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

im Studiengang
Medieninformatik (MI7)

Semantische Beziehungen in Texten mit Word2Vec

und der Vergleich zwischen allgemeinen und domänenspezifischen Korpora als Trainingsdaten

vorgelegt von Ruben Müller
an der Hochschule der Medien Stuttgart,
am 29.07.2015
zur Erlangung des akademischen Grades eines Bachelor of Science

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, ehrenwörtlich, dass ich die vorliegende Bachelorarbeit (bzw. Masterarbeit) mit dem Titel: „Semantische Beziehungen in Texten mit Word2Vec und der Vergleich zwischen allgemeinen und domänenspezifischen Korpora als Trainingsdaten“ selbstständig 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 ehrenwörtlichen Versicherung und die prüfungsrechtlichen Folgen (§26 Abs. 2 Bachelor-SPO (6 Semester), § 24 Abs. 2 Bachelor-SPO (7 Semester), § 23 Abs. 2 Master-SPO (3 Semester) bzw. § 19 Abs. 2 Master-SPO (4 Semester und berufsbegleitend) der HdM) einer unrichtigen oder unvollständigen ehrenwörtlichen Versicherung zur Kenntnis genommen.

Filderstadt, den 29. Juli 2015

Ruben Müller

Kurzfassung

Diese Bachelorarbeit beschäftigt sich mit der Analyse von semantischen Beziehungen innerhalb mit Word2Vec gelernten Modellen. Dazu sollen zum einen der allgemeine komplette Wikipedia-Korpus gelernt und analysiert werden, was als semantisch ähnlich erkannt wird. Zum anderen soll ein Korpus über die spezielle Domäne Technologie erstellt und gelernt werden. Diese beiden Korpora sollen sich dann gegenüber gestellt und mit fünf Experimenten 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

This bachelor thesis deals with the analysis of semantic relations in models which are learnd by Word2Vec. For this purpose on the one hand the whole and general wikipedia body shall be learned and analyzed what is detected as semantic similar. On the other hand a body covering the specific domain technology shall be created and learned. These both bodys shall then be compared against each other by five experiments to determine what is recognized as semanic similar.
Goal of this thesis will be to determine whether a general body or a domain specific body gives more precise results with regard to semantic similarity. Instead of using a general body it coud be better to choose between multiple domain specific bodys with regard to the actual domain.

Inhaltsverzeichnis

Begriffsverzeichnis

Begriff	Erklärung
Ähnliche Worte	Worte, die im Word2Vec-Modell mit der Methode *gensim.models.word2vec.Word2Vec.most_similar()*, erhalten werden
SVM	Support Vector Machine
NBC	Naive Bayes Classifier
CBOW	Continuous bag-of-words
tf-idf	Term-Frequency Inverse Document Frequency
LSI	Latent Semantic Indexing
NLP	Natural Language Processing
NNMF	Nicht-negative Matrix Faktorisierung
NNLM	Neuronal Net Language Model
NCE	Noise Contrastive Estimation

Kapitel 1

Einleitung

1.1 Motivation und Problemstellung

Um semantische Beziehungen zwischen Worten abzubilden, gibt es unter anderem die Möglichkeit, Modelle mittels Word2Vec darzustellen. Wie diese Modelle die Beziehungen aufbauen, hängt nicht nur von den Trainingsparametern der Modelle ab, sondern auch von den verwendeten Trainingsdaten. So können beispielsweise allgemeine Daten, die keine spezielle Fachdomäne abbilden, als Trainingsdatenkorpus benutzt werden oder aber ausschließlich Daten über eine spezielle Domäne.
In dieser Arbeit wird untersucht, wie sich unterschiedliche Textgrundlagen (Korpora) als Trainingsdaten auf die semantischen Beziehungen zwischen Worten im Word2Vec Modell auswirken.
Die Ergebnisse dieser Arbeit sollen es ermöglichen, dass für eine spezielle Fragestellung oder Anwendung des Word2Vec Modells, das richtige Modell, bzw die richtigen Trainingsdaten ausgewählt werden können.

1.2 Verwandte Arbeiten

Worte als kontinuierliche Vektoren darzustellen hat eine lange Geschichte [HMR86], [WH86],[Elm90]. Auch die Benutzung neuronaler Netze um Wortvektoren zu lernen, wurde bereits bei [BDVJ03] beschrieben und in [Mik07] und [MKB$^+$09] erweitert. Die Arbeitsweise von Word2Vec wurde in [MCCD13] und [MSC$^+$13] beschrieben, die Ähnlichkeiten zwischen Worten mit einfachen algebraischen Operationen zu berechnen wurde in [MYZ13] erklärt.

1.3 Aufbau der Arbeit

Im folgenden Abschnitt wird kurz der Aufbau dieser Arbeit beschrieben.

Kapitel 2: Daten und Vorverarbeitung
In diesem Kapitel wird erläutert, welche Daten in dieser Arbeit benutzt werden. Des Weiteren wird kurz erklärt, in welchen Vorverarbeitungsschritten diese Daten bearbeitet wurden um in den nächsten Kapiteln verwendet werden zu können.

Kapitel 3: Word2Vec
Hier werden kurz die unterschiedlichen Trainingsmöglichkeiten im Word2Vec Modell erklärt und die wichtigsten Parameter vorgestellt.

Kapitel 4: Wikipedia-Korpus
In Kapitel 4 werden der komplette Wikipedia-Korpus sowie ein Teilkorpus daraus vorgestellt und die zum Training des Word2Vec Modells ausgewählten Parameter erläutert. Außerdem werden die Testdaten vorgestellt.

Kapitel 5: Experimente
Kapitel 5 beschäftigt sich mit den Ergebnissen der durch die Testdaten erhaltenen ähnlichen Worte der unterschiedlichen Word2Vec Modelle.

Kapitel 6: Fazit und Ausblick
Das Kapitel 6 beinhaltet ein Fazit der Arbeit und es wird ein Ausblick auf zukünftige Arbeiten gegeben.

Kapitel 7: Anhang
Hier sind die Testdaten und weitere Auflistungen ausführlich dargestellt.

Kapitel 2

Daten und Vorverarbeitung

2.1 Datenbasis

In dieser Arbeit werden unterschiedliche Datenkorpora als Trainingsdaten für unterschiedliche Word2Vec Modelle benutzt. Da zum erfolgreichen Training des Word2Vec Modells eine sehr große Menge an Daten benötigt werden[1], wird für den ersten Korpus der komplette englischsprachige Wikipedia Korpus verwendet[2]. Der zweite in der Arbeit benütze Korpus besteht auch aus Wikipedia Artikeln, allerdings ausschließlich aus technologiespezifischen Artikeln. Hierzu wurde der komplette englische Wikipedia Korpus zuerst in die einzelnen Artikel aufgeteilt und diese wurden dann mit einem NBC in die Klassen *tech, entertainment, sport, science, politic* eingeteilt. Im Anschluss wurden dann die als *tech* klassifizierten Artikel zusammengefasst.

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.
Der **NBC**, der zum Klassifizieren der Artikel verwendet wurde, ist der im DataMining Praktikum[4] selbst implementierte.

