage gesucht wird. Vor dieser Überprüfung werden Stopwords entfernt.

Wenn eine Frage in der Liste oben mit "ja" beantwortet werden kann, ist die GT-Entity keine Magic Entity. Zu beachten ist, dass die nachfolgenden Prüfungen nur auf Schritt zwei aufbauen. D.h. Begriffe in Klammern werden in Schritt zwei ganz entfernt und sind daher in den nachfolgenden Schritten ausgeschlossen. Die Änderungen bzw. Prüfungen, die in den Schritten drei bis fünf vorgenommen werden, sind unabhängig voneinander, resp. die Änderungen werden wieder zurückgesetzt, bevor mit dem nächsten Schritt fortgefahren wird.

Die Ausgabe dieses Prozessschrittes ist eine Liste mit GT-Entities, die Magic Entities sind, d.h. nicht aus der NL-Frage extrahierbar sind.

Spezialfälle: "Exact Matches"
Bei einem "Exact Match" stimmen die GT-Entity und die Preprocessing-Entity exakt überein. Ist dies der Fall, entspricht die Bewertung 1.0, ansonsten 0.0.

Beispiel 5.2.10
GT-Entity: http://dbpedia.org/resource/Los_Angeles_Kings
Preprocessing-Entity: http://dbpedia.org/resource/Los_Angeles_Kings
In diesem Fall beträgt der Score 1.0, da die Entities exakt gleich sind.

Beispiel 5.2.11
GT-Entity: http://dbpedia.org/resource/Los_Angeles_Kings
Preprocessing-Entity: http://dbpedia.org/resource/Team
In diesem Fall beträgt der Score 0.0, da die Entities nicht exakt gleich sind.

5.3. Ergebnisse

5.3.1. Analyse des Scoring NERP-Evaluation

Für das Testen werden die Resultate der Entity Extrahierung des Preprocessing der Bachelorarbeit [3] verwendet. Es werden für die allgemeine Bewertung der NERP-Evaluation 100 preprocessed NL-Fragen und SPARQL-Queries des SQA2018 Datensatzes verwendet. Auf diesen 100 Fragen und Queries erfolgt eine manuelle Prüfung, in der die GT-Entities mit den Preprocessing-Entities verglichen werden und in drei Kategorien eingeteilt werden:

1. **Exact Match**: Das Preprocessing ermittelt eine exakt gleiche Entity, die in der SPARQL-Query ebenfalls verwendet wird. Hier ist eine Zuordnung an die GT-Entity sinnvoll.
2. **Partial Match**: Das Preprocessing ermittelt eine Entity, die ähnlich zu einer GT-Entity ist und für die Beantwortung der Frage verwendet werden könnte (sofern es kein Exact Match gibt). Hier ist eine Zuordnung an die GT-Entity erwünschenswert.
3. **No Match**: Die Preprocessing-Entities, die stark von den GT-Entities abweichen und praktisch keine Korrelation haben. Diese Entitäten sollten keiner GT-Entität zugewiesen werden.

In Listing 5.2 ist beispielhaft aufgezeigt, wie Entities in diese drei Kategorien eingeordnet werden.

Listing 5.2: Beispiel Kategorisierung von Preprocessing-Entities basierend auf eine GT-Entity.

```
1 NL−Frage: count the wars in which people awarded with the croix de guerre fought.
2 GT−Entity: http://dbpedia.org/resource/Croix_de_guerre_1939−1945_(France) \\
3
4 Exact Match: http://dbpedia.org/resource/Croix_de_guerre_1939−1945_(France)− Die
      Preprocessing−Entity entspricht exakt der GT−Entity
5
6 Partial Match: http://dbpedia.org/resource/Croix_de_Guerre − Entspricht nahezu der
      GT−Entity, und könnte für die Beantwortung der NL−Frage ebenfalls verwendet
      werden, sofern eine korrekte Query erstellt werden würde (Diese muss nicht
      zwingend die gleiche Struktur haben, wie die GT−Query). In diesem Beispiel wü
      rden die Antworten nicht exakt gleich sein, wie wenn die Query mit der GT−Entity
      durchgeführt wird. Allerdings wären die Antworten trotzdem nach der Formulierung
      der Frage korrekt, da in der Frage nicht explizit nach dem Zeitraum 1939 − 1945
      gefragt wird.
7
8 No Match: http://dbpedia.org/resource/Count − Hat keinen bedeutsamen
      Zusammenhang mit dem Inhalt der Frage und beantwortet die Frage nicht oder nur
      über mehrere komplexe Triples.
```

Wie in Abbildung 5.2 illustriert, werden in den 100 SPARQL-Queries 147 GT-Entities verwendet. Von den insgesamt 360 Preprocessing-Entities sind 65 Exact Matches, 37 Partial Matches und 256 Non Matches. Das bedeutet, dass im Idealfall bei 104 GT-Entities ein Exact Match oder Partial Match gefunden wird, und bei 43 GT-Entities gar kein Match gefunden wird.

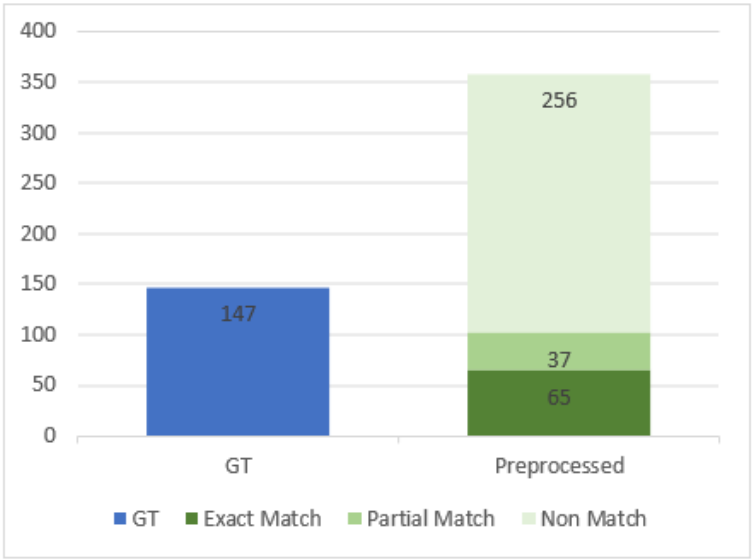


Abbildung 5.2.: Die Anzahl GT-Entities und die Anzahl Preprocessing-Entities. Die Preprocessing-Entities sind jeweils den Kategorien Exact Match, Partial Match und Non Match zugeordnet.

Die nachfolgenden Abschnitte beschreiben die Auswertungen der Entity Similarity, Type Similarity und Syntax Evaluation. Zuletzt wird die Anwendung von allen drei kombiniert ausgewertet. Die Auswertungen zeigen auf, wie präzise die NERP-Evaluation nach verschiedenen Kriterien ist. Die Genauigkeit wird daran bewertet, ob die NERP-Evaluation die Exact Matches und Partial Matches (falls eine GT-Entity kein Exact Match hat) den korrekten GT-Entities zuordnet, und ob solche GT-Entities, die keinen Match haben sollten, korrekt als solche erkannt werden.

Zudem wird analysiert, wie viele Matches fehlerhaft sind, d.h. wie oft ein Non Match einer GT-Entity zugeordnet wird. Ein Non Match wird einer GT-Entity zugeordnet, wenn er den höchsten Score aller Preprocessing-Entities erreicht und über die definierte Untergrenze liegt. Bei Erhöhung der Untergrenze fallen entsprechende Non Matches weg, dies hat allerdings auch eine Auswirkung auf die Zuordnung der Exact Matches und/ oder der Partial Matches bei anderen GT-Entitäten. Es ist möglich, dass gewisse Non Matches höhere Scores erzielen als die Partial Matches.

Bemerkung: Ein Non Match gilt in den Abbildungen als “korrekt zugeordnet”, wenn keine Zuordnung gemacht wird.

Entity Similarity Evaluation

Die Durchführung der Entity Similarity Evaluation ordnet alle Exact Matches korrekt den GT-Entities zu, unabhängig von der gesetzten Untergrenze für den Score. Sie haben alle einen Score von jeweils 1.0, was bedeutet, dass *Magnitude* erkennt, dass es sich um die identischen Entities handelt.

Die höchste Anzahl Partial Matches wird bei einer Untergrenze von 0.1 erreicht. In diesem Fall werden 18 von 37 korrekt zugeordnet. Bei den anderen 19 wird stattdessen eine Preprocessing-Entity zugeordnet, welche zu den Non Matches gehört, also keiner GT-Entity zugeordnet werden sollte.

In einem weiteren Schritt wird geprüft, welche Auswirkungen das Hochsetzen der Untergrenze auf die Zuordnung der Entities hat. Mit der Erhöhung der Untergrenze werden immer mehr Non Matches korrekt erkannt und als solche eingestuft. Zwischen 0.1 und 0.5 verbessert sich diese Erkennung insgesamt um 25 Non Matches. Allerdings nimmt die Anzahl Fehler bei der Zuordnung der Partial Matches zu. Das bedeutet, die Scores der Partial Matches sind niedrig und fallen deswegen unter die Untergrenze. Konkret betrifft das 3 Preprocessing-Entities, respektive es werden statt 48.4% noch 40.5% richtig zugeordnet. Abbildungen 5.3 und 5.4 zeigen diese Entwicklung auf.

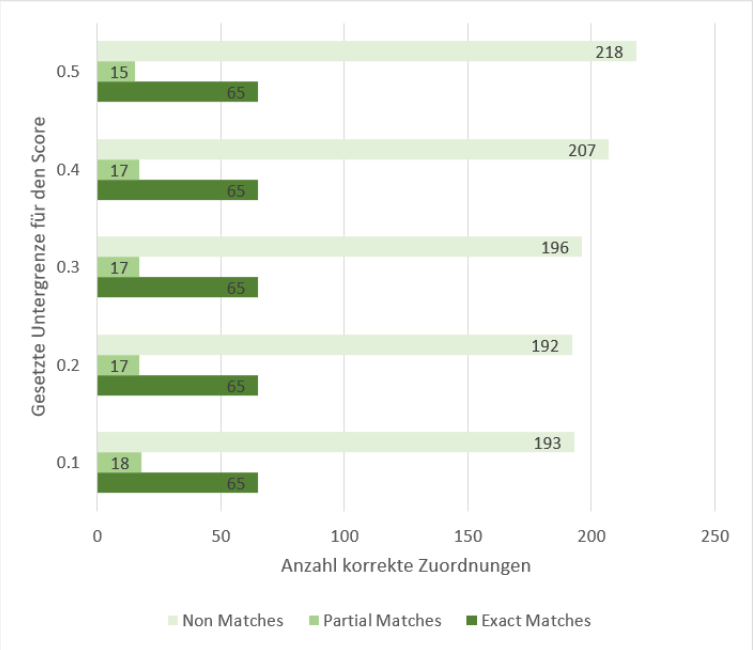


Abbildung 5.3.: Die Anzahl korrekt zugeordneter Preprocessing-Entities nach Resultat der Entity Similarity Evaluation, eingeteilt nach ihrer Kategorisierung und nach der gesetzten Untergrenze.

