Bachelorarbeit

Entitätserkennung auf einem Web-Korpus

Manuel Ruder

26. 08. 2014

Albert-Ludwigs-Universität Freiburg im Breisgau
Technische Fakultät
Institut für Informatik

Bearbeitungszeitraum
26. 05. 2014 – 26. 08. 2014

Gutachter
Prof. Dr. Hannah Bast
Betreuer
Elmar Haussmann

Erklärung

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Freiburg, 25.8.2014
Ort, Datum

Manuel Ruder

Inhaltsverzeichnis

Zusammenfassung

Broccoli ist eine semantische Suchmaschine, die für das Durchsuchen der englischsprachigen Wikipedia entwickelt wurde. In einer früheren Arbeit wurde damit begonnen, *broccoli* für das Durchsuchen allgemeiner Web-Seiten zu erweitern. Hierzu mussten sowohl der Entitätserkenner als auch die Komponente zum Parsen der Seiten angepasst bzw. verallgemeinert werden. Hierbei zeigten sich einige Schwierigkeiten, wie etwa eine hohe Zahl an Falscherkennungen von Entitäten. In dieser Arbeit soll die Qualität der Suche durch einen verbesserten Entitätserkenner gesteigert werden. Dazu wurde unter anderem die Erkennung von Eigennamen verbessert. Ein neu geschriebener Disambiguationsalgorithmus soll zudem im Text vorhandene Mehrdeutigkeiten auflösen. Durch eine ebenfalls neu geschriebene Evaluations-Komponente können Falscherkennungen genauer untersucht werden als bisher. Für die Evaluations-Komponente wurde eine Auswahl an Artikeln aus dem Korpus mit Ground Truth Annotationen versehen.

1. Einleitung

Das maschinelle Verarbeiten natürlicher Sprache spielt eine wichtigere Rolle in vielen Teilbereichen der Informatik. Im Rahmen von semantischen Suchmaschinen ist das Verarbeiten natürlicher Sprache ein wichtiger Bestandteil, um zu der Suchanfrage des Benutzers passende Ergebnisse zu finden. Mit semantischen Suchmaschinen ist es möglich, gezielt nach Entitäten oder Klassen von Entitäten zu suchen.
Eine der Herausforderungen ist hierbei, dass sich unterschiedliche Begriffe und Namen syntaktisch gleichen, aber semantisch verschieden sein können. So gibt es beispielsweise zu fünf verschiedene Personen in der deutschsprachigen Wikipedia einen Artikel, die sich den Namen „Michael Jordan“ teilen[1]. Die konkrete semantische Bedeutung hängt dabei häufig davon ab, in welchem Kontext ein Name oder Begriff erwähnt wird.
Sucht der Benutzer in einer semantischen Suchmaschine beispielsweise nach dem Basketballspieler Michael Jordan, so sollen nur solche Ergebnisse gelistet werden, in denen tatsächlich der Basketballspieler und nicht der gleichnamige Schauspieler oder Universitätsprofessor gemeint ist.
broccoli ist eine solche semantische Suchmaschine, die ursprünglich für das Durchsuchen der englischsprachigen Wikipedia entwickelt wurde.
Abbildung 1.1 zeigt, wie *broccoli* den Benutzer bei Formulierung einer semantischen Suchanfrage unterstützt.

Von Phillip Bausch wurde *broccoli* in einer früheren Arbeit für das Durchsuchen allgemeiner Web-Seiten erweitert, da in Wikipedia oft aktuelle Informationen z.B. zu politischen Ereignissen fehlen. [Bau14] Als Datenquelle wurde CommonCrawl benutzt, woraus unter anderem 600 Tausend Nachrichtenseiten extrahiert wurden. Hierfür mussten einerseits die Web-Seiten von nicht relevanten Elementen befreit werden, wie etwa Navigationsleiten oder Werbung. Zudem musste der Entitätserkenner angepasst werden, damit dieser auch auf einem allgemeinen Web-Korpus funktioniert.
In dieser Arbeit soll dies fortgeführt werden. Vor allem soll die Qualität der Suche durch eine verbesserte Entitätserkennung gesteigert werden, da hier nach einer kurzen Analyse in Kapitel 2 das größte Verbesserungspotential gesehen wurde.

Bei der Entitätserkennung unterscheidet man in der Regel zwischen *Named Entity Recognition (NER)* und *Entity Disambiguation*.
Unter ersterem versteht man das Erkennen von Eigennamen in einem Text, sowie teils auch die Zuordnung zu groben Kategorien (häufig u.a. Person, Location und

[1] http://de.wikipedia.org/wiki/Michael_Jordan_%28Begriffskl%C3%A4rung%29

Organisation).
Unter *Entity Disambiguation*, manchmal auch als *Entity Linking* bezeichnet, versteht man die Zuordnung einer namentlichen Erwähnung einer Entität in einem Dokument zu der Entität selbst. Die Schwierigkeit besteht dabei darin, Mehrdeutigkeiten aufzulösen, daher auch der Name *Disambiguation*.

In dieser Arbeit sollen beide Aspekte berücksichtigt werden. Unter dem Begriff „Entitätserkennung“ soll dabei sowohl das Erkennen von Eigennamen (NER), als auch die Disambiguation verstanden werden.

Evaluiert wurden die verschiedenen Ansätze zur Verbesserung unter anderem auf dem ERD-50-Korpus, welcher eine Teilmenge des bei der *2014 Entity Recognition and Disambiguation Challenge* [ERD2014] verwendeten Korpus darstellt.

Abbildung 1.1.: Auswahl von Entitäten mit *broccoli*.
Der Benutzer bekommt bei der Eingabe eines oder mehrerer Wörter (oben) eine Auswahl an Instanzen (blauer Kasten), welche jeweils die Entitäten widerspiegeln. Klickt der Benutzer auf eines der Einträge im blauen Kasten, werden gezielt solche Dokumente herausgesucht, die sich auf die betreffende Entität beziehen.

2. Analyse der bisherigen Ansätze

Im ersten Teil dieser Arbeit wurden die begonnenen Verallgemeinerungen und Anpassungen an *broccoli* evaluiert, um die tatsächlichen Probleme zu identifizieren.

2.1. Extrahieren von Text aus HTML-Dokumenten

Auf einer Nachrichten-Seite beispielsweise gibt es neben dem eigentlichen Nachrichtentext noch weitere Elemente, die nicht relevant für die semantische Suche sind. Dazu zählen etwa Werbung, Benutzerkommentare, eine Vorschau zu weiteren Artikeln, und vieles mehr. Jedes Nachrichtenportal verwendet dabei typischerweise ein eigenes Layout, sodass ein allgemeiner Ansatz gefunden werden muss, relevante Bereiche zu extrahieren.

In der früheren Arbeit wurde dafür Boilerpipe verwendet. Boilerpipe[1] ist eine in Java geschriebene Bibliothek, die einen für dieses Anwendungsszenario geeigneten *ArtikleExtractor* mitbringt.

Zur Evaluation wurden zufällig 100 Seiten aus dem Nachrichten-Korpus ausgewählt und mittels Boilerpipe verarbeitet. Die daraus entstandenen Texte wurden manuell mit den Web-Dokumenten verglichen. Größere Probleme konnten hier nicht gefunden werden. Häufig wurde fälschlicherweise der Disclaimer einer Nachrichtenseite in den Text miteinbezogen, was jedoch die semantische Suche nur in geringem Maß beeinträchtigt.
Der für die Suche relevante Text war in den meisten Fällen vollständig enthalten. Nur in einem Zehntel der Fälle haben einige wenige Sätze aus längeren Artikeln gefehlt. In keinem der Fälle kam es zu einem Totalausfall.

Daher wurde entschieden, auch weiterhin Boilerpipe für das Extrahieren der Texte zu verwenden.

Dabei muss jedoch berücksichtigt werden, dass der *ArtikleExtractor* auf das Extrahieren von englischsprachigen Nachrichten-Seiten bzw. auf Seiten mit einem ähnlichen Aufbau (ein zentrales Textfeld) optimiert ist. Sollten weitere Internet-Quellen in den Korpus aufgenommen werden, wie etwa Internet-Foren, muss dieser Schritt möglicherweise angepasst werden. Dies war jedoch nicht Gegenstand dieser Arbeit.

[1] https://code.google.com/p/boilerpipe/

2.2. Sprachverarbeitung

Natürliche Sprachverarbeitung spielt eine wichtige Rolle bei semantischen Suchmaschinen. Unter anderen müssen in dem Dokument Entitäten erkannt werden.