[1] In [MSC+13] werden Trainingsdaten mit bis zu 30 Milliarden Worten 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
[4] https://www.hdm-stuttgart.de/~maucher/Data_Mining_SS15.html und https://www.mi.hdm-stuttgart.de/mib/studium/intern/skripte/Data_Mining/WS1213/V4SpamFilter.pdf, abgerufen am 17.07.2015

PyCharm[5] wurde als Editor und zum Ausführen der Python Skripte verwendet. **Thesaurus** ist ein Webservice[6], der mittels einer HTTP GET Nachricht, mit den Parametern *word*, *language* und *key*, eine Liste von Synonymen des Wortes *word* liefert. Die Synonyme stammen aus den Thesaurus Lexika von OpenOffice[7].

2.3 Vorverarbeitung

Der komplette Wikipediadump kann als komprimierte XML-Datei heruntergeladen werden[8]. Die entpackte XML-Datei hat eine Größe von 48,8 GB. Um die Daten als Trainingsdaten für Word2Vec Modelle zu verwenden, müssen diese zuerst bereinigt werden..

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[9], welches von Mikolov et al. auf der Google Code Seite von Word2Vec vorgeschlagen wird[10]. Dieses Skript wurde speziell zum Bereinigen von Wikipediadaten erstellt und wurde leicht verändert, sodass Zahlen, Satzzeichen und die Groß- und Kleinschreibung erhalten bleiben. Außerdem werden die Umlaute in *ae*, *oe* und *ue* umgewandelt[11]. 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*[12], 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[13] geparst und dann in einzelne Dateien geschrieben werden. Diese einzelnen Dateien können darauf mit dem bereits erwähnten 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.

[5] https://www.jetbrains.com/pycharm/, abgerufen am 02.07.2015
[6] http://thesaurus.altervista.org/thesaurus/v1, abgerufen am 20.07.2015
[7] https://www.openoffice.org/, abgerufen am 20.07.2015
[8] http://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2, abgerufen am 09.04.2015
[9] http://mattmahoney.net/dc/textdata.html unter Appendix A, abgerufen am 25.06.2015
[10] https://code.google.com/p/word2vec/, abgerufen am 25.06.2015
[11] Für das komplette geänderte Skript siehe Anhang 7.3
[12] Diese Klasse kann auch als Input zum Trainieren des Word2Vec Modells genutzt werden.
[13] Es wurde die Klasse *xml.sax.handler.ContentHandler* verwendet.

Kapitel 3

Word2Vec

Ziel dieses Kapitels ist es, Word2Vec zu erklären und dazulegen, welche wichtigen Parameter und Algorithmen bei der Berechnung und dem Erstellen von Word2Vec Modellen benutzt werden können.
Wie bereits die Bezeichnung Word2Vec verdeutlicht, werden im Word2Vec Modell Worte als Vektoren dargestellt. Diese Idee der Darstellung von Worten ist nicht neu und hat schon eine längere Geschichte [HMR86],[WH86],[Elm90], auch im Bereich des *Information retrieval* werden Daten als Vektoren dargestellt, so werden beispielsweise beim *LSI* [DDL+90] Dokumente als Vektoren dargestellt.
Ein Ziel von Word2Vec ist es, semantisch oder syntaktisch ähnliche[1] Worte zu finden. Für dieses Vorhaben gab es schon vor Word2Vec andere statistische Methoden, so können neben dem schon genannten LSI auch mit dem Verfahren der NNMF ähnliche Worte gefunden werden. Allerdings werden, bei den beiden genannten Verfahren und anderen NLP-Techniken (z.B. tf-idf), Worte nur als Anzahl in den jeweiligen Dokumenten dargestellt, dort kann eine Ähnlichkeit zwischen Worten nur dann gefunden werden, wenn diese Worte in vielen Dokumenten gemeinsam vorkommen. Beim Word2Vec Modell wird ein anderer Ansatz benutzt, hier werden Worte als ähnlich erkannt, wenn diese in den Trainingsdaten häufig im gleichen Kontext vorkommen.

Im ersten Schritt der Modellbildung werden alle unterschiedlichen Worte aus den Trainingsdaten, in einem *Vokabular* abgebildet und alle Worte, die eine Mindestanzahl erfüllen, werden mittels *distributed representation* in einem n-dimensionalen Vektorraum dargestellt [MCCD13].
Im nächsten Schritt dienen diese Vektorrepräsentationen der Worte als Input für

[1] Die Ähnlichkeit zwischen zwei Worten kann berechnet werden, indem die Kosinusähnlichkeit zwischen den Vektordarstellungen der beiden Worte gebildet wird.

ein neuronales Netz. Mehrere nicht lineare verdeckte Schichten in neuronalen Netzen machen die Modelle genauer, allerdings kommt auch die meiste Komplexität des ganzen Models von solchen verdeckten Schichten. Durch die Komplexität steigt auch die Trainingsdauer. Mikolov et al. [MCCD13] haben deshalb neuronale Netze ohne verdeckte Schichten bevorzugt, da damit sehr große Datenmengen viel effizienter gelernt werden können. So können mit einer optimierten Implementierung auf einem einzigen Rechner mehr als 100 Milliarden Worte pro Tag trainiert werden [MSC+13].
In dem neuronalen Netz werden die Vektoren der Worte dann gelernt und angepasst. Wie schon genannt, werden im Word2Vec Modell Worte als ähnlich erkannt, wenn sie in den Trainingsdaten oft im gleichen Kontext vorkommen, das wird durch die Architektur und Arbeitsweise der neuronalen Netze erreicht. Mikolov et al. stellen zwei Architekurtypen von neuronalen Netze ohne verdeckte Schichten vor [MCCD13]. Zum einen die CBOW und zum anderen die Skip-gram Architektur. In der Abbildung 3.1 sind die Unterschiede der beiden Architekturen dargestellt.