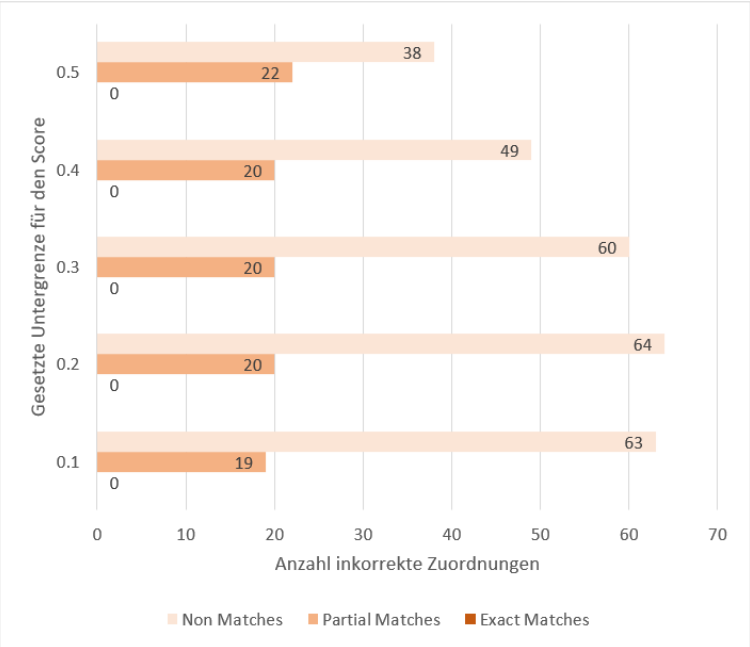


Abbildung 5.4.: Die Anzahl falsch zugeordneter Preprocessing-Entities nach Resultat der Entity Similarity Evaluation, eingeteilt nach ihrer Kategorisierung und nach der gesetzten Untergrenze.

Die Erhöhung der Untergrenze bewirkt auch, dass mehr GT-Entities gar keinen Match haben, was in Abbildung 5.5 ersichtlich ist. Diese Entwicklung ist positiv, sofern sie bewirkt, dass weniger Non Matches den GT-Entities zugeordnet werden. Bei dieser Evaluierung ist die Entwicklung positiv, sowie auch negativ. Zwar werden bei zunehmend mehr GT-Entities, die keinen Exact- oder Partial Match haben, korrekterweise keine Non Matches zugeordnet, allerdings werden wiederum bei mehr GT-Entities, die einen Match haben sollten, kein Match gefunden.

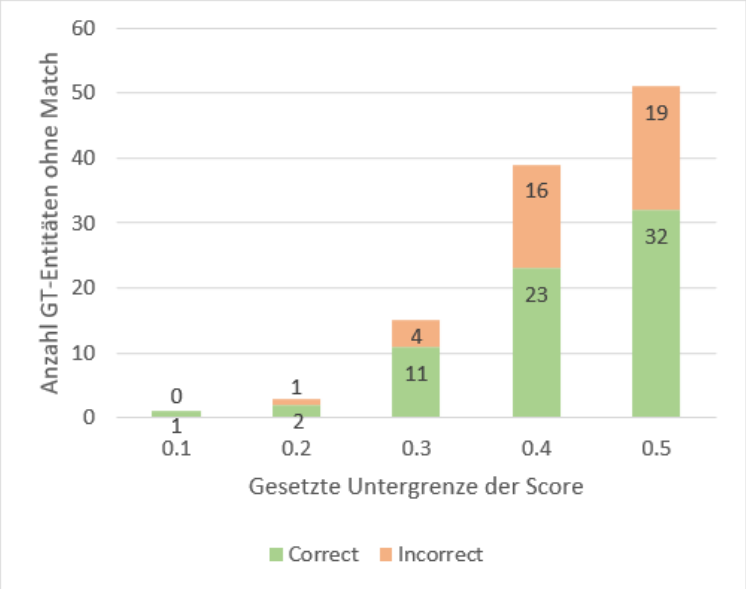


Abbildung 5.5.: Die Anzahl GT-Entities ohne Match nach der Entity Similarity Evaluation, basierend auf die steigende Untergrenze.

In Tabelle 5.1 ist der Median der korrekt zugeordneten Partial Matches und der zugeordneten Non Matches bei einer Untergrenze von 0.1 aufgezeigt. Die Gegenüberstellung der Werte zeigt, dass der Score sich um 0.16 jeweils unterscheidet.

Partial Match Median	**Non Match Median**
0.63	0.47

Tabelle 5.1.: Median Scores der zugeordneten Partial Matches und Non Matches der Entity Similarity Evaluation bei der Untergrenze 0.1

Nach manueller Analyse der Ausgaben der Entity Similarity lässt sich feststellen, dass gewisse Zuordnungen, die nicht auftreten sollten, hohe Scores (> 0.5) haben. Wiederum weisen Partial Matches teilweise niedrige (< 0.5) Scores auf. Die evaluierte Datenmenge ist zu klein um genau zu erkennen, bei welchen Arten von Entities dies vor allem auftritt. Eine Art, die sich hervorhebt, sind Entities, die einem Lied-, Bücher- oder Filmtitel entsprechen. Bei diesen ist eine Zuordnung schwierig, da sie meist aus mehreren Wörtern bestehen und grundsätzlich alle verwendeten Wörter berücksichtigt werden müssen.

Beispiel 5.3.1
GT-Entity: http://dbpedia.org/resource/He's_a_Liar
Zugeordnete Entity: http://dbpedia.org/resource/Liar_paradox
Score: 0.88

Weitere Entity: http://dbpedia.org/resource/Song
Score: 0.21
In diesem Beispiel handelt es sich bei “He’s a Liar” um ein Lied. Die Entity “Song” erhält ein sehr niedriger Score, während “Liar paradox” mit einem ein sehr hohen Score bewertet wird und dementsprechend als Match zugeordnet ist (obwohl wenig Ähnlichkeiten vorhanden sind).

Anhand der Ergebnisse ist für die Entity Similarity bei 0.5 die optimale Untergrenze. Gegenüber der kleinsten Untergrenze von 0.1 werden drei Partial Matches nicht mehr korrekt zugeordnet, während 25 mehr Non Matches korrekt als solche erkannt werden. Diese Untergrenze bewirkt auch, dass mehr GT-Entitäten, die einen (Partial) Match haben sollten, gar keinen Match haben. Da bereits bei der Untergrenze 0.1 für 19 GT-Entitäten nicht der entsprechende Partial Match gefunden wird, verschlechtert sich das Ergebnis nur minimal. Es wird einfach bei der höheren Untergrenze statt einen falschen Non Match zuzuordnen fälschlicherweise gar kein Match zugeordnet. Zudem wird bei 31 zusätzlichen GT-Entitäten richtig erkannt, dass es keinen Match für diese gibt.

Die Scores werden mittels *Magnitude* ermittelt und sind davon abhängig, welches Word Embedding Modell verwendet wird. Im Rahmen dieser Arbeit wird nur das Word Embedding Modell “English Wikipedia 2017 + subword 16B Heavy” getestet. Eine Evaluation von mehreren Modellen wäre aber ein weiterer Schritt, um gegebenenfalls ein Modell zu finden, das die Daten realistischer bewertet. Ein passenderes Modell bewirkt, dass die Partial Match Zuordnungen besser erkannt werden und damit ein deutlicherer Unterschied erkennbar ist, wenn die Untergrenze der Scores hochgesetzt wird.

Type Similarity Evaluation

Gegenüber der Entity Similarity Evaluation ist die Anzahl der korrekten Zuordnungen in der Type Evaluation allgemein kleiner. Bei der Type Evaluation werden die Typen der Entities statt die Entities selbst miteinander verglichen. Die Zuordnung ist also davon abhängig, wie ähnlich sich die *Typen* sind.

Im Gegensatz zu der Evaluierung der Entity Similarity werden bei der Evaluierung der Type Similarity nicht alle Exact Matches korrekt zugeordnet. Das liegt daran, dass mehrere Entities denselben Typ haben können. Das ist im Beispiel 5.3.2 genauer gezeigt.

Beispiel 5.3.2

Korrekte GT-Entity: http://dbpedia.org/resource/Ptolemy_XIII_Theos_Philopator
Falsche GT-Entity: http://dbpedia.org/resource/Cleopatra_V_of_Egypt
Exact Match Entity: http://dbpedia.org/resource/Ptolemy_XIII_Theos_Philopator
Score: 1.0
Obwohl die Exact Match Entity identisch ist zu der ersten GT-Entity, wird sie der Entity "Cleopatra V of Egypt" zugeordnet. Das passiert, weil beide Entities den Typ "Agent" haben und daher nicht mehr voneinander unterscheidbar sind, resp. beide gleich gut zu der Exact Match Entity passen.

Das Matching der Types ist unter anderem schwierig, da für manche Entities kein Typ gefunden wird. In diesen Fällen erhält die Entity den Typ "Thing", der zuoberst in der Typenhierarchie liegt. Sollte nun bei einer Preprocessing-Entity ebenfalls der Typ "Thing" ermittelt werden, dann werden diese einander zugeordnet, obwohl die Entities nur wenige oder keine Ähnlichkeiten haben.

Beispiel 5.3.3

GT-Entity: http://dbpedia.org/resource/He's_a_Liar
Zugeordnete Entity: http://dbpedia.org/resource/Song
Score: 1.0
Weitere Entity: http://dbpedia.org/resource/Liar_paradox
Score: 0.21
Im Gegensatz zu der Entity Simularity Evaluierung wird hier die Entity "Song" statt "Liar paradox" zugeordnet. "Song" und "He's a Liar" stimmen zu 100% überein, weil sie beide den Typ "Thing" erhalten haben.