Bei der Verwendung des Wikipedia-Korpus konnte man sich eine Besonderheit zunutze machen: In Wikipedia sind wichtige Begriffe und Namen untereinander verlinkt. So kann man relativ einfach Entitäten in einem Wikipedia-Artikel erkennen, indem man sich an diesen Verlinkungen orientiert.

Auf einem allgemeinen Web-Korpus funktioniert dies nicht, da es hier keine Verlinkungen gibt. Stattdessen müssen Entitäten hier anhand syntaktischer Merkmale erkannt werden. Dies führt zu einer ganzen Reihe von neuen Problemen, wie etwa, dass Mehrdeutigkeiten aufgelöst werden müssen, da in vielen Fällen derselbe Name von mehreren Entitäten geteilt wird.

Von Phillip Bausch wurde dazu bereits ein einfacher Entitätserkenner geschrieben (grobe Skizze siehe Algorithmus 1), der für jedes Wort und für jede Verkettung von bis zu fünf hintereinander stehenden Wörtern prüft, ob diese syntaktisch identisch zu einem Namen oder Alias einer Entität ist. Im Folgenden werden die unterschiedlichen Namen und Aliase einer Entität auch unter dem Begriff „Oberflächenform“ einer Entität zusammengefasst. Als Datenstruktur wird eine Hash-Map verwendet, die zu jeder Oberflächenform die dazu gehörende Entität speichert. Somit kann in konstanter Zeit bestimmt werden, ob ein String eine Oberflächenform einer Entität ist.
Wenn eine Entität gefunden wird, wird eine entsprechende Annotation hinzugefügt. Dabei kann es auch vorkommen, dass sich zwei Annotationen überlappen. Beispielsweise ist „Late Show with David Lettermann“ eine Oberflächenform der gleichnamigen Fernsehshow. Gleichzeitig wäre aber auch „David Lettermann“ eine Oberflächenform des Moderators. In einem solchen Fall wird immer nur die längere Entität behalten, genauer: Die Entität, die mehr Buchstaben in dem zu annotierenden Dokument abdeckt.

Zusätzlich wurde auch eine Erkennung von Koreferenzen implementiert. Zum einen wurde erkannt, wenn statt des vollen Namens einer Entität im weiteren Verlauf eines Textes nur noch ein Teil davon erwähnt wird (Beispiel: „Obama“ für „Barack Obama“). Zum anderen wurden Pronomen mit der jeweils letztgenannten Entität verglichen, und falls diese in Geschlecht mit dem Pronomen übereinstimmt, wurde dies als Koreferenz für die zuletzt genannte Entität aufgefasst.

Zur Evaluation des Entitätserkenners wurde zunächst eine Evaluations-Komponente geschrieben, die in Kapitel 5 genauer beschrieben wird. Mit dieser war es möglich, Falscherkennungen in einem Text genauer zu untersuchen.
Bei einer ersten Analyse wurden zum einen eine hohe False-Positive-Rate festgestellt. Philipp Bausch hatte in seiner Arbeit bereits ein Verhältnis von 1:4 von erkannten

Entitäten zu der Anzahl der Wörter beschrieben. Dies konnte im wesentlichen bestätigt werden.

Auch eine hohe Zahl an falsch erkannten Entitäten konnte beobachtet werden. Ein Teil der Falscherkennungen konnte auf Fehler in der Ontologie-Datenbank zurückgeführt werden. Zum Beispiel wurde bereits beim Erstellen der Ontologie-Datenbank mehrdeutige Oberflächenformen herausgefiltert. Dadurch waren viele wichtige Zuordnungen nicht enthalten (Bsp. Für „President Bush“ kann und sollte es mehr als eine Entität geben). Zudem fehlte ein Disambiguationsalgorithmus für eben diesen Fall.

Aufgrund dessen wurde entschieden, dass im weiteren Verlauf dieser Arbeit der Fokus auf die Verbesserung der Entitätserkennung gelegt wird.

```
input  : A map String ↦ Entity entityMap
input  : An array of String tokens /* The words of the document */
output : A set of EntitiyAnnotations L

L ← ∅
/* Iterate over all tokens */
for i ← 0 to tokens.size() do
    /* Only recognize if a noun */
    if not token i is a noun then
        continue for
    end
    /* Start with the longest sequence of tokens */
    for j ← Min(N, i) downto 0 do
        /* Concatenate tokens */
        surfaceForm ← tokens[i − j]_tokens[i − (j + 1)]_ · · · _tokens[i − 0]
        /* Prefer longest sequence. Break if found a match. */
        if entityMap.containsKey(surfaceForm) then
            entity ← entityMap.get(surfaceForm)
            CreateAnnotation(entity, L)
            break for
        end
    end
end
return L

Procedure CreateAnnotation(newEntity, L)
    for entity ∈ L do
        if Overlap(entity, newEntity) then
            L ←L \{entity, newEntity}
            L ←L ∩{DisambiguateOverlap(entity, newEntity)}
        end
    end

/* Always returns the longer (#characters) entity. */
Function DisambiguateOverlap(entity1, entity2)
```

Algorithmus 1 : Bisherige Entitätserkennung

3. Stand der Forschung

Ein häufig anzutreffendes Verfahren zur *Entity Disambiguation* besteht darin, die richtige Entität mithilfe von Kontext-Informationen zu bestimmen. Dazu wird für jede in Frage kommende Entität die Beschreibung oder Definition in einem Lexikon (z.B. Wikipedia) herangezogen und mit dem Text des Dokumentes verglichen. Es wird letztlich die Entität ausgewählt, dessen Beschreibung die größte Übereinstimmung mit dem Dokument aufweist.
Zur Ermittlung der kontextbezogenen Übereinstimmung können aus den Texten und Artikeln Merkmals-Vektoren (engl. *feature vectors*) erstellt werden, wobei es für jedes Wort einen Eintrag im Vektor gibt, der z.B. beschreibt, wie häufig ein Wort in einem Text erwähnt wird. Diese Vektoren können dann etwa mit der Kosinus-Ähnlichkeit (engl.*cosine similarity*) verglichen werden.

In Wikify! [MC07] wird dieser Ansatz unter anderem genutzt, um Schlüsselwörter in einem Text zu den entsprechenden Wikipedia-Artikeln zu verlinken.
Als Schlüsselwörter werden dabei u.a. Fachbegriffe, Namen, Orte, usw. angesehen. Das Ziel war aber nicht, alle vorkommenden Begriffe zu verlinken, sondern nur solche Begriffe, die als besonders wichtig für das Textverständnis angesehen wurden. In einer als *"keyword extraction"* bezeichneten Phase wurde daher zunächst wichtige Schlüsselworte bestimmt.
Die Kontext-Informationen zur Disambiguation wurden aus den Wikipedia-Artikeln der Entitäten gewonnen, und jeweils mit dem Absatz des Dokumentes verglichen, in dem die Entität erwähnt wird.
In der Publikation wurde zusätzlich ein Ansatz implementiert, der auf maschinellem Lernen basiert und eine Klassifikation der Entitäten mithilfe eines Trainingsdatensatzes vornimmt.
Beide Ansätze wurden hinsichtlich Precsision, Recall und F-Maß evaluiert und mit einer *random baseline* verglichen. Beide Ansätze schnitten dabei deutlich besser ab als die *random baseline*, wobei der auf maschinellem Lernen basierte Ansatz in dieser Publikation das bessere Ergebnis erzielte. Bei der Evaluation wurde außerdem eine ideale *keyword extraction* angenommen. Die *keyword extraction* wurde zuvor getrennt von der Disambiguation evaluiert.

In [LPLA10] wurden verschiedene Varianten zur Gewinnung der Kontext-Vektoren beschrieben und diskutiert. Die Publikation ist auf die Disambiguation von Eigennamen, speziell von Personen, in einem Web-Dokument fokussiert.
Die Kontext-Vektoren der Entitäten wurden dabei aus den Einleitungen der dazu-

gehörigen Wikipedia-Artikel erzeugt — je nach Variante wurden alle Wörter, nur groß geschriebene Wörter oder nur Wörter, welche Teil eines Wikipedia-Links sind, miteinbezogen.
Evaluiert wurden die Ansätze unter anderem auf dem Kulkarni Name Corpus, welcher eine bewusste Auswahl an Artikeln mit mehrdeutigen Personennamen enthält. Dabei konnte gezeigt werden, dass es weder notwendig noch vorteilhaft ist, alle Wörter aus der Wikipedia-Einleitung miteinzubeziehen. Tatsächlich konnte bei der Variante, bei der nur groß geschriebene Wörter miteinbezogen wurden, eine höhere Genauigkeit als mit allen Wörtern erzielt werden. Die höchste Genauigkeit wurde mit den Wörter aus den Wikipedia-Links erreicht.

