Semantische Beziehungen in Texten mit Word2Vec

und der Vergleich zwischen allgemeinen und domänenspezifischen Korpora als Trainingsdaten

B A C H E L O R A R B E I T

im Studiengang
Medieninformatik (MI7)
an der Hochschule der Medien in Stuttgart
vorgelegt von Ruben Müller

im Juli 2015

Erstprüfer: Prof. Dr-Ing. Johannes Maucher,
Hochschule der Medien, Stuttgart

Zweitprüfer: M.Sc. Andreas Stiegler,
Hochschule der Medien, Stuttgart

Erklärung

Hiermit versichere ich, Ruben Müller, an Eides Statt, dass ich die vorliegende Bachelorarbeit mit dem Titel: „Semantische Beziehungen in Texten mit Word2Vec“ selbständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.
Ich habe die Bedeutung der eidesstattlichen Versicherung und die prüfungsrechtlichen Folgen (§ 23 Abs. 2 Bachelor-SPO (7 Semester) der HdM) sowie die strafrechtlichen Folgen (gem. § 156 StGB) einer unrichtigen oder unvollständigen eidesstattlichen Versicherung zur Kenntnis genommen.

Filderstadt, den XX. Juli 2015

Ruben Müller

Kurzfassung

Diese Bachelorthesis beschäftigt sich mit der Analyse von semantischen Beziehungen innerhalb mit Word2Vec gelernten Modellen.
Dazu sollen zum einen schon der vorhandene allgemeine Wikipedia-Korpus gelernt und analysiert werden, was als semantisch ähnlich erkannt wird. Zum anderen soll ein Korpus über eine spezielle Domäne erstellt und gelernt werden. Welche spezielle Domäne analysiert und verglichen werden soll, wird im Laufe der Bearbeitung festgelegt.
Diese beiden Korpora sollen sich dann gegenüber gestellt und analysiert werden, was jeweils als semantisch ähnlich erkannt wird.
Ziel dieser Arbeit soll es sein, festzustellen ob ein allgemeiner Korpus oder ein spezieller Domänenkorpus genauere Resultate im Hinblick auf semantische Ähnlichkeiten erzielt. Anstatt eines allgemeinen Korpus zu verwenden, könnte es sich dann anbieten zwischen mehreren speziellen Korpora auszuwählen, je nachdem welche Domäne aktuell bearbeitet werden soll.

Abstract

Inhaltsverzeichnis

Begriffsverzeichnis

Begriff	Erklärung
Ähnliche Worte	Im Word2Vec-Modell, mit der Methode *most_similar()*, erhaltene Worte.
SVM	Support Vector Machine
NBC	Naive Bayes Classifier
CBOW	Continuous bag-of-words

Kapitel 1

Einleitung

1.1 Motivation

1.2 Problemstellung

1.3 Aufbau der Arbeit

Kapitel 2

Daten und Vorverarbeitung

2.1 Datenbasis

Zum erfolgreichen Training des Word2Vec Modells wird eine sehr große Menge an Daten gebraucht[1].
In dieser Arbeit werden unterschiedliche Datenkorpora als Trainingsdaten für unterschiedliche Word2Vec Modelle benutzt.
Für den ersten Korpus wurde der komplette englischsprachige Wikipedia Korpus verwendet[2].
Der zweite in der Arbeit verwendete Korpus besteht auch aus Wikipedia Artikeln, allerdings wurden hier nur technologiespezifische Artikel verwendet. Der komplette englische Wikipedia Korpus wurde zuerst in die einzelnen Artikel aufgeteilt und diese wurden dann mit einem NBC in die Klassen *tech, entertainment, sport, science, politic* eingeteilt.

2.2 Externe Programme und Hilfsmittel

Dieser Abschnitt enthält eine Auflistung mit kurzen Beschreibungen, der in dieser Arbeit verwendeten Hilfsmittel und externen Programme.

gensim(ŘS10)[3] ist eine Bibliothek für Python. Sie enthält unter anderem eine performanzoptimierte Implementierung von Word2Vec.