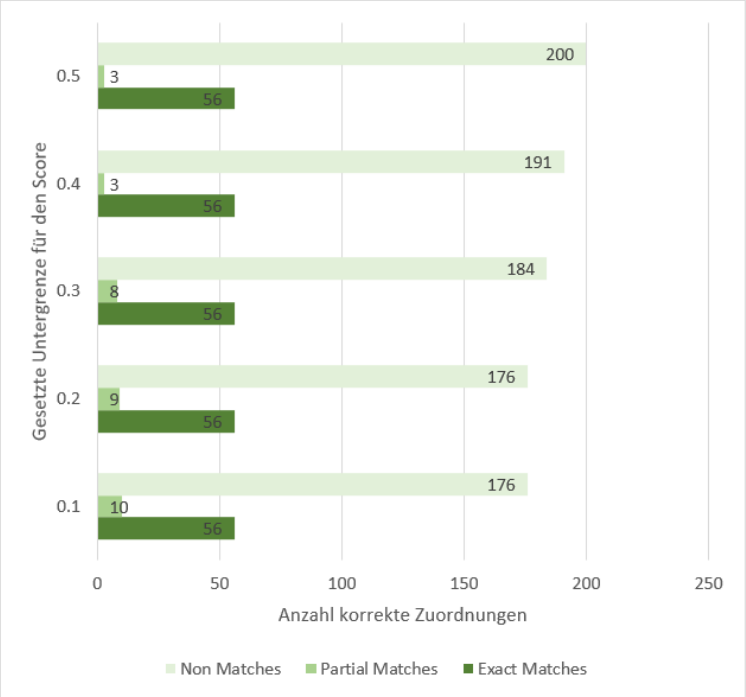


Abbildung 5.6.: Die Anzahl korrekt zugeordneter Preprocessing-Entities nach Resultat der Type Similarity Evaluation, eingeteilt nach ihrer Kategorisierung und nach der gesetzten Untergrenze.

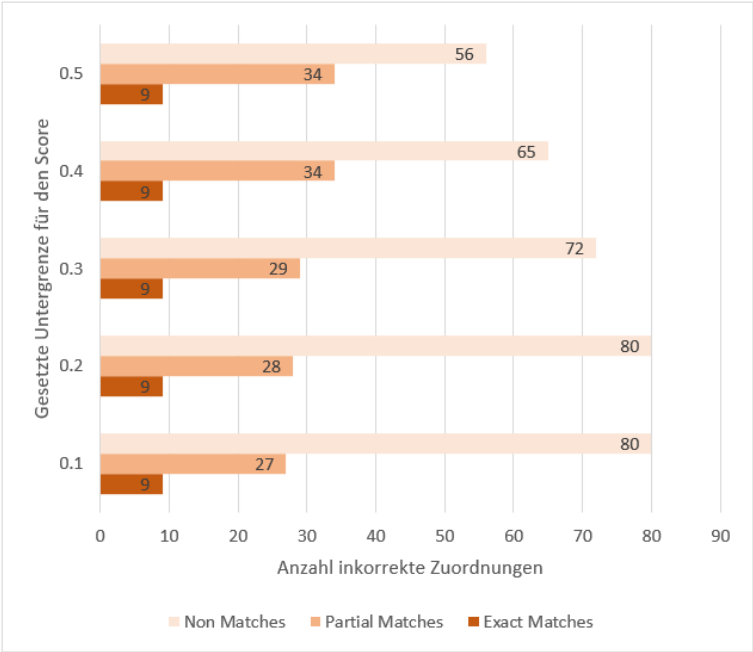


Abbildung 5.7.: Die Anzahl falsch zugeordneter Preprocessing-Entities nach Resultat der Type Similarity Evaluation, eingeteilt nach ihrer Kategorisierung und nach der gesetzten Untergrenze.

Dafür ist das Verhältnis der Zuordnung von keinen Matches zu GT-Entities bei einer Untergrenze von 0.5 kleiner gegenüber der Entity Similarity Evaluation, wie in wie in Abbildung 5.8 ersichtlich ist.

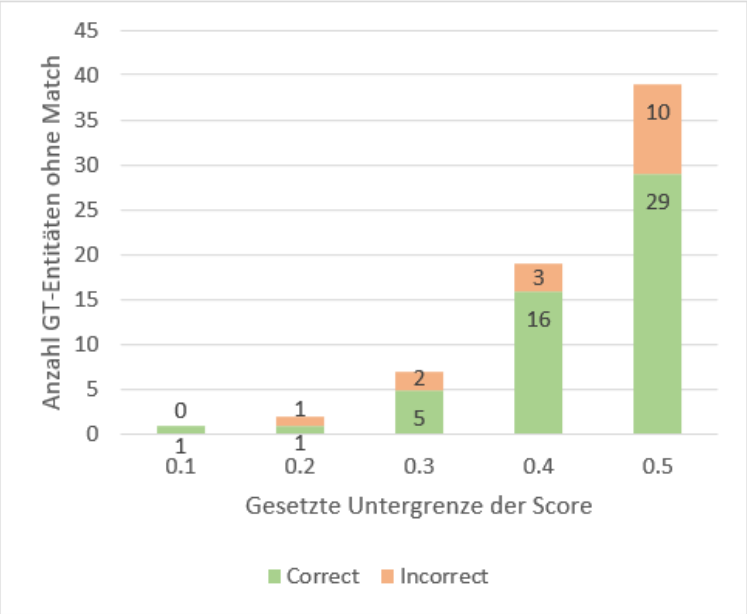


Abbildung 5.8.: Die Anzahl GT-Entities ohne Match nach der Type Similarity Evaluation, basierend auf die steigende Untergrenze.

Es werden nur 10 Partial Matches bei der niedrigsten Untergrenze korrekt zugeordnet. Zwischen den Untergrenzen 0.3 und 0.4 nimmt die Anzahl der korrekten Partial Matches um mehr als 50% ab, was bedeutet, dass die Mehrheit der Scores der korrekt zugeordneten Partial Matches unterhalb von 0.4 liegen. Hingegen liegt der Median Score der falsch zugeordneten Non Matches 0.11 höher. Dieser Unterschied ist auch in Tabelle 5.2 ersichtlich.

Tabelle 5.2.: Median Scores der korrekt zugeordneten Partial Matches und der falsch zugeordneten Non Matches der Type Similarity Evaluation bei der Untergrenze 0.1

Partial Match Median	**Non Match Median**
0.35	0.46

Zwar werden in der Type Evaluierung nicht alle Exact Matches erkannt, mit der steigenden Untergrenze bleiben die Anzahl Zuordnungen jedoch stabil. Der Sprung in der Anzahl korrekter Partial Matches zwischen 0.3 und 0.4 spricht dafür, die Untergrenze auf höchstens 0.3 zu setzen. Dafür werden aber bei der Untergrenze 0.5 bei 29 GT-Entitäten korrekt keine Matches zugeordnet.

Syntax Evaluation

In der Syntax Evaluation werden alle Exact Matches der GT-Entities korrekt erkannt und mit einer Score von 1.0 bewertet. Die Bewertung ist immer gleich, unabhängig von der gesetzten Untergrenze.

Bei den Partial Matches werden 17 von 37, also 45.9%, bei der niedrigsten Untergrenze korrekt zugeordnet. Zwischen den Untergrenzen 0.3 und 0.4 gibt es einen Sprung in der Anzahl korrekt zugeordneter Partial Matches, da vier Stück wegfallen.

Die Anzahl korrekt zugeordneter Non Matches, d.h. Non Matches, wo korrekt keine Zuordnung gemacht wird, steigt mit der Erhöhung der Untergrenze in fast gleich grossen Schritten an. Bei den Entity- und Type Similarity Evaluationen sind die Unterschiede zwischen den einzelnen Untergrenzen grösser.

Die Anzahl der korrekt zugeordneten Non Match Entities ist gegenüber der Entity Similarity Evaluierung und der Type Similarity Evaluierung bei einer Untergrenze von 0.1 um 27 Entities höher. Der Unterschied nimmt aber mit der steigenden Untergrenze ab und ist schliesslich identisch zu der Anzahl Non Match Entities der Entity Simularity Evaluierung (vgl. Abbildung 5.10).

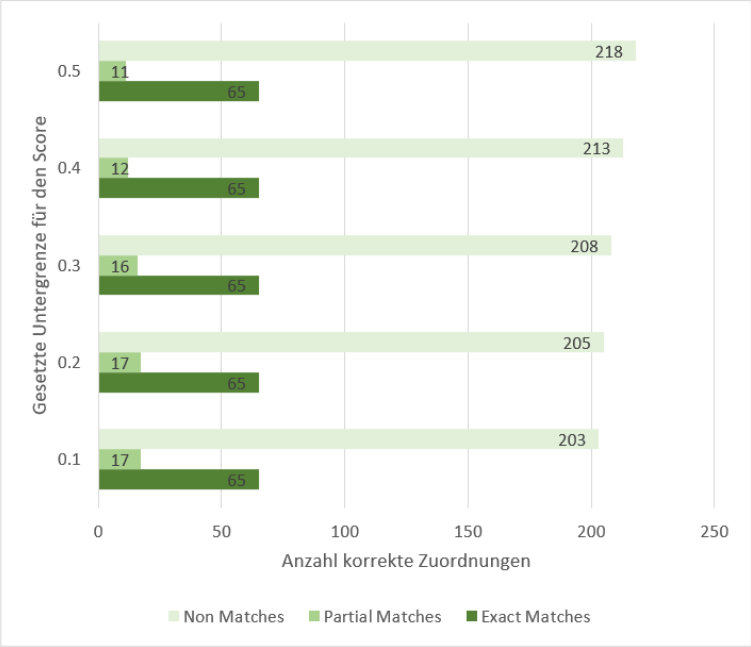


Abbildung 5.9.: Die Anzahl korrekt zugeordneter Preprocessing-Entities nach Resultat der Syntax Evaluation, eingeteilt nach ihrer Kategorisierung und nach der gesetzten Untergrenze.

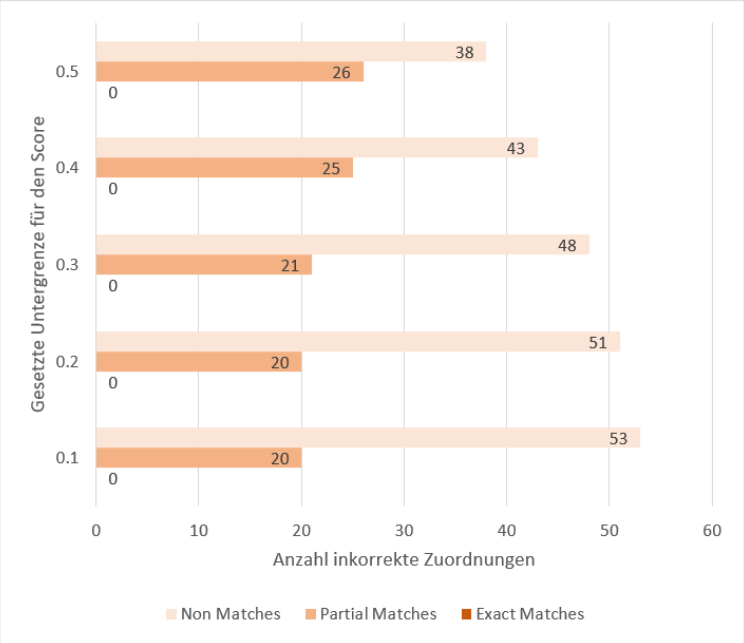


Abbildung 5.10.: Die Anzahl falsch zugeordneter Preprocessing-Entities nach Resultat der Syntax Evaluation, eingeteilt nach ihrer Kategorisierung und nach der gesetzten Untergrenze.

