Beuth Hochschule für Technik

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Fachbereich VI
Informatik und Medien

Bachelorarbeit

Verbessertes Entity Linking mit neuronalen Word Embeddings

Robert Dziuba
Beuth Hochschule für Technik
Studiengang Medieninformatik (B.Sc.)
Matrikelnummer 821303

1. Gutachter Prof. Dr. habil. Alexander Löser
Fachbereich VI - Informatik und Medien
Beuth Hochschule für Technik

2. Gutachter Prof. Dr.-Ing. Joachim Schimkat
Fachbereich VI - Informatik und Medien
Beuth Hochschule für Technik

Betreuer M.Sc. Sebastian Arnold
Fachbereich VI - Database Systems and Text-based Information Systems (DATEXIS)
Beuth Hochschule für Technik

13. November 2017

Zusammenfassung

Eine große Herausforderung in der Computerlinguistik ist die Verlinkung eines gesuchten Wortes mit einer Entität aus einer Wissensbasis, der sogenannten Knowledge Base. Oft wird erst durch den Kontext, in dem ein Wort eingebettet ist, sein Sinn deutlich. In dieser Arbeit sollen Entitäten aus der Knowledge Base mit der Hilfe eines Paragraph Vector Modells, einem neuronalen Netz, in einem Vektorraum abgebildet werden. Diese Abbildungen, auch Word Embeddings genannt, werden dann mit dem Kontext des gesuchten Wortes verglichen und die Entität mit der höchsten Übereinstimmung mit diesem Wort verlinkt. Es wird gezeigt, dass mit dieser Methode bessere Ergebnisse in der Disambiguierung erreicht werden, als mit einer normalen Volltextsuche auf der Knowledge Base.

Abstract

A major challenge in computational linguistics is the linking of a mention with an entity from a knowledge base. Frequently, the meaning of a word becomes clear by regarding the context in which the word is embedded. In this thesis, entities from the knowledge base are mapped to a vector space using a neural network, called paragraph vector.
Accordingly, these word embeddings are compared to the context of the mention and get linked to the entity with the most likely context. It is shown that this method achieves better results in disambiguation than a normal full text search on the knowledge base.

Inhaltsverzeichnis

1 Einführung

In diesem Kapitel soll der Leser als erstes mit dem Problem des *Entity Linkings* sensibilisiert werden. Danach wird das Ziel dieser Arbeit definiert und die Methodiken, um dieses Ziel zu erreichen, werden benannt. Zum Ende wird eine Übersicht zu den einzelnen Gliederungspunkten gegeben.

1.1 Problemstellung

> *You shall know a word by the company it keeps!*
>
> **— J. R. Firth**
> (Linguist)

Dinge definieren sich über den Kontext, der sie beschreibt. Nehmen wir z.B. einen Satz aus der US-amerikanischen Lokalzeitung *Indiana Gazette*:

„*Instead of Los Angeles International, for example, consider flying into Burbank or John Wayne Airport in Orange County, Calif., or use Westchester County Airport instead of JFK in New York.*“[1]

Wir wissen aus dem Kontext heraus, dass es sich bei *Burbank* um einen Ort in den USA handeln muss. Aber der Kontext enthält noch mehr Informationen. Durch das Verb *flying* und den anderen erwähnten Flughäfen können wir daraus schließen, dass es sich bei *Burbank* auch um einen Flughafen handeln muss. Und obwohl wir wahrscheinlich nicht wissen, was *Burbank* ist, so können wir es trotzdem einer konkreten Entität zuordnen, da es in einem Kontext „eingebettet“ ist.
Nach einer Suche im Internet finden wir heraus, dass *Burbank* eine Stadt in Los Angeles County ist. Und *Burbank* hat auch einen Flughafen. Den Bob Hope Airport. Aber wie können wir es schaffen, diese, für uns relativ einfachen Schlussfolgerungen, einer Maschine beizubringen?
In der Computerlinguistik wird das gesuchte Wort (Burbank) *Mention* und das Objekt, was es tatsächlich beschreibt (Bob Hope Airport), *Entity* genannt (vgl. [RMD13, S.

[1] Indiana Gazette vom 27. Dezember 1983, S. 39, gefunden im WNED-Dataset, /wned-datasets/ace2004/RawText/NYT20001217.2241.0165

1]). Das Auflösen einer Mention in eine konkrete Entity bezeichnet man als *Named-Entity Linking* oder auch *Entity Linking* (vgl. [SWH15, S. 443]). Entity Linking ist ein Teilgebiet der Computerlinguistik, an dem zur Zeit aktiv geforscht wird.

1.2 Ziel der Arbeit

In der vorliegenden Arbeit soll gezeigt werden, wie das Entity Linking durch neuronale *Word Embeddings* verbessert werden kann. Zu diesem Zweck soll das Disambiguierungensverfahren des *Tasty*-Projekts[2] optimiert werden. In diesem Verfahren werden durch eine Volltextsuche auf 3,5 Millionen verschiedene Entities aus der Wikipedia mögliche Kandidaten für eine Mention gesucht. Die Disambiguierung des besten Kandidaten erfolgt durch eine Ranking-Funktion eines *Lucene Index*[3] (vgl. [Arn+16]).
Es soll herausgestellt werden, welche Probleme bei der Disambiguierung von Entities auftreten können und wie ein neuronales Netz durch das Erlernen von Word Embeddings dabei helfen kann, diese Probleme zu lösen. Ziel des Word Embeddings ist es, den Kontext, in dem eine Entity oft benutzt wird, möglichst präzise zu erlernen. Diese erlernten Kontext-Charakteristika sollen dann benutzt werden, um Ähnlichkeiten im Mention-Kontext zu finden und dadurch auf die richtige Entity zu schließen.
Weiter sollen auf Basis mehrerer Goldstandard-Datensätze unsere Ergebnisse evaluiert werden. Abschließend wollen wir die Frage beantworten, ob neuronale Word Embeddings in der Lage sind, mit kontext-basierten Information die *Tasty*-Disambiguierung zu verbessern.

1.3 Methodik

Um eine Entity maschinell interpretieren zu können, werden Eigenschaften, die diese Entity definieren, in einem Vektorraum abgebildet. Diese Abbildungen, auch *Entity Embeddings* genannt, haben die Beschaffenheit, dass sie gleichartige Kontexte in ähnlichen Richtungen im Vektorraum projizieren. Diese Entity Embeddings sollen von einem neuronalen Netz gelernt werden und später dabei assistieren, zu einer Mention und seinem Kontext ähnliche, schon erlernte Entities, zu finden.

Zum Erlernen der Entity Embeddings wird das von Le und Mikolov [LM14] entwickelte *Paragraph Vector*-Modell angewendet. Es basiert auf der Grundlage des von Mikolov et al. [Mik+13a] vorgestellten *Word2vec*-Modells, welches bereits im Erlernen von neuronalen Word Embedding Anwendung findet. In ihrer Arbeit beschreiben

[2] `http://demo.datexis.com/tasty/`
[3] Lucene 6.1.0 `http://lucene.apache.org`

Le und Mikolov [LM14], wie sie die Repräsentation von Wörtern im Vektorraum auf Paragraphen erweitern. Diese Paragraphen können Text-Sequenzen beliebiger Länge sein, wie z.B Sätze, Absätze oder Dokumente (vgl. [LM14, S. 1]). In dieser Arbeit wird das Paragraph Vector-Modell erweitert, um Text-Sequenzen auf Entity IDs abzubilden.

Als Trainingskorpus dienen Artikel der englischen Wikipedia, aus denen Sätze, die mit Links auf andere Artikel verweisen, extrahiert werden. Dabei entsteht ein Datensatz mit 2.725.363 Entities und 63.812.227 Sätzen, auf dem das Netz die Entity Embeddings erlernen soll.

In einem Lucene Index wird zu einer Mention nach passenden Entities gesucht. Aus dem Suchergebnis, bestehend aus bis zu 1.000 Kandidaten, soll mit Hilfe des Paragraph Vector-Modells der Kandidat gefunden werden, dessen Repräsentation im Vektorraum der der Mention am ähnlichsten ist. Für diese Ähnlichkeitsüberprüfungen werden verschiedene Strategien entwickelt, die durch die Berechnung der *Cosine Similarity* den besten Kandidaten ermittelt. Dabei wird neben der Mention auch der Kontext, in dem die Mention erwähnt wird, mit betrachtet. Es soll gezeigt werden, dass durch simple Vektor Algebra der Disambiguierungprozess stark verbessert werden kann.

Zum Schluss sollen die Modelle und Strategien anhand eines Testdatensatzes überprüft und analysiert werden.

1.4 Gliederung und Aufbau

Diese Arbeit gliedert sich in folgende sechs Kapitel:

Kapitel 1: In der „Einführung“ wird auf die Problemstellung eingegangen und das Ziel der Arbeit definiert. Außerdem wird ein Überblick über die verwendeten Methodiken gegeben.

Kapitel 2: Weiter soll in „Grundlagen und verwandte Arbeiten“ ein Verständnis der aktuellen Techniken gegeben und Arbeiten aus dem gleichen Umfeld vorgestellt werden.

Kapitel 3: Danach wird in „Named Entity Linking mit Kontextinformation“ ein erster Ansatz diskutiert und auf allgemeine Probleme der Disambiguierung aufmerksam gemacht. Weiter wird das Problem formalisiert und es werden verschiedene Lösungsansätze diskutiert.

Kapitel 4: In der „Implementierung“ wird die technische Umsetzung erörtert und die Integration in das vorhandene Framework gezeigt.

Kapitel 5: Daraufhin werden in „Evaluierung“ Testumgebung und Metriken erklärt, sowie die Ergebnisse ausgewertet und besprochen. Danach erfolgt eine Analyse der erarbeiteten Lösungsansätze und dessen Probleme.

Kapitel 6: In „Zusammenfassung und Ausblick“ werden alle Schritte noch einmal zusammengefasst und auf Erkenntnisse aufmerksam gemacht. Abschließend werden zukünftige Schritte erläutert.

2 Grundlagen und verwandte Arbeiten

In diesem Kapitel wollen wir die Abbildung von Wörtern im Vektorraum kennenlernen. Des Weiteren sollen die Stärken von Word Embeddings gezeigt werden. Danach werden aktuelle Arbeiten aus der Forschung zu diesem Thema vorgestellt.

2.1 Grundbegriffe

Repräsentation von Wörtern im Vektorraum Um Wörter in einem Vektorraum darzustellen, werden alle Wörter aus einem Dokument als ungeordnete Menge in einem Wortvokabular abgespeichert. Die Position des Wortes im Wortvokabular wird als Koordinate im Vektorraum kodiert.

Local Representations In einer *Local* (oder *One-Hot-*) *Representation* wird jedes Wort in einem Wortvokabular T mit einer fester Größe, als ein binärer Vektor $\overrightarrow{v} \in \{0,1\}_T$ dargestellt. Dabei entspricht jede Position des Vektors $\overrightarrow{v}$ einem Wort. Wie in der Abbildung 2.1 dargestellt, wird das Wort *Burbank* durch einen Vektor repräsentiert, der an der Stelle des Wortes eine Eins hat und an den restlichen Stellen Nullen. Somit wird jedes Wort im Vektorraum unterschiedlich dargestellt, ist aber nicht mehr mit anderen Vektoren vergleichbar (vgl. [MC17, S.13]).

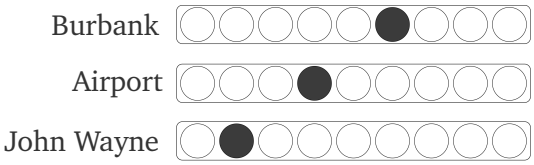


Abb. 2.1.: *Local represantation* der Wörter *Burbank*, *Airport* und *John Wayne* im Wortvokabular Vektor (vgl. [MC17, S. 13])

Distributed Representations Bei *Distributed Representations* wird ein Wort über mehrere Positionen definiert. Der Vektor $\overrightarrow{v} \in \mathbb{R}^{|k|}$ ist nicht mehr über eine einzelne Dimension interpretierbar. Die Positionen im Vektor $\overrightarrow{v}$ zeigen auf Wörter im Dokument, mit dem das Wort im Zusammenhang aufgetreten ist (vgl. [MC17, S.14]). In Abbildung 2.2 sieht man z.B., dass das Wort *Burbank* mit den Wörtern

flying und *airplane* im Kontext zusammen aufgetreten ist. Der Vektor bildet somit die Eigenschaften (*features*), die das Wort beschreiben, im Vektorraum ab. Diese Eigenschaften können per Hand gesetzt oder auch von einem Algorithmus erlernt werden. Ein Vorteil dieser Darstellung ist, dass Wörter, die im gleichen Kontext auftreten, auch semantisch ähnlich sind. So sieht man in der Abbildung 2.2, dass die Vektoren von *Burbank* und *Airport* ähnlicher sind, als die Vektoren *Burbank* und *John Wayne*.

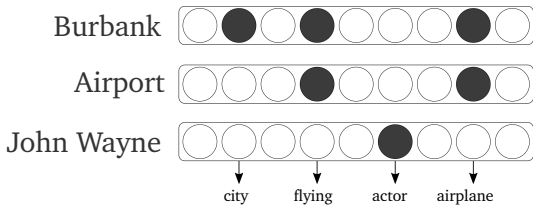


Abb. 2.2.: *Distributed represantation* der Wörter *Burbank*, *Airport* und *John Wayne* im Wortvokabular Vektor (vgl. [MC17, S. 14])

Cosine Similarity Um die Ähnlichkeit zweier Vektoren zu messen, wird die *Cosine Similarity* benutzt. Sie berechnet den Winkel zwischen beiden Vektoren. Der Wertebereich liegt zwischen $[-1, 1] \in \mathbb{R}$. Wenn die Vektoren in die gleiche Richtung zeigen, liegt der Wert bei 1. Die 0 bedeutet, dass die Vektoren orthogonal zueinander sind und -1 , dass sie voneinander weg zeigen. Die Formel für die Vektoren $\overrightarrow{v}$ und $\overrightarrow{w}$ wäre dann (vgl. [JM17, S. 279-280]):

$$cosine(\overrightarrow{v}, \overrightarrow{w}) = \frac{\overrightarrow{v} \cdot \overrightarrow{w}}{|\overrightarrow{v}||\overrightarrow{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}} \tag{2.1}$$

Neural Word Embedding Sind die Vektoren der Distributed Representations mindestens so groß wie das Wortvokabular ($k \geq |T|$), so spricht man von *Explicit Vector Representations*. Diese Repräsentationsart hat aber den Nachteil, dass die hohe Dimensionalität eine sehr geringe Datendichte vorweist, da die meisten Stellen im Vektor Null sind. Dieser Umstand macht das Modell sehr unhandlich und langsam bei großen Datenmengen.
Ziel ist es also, die Repräsentation der Wörter in einem kleineren Raum mit $k \ll |T|$ „einzubetten“ (embedding), sodass die Eigenschaften und Beziehungen zwischen den Wörtern erhalten bleiben. Beim *Neural Word Embedding* lernen Modelle diese Embeddings, indem sie versuchen, Eigenschaften eines Wortes vorherzusagen (vgl. [MC17, S. 18]).