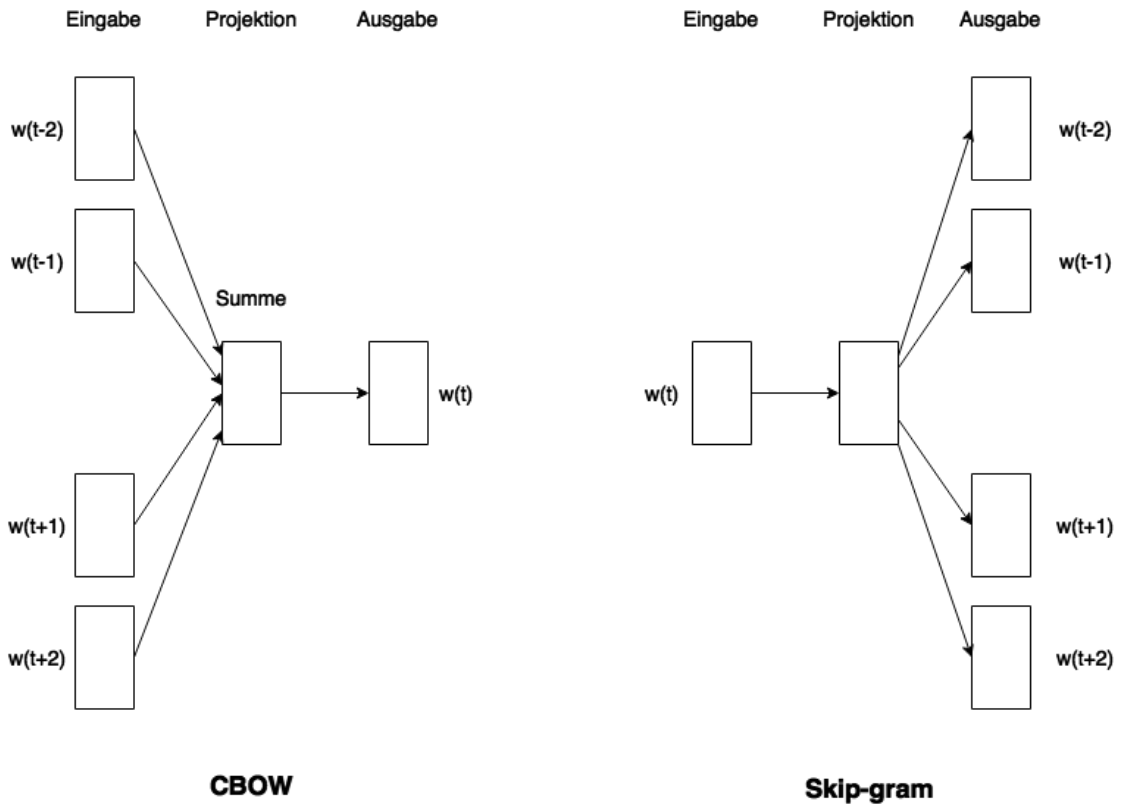

Abbildung 3.1: CBOW und Skip-gram im Vergleich, übersetzt nach [MCCD13]

3.1 CBOW

Durch Bengio et al. [BDVJ03] wurde das *Feedforward Neural Net Language Model (NNLM)* beschrieben. Es ist ein neuronales Netz, mit Eingabe-, Projektions-, verdeckten- und Ausgabeschichten. Ziel des neuronalen Netzes ist es, aus N Eingabewörtern möglichst genau das darauf folgende Wort vorherzusagen.
Die CBOW Architektur ist ähnlich zum NNLM, allerdings wurden die verdeckten Schichten entfernt und die Projektionsschicht wird von allen Worten geteilt. So werden alle Worte auf die gleiche Position projiziert (das vorherzusagende Wort). Es werden auch nicht nur vergangene Worte, sondern auch Worte, die dem zu vorhersagenden Wort nachstehen, beim Training benutzt. Trainingskriterium ist hier das momentane (mittlere) Wort korrekt vorherzusagen, so lange werden die Vektoren der Wortrepräsentationen ständig angepasst. Kurz gesagt, beim CBOW Algorithmus wird ein Wort aus dem Kontext vorhergesagt.

3.2 Skip-gram

Die Architektur des Skip-grams ist ähnlich zur CBOW Architektur. Allerdings wird bei Skip-gram nicht aus dem Kontext ein Wort vorhergesagt, sondern es wird ein Bereich an Worten, vor und nach dem aktuellen, anhand des aktuellen Wortes vorhergesagt. Wird der Bereich, der vorhergesagt wird, vergrößert, verbessern sich die Wortvektoren, allerdings erhöht dies die Berechnungskomplexität des gesamten Models. Da weiter entferntere Worte weniger mit dem aktuellen Wort zu tun haben, als nahestehende, wird dies in den Gewichten beim Training beachtet[MCCD13]. Kurz gesagt, beim Skip-gram Algorithmus wird aus einem Wort der Kontext vorhergesagt.

Zur Skip-gram Architektur haben Mikolov et al. [MSC+13] eine weitere Arbeit verfasst, in der sie Methoden zur Verbesserung vorstellen, sowie die Trainingsfunktion des *hierarchical softmax* erläutern und eine Alternative, das *negative sampling*, vorschlagen. Im Folgenden werden *hierarchical softmax* und *negative sampling* vorgestellt.

Hierarchical softmax[2]

Ein mehr formelles Trainingsziel des Skip-gram Modells beschreiben Mikolov et al.[MSC+13] als die durchschnittliche log Wahrscheinlichkeit zu maximieren, ge-

[2]Vergleiche [MSC+13] in diesem Abschnitt.

geben einer Sequenz von Trainingsworten $w_1, w_2, ..., w_T$,

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t) \tag{3.1}$$

wobei c die Größe des Kontextes darstellt[MSC+13]. Die Definition der allgemeinen Softmax Funktion in Skip-gram definiert $p(w_{t+j}|w_t)$, laut Mikolov et al. [MSC+13], als:

$$p(w_O|w_I) = \frac{\exp({v'_{w_O}}^T v_{w_I})}{\sum_{w=1}^{W}\exp({v'_w}^T v_{w_I})} \tag{3.2}$$

mit v_w und v'_w als „Input"und „Output" der Vektorrepräsentation des Wortes w, sowie W als Anzahl der Worte im Vokabular. Weil die Wahrscheinlichkeit p proportional zu W ist, ist diese Definition nicht praktikabel, da W oft sehr groß $(10^5 - 10^7)$ ist[MSC+13].

Eine Annäherung an die allgemeine softmax Funktion ist die hierarchical softmax Funktion. Morin und Bengio verwendeten diese im Kontext der *neuronal network language models* zuerst [MB05]. Nach Mikolov et al. [MSC+13] ist der Hauptvorteil, dass anstatt W Ausgabeknoten im neuronalen Netz nur ungefähr $log_2(W)$ Knoten ausgewertet werden müssen, um die Wahrscheinlichkeiten zu errechnen. Weiter beschreiben sie, dass jedes Wort w durch einen geeigneten Pfad vom Wurzelknoten aus erreicht werden kann. So ist $n(w, j)$ der j-te Knoten auf dem Weg von der Wurzel zum Wort w mit der Pfadlänge von $L(w)$. Daraus ergibt sich, dass $n(w, 1)$ der Wurzelknoten ist und $n(w, L(w)) = w$. Außerdem ist $ch(n)$ als beliebiger, aber fixer Kindknoten von w definiert[MSC+13]. Nach Mikolov et al. [MSC+13] sei der Ausdruck [[x]] gleich 1, wenn x wahr ist, andernfalls -1. Das *hierarchische softmax* definiert die Wahrscheinlichkeit $p(w_O|w_I)$ wie folgt [MSC+13]:

$$p(w|w_I) = \prod_{j=1}^{L(w)-1}\sigma([[n(w, j+1) = ch(n(w, j))]] \cdot {v'_{n(w,j)}}^T v_{w_I}) \tag{3.3}$$

wobei σ die Sigmoid-Funktion $\sigma(x) = \frac{1}{1+\exp(-x)}$ darstellt. Nach Mikolov et al. [MSC+13] ist belegt, dass $\sum_{w=1}^{W} p(w|w_I) = 1$, das impliziert, dass die Berechnungskosten von $\log p(w_O|w_I)$ proportional zu $L(w_O)$ sind, wobei dies im Durchschnitt nicht größer als $\log W$ ist. Anders als bei der allgemeinen Definition des softmax im Skip-gram, bei der jedem Wort w zwei Darstellungen v_w und v'_w , laut Mikolov et al. [MSC+13], zugeordnet werden, wird bei der Definition des hierarchical softmax jedem Wort w eine Darstellung v_w und jedem inneren Knoten n des Binärbaumes je eine Dar-

stellung v'_n zugeordnet. Mikolov et al. [MSC+13] erklären, dass die Struktur des Baumes, der beim hierarchical softmax benutzt wird, einen großen Einfluss auf die Performanz hat. Sie verwenden einen Huffman Binärbaum, dieser ist so aufgebaut, dass häufig genutzte Worte kurze Kodierungen haben, was sich positiv auf das Trainingstempo auswirkt.