Die Anzahl GT-Entitäten ohne Match entwickelt sich ähnlich zu der Entity Similarity. In der Syntax Evaluation werden am meisten GT-Entitäten erkannt, die keinen Match haben sollten. Allerdings wird auch bei 14 GT-Entitäten, die einen Match haben sollten, fälschlicherweise kein Match zugeordnet.

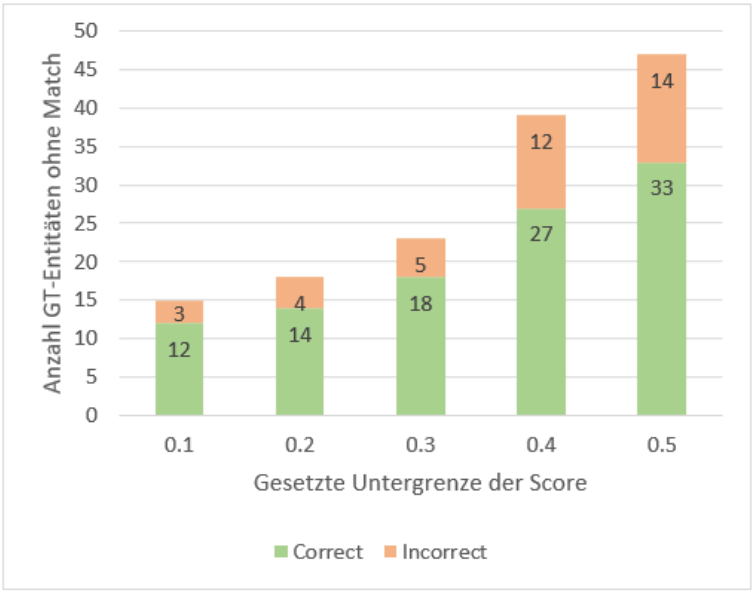


Abbildung 5.11.: Die Anzahl GT-Entities ohne Match nach der Syntax Evaluation, basierend auf die steigende Untergrenze.

Der Median der Partial Matches liegt bei 0.5. Diese Bewertung ist leicht verwirrend, da es bei der Untergrenze 0.3 einen Sprung in den zugeordneten Partial Matches gibt (vgl. Abbildung 5.9). Was dieser Median aber aussagt, ist, dass etwa die Hälfte der Partial Matches bei 0.5 oder höher liegen. Trotzdem werden bei einer Untergrenze von 0.5 11 Partial Matches korrekt erkannt, was nur 29.7% entspricht. Das bedeutet, dass es einige Non Matches gibt, die einen Score von 0.5 oder höher besitzen und daher statt die Partial Matches zugeordnet werden.

Tabelle 5.3.: Median Scores der korrekt zugeordneten Partial Matches und der falsch zugeordneten Non Matches der Syntax Evaluation bei der Untergrenze 0.1

Partial Match Median	**Non Match Median**
0.5	0.33

Bei der Syntax Evaluierung gibt es, ähnlich zu der Type Similarity Evaluierung, einen grösseren Sprung in den Partial Matches bei der Untergrenze 0.4. Es fallen vier Partial Matches weg, die zuvor korrekt erkannt wurden. Dafür gibt es ebenfalls einen Sprung bei den GT-Entitäten, wo korrekt keine Matches zugewiesen werden (vgl. Abbildung 5.11). Dies spricht wieder für die höchste Untergrenze 0.5, wo die meisten GT-Entitäten, die keinen Match haben, korrekt als solche erkannt werden.

Alle kombiniert

Alle drei Verfahren werden kombiniert angewendet. Pro Preprocessing-Entity wird je für Entity Simularity, Type Simularity und Syntax eine Score berechnet. Die Zuordnung der Preprocessing-Entity zu einer GT-Entity erfolgt anhand des durchschnittlichen Scores, welcher aus der Summe aller Scores durch drei geteilt ermittelt wird. Alle drei Scores sind gleich gewichtet.

Die Resultate der kombinierten Anwendung zeigen gegenüber den anderen Verfahren keine offensichtliche Verbesserungen auf. Die Ergebnisse sind ähnlich zu denen der Entity Similarity Evaluation und der Syntax Evaluation. Die Exact Matches werden alle korrekt zugeordnet, unabhängig von der gesetzten Untergrenze. Die korrekten Partial Matches sind anfangs bei fast der Hälfte, analog zu der Entity Similarity Evaluation, nehmen dann aber bei einer höheren Untergrenze stark ab. Bei der Untergrenze 0.5 werden nur noch 8 der Partial Matches korrekt identifiziert. Zudem gibt es bei dieser Untergrenze einen auffälligen Sprung in der Anzahl korrekt erkannter Partial Matches.

Das korrekte Erkennen von Non Matches entwickelt sich ähnlich zu der Entity Similarity, wobei Zwischenschritte in etwa gleich gross sind, was bei der Entity Similarity nicht der Fall ist.

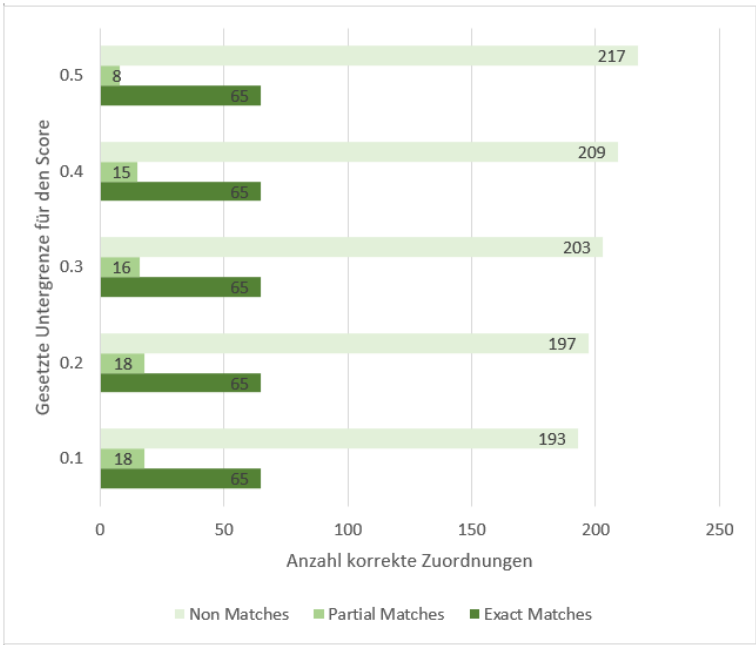


Abbildung 5.12.: Die Anzahl korrekt zugeordneter Preprocessing-Entities nach Resultat der kombinierten Evaluation, eingeteilt nach ihrer Kategorisierung und nach der gesetzten Untergrenze.

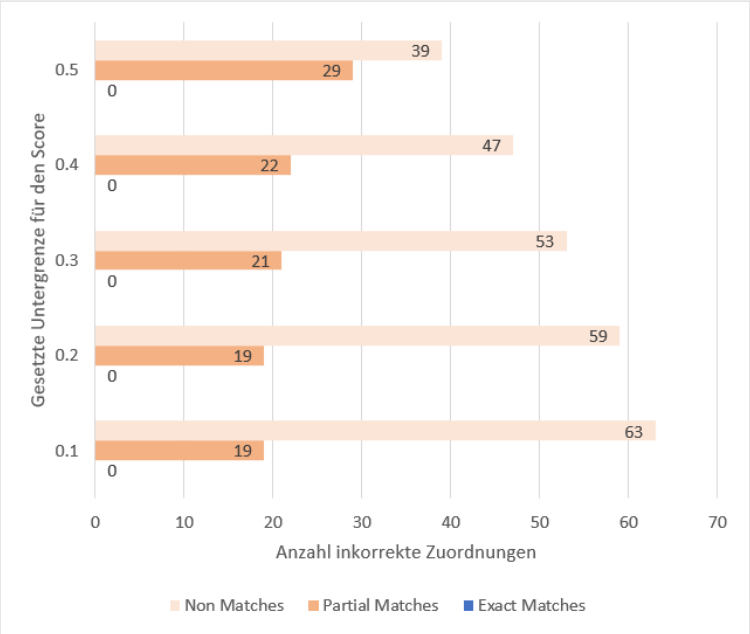


Abbildung 5.13.: Die Anzahl falsch zugeordneter Preprocessing-Entities nach Resultat der kombinierten Evaluation, eingeteilt nach ihrer Kategorisierung und nach der gesetzten Untergrenze.

In der kombinierten Anwendung der Evaluationen wird bei der Untergrenze 0.5 die meisten GT-Entities erkannt, die keinen Match haben sollten, nämlich 35 von 43 resp. 81.4%. Dafür werden auch von allen Evaluationen am meisten den GT-Entities fälschlicherweise keine Matches zugeordnet.

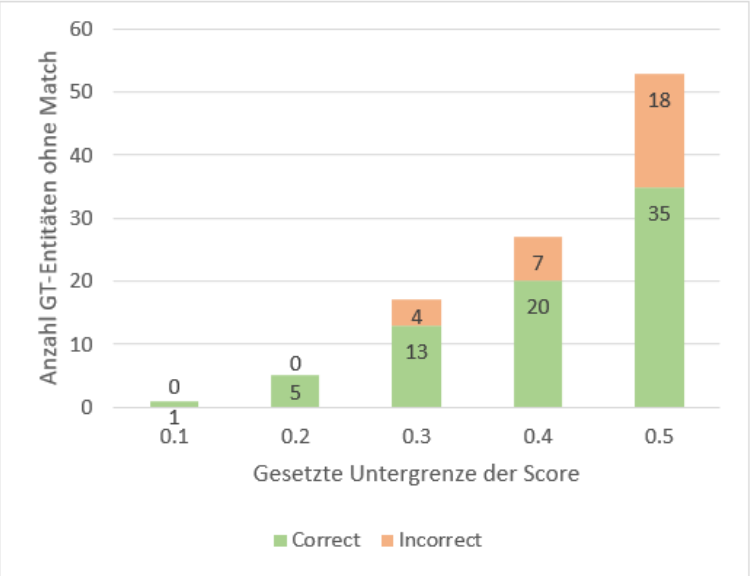


Abbildung 5.14.: Die Anzahl GT-Entities ohne Match nach der kombinierten Evaluation, basierend auf die steigende Untergrenze.