Word2vec Eines dieser Neural Word Embedding-Modelle ist das *Word2Vec*-Modell. Für das Erlernen der Embeddings stellen Mikolov et al. [Mik+13a] zwei Methoden vor: *Skip-Gram* und *CBOW (continuous bag of words)*. Für diese Arbeit wollen wir uns mit der Skip-Gram-Methode näher befassen.

Bei der Skip-Gram-Methode wird ein Modell durch die Vorhersage benachbarter Wörter trainiert [Mik+13b]. Dabei „wandert" ein Fenster (*Sliding Window*) über einen Trainingskorpus und versucht, Wörter, die vor und nach einem vorgegebenen Wort stehen, vorherzusagen. Dieser Ansatz beruht auf der Idee, dass Wörter, die semantisch ähnlich sind, auch häufig im Text nahe beieinander auftreten. Daher sind Embeddings, die gut in der Vorhersage benachbarter Wörter sind, auch gut darin, Ähnlichkeiten zu repräsentieren. (vgl. [JM17, S. 290]).

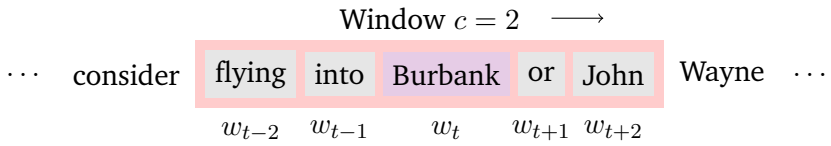


Abb. 2.3.: Sliding Window der Größe $c = 2$ mit dem gegebenen Wort w_t und den gesuchten Wörtern $\{w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}\}$ (vgl. [JM17, S. 117]).

Zu diesem Zweck lernt das Modell für alle Wörter w aus einem Trainingskorpus T ein Word Embedding. Im Lernprozess versucht das Modell, die Word Embeddings so zu verändern, dass die Ähnlichkeit zwischen dem gesuchten Wort w_t und dem Kontext Wort w_c verbessert wird $p(w_c|w_t)$.

Wie in Abbildung 2.3 gezeigt, wird dabei mit dem Sliding Window der Größe c über den Trainingskorpus T iteriert und mit Hilfe einer Log-Wahrscheinlichkeitsberechnung die Ähnlichkeit zwischen dem gegebenen Wort w_t und seinen benachbarten Wörtern w_{t+j} maximiert (vgl. [Mik+13b, S. 2f]).

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} \log\ p(w_{t+j} \mid w_t) \tag{2.2}$$

Mikolov et al. [Mik+13a] konnten belegen, dass Word Embeddings mit niedrigeren Dimensionen in vielen Bereichen besser abschneiden, als Explicit Vector Representation. Sie vermuten, dass möglicherweise die bessere Verallgemeinerung der Wörter der Grund dafür sein könnte.

Knowledge Base Die *Knowledge Base* ist die Wissensbasis, die die Entities enthält, auf die wir linken können. Sie wird auch als „Modell der Welt" bezeichnet, da im

Disambiguierungsprozess ausschließlich auf diese Entities zurückgegriffen werden kann. Die Knowledge Base kann man sich wie eine Datenbank vorstellen, die Metadaten zu jeder Entity besitzt und diese durch einen Primärschlüssel eindeutig ausweist. Diese Daten können sowohl der Name der Entity sein, als auch alternative Bezeichnungen oder Beschreibungen. Oft enthält die Knowledge Base nur Referenzen auf den tatsächlichen Artikel, da sie selbst nicht Träger dieser Information ist. Ein Beispiel für eine Wissensbasis ist die *Wikidata Knowledge Base*[1].

2.2 Named Entity Recognition

Der erste Schritt in der Informationsextraktion ist das Erkennen einer Entity im Text. Meist ist das *Wer*, *Was* oder *Wo* im Kontext gemeint, also alles was man mit einem Eigennamen bezeichnen kann (vgl. [JM17, S. 349 ff.]). *Named Entity Recognition* bedeutet, Textbereiche zu finden, die Eigennamen bilden und diese dann einem Entity-Typ zuzuordnen. Ein Projekt, das die Named Entity Recognition gut visualisiert, ist *Tasty*[2].

Tasty (tag-as-you-type) ist ein Texteditor, der selbstständig Entities im Text erkennt und diese mit zusätzlichen Informationen aus der Wikipedia anreichert (vgl. [ADL16, S. 111]). Für die Named Entity Recognition benutzt *Tasty* ein neuronales Netz, das den Input des Autors analysiert und wichtige Mentions markiert (vgl. [Arn+16, S. 112f]). Diese Mentions werden dann mit den entsprechenden Wikipedia Artikeln verknüpft. Für das Entity Linking wird ein Lucene Index mit den Referenzen aller Artikel der englischen Wikipedia benutzt. Dazu wird durch eine Volltextsuche mögliche Kandidaten für unserer Mention zurückgegeben, die mit einem Ähnlichkeits-Ranking vorsortiert sind (vgl. [Arn+16, S. 113]). Es wird die Entity gewählt, bei der es die höchsten Übereinstimmungen zwischen Mention und Kandidaten gibt. Der genaue Ablauf wird in Abschnitt 3.3.1 näher erläutert. Dieser Prozess wird *Entity Disambiguation* genannt und soll als Ansatzpunkt für unsere Disambiguation-Optimierung dienen.

2.3 Named Entity Disambiguation

Die *Named Entity Disambiguation* ist ein Teilgebiet im Entity Linking. Hierbei wird versucht, bei einer mehrdeutigen Mention mit einer Vielzahl von Entities, die semantisch richtigen zu finden. Im Abschnitt 3.1.2 werden semantische Mehrdeutigkeiten ausführlicher beschrieben.

[1] `https://www.wikidata.org/wiki/Wikidata:Main_Page`
[2] `http://demo.datexis.com/tasty/`

Ein möglicher Lösungsansatz wird in der Arbeit von Pappu et al. [Pap+17] beschrieben. Sie schlagen vor, statt nur nach der Mention in der Knowledge Base zu suchen, den Kontext, in dem die Mention eingebettet ist, zusätzlich mit in die Named Entity Disambiguation einzubeziehen. Hierzu benutzen sie den Paragraph Vector, der ähnlich wie das Word2vec-Modell mit Word Embedding arbeitet. Aber statt die Eigenschaften von Wörtern abzubilden, kann dieses Modell Texte mit beliebiger Länge repräsentieren.

2.4 Paragraph Vector

Der *Paragraph Vector* wurde für das Problem vorgestellt, dass bei der Vektorisierung von ganzen Sätzen, Paragraphen oder Dokumenten in einer Local Representation, oft die Semantik verloren geht. Außerdem werden unterschiedliche Sätze identisch dargestellt, wenn sie die gleichen Wörter benutzen, da die Reihenfolge der Wörter verloren geht (vgl. [LM14, S. 1]).
Mit dem Paragraph Vector stellen Le und Mikolov [LM14] ein Modell vor, das von dem vorher beschriebenen Word2vec inspiriert wurde. Dabei nutzen sie die Fähigkeit, durch den Vorhersageprozess auch die Semantik zu erfassen (vgl. Le und Mikolov [LM14, S. 3]). Der Kontext des Paragraphen kann dabei ein Textstück beliebiger Länge sein. Statt sich für ein Wort, nur die z.B. fünf benachbarten Wörter vorhersagen zu lassen, werden hier für einen gegebenen Paragraphen alle Wörter aus seinem Kontext prognostiziert (Abbildung 2.4). Resultierend daraus entstehen zwei verschiedene Embeddings: ein Entity Embedding und ein Word Embedding. Nach dem Training ist es nicht nur möglich, sich zu einem Paragraphen das passenden Word Embedding ausgeben zu lassen, sondern auch zu einem Kontext den ähnlichsten Paragraphen zu finden.

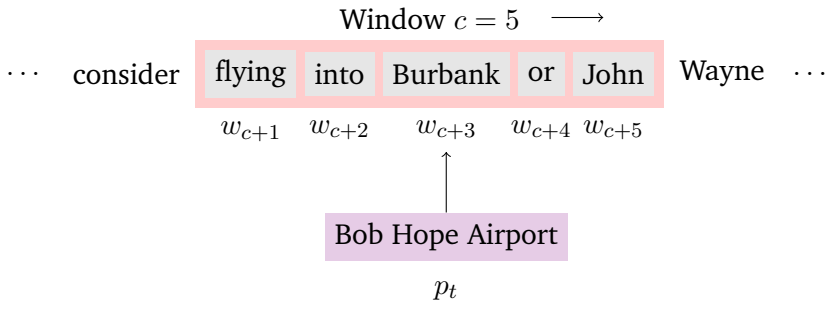


Abb. 2.4.: Paragraph Bob Hope Airport p_t mit den gesuchten Wörtern $\{w_{t+1}, \ldots, w_{t+5}\}$ aus dem Kontext in einem Sliding Window der Größe $c = 5$ (vgl. [JM17, S. 117] und [Mik+13b, S. 4]).

2.5 Zusammenfassung

Dieses Kapitel zeigt auf, dass durch Word Embeddings Eigenschaften und Beziehungen zwischen Wörtern auch auf kleinerem Raum dargestellt werden können. Des Weiteren wurde erläutert, wie Ähnlichkeiten zwischen verschiedenen Wörtern berechnet werden können. Abschließend wurden drei verschiedene Arbeiten aus dem gleichen Themengebiet vorgestellt, aus denen die Betrachtung des Kontext bei der Disambiguierung sowie der Paragraph Vector als Lösungsansatz in Frage käme. In den folgenden Kapiteln sollen diese Ansätze weitergeführt werden.

3 Named Entity Linking mit Kontextinformation

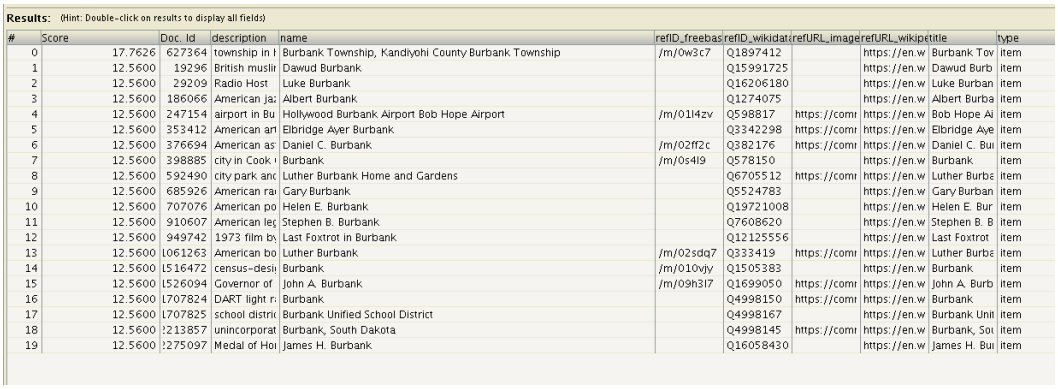
In diesem Kapitel soll ein theoretischer Lösungsansatz diskutiert werden. Dabei werden im ersten Teil die Probleme der Disambiguierung von Named Entities näher beleuchtet. Der zweite Teil beschäftigt sich mit den Entity Embeddings als Teil eines Lösungsansatzes. Zum Schluss wird die Architektur des Lösungsansatzes beschrieben.

3.1 Problem: Disambiguierung von Named Entities

Wie in Kapitel 2 beschrieben, soll eine Mention mit einer Entity aus der Knowledge Base verknüpft werden. Hierzu werden im ersten Schritt für eine Mention mögliche Kandidaten aus der Knowledge Base gesucht. Aus dieser Menge soll im zweiten Schritt der Kandidat gewählt werden, der die Mention semantisch richtig beschreibt. Dieser Prozess wird Disambiguierung genannt. Die Aufgabe der Disambiguierung ist es, zu bestimmen, welchen Sinn die Mention in einem bestimmten Kontext hat. Wenn wir z.B. mit der im ersten Kapitel beschriebenen Mention „Burbank“ in unserer Knowledge Base suchen, erhalten wir eine Vielzahl von Kandidaten. Ziel ist es, aus der Ergebnismenge die Entity „Bob Hope Airport“ auszuwählen, da sie „Burbank“ in diesem Kontext richtig beschreibt.

3.1.1 String Matching als naiver Ansatz

Der erste Ansatz für dieses Problem ist die in Kapitel 2.2 erwähnte Suche auf allen Entities. Hierzu wurde der Lucene Index d aufgebaut, der den Namen jeder Entity aus der englischen Wikipedia enthält. Diese Namen wurden mit alternativen Beschreibungen wie z.B. Linktexten, Pseudonymen sowie Synonymen angereichert. Bei einer Suche nach einer Mention m werden die Entities c zurückgegeben, deren Namen unseren gesuchten Begriff enthalten $C_m = \{c_j | \forall m \in d : c.name \approx m.name\}$ (vgl. [ADL16, S.113]).