In [LME12] wird zur Disambiguation mithilfe der Kontext-Vektoren zusätzlich der Popularitätswert einer Entität miteinbezogen. Als Popularitätswert einer Entität wird dabei die Anzahl der Wikipedia-Artikel gezählt, die auf diese Entität verlinken. Letztlich wird daraus ein *final link score* berechnet, der sich aus dem Produkt aus Popularitätswert und der Kosinus-Ähnlichkeit zusammensetzt.

Die Publikation [Cuc07] beschreibt ebenfalls die *Entity Disambiguation* basierend auf Kontext-Informationen aus Wikipedia.
Die Kontext-Informationen wurden aus den Verlinkungen der Wikipedia-Artikel untereinander berechnet, es wurden aber auch in Betracht gezogen, Kontext-Informationen aus häufig vorkommenden Wörtern oder anhand der Tf-Idf Werte zu gewinnen. Tf-Idf steht für „term frequency - inverse document frequency“, und kann zur Beurteilung der Wichtigkeit einzelner Worte herangezogen werden.
Zusätzlich wurde auch eine Vorgehensweise zur *Named Entity Recognition* beschrieben, welche die genaue Start- und Endposition eines Eigennamens ermittelt. Hierfür wurden Groß- und Kleinschreibung, Web-Statistiken sowie Statistiken eines Trainingsdatensatzes (CoNLL-2003) miteinbezogen.

Auch gibt es eine Vielzahl aktiver Forschungsprojekte, die sich hauptsächlich mit der *Named Entity Recognition* beschäftigen. So gibt es zum Beispiel von der *Stanford Natural Language Processing Group* eine Bibliothek *Stanford NER*[1], die für das Erkennen und Klassifizieren von Eigennamen genutzt werden kann. Dazu werden statistische Modelle auf Basis von Conditional Random Fields genutzt. [FGM05].

[1] http://nlp.stanford.edu/software/CRF-NER.shtml

4. Verbesserung der Entitätserkennung

Die Verarbeitung der Texte wird in *broccoli* mithilfe einer UIMA-Pipeline durchgeführt. UIMA ist ein von Apache Foundation betreutes Framework für die Programmiersprache Java, und eignet sich insbesondere für die Verarbeitung von natürlicher Sprache. Ein wesentlicher Bestandteil der UIMA-Pipeline sind Annotationen, die von den einzelnen Komponenten der Pipeline eingefügt werden können.

Die Aufgabe des Entitätserkenners ist es, in dem Dokument erwähnte Entitäten zu erkennen, zu disambiguieren und mit einer Annotation versehen. Die Annotationen sollen dabei überschneidungsfrei sein.

Der Entitätserkenner kann dabei bereits auf Annotationen früherer Komponenten zugreifen. Dazu zählen:

- Die *Tokenizer*-Komponente unterteilt den Text eines Dokumentes in einzelne Wörter, und erstellt eine Annotation für jedes Wort.
- Die *POS-Tagger*-Komponente klassifiziert die einzelnen Wörter nach Wortart (u.a. Nomen, Verb, Adjektiv), und fügt diese Information zu den jeweiligen Annotationen der einzelnen Wörter hinzu.

4.1. Ontologie

Als Ontologie für den Entitätserkenner wird die Freebase-Datenbank verwendet.

Dem Entitätserkenner stehen dabei die Meta-Informationen aus Freebase zur Verfügung, wie etwa die Topics / Klassen, die Aliase, und die Beschreibungen. Für Entitäten der Klasse *person* steht zusätzlich das Geschlecht zur Verfügung.

Darüber hinaus werden diese Meta-Informationen durch Daten ergänzt, die aus der englischsprachigen Wikipedia gewonnen werden. Dazu gehören ein Popularitätswert für jede Entität sowie zusätzliche Aliase. Der Popularitätswert entspricht jeweils der Anzahl der Wikipedia-Seiten, die auf eine Entität verlinken. Die zusätzlichen Aliase werden aus den Wikipedia Weiterleitungen gewonnen. Diese Informationen waren bereits von Anfang an in der Ontologie vorhanden.

Zusätzlich wurde damit experimentiert, Aliase aus den Beschreibungen zu gewinnen. Zum Beispiel werden die „Detroit Tigers“ oft nur als „Tigers“ bezeichnet. Daher

wurden Namensbestandteile, die in der Beschreibung häufiger erwähnt werden als der Name selbst, als Alias hinzugefügt.

4.2. Datenstruktur

Wie in dem bereits vorhandenen Entitätserkenner wird eine Hash-Map verwendet, um einen möglichst effizienten Zugriff auf die Entitäten zu bekommen, bei der die Oberflächenform der Entität als Schlüssel verwendet wird.
Falls eine Entität mehrere Oberflächenformen hat, wird die Entität entsprechend mehrfach — einmal für jede Oberflächenform — in der Hash-Map gespeichert.

Allerdings hat dieser Ansatz den Nachteil, dass es für jede Oberflächenform nur eine Entität geben kann, was in der Praxis nicht immer zutrifft. Daher wurde die Hash-Map so erweitert, dass für jede Oberflächenform eine Liste an Entitäten gespeichert werden kann, eine Art Multi-Map. Dies hat aber zur Folge, dass bei jeder Textstelle entschiedenen werden muss, welche Entität aus der Liste an dieser Stelle zutreffend ist. Ein Algorithmus zur Disambiguation wird in Abschnitt 4.4 beschrieben.

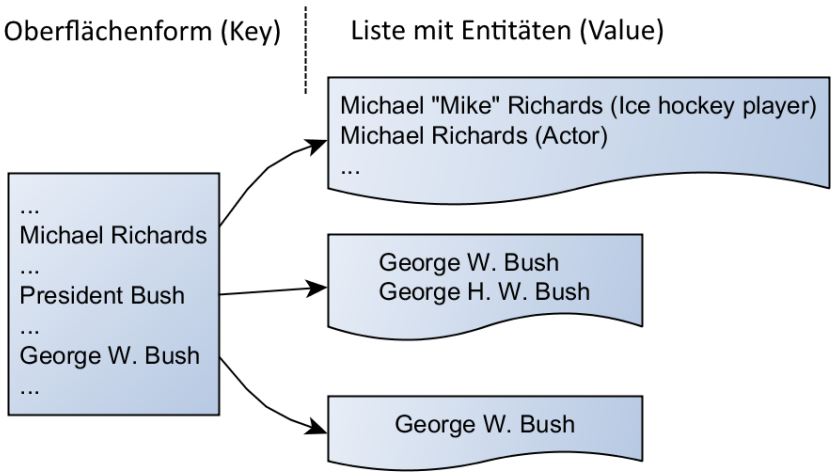

Abbildung 4.1.: Die Speicherung der Entitäten in einer Hash-Map.

4.3. Finden von Eigennamen

Der Erkennung der Entitäten wurde zunächst in zwei Probleme aufgeteilt: Das Erkennen von Eigennamen und das Erkennen von Gattungsnamen. Ein Teil der False-positives konnte darauf zurückgeführt werden, dass Wörter in einem Text fälschlicherweise für Eigennamen gehalten wurden.

Zur Erkennung von Eigennamen bietet es sich an, die POS-Tags zu berücksichtigen. Der für *broccoli* verwendete POS-Tagger erkennt nicht nur, ob es sich bei einem Wort um ein Nomen handelt, sondern unterscheidet darüber hinaus zwischen *proper nouns* (NNP- oder NNPS-Tag) und gewöhnlichen *nouns* (NN- oder NNS-Tag).
Die Anfangs- und Endpositionen lassen sich jedoch nicht aus den POS-Tags ableiten. Beispielsweise würden in dem Namen `Memorial Institute for the Prevention of Terrorism` nur die Worte Memorial, Institute, Prevention und Terrorism mit NNP-Tags versehen. Die übrigen Worte, die aber ebenfalls Bestandteil des Eigennamens sind, werden nicht als *proper nouns* gekennzeichnet.