[1] In (MSC+13) werden Trainingsdaten mit bis zu 30 Milliarden Wörtern benutzt.
[2] Dump von März 2015, http://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2, abgerufen am 09.04.2015
[3] https://radimrehurek.com/gensim/, abgerufen am 24.06.2015

Der **NBC**, der zum Klassifizieren der Artikel verwendet wurde, ist der im DataMining Praktikum selbst implementierte.

PyCharm[4] wurde als Editor und zum Ausführen der Python Skripte verwendet.

2.3 Vorverarbeitung

Der komplette Wikipediadump kann als komprimierte XML-Datei heruntergeladen werden[5]. Die entpackte XML-Datei hat eine Größe von 48,8 GB. Die Daten müssen zunächst bereinigt werden um als Trainingsdaten für Word2Vec Modelle zu dienen. Um das Wiki-Markup, wie unter anderem Links, Referenzen oder Zitate, und die XML-Tags zu entfernen, kann ein schon vorgefertigtes Perl Skript von Matt Mahoney benutzt werden[6], welches von Mikolov et al. auf der Google Code Seite von Word2Vec vorgeschlagen wird[7]. Dieses Skript wurde speziell zum Bereinigen von Wikipediadaten erstellt und wurde leicht verändert, sodass Zahlen, Satzzeichen und die Groß- und Kleinschreibung erhalten bleibt. Außerdem werden die Umlaute in *ae*, *oe* und *ue* umgewandelt. Da die bis hierher bereinigten Daten eine Gesamtgröße von 18 GB haben, ist es nötig die Daten weiter aufzubereiten, sodass sie als Input für die Klasse *gensim.models.word2vec.LineSentence*[8], aus der Gensim Bibliothek dienen. Dazu müssen die einzelnen Sätze in je einer Zeile stehen, alle Worte klein geschrieben , frei von Satzzeichen und mit Leerzeichen getrennt sein.

Um den kompletten Wikipediadump in die einzelnen Wikipedia Artikel aufzuteilen muss die XML-Datei mittels eines SAX-Parsers[9] geparst und dann in einzelne Dateien geschrieben werden. Diese einzelnen Dateien können dann mit dem oben genannten Perl Skript gereinigt und dann mittels NBC klassifiziert werden. Nach der Klassifizierung können die Dateien wieder in eine große Datei zusammengefasst und weiter verarbeitet werden. Auch diese Daten müssen, wie der Gesamtkorpus, mit der Klasse *gensim.models.word2vec.LineSentence* verarbeitet werden.

[4] https://www.jetbrains.com/pycharm/, abgerufen am 02.07.2015

[5] http://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2, abgerufen am 09.04.2015

[6] http://mattmahoney.net/dc/textdata.html unter Appendix A, abgerufen am 25.06.2015

[7] https://code.google.com/p/word2vec/, abgerufen am 25.06.2015

[8] Diese Klasse kann auch als Input zum Trainieren des Word2Vec Modells genutzt werden.

[9] Es wurde die Klasse *xml.sax.handler.ContentHandler* verwendet.

Kapitel 3

Word2Vec

Im folgenden Kapitel wird kurz erklärt was Word2Vec[1] ist und welche wichtigen Parameter und Algorithmen bei der Berechnung und dem Erstellen von Word2Vec Modellen benutzt werden können.

In Word2Vec Modellen werden Worte als Vektoren dargestellt[2]. Hier wird mittels *distributed representation*(MCCD13) ein n-dimensionaler Vektorraum erzeugt, in dem jedes Wort aus den Trainingsdaten durch einen Vektor dargestellt wird.
Als nächster Schritt werden die Vektoren in ein neuronales Netz gegeben und dort mittels eines Lernalgorithmus so verändert, dass Worte mit ähnlicher Bedeutung ähnliche Vektoren haben. So kann die Ähnlichkeit zwischen zwei Vektoren mit der Kosinus-Ähnlichkeit berechnet werden.
Die Berechnung der Wortvektoren kann mit neuronalen Netzen unterschiedlicher Architektur erreicht werden, *CBOW* oder *Skip-gram*. Desweiteren stehen unterschiedliche Lernalgorithmen für die neuronalen Netze zur Verfügung, *hierarchical softmax* und *negative sampling*.
Beim Training können unterschiedliche Parameter eingestellt werden.