Results: (Hint: Double-click on results to display all fields)

#	Score	Doc. Id	description	name	refID_freebas	refID_wikidat	refURL_image	refURL_wikipétitle	type
0	17.7626	627364	township in I	Burbank Township, Kandiyohi County Burbank Township	/m/0w3c7	Q1897412		https://en.w	Burbank Tov	item
1	12.5600	19296	British muslir	Dawud Burbank		Q15991725		https://en.w	Dawud Burb	item
2	12.5600	29209	Radio Host	Luke Burbank		Q16206180		https://en.w	Luke Burban	item
3	12.5600	186066	American jaz	Albert Burbank		Q1274075		https://en.w	Albert Burba	item
4	12.5600	247154	airport in Bu	Hollywood Burbank Airport Bob Hope Airport	/m/01l4zv	Q598817	https://comr	https://en.w	Bob Hope Ai	item
5	12.5600	353412	American art	Elbridge Ayer Burbank		Q3342298	https://comr	https://en.w	Elbridge Aye	item
6	12.5600	376694	American as	Daniel C. Burbank	/m/02ff2c	Q382176	https://comr	https://en.w	Daniel C. Bui	item
7	12.5600	398885	city in Cook	Burbank	/m/0s4l9	Q578150		https://en.w	Burbank	item
8	12.5600	592490	city park anc	Luther Burbank Home and Gardens		Q6705512	https://comr	https://en.w	Luther Burba	item
9	12.5600	685926	American ra	Gary Burbank		Q5524783		https://en.w	Gary Burban	item
10	12.5600	707076	American po	Helen E. Burbank		Q19721008		https://en.w	Helen E. Bur	item
11	12.5600	910607	American leg	Stephen B. Burbank		Q7608620		https://en.w	Stephen B. B	item
12	12.5600	949742	1973 film by	Last Foxtrot in Burbank		Q12125556		https://en.w	Last Foxtrot	item
13	12.5600	L061263	American bo	Luther Burbank	/m/02sdq7	Q333419	https://comr	https://en.w	Luther Burba	item
14	12.5600	L516472	census-desi	Burbank	/m/010vjy	Q1505383		https://en.w	Burbank	item
15	12.5600	L526094	Governor of	John A. Burbank	/m/09h3l7	Q1699050	https://comr	https://en.w	John A. Burb	item
16	12.5600	L707824	DART light r	Burbank		Q4998150	https://comr	https://en.w	Burbank	item
17	12.5600	L707825	school distri	Burbank Unified School District		Q4998167		https://en.w	Burbank Unif	item
18	12.5600	213857	unincorporat	Burbank, South Dakota		Q4998145	https://comr	https://en.w	Burbank, Sou	item
19	12.5600	275097	Medal of Hor	James H. Burbank		Q16058430		https://en.w	James H. Bur	item

Abb. 3.1.: Suchergebnis des *Lucene Index* für die Suche nach Burbank.

In Abbildung 3.1 wird schnell sichtbar, dass dieser Ansatz oftmals scheitern kann. Vor allem, wenn der Name der Entity mehrdeutig ist oder es verschiedene Variationen des Namens gibt (vgl. [SWH15, S. 1]). Der hier benutzte Lucene Index bewertet die Ergebnisse nach Häufigkeit des Suchbegriffs im Namensfeld und gibt eine Liste zurück, die nach diesem Ranking sortiert ist. Es lässt sich erkennen, dass der Artikel für den Bob Hope Airport an fünfter Stelle steht. Noch schwieriger wird es, wenn nach Mentions gesucht wird, die im Kontext nur beschrieben, nicht aber mit ihrem richtigen Namen genannt werden.

3.1.2 Auflösung von Semantischer Mehrdeutigkeit

Wie im vorherigen Abschnitt beschrieben, stellt die Mehrdeutigkeit und Variation der Namen ein großes Problem im Entity Linking dar. In der natürlichen Sprache gibt es viele Arten, Dinge zu beschreiben. Bezeichnen zwei gleiche Wörter unterschiedliche Dinge (Golf = Auto, Golf = Sport), so sprechen wir von *Homonymen*. Wiederum sind *Synonyme* unterschiedliche Wörter, die das Gleiche meinen (Schiff, Boot). *Hyperonym und Hyponym* bezeichnen die semantisch-begriffliche Unterordnung eines Wortes unter ein anderes Wort (Katze = Hyponym, Säugetier = Hyperonym). Häufig werden lange oder oft gebrauchte Wörter durch Abkürzungen ersetzt. Auf der anderen Seite können Begriffe auch auf ganz andere Entities zeigen, wie in unserem Beispiel „Burbank“ und „Bob Hope Airport“. Aber auch Rechtschreibfehler machen eine eindeutige Zuordnung sehr schwierig (vgl. [SWH15, S. 1], [Hac+13, S. 29], [RMD13, S. 4]). Diese Mehrdeutigkeit kann durch die Betrachtung des Kontexts aufgelöst werden.

3.1.3 Formalisierung des Problems

Durch die Mehrdeutigkeit von Wörtern werden diese oft erst im Kontext deutlich. Das Ziel ist es also, bei der Disambiguierung von mehrdeutigen Entities den Kontext mit in die Suche einzubeziehen. Dazu soll nun bei der Suche einer Mention auch der Kontext, in dem sie erwähnt wird, mit einbezogen werden. Gegeben sei eine Mention m und eine Menge von möglichen Kandidaten C_m. Gesucht ist die Entity E_m, die der Mention m am ähnlichsten ist ($E_m = \{(m, \hat{c})\}$). Der beste Kandidat $\hat{c}$ wird ermittelt, indem die Ähnlichkeit zwischen dem Kontext der Mention $vec(m)$ und dem Kontext des Kandidaten $vec(c)$ maximiert wird.

$$\hat{c} = \sum_{c \in |C_m|} cosSim(c) \tag{3.1}$$

$$\text{mit } \; cosSim(c) = cosine(vec(m), vec(c)) \tag{3.2}$$

3.2 Methodik: Entity Embeddings

Im Kapitel 2.1 wurde gezeigt, dass bei Word Embeddings Wörter mit ähnlichen Bedeutungen oder verwandten grammatikalischen Eigenschaften im Vektorraum eng beieinander liegen. Überdies wurde mit dem Paragraph Vector von Le und Mikolov [LM14] im Kapitel 2.4 eine Methode vorgestellt, die mit den gleichen Eigenschaften auch Texte beliebiger Länge im Vektorraum abbilden kann. Mit diesem Verfahren wollen wir die Entities aus der Knowledge Base in einem Entity Embedding abbilden. Ziel dieses Embeddings ist es, die Charakteristika der Entity wiederzugeben.

3.2.1 Repräsentation von Kontext im Vektorraum

Um die Merkmale einer Entity erfassen zu können, müssen wir den Kontext, der sie beschreibt, in einem Vektorraum darstellen. Hierzu benutzen wir das Paragraph Vector Modell, das in Kapitel 2.4 vorgestellt wurde. Im Trainingsprozess soll das Netz für jede Entity aus der Knowledge Base ein Embedding lernen. Hierfür muss das Netz zu einer Entity Wörter aus dem Kontext, in dem es erwähnt wird, vorhersagen. Wie in Kapitel 2.4 beschrieben, wandert ein Sliding Window über den Kontext, aus dem in jeder Iteration per Zufall ein einzelnes Wort erraten werden muss (vgl. [LM14, S. 4]). Durch diesen Vorhersageprozess entsteht ein für jede Entity eindeutiges Entity Embedding. Des Weiteren entsteht für jedes Wort aus dem erlernten Kontext ein eindeutiges Word Embedding (vgl. [LM14, S. 3]). Weiter geben Le und Mikolov

[LM14] an, dass alle Entity Embeddings mit den selben Word Embeddings arbeiten. Diese beiden Embeddings sind so optimiert, dass ein Entity Embedding immer auf alle Wörter zeigt, die im Kontext der Entity gelernt worden sind. Jedes Word Embedding zeigt wiederum auf alle Entity Embeddings, die dieses Wort im Kontext benutzt haben. Ist ein einzelnes Word Embedding sehr präzise, da es z.B. den exakten Namen der Entity codiert, so reicht das schon aus, um auf eine Entity zu zeigen (z.B Berlin). Sollte das Word Embedding schwächer sein, also nur einen allgemeineren Charakter haben (z.B Stadt), dann zeigt es eher in die Richtung von mehreren Entities. Kombiniert man allerdings mehrere Word Embeddings miteinander (z.B. Stadt, Spree, Fernsehturm), zeigt das Ergebnis desto präziser auf eine konkrete Entity (Berlin). Die Kombination dieser Word Embeddings wird auch als *Context Embedding* bezeichnet. Mikolov et al. [Mik+13a] sagen aus, dass diese Word Embeddings über die reinen syntaktischen Zusammenhänge hinausgehen, da sie scheinbar auch Bedeutungen kodieren können. Diese Aussage beweisen sie, indem sie durch einfache Vektor Algebra Analogiefragen beantworten können, wie z.B. dass das Ergebnis aus $vector(\text{"}King\text{"}) - vector(\text{"}Man\text{"}) + vector(\text{"}Woman\text{"})$ in die Nähe des Vektors „*Queens*“ zeigt (vgl. [Mik+13a, S. 2]).

3.2.2 Disambiguierung im Vektorraum

Für die Disambiguierung soll nun der Kontext mit in die Bewertung einbezogen werden. Gesucht sei die korrekte Entity für eine Mention, die durch den Kontext eines Satzes beschrieben wird. Für den „Eins-zu-eins-Match“ wird für die Mention ein Context Embedding berechnet. Dieses *Mention Embedding* v_m soll in die Richtung der Entity zeigen, in deren Kontext die Mention direkt erwähnt wurde. Zusätzlich soll aber auch der Satz, in dem die Mention erwähnt wird, durch ein Context Embedding vektorisiert werden. Dadurch lässt sich ein etwas allgemeineres Bild der Mention herausstellen: Es finden sich auch Entities, die im gleichen Kontext stehen wie jener, der im Ausgangssatz beschrieben wurde. Dieses Embedding soll hier *Sentence Embedding* v_s genannt werden.
Durch das Mention Embedding ergibt sich jeweils eine sehr individuelle, und mit dem Sentence Embedding eine sehr allgemeine Ähnlichkeit. Die Vermutung liegt nahe, dass durch eine Konkatenation der beiden Embeddings mit $v_{m \oplus s}$, die Vorteile beider Embeddings genutzt werden können. Dadurch werden die Eigenschaften der Embeddings in einem neuen Vektorraum abgebildet. Somit ginge bei sehr allgemeinen Sätzen nicht der Fokus auf die Mention verloren. Stattdessen würde das Embedding bei sehr ähnlichen oder gleichnamigen Entities in die Richtung des Kontextes des gesuchten Satzes gezogen werden.
Mit dem gleichen Fokus wollen wir auch die Addition der beiden Embeddings v_{m+s} betrachten. Wie bei der Konkatenation soll der Einfluss beider Embeddings beurteilt werden, während sie im gleichen Vektorraum bleiben.

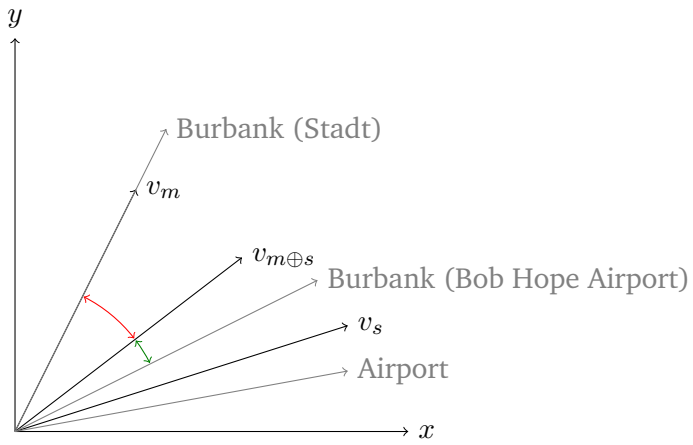


Abb. 3.2.: Stark vereinfachte Visualisierung der Vektorverteilung und Ähnlichkeitsanalyse (vgl. [MC17, S. 15])

Abb. 3.2 zeigt das Mention Embedding v_m = „Burbank“ (Stadt) und das Sentence Embedding v_s = „. . . consider flying into Burbank or John Wayne Airport . . . “. Gesucht ist die Entity Burbank (Bob Hope Airport). Obwohl das Embedding v_m auf die Stadt Burbank zeigt, verschiebt sich durch die Konkatenierung der Embeddings $v_{m\oplus s}$ in Richtung Burbank (Bob Hope Airport). Der Winkel zwischen $v_{m\oplus s}$ und Burbank (Bob Hope Airport) ist viel kleiner als der Winkel zwischen $v_{m\oplus s}$ und Burbank (Stadt). Durch die Konkatenierung der Embeddings v_m und v_s liegt das Embedding also näher an Burbank (Bob Hope Airport) als an Burbank (Stadt).

Aus dieser Erörterung können wir einige Thesen ableiten, die als Disambiguierungs-Strategien dargelegt werden sollen. Hierbei soll aus der Ergebnismenge C_m die Entity $\hat{c}$ ermittelt werden. Das wird erreicht, indem der Kandidat $c \in C_m$ ausgewählt wird, bei dem die Ähnlichkeit zu der Mention m am höchsten ist. Um die Ähnlichkeit zwischen den Kandidaten aus der Ergebnismenge und der Mention zu vergleichen, wenden wir die im Kapitel 2.1 erwähnte Cosine Similarity an. Diese berechnet den Winkel zwischen den Embeddings der Kandidaten und den der gesuchten Mention. Die Ähnlichkeit wird dann mit einem Wert zwischen -1 und $+1$ ausgedrückt, wobei die hohe Ähnlichkeit gegen $+1$ geht. Der Kandidat mit dem höchsten Wert wird als richtige Entity übernommen. Das heißt, dass alle Strategien die $cosSim()$ maximieren sollen.