Tabelle 5.4 zeigt, dass der Median der Partial Matches und der Median der Non Matches, die falsch zugeordnet werden, nahe beieinander liegen. Trotzdem werden bei einer Untergrenze von 0.5 nur 8 Partial Matches korrekt erkannt, was 21.6% entspricht.

Tabelle 5.4.: Median Scores der korrekt zugeordneten Partial Matches und der falsch zugeordneten Non Matches der kombinierten Evaluation bei der Untergrenze 0.1

Partial Match Median	**Non Match Median**
0.486	0.45

Bei der kombinierten Evaluierung gibt es ebenfalls einen auffallenden Sprung in den Partial Matches, nämlich bei 0.5. Allerdings bringt diese Evaluierung die besten Ergebnisse bezüglich der Erkennung von GT-Entitäten ohne Matches. Betrachtet auf die Anzahl korrekter Zuordnungen an GT-Entitäten würde sich die Untergrenze von 0.5 am besten anbieten. Darauf wird in der Zusammenfassung näher eingegangen.

5.3.2. Zusammenfassung

Tabelle 5.5 zeigt eine Übersicht der korrekt erkannten Exact Matches auf. Bei den Entity Similarity, Syntax und kombinierten Evaluationen werden unabhängig von der gesetzten Untergrenze immer alle Exact Matches korrekt erkannt. Bei der Type Similarity hingegen werden immer neun Exact Matches nicht erkannt.

Tabelle 5.5.: Vergleich der korrekt erkannten Exact Matches

	Lower Bound				
	0.1	**0.2**	**0.3**	**0.4**	**0.5**
Entity Similarity	65	65	65	65	65
Type Similarity	59	56	56	56	56
Syntax	65	65	65	65	65
Combined	65	65	65	65	65

Der Vergleich der korrekten Partial Matches in Tabelle 5.6 zeigt, dass die Entity Similarity die besten Ergebnisse erzeugt, da die Anzahl der Partial Matches mit der Erhöhung der Untergrenze nur wenig zurückgeht.

Tabelle 5.6.: Vergleich der korrekt erkannten Partial Matches

	Lower Bound				
	0.1	**0.2**	**0.3**	**0.4**	**0.5**
Entity Similarity	18	17	17	17	15
Type Similarity	10	9	8	3	3
Syntax	17	17	16	12	11
Combined	18	18	16	15	8

Bei der korrekten Erkennung von Non Matches, also dass Non Match Entities keiner GT-Entity zugeordnet werden, ist das erzielte Ergebnis der Type Similarity Evaluation am tiefsten. Wie in 5.7 ersichtlich weisen die anderen drei Evaluationen ähnliche Ergebnisse auf.

Tabelle 5.7.: Vergleich der korrekt erkannten Non Matches

	Lower Bound				
	0.1	**0.2**	**0.3**	**0.4**	**0.5**
Entity Similarity	193	192	196	207	218
Type Similarity	176	176	184	191	200
Syntax	203	205	208	213	218
Combined	193	197	203	209	217

Tabelle 5.8 zeigt den Vergleich der korrekten und inkorrekten Non Matches der Evaluationen. Die Syntax Evaluation erkennt bei der niedrigsten Untergrenze die

meisten GT-Entities, die keinen Match haben. Bei der höchsten Untergrenze erreicht aber die Entity Similarity Evaluation ähnliche Ergebnisse.

Tabelle 5.8.: Vergleich der Anzahl GT-Entities mit der korrekten oder falschen Zuordnung von "None", also kein Match.

	Lower Bound									
	0.1		0.2		0.3		0.4		0.5	
	Correct	Incorrect	Correct	Incorrect	Correct	Incorrect	Correct	Incorrect	Correct	Incorrect
Entity Similarity	1	0	1	1	5	2	16	3	29	10
Type Similarity	1	0	2	1	11	4	23	16	32	19
Syntax	12	3	14	4	18	5	27	12	33	14
Combined	1	0	5	0	13	4	20	7	35	18

Anhand der Tabellen oben ist es nicht direkt ersichtlich, welche Evaluationen insgesamt die besten Ergebnisse erzielen. Da das Ziel ist, bei der Evaluation möglichst viele korrekte Zuweisungen der GT-Entitäten zu machen, ist in der Tabelle 5.9 dargestellt, wie viel Prozent der GT-Entities korrekte Zuweisungen erhalten haben. D.h. die Anzahl korrekter Partial Matches und Exact Matches und die Anzahl GT-Entitäten, bei denen korrekt erkannt wird, das keine Zuweisung existiert, werden aufsummiert und in Relation zu der Gesamtanzahl GT-Entitäten gesetzt.

Diese Übersicht zeigt, dass die Entity Similarity und Syntax Evaluationen mit einer Untergrenze von 0.5 die höchste Anzahl korrekte Zuweisungen der GT-Entitäten erreichen. Anzumerken ist, dass die GT-Entitäten mit Partial Matches gleich gewichtet werden wie die GT-Entitäten ohne Matches. Zudem ist anzumerken, dass wenn die Untergrenze weiter hochgesetzt wird, alle GT-Entitäten ohne Match definitiv erkannt werden.

Tabelle 5.9.: Übersicht der Anzahl korrekter Zuweisungen der GT-Entities, in Prozent ausgedrückt

	Lower Bound				
	0.1	**0.2**	**0.3**	**0.4**	**0.5**
Entity Similarity	57.10%	55.80%	57.80%	67.30%	74.10%
Type Similarity	45.60%	45.60%	51.00%	55.80%	61.90%
Syntax	63.90%	65.30%	67.30%	70.70%	74.10%
Combined	57.10%	59.90%	63.90%	68.00%	73.50%

Nach der vorhergehenden Auswertung hat die kombinierte Anwendung der Evaluationen zu keiner Verbesserung geführt. Um eine optimale Kombination der Evaluationsmethoden zu finden, ist es sinnvoll, weitere Analysen mit unterschiedlichen Gewichtungen auszuführen. Eine weitere Möglichkeit ist, die Evaluationen statt kombiniert nacheinander auszuführen. Beispielsweise würde Anfangs die Entity Similarity Evaluation ausgeführt werden und gewisse Zuordnungen bereits erkennen. Alle GT-Entities, für die keine optimale Zuordnung gefunden werden konnte, würden im nächsten Prozessschritt erneut evaluiert werden. Für ein solches Vorgehen muss

die Reihenfolge der Evaluationen geprüft werden, d.h. in welcher Reihenfolge die Evaluationen die besten Ergebnisse erzeugen.

Ein wichtiger Fokus für die Weiterentwicklung dieses Konzeptes ist die Evaluationen zu verfeinern oder zusätzliche Evaluationen durchzuführen, damit Partial Matches besser erkannt werden, resp. diese besser von Non Matches unterschieden werden können. In der aktuellen Umsetzung sind die Partial Matches fast nicht von den Non Matches unterscheidbar, da ihre Scores nahe beieinander liegen, und weil sehr viele Non Matches existieren.

Zudem ist für eine genauere Auswertung der NERP-Evaluierung die Ausführung mit mehr Daten nötig. Allerdings verursacht eine umfassende Auswertung mit einer grösseren Datenmenge einen hohen Aufwand, da Alternativen zu den GT-Entities (Partial Matches) vorab gesammelt und vorbereitet werden müssen. Es müsste eine Liste geben, wo alle Entitäten aufgelistet werden, die für eine NL-Frage genutzt werden können.

Da die Type Evaluation Exact Matches nicht immer erkennt, wird aufgrund der Resultate dieser Analyse vor der Ausführung der Berechnungen direkt geprüft, ob zwei Entities identisch sind. Ist dies der Fall, wird direkt der Score 1.0 für die entsprechende Evaluation gesetzt. Diese direkte Prüfung erfolgt auch bei der Entity Similarity und der Syntax Evaluation, bevor die eigentlichen Analysen ausgeführt werden.

5.3.3. Analyse der Erkennung von Magic Entities

Die detaillierte Analyse der Ergebnisse der Erkennung der Magic Entities wird ebenfalls auf eine eingeschränkte Menge von 100 Fragen ausgeführt. Abbildung 5.15 zeigt die Anzahl erkannter Magic Entities pro zusätzlicher Prüfung. Dabei bauen die Funktionen in der horizontalen Achse aufeinander auf, d.h. die erste Funktion prüft nur "Exact Matches" (wenn die GT-Entity ohne Veränderungen in der NL-Frage gefunden wird). Im nächsten Schritt werden zusätzlich die Klammern und ihre Werte im Label entfernt, wenn kein Exact Match gefunden wird. Zuletzt werden alle Funktionen angewendet. Für die 100 Fragen existieren keine Magic Entities, d.h. alle 147 GT-Entities sollten aus der NL-Frage extrahiert werden können. Im Idealfall würden die angewendeten Funktionen also 0 Magic Entities erkennen.

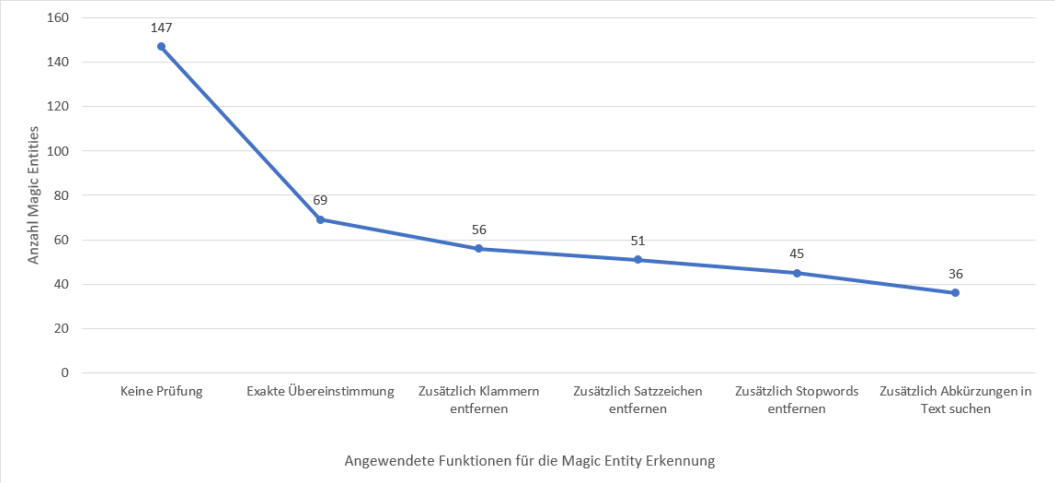


Abbildung 5.15.: Anzahl identifizierter Magic Entities bei den 100 Queries

Mehr als 50% der GT-Entities haben in der NL-Frage eine direkte Übereinstimmung, d.h. ihr Label muss nicht angepasst werden, um einen Match zu finden. Das Entfernen von Klammern (die grundsätzlch rein informativ sind) erkennt weitere 8.8% der Entities. Bei den angewendeten Daten bringt das Entfernen von Satzzeichen (wie ‚-':.) den geringsten Gewinn, es werden nur zusätzlich 3.4% der Entities erkannt. Einen grösseren Gewinn bringt hingegen wieder die letzte Funktion, wo Abkürzungen der Entity im Text der NL-Frage gesucht werden. Es können weitere 6.1% erkannt werden.