3.1 Parameter

size

Mit dem Parameter *size* wird die Anzahl der Dimensionen der Wortvektoren eingestellt. In einem n-Dimensionalen Vektorraum nimmt n den Wert von *size* an.

[1] https://code.google.com/p/word2vec/
[2] Word2Vec heißt wörtlich Wort zu Vektor.

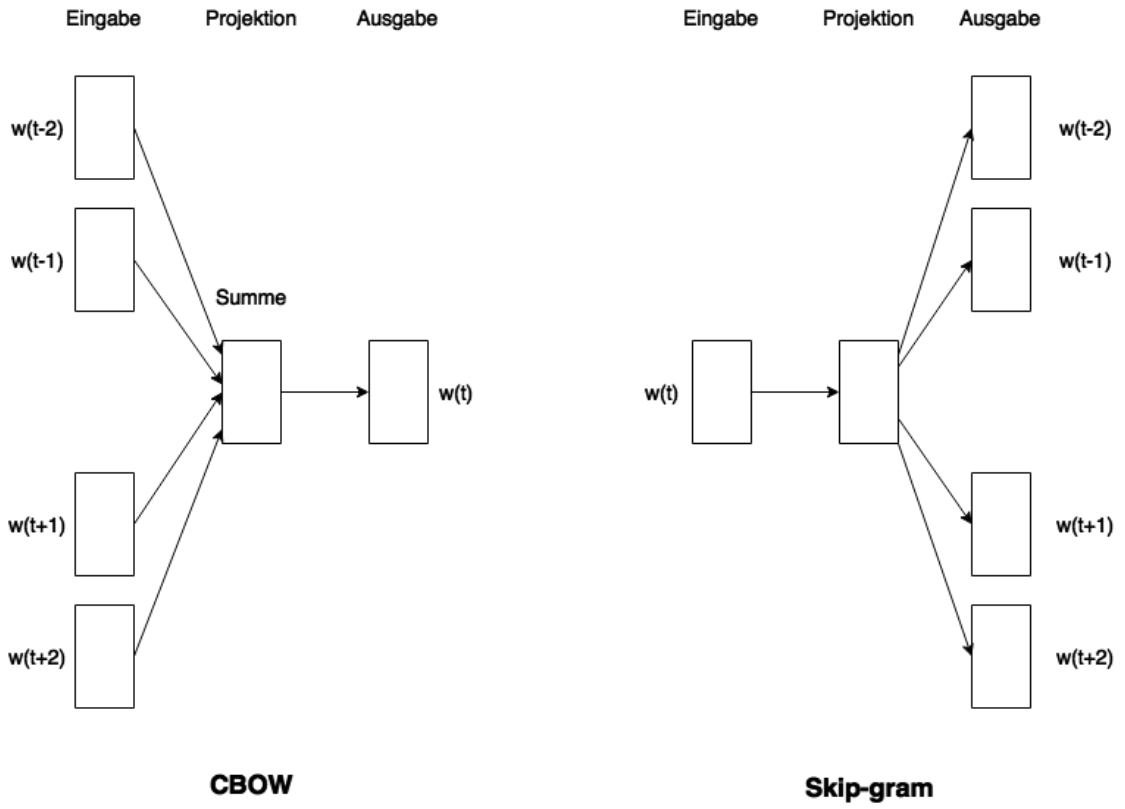

Abbildung 3.1: CBOW und Skip-gram im Vergleich, übersetzt nach (MCCD13)

window

window ist der maximale Abstand zwischen benachbarten Worten, innerhalb eines Satzes, die zur Berechnung der Wordvektoren betrachtet werden.

min_count

Der Parameter *min_count* gibt an, wie oft ein Wort in den Testdaten mindestens vorkommen muss, um in das Wörterbuch aufgenommen zu werden.

negative

Dieser Parameter wird nur benötigt, wenn als Lernalgorithmus negative sampling verwendet wird[3]. Er gibt an, wie viele zufällig ausgewählten Worte verwendet werden

[3]siehe 3.5 Negative sampling

sollen.