SortCandidatesByLucene: Dieser Algorithmus übernimmt die Sortierung, die der Lucene Index vorgibt. Somit wird der Kandidat mit dem höchsten Score als Entity $\hat{c}$ ausgewählt. Diese Sortierung dient als Referenzwert, um zu sehen, wie sich die anderen Sortierungsalgorithmen vom Lucene Ranking unterscheiden.

$$\hat{c} = \sum_{c \in |C_m|} score(c, m) \tag{3.3}$$

SortCandidatesByParVecMention: Hier wird aus dem Ergebnis des Lucene Index der Kandidat gewählt, dessen Embedding dem gesuchten Mention Embedding am ähnlichsten ist.

$$\hat{c} = \sum_{c \in |C_m|} cosSim(c) \tag{3.4}$$

$$\text{mit } \; cosSim(c) = cosine(vec(m), vec(c)) \tag{3.5}$$

SortCandidatesByParVecSentence: Diese Strategie hat das gleiche Ziel, wie der SortCandidatesByParVecMention Algorithmus. In diesem Fall wir statt der Mention das Sentence Embedding $vec(s)$ mit dem Kandidaten verglichen.

$$\hat{c} = \sum_{c \in |C_m|} cosSim(c) \tag{3.6}$$

$$\text{mit } \; cosSim(c) = cosine(vec(s), vec(c)) \tag{3.7}$$

SortCandidatesByParVecMentionConcatSentence: Wie bereits beschrieben, wird hier die Konkatenation des Mention und Sentence Embeddings berechnet. Da der Vektorraum erweitert wird, müssen auch die Kandidaten konkateniert werden, um sie vergleichen zu können.

$$\hat{c} = \sum_{c \in |C_m|} cosSim(c) \tag{3.8}$$

$$\text{mit } \; cosSim(c) = cosine(concat(m, s), concat(c, c)) \tag{3.9}$$

$$\text{und } \; concat(a, b) = vec(a) \oplus vec(b) \tag{3.10}$$

SortCandidatesByParVecMentionAddSentence: Eine Sortierung nach dieser Strategie soll ebenfalls die Summe der Eigenschaften der Mention und des Sentence Embeddings betrachten. Mit der Addition bleiben wir aber im gleichen Vektorraum wie die Kandidaten.

$$\hat{c} = \sum_{c \in |C_m|} cosSim(c) \tag{3.11}$$

$$\text{mit } \; cosSim(c) = cosine((vec(c) + vec(s)), vec(c)) \tag{3.12}$$

SortCandidatesByParVecMentionAddSentenceWeighted: Im Laufe der Evaluierung ist aufgefallen, dass das Mention Embedding sehr aussagekräftig ist. Daher soll hier auch eine Addition der Mention und der Sentence Embeddings betrachtet werden. Dabei soll jedoch die Mention stärker gewichtet werden, um ihr mehr Einfluss in der Berechnung zuzuweisen. Dem Sentence Embedding wurde weniger Einfluss gegeben. Die Gewichtung wurde für die Mention mit 0.7 und für den Sentence mit

0.3 festgelegt:

$$\hat{c} = \sum_{c \in |C_m|} cosSim(c) \tag{3.13}$$

$$\text{mit } \; cosSim(c) = cosine((0.7vec(c) + 0.3vec(s)), vec(c)) \tag{3.14}$$

3.2.3 Trainingsdaten

Damit das Modell Mehrdeutigkeiten besser auflösen kann als der Lucene Index, muss es die Charakteristika und alternative Beschreibungen einer Entity kennenlernen. Da die Entities der Knowledge Base auf Artikel der Wikipedia verlinken, ist der Korpus der Wikipedia für diesem Fall prädestiniert. In den Artikeln der Wikipedia werden oft andere Entities erwähnt. Durch Links werden diese Entities mit den korrespondierenden Artikeln verknüpft. Der Link im Text zeigt somit auf die Entity, die im Satz beschrieben wird. Oft ist die Entity, die im Link genannt wird, auch der exakte Name der Entity. Aber die Entity wird auch mit alternative Bezeichnungen verlinkt. Im folgenden werden drei Sätze aus dem Wikipedia Korpus gezeigt, die auf die Entity *Bob Hope Airport* verlinken. Die eckigen Klammern zeigen den Begriff, mit der die Entity verlinkt wurde.

„*Also, the Authority is proposing that land owned by the [Bob Hope Airport] in Burbank be used for the HSR station.*“[1]

Der Satzt zeigt, wie die Entity *Bob Hope Airport* im Kontext als „Bob Hope Airport“ beschrieben wird.

„*Bong then became a test pilot assigned to Lockheed's Burbank, California, plant, where he flew P-80 Shooting Star jet fighters at the [Lockheed Air Terminal].*“[2]

In diesem Beispiel kann man gut erkennen, das die Entity selbst gar nicht mehr genannt wird, sondern nur noch mit „Lockheed Air Terminal“ beschrieben wird.

„*For example, they served Los Angeles via [Burbank], San Francisco via Oakland, Seattle and Vancouver via Bellingham, WA and Boston via Portsmouth, NH.*“[3]

[1] Satz aus der Wikipedia `https://en.wikipedia.org/wiki/History_of_California_High-Speed_Rail` abgerufen am 30.10.2017

[2] Satz aus der Wikipedia `https://en.wikipedia.org/wiki/Richard_Bong` abgerufen am 30.10.2017

[3] Satz aus der Wikipedia `https://en.wikipedia.org/wiki/John_Glenn_Columbus_International_Airport` abgerufen am 30.10.2017

Hier wird die Entity mit dem übergeordneten Begriff „Burbank“ in Verbindung gesetzt.

Aus all diesen Sätzen lässt sich ableiten, dass der Bob Hope Airport unter verschiedenen Namen vorkommt. Des Weiteren ist auch die Information der Sätze selbst sehr aussagekräftig. So z.B. ist der Flughafen örtlich auf Burbank, San Francisco, California begrenzt. Außerdem geht aus den Sätzen hervor, dass die Entity mit dem Verb „flew“ in Verbindung steht. Und das wichtigste, dass es durch das Wort „Airport“ als ein Flughafen beschrieben wird. Wir können also erkennen, dass durch die vielseitigen Namen und den informationsreichen Sätzen die Eigenschaften der Entities sehr gut dargestellt werden.
Wir wollen diese Eigenschaften nutzen und damit unser Modell trainieren. Dafür werden aus dem Wikipedia Korpus alle Sätze, die auf Entities verlinken, aus den Artikeln extrahiert. Vorherige oder nachfolgende Sätze werden in dieser Arbeit ignoriert, da davon ausgegangen wird, dass nur der Satz, in der die Entity erwähnt wird, sie auch am besten beschreibt.

Für die Extraktion der Sätze wurde ein Dump der englischen Wikipedia[4] heruntergeladen. Danach wurde jeder Satz extrahiert, der auf eine Entity in Wikipedia verlinkt. Zuvor wurden alle Links aus dem Satz entfernt und mit dem Linktext ersetzt. Das Netz soll zu einer Entity den Kontext lernen und sich nicht an den Links orientieren. Damit der Kontext später wieder einer Entity zugeordnet werden kann, wurde die verlinkte Entity auf ihre ID aus der Knowledge Base aufgelöst. Mit diesem Verfahren wurden Trainingsdaten in einem Format erstellt, wie in Listing 3.1 gezeigt.

```
1 ...
2 Q598817 Slick Airways had its original fleet of Curtiss C
     -46 Commando aircraft based at Lockheed Air Terminal (
     Burbank) and San Francisco Airport .[5]
3
4 Q598817 The aircraft were operated out of Burbank Airport
      in southern California ( BUR , now known as Bob Hope
     Airport ) .[6]
5 ...
```

Listing 3.1: Auszug aus dem Trainingsdatensatz mit Trainingssätzen und ihrer Entity ID

[4] `https://dumps.wikimedia.org/enwiki/20170820/enwiki-20170820-pages-articles-multistream.xml.bz2`, heruntergeladen am 11.08.2017
[5] Satz aus der Wikipedia `https://en.wikipedia.org/wiki/Slick_Airways` abgerufen am 29.10.2017
[6] Satz aus der Wikipedia `https://en.wikipedia.org/wiki/Sierra_Pacific_Airlines` abgerufen am 29.10.2017

Nach der Extraktion der Daten erhalten wir einen Trainingsdatensatz mit 2.725.363 Entities und 63.812.227 Beispielsätzen mit einem Gesamtumfang vom 11 GB. Diese Zahlen ergeben sich, da Entities und Beispielsätze mehrfach enthalten sein können. Eine Entity kann mehrmals in verschieden Sätzen erwähnt werden, genauso wie in einem Satz mehrere Entities vorkommen können.
Um festzustellen, ob die Entities mit Links wie z.B. „see here“ verlinkt waren und dadurch die Trainingsdaten verunreinigten, wurden alle extrahierten Linktexte nach der Häufigkeit sortiert und analysiert. Das Ergebnis kann in A.2 betrachtet werden. Eine Verunreinigung konnte allerdings nicht festgestellt werden.

3.3 Architektur

3.3.1 *Knowledge Base*

Die hier benutzte Knowledge Base ist ein Lucene Index. Die Daten basieren auf der Wikidata Knowledge Base und beinhalten 3,5 Millionen Entities der englischen Wikipedia. Die Entities beinhalten unter anderem ID, Namen, Beschreibungen und die Wikipedia Referenz der Entity. Die Namen wurden mit Linktexten, Pseudonymen und Synonymen angereichert.
Wir haben im Abschnitt 3.1.1 die Probleme der Disambiguierung im Lucene Index kennengelernt. Wir gehen davon aus, dass die gesuchte Entity mit in der Ergebnismenge enthalten ist. Lucene schafft es aber nicht, aus dieser Menge den richtigen Kandidaten herauszufinden und damit die Mention mit der korrekten Entity zu verlinken.
Bei einer Suche nach einer Mention benutzt Lucene die *BM25 Similarity*, die eine bestimmte Folge von Begriffen im Index sucht (vgl. [Arn+16, S. 113]). Wie im Abschnitt 3.1.1 beschrieben, versucht Lucene somit alle Kandidaten zu finden, dessen Name mit dem Suchterm komplett oder teilweise übereinstimmt. Hierbei nutzt es auch die alternativen Beschreibungen, mit denen die Entities im Lucene Index angereichert wurden. Lucene rankt die Relevanz der Kandidaten *Short Text Similarity* (vgl. [KR15]), die die Häufigkeit des gesuchten Begriffs misst. Enthalten die alternativen Beschreibungen ebenfalls den Suchterm, wird diese Emtity von Lucene viel höher bewertet. Ein weiteres Problem sind die im Abschnitt 3.1.2 besprochenen Mehrdeutigkeiten, die Lucene ohne den Kontext nicht auflösen kann.
Hier soll unser Paragraph Vector Modell in die Disambiguierung eingreifen. Für jeden Kandidaten aus der Ergebnismenge der Lucene Suche lassen wir uns das passende Paragraph Embedding ausgeben. Ebenfalls wird für die Mention und den Satz, in dem sie erwähnt wird, ein Mention Embedding, sowie das Sentence Embedding berechnet. Um zu garantieren, dass der gesuchte Kandidat auch in der Ergebnismenge enthalten ist, lassen wir uns von Lucene pro Suche bis zu 1.000 Kandidaten

zurückgeben.
Um nun den besten Kandidaten aus der Ergebnismenge herauszufiltern, wird für jeden mit Hilfe der im Abschnitt 3.2.2 erarbeiteten Sortierungsstrategie ein neues Ranking berechnet und die Ergebnismenge neu sortiert. Der Kandidat mit dem höchsten Ranking wird als Entity für unsere gesuchte Mention gewählt.

3.3.2 Maschine Learning Modell

Der Paragraph Vector ist ein *unsupervised* Algorithmus, der keine vorgelabelten Annotation benötigt (vgl. [LM14]). Um ein neuronales Netz zu trainieren, müssen die Trainingsdaten vorher aufgearbeitet werden. Das Modell versucht, wie in Kapitel 2.4 erklärt, Wörter aus einem Kontext vorherzusagen. Um die Wörter aus dem Trainingskorpus zu extrahieren, wird ein *Tokenizer* benutzt, der die Sätze an seinen Leerzeichen trennt und die Wörter normalisiert. Dazu werden alle Satzzeichen entfernt und die Wörter in Kleinbuchstaben umgewandelt (vgl. [JM17, S. 23]).
In machen Fällen kann es sinnvoll sein, auch Sonderzeichen wie Akzente und Umlaute oder Zahlen, zu entfernen. Daher soll auch ein Modell mit diesen Einstellungen trainiert werden.
Die Parameter, die für das Training des Modells benutzt werden, müssen individuell für das Problem angepasst werden. Für den Evaluierungprozess sollen mit Hilfe der folgenden Parameter verschiedene Modelle trainiert werden.

Die *Window Size* kann beeinflussen, wie viele der Wörter im Kontext auf einmal prognostiziert werden müssen. Ein kleines Window könnte eine sehr limitierte Sicht auf die Sätze verursachen.
Die *Layer Size* ist die Größe unseres Word Embeddings aus Kapitel 2.1 und gibt die Anzahl der Merkmale im Word Embedding an. Durch eine Erhöhung der Dimensionen kann die Präzision des Modells gesteigert werden.
Mit der *Iteration* wird angegeben, wie oft das Netz das Embedding für eine Satz im Korpus aktualisieren kann. Zu wenige Iterationen bedeuten, dass es möglicherweise keine Zeit hat, alles zu lernen. Zu viele verlangsamen das Training des Netzes.
Epochen ist die Anzahl, wie oft ein Training wiederholt wird. Es unterscheidet sich von der Iteration dahingehend, als dass pro Epoche der Trainingsdatensatz einmal komplett durchlaufen wurde (vgl. [PG17])

3.4 Zusammenfassung

In Kapitel 3 wurde zunächst der String Match als Ansatz verworfen, da die semantischen Mehrdeutigkeiten nicht genügend aufgelöst werden konnten. Im zweiten Abschnitt konnten dank des Entity Embedding neue Strategien für die Disambiguie-

rung entwickelt werden. Außerdem konnten verschiedene Trainingsarten für das Modell erarbeitet werden, womit durch unterschiedliche Parameter die Eigenschaften der Trainingsdaten anders priorisiert werden konnten.

4 Implementierung

In Kapitel 3 wurde methodisch ein Lösungsansatz ausgearbeitet. Dieses Kapitel behandelt die technische Implementierung in mehreren Schritten. Als erstes wird gezeigt, wie die Daten aus dem Wikipedia Korpus extrahiert werden. Im zweiten Schritt wird auf die Implementierung des Modells eingegangen. Zum Schluss wird der technische Aufbau der Evaluation behandelt.