Negative sampling[3]

Als Alternative zur hierarchical softmax Funktion kann *Noise Contrastive Estimation(NCE)* benutzt werden [MSC+13]. Die NCE wurde zuerst von Gutmann und Hyvarinen [GH12] beschrieben und zuerst durch Mnih und Teh [MT12] im Bereich der Sprachmodellierung verwendet. Nach Mikolov et al. [MSC+13] sieht die NCE es für notwendig, dass ein gutes Modell Daten von Rauschen unterscheiden kann. Mikolov et al. [MSC+13] erläutern, dass die NCE zur Maximierung der log Wahrscheinlichkeit der softmax Funktion dienen kann, wobei sich aber das Skip-gram Modell nur mit dem Lernen von qualitativ hochwertigen Vektordarstellungen beschäftigt. Deshalb haben Mikolov et al. [MSC+13], mit der Voraussetzung, dass bei der Vektordarstellung die Qualität erhalten bleibt, das *Negative sampling* wie folgt definiert:

$$\log \sigma({v'_{w_O}}^T v_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} \left[\log \sigma(-{v'_{w_i}}^T v_{w_I}) \right] \tag{3.4}$$

Mit diesem Term wird dann jeder $\log p(w_O|w_I)$ Term im Skip-gram ersetzt. Ziel ist es, das Zielwort w_O aus der Rauschverteilung $P_n(w)$ mittels logistischer Regression zu erkennen[MSC+13]. Bei der Regression gibt es k Negativbeispiele, nach Experimenten von Mikolov et al. [MSC+13] sollte k zwischen 5-20 in kleineren Trainingssets und zwischen 2-5 bei großen Trainingssets liegen. Mikolov et al. [MSC+13] fanden auch durch Experimente eine optimale Rauschverteilung für $P_n(w)$ und dass die Unigramverteilung $U(w)$ um 3/4 potenziert (z.B. $U(w)^{3/4}/Z$), die Unigram- und Uniformverteilung weit übertreffen.
Der Hauptunterschied zwischen Negative sampling und NCE ist nach Mikolov et al. [MSC+13], dass bei der NCE sowohl die Stichprobe, als auch die numerische Wahrscheinlichkeit der Rauschverteilung gebraucht wird. Hingegen braucht *Negative sampling* nur die Stichproben. Ein weiterer Unterschied sei, dass NCE die log Wahrscheinlichkeit des softmax durchschnittlich maximiert, was aber nicht wichtig für diese Anwendung ist[MSC+13].

[3]Vergleiche [MSC+13] in diesem Abschnitt.

Kurz gesagt, unterscheidet sich das negative sampling 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.

Zum Training mit diesen Funktionen können folgende Parameter verwendet werden:

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. In [MCCD13] werden bei den Beispielen Werte zwischen 50 und 1000 für *size* verwendet.

window

window ist der maximale Abstand zwischen benachbarten Worten, innerhalb eines Satzes, die zur Berechnung der Wordvektoren betrachtet werden (≙ Kontext). Auf der Google Code Seite von Word2Vec[4] wird eine *window* Größe bei der Skip-gram Architektur von ungefähr zehn, bei CBOW von ungefähr fünf vorgeschlagen.

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. In der Gensim-Implementierung von Word2Vec[5] wird ein voreingestellter Wert von fünf verwendet.

negative

Dieser Parameter wird nur benötigt, wenn als Lernalgorithmus negative sampling verwendet wird. Er gibt an, wie viele zufällig ausgewählten Worte beim Test auf Ähnlichkeit verwendet werden sollen. Im Artikel von Mikolov et al. [MSC+13] wird für diesen Wert bei kleinen Trainingskorpora ein Wert aus dem Bereich zwischen 5 und 20 vorgeschlagen, für große Trainingskorpora zwischen 2 und 5.

[4]https://code.google.com/p/word2vec/, abgerufen am 26.07.2015
[5]https://radimrehurek.com/gensim/, abgerufen am 24.06.2015

3.3 Unterschiedliche Arten von Ähnlichkeiten

Die gelernten Wortrepräsentationen sind erstaunlicherweise sehr gut im Erkennen von syntaktischen und semantischen Regelmäßigkeiten in natürlicher Sprache und diese bedeutungsvollen Beziehungen werden in einer sehr einfachen Art und Weise erfasst[MYZ13]. Es kann festgestellt werden, dass *groß* zu *größer* die gleiche Beziehung hat, wie *klein* zu *kleiner*[MCCD13]. Dadurch können Fragen in der Art „Welches Wort ist ähnlich zu *klein*, in der Form in der *am größten* zu *groß* ähnlich ist?“ an das Modell gestellt werden. Um solche Fragen beantworten zu können, reicht es aus, einfache algebraische Operationen auf den Vektordarstellungen der Worte auszuführen[MCCD13]. So muss, um das Wort zu finden, welches ähnlich zu *klein* im gleichen Sinn wie *am größten* zu *groß* ist, folgender Vektor errechnet werden $X = vector(\text{„am größten“}) - vector(\text{„groß“}) + vector(\text{„klein“})$, dann wird der Vektorraum abgesucht um das nächste Wort dieses Vektors X zu finden[MCCD13]. Der Abstand wird mit der Kosinusähnlichkeit gemessen. Das dann erhaltene Wort wird als Lösung für die Frage verwendet. Jede solche Ähnlichkeit wird mit einem beziehungsspezifischen Offsetvektor charakterisiert[MYZ13], so sind Wortpaare, die die gleiche Art von Ähnlichkeit haben, durch den gleichen Offsetvektor getrennt. So stellt die Ähnlichkeit zwischen *Mann* und *Frau*, *Onkel* und *Tante* sowie *König* und *Königin*, die Beziehung *männlich:weiblich* dar. In Figur 3.2 ist dies veranschaulicht. Dadurch, dass jede Beziehung ihren eigenen Offsetvektor hat, erlaubt es das Modell, dass die einzelnen Worte viele unterschiedliche Beziehungen haben können.

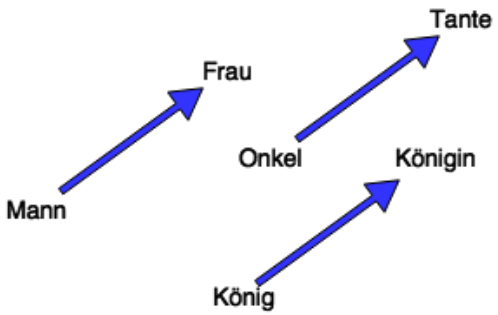

Abbildung 3.2: Ähnlichkeiten von Worten, als Offsetvektoren dargestellt. Nach Figur 2 in [MYZ13].

Wortvektoren, die solche semantische Ähnlichkeiten darstellen können, könnten auch zur Verbesserung in vielen bereits existierenden NLP Anwendungen, wie maschinelles Übersetzen, Informationsgewinnung oder Systeme zur Fragebeantwortung, verwendet werden[MCCD13].

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, dazu zählt 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, eignet sich der Wikipedia Korpus besonders gut, da Wikipedia eine breite Menge an Themen beschreibt.
Wie in 2.3 beschrieben, müssen die Daten erst einer Säuberung unterzogen werden, um dann anschließend für das Training im Word2Vec Modell benutzt 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 09.04.2015

[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], wurde der *hierarchical softmax* Lernalgorithmus gewählt, da das Modell (wie im Folgenden gezeigt wird), eine hohe Dimension an Vektoren hat und auf technologiespezifische Daten getestet werden soll (siehe 7.2 Testdaten).