3.2 CBOW

CBOW ist die Abkürzung für Continuous bag-of-words (dt. stetige Menge an Worten). Beim CBOW wird ein neuronales Netz ohne verdeckte Schichten (hidden layer) verwendet(MCCD13).
In der CBOW Architektur wird aus dem Kontext ein Wort vorhergesagt (siehe Abbildung 3.1). Die Anzahl, der aus dem Kontext zu verwendende Worte, wird mit dem Parameter *window* angegeben.

3.3 Skip-gram

Bei der Skip-gram Architektur wird auch, wie beim CBOW Model, ein neuronales Netz ohne verdeckte Schichten (hidden layer) verwendet (MCCD13).
Allerdings wird hier nicht ein Wort aus dem Kontext vorhergesagt, sondern aus einem Wort wird der Kontext vorhergesagt (siehe Abbildung 3.1).

Mehrere verdeckte Schichten in neuronalen Netzen machen die Modelle genauer, allerdings kommt auch die meiste Komplexität des ganzen Models von solchen verdeckten Schichten (MCCD13). Mikolov et al. haben deshalb neuronale Netze ohne verdeckte Schichten bevorzugt, da damit sehr große Datenmengen viel effizienter gelernt werden können.

3.4 Hierarchical softmax

Eine Annäherung an das allgemeine softmax ist das hierarchical softmax (MSC$^+$13). Der Hauptvorteil ist, dass anstatt W Ausgabeknoten im neuronalen Netz nur ungefähr $log_2(W)$ Knoten ausgewertet werden müssen um die Wahrscheinlichkeiten zu errechnen.
Bei diesem Lernalgorithmus wird das Wörterbuch als Huffman Binärbaum dargestellt. Dies hat den weiteren Vorteil, dass häufig genutzte Worte kurze Kodierungen haben, was sich auf das Trainingstempo positiv auswirkt.

3.5 Negative sampling

Alternativ zum hierarchical softmax kann das negative sampling als Lernalgorithmus für das neuronale Netz verwendet werden (MSC+13).
Das negative sampling unterscheidet sich zum hierarchical softmax insofern, dass nicht die Ähnlichkeiten zu anderen Worten berechnet werden, sondern es wird davon ausgegangen, dass zufällig ausgewählte Worte mit einer hohen Wahrscheinlichkeit unähnlich, dem zu testenden Wort, sind. Wie viele solcher zufällig ausgewählter Worte benutzt werden sollen kann mit dem Parameter *negative* angegeben werden.

Kapitel 4

Wikipedia-Korpus

Im Folgenden Kapitel werden die in der Arbeit verwendeten Korpora erläutert und die im Word2Vec Modell benutzten Parameter begründet.
Des weiteren werden die benutzten Testdaten vorgestellt.

4.1 Gesamtkorpus

Da sich die Qualität der Wortvektoren im Word2Vec Modell wesentlich mit der Menge an Trainingsdaten erhöht(MCCD13), werden möglichst große Textkorpora bevorzugt. Auf der Google Code Seite von Word2Vec[1], werden für Forschungszwecke einige Beispiele für online verfügbare große Korpora genannt. Unter anderem auch der neueste Wikipedia Auszug[2].
Da in dieser Arbeit ein allgemeiner und ein domänenspezifischer Korpus als Trainingsdaten für das Word2Vec Modell verglichen werden sollen, eignet sich der Wikipedia Korpus gut.
Wie in 2.3 beschrieben, müssen die Daten erst einer Säuberung unterzogen werden um dann anschließend im Word2Vec Modell trainiert zu werden. Der bereinigte, komplette englischsprachige Wikipedia Korpus[3] enthält 8.392.453 Artikel, 242.144.317 Sätze und 2.919.802.692 Worte.

Die Skip-gram Architektur verhält sich im Bezug auf syntaktische Ähnlichkeit etwas schlechter als die CBOW Architektur, allerdings ist die Skip-gram Architektur im Bezug auf semantische Ähnlichkeit der CBOW weit überlegen(MCCD13).