4.1 Erstellung von Trainingsdaten aus dem Wikipedia Dump

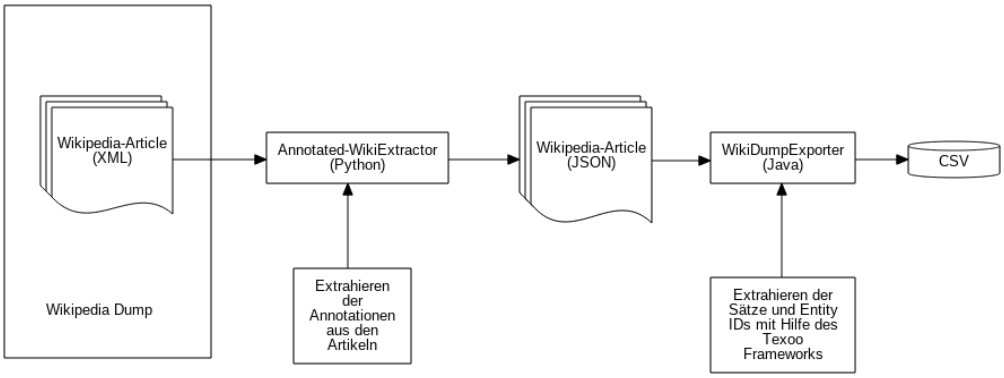


Abb. 4.1.: ETL Diagramm zur Erstellung der Trainingsdaten

Wikipedia Dump Die Wikipedia stellt einen Dump[1] mit all ihren Artikeln in verschiedenen Sprachen im Internet frei zur Verfügung. Dieser Datensatz liegt im XML Format vor und enthält zu jedem Artikel den Text sowie die eindeutige ID, die Wikipedia URL und den Title. Außerdem enthält der Artikeltext Querverweise auf andere, im Text erwähnte, Artikel in Form eines Links.

```
1 <doc id="" url="" title="">
2     Article Text...
```

[1] `https://dumps.wikimedia.org/enwiki/20170820/enwiki-20170820-pages-articles-multistream.xml.bz2`, heruntergeladen am 11.08.2017

```
</doc>.
```

Listing 4.1: Beispiel eines Wikipedia Artikels in XML Format

Da wir uns für die Evaluation nur auf die englisch Sprache konzentrieren wollen, wurde nur der englische Datensatz mit einer Gesamtgröße von 14 GB im gepackten Zustand, heruntergeladen.

Annotated-WikiExtractor Die Links im Artikeltext sind die Entities, die das Modell lernen soll und die Sätze, in dem sie eingebettet sind, sollen als beschreibender Kontext dienen. Zu diesem Zweck müssen als erstes die Links aus dem Artikel extrahiert werden. Das Python Script Annotated-WikiExtractor[2] kann die Wikipedia Dump XML Datei einlesen und sie in ein JSON Format umwandeln. Abbildung 4.2 zeigt das JSON Format.

```
1  ...
2  {"url": "http://en.wikipedia.org/wiki/Anarchism",
3      "text": "Anarchism.\nAnarchism is ...",
4      "id": 12,
5      "annotations": [
6      {"to": 46, "from": 26, "id": "Political_philosophy",
           "label": "political philosophy"},
7      {"to": 72, "from": 67, "id": "State_(polity)", "label
           ": "state"},
8      ...
9  ]}
10 ...
```

Listing 4.2: Beispiel eines Wikipedia Artikels in JSON Format

Bei der Umwandlung werden die Verlinkungen aus dem Text entfernt und mit dem Linktext ersetzt. Die Attribute der Links werden als *annotation* im JSON gespeichert. Wie in Abbildung 4.2 in der Zeile 5 zu erkennen, enthält eine *annotation* alle Attribute des Links sowie dessen Position. Das Script speichert die JSON-Objekte als String in eine Datei ab.

Texoo-Framework Um später die passenden Sätze an der Position unserer Entities zu erhalten, wird das auf Java basierende Texoo-Framework[3] benutzt. Dieses Framework beinhaltet ein Dokumentenmodell auf welches unser Wikipedia-Artikel abgebildet werden soll. Das Modell gibt uns mit der Klasse *Document* die Möglichkeit,

[2] https://github.com/jodaiber/Annotated-WikiExtractor
[3] Texoo Github Repository, https://github.com/sebastianarnold/TeXoo-develop/tree/rdziuba2017 (private)

auf bestimmte Sätze im Text zuzugreifen.
Des Weiteren kann *Document* auch verschiedene Typen von Annotationen aufnehmen. Der Type, den wir benutzen wollen, ist die *NamedEntityAnnotation*. Dieses Format soll die Annotationen des Python Script übernehmen sowie zusätzlich alle möglichen Kandidaten, die in einer Lucene Index Abfrage zu dieser Entity gefunden worden sind. Die Kandidaten werden als *ArticleRef* im Objekt angelegt. In der Abbildung 4.2 ist das Dokumentenmodell dargestellt.

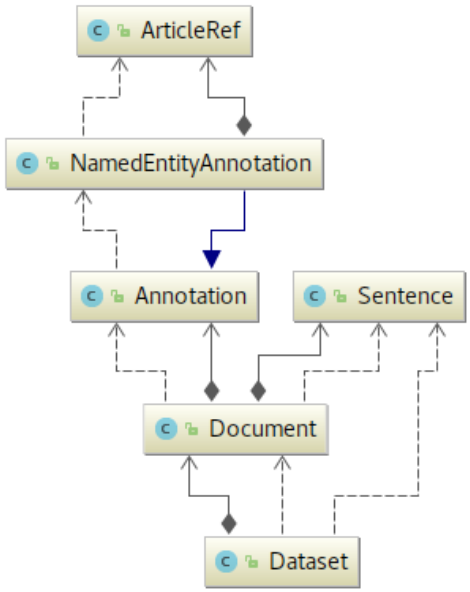


Abb. 4.2.: Klassendiagramm des Texoo Dokumentenmodells

WikiDumpExporter Das Texoo-Framework wurde mit dem *Maven Build-Management-Tool*[4] eingebunden. Alle relevanten Dependencies werden in Kapitel A.1 aufgeführt. Für die Umwandlung der JSON-Objekte in die Modell-Trainingsdaten benutzen wir den *WikiDumpExporter*. Dieser wandelt im ersten Schritt mit der Jackson[5] API die JSON-Objekte in ein Texoo *Document* um. Im zweiten Schritt iteriert der Exporter über alle *NamedEntityAnnotation* die im *Document* Objekt enthalten sind und verschafft sich über die Position der Annotation den Satz im Text.
Die Entity, stellvertretend durch ihre ID, und der sie beschreibende Satz werden in einer Datei abgespeichert. Listing 4.3 veranschaulicht dieses Format. Entities und Beispielsätze können mehrfach enthalten sein.
Da wir mit einer sehr großen Datenmenge arbeiten, wurde entschieden, die Trainingsdaten in einem CSV Format abzuspeichern, da dieses sehr sparsam ist. Nach der

[4] `https://maven.apache.org/`
[5] `http://wiki.fasterxml.com/JacksonHome`

Extraktion der Daten erhalten wir eine Datei mit 2.725.363 Entities und 63.812.227 Beispielsätzen mit einem Gesamtumfang vom 11 GB.

```
1 ...
2 Q598817 Slick Airways had its original fleet of Curtiss C
     -46 Commando aircraft based at Lockheed Air Terminal (
     Burbank) and San Francisco Airport .[6]
3 ...
```

Listing 4.3: Auszug eines Trainingssatzes mit einer Entity ID.

4.2 Implementierung des Modells

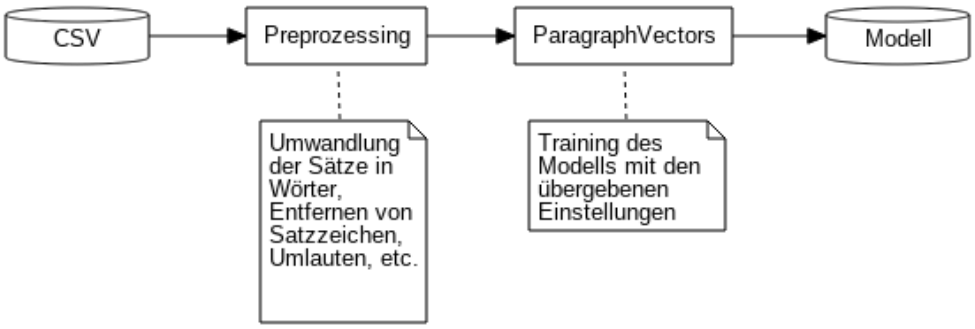


Abb. 4.3.: Übersicht der Trainingsphase des Paragraph Vector Modells

ParagraphVectors Für das Paragraph Vector-Modell benutzen wir das Java Framework *deeplearning4j*[7]. Der *ParagraphVectors.Builder* erstellt und trainiert das Modell. Dafür werden alle Daten aus unserer CSV Datei eingelesen. Da der Datensatz 11 GB groß ist, kommt es zu Problemen, diese Datenmenge im Arbeitsspeicher zu halten. Dafür wurde ein *LabelAwareStreamSentenceIterator* konzipiert, der über die Daten Zeile für Zeile in einem Stream iteriert. Dadurch konnte die Arbeitsspeicherbenutzung reduziert werden.

Preprocessing Im Kapitel 3.3.2 wurde beschrieben, dass für die Extraktion der Wörter aus dem Satz ein *Tokinizer* verwendet wird. Weiter haben wir geklärt, dass es notwendig ist, vor dem Training die Wörter zu transformieren und störende Elemente zu entfernen. Diese Aufgabe übernimmt das *preprocessing*. Der *CommonPreprocessor*

[6] Satz aus der Wikipedia `https://en.wikipedia.org/wiki/Slick_Airways` abgerufen am 29.10.2017
[7] `https://deeplearning4j.org`

tokenisiert alle Wörter aus dem Satz, entfernt alle Satzzeichen und wandelt alle Wörter in Kleinbuchstaben um. Zur zusätzlichen Entfernung aller Akzente, Sonderzeichen und Zahlen wird der *MinimalLowercasePreprocessor* benutzt. Für das jeweilig Training kann einer dieser *Preprocesser* dem *ParagraphVectors.Builder* übergeben werden.

Parameter Des Weiteren wurden in Kapitel 3.3.2 verschieden Parameter vorgestellt, die das Training des Modells beeinflussen. Hierfür sollen folgende Parameter verändert werden, dessen Ergebnis dann in der Evaluation getestet wird. Um zu sehen, wie die Dimensionsgröße das Modell verändert, wurde die *layerSize* einmal auf 100 und einmal auf 300 gesetzt. Zusätzlich wurde die Häufigkeit, wie oft das Modell die Daten durchlaufen kann, einmal mit einer Epochen und einmal mit zehn Epochen angegeben. Abschließend wurde mit *iterations* festgesetzt, wie oft ein Satz gelernt werden muss.

```
1  // pre-built WordVectors model for ParagraphVectors
2  ParagraphVectors model = new ParagraphVectors.Builder()
3      // number of iterations done for each mini-batch during
           training
4      .iterations(1)
5      // number of epochs (iterations over whole training corpus)
           for training
6      .epochs(1)
7      // number of dimensions for output vectors
8      .layerSize(100)
9      // initial learning rate for model training
10     .learningRate(0.025)
11     // defines from which learning rate learning is no longer
           effective
12     .minLearningRate(0.001)
13     // amount of words you process at a time
14     .batchSize(50)
15     // attaches pre-defined labels source to ParagraphVectors
16     .labelsSource(source)
17     // defines context window size
18     .windowSize(10)
19     // defines the iterator (LabelAwareStreamSentenceIterator)
20     .iterate(iter)
21     // tokinizer to be used for strings tokenization during
           training
22     .tokenizerFactory(t) // (CommonPreprocessor or
           MinimalLowercasePreprocessor)
23     .build();
24
25  // starts training
```

```
26 model.fit();
```

Listing 4.4: Beispielcode für das Training eines *Paragraph Vector*-Modells.

Zusätzliche Parameter wurden aus dem Code Beispiel des Github Repository[8] von *deeplearningl4j* übernommen. In Listing 4.4 ist eine Beispielimplementierung des *ParagraphVectors.Builder* zu sehen.

Nach dem Training werden die Modelle als zip-Datei abgespeichert, wie in der Tabelle 4.1 gezeigt.

Modellname	Layer	Iteration	Epochen	Tokenizer	Größe	Trainingszeit
100	100	1	1	CommonPrepr.	8.7 GB	0:56 St.
100mlp	100	1	1	MinimalLowercasePrepr.	8.8 GB	1:00 St.
300	300	1	1	CommonPrepr.	27.2 GB	1:33 St.
100_10i	100	10	1	CommonPrepr.	8.6 GB	8:50 St.
100mlp_10i	100	10	1	MinimalLowercasePrepr.	8.7 GB	10:21 St.
300_10i	300	10	1	CommonPrepr.	27 GB	15:45 St.
100_10e	100	1	10	CommonPrepr.	8.7 GB	21:50 St.
100mlp_10e	100	1	10	MinimalLowercasePrepr.	8.7 GB	10:00 St.
300_10e	300	1	10	CommonPrepr.	27.0 GB	14:36 St.

Tab. 4.1.: Übersicht aller trainierten Modelle

4.3 Implementierung der Evaluation

WNED Testdatensatz Da die Erfolge der Modelle evaluiert werden müssen, brauchen wir vorannotierte Testdaten. Unter dem WNED Datensatz wurden Testdatensätze aus fünf verschieden Themengebieten zusammengefasst. Diese Korpura enthalten Texte, in denen schon Wörter markiert und mit den richtigen Wikipedia Artikeln (Annotationen) verlinkt sind. Die Datensätze werden in Kapitel 5.1.1 näher erläutert. In der Evaluierungphase werden diese Datensätze eingelesen und wie die Artikel des Wikipedia Dump im Kapitel 4.1 in eine Texoo *Document* umgewandelt. Anschließend werden die oben erwähnten *NamedEntityAnnotion* mit den Kandidaten aus der Lucene Index Suche angereichert.

Sortierungsstrategien Im Kapitel 3.2.2 wurden mehrere Arten von *embeddings* vorgestellt, die zur Findung des besten Kandidaten beitragen sollen. Diese verschiedenen Disambiguierungsprozesse werden als Strategie unter dem Interface *EvaluationSortStrategy* zusammengefasst und bei jeder Disambiguierung einzeln bewertet.