Um die optimalen Parameter für das Training des Modells herauszufinden, wurde ein kleinerer Wikipedia Auszug[5] als Trainingsdaten benutzt und mit unterschiedlichen Parametern wurden mehrere Modelle gelernt und evaluiert (siehe Tabelle 4.1). Die Word2Vec-Klasse in der Gensim-Implementierung beinhaltet eine Evaluationsfunktion[6], 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. Ein solches 4-Tupel besteht aus zwei Wortpaaren, deren Worte die gleiche Beziehung untereinander haben, zum Beispiel „Athens Greece Berlin Germany“, hier stellen beide Wortpaare die Beziehung zwischen einem Land und dessen Hauptstadt dar. Auf der Google Code Seite ist eine solche Datei als Beispiel vorhanden[7]. In diesem Beispiel sind 14 Kategorien[8] mit insgesamt 19544 4-Tupel aufgelistet.
Das Word2Vec Modell stellt eine Funktion bereit, *most_similar(positive=[], negative=[], topn=N)*, bei der die Vektoren der Worte im Array *positive* aufaddiert und die Vektoren der Worte in *negative* davon subtrahiert werden. Die Funktion liefert die N Worte zurück, deren Vektordarstellungen eine maximale Kosinusähnlichkeit mit dem Ergebnis der Vektoraddition und -subtraktion haben. Diese Funktion wird auch in der Evaluationsfunktion benutzt, indem die ersten drei Worte des 4-Tupels in *positive* und *negative* eingeteilt werden[9], wird das vierte Wort, hier „Germany“, korrekt vom Model vorhergesagt, wird dieser Testdatensatz als erfolgreich gezählt. Sollten Worte nicht im Modell dargestellt sein, wirkt sich dies nicht negativ auf die Accuracy aus, denn die Gesamtaccuracy eines Modells wird als Verhältnis, zwischen

[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] *gensim.models.word2vec.Word2Vec.accuracy(FILENAME)*

[7] https://code.google.com/p/word2vec/source/browse/trunk/questions-words.txt, abgerufen am 29.06.2015

[8] *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*

[9] *most_similar(positive=[‘Greece‘,‘Berlin‘], negative=[‘Athens ‘], topn=1)*

der Anzahl der richtig vorhergesagten Worte und der Anzahl der Testdatensätze, bei denen alle Worte eine Vektordarstellung im Modell haben, dargestellt.
In der Tabelle 4.1 werden die unterschiedlichen Parameter[10] und die erreichte Gesamtaccuracy dargestellt. In diesem kleineren Modell waren nur bei 13144 der 19544 4-Tupel alle Worte als Vektordarstellungen im Modell vorhanden.

Tabelle 4.1: Vergleich Parameter

Size	**Window**	**Min_count**	**Gesamtaccuracy**
400	10	5	53,5% (7027/13144)
400	10	10	52,5% (6905/13144)
300	10	5	52,5% (6860/13144)
300	10	10	52,9% (6951/13144)
200	10	5	50,5% (6632/13144)
200	10	10	50,5% (6636/13144)
100	10	5	42,0% (5517/13144)
100	10	10	41,3% (5431/13144)

Auf dem kleineren Wikipedia Korpus erzielte das Modell mit den Parametern *size* = 400, *window* = 10, *min_count* = 5 die beste Gesamtaccuracy mit 53,5%. Diese Parameter wurden auf das Gesamtmodell übertragen. Die Trainingszeit beträgt bei diesen Parametern 10,5h[11] beim Gesamtmodell. Nachdem ein kleinerer Fehler im Reiningungsskript aufgetretetn war[12], musste das komplette Modell nochmals gelernt werden. Hier fiel dann die Entscheidung, andere Parameter zu benutzen und die *size* Größe auf 300 zu reduzieren, da dies die Accuracy nur sehr gering verändert und sich die Trainingszeit auf 7,7h verkürzte. Dies zahlte sich im Laufe der Arbeit aus, da sich das Perl-Reinigungsskript ein paar Mal geringfügig änderte um so kleinere Fehler aus dem Korpus zu entfernen[13].

4.2 Teilkorpus

Der zweite Korpus ist ein Teilkorpus des Gesamtkorpus, bestehend aus Technologieartikeln. Um diesen Korpus zu erstellen, muss der Gesamtkorpus, wie in 2.3

[10] Auf der Google Code Seite von Word2Vec wird eine window size bei der Skip-gram Architektur von 10 vorgeschlagen, somit wird sie hier nicht verändert.
[11] Leistungsdaten: PC mit 32 GB RAM, i7-3770 Quadcore bei 3,4 GHz
[12] Unbekannte Zeichen wurden mit einem Leerzeichen ersetzt, anstatt gelöscht zu werden, somit entstanden Wortfragmente als eigene Worte.
[13] Siehe 7.3

beschrieben, zuerst in die einzelnen Wikipediaartikel unterteilt werden. Diese Artikel werden dann mithilfe eines NBC klassifiziert. Ein NBC basiert auf dem Prinzip des überwachten Lernens. Hierzu müssen vor dem Training von Hand ausgewählte Artikel in die fünf zu klassifizierenden Themenbereiche[14] eingeteilt werden. In jeder Kategorie wurden gleich viele Artikel ausgewählt, somit entsteht kein Ungleichgewicht beim Training. Es wurden je Kategorie 100 Artikel verwendet. Mit diesen Daten kann der NBC dann trainiert werden. Im Anschluss daran, kann der NBC für die Klassifikation der Wikipediaartikel verwendet werden. Die Artikel, die in die Kategorie *tech* eingeteilt werden, bilden dann den technologiespezifischen Teilkorpus, der als zweiter Korpus in dieser Arbeit verwendet wird.

Dieser domänenspezifische Teilkorpus enthält 187.144 Artikel (2,2% im Vergleich zum Gesamtkorpus), 9.866.096 Worte(0,34%) und 1.240.949 Sätze(0,51%). Hier beträgt die Trainingszeit ca. 2 Minuten.
Wird dieser Korpus mit der *.accuracy()*-Methode evaluiert, erreicht er eine Gesamtaccuracy von nur 7,0% (490/6968). Das hat den Grund, dass die Testfragen von allgemeiner Art sind und diese Beziehungen in einem reinen Technologiekorpus sehr selten bis gar nicht vorkommen.

Die Wahl von anderen Daten wäre auch möglich gewesen, beispielsweise könnten Korpora aus Foren zu technologischen Themen oder Technews Internetseiten, automatisch mittels Crawling, erstellt werden. Die Verwendung des Wikipedia Teilkorpus ist jedoch zielführender, da hier die gleichen Texte in beiden Korpora vorhanden sind und somit auch die gleichen Beziehungen zwischen den einzelnen Worten, um so einen möglichst objektiven Vergleich zwischen den unterschiedlichen Korpora zu ermöglichen.

4.3 Testdaten

Im folgenden Kapitel sollen die beiden Word2Vec Modelle, deren Trainingsdaten aus dem Gesamtkorpus, beziehungsweise dem Technologiekorpus bestehen, miteinander verglichen werden. Für diesen Vergleich werden einige Experimente durchgeführt (siehe 5.1 - 5.5), in diesen Experimenten werden die semantischen Beziehungen zwischen Worten betrachtet. Um diese Experimente durchführen zu können, werden Testdaten benötigt.