[1] https://code.google.com/p/word2vec/, abgerufen am 29.06.2015

[2] http://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2, abgerufen am 29.06.2015

[3] Dump von März 2015, http://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2, abgerufen am 09.04.2015

Aus diesem Grund wurde die Skip-gram Architektur ausgewählt.
Da der *hierarchical softmax* Lernalgorithmus eher für selten vorkommende Worte und der *negative sampling* Lernalgorithmus eher für häufig vorkommende Worte und niedrigdimensionale Vektoren geeignet ist[4], fiel die Wahl auf den *hierarchical softmax* Lernalgorithmus, da das Modell (wie im Folgenden gezeigt) eine hohe Dimension der Vektoren hat und auf technologiespezifische Daten getestet werden soll (siehe 7.1 Testdaten).

Um die optimalen Parameter für das Training des Modells herauszufinden wurde ein kleinerer Wikipedia Auszug[5] benutzt und mit den unterschiedlichen Parametern[6] gelernt und evaluiert (siehe Tabelle 4.1).
Die Word2Vec-Klasse in der Gensim-Implementierung beinhaltet eine Evaluationsfunktion[7], die die Accuracy des Models berechnet. Die Funktion erwartet einen Dateinamen einer Datei, in der jede Zeile ein 4-Tupel ist und die einzelnen Abschnitte mit „: SECTION NAME“ unterteilt sind. Auf der Google Code Seite ist eine solche Datei als Beispiel vorhanden[8]. In diesem Beispiel sind 14 Kategorien[9] mit insgesamt 19544 4-Tupel aufgelistet.

Tabelle 4.1: Vergleich Parameter

Size	**Window**	**Min_count**	**Gesamtaccuracy**
400	10	5	52,9%
400	10	10	52,5%
300	10	5	**53,3%**
300	10	10	52,9%
200	10	5	50,5%
200	10	10	50,5%
100	10	5	42,0%
100	10	10	41,3%

[4]https://code.google.com/p/word2vec/#Performance, abgerufen am 01.07.2015
[5]http://mattmahoney.net/dc/enwik9.zip, abgerufen am 30.06.2015, beinhaltet die ersten 1 Milliarde Zeichen des Gesamtkorpus
[6]Siehe 3.1. Auf der Google Code Seite von Word2Vec wird eine window size bei der Skip-gram Architektur von um die 10 vorgeschlagen.
[7]*gensim.models.word2vec.Word2Vec.accuracy(FILENAME)*
[8]https://code.google.com/p/word2vec/source/browse/trunk/questions-words.txt, abgerufen am 29.06.2015
[9]*capital-common-countries, capital-world, currency, city-in-state, family, gram1-adjective-to-adverb, gram2-opposite, gram3-comparative, gram4-superlative, gram5-present-participle, gram6-nationality-adjective, gram7-past-tense, gram8-plural, gram9-plural-verbs*

Auf dem kleineren Wikipedia Korpus erzielte das Modell mit den Parametern

$$size = 300, window = 10, min\_count = 5$$

die beste Gesamtaccuracy mit 53,3%.
Diese Parameter wurden dann auch beim Gesamtmodell angewandt.
Das Gesamtmodell erreichte sogar eine Gesamtaccuracy von 59,7%. Die Trainingszeit beträgt bei diesen Parametern 7,7h[10] beim Gesamtmodell.

4.2 Teilkorpus

Der zweite Korpus ist wie in 2.3 beschrieben, ein Teilkorpus, bestehend aus Technologieartikeln des Gesamtkorpus. Dieser domänenspezifische Teilkorpus enthält 187.144 Artikel (2,2% im Vergleich zum Gesamtkorpus), 9.866.096 Wörter(0,34%) und 3.166.065 Sätze(1,3%). Hier beträgt die Trainingszeit ca. 2 Minuten.
Wird dieser Korpus mit der *.accuracy()*-Methode evaluiert, erreicht er eine Gesamtaccuracy von nur 7,0%. Das hat den Grund, dass die Testfragen sehr allgemeiner Art sind und diese Beziehungen in einem reinen Technologiekorpus sehr selten bis gar nicht vorkommen.

Es wäre auch möglich gewesen, andere Daten[11] zu benutzen. Die Wahl fiel aber auf den Teilkorpus von Wikipedia, da hier die gleichen Texte in beiden Korpora vorhanden sind und somit auch die gleichen Beziehungen zwischen den einzelnen Wörtern, um so einen möglichst objektiven Vergleich zwischen den unterschiedlichen Korpora zu ermöglichen.