Der in Abschnitt 2.2 beschriebene Algorithmus wurde zunächst so abgeändert, dass nur Wörter und Phrasen berücksichtigt werden, die mindestens ein als *proper noun* klassifiziertes Wort enthalten.
Die Erkennung von Gattungsnamen wurde später wieder hinzugefügt, dabei wurden aber nur bestimmte Entitäten zugelassen. Unter anderem wurden Entitäten, die sich auf Personen, Organisationen, Orte oder Film- bzw. Buchtitel beziehen, herausgefiltert. Eine Textstelle wurde als Gattungsname angesehen, wenn diese nur als gewöhnliche *nouns* (NN- oder NNS-Tag) klassifizierte Nomen enthält.

Dadurch konnte bereits ein Teil der False-Positives vermieden werden. Ein weiterer Teil der False-Positives konnte aber darauf zurückgeführt werden, dass vermeintliche Entitäten in einzelnen Wörtern eines Eigennamens gefunden wurden, wenn der Eigenname als ganzes nicht in der Ontologie enthalten ist. Zum Beispiel wurden oft die Vor- und Nachnamen von Personen, die es nicht in der Ontologie gibt, fälschlicherweise als zwei einzelne Namen erkannt.

Hierzu wurden zwei Ansätze getestet, um dieses Problem abzumildern.

4.3.1. NIL-Vorhersage

Der erste Ansatz basiert auf der Beobachtung, dass zwei verschiedene Eigennamen nur selten direkt hintereinander stehen.
Wenn der Entitätserkenner zwei Eigennamen gefunden hat, die direkt nebeneinander stehen, ist dies möglicherweise ein Indiz dafür, dass es sich um eine Fehlerkennung handelt.
Daher wurde eine Art *NIL*-Vorhersage hinzugefügt, die als Entität erkannte Einzelworte immer dann für ungültig (*NIL*) erklärt, wenn direkt davor oder danach ebenfalls ein Eigenname steht. Zwei Eigennamen werden dabei als nebeneinanderstehend betrachtet, wenn sich dazwischen kein Satzzeichen oder eine anderes Wort befindet. Der Entitätserkenner gibt in einem solchen Fall zurück, dass an dieser Stelle keine Entität gefunden wurde.

Darüber hinaus werden einmal als *NIL* erkannte Eigennamen in einem Dokument in einer Liste gespeichert. Taucht ein Eigenname im weiteren Verlauf eines Dokumentes ein weiteres Mal auf, werden diese als Koreferent aufgefasst und ebenfalls als *NIL* erkannt.

4.3.2. Stanford Named-Entity-Recogizer

Der Stanford NER markiert Eigennamen im Text, sodass sich daraus die Anfangs- und Endpositionen der Eigennamen herleiten lassen. Zusätzlich werden die Eigennamen klassifiziert.
Indem die erkannten Anfangs- und Endpositionen bei der Suche berücksichtigt werden, kann verhindert werden, dass einzelne Teilworte eines Eigennamens fälschlicherweise erkannt werden. So darf eine Entität nur erkannt werden, wenn die Textstelle entweder mit einer NER-Annotation übereinstimmt oder diese vollständig enthält. Letzteres war notwendig, da die NER-Annotationen in einigen Fällen nicht den gesamten Eigennamen abdecken (z.B. in „South Philadelphia High School“ wurde lediglich „South Philadelphia“ annotiert).

Der Stanford NER bietet verschiedene Klassifikations-Modelle zur Auswahl, die jeweils auf unterschiedlichen Datensätzen trainiert wurden. Nach einem kurzen Experiment mit der Web-Applikation[1] wurde das Modell auf Basis von CoNLL ausgewählt.

4.4. Disambiguation von Mehrdeutigkeiten

Falls sich zwei Entitäten teilweise überschneiden, wird wie im bisherigen Ansatz immer die Entität beibehalten, die sich auf den längeren Namen bezieht.

Allerdings kann es im Gegensatz zum bisherigen Ansatz auch vorkommen, dass sich zwei Entitäten auf eine identische Textstelle beziehen. In einem solchen Fall müssen diese disambiguiert werden. Ein triviales Disambiguationsverfahren besteht darin, immer die Entität zu wählen, die den höchsten Popularitätswert hat. Das führt im Schnitt zu einem guten Ergebnis, ist aber vor allem problematisch, wenn es zwei oder mehr Entitäten mit ähnlichem Popularitätswert gibt — hier ist Wahrscheinlichkeit einer Fehlentscheidung am größten, wenn man nur den Popularitätswert berücksichtigt.

Ein präziseres Verfahren ist es, zusätzlich den Kontext zu berücksichtigen, in dem der Name verwendet wird, wie in Kapitel 3 beschrieben.

Beispielsweise soll im folgenden Satz die Entiät für „Michael Richards“ bestimmt werden. Hier könnte es sich einerseits um den gleichnamigen Ice-Hockey Spieler, andererseits um den Schauspieler und Stand-up Comedian Michael Richards handeln, die beide einen Popularitätswert in ähnlicher Größenordnung haben (19 451 zu 10 438).

> ***Michael Richards**, who played Jerry Seinfeld's eccentric neighbor Kramer on the sitcom "Seinfeld", publicly apologized for racist comments he made [...] he was working as a stand-up comic.*

[1] http://nlp.stanford.edu:8080/ner/

Abbildung 4.2 zeigt jeweils die kontextbezogene Übereinstimmung zwischen den Entitäten und obigem Satz, was als Entscheidungsgrundlage für den Disambiguationsalgorithmus genutzt werden kann.

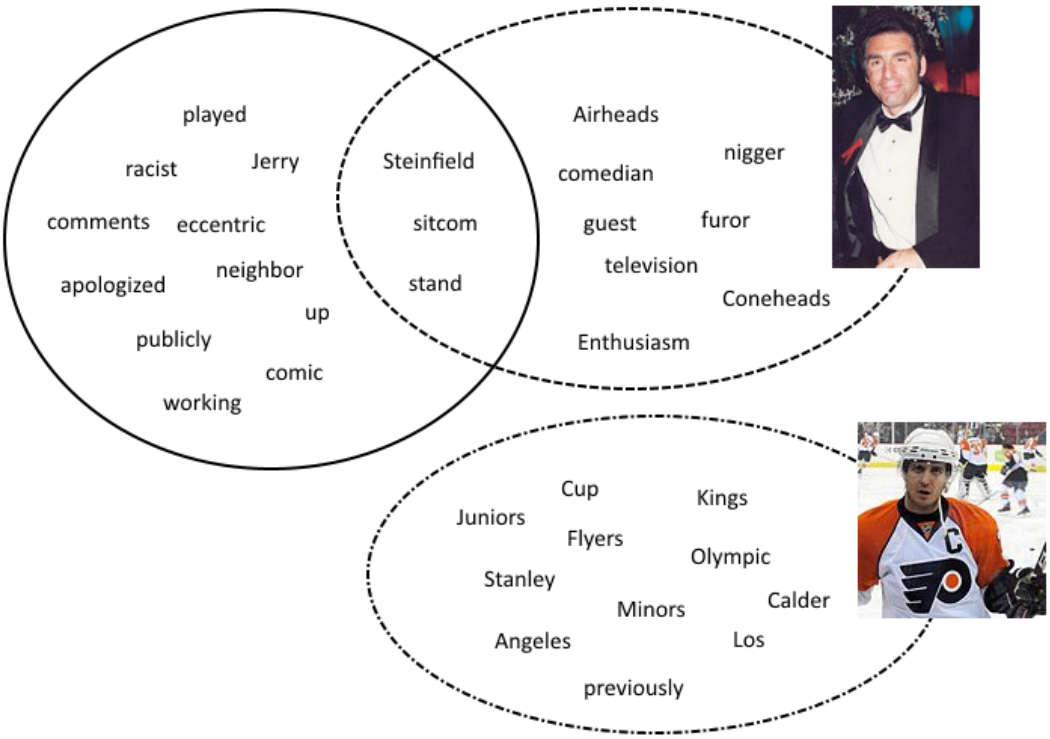

Abbildung 4.2.: Die Worte aus dem ersten Satz des News-Artikels (durchgezogener Kreis) zusammen mit einer Auswahl an Worten aus dem Wikipedia-Artikel des Schauspielers Michael Richards (oben), sowie des Ice-Hockey Spielers Michael "Mike" Richards (unten). Bildnachweis: Freebase.com

4.4.1. Kontext der Entitäten

Der Kontext einer Entität wird aus der Beschreibung der Entität aus Freebase gewonnen. Hierbei handelt es sich meistens um die Einleitung eines dazugehörigen Wikipedia-Artikels. Dabei muss berücksichtigt werden, dass es zu einer Vielzahl an eher unbekannten Entitäten keine oder nur eine sehr kurze Beschreibungen gibt.