[14] tech, entertainment, sport, science, politic

Es wird die Annahme getroffen, dass die Testdaten aus der Domäne Technologie (oder einer Unterdomäne) stammen müssen, da das Vokabular eines Modells aus Technologieartikeln besteht und auf diesen trainiert wird und da dann eher sichergestellt ist, dass diese Worte auch wirklich im Modell vorhanden sind und somit eine Vektordarstellung haben.
Um diese Annahme zu überprüfen, wurden aus den drei Domänen *Medizin*, *Mode/Fashion* und *Technologie/Pc/Internet* je 50 Begriffe ausgewählt um zu testen, ob diese überhaupt im Technologiemodell vorhanden sind[15]. Die Begriffe wurden von Hand aus Wikipedia Artikeln ausgesucht, um sicherzustellen, dass diese Begriffe im Gesamtmodell eine Vektorrepräsentation haben.
Wird im Word2Vec Modell mit der Methode *most_similar(positive=['word'], topn=1)* nach ähnlichen Worten gesucht, aber das Suchwort ist nicht im Modellvokabular vorhanden, wirft diese Methode eine „KeyError"-Exception. Dieses Verhalten wird bei den Tests zur Überprüfung, ob die Testbegriffe im Modellvokabular vorhanden sind, benutzt. Es wird gezählt, bei wie vielen Suchbegriffen eine solche Exception geworfen wird und diese somit nicht im Modell enthalten sind.

Die folgende Tabelle zeigt die Resultate der Worttests. Es sind Domäne, Gesamtanzahl an Testdaten, im Modell gefundene Testdaten und im Modell nicht gefundene Testdaten dargestellt.

Tabelle 4.2: Testdatenvergleich

Domäne	**Gesamtanzahl**	**gefundene Testdaten**	**nicht gefundene Testdaten**
Medizin	50	33	17
Mode/Fashion	50	21	29
Technologie/Pc/Internet	50	45	5

Die Resultate zeigen, dass am meisten Testdaten aus der Domäne *Technologie/Pc/Internet* im Modell gefunden werden. Somit bestätigt sich die Annahme, dass die Testdaten auch aus dieser Domäne stammen sollten. Die Testdaten wurden dann auf 236 Begriffe erweitert[16].

[15] Eine vollständige Auflistung dieser Daten ist im Anhang unter 7.1 Testdatenvergleich zu finden.
[16] Siehe Anhang 7.2 Testdaten.

Kapitel 5

Experimente

In diesem Kapitel sollen die beiden unterschiedlichen Word2Vec Modelle mit ihren Trainingskorpora (Gesamtkorpus und Technologiekorpus) untersucht werden. Dies soll durch fünf Experimente realisiert werden. In allen Experimenten werden semantischen Beziehungen zwischen Worten untersucht. Dabei ist jedes Experiment in drei Teile aufgeteilt, Beschreibung, Durchführung und Interpretation/Ergebnis.
Als Grundlage für die Experimente dienen die Testdaten[1] und die mit der Methode *most_similar()*[2] gefunden fünf ähnlichsten Worte. Zum einfacheren Vergleichen der ähnlichen Worte der unterschiedlichen Korpora wurde eine Datei erzeugt, in der die Testdaten und ihre fünf ähnlichsten Worte, mit den dazugehörigen Kosinusähnlichkeiten, leserlich formatiert, ausgegeben werden.

5.1 Synonymsuche durch Rekursion

Beschreibung

Im ersten Experiment wird untersucht, ob es möglich ist, im Word2Vec Modell Synonyme eines Begriffs zu finden. Hierzu sollen allerdings nicht die ähnlichen Worte der Testbegriffe betrachtet werden, sondern deren ähnliche Worte. Die Suche nach ähnlichen Worten wird somit ein Mal rekursiv angewandt. Die Überlegung ist, dass durch den rekursiven Aufruf Rückbeziehungen zu den ursprünglichen Suchbegriffen und deren Synonymen entstehen. Eine Anwendung für die Synonymsuche durch Rekursion könnte ein Thesaurus-Lexikon sein.

[1] Siehe 7.2 Testdaten
[2] gensim.models.word2vec.Word2Vec.most_similar()

Durchführung

Eine Liste mit den Synonymen der Testbegriffe wird vor dem Experiment mit der Hilfe eines Webservices[3] erstellt. Dieser Webservice liefert aber nur für 108 der insgesamt 236 Testwörter Synonyme. So wurden für dieses Experiment nur diese 108 Testwörter benutzt.
Im ersten Schritt dieses Experiments müssen die ähnlichen Worte (W_1) der Testbegriffe, mit der Methode *most_similar()* des Word2Vec Modells, herausgefunden werden. Diese fünf Worte (W_1) werden in der späteren Analyse nicht betrachtet. Im nächsten Schritt werden die fünf erhaltenen ähnlichen Worte (W_1) dann als Eingabe für die *most_similar()*-Methode benutzt und liefern wieder jeweils fünf ähnliche Worte (W_2) zurück. Die insgesamt 25 Begriffe werden im letzten Schritt des Experiments dann mit den Synonymen des ursprünglichen Testbegriff verglichen. Sobald einer der 25 erhaltenen Begriffe in der Liste der Synonyme des jeweiligen Testwortes enthalten ist, wird dieses Wort als *Synonym enthalten* gezählt.
Die nachfolgende Tabelle 5.1 listet die Resultate dieses Experiments auf. Es sind die Gesamtanzahl der Testworte, die Anzahl an Testworte, bei denen Synonyme gefunden wurden und die Anzahl der Testworte, bei denen keine Synonyme gefunden wurden, sowie die Anzahl der Worte, die nicht im jeweiligen Modell enthalten sind, abgebildet.

Tabelle 5.1: Experiment 1: Synonyme durch Rekursion

Modell	**Gesamtanzahl**	**Synonyme nicht enthalten**	**Synonyme enthalten**	**Worte nicht im Korpus**
Gesamt	108	96	12	0
Technologie	108	101	2	5

[3]Über http://thesaurus.altervista.org/, abgerufen am 20.07.2015

Tabelle 5.2: Alle gefundenen Synonyme durch Rekursion

Suchbegriff	**Modell**	**Synonym**
engineering	Technologie	discipline
power	Technologie	supply
auto	Gesamt	automobile
automobile	Gesamt	car
cyberwar	Gesamt	cyberterrorism
encryption	Gesamt	cryptography
google	Gesamt	search
linux	Gesamt	unix
mouse	Gesamt	rodent
processor	Gesamt	cpu
sun	Gesamt	star
telecom	Gesamt	telecommunication
television	Gesamt	tv
wifi	Gesamt	wlan

In der Tabelle 5.2 sind alle gefundenen Synonyme und die dazugehörigen Testworte, sowie der Korpus in dem sie auftreten, aufgelistet.

Interpretation/Ergebnis

Wie aus den Daten hervorgeht, werden im Gesamtmodell nur bei zwölf, im Technologiemodell nur bei zwei Testbegriffen Synonyme gefunden. Ein Grund, der die geringe Anzahl an Synonymen erklären könnte, könnte mit der räumlichen Darstellung der Worte im Word2Vec Modell zusammenhängen. Die ähnlichen Worte (W_1) eines Suchbegriffs (O) haben einen minimalen räumlichen Abstand zum Suchbegriff (O). Die durch die Rekursion erhaltenen ähnlichen Worte (W_2) der ähnlichen Worte (W_1), haben wieder einen minimalen räumlichen Abstand (zu W_1), allerdings ist der Abstand vom ursprünglichen Suchbegriff (O) zu den in W_2 enthaltenen Worten größer als zu den Worten aus W_1, denn sonst wären sie schon in der Menge W_1 enthalten. Also sind diese Worte noch weiter vom ursprünglichen Suchbegriffs (O) entfernt und somit unähnlicher. Somit ist das Finden von Synonymen durch Rekursion im Word2Vec Modell nicht zielführend.
Des Weiteren kann das Ergebnis mit den verwendeten Synonymen zusammen hängen. Werden ausführlichere Listen mit Synonymen verwendet, sollte sich damit auch die Anzahl der vom Modell gefundenen Synonyme erhöhen.