4.3 Testdaten

Die Testdaten umfassen 236 Begriffe (siehe Anhang 7.1) aus der Domäne Technologie. Die Testdaten werden dazu verwendet, um die ähnlichen Worte in den Modellen zu untersuchen und zu vergleichen. Die genauen Fragestellungen und Experimente werden im Kapitel 5 Experimente ausführlich beschrieben.

[10] Leistungsdaten: PC mit 32 GB RAM, i7-3770 Quadcore bei 3,4 GHz
[11] z.B. Foren, Technews Internetseiten o.ä.

Kapitel 5

Experimente

In diesem Kapitel sollen die unterschiedlichen Korpora (Gesamtkorpus[1] und Techkorpus[2]) untersucht werden. Dies soll durch ausgewählte Fragestellungen realisiert werden.
Die Fragestellungen beziehen sich immer auf die Ergebnisse, die aus den Tastdaten[3] erhaltenen ähnlichen Worten.
Jedes Experiment ist in drei Teile aufgeteilt Beschreibung, Durchführung und Interpretation/Ergebnis.
Zum Vergleichen der ähnlichen Worte der unterschiedlichen Korpora wurde eine Datei erzeugt, in der die Testdaten und ihre fünf ähnlichsten Worte sowie deren fünf ähnlichsten Worte (Rekursion), leserlich formatiert, ausgegeben werden.

5.1 Synonymsuche durch Rekursion

5.1.1 Beschreibung

Es soll untersucht werden, ob es möglich ist, Synonyme eines Begriffs zu finden, indem man die vom Modell erhaltenen ähnlichen Worte dieses Begriffs erneut als Input(Suchbegriffe) der Methode *most_similar()* verwendet. Also durch einmalige Rekursion.

[1] vgl. 4.1
[2] vgl. 4.2
[3] vgl. 7.1

5.1.2 Durchführung

Oft war der Suchbegriff aus den Testdaten mit in den ähnlichen Wörtern der ähnlichen Wörter. Da in diesem Experiment pro Suchbegriff 25 Ergebnisbegriffe untersucht werden mussten, nahm es etwas mehr Zeit in Anspruch als die übrigen Experimente. Zudem waren viele Begriffe sehr themenspezifisch, deren Zusammenhang mit dem Suchbegriff aus den Testdaten erst einmal herausgefunden werden musste.
Im Technologiekorpus waren einige Suchbegriffe nicht enthalten.

Tabelle 5.1: Synonyme durch Rekursion

Korpus	Synonyme nicht enthalten	Synonyme enthalten	Wort nicht im Korpus	Relation
Komplett	187	49	0	20,8%
Technologie	191	20	25	8,5%