Um die wichtigsten Schlüsselworte einer Entität in der dazugehörigen Beschreibung zu finden, werden die einzelnen Worte darin nach Tf-idf gewichtet und nur eine feste Anzahl der Worte mit den höchsten Tf-idf Werten wird für jede Beschreibung beibehalten. Dies soll zum einen den Speicherverbrauch verringern und die Performance erhöhen. Auf der anderen Seite hat dies den positiven Nebeneffekt, dass Stoppworte herausgefiltert werden.
Da das Berechnen der Tf-Idf Werte bei einer großen Zahl an Beschreibungen einen

hohen Arbeitsspeicherverbrauch nach sich zieht, wurde dieser Schritt unabhängig von der UIMA-Pipeline mit einem in C++ geschriebenen Programm durchgeführt. Die dadurch ermittelten Schlüsselworte werden zusammen mit den Tf-Idf Werten in eine Datei geschrieben, welche anschließend vom Entitätserkenner ohne großen Rechenaufwand nur noch eingelesen werden muss.

Innerhalb der UIMA-Pipeline werden die Schlüsselwörter für jede Entität gemäß dem *Vector Space Model* in Vektoren gespeichert, wobei für jedes Wort der dazugehörige Tf-Idf Wert verwendet wird. Da es für jede Entität nur eine sehr kleine Anzahl an Einträgen gibt, werden die Einträge in sogenannten dünnbesetzten Vektoren (engl. *sparse vectors*) gespeichert.

4.4.2. Kontext des Artikels

Für einen im Artikel erwähnten Eigennamen wurden jeweils zehn umliegende Sätze sowie die ersten fünf Sätze des Artikels als Basis für den Kontext-Vektor genutzt. Damit soll es einerseits möglich sein, dass in einem Dokument verschiedene Vorkommen eines Eigennamens zu verschiedenen Entitäten gelinkt werden kann. Andererseits wurde aber beobachtet, dass am Anfang des Artikels häufig für die Disambiguation wichtige Schlüsselworte stehen.

Der Kontext des Artikels wird analog in dünnbesetzten Vektoren gespeichert, wobei alle Adjektive, Nomen und Verben berücksichtigt werden. Dabei wird jeweils die Anzahl der Vorkommen jedes Wortes gespeichert. Die Informationen über die Wortart werden aus den POS-Tags gewonnen. Die Beschränkung auf bestimmte Wortarten soll als zusätzlicher Schutz gegen Stoppworte dienen.

4.4.3. Disambiguationsalgorithmus

Die Ähnlichkeit zweier Vektoren wird hier durch das Skalarprodukt der beiden Vektoren bestimmt. Das Skalarprodukt zweier Vektoren ist umso größer, je mehr gemeinsame Wörter in den durch die Vektoren repräsentierten Kontexten vorhanden sind, und ist 0, wenn beide Kontexte keine Gemeinsamkeiten haben. Auf eine Längennormalisierung der Vektoren wird, ähnlich wie in [Cuc07], verzichtet, da dies einerseits den Rechenaufwand verringert. Andererseits werden Entitäten mit einer sehr kurzen, und daher unter Umständen weniger aussagekräftigen Beschreibung entsprechend geringer gewichtet.

Sei c_{doc} der Kontext-Vektor des Dokumentes und c_{entity} der Kontext-Vektor der Entität. Die Ähnlichkeit beider Vektoren lässt sich schreiben als:

$$\text{similariy}(c_{doc}, c_{entity}) = c_{doc} \cdot c_{entity}$$

Der zweite Faktor des Disambiguationsalgorithmus ist der Popularitätswert der Entität. Zur Berechnung des Score-Wertes, welcher letztlich zur Disambiguation verwendet wird, wird das Produkt aus Popularität und Ähnlichkeitsmaß gebildet, wobei

zu dem Ähnlichkeitsmaß zuvor noch ein konstanter Wert $\epsilon > 0$ addiert wird. Dies soll verhindern, dass das Produkt 0 wird, wenn das Ähnlichkeitsmaß 0 ist. Zudem kann mit einer geeigneten Wahl von ϵ der Popularität ein höheres Gewicht gegeben werden. Gelegentlich konnte beobachtet werden, dass eine sehr wenig populäre Entität mit wenig aussagekräftiger Beschreibung nur zufällige Ähnlichkeit mit dem Dokument aufweist, während die eigentlich gemeinte weitaus populärere Entität kaum Übereinstimmungen aufweist. Da die Vektoren nicht normalisiert werden, wurde $\epsilon = 5$ festgesetzt, was in einem kurzen Experiment die besten Ergebnisse erzielte.

Der Score-Wert einer Entität, gegeben der Kontext des Dokumentes, lässt sich folglich schreiben als:

$$\text{score}_{doc}(entity) = (\text{similariy}(c_{doc}, c_{entity}) + \epsilon) \cdot \text{popularity}(entity)$$

Letztlich wird folgendes Maximierungsproblem gelöst, um die Entität aus einer Liste von Kandidaten E für ein Dokument *doc* zu finden:

$$\arg \max_{entity \in E} \text{score}_{doc}(entity)$$

Algorithmus 2 zeigt eine grobe Skizze des Disambiguationsalgorithmus. Die Kontext-Vektoren sind dabei als Hash-Map gespeichert. Diese bilden von einem Wort auf den jeweiligen Score (bei der Beschreibung der Entitäten) bzw. auf die Anzahl der Vorkommen (bei den Dokumenten) ab.

Die Funktion *Similarity* hat konstante Laufzeit, wenn davon ausgegangen wird, dass es für jede Entität eine konstante Zahl an Einträgen im Kontext-Vektor gibt. Die Kontext-Vektoren für die Entitäten werden bereits beim Einlesen der Ontologie erstellt.

Somit ist die Laufzeit insgesamt linear in der Anzahl der Entitäten für einen Disambiguationsvorgang (exkl. Berechnung des Kontext Vektors des Dokumentes).

4.5. Koreferenzen

Die bereits vorhandenen Ansätze zum Erkennen von Koreferenz wurde übernommen und erweitert, sodass auch Koreferenzen der Art „the actor“, „the school“ oder „the president“ erkannt werden.

Dazu wird in einem Dokument eine Liste aller bisher erkannten Entitäten geführt. Falls in einem Dokument das Wort „the“ gefolgt von einem Nomen (NN- oder NNS-Tag) auftaucht, wird geprüft, ob das Nomen eine Klasse von einer der bisher erkannten Entitäten ist.
Beispielsweise ist der Schauspieler „Michael Richards“ eine Entität der Klasse *actor*. Daher würden Vorkommen von „the actor“ in einem Dokument, wenn zuvor der Schauspieler „Michael Richards“ erkannt wurde, als Koreferenz zu Michael Richards aufgefasst. Falls es mehrere Entitäten zu einer Klasse gibt, wird immer die zuletzt erkannte Entität gewählt.

```
input  : A document context vector D
input  : A list of candidates E
output : An entity best

best ← nil
for entity ∈ E do
    if best = nil then
        best ← entity
    end
    else
        /* Take the one with the higher score value */
        score1 ← (Similarity(D, entity.contextVector) + ϵ) · entity.popularity
        score2 ← (Similarity(D, best.contextVector) + ϵ) · best.popularity
        if score1 > score2 then
            best ← entity
        end
    end
end
return best

Function Similarity(context1, context2)
    larger ← The larger of the two context vectors
    smaller ← The smaller of the two context vectors
    score ← 0
    for entry1 ∈ smaller do
        entry2 ← larger.get(entry1.key)
        score ← score + entry1.value · entry2.value
    end
    return score
```

Algorithmus 2 : Algorithmus zur Disambiguation.