Mit diesen Schritten ist es möglich, 111 resp. 75.5% GT-Entities in der NL-Frage zu identifizieren. Es müssten damit noch 36 zusätzlich identifiziert werden. Die restlichen GT-Entities werden aufgrund von unterschiedlichen Ursachen nicht identifiziert. Ein Grund ist, dass Schreibfehler nicht berücksichtigt werden, die in der

NL-Frage vorkommen. Zusätzlich werden Entities auch nicht erkannt, wenn das Label aus zusätzlichen Wörtern oder Informationen (z.B. Jahreszahlen) besteht. Die nachfolgenden Beispiele zeigen zwei solche Fälle auf.

Beispiel 5.3.4
NL-Frage: list the magazines published by playbow enterprises ?
GT-Entity: "Playboy Enterprises"
Aufgrund des markierten Tippfehlers wird die Entity in der Frage nicht erkannt und deshalb als Magic Entity eingestuft.

Beispiel 5.3.5
NL-Frage: who was buried at st. mary's church and fought the battle of benedict arnold's expedition to quebec ?
GT-Entity: "St Mary's Church, Battersea"
Da das markierte Wort in der NL-Frage nicht vor kommt, wird die Entity in der Frage nicht erkannt und deshalb als Magic Entity eingestuft.

Um alle GT-Entities erkennen zu können, sind zusätzliche Überprüfungen nötig. Neben den erwähnten Funktionen, wo Schreibfehler berücksichtigt werden sollten und ebenfalls die Platzierung resp. Verwendung der Wörter, ist auch sinnvoll, die Suche nach Abkürzungen zu erweitern, so dass aus Abkürzungen in den Labels der Entities die zugehörigen Wörter in der NL-Frage gegebenenfalls gefunden werden. Zudem werden Abkürzungen nur aus Anfangsbuchstaben der Wörter gebildet, und nur für alle Wörter eines Labels. Bei Personennamen kann es aber vorkommen, dass beispielsweise nur der zweite Vornamen abgekürzt wird, d.h. nur der erste Buchstaben geschrieben wird. Solche Fälle werden in der aktuellen Funktion nicht berücksichtigt.

6. Diskussion und Ausblick

6.1. Allgemein

Das Ziel dieser Arbeit war es einerseits, ein Evaluationstool zu entwicklen, welches das Entity-Linking aus dem Preprocessing evaluiert. Es wurde dazu ein Konzept entworfen, sowie auch das Tool anhand von einer begrenzten Menge Testdaten evaluiert und getestet, welches im Kapitel 5 beschrieben wird. Anderseits war ein weiteres Ziel der Arbeit, das Modell mittels der Baseline und dem Named Entity Recognition (NER) zu trainieren und die Resultate daraus zu evaluieren. Weiteres dazu ist im Kapitel 4 ersichtlich.

6.2. Preprocessing

Da bei der Entitäten Extrahierung das öffentliche Spotlight verwendet wurde, ist die Laufzeit, insbesondere bei vielen Datensätzen, sehr langsam. Dies kann durch ein lokales Spotlight stark verbessert werden, wurde bei dem Preprocessing jedoch nicht implementiert.

Bei der Property Extrahierung könnte man, wenn die Ontology "bekannt" ist, sich auf die Property in der Ontology beschränken, weil nur die von der Ontology vorgegeben Eigenschaften verwendet werden sollten.

In der Entity Extrahierung werden mittels der aktuellen Confidence Score von 0.1 sehr viele überflüssige Entitäten extrahiert. Die Confidence Score wurde bereits im Rahmen der Bachelorarbeit [3] in Detail evaluiert und aufgrund der Ergebnisse so gesetzt. Um die Anzahl der überflüssigen Entitäten zu reduzieren, könnte zusätzlich pos-Tagging angewendet werden, wie bereits in [3] erwähnt. Mittels POS-Tagging ist es möglich, Nomen und Named Entities zu erkennen und daher solche Entitäten auszuschliessen, die auf andere Wortarten wie Verben oder Adjektiven basieren.

Weitere mögliche Optimierungen für das Preprocessing der Bachelorarbeit [3] sind in der Arbeit genannt.

6.3. Evaluation

Bei der NERP-Evaluation wurde für das Word Embedding das Tool Magnitude verwendet. Dieses bietet verschiedene Modelle an, welche jedoch nicht auf DBpedia trainiert wurden und deshalb nur bedingt geeignet sind. Für die NERP-Evaluation wäre es von Vorteil, ein Word Embedding zu nutzen, welches auf DBpedia trainiert wurde.

Mittels der aktuellen Umsetzung der NERP-Evaluation ist das Erkennen von Partial Matches nur bedingt möglich. Das ist, weil eine klare Grenze zwischen Partial Matches und Non Matches nicht existiert, d.h. einige Non Matches werden gleich oder höher bewertet als die Partial Matches. Sie können daher in keiner der Evaluationen eindeutig voneinander unterschieden werden. Es ist daher empfehlenswert, die Evaluationen weiter zu verfeinern, in dem zum Beispiel ein passenderes Modell für das Word Embedding verwendet wird (im letzten Paragraph bereits erwähnt). Auch die Syntax Evaluation könnte verfeinert werden, in dem Abkürzungen (zum Beispiel bei zweiten Vornamen) berücksichtigt werden. Weiter kann die Kombination, Reihenfolge und Gewichtung der einzelnen Evaluationen angepasst und analysiert werden.

Eine weitere Möglichkeit ist, zusätzliche Evaluationen durchzuführen. Um Non Matches auszuschliessen, könnte mittels einem POS-Tagger die NL-Frage analysiert werden. Wenn Wörter, woraus die Preprocessing-Entity abgeleitet wurde, keine Nomen oder Named Entities sind, dann können diese Preprocessing-Entities von der Evaluation ausgeschlossen werden.

Zudem wird bei der NERP-Evaluation nur die Entity Extrahierung betrachtet. Dies könnte mit der Evaluierung der Ontology Extrahierung und der Property Extrahierung erweitert werden. Eine weitere mögliche Prüfung wäre, die Result Set Accuracy zu messen. Diese Messung würde anhand von den Entities, Ontologies und Properties erfolgen, die gemäss der Evaluation am meisten geeignet sind, um eine NL-Frage zu beantworten.

6.4. Zielerreichung

Leider konnte nicht der ganze DBNQA Datensatz von 640'000 Daten evaluiert werden, weil dies mehrere Wochen Zeit beanspruchen würde. Des weiteren ist die Evaluation des Named Entity Recognition (NER) unvollständig. Ursache dafür ist, dass bei der Evaluation des Modells keine F-Scores ermittelt werden konnten. Das Problem

muss mittels weiteren Evaluationen untersucht werden, um die eigentliche Fehlerquelle dazu zu finden. Ein grundlegendes Problem, das aus den Ergebnissen erkennbar ist, ist dass keine Werte der Konfusionsmatrix (True Positive, False Positive, etc.) ermittelt werden konnten. Diese werden für die Berechnung der Ausbeute und der Präzision benötigt, welche für den F-Score notwendig sind.

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

A. Anhang

A.1. Glossar

anaconda *Anaconda* is a software which makes it easier to manage python packages. iv

ngram Ein ngram ist ein Fragment eines Textes mit der Länge n. Bei einem Monogramm (1-gram) besteht das Fragment aus einem Wort, bei Bigramm (2-gram) aus zwei und bei Trigramm (3-gram) aus drei. 21, 24

pip *pip* is the package manager from python, to download and manage different python packages. iv

Semantic Web Fügt einem Begriff eine semantisch Bedeutung zu, so dass dies von einer Maschine einfacher gelesen werden kann. 4

SPARQL-Template Vorlage für eine SPARQL-Query. Enthält Platzhalter für URI's (Entitäten, Eigenschaften und Ontologien), die je nach NL-Frage mit entsprechenden Daten ersetzt werden. 14

A.2. Akronym Glossar

API Application Programming Interface 13, 18

DBNQA DBpedia Neural Question Answering 27, 28, 60

GT Ground Truth 32, 33, 35–44, 47–58, 62–65

NER Named Entity Recognition 13, 18, 28, 30, 31, 59, 60

NERP NER-Preprocessing 31–33, 38, 40, 56, 60

NL Natural Language vii, ix–xi, 14, 15, 17–27, 32, 33, 36–38, 56–58, 60, 65

NMT Neural Machine Translation ix, 2, 14, 15, 28, 31, 32

POS Part-Of-Speech viii, 12, 13, 21, 23, 59, 60, 62, 64

RDF Resource Description Framework 4, 5, 20, 36

SPARQL SPARQL Protocol And RDF Query Language vii, ix–xi, 4, 5, 8, 14, 15, 17–20, 22–27, 32, 38, 39, 65

SQA2018 Scalable Question Answering 2018 27, 28

URI Uniform Resource Identifier x, 4, 6, 14–18, 21–25, 32

W3C World Wide Web Consortium 5

B. Technische Dokumentation

Dieses Kapitel beschreibt, wie die nötige Infrastruktur für die Anwendung NL2CL in Betrieb genommen wird, sowie wie das Preprocessing der Anwendung ausgeführt wird.

B.1. Vorbereitungen

B.1.1. Hardware-Voraussetzungen

Um das Training zu beschleunigen, ist ein Hardware-System mit einer Nvidia-Grafikkarte empfehlenswert. Falls eine Grafikkarte von Nvidia verwendet wird, muss die folgende Software installiert werden:

- Nvidia GPU-Treiber für die entsprechende Grafikkarte
- Nvidia Cuda 9.0
- Nvidia Cudnn 7.5

Als Betriebssystem ist Ubuntu 16.04 empfehlenswert. Die beschriebenen Docker-Images in Abschnitt **docker** verwenden diese Ubuntu Version.

B.1.2. Software-Voraussetzungen

Um die Software ausführen zu können, muss Python 3.5.2 installiert werden. Zudem sind mehrere Python-Pakete nötig. Die nachfolgende Liste aus [3] enthält alle nötigen Pakete inklusive ihrer Versionen.