[8] `https://github.com/deeplearning4j/dl4j-examples/blob/master/dl4j-examples/src/main/java/org/deeplearning4j/examples/nlp/paragraphvectors/ParagraphVectorsClassifierExample.java`

4.4 Zusammenfassung

Das Kapitel 4 gibt uns einen Überblick über die technische Implementierung. Als Datengrundlage wurde der englische Wikipedia Datensatz genommen, da auch die Daten aus dem Lucene Index auf den Wikipedia Indizes basieren. Um die Daten in ein JSON Format umzuwandeln, wurde eine Phyton Script genutzt. Dieses extrahierte außerdem die Annotationen aus dem Text und versah sie mit den ehemaligen Positionen im Dokument. Für die Generierung der Trainingsätze wurden mit Hilfe eines Texoo-Frameworks alle Sätze, in denen eine Entity erwähnt wurde, und die dazugehörige Entity ID, in eine CSV Datei geschrieben. Anschließend wurden die Modelle, wie in Kapitel 3.3.2 beschrieben, trainiert. Zum Schluss wurde auf die Implementierung der Evaluation eingegangen.

5 Evaluierung

Dieses Kapitel beschäftigt sich mit der Evaluierung der gelernten Modelle und den erarbeiteten Sortierungsstrategien. Im Abschnitt 5.1 wird die Testumgebung erläutert. Hier wird genauer auf die Testdaten, Sortierungsstrategien und Qualitätsmaße eingegangen. Im nächsten Abschnitt werden dann die Ergebnisse der Evaluierung demonstriert und im letzten Kapitel diese dann diskutiert und bewertet.

5.1 Aufbau der Testumgebung

Um die Qualität der Modelle messen zu können, müssen repräsentative Messtechniken und Daten herangezogen werden, anhand derer die Korrektheit überprüft werden kann. Diese sollen nun in den folgenden Abschnitten erklärt werden.

5.1.1 Testdaten

Der WNED[1] Datensatz wurde von Ganea und Hofmann [GH17] bereitgestellt. Dieser beinhaltet Datensätze mit 792 Artikeln und 19747 Annotationen. Eine genaue Auflistung der Artikel und deren Annotationen pro Datensatz ist in der Tabelle 5.1 aufgeführt. Dabei sind alle Annotation mit den Identifikatoren der englischen Wikipedia verlinkt. Bei den Datensätze handelt es sich um Dokumente aus verschieden Quellen.

- Der MSNBC Korpus beinhaltet jeweils 2 Zeitungsartikeln aus 10 verschiedenen Themengebieten [Cuc07, S. 715].
- Der AQUAINT Datensatz, mit 50 Dokumenten aus Xinhua News Service, the New York Times, und the Associated Press [MW08].
- Der ACE2004 ist ein Ausschnitt aus dem ACE co-reference Datensatz mit englischen Nachrichten aus dem Rundfunk, Zeitungen und Nachrichtendienste [Dod+04].

[1] `https://dataverse.library.ualberta.ca/dataset.xhtml?persistentId=doi:10.7939/DVN/10968`

- Der ClueWeb12 Datensatz[2] wurde von Google automatisiert annotiert und enthält Texte mit Annotationen aus mehreren Webseiten [@GRS13].

- Ausschnitt aus der Wikipedia, mit 345 verschieden Artikeln [GB14]

Datensatz	Artikel	Annotationen
MSNBC	20	739
AQUAINT	50	727
ACE2004	57	306
ClueWeb12	320	11154
Wikipedia	345	6821

Tab. 5.1.: Anzahl der Artikel und Annotationen aus den Evaluationsdatensätzen

Diese Datensätze dienen in der folgenden Evaluation als Goldstandard. Das bedeutet, dass vorgelabelte Annotationen als Mention dienen und vom System die richtigen Entites vorhersagen (predict) werden müssen. Anschließend wird das Ergebnis mit dem Goldstandard verglichen.

5.1.2 Strategien der Entity Disambiguation

Wir haben im Kapitel 3.2.2 verschiedene Sortierungsstrategien erarbeitet, die in der Evaluation getestet werden sollen. Mit den gelabelten Mentions aus dem Goldstandard werden mit einer Volltextsuche in Lucene mögliche Kandidaten gesucht. Danach wird mit der jeweiligen Sortierungsstrategie der beste Kandidat aus der Ergebnismenge disambiguiert und mit dem Goldstandard verglichen.

Die Strategien werden wie folgt abgekürzt:
SortCandidatesByLucene: $Lucene$
SortCandidatesByParVecMention: $Mention$
SortCandidatesByParVecSentence: $Sentence$
SortCandidatesByParVecMentionConcatSentence: $M \oplus S$
SortCandidatesByParVecMentionAddSentenceWeighted: $M_w + S_w$

5.1.3 Qualitätsmaße

Die Tabelle 5.2 stellt eine Wahrheitsmatrix dar. Diese vergleicht jedes Ergebnis der vorhergesagten Entity mit dem Goldstandard. Anhand der Werte der Wahrheitsmatrix können nun sowohl die Genauigkeit (precision), die Trefferquote (recall) als auch das F-Maß ermittelt werden (vgl. [JM17, S.84]).

[2] `http://lemurproject.org/clueweb12/`

		Goldstandard		
		Gold positiv	Gold negativ	
Predicted Entity	Pred positiv	**true positiv**	**false positiv**	**precision**
	Pred negativ	**false negativ**	**true negativ**	
		recall		

Tab. 5.2.: Wahrheitsmatrix zur Bestimmung der Korrektheit der vorhergesagten Ergebnisse (vgl. [JM17, S. 84])

Precision misst den prozentualen Anteil der Annotationen, die das Modell korrekt detektiert hat (vgl. [JM17, S.84]).

$$precision = \frac{\text{true positives}}{\text{true positives} + \text{false positives}} \tag{5.1}$$

Recall misst den Prozentsatz der tatsächlich gefundenen Annotationen(vgl. [JM17, S.84]).

$$recall = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}} \tag{5.2}$$

Bei einem kleinen Dokument mit wenigen Annotationen würden Precision und Recall viel mehr ins Gewicht fallen, als bei einem großen Dokument, mit mehr Annotationen. Daher wurde hier mit der micro precision bzw. dem micro recall gearbeitet, die den Durchschnitt des gesamten Datensatzes berechnet und nicht pro Dokument.

Das F-Maß ist ein gewichtetes harmonisches Mittel aus der Precision und dem Recall (vgl. [JM17, S. 84-85]). Es ist eine konservative Metrik, da es beim Mittelwert zweier Zahlen immer das Minimum wählt.

$$F_1 = \frac{2PR}{P + R} \tag{5.3}$$

Mit dem Mean Reciprocal Rank (MRR) kann gemessen werden, an welchem Rang die korrekte Entity im Suchergebnisses steht. Dazu wird jeder Kandidat mit dem Kehrwert des Rangs des Suchergebnisses gewichtet. Steht die richtige Entity zum Beispiel an der dritten Stelle, so ist der reziproke Rang $\frac{1}{3}$. Die reziproken Ränge

werden addiert und durch die Gesamtsumme geteilt, um einen Mittelwert zu erhalten (vgl. [JM17, S.415-416]). Die Formel für eine Anfrage mit N Ergebnissen lautet:

$$MRR = \frac{1}{|N|} \sum_{i=1}^{|N|} \frac{1}{rank_i} \tag{5.4}$$

Im Goldstandard sind auch Mentions enthalten, die nicht mit der Wikipedia verlinkt werden können- sogenannte „NILs“ (non-linkable entities) (vgl. [LSW15, S. 3]). Da die Behandlung von NILs ein eigenes Fachgebiet im Named Entity Linking darstellt, wurden diese in dieser Arbeit nicht näher betrachtet und verworfen.

5.2 Ergebnisse

In diesem Abschnitt werden die Ergebnisse der Testdaten thematisiert. Hierfür wurde eine Durchschnitt aller im Test befindlichen Testdatensätze aus den verschieden Domänen berechnet. Für jede einzelne Strategie werden die Werte des MRR, der Precision, des Recalls und der F_1 betrachtet.

5.2.1 Ergebnisse der Strategien und Modelle

In den folgenden Tabellen werden die Durchschnittswerte der Evaluation aller Datensätze näher beleuchtet. Bei jeder Testphase werden die Werte des Lucene Index angegeben, um einen Referenzwert zu haben. Lucene erreicht eine Genauigkeit von 48,24% und eine Trefferquote von 45,10%. Nur jede zweite Entity kann also identifiziert werden und nur die Hälfte ist richtig annotiert. Im Suchergebnis steht der richtige Kandidat im Durchschnitt an zweiter Stelle.

Modelle mit 1 Iteration und 1 Epoche
Die Tabelle 5.3 zeigt, dass die Modelle in einigen Strategien bereits besser sind, als der Lucene Index. Die Modelle wurden mit einer Window Size der Länge 10 trainiert. Über die Datensätze wurde nur einmal iteriert und jeder Satz wurde nur ein mal von den Modellen gesehen. Bereits hier kann man sehen, dass das Modell mit 300 Layern die Entity Embeddings am besten auflösen konnte. Aber auch das Modell 100mlp überzeugt mit seinen Werten.

Modelle mit 10 Iterationen und 1 Epoche
Das Modell mit 10 Iterationen kann nicht überzeugen. In Tabelle 5.4 lässt sich erkennen, dass sich die Werte verschlechtert haben, obwohl das Netz jeden Satz 10 Mal hintereinander gesehen hat. Die Modelle mit 300 Layern und jene mit dem Mini-

Strategie + Modell	MRR ∅	Prec	Rec	F1
Lucene	0.534	48.24	45.10	46.60
$Mention$_100	0.501	44.67	41.41	42.97
$Sentence$_100	0.407	35.59	33.42	34.47
$M \oplus S$_100	0.540	50.28	46.67	48.40
$M_w + S_w$_100	0.352	34.00	31.66	32.78
$Mention$_100mlp	0.569	54.54	50.89	52.63
$Sentence$_100mlp	0.428	38.49	36.17	37.28
$M \oplus S$_100mlp	0.577	55.06	51.49	53.20
$M_w + S_w$_100mlp	0.407	39.27	36.78	37.97
$Mention$_300	0.590	57.49	53.65	55.49
$Sentence$_300	0.458	41.58	39.05	40.27
$M \oplus S$_300	**0.613**	**60.71**	**56.74**	**58.64**
$M_w + S_w$_300	0.415	40.45	37.89	39.12

Tab. 5.3.: Modell mit 100/ 300 Layern + CommonPreprocessor/ MinimalLowercasePreprocessor + 1 Iteration + 1 Epoche

malLowercasePreprocessor schneiden sogar schlechter ab, als die Vorgängermodelle mit nur einer Iteration.

Strategie + Modell	MRR ∅	Prec	Rec	F1
Lucene	0.534	48.24	45.10	46.60
$Mention$_100_10i	0.537	51.50	47.82	49.58
$Sentence$_100_10i	0.417	37.85	35.57	36.67
$M \oplus S$_100_10i	0.539	51.34	47.71	49.45
$M_w + S_w$_100_10i	0.363	36.13	33.67	34.85
$Mention$_100mlp_10i	0.555	52.35	48.83	50.52
$Sentence$_100mlp_10i	0.423	37.98	35.73	36.82
$M \oplus S$_100mlp_10i	0.545	51.06	47.80	49.36
$M_w + S_w$_100mlp_10i	0.399	38.18	35.76	36.92
$Mention$_300_10i	0.567	54.33	50.76	52.47
$Sentence$_300_10i	0.444	40.87	38.44	39.61
$M \oplus S$_300_10i	**0.577**	**55.27**	**51.73**	**53.43**
$M_w + S_w$_300_10i	0.399	38.25	35.88	37.02

Tab. 5.4.: Modell mit 100/ 300 Layern + CommonPreprocessor/ MinimalLowercasePreprocessor + 10 Iterationen + 1 Epoche

Modelle mit 1 Iteration und 10 Epochen

Die Modelle in der Tabelle 5.5 sind die erfolgreichsten. Der MRR liegt bei der Strategie der Vektorkonkatenation um 10% höher, als der Lucene Durchschnitt. Beim F_1 Ergebnis ist das Modell sogar um 13,3% besser als Lucene.

Strategie + Modell	**MRR ∅**	**Prec**	**Rec**	**F1**
Lucene	0.534	48.24	45.10	46.60
$Mention$_100_10e	0.576	54.61	50.99	52.73
$Sentence$_100_10e	0.448	40.38	37.97	39.13
$M \oplus S$_100_10e	0.592	56.73	53.10	54.84
$M_w + S_w$_100_10e	0.410	39.86	37.35	38.55
$Mention$_100mlp_10e	0.580	55.84	52.15	53.91
$Sentence$_100mlp_10e	0.455	40.97	38.53	39.70
$M \oplus S$_100mlp_10e	0.594	57.20	53.52	55.28
$M_w + S_w$_100mlp_10e	0.412	40.31	37.78	38.99
$Mention$_300_10e	0.594	57.00	53.19	55.01
$Sentence$_300_10e	0.517	48.74	45.73	47.18
$M \oplus S$_300_10e	**0.630**	**62.01**	**57.96**	**59.90**
$M_w + S_w$_300_10e	0.418	40.76	38.17	39.41

Tab. 5.5.: Modell mit 100/ 300 Layern + CommonPreprocessor/ MinimalLowercasePreprocessor + 1 Iteration + 10 Epochen

5.2.2 Ergebnisse des Goldstandards

Dieser Abschnitt befasst sich mit den erfolgreichsten Modellen und den dazugehörigen Datensätzen in der Tabelle 5.6. Auffallend ist hier, dass die Modelle vor allem im ClueWeb Korpus besonders schlecht sind, bei dem die Annotationen automatisiert erstellt wurden. Gute Ergebnisse erhalten alle Modelle beim ACE2004 und Wikipedia Korpus. Dass unsere Modelle so gut bei Wikipedia abgeschnitten haben, liegt daran, dass die Modelle mit Beispielen aus diesem Korpus trainiert wurden.