5. Evaluation

5.1. Testdatensätze

5.1.1. Eigene Auswahl

Zur Evaluation wurden zufällig Nachrichten-Artikel aus dem gesamten Korpus ausgewählt, der aus etwa 600 Tausend Artikeln besteht. Dabei wurden Artikel ausgelassen, welche keine oder nur sehr wenige Eigennamen aus der Ontologie enthalten. Insgesamt wurden 16 Artikel aufgenommen, die mit über 670 Annotationen zu Eigennamen und 200 Annotationen zu Koreferenzen versehen wurden, wie im folgenden beschrieben.

Für jeden Artikel wurden die Eigennamen mit Grotund Truth Markern versehen, die ähnlich wie die Wikipedia-Links formatiert sind: Der Eigenname wird zweifach in eckige Klammern gesetzt. Falls der Name der Entität von der Schreibweise im Text abweicht, wird diese durch einen senkrechten Strich getrennt vor den Eigennamen geschrieben. Koreferenzen wurden ebenfalls annotiert und mit einem Asterisk-Symbol versehen, um später eine getrennte Evaluation mit und ohne Koreferenzen durchführen zu können.

Im Folgenden ein beispielhafter Satz mit Grotund Truth Markern:

```
[[CC_Sabathia|C.C. Sabathia]] has been one of the best pitchers
in baseball this season, but watching [[CC_Sabathia*|him]] this
[[October]], who would know?
```

5.1.2. ERD-50

Der ERD-50 Korpus ist eine Untermenge des bei der [ERD2014] verwendeten Korpus für den *"long text track"*. Der tatsächlich für die Challenge verwendete Korpus (ERD-100) umfasst weitere 50 Dokumente, die jedoch nicht veröffentlicht wurden.
Der ERD-50 Korpus wurde nach dem Wettbewerb öffentlich zugänglich gemacht[1].

Die Basis für den Korpus bilden zur Hälfte Nachrichten-Artikel aus dem Webportal MSN, und zum anderen nicht näher benannte Dokumente aus dem Internet. Annotiert wurden ausschließlich Eigennamen, und keine Koreferenzen.

[1] https://github.com/aolieman/sigir-erd-14-server

Der Datensatz beinhaltet 1 100 Ground Truth Annotationen, teilweise sind aber auch Artikel ohne eine einzige Annotation enthalten.

Die Ground Truth Annotationen sind beim ERD-50 Korpus in einer separaten Datei gespeichert, ähnlich den ClueWeb annotations[2].

Die Annotationen beziehen sich allerdings nur auf eine Teilmenge der Freebase-Datenbank, während für die Entitätserkennung die gesamte Datenbank verwendet wird. Da der ERD-50 Korpus erst sehr spät veröffentlicht wurde und es lange Zeit auch unklar war, ob dieser überhaupt noch rechtzeitig für diese Arbeit veröffentlicht wird, konnte darauf nicht mehr rechtzeitig reagiert werden. Dies muss bei der weiteren Evaluation berücksichtigt werden.

[2]http://web-ngram.research.microsoft.com/erd2014/Datasets.aspx

5.2. Testmethodik

Zur Auswertung wurde eine eigene Evaluationskomponente geschrieben, welche die Dokumente und Ground Truth Marker einliest, die Dokumente an den Entitätserkenner übergibt und anschließend die erkannten Entitäten mit den Ground Truth Markern vergleicht.

Hierbei wird eine Ground Truth Annotation als „richtig erkannt“ gewertet, wenn die richtige Entität erkannt wurde und ihre Position genau mit der Position der Ground Truth Annotation übereinstimmt. Ground Truth Annotationen, welche zwar richtig erkannt wurden, die Position der Entität aber nur teilweise überlappen, werden als „partiell richtig erkannt“ gezählt.

Ground Truth Annotationen, zu welchen keine Entität erkannt wurde, werden als „nicht erkannt“ gewertet.

Entitäten werden als „falsch erkannt“ gezählt, wenn diese eine anderslautende Ground Truth Annotation überlappen bzw. als „False-Positive“ (FP), wenn diese gar keine Ground Truth Annotation überlappen.

Zusätzlich wird für jedes Dokument Precision, Recall und daraus das F-Measure (F1) als gewichtetes harmonischen Mittel berechnet.

Ähnlich dem Regelwerk der [ERD2014] (“*...relaxed correctness of the entity mention boundaries...*”) werden bei der Berechnung von Precision und Recall die partiellen Übereinstimmungen als vollwertige Übereinstimmungen gezählt:

$$Precision = \frac{\#\text{Richtig erkannt} + \#\text{Partiell richtig erkannt}}{\#\text{Entitäten}}$$

$$Recall = \frac{\#\text{Richtig erkannt} + \#\text{Partiell richtig erkannt}}{\#\text{Ground Truth Annotationen}}$$

$$F - measure = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Abbildung 5.1 zeigt eine beispielhafte Ausgabe der Evaluationskomponente.

Precision, Recall und das F-Maß wurden dabei einmal pro Dokument und einmal insgesamt berechnet. Bei der Berechnung der Gesamtwerte wurden jeweils die Kennzahlen (Zahl der richtig erkannten Entitäten, etc.) der einzelnen Dokument aufaddiert.

Zudem wurden bei der Evaluation nur Eigennamen miteinbezogen. Andere Wortarten, wie etwa Gattungsnamen, werden nur angezeigt, fließen aber nicht in die

Berechnung der Maßzahlen mit ein. Dadurch soll die Evaluation kompatibel und vergleichbar zu externen Datensätzen (wie dem ERD-50) bleiben, da in den meisten Fällen nur Eigennamen annotiert sind.

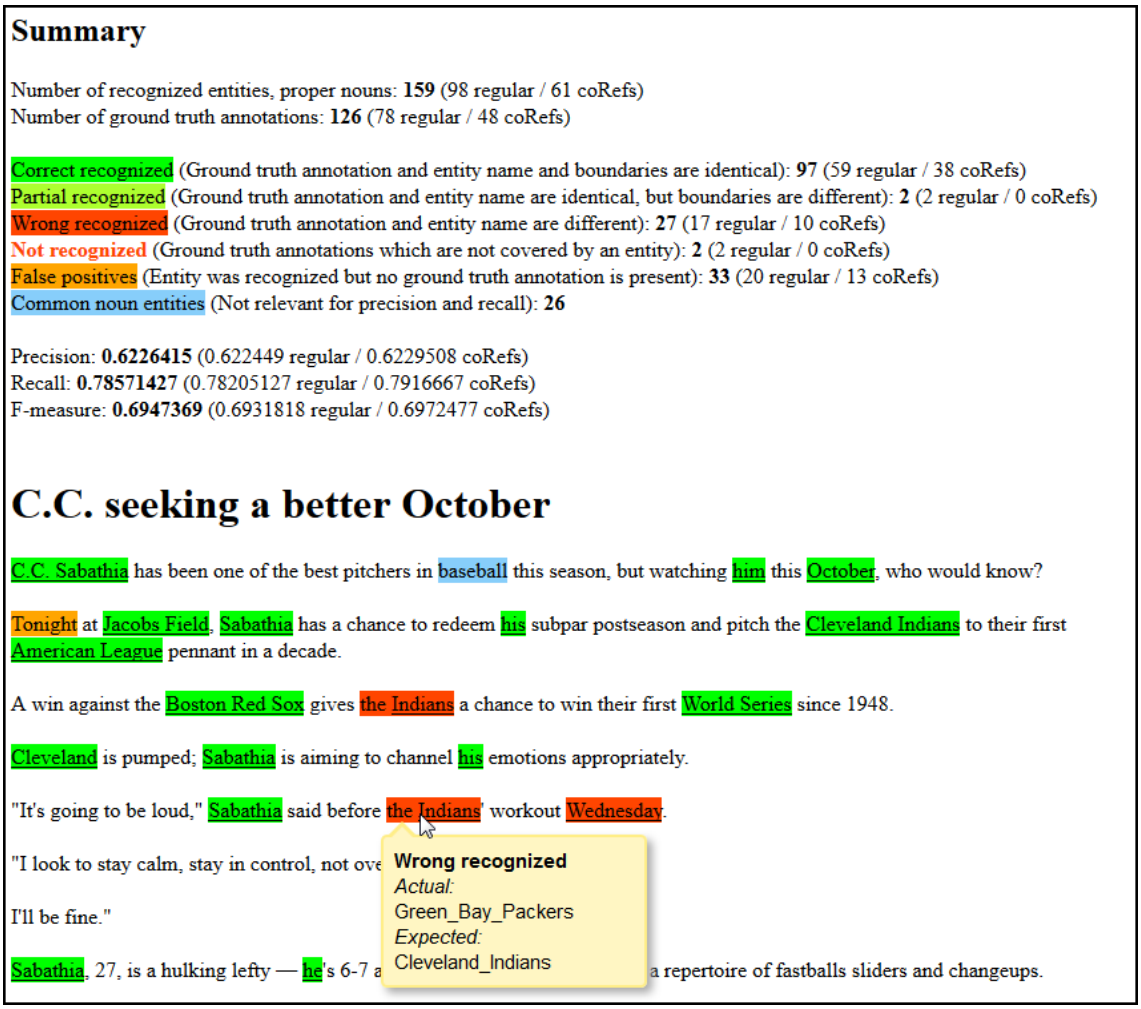

Abbildung 5.1.: Die Ausgabe der Evaluations-Komponente. Es wird eine Datei pro Dokument erstellt, die die erkannten Entitäten mit der Ground Truth vergleicht und das Ergebnis graphisch und mit Hilfe von Tooltips darstellt.