- keras 2.1.5
- tensorflow-gpu 1.6.0 wenn eine Grafikkarte verwendet wird, ansonsten tensorflow 1.6.0
- numpy 1.14.2

- pandas 0.22.0
- sklearn 0.19.1
- scipy 1.0.1
- pyparsing 2.2.0
- rdflib 4.2.2
- SPARQLWrapper 1.8.1
- pyspotlight 0.7.2
- nltk 3.2.5

Die beschriebenen Docker-Images in Abschnitt **docker** enthalten bereits alle nötigen Python-Pakete. Falls die Pakete manuell installiert werden müssen, ist es empfehlenswert, pip zu verwenden. Im Paket-Manager anaconda sind nicht alle Pakete verfügbar.

Zusätzlich zu den Python-Paketen ist Stanford CoreNLP Version 3.9.2 für das Preprocessing nötig. Die Bibliothek ist auf [17] zum Download verfügbar. Die Applikation muss via http://localhost:9000 verfügbar sein. Den Befehl um den Stanford CoreNLP Webserver zu starten ist in Abschnitt B.3.3 beschrieben.

B.2. Verwendung der Docker Images auf dem GPU-Server

Für die Anwendung wird ein Docker Image verwendet, welches die nötigen Komponenten zur Verfügung stellt. Um die Anwendung effizient mit genügend Ressourcen zu betreiben, wird der GPU-Server, welcher von der ZHAW zur Verfügung gestellt wird verwendet. Dieser Abschnitt beschreibt, wie die Images auf dem Server in Betrieb genommen werden. Der Zugriff auf den Server erfolgt dabei jeweils über SSH mittels dem Terminal vom lokalen Rechner.

B.2.1. Beschreibung des Docker Images

B.2.2. Installation der Images

Für die Verwendung der Dockerfiles auf dem GPU-Server muss vorab folgendes installiert und erstellt werden:

- Docker muss auf dem lokalen Rechner installiert werden (Windows oder Mac)
- Auf docker hub muss ein Repository erstellt werden, wo die Images für den GPU-Server verwaltet werden

B.2.2.1. Vorgehen für Build und Push der Images

Dieser Abschnitt kann übersprungen werden, wenn sich die Images bereits auf dem Repository befinden, oder nicht angepasst wurden.

Image aus Dockerfile builden

Dieser Schritt erfolgt lokal auf dem Rechner und ist nur nötig, wenn das Dockerfile für den GPU-Cluster geändert wurde und nun auf dem docker hub aktualisiert werden muss. Um ein Docker Image aus einem Dockerfile zu builden, sind folgende Schritte notwendig:

1. Navigiere in das Verzeichnis, wo sich die Dockerfiles befinden
2. In der Kommandozeile den Befehl aus Listing B.1 ausführen, um aus dem Dockerfile ein Image zu builden. Die entsprechenden Variablen sind zu ersetzen
3. Mittels dem Befehl aus Listing B.2 die Docker Images anzeigen
4. Das neue Image wird bei Repository und Tag "none" gesetzt haben. Dies kann mitells dem Befehl aus Listing B.3 geändert werden. Dabei zu beachten ist, dass für die Variable "repository" und "user_name" die Daten aus dem bereits erstellten Repository angegeben werden. Die Variable “tag” ist optional, Default dafür ist latest

Listing B.1: Befehl um das Docker Image zu builden.

```
1 docker build -f "dockerfile_name" "dockerfile_path"
```

Listing B.2: Befehl um die Docker Images anzeigen zu lassen.

```
1 docker images
```

Listing B.3: Befehl um die Docker Images anzeigen zu lassen.

```
1 docker tag "image_id" "user_name"/"repository":"tag"
```

Image auf docker hub pushen

Sobald das Image gemäss dem obrigen Schritt lokal gebaut wurde, kann dieses auf das entsprechende Repository gepushed werden. Listing B.4 zeigt den entsprechenden Befehl dazu. Dazu müssen die gleichen Werte für die Variablen "user_name", "repository" und "tag" wie bei Listing B.3 verwendet werden!

Listing B.4: Befehl um Image auf den docker hub zu pushen.

```
1 docker push "user_name"/"repository":"tag"
```

Image auf GPU-Server laden

Um das Image aus dem docker hub für den GPU-Server zu laden müssen folgende Schritte durchgeführt werden:

1. In das gewünschte Verzeichnis auf dem Server wechseln, wo das Image abgelegt werden soll
2. Den Befehl gemäss Listing B.5 ausführen. Die Werte für die Variablen "user_name", "repository" und "tag" können aus dem Repository entommen werden

Listing B.5: Befehl um Image vom docker hub zu pullen.

```
singularity pull docker://"user_name"/"repository":"tag"
```

B.2.3. Nötiger Code auf GPU-Server laden

Der benötigte Code für die Ausführung der Anwendung ist nicht im Docker Image enthalten. Dieser muss manuell auf den Server geladen werden. Um den Code auf den Server zu laden gibt es grundlegend zwei Möglichkeiten dies zu tun:

1. Den Code von einem eigens angelegten Github Repository auf den Server in ein gewünschtes Verzeichnis laden
2. Mittels SFTP den Code in ein gewünschtes Verzeichnis auf den Server laden

B.3. Preprocessing

B.3.1. Skripte und Dateien

In der folgenden Tabelle B.1 sind alle Dateien resp. Verzeichnisse aufgelistet, die für die Ausführung des Preprocessing nötig sind. Der Pfad ist jeweils ausgehend vom *src* Verzeichnis angegeben.

B.3.2. Daten

Alle Ausgaben und Eingaben werden beim Preprocessing ins Verzeichnis *Data* geschrieben resp. von dort gelesen. Dabei gibt es drei Unterverzeichnisse zu unterscheiden:

Für die Ausführung des Preprocessing muss das Verzeichnis *Data/SQA2018* mindestens die Dateien *queries.txt* und *questions.txt* enthalten. Diese Dateien enthalten die ursprünglichen NL-Fragen und SPARQL-Queries. Weitere Dateien und Unterverzeichnisse werden automatisch vom Preprocessing angelegt. Die vom Preprocessing

Tabelle B.1.: Dateien, die für das Preprocessing verwendet werden

Name	Description
./OneTimePreprocessing.py	Dieses Skript führt das Preprocessing aus. In B.3.3 ist beschrieben, wie das Preprocessing mit diesem Skript gestartet wird.
./build_inverted_indices.py	Dieses Skript bereitet die nötigen *.obj Dateien für die Eigenschaften Extrahierung und Ontology Extrahierung vor. Dafür ist die entsprechende *.owl Datei des DBpedia Dumps 2016-04 nötig. Die Datei muss im Verzeichnis Data/Ontology liegen.
./Preprocessing/DatasetSplitter.py	Aufteilung der gegebenen Daten in dev, train und test.
./Preprocessing/Pipeline.py	Definiert und führt die Preprocessing-Pipeline aus. Ruft die nötigen Funktionen in QueryEnumerator.py auf.
./Preprocessing/QueryEnumerator.py	Enthält oder ruft alle Funktionen spezifisch für die Entitäten Extrahierung, Eigenschaften Extrahierung und Ontologie Extrahierung auf. Die Entitäten Extrahierung erfolgt mit dem Online-Service von Spotlight über *http://api.dbpedia-spotlight.org/en/annotate*.
./Preprocessing/UriExtractor.py	Enthält die nötigen Funktionen, um die Eigenschaft Extrahierung und Ontologie Extrahierung durchzuführen. Wird von QueryEnumerator.py aufgerufen. Führt die Lookups in den *.obj Files aus. Für das POS-Tagging wird Stanford CoreNLP verwendet, welcher via http://localhost:9000 angesprochen wird.
./Preprocessing/VocabBuilder.py	Baut das Vokabular für eine Pipeline auf.

Tabelle B.2.: Datenverzeichnisse für das Preprocessing und Training

Name	Description
./Ontology	Enthält die nötige *.owl-Datei aus dem DBpedia Dump 2016-04, welches für die Erzeugung der Lookup-Daten für Eigenschaften und Ontologien Extrahierung nötig sind.
./serialized	Enthält die *.obj-Dateien, die für die Lookups der Eigenschaften und Ontologien nötig sind.
./SQA2018	In diesem Verzeichnis befinden sich alle Ein- und Ausgabedaten des Preprocessing und des Trainings. Ausserdem sind müssen hier die ursprünglichen Fragen und Queries der SQA2018 abgelegt werden.

angelegten Unterverzeichnisse werden auch für das Trainieren des Modells verwendet.

Dabei bedeuten die angelegten Verzeichnisse folgendes (alle im Data/SQA2018 Verzeichnis):

Tabelle B.3.: Vom Preprocessing erstellte Verzeichnisse

Name	**Description**
baseline	Enthält die nötigen Baseline-Eingabedaten für das Trainieren des NMT-Modells.
var2normalized_attention _normed_bahdanau	Enthält die nötigen Eingabedaten für das Trainieren des NMT-Modells nach der Entitäten Extrahierung, ohne *resource*-Methode.
var2normalized_attention _normed_bahdanau_purged	Bei purged werden nur die SPARQL-Queries und NL-Fragen berücksichtigt, die auch Entitäten enthalten. D.h. alle Eingaben ohne Entitäten werden im Training nicht berücksichtigt.
var2normalized_attention _normed_bahdanau _ontology	Enthält die nötigen Ontology-Eingabedaten für das Trainieren des NMT-Modells.
var2normalized_attention _normed_bahdanau _property	Enthält die nötigen Property-Eingabedaten für das Trainieren des NMT-Modells.
var2normalized_attention _normed_bahdanau _entity_property _clean_full_new2	Enthält eine Migration der Ergebnisse der Entitäten und Eigenschaften Extrahierungen. Dabei werden für beide die *clean*-Resultate verwendet.
var2normalized_attention _normed_bahdanau _entity_ontology _clean_full_new	Enthält eine Migration der Ergebnisse der Entitäten und Ontologien Extrahierungen. Dabei werden für beide die *clean*-Resultate verwendet.
var2normalized_attention _normed_bahdanau _entity_property_ontology	Enthält eine Migration der Ergebnisse der Entitäten, Eigenschaften und Ontologien Extrahierungen. Dabei werden für Entitäten und Eigenschaften die *clean*-Resultate verwendet, für die Ontologien nicht.

Spezielle Varianten der Verzeichnisse Bei einigen Verzeichnisse werden bestimmte Bezeichnungen angehängt. Die Bedeutungen dieser Bezeichnungen sind in Tabelle B.4:

Resultate des Preprocessing Innerhalb der erstellten Verzeichnisse befinden sich die Dateien, die für das Trainieren eines Modells nötig sind. B.6 beschreibt, wofür die wichtigsten Dateien jeweils verwendet werden.