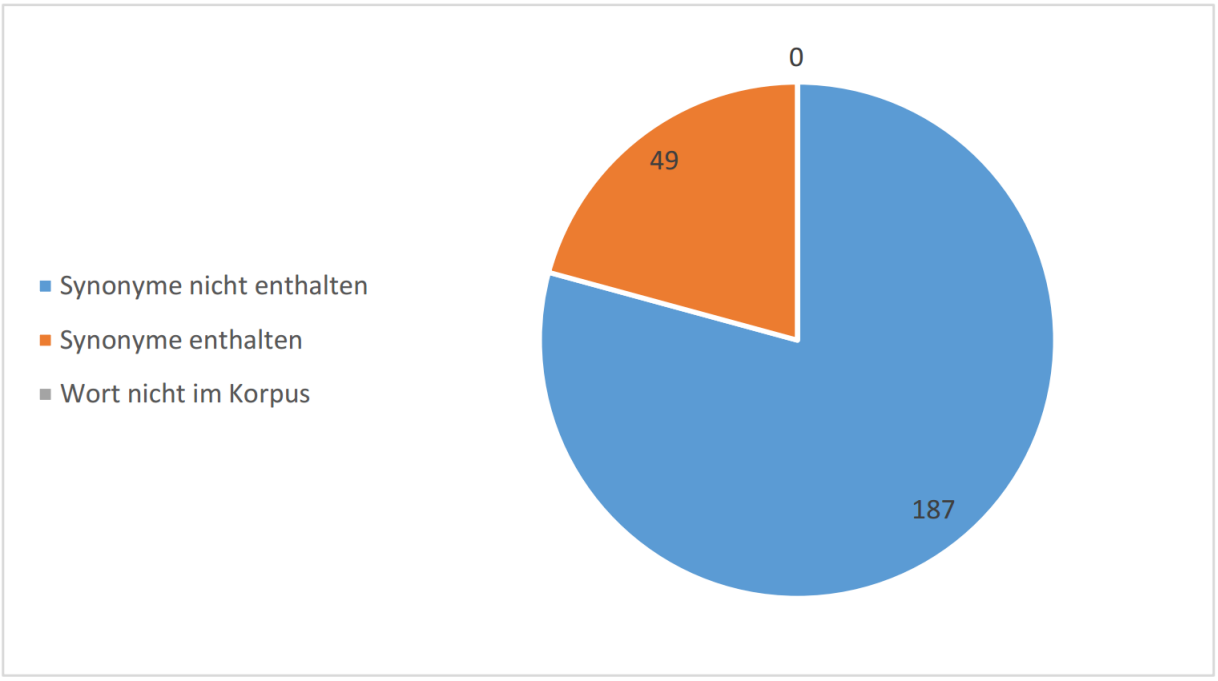

Abbildung 5.1: Synonyme durch Rekursion kompletter Korpus.

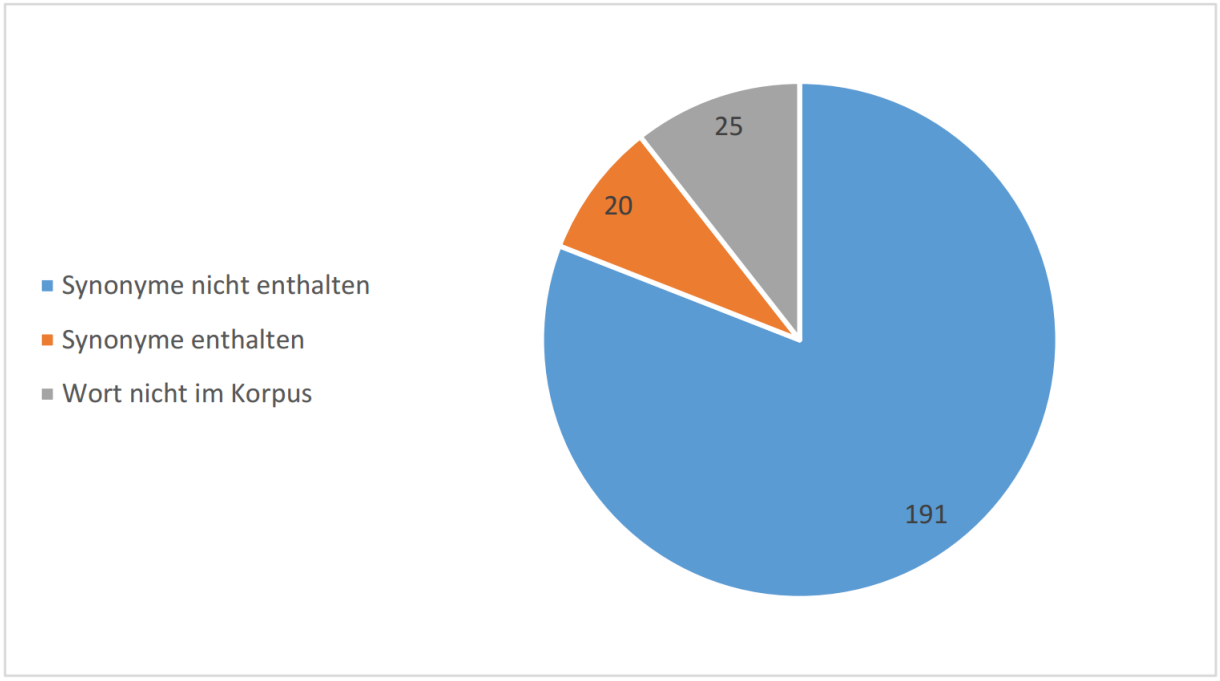

Abbildung 5.2: Synonyme durch Rekursion Technologiekorpus.