5.3. Auswertung

Tabelle 5.1 zeigt die Ergebnisse der im letzten Abschnitt beschriebenen Evaluations-Komponente. Es wurde die „NIL-Vorhersage“ sowie die Disambiguation mit Kontext-Vektoren verwendet. Der F1-Wert lag bei 0,682 bei der Eigenen Auswahl bzw. 0,536 bei ERD-50.

	#E	#G	Korrekt	Partiell	Falsch	Nicht	FP
Eigene Auswahl	995	881	617	23	187	58	168
ERD-50	1589	1164	649	89	261	192	590

Tabelle 5.1.: Ausgabe der Evaluations-Komponente.

Im Folgenden werden die Ergebnisse für die einzelnen Verbesserungen gezeigt, sowie eine Analyse der noch verbleibenden Falscherkennungen. Für die Eigene Auswahl beziehen sich die Werte auf den Gesamt-Score mit Koreferenzen. Dadurch werden z.B. Folgefehler aufgrund falscher Disambiguation besser sichtbar. Eine getrennte Betrachtung der Koreferenzen findet in Unterabschnitt 5.3.3 statt.

5.3.1. Finden von Eigennamen

Zunächst wurden die verschiedenen Ansätze zum Finden von Eigennamen evaluiert. „Unbeschränkt“ beschreibt hierbei den Basis-Ansatz, bei dem sämtliche im Text vorkommenden als *proper nouns* klassifizierte Wörter erkannt werden dürfen. „Nil-Vorhersage“ und „Stanford NER“ beschreiben die in Unterabschnitt 4.3.1 bzw. Unterabschnitt 4.3.2 beschriebenen Verbesserungen.
Der „ideal“-Wert bezieht sich darauf, dass dem Entitätserkenner die Anfangs- und Endpositionen der Entitäten vorgegeben wurden, nicht jedoch die Entität selbst. Dies ist somit ein rein theoretischer Wert.

	Eigene Auswahl			*ERD-50*		
	Precision	Recall	F1	Precision	Recall	F1
Ideal	*0,814*	*0,743*	*0,777*	*0,771*	*0,637*	*0,698*
Unbeschränkt	0,583	**0,734**	0,650	0,316	**0,678**	0,431
Nil-Vorhersage	**0,643**	0,726	**0,682**	0,464	0,636	**0,536**
Stanford NER	0,640	0,663	0,651	**0,551**	0,514	0,532

Tabelle 5.2.: Vergleich verschiedener Varianten zum Finden von Eigennamen.

Die unbeschränkte Variante erzielt erwartungsgemäß den höchsten Recall, aber eine niedrige Precision aufgrund vieler False-Positives, vor allem auf dem ERD-50 Korpus. Die beiden erweiterten Ansätze können einen Teil davon wirkungsvoll vermeiden. Der Abstand zum idealen Wert zeigt aber, dass es nach wie vor viele False-Positives gibt.

Bei der NIL-Vorhersage konnte nur ein leichter Rückgang des Recall-Wertes beobachtet werden. Insgesamt erzielte diese Variante somit auf beiden Datensätzen das beste Ergebnis.
Anders sieht es bei der Verwendung des Stanford NER aus. Hier wurden einige der Entitäten nicht mehr richtig erkannt. Abbildung A.1 zeigt beispielhaft die Annotationen des Stanford NER. Unter anderem nicht erkannt, aber als Ground Truth Annotation vorhanden war „Assistant Attorney General“.

Darüber hinaus wurden auch viele Entitäten nicht erkannt, weil der im Dokument verwendete Name keine Oberflächenform der Entität ist. Der in Abschnitt 4.1 beschriebene Ansatz erwies sich diesbezüglich als nicht besonders erfolgreich, da nur sehr wenige zusätzliche Entitäten erkannt wurden, wie in Tabelle 5.3 zu sehen ist.

	Eigene Auswahl			*ERD-50*		
	Precision	Recall	F1	Precision	Recall	F1
Ohne zusätzliche Aliase	0,643	0,724	0,681	**0,465**	0,633	0,536
Inkl. zusätzlicher Aliase	0,643	**0,726**	**0,682**	0,464	**0,636**	0,536

Tabelle 5.3.: Auswertung der Erzeugung zusätzlicher Aliase.

Dass fehlende Aliase in der Ontologie tatsächlich ein Problem sein könnten, kann indirekt aus Tabelle 5.5 geschlossen werden. Bei idealem Finden der Eigennamen und idealer Disambiguation sind nicht erkannte Entitäten hauptsächlich auf fehlende Oberflächenforen in der Ontologie zurückzuführen.
Bei einer manuellen Analyse der *Eigenen Auswahl* konnten 70 Fälle gefunden werden, bei denen die Entität aufgrund einer fehlenden Oberflächenform nicht oder falsch erkannt wurde. Die entspricht mehr als 10% aller im Text vorkommenden Entitäten (ohne Koreferenzen).

5.3.2. Disambiguation

Die Disambiguation wurde einmal unter realen Bedingungen („NIL-Vorhersage“), sowie einmal unter idealisierten Bedingungen getestet. Bei letzterem wird ähnlich zu [MC07] ein ideales Finden der Eigennamen angenommen.
Bei der Variante „Popularitätswert“ wurde immer die Entität mit dem höheren Popularitätswert ausgewählt. „Kontext“ bezieht sich auf den in Abschnitt 4.4 beschriebenen Algorithmus.
Zusätzlich gibt es den Referenzwert *„Ideal“* unter der Annahme, dass der Disambiguationsalgorithmus immer die richtige Entität auswählt, sofern diese in der Liste der Kandidaten enthalten ist.

	Eigene Auswahl			*ERD-50*		
	Precision	Recall	F1	Precision	Recall	F1
Ideal	*0,674*	*0,757*	*0,713*	*0,475*	*0,651*	*0,549*
Popularitätswert	0,612	0,70	0,653	**0,465**	**0,638**	**0,538**
Kontext	**0,643**	**0,726**	**0,682**	0,464	0,636	0,536

Tabelle 5.4.: Auswertung der Disambiguation.

	Eigene Auswahl			*ERD-50*		
	Precision	Recall	F1	Precision	Recall	F1
Ideal	*0,850*	*0,776*	*0,811*	*0,794*	*0,656*	*0,718*
Popularitätswert	0,780	0,712	0,744	**0,778**	**0,643**	**0,704**
Kontext	**0,815**	**0,743**	**0,777**	0,771	0,637	0,698

Tabelle 5.5.: Auswertung der Disambiguation bei idealem Finden der Eigennamen.

Auf der Eigenen Auswahl konnte hier ein deutlicher Fortschritt erzielt werden. Allerdings gibt hier nach wie vor Entitäten, die nicht richtig disambiguiert wurden, was der Abstand zum Idealwert zeigt. Zu den auch mit Kontext-Informationen nicht richtig disambiguierten Entitäten gehört z.B. „President Bush“, was sowohl für „George W. Bush“ als auch für „George H. W. Bush“ stehen könnte. Beide Entitäten haben einen ähnlichen Kontext und sind daher mit dieser Methode nur schwer auseinanderzuhalten.
Auf dem ERD-50 Korpus liegt dir Variante „Popularitätswert“ bereits nahe am Idealwert. Die Einbeziehung der Kontext-Informationen bringt hier sogar eine leichte Verschlechterung. Dies zeigt, dass viele der Eigennamen bereits eindeutig oder leicht zu disambiguieren sind, während die übrigen Fälle selbst durch Einbeziehung der Kontext-Informationen nicht richtig disambiguiert werden können.