Tabelle B.4.: Vom Preprocessing erstellte spezielle Verzeichnisse

*_attention_normed_bahdanau	Für das Trainieren eines Modells mittels Bahdanau-Attention [3].
*_dropout50	Diese Verzeichnisse enthalten Daten, die für das Trainieren mit Dropout auf 50% nötig sind. Die Daten sind identisch zu den Ergebnissen der Standard-Pipeline (ohne den Endungen in dieser Tabelle).
*_clean_full, *_clean	In diesem Fall wird zusätzlich die SPARQL-Query überprüft, und es werden nur die URI's in der NL-Frage ersetzt, die auch in der SPARQL-Query vorkommen. Dies entspricht also dem "Best Case", d.h. eine fast exakte Extrahierung der Entitäten, Eigenschaften und/ oder Ontologien. Bei den Entitäten handelt es sich hier um die *Resource-Methode*.

Tabelle B.5.: Vom Preprocessing erstellte Dateien, die für das Training nötig sind.

dev.*	Daten für die Evaluation des Trainings.
train.*	Daten für das eigentliche Trainieren.
vocab.*	Vokabular der NL-Fragen und SPARQL-Queries für das Training.
.tok.	Tokens der NL-Fragen und SPARQL-Queries für das Training.
*.en	Daten, die mit der NL-Frage zusammenhängen.
*.sparql	Daten, die mit der SPARQL-Query zusammenhängen.

B.3.3. Preprocessing mittels Singularity ausführen

Das Preprocessing muss vor dem Trainieren des NMT-Modells ausgeführt werden. Das Ausführen ist nur einmal nötig.
Sobald sich der benötigte Code sowie die benötigten Images sich auf dem Server befinden, kann Singularity dazu verwendet werden, um den Code auszuführen. Bei Singularity ist es möglich, direkt über das Hostsystem auf die Files zuzugreifen, sofern diese dort abgelegt sind. Somit ist es also nicht notwendig, die Dateien innerhalb des Images zu laden. Folglich werden die einzelne Schritten beschrieben, um die nötigen Images sowie das Preprocessing zu starten.

Wichtig:
Damit auf dem Server mit mehreren Session gearbeitet werden kann, muss das Linux-Paket "Screen" auf dem Server installiert sein .

1. Für das Preprocessing werden zwei Instanzen des Images gestartet, eines für den Stanford CoreNLP Server und eines für den Code der Anwendung.
   a) Ein Screen mittels Befehl aus Listing B.6 starten. Um den Screen zu identifizieren, wird der Variable ßession_nameëinen gewählten Namen für die Session zugewiesen
   b) In das Verzeichnis wechseln, wo das *.simg abgelegt ist und mit Listing B.7 das Image starten. Die Variable "image_name" muss durch den entsprechenden Namen des Images ersetzt werden. Das erfolgreiche starten des Images wird dadurch gekennzeichnet, dass neu "Singularity" am Anfang der Zeile im Terminal steht
   c) In das Hauptverzeichnis des Stanford CoreNLP navigieren (stanford-corenlp-full) und mit Listing B.8 den Server starten
   d) Sobald der Server gestartet ist, kann der Screen mittels Ctrl+A, D verlassen werden
2. Ein weiterer Screen mittels Befehl aus Listing B.6 starten
3. Ein weiteres Image starten. Dazu wieder in das Verzeichnis wechseln, wo das *.simg abgelegt ist und mit Listing B.7 das Image starten
4. Zum Verzeichnis des Codes der Anwendung wechseln und das Preprocessing mittels Listing B.9 starten.

Listing B.6: Befehl um einen Screen zu erstellen.

```
screen -S "session_name"
```

Listing B.7: Befehl um das Image zu starten.

```
srun --pty --ntasks=1 --cpus-per-task=2 --mem=4G --gres=gpu:1 singularity shell "
    image_name".simg
```

Listing B.8: Befehl um den StanfordCoreNLP Server zu starten.

```
java mx4g cp "*" edu.stanford.nlp.pipeline.StanfordCoreNLPServer port 9000 timeout
    15000
```

Listing B.9: Befehl um Preprocessing zu starten.

```
python3 OneTimePreprocessing.py
```

Hinweis: Falls die NL-Fragen und SPARQL-Queries neu in dev und test aufgeteilt werden müssen (z.B. wenn andere Daten als die SQA2018-Daten verwendet werden, oder die Ratio verändert wird), dann muss in *OneTimePreprocessing.py* die globale Variable *FIRST_TIME* auf *True* gesetzt werden.

B.3.4. Zusätzliches

Weitere Angaben zu der Verwendung des Preprocessing befinden sich in der technischen Dokumentation von [3].

B.4. Openlink Virtuoso

Für das verwalten und bereitstellen der Datensätze von DBpedia wird Openlink Virtuoso verwendet.

B.4.1. Anforderungen

Um Virtuoso effizient zu nutzen wird ein System mit folgenden Anforderungen vorausgesetzt:

- Linux Maschine mit root Zugang und Docker installiert
- Mindestens 64Gb Arbeitsspeicher (RAM)
- Mindestens 128Gb freier Speicherplatz für die Datensätze von DBpedia

B.4.2. Installation mittels Docker

Zur Installation steht ein Skript zur Verfügung, welches mittels Listing B.10 in der Shell ausgeführt werden kann. Nach erfolgreicher Installation wird Virtuoso automatisch mit 64Gb Arbeitspeicher gestartet und ist über Port 8891 erreichbar.

Listing B.10: Befehl um die Installation von Virtuoso zu starten.

```
1 sh install_virtuoso.sh
```

B.5. Evaluation

Die Evaluations-Software befindet sich im Ordner *Evaluation*. Dafür wird Python (empfohlene Version: 3.6.5) und ein Magnitude Model, welches im Ordner *word_embedding_models* abgelegt werden muss, benötigt.

In der nachfolgenden Tabelle ist kurz die Struktur des Verzeichnisses *Evaluation* und die wichtigsten Dateien erklärt.

Tabelle B.6.:

Name	**Description**
./pipeline_evaluation.py	Dieses Skript führt die Evaluation des Preprocessing aus, anhand der Daten, die im Verzeichnis *testData* liegen.
./entity_linking_evaluation.py	
./exact_partial_matches.json	Enthält die Liste von Exact Matches und Partial Matches, die in Abschnitt 5 analysiert werden.
./utils	Enthält Utility-Funktionen, die für die Evaluation nötig sind.
./testData	Enthält Testdaten, die für die Evaluation des Preprocessing verwendet werden.
./scorers	Enthält die nötigen Module für die Entity Similarity, Type Similarity und Syntax Evaluierungen und die nötigen Module für die Ermittelung der Magic Entities.

Um die Evaluation des Preprocessing zu starten, muss der folgende Befehl ausgeführt werden:

Listing B.11: Befehl um die Evaluation zu starten.

```
1 python pipeline_evaluation.py --questions testData/dev.en --gt_file testData/dev.sparql
2     --prep_file testData/dev.dbpedia.en
```

Literatur

[1] *Alphabetical list of part-of-speech tags used in the Penn Treebank Project*. URL: https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html (besucht am 03. 01. 2019).

[2] Ann-Kathrin Hartmann, Tommaso Soru und Edgard Marx. "Generating a Large Dataset for Neural Question Answering over the DBpedia Knowledge Base". In: (Apr. 2018).

[3] Nicolas Hoferer u. a. "Automatische Übersetzung von natürlicher Sprache nach SPARQL mit neuronalen Netzen". In: *Bachelor Thesis Spring 2018 ZHAW* (2018).

[4] Nicolas Hoferer u. a. "Automatische Übersetzung von natürlicher Sprache nach SPARQL mit neuronalen Netzen". In: *Bachelor Thesis Spring 2018 ZHAW* (2018), S. 22.

[5] Nicolas Hoferer u. a. "Automatische Übersetzung von natürlicher Sprache nach SPARQL mit neuronalen Netzen". In: *Bachelor Thesis Spring 2018 ZHAW* (2018), S. 57–59.

[6] Nicolas Hoferer u. a. "Automatische Übersetzung von natürlicher Sprache nach SPARQL mit neuronalen Netzen". In: *Bachelor Thesis Spring 2018 ZHAW* (2018), S. 34.

[7] Nicolas Hoferer u. a. "Automatische Übersetzung von natürlicher Sprache nach SPARQL mit neuronalen Netzen". In: *Bachelor Thesis Spring 2018 ZHAW* (2018), S. 53–55.

[8] Nicolas Hoferer u. a. "Automatische Übersetzung von natürlicher Sprache nach SPARQL mit neuronalen Netzen". In: *Bachelor Thesis Spring 2018 ZHAW* (2018), S. 63.

[9] Nicolas Hoferer u. a. "Automatische Übersetzung von natürlicher Sprache nach SPARQL mit neuronalen Netzen". In: *Bachelor Thesis Spring 2018 ZHAW* (2018), S. 64.

[10] Nicolas Hoferer u. a. "Automatische Übersetzung von natürlicher Sprache nach SPARQL mit neuronalen Netzen". In: *Bachelor Thesis Spring 2018 ZHAW* (2018), S. 45.

[11] Christopher D. Manning, Prabhakar Raghavan und Hinrich Schütze. *Introduction to Information Retrieval*. Online Edition. Cambridge University Press, 2008, S. 32. ISBN: 0521865719. URL: https://nlp.stanford.edu/IR-book/pdf/irbookprint.pdf.

[12] Eugene Brevdo und Rui Zhao Minh-Thang Luong.

[13] Martin F Porter. "An algorithm for suffix stripping". In: *Program* 14.3 (1980), S. 130–137.

[14] *Resource Description Framework*. URL: https://www.w3.org/TR/rdf-sparql-query/ (besucht am 22. 12. 2018).

[15] Kai-Uwe Saake Gunter und Sattler. "Algorithmen und Datenstrukturen". In: 5. Aufl. Heidelberg: dpunkt.verlag, 2014. Kap. 17.

[16] *SPARQL*. URL: https://www.w3.org/RDF/ (besucht am 22. 12. 2018).

[17] *Stanford CoreNLP - Natural Language Software*. Version 3.9.2. URL: https://nlp.stanford.edu/software/tagger.html (besucht am 03. 01. 2019).

[18] *Stemmers*. URL: http://www.nltk.org/howto/stem.html (besucht am 04. 01. 2019).

[19] *The English (Porter2) stemming algorithm*. URL: http://snowballstem.org/algorithms/english/stemmer.html (besucht am 04. 01. 2019).

[20] *The Porter Stemming Algorithm*. URL: https://tartarus.org/martin/PorterStemmer/index.html (besucht am 04. 01. 2019).