5.2 Konkretisierungen

Beschreibung

In diesem Experiment soll hauptsächlich das domänenspezifischen Technologiemodell untersucht werden. Es wird untersucht, ob die ähnlichen Worte der Testdaten eine Konkretisierung des jeweiligen Suchbegriffs darstellen. Dies ist gerade bei mehrdeutigen Worten interessant, da dann eine Sinnrichtung in den ähnlichen Worten dominant vertreten sein könnte und somit klar ist, in welchem Kontext das mehrdeutige Wort in dieser Domäne gemeint ist. Es wird vermutet, dass im Technologiemodell die Konkretisierungen stärker vorhanden sind als im allgemeinen Modell. In diesem Experiment werden wieder alle 236 Testdaten verwendet.

Es wird definiert, dass von einer Konkretisierung gesprochen werden kann, wenn das ähnliche Wort eine genauere Bedeutung des Suchbegriffs darstellt (z.B. wird *Wissenschaftler* durch *Physiker* konkretisiert). In der Abbildung 5.1 sind die Abstraktionsbeziehungen zwischen Worten als Baumdiagramm dargestellt. Gelb markiert ist der aktuelle Suchbegriff, alle Kindknoten des Suchbegriffs sind grün markiert, hier kann von einer Konkretisierung des Suchbegriffs gesprochen werden. Rot markiert sind alle anderen Knoten, hier kann nicht von einer Konkretisierung gesprochen werden. Zur vereinfachten Darstellung ist die Abbildung in 2D und mit einer einfachen Struktur ausgewählt, in komplizierteren Beispielen, unter anderem auch mit mehrdeutigen Begriffen, muss überlegt werden ob eine solche Darstellung überhaupt noch möglich ist.

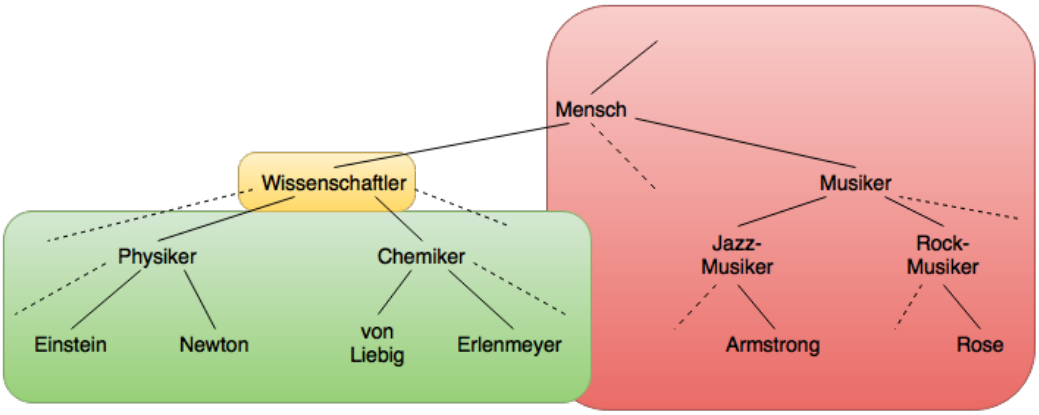

Abbildung 5.1: Beispiel der Beziehungen zwischen Worten, im Bezug auf Konkretisierung, als Baumdiagramm dargestellt.

Durchführung

Die Fragestellung bezieht sich zwar eigentlich auf das domänenspezifische Modell, allerdings wurde auch das allgemeine Modell auf diese Fragestellung hin untersucht um festzustellen, ob dieses tatsächlich ein schlechteres Verhalten bezüglich der Anzahl an gefundenen Konkretisierungen aufweist.
Bei diesem Experiment werden die Testworte mit ihren fünf ähnlichsten Worten verglichen. Diese Untersuchung erfolgt von Hand, da es keine Auflistungen mit Konkretisierungen von Begriffen gibt. Außerdem wäre es zu aufwändig, eine solche Liste zu erstellen, da nicht vorhersehbar ist, wie stark die Suchbegriffe konkretisiert werden und somit sehr viele Daten erzeugt werden müssten. Sollten die Modelle hinsichtlich der Konkretisierungstiefe[4] untersucht werden, könnte das in einem zusätzlichen Experiment analysiert werden. Dann müssten solche Auflistungen mit Konkretisierungen von Begriffen erstellt werden, um das unterschiedliche Verhalten aufzuzeigen. In diesem Experiment geht es aber darum, erst einmal herauszufinden, ob überhaupt eine Konkretisierung stattfindet, unabhängig von der Konkretisierungstiefe.
Stellen mindestens drei der fünf ähnlichen Worte eine Konkretisierung des Suchbegriffs dar, so wird dieser Eintrag als *konkretisiert* markiert. Bei diesem Experiment war das Nachschlagen von mehreren ähnlichen Worten sehr zeitintensiv, denn um beurteilen zu können, ob es sich bei diesem Wort um eine Konkretisierung des Suchbegriffs handelt, müssen die inhaltlichen Beziehungen zwischen Suchbegriff und ähnlichen Worten klar sein. Um die Bedeutungen der Worte und die inhaltlichen Beziehungen zu recherchieren, wurde Wikipedia[5] und Google[6] benutzt.

In der folgenden Tabelle sind die Gesamtanzahl der Testworte, die Anzahl der Suchbegriffe, die mit *konkretisiert* markiert wurden, die Anzahl der Suchbegriffe, die nicht mit *konkretisiert* markiert wurden, sowie die Anzahl der nicht im Modell enthaltenen Worte, dargestellt. Die Daten sind nach Modellen getrennt aufgelistet. Des Weiteren sind einige Beispiele in Tabelle 5.4 aufgelistet. Diese Tabelle enthält neben den Suchbegriffen, Modell und den gefundenen konkretisierten ähnlichen Worten auch die Kosinusähnlichkeiten zu den Suchbegriffen.