5.3.3. Koreferenz

Tabelle 5.6 zeigt die Auswertung getrennt nach Koreferenz und den übrigen Entitäten.

	Precision	Recall	F1
Gesamt	*0,643*	*0,726*	*0,682*
Ohne Koreferenz	0,708	0,745	0,726
Nur Koreferenz	0,481	0,665	0,558

Tabelle 5.6.: Auswertung der Koreferenz auf der Eigenen Auswahl.

Koreferenz wurde häufig falsch erkannt. Unter anderem gibt es bei dem Pronomen „it“ das Problem, dass dieses oft als Expletivum gebraucht wird und sich daher nicht auf eine Entität bezieht.

5.4. Performanceanalyse

Ein wichtiger Aspekt ist zudem die Laufzeit und der Arbeitsspeicherverbrauch. Zur Performanceanalyse wurden neben den 600 Tausend Nachrichten-Seiten zusätzlich die über 3 Millionen Seiten der .edu Top-Level-Domain einbezogen, wie das bereits in der ursprünglichen Arbeit [Bau14] der Fall war. Als Referenzsystem diente wie in der bisherigen Arbeit ein Intel System mit 24 logischen Kernen, das auf 96GB Arbeitsspeicher zugreifen konnte.

Für das Speichern der Entitäten inklusive der Kontext-Informationen werden etwa 45GB Arbeitsspeicher benötigt. Zusätzliche 25GB werden im Verlaufe der Verarbeitung der knapp 4 Millionen Seiten benötigt, sodass der Arbeitsspeicherverbrauch in der Spitze bis zu 70GB beträgt.

Der gesamte Vorgang dauert etwa 17 Stunden. Davon fallen jedoch lediglich 15% der Zeit für die Entitätserkennung an. Ein Großteil der Rechenzeit wird für das *POS-Tagging* benötigt ($> 50\%$), sowie für die Komponente zum Schreiben des Volltext-Index für *broccoli* (etwa 25%). Der Stanford NER konnte aus Zeitgründen nicht auf dem vollen Korpus getestet werden. In einer kleineren Auswahl zeigt sich aber, dass der Rechenaufwand mindestens in der Größenordnung des *POS-Taggers* liegt.

6. Fazit

In dieser Arbeit wurden zahlreiche Ansätze zur Verbesserung der Entitätserkennung beschrieben und evaluiert. Dabei konnten Fortschritte bei der Vermeidung von False-Positives erzielt werden. Ebenso konnte die *Entity Disambiguation* durch Einbeziehung von Kontext-Informationen verbessert werden. Jedoch konnten nicht alle Faslcherkennungen behoben werden. Probleme gibt es nach wie vor mit False-Positives. Zudem ist der in einem Dokument verwendete Name für eine Entität oft nicht als Alias in der Ontologie-Datenbank vorhanden. Dadurch wurden einige Entitäten nicht erkannt.

A. Anhang

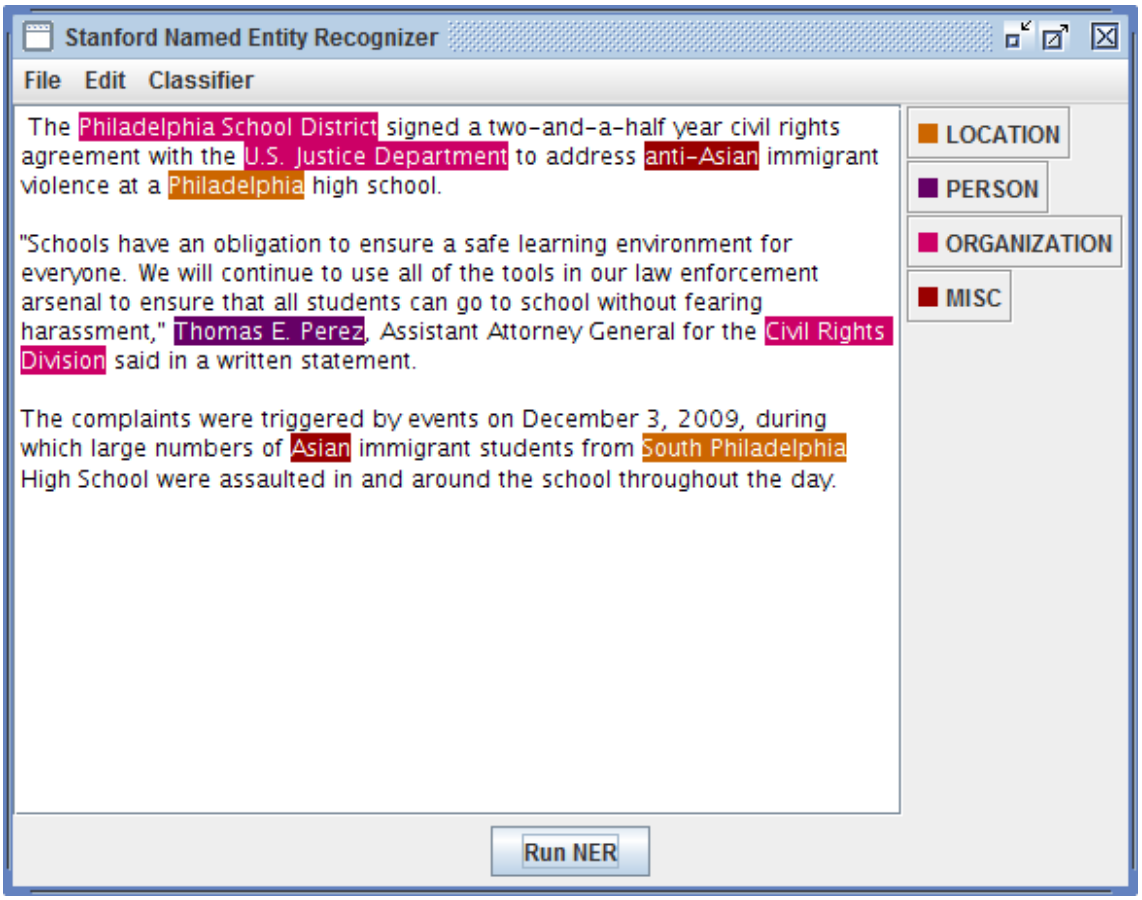

Abbildung A.1.: Beispielhafte Ausgabe des Stanford NER, die mit der Demo-Applikation erstellt wurde.

Literaturverzeichnis

[Bau14] Bausch, Philipp: *Semantische Suche auf einem WEB-Korpus*, Albert-Ludwigs-Universität Freiburg im Breisgau, Bachelorarbeit, 2014

[Cuc07] Cucerzan, Silviu: Large-scale named entity disambiguation based on Wikipedia data. In: *In Proc. 2007 Joint Conference on EMNLP and CNLL*, 2007, S. 708–716

[ERD2014] Carmel, David ; Chang, Ming-Wei ; Gabrilovich, Evgeniy ; Hsu, Bo-June ; Wang, Kuansan: ERD 2014: Entity Recognition and Disambiguation Challenge. In: *SIGIR Forum* (2014)

[FGM05] Finkel, Jenny R. ; Grenager, Trond ; Manning, Christopher: Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. In: *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*. Stroudsburg, PA, USA : Association for Computational Linguistics, 2005 (ACL '05), S. 363–370

[LME12] Lin, Thomas ; Mausam ; Etzioni, Oren: Entity Linking at Web Scale. In: *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction*. Stroudsburg, PA, USA : Association for Computational Linguistics, 2012 (AKBC-WEKEX '12), S. 84–88

[LPLA10] Lommatzsch, Andreas ; Ploch, Danuta ; Luca, Ernesto William D. ; Albayrak., Sahin: Named Entity Disambiguation for German News Articles. In: Atzmüller, Martin (Hrsg.) ; Benz, Dominik (Hrsg.) ; Hotho, Andreas (Hrsg.) ; Stumme, Gerd (Hrsg.): *Proceedings of LWA2010 - Workshop-Woche: Lernen, Wissen & Adaptivitaet*. Kassel, Germany, 2010

[MC07] Mihalcea, Rada ; Csomai, Andras: Wikify!: Linking Documents to Encyclopedic Knowledge. In: *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*. New York, NY, USA : ACM, 2007 (CIKM '07). – ISBN 978–1–59593–803–9, S. 233–242