5.1.3 Interpretation/Ergebnis

Wie aus den Daten hervorgeht, werden zwar einige Synonyme der Suchbegriffe gefunden, allerdings nicht in großen Mengen. Dazu kommt auch, dass sich die beiden Korpora nochmal unterscheiden. Im kompletten Korpus wurden in *20,8%* der Testfälle Synonyme gefunden, wobei es im Technologiekorpus sogar nur weniger als halb so viele (*8,5%*) waren.

Somit ist dieses Experiment widerlegt.

5.2 Konkretisierungen

5.2.1 Beschreibung

.

5.2.2 Durchführung

.

5.2.3 Interpretation/Ergebnis

.

5.3 Verallgemeinerungen

5.3.1 Beschreibung

.

5.3.2 Durchführung

.

5.3.3 Interpretation/Ergebnis

.

5.4 Unterschiedliche Beziehungen

5.4.1 Beschreibung

.

5.4.2 Durchführung

.

5.4.3 Interpretation/Ergebnis

.

5.5 Mehrdeutigkeit

5.5.1 Beschreibung

.

5.5.2 Durchführung

.

5.5.3 Interpretation/Ergebnis

.

Kapitel 6

Fazit und Ausblick

6.1 Fazit

6.2 Ausblick

Subsampling, N-Gram (Phrases) lernen

Quellenverzeichnis

Literaturverzeichnis

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[MSC+13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

[ŘS10] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. http://is.muni.cz/publication/884893/en.

papers und so

Tabellenverzeichnis

Abbildungsverzeichnis

Kapitel 7

Anhang

7.1 Testdaten

Tabelle 7.1: Testdaten Teil 1

3d	3ds	3g	4chan
4g	acer	acta	activision
adobe	amazon	android	anonymous
aol	apple	app	augmented
arcade	architecture	arpanet	asus
auto	automobile	battlefield	bing
biometrics	bitcoin	bittorrent	blackberry
blizzard	blogging	blog	bluray
broadband	browser	casual	chatroulette
chrome	chromebook	cispa	computing
console	cookies	craigslist	crowdfunding
crowdsourcing	cryptocurrency	cybercrime	cyberwar
darknet	data	dell	diablo
doodle	dotcom	drone	dropbox
e3	ebay	email	emoji
encryption	energy	engine	engineering
ereader	events	facebook	fat
filesharing	firefox	flickr	foursquare
gadget	game	gameplay	gamergate

Tabelle 7.2: Testdaten Teil 2

games	gaming	ghz	gmail
google	googlemail	gps	groupon
gta	hacking	halo	handheld
hardware	hashtag	hd	heartbleed
htc	html5	i	ibm
icloud	ie	imac	indie
instagram	intel	internet	ios
ipad	iphone	ipod	isp
itunes	keyboard	kickstarter	kindle
kinect	laptop	lenovo	lg
limewire	link	linkedin	linux
live	machinima	macintosh	macworld
malware	mario	megaupload	microsoft
minecraft	mmorpg	mobile	monitor
motoring	mouse	mozilla	myspace
nes	net	netbook	nfs
nintendo	nokia	oracle	ouya
p2p	paypal	pc	phablet
phishing	photography	photoshop	pi
pinterest	piracy	pirate	platform
playback	playstation	pokemon	power
processor	programming	ps	ps2
ps3	ps4	psp	python
raider	ram	raspberry	rayman
recommendation	reddit	retro	robot
rpg	rts	safari	samsung
search	security	seo	skype
smartphone	smartphones	smartwatch	smartwatches
software	sonic	sony	sopa
spam	spotify	steam	stream
starcraft	stuxnet	sun	surface
symbian	tablet	technology	technophile
ted	telecom	television	tetris
titanfall	tomb	trojan	tumblr
twitch	twitter	viber	vine
virus	warcraft	web	whatsapp
wheel	wifi	wii	wikipedia
windows	windows7	wireless	worms
wow	xbox	xp	y2k
yahoo	youtube	zelda	zynga