		ACE2004		AQUAINT		CLUEWEB		MSNBC		Wikipedia	
Modell	Strategie	MRR ∅	F_1	MRR ∅	F_1	MRR ∅	F_1	MRR ∅	F_1	MRR ∅	F_1
-	Lucene	0.559	55.44	0.507	43.67	0.480	38.12	0.573	48.06	0.552	47.72
100_10e	$Mention$	0.648	60.98	0.591	56.48	0.492	42.96	0.577	52.10	0.571	51.11
	$Sentence$	0.357	29.42	0.408	34.34	0.401	33.45	0.457	42.10	0.617	56.33
	$M \oplus S$	0.613	57.57	0.574	53.16	0.508	44.47	0.600	56.77	0.667	62.24
	$M_w + S_w$	0.385	35.82	0.414	39.16	0.345	30.92	0.409	41.61	0.497	45.25
100mlp_10e	$Mention$	0.646	61.83	0.594	57.08	0.497	43.31	0.591	55.97	0.573	51.38
	$Sentence$	0.383	29.85	0.408	34.94	0.403	33.70	0.465	43.87	0.617	56.13
	$M \oplus S$	0.621	58.42	0.577	54.82	0.511	44.68	0.593	56.29	0.669	62.21
	$M_w + S_w$	0.382	35.39	0.418	39.46	0.348	31.22	0.415	43.55	0.497	45.35
300_10e	$Mention$	**0.672**	**65.67**	0.611	58.89	0.510	44.65	0.591	53.55	0.585	52.31
	$Sentence$	0.461	42.22	0.483	43.98	0.450	38.76	0.531	49.52	0.657	61.41
	$M \oplus S$	0.666	65.25	**0.632**	**61.90**	**0.537**	**47.72**	**0.634**	**60.81**	**0.680**	**63.83**
	$M_w + S_w$	0.399	37.53	0.425	40.21	0.350	31.46	0.414	42.10	0.502	45.77

Tab. 5.6.: Übersicht der Erfolgreichsten Netze auf den einzelnen Korpora.

5.3 Diskussion und Bewertung

An dieser Stelle soll ein Resumé aus den erworbenen Daten gezogen werden. Die Thesen aus Kapitel 3 können nun bestätigt oder widerlegt werden. Zusätzlich wird der zu Beginn der Arbeit vorgestellte Satz mit Hilfe eines Modells wieder in den Fokus rücken.

Neuronale Word Embeddings sind nicht automatisch besser als Lucene Exact Matches
Die vorangegangenen Tests haben gezeigt, dass abhängig von Strategie und Modell die Disambiguierung durch neuronale Word Embeddings ein besseres Ergebnis erzielt können, als der Exact Match von Lucene.

Mention Embeddings sind besser als Lucene Exact Match
Die These aus Kapitel 3.2.2 besagt, dass das Embedding der Mention verstärkt auf eine gleichnamige Entity zeigt. Damit ist dieses Embedding präziser als der Lucene Index Rank. In Kapitel 3.1.1 haben wir gesehen, das Lucene sich auf die Häufigkeit der Namen konzentriert. Das bedeutet, dass wenn eine Entity besonders häufig den gesuchten Text der Mention im Namensfeld hat, dieser von Lucene priorisiert wird. Im Gegensatz zeigt das Mention Embedding auf die Entity, welche unsere Mention am meisten im Kontext benutzt hat. Das Beispiel zeigt einen Satz aus dem Trainingsdatensatz ACE2004 mit der gesuchten Mention „Riyadh“:

„*London 10-16 (AFP) - A Saudi Airways spokesman in London announced that passengers on the Saudi 777 Boeing, hijacked to Baghdad on Saturday and which arrived in Riyadh yesterday evening Sunday, left the Saudi capital for London at 2h43 local time (GMT 22h43), approximately 30 hours after the hijacking operation by two Saudi hijackers began.*“[3]

Hier wurde vom Lucene Index die Entity (Q1249255) Riyadh Region als die Richtige ausgewählt, da im Namensfeld die Begriffe *Riyadh Province, Riyadh Region, Riyadh* auftauchen. Zum Vergleich enthält die Entity (Q3692) Riyadh nur den Text *Riyadh*. Das Mention Embedding zeigte auf die richtige Entity (Q3692) Riyadh. Es ließ sich nicht von den häufiger erwähnten Begriffen im Namensfeld „ablenken“.

Sentence Embeddings sind schlechter als Mention Embeddings
Wie bereits in Kapitel 3.2.2 vermutet, ist das Sentence Embedding sehr allgemein. Der Vektor zeigt in die Richtung der Entities, die einen ähnlichen Kontext vorzeigen, wie der der Mention. Ein Grund könnte sein, dass bei der Lernphase mehrere Entities

[3] aus dem WNED-Dataset, `/wned-datasets/ace2004/RawText/20001015_AFP_ARB.0229.eng`

mit einem Satz trainiert worden sind, da alle in diesem vorkamen. Des Weiteren können ähnliche Entities auch in vergleichbaren Kontexten vorkommen. Somit zeigt z.B. der Satz „Die Birne ist eine Frucht.“ auf die Entities Orange und Banane, wenn das Netz mit Sätzen wie „Die Orange ist eine Frucht.“ und „Die Banane ist eine Frucht.“ trainiert wurde.

Konkatenierte Entity Embeddings sind besser als Lucene
Die beste Strategie ist die Konkatenation der Mention und Sentence Embeddings. Wie schon in Kapitel 3.2.2 angedeutet, werden hier die Eigenschaften der beiden Vektoren am besten ausgenutzt. Die Kombination beider Vektoren scheint wirklich die Schwächen des einzelnen Vektors auszubalancieren. Hier wird die Abstraktion des Sentence Embeddings zum Vorteil, da es das Mention Embedding auf der Kontextebene beeinflusst und die Semantik mit beachtet wird.

Addition und Konkatenation der Vektoren verhalten sich identisch
In der Evaluation ist aufgefallen, dass das Ergebnis der Cosine Similarity bei einer reinen Addition der Mention und mit dem Sentence Embedding identisch mit den konkatenierten Vektoren war. Nach anschließender Prüfung konnte festgestellt werden, dass sich die Vektoren $vec(m) + vec(s)$ im Vektorraum genauso verhalten, wie $vec(m) \oplus vec(s)$, wenn auch die Vektoren der Kandidaten $vec(c) \oplus vec(c)$ konkateniert wurden.

$$cosine((vec(c) + vec(s)), vec(c)) \equiv cosine(concat(m, s), concat(c, c)) \tag{5.5}$$

Gewichtete Embeddings können die Addition nicht verbessern
Es konnte leider nicht bestätigt werden, dass eine gewichtete Addition die gewünschten Verbesserung bringt. Das Verstärken des Mention Embeddings und das Abschwächen des Sentence Embeddings brachte eher das Gegenteil und ließ das resultierende Embedding ungenauer werden.

Konzentration auf die wichtigen Wörter
Das Modell, das den MinimalLowercasePreprocessor (Kapitel 4.2) benutzt, kann bei einer etwas geringeren Größe im Durchschnitt bessere Ergebnisse liefern. Durch das Entfernen der Nummern, Akzente und Umlaute, musste sich das Modell wahrscheinlich noch mehr auf den Kontext konzentrieren und konnte sich nicht an Akzenten oder Zahlen orientieren.

Es ist wichtig, was gelernt wird
Wie bereits aufgezeigt, hat der Datensatz der Wikipedia im Durchschnitt am besten abgeschnitten, da unser Modell ebenfalls auf Wikipedia trainiert wurde. Es ist also wichtig, auf welcher Datenbasis das Modell trainiert wurde. Andersherum könnte

man auch sagen, dass ein Modell, das mit einem fachspezifischen Korpus trainiert wurde, sein Wissen nicht so gut auf andere Korpora übertragen kann.

Mehr Layers führen zu besseren Ergebnissen
Wie bereits vermutet, können in mehr Layern mehr Information, kodiert werden. Es stellt sich heraus, dass das Modell mit 300 Layern in jedem Test am besten abschnitt.

Trainingsaufwand steigt mit Modellgröße
Mehr Information bedeutet auch mehr Zeit für das Lernen sowie mehr Platz. Die Modelle mit 300 Layern liegen bei 27 GB, die im Arbeitsspeicher gehalten werden müssen, was ein lokales Arbeiten schwieriger macht. Für noch größere Modell muss auf einem Cluster gearbeitet werden.

Trainingszeit steigt überproportional mit Iteration/ Epochen
Im Modell-Training ist aufgefallen, dass die Zeit, die das Modell braucht, um über mehrere Iteration/ Epochen zu lernen, überproportional hoch ist. So lernte das 100 Layer Model bei einer Iteration in 0:56 Stunden den gesamten Wikipedia-Datensatz. Für zehn Iteration brauchte es bereits 8:50 Stunden. Für die zehn Epochen waren es erstaunliche 21:50 Stunden. Die Modelle wurden auf dem Beuth Hochschul-Cluster trainiert, dadurch kann nicht ausgeschlossen werden, dass der Cluster zusätzlich mit anderen Aufgaben belegt war und es deshalb zu längeren Trainigsphasen kam.

Disambiguierungsstrategien anhand eines Beispielsatzes
Letztlich soll der am Anfang in Kapitel 1.1 genannte Satz noch einmal in den Fokus rücken und deutlich machen, wie die einzelnen Strategien die Mentions aufgelöst haben.

„Instead of Los Angeles International, for example , consider flying into Burbank or John Wayne Airport in Orange County, Calif., or use Westchester County Airport instead of JFK in New York.“[4]

In der Tabelle 5.7 sind alle Annotationen aus diesem Satz mit den Goldstandard Entities aufgeführt. Daneben stehen die Auswertungen der verschiedenen Algorithmen. Es ist sehr schön zu sehen, dass eindeutige Mention, wie *Los Angeles International Airport*, von allen Strategien eindeutig aufgelöst werden konnten.
Im Fall *Burbank* kann man gut erkennen, wie das Sentence Embedding den Kontext des Satzes richtig interpretiert und das Mention Embedding bei der Konkatenation in die richtige Richtung „zieht“.
Bei der Mention *Orange County* sieht man aber auch die Grenzen des Modells. Das

[4]Im Goldstandart wurde „New York“ nicht annotiert und daher hier auch nicht mit aufgeführt.

Mention Embedding stuft *Orange County Board of County Commissioners* als richtige Entity ein. Das Sentence Embedding kann aus dem Kontext, der sehr „airportlastig“ ist, nicht auf die allgemeinere Entity *Orange County* schließen.
Erschwerend kommt hinzu, dass Lucene nur exakte Übereinstimmungen finden kann. Da *Calif* nicht als Abkürzung für *California* interpretiert wurde, war diese Entity nicht im Suchergebnis enthalten. Die Modelle konnten diesen Mangel nicht mehr ausgleichen.
Wie bei *Burbank* konnte auch für *JFK*, durch den Kontext, die richtige Entity disambiguiert werden.

Mention	Gold	Lucene	Mention	Sentence	$\mathbf{M} \oplus \mathbf{S}$	$\mathbf{M_w} + \mathbf{S_w}$
Los Angeles International	**(Q8731) Los Angeles International Airport**	**(Q8731) Los Angeles International Airport**	**(Q8731) Los Angeles International Airport**	**(Q8731) Los Angeles International Airport**	**(Q8731) Los Angeles International Airport**	**(Q8731) Los Angeles International Airport**
Burbank	**(Q598817) Bob Hope Airport**	(Q4998146) Burbank	(Q7720617) The Burbank Studios	**(Q598817) Bob Hope Airport**	**(Q598817) Bob Hope Airport**	(Q7720617) The Burbank Studios
John Wayne Airport	**(Q1692316) John Wayne Airport**	**(Q1692316) John Wayne Airport**	**(Q1692316) John Wayne Airport**	**(Q1692316) John Wayne Airport**	**(Q1692316) John Wayne Airport**	**(Q1692316) John Wayne Airport**
Orange County	**(Q5925) Orange County**	**(Q109215) Orange County**	(Q7099547) Orange County Board of County Commissioners	(Q1692316) John Wayne Airport	(Q1692316) John Wayne Airport	(Q7099547) Orange County Board of County Commissioners
Calif	**(Q99) California**	(Q5036322) Calif	(Q5036322) Calif	(Q5036322) Calif	(Q5036322) Calif	(Q5036322) Calif
Westchester County Airport	**(Q2876027) Westchester County Airport**	**(Q2876027) Westchester County Airport**	**(Q2876027) Westchester County Airport**	**(Q2876027) Westchester County Airport**	**(Q2876027) Westchester County Airport**	**(Q2876027) Westchester County Airport**
JFK	**(Q8685) John F. Kennedy International Airport**	**(Q8685) John F. Kennedy International Airport**	(Q741823) JFK	**(Q8685) John F. Kennedy International Airport**	**(Q8685) John F. Kennedy International Airport**	(Q741823) JFK

Tab. 5.7.: Ergebnisse der Disambiguierung aller Annotationen im Beispielsatz mit allen Strategien mit dem Modell: 100 Layer + CommonPreprocessor + 10 Epochen

Lucene limitiert das Suchergebnis

Wenn ein gesuchter Begriff nicht im Lucene Index enthalten ist oder überhaupt keine Übereinstimmung mit dem Namen der Entity hat, ist die richtige Entity nicht in der Ergebnismenge enthalten und kann somit auch nicht disambiguiert werden. In der Evaluation wurden 19.165 Annotation überprüft. 718 Annotationen waren NILs und bei 2.867 Annotation war die richtige Entity nicht in den ersten 1.000 Ergebnissen enthalten. Die Suchmenge wurde daraufhin erhöht aber die Evaluierung aus Performance-Gründen abgebrochen, da zu viele Kandidaten im Speicher gehalten werden mussten.

Schwächen der Word Embeddings
In dieser Arbeit wurde keine explizite Fehleranalyse der Word Embeddings erstellt. Trotzdem sollen hier an einigen Beispielen Schwächen der Word Embeddings gezeigt werden.

„*Order premium Minnesota Wild Hockey Tickets online for the 2012 - 2013 NHL season at Xcel Energy Center .*“[5]

In diesem Beispiel war die Abkürzung *NHL* gesucht und die (Q1215892) *National Hockey League* gemeint. Das Modell schlug die Entity (Q29144) *National Hockey League All-Star Game* vor. In vielen Fällen hatte das Modell Probleme, die Abkürzungen richtig aufzulösen.

„*In 1799 Elijah Paine built the famous Paine Turnpike which ran from Brookfield , through northern Northfield , to Montpelier , opening up trade and travel on this Boston-Montreal route .*“[6]

Boston war in diesem Beispiel gesucht und die Entity (Q100) *Boston* gemeint. Es wurde aber auf (Q16952917) *Boston and Providence Railroad* aufgelöst. Wahrscheinlich hat der Kontext zu viel Einfluss gehabt.

„*Ford does n't even have to sell them to the public .*“[7]

In manchen Fällen hätte wahrscheinlich selbst ein Mensch Schwierigkeiten, auf die richtige Entity zu schließen, da der Satz aus dem Kontext gerissen war. Gesucht war hier *Ford* mit der Firma (Q44294) *Ford*, aber es wurde der Unternehmer (Q516858) *Henry Ford II* disambiguiert.

5.4 Zusammenfassung

In diesem Kapitel wurden die Daten, auf denen die Evaluierung basiert, vorgestellt. Um den Erfolg unserer Modelle zu messen, wurden die Messtechniken precision, recall, F_1 und MRR näher beleuchtet. Im Abschnitt 5.2 wurden die Ergebnisse der Strategien und Modelle analysiert und ausgewertet. Danach wurden die Ergebnisse diskutiert und Thesen bestätigt oder widerlegt.

[5] aus dem WNED-Dataset, `/wned-datasets/clueweb12/RawText/clueweb12-0503wb-08-14873`
[6] aus dem WNED-Dataset, `/wned-datasets/clueweb12/RawText/clueweb12-0503wb-96-11521`
[7] aus dem WNED-Dataset, `/wned-datasets/clueweb12/RawText/clueweb12-0502wb-61-32124`

6 Zusammenfassung und Ausblick

In diesem Kapitel werden alle Schritte der Arbeit rekapituliert und die einzelnen Erkenntnisse zusammengefasst. Des Weiteren werden theoretische Schritte besprochen, wie eine Weiterverarbeitung der Erkenntnisse aussehen könnte.

6.1 Zusammenfassung

Zu Beginn dieser Arbeit stand die Frage, ob es möglich ist, mit Hilfe des Kontexts für eine Mention die semantisch richtige Entity zu finden. Letztlich sollte im Disambiguierungsprozess nicht nur die Mention, sondern auch der Kontext, in dem sie eingebettet ist, mit in die Betrachtung einfließen.

Dazu wurden verschiedene Arbeiten aus der aktuellen Forschung präsentiert und die Vorteile der neuronalen Word Embeddings vorgestellt. Weiter wurde dargelegt, wie Entity Embeddings in der Frage der Kontextbetrachung weiterhelfen könnten.

Um die Disambiguierung zu verbessern, wurden mehrere Strategien erarbeitet und verschiedene Lernmethodiken analysiert. Außerdem wurde mit der Wikipedia ein guter Datensatz gefunden, um die Modelle zu trainieren. Aus diesem Korpus wurden 2.725.363 Entities und 63.812.227 Beispielsätze extrahiert, die als Grundlage für den Lernprozess dienten.

In der Evaluationsphase konnte gezeigt werden, dass eine Verbesserung des Entity Linkings durch neuronale Word Embeddings möglich ist und sehr gute Ergebnisse erzielen kann. Zudem haben wir gesehen, dass die Modelle in der Lage waren, die Mention mit Hilfe des Satzes semantisch richtig einzuordnen.

Insgesamt konnte eine Verbesserung von über 13% im F_1 Score im Vergleich zum Exact Match von Lucene erreichen werden. Der MRR hat gezeigt, dass die richtigen Entities immer zwischen dem ersten und dem zweiten Platz lagen, was eine Verbesserung von 10% gegenüber des Lucene Index bedeutet.
Leider ist die Größe der Modelle für ein lokales Arbeiten viel zu groß und auch kleinere Server dürften bei 27GB Arbeitsspeicher, nur für das Modell des Entity Linkings, Probleme bekommen.

6.2 Ausblick

Minimierung der Modellgröße
Eines der wichtigsten Schritte ist es, die Modellgröße zu verkleinern, damit ein effektives Arbeiten möglich wird. In der Arbeit von Arnold et al. [Arn+16] wird mit Hilfe von Tri-Grammen die Modellgröße extrem verringert. In diesem Modell werden keine Wörter mehr gelernt, sondern nur noch Zeichenketten der Länge 3. Damit muss das Modell nicht mehr Millionen von Wörtern lernen, sondern nur noch Kombinationen von wenigen 10.000 Tri-Grammen.

Lucene beim Entity Linking weglassen
Die Modelle können nur damit arbeiten, was der Lucene Index findet. Im oberen Beispiel war in der Lucene-Suche zur Mention *Calif* die korrekte Entity nicht im Ergebnis dabei. Es sollte daher versucht werden, das Entity Linking ohne Lucene zu realisieren. *Deeplearning4j* bietet in seinem Framework die Möglichkeit, mit *Next Nearest Neighbor* ähnliche Embeddings im Paragraph Vector-Modell zu suchen. Allerdings ist das bei 2.725.363 Entities und einem Vektor mit 100 Stellen ein sehr zeitaufwendiges Verfahren, das mehrere Sekunden pro Mention dauert. Eventuell wäre es möglich, den Lucene-Index durch einen Vektor-Index zu ersetzen und mit einem effizienten *K-Nearest Neighbors*-Algorithmus die Suche zu verbessern.

Mehr Kontext
In der aktuellen Arbeit wurde nur der Satz, in dem die Entity beschrieben wurde, als Kontext gelernt. Eventuell könnte eine Vergrößerung des Kontext mit dem vorausstehenden oder nachfolgenden Satz Verbesserungen in der Disambiguierung schaffen, da mehr Information über Entity gewonnen wird.

Größeres Window
Mit der Größe des Windows kann auch der zu erlernende Kontext bestimmt werden. Die hier benutzte Window Size von zehn Wörtern gibt an, wie viel Kontext das Modell mit einem Mal lernen kann. Mit einer Vergrößerung können hier vielleicht mehr Informationen extrahiert werden. Andererseits zeigt das Experiment mit den Sätzen, dass zu viel Kontext die Ergebnisse auch schlechter machen kann.

Einfluss anderer Entities im Text nutzen
Wie Pappu et al. [Pap+17] in ihrer Arbeit gezeigt haben, sind die vorherigen erwähnten Entities in einem Text wichtige Indikationen für die Disambiguierung einer Entity, da sie diese positiv beeinflussen können. Eventuell wäre es möglich, diese mit in das Entity Linking einfließen zu lassen.

Anlagen

A

A.1 Maven Dependency

Die Implementierung dieses Projektes wurde mit der Hilfe des Maven Build-Management-Tool[1] der Apache Software Foundation realisiert. In der folgenden Liste werden die relevantesten Abhängigkeiten, die zur Funktion dieses Programms von Nöten sind, aufgelistet.

- Paketname: commons-cli, Version: 1.4, GroupID: commons-cli,
  Funktion: Zum Präsentieren, Verarbeiten und Validieren von Eingaben einer Befehlszeilenschnittstelle.
- Paketname: hamcrest-all, Version: 1.3, GroupID: org.hamcrest,
  Funktion: Erweiterung der JUnit Tests speziell für die Validierung von Arrays.
- Paketname: texoo-core , Version: 0.7, GroupID: de.datexis,
  Funktion: Für das Extrahieren und Evaluieren der Daten aus dem Wikipedia-korpus.
- Paketname: deeplearning4j-core , Version: 0.9.1, GroupID: org.deeplearning4j,
  Funktion: Erstellung des Paragraph Vector Modell
- Paketname: jackson.core , Version: 2.6.5, GroupID: com.fasterxml.jackson.core,
  Funktion: Parsen der JSON Daten.
- Paketname: lucene-core , Version: 6.4.2, GroupID: org.apache.lucene,
  Funktion: Schnittstelle für den Lucene Index.

A.2 Überprüfung der Daten im Wikipedia Korpus

Für das Erlernen der Beispielsätze war es nötig, dass der Wikipedia Datensatz verlässliche Daten enthält. Zur Validierung dieser Daten wurden alle Mentions aus den Wikipedia-Artikeln in einer MySQL Datenbank abgelegt und die Häufigkeit der Mentions gezählt. Ziel war es auszuschließen, dass Artikel mit Texten wie „see here“ verlinkt wurden und die Trainingsdaten „verschmutzen". In der Tabelle A.1 kann man die zwanzig häufigsten Mention mit ihrer Anzahl sehen. Es konnte bei Mention, die mehr als 500 Mal erwähnt wurden, das Problem nicht bestätigt werden.

[1] `https://maven.apache.org/`

mention	total
United States	174118
World War II	112112
France	104556
India	99781
Iran	85698
American	83881
Australia	81252
Germany	80568
moth	76587
New York City	75861
Japan	73658
Canada	72829
England	71873
London	71054
National Register of Historic Places	70252
football	67569
footballer	65463
village	64827
English	59250
Russia	59045
California	56510

Tab. A.1.: Übersicht der zwanzig am häufigsten erwähnten Mention in der englischen Wikipedia

Literatur

[ADL16] Sebastian Arnold, Robert Dziuba und Alexander Löser. „TASTY: Interactive Entity Linking As-You-Type“. In: *COLING 2016, 26th International Conference on Computational Linguistics, Proceedings of the Conference System Demonstrations, December 11-16, 2016, Osaka, Japan*. Hrsg. von Hideo Watanabe. ACL, 2016, S. 111–115.

[Arn+16] Sebastian Arnold, Felix A. Gers, Torsten Kilias und Alexander Löser. „Robust Named Entity Recognition in Idiosyncratic Domains“. In: *CoRR* abs/1608.06757 (2016). arXiv: 1608.06757.

[Cuc07] Silviu Cucerzan. „Large-Scale Named Entity Disambiguation Based on Wikipedia Data“. In: *EMNLP-CoNLL 2007, Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, June 28-30, 2007, Prague, Czech Republic*. Hrsg. von Jason Eisner. ACL, 2007, S. 708–716.

[Dod+04] George R. Doddington, Alexis Mitchell, Mark A. Przybocki et al. „The Automatic Content Extraction (ACE) Program - Tasks, Data, and Evaluation“. In: *Proceedings of the Fourth International Conference on Language Resources and Evaluation, LREC 2004, May 26-28, 2004, Lisbon, Portugal*. European Language Resources Association, 2004.

[GB14] Zhaochen Guo und Denilson Barbosa. „Robust entity linking via random walks“. In: *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. ACM. 2014, S. 499–508.

[GH17] Octavian-Eugen Ganea und Thomas Hofmann. „Deep Joint Entity Disambiguation with Local Neural Attention“. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*. Hrsg. von Martha Palmer, Rebecca Hwa und Sebastian Riedel. Association for Computational Linguistics, 2017, S. 2609–2619.

[Hac+13] Ben Hachey, Will Radford, Joel Nothman, Matthew Honnibal und James R. Curran. „Evaluating Entity Linking with Wikipedia“. In: *Artif. Intell.* 194 (2013), S. 130–150.

[JM17] Daniel Jurafsky und James H. Martin. „Speech and Language Processing“. PDF. Erweiterte und aktualisierte Version als PDF online erhältlich unter https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf; abgerufen am 01. Oktober. 2017.

[KR15] Tom Kenter und Maarten de Rijke. „Short Text Similarity with Word Embeddings“. In: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. CIKM ’15. Melbourne, Australia: ACM, 2015, S. 1411–1420.

[LM14] Quoc V. Le und Tomas Mikolov. „Distributed Representations of Sentences and Documents“. In: *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*. Bd. 32. JMLR Workshop and Conference Proceedings. JMLR.org, 2014, S. 1188–1196.

[LSW15] Xiao Ling, Sameer Singh und Daniel S. Weld. „Design Challenges for Entity Linking“. In: *TACL* 3 (2015), S. 315–328.

[MC17] Bhaskar Mitra und Nick Craswell. „Neural Models for Information Retrieval“. In: *CoRR* abs/1705.01509 (2017).

[Mik+13a] Tomas Mikolov, Kai Chen, Greg Corrado und Jeffrey Dean. „Efficient Estimation of Word Representations in Vector Space“. In: *CoRR* abs/1301.3781 (2013).

[Mik+13b] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado und Jeffrey Dean. „Distributed Representations of Words and Phrases and their Compositionality“. In: *CoRR* abs/1310.4546 (2013).

[MW08] David N. Milne und Ian H. Witten. „Learning to link with wikipedia“. In: *Proceedings of the 17th ACM Conference on Information and Knowledge Management, CIKM 2008, Napa Valley, California, USA, October 26-30, 2008*. Hrsg. von James G. Shanahan, Sihem Amer-Yahia, Ioana Manolescu et al. ACM, 2008, S. 509–518.

[Pap+17] Aasish Pappu, Roi Blanco, Yashar Mehdad, Amanda Stent und Kapil Thadani. „Lightweight Multilingual Entity Extraction and Linking“. In: *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, WSDM 2017, Cambridge, United Kingdom, February 6-10, 2017*. Hrsg. von Maarten de Rijke, Milad Shokouhi, Andrew Tomkins und Min Zhang. ACM, 2017, S. 365–374.

[PG17] Josh Patterson und Adam Gibson. *Deep Learning: A Practitioner’s Approach*. Beijing: O’Reilly, 2017.

[RMD13] Delip Rao, Paul McNamee und Mark Dredze. „Entity linking: Finding extracted entities in a knowledge base“. In: *Multi-source, multilingual information extraction and summarization*. Springer, 2013, S. 93–115.

[SWH15] Wei Shen, Jianyong Wang und Jiawei Han. „Entity Linking with a Knowledge Base: Issues, Techniques, and Solutions“. In: *IEEE Trans. Knowl. Data Eng.* 27.2 (2015), S. 443–460.

Webpages

[@GRS13] Evgeniy Gabrilovich, Michael Ringgaard und Amarnag Subramanya. *FACC1: Freebase annotation of ClueWeb corpora, Version 1 (Release date 2013-06-26, Format version 1, Correction level 0)*. 2013. URL: `http://lemurproject.org/clueweb12/` (besucht am 30. Okt. 2017).

Abbildungsverzeichnis

Tabellenverzeichnis

Erklärung

Ich versichere, dass ich diese Abschlussarbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Datum Unterschrift