[4]Damit ist gemeint ob z.B. der Begriff *Auto* mit *BMW, Mercedes, VW* oder aber mit *X5, A-Klasse, Tuareg* konkretisiert wird.
[5]https://en.wikipedia.org/, abgerufen am 21.07.2015
[6]https://www.google.de/, abgerufen am 21.07.2015

Tabelle 5.3: Experiment 2: Konkretisierung

Modell	**Gesamtzahl**	**ähnliche Worte nicht konkretisiert**	**ähnliche Worte konkretisiert**	**Wort nicht im Korpus**
Gesamt	236	199	37	0
Technologie	236	146	65	25

Tabelle 5.4: Beispiele zur Konkretisierung

Suchbegriff	**Modell**	**Konkretisierung**
apple	Technologie	iphone (Kosinusähnlichkeit: 0,587), a6x (0,579) ipad (0,570), a5x (0,559)
architecture	Gesamt Technologie	revival (0,774), neoclassical (0,677), gothic (0,667) armv7a (0,514), multicore (0,493), xmos (0,477)
fat	Technologie	exfat (0,751), vfat (0,724), fat32 (0,709)
intel	Technologie	i7 (0,707), t2300 (0,702), i5 (0,666), i52450m (0,663)
photography	Technologie	slrs(0,544), panoramic (0,543), polaroid (0,531)
power	Gesamt	electricity (0,679), surgut2 (0,637), egbin (0,627)

Interpretation/Ergebnis

Im Technologiemodell konkretisieren in 65 der 236 Testfälle die ähnlichen Worte den jeweiligen Suchbegriff, im Gesamtmodell jedoch nur in 37 Fällen. Somit bestätigt sich die Annahme, dass im Gesamtmodell weniger Konkretisierungen auftreten. Dies hat den Grund, dass sich die Daten im Gesamtmodell nicht auf eine Fachrichtung beschränken, wie dies im Technologiemodell der Fall ist. So werden beim Training mit Technologiedaten im Word2Vec Modell die Beziehungen beibehalten, die beim Training mit dem kompletten Wikipediakorpus durch andere, allgemeine Worte noch weiter verändert werden.
Da die Anzahl der Testfälle, bei denen die ähnlichen Worte den Testbegriff konkretisieren, nicht so hoch ist, kann daraus geschlossen werden, dass diese Begriffe und die Worte, die sie konkretisieren, nicht so häufig im gleichen Kontext in den Trainingsdaten vorhanden waren.

5.3 Verallgemeinerungen

Beschreibung

In diesem Experiment wird analysiert, ob die ähnlichen Worte eines Begriffs eine Verallgemeinerung dessen darstellen. Es wird außerdem untersucht, ob im allgemeinen Gesamtmodell mehr Begriffe in ihren ähnlichen Worten verallgemeinert werden, als im Technologiemodell. Ein Anwendungsbeispiel für die Verallgemeinerung der Suchbegriffe könnte ein Lexikon sein, bei dem die Worte mit ähnlichen Begriffen beschrieben werden. In diesem Experiment werden wieder alle 236 Testdaten verwendet.
Es wird definiert, dass von einer Verallgemeinerung gesprochen werden kann, wenn das ähnliche Wort eine abstraktere Bedeutung des Suchbegriffs darstellt (z.B. wird *Physiker* durch *Mensch* verallgemeinert) oder das ähnliche Wort bildet einen vergleichbaren Begriff (kein Synonym) ab (z.b. kann *Physiker* mit *Chemiker* verallgemeinert werden). In der Abbildung 5.2 sind die Abstraktionsbeziehungen zwischen Worten als Baumdiagramm dargestellt. Gelb markiert ist der aktuelle Suchbegriff, Elternknoten und direkte Geschwisterknoten sind grün markiert, hier kann von einer Verallgemeinerung des Suchbegriffs gesprochen werden. Rot markiert sind die Kindknoten und weiter entfernte Knoten, hier kann nicht von einer Verallgemeinerung gesprochen werden. Zur vereinfachten Darstellung ist die Abbildung in 2D und mit einer einfachen Struktur ausgewählt, in komplizierteren Beispielen, unter anderem auch mit mehrdeutigen Begriffen, muss überlegt werden ob eine solche Darstellung überhaupt noch möglich ist.

Durchführung

In diesem Experiment werden die Testdaten, wie im vorherigen Experiment, mit ihren fünf ähnlichsten Worten verglichen. Auch diese Untersuchung wird von Hand durchgeführt, denn auch hier soll, wie im vorherigen Experiment, erst einmal untersucht werden, ob Verallgemeinerungen überhaupt auftreten. In einem weiteren Experiment könnte auch hier untersucht werden, welchen Grad die Verallgemeinerungen haben[7]. Verallgemeinern drei der fünf ähnlichen Worte den Suchbegriff, wird dieser als *verallgemeinert* markiert.

Wie beim vorherigen Experiment mussten viele Begriffe nachgeschlagen werden, um die inhaltlichen Beziehungen zwischen dem Suchbegriff und den jeweiligen ähn-

[7] Damit ist gemeint ob *Albert* mit *Einstein*, *Physiker*, *Wissenschaftler* oder *Mensch* verallgemeinert wird.

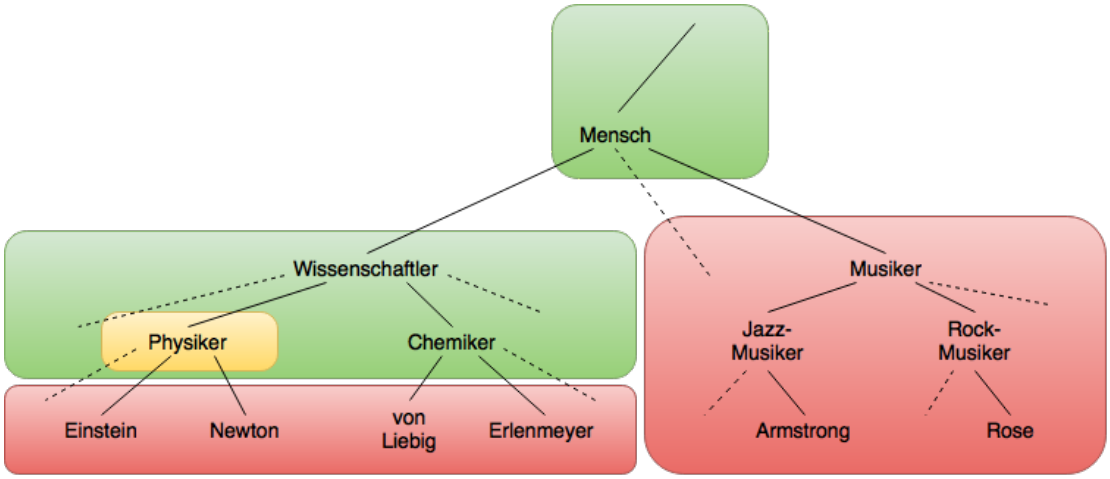

Abbildung 5.2: Beispiel der Beziehungen zwischen Worten, im Bezug auf Verallgemeinerung, als Baumdiagramm dargestellt.

lichen Worten beurteilen zu können.

In Tabelle 5.5 sind die Gesamtzahl der Testworte, die Anzahl der Suchbegriffe, die mit *verallgemeinert* markiert wurden, die Anzahl der Suchbegriffe, die nicht mit *verallgemeinert* markiert wurden, sowie die Anzahl der Worte, die nicht im Modell enthalten sind, aufgelistet. Die Daten sind nach Modellen getrennt aufgelistet. Die Tabelle 5.6 enthält einige Beispiele der gefundenen Verallgemeinerungen. Es sind neben den Suchbegriffen, Modell und den gefundenen verallgemeinerten ähnlichen Worten, auch deren Kosinusähnlichkeiten zu den Suchbegriffen dargestellt.

Tabelle 5.5: Experiment 3: Verallgemeinerung

Modell	**Gesamtzahl**	**ähnliche Worte nicht verallgemeinert**	**ähnliche Worte verallgemeinert**	**Wort nicht im Korpus**
Gesamt	236	102	134	0
Technologie	236	144	67	25

Tabelle 5.6: Beispiele zur Verallgemeinerung

Suchbegriff	**Modell**	**Verallgemeinerung**
apple	Gesamt	blackberry (Kosinusähnlichkeit: 0,704), raspberry (0,657), webos (0,643)
asus	Gesamt	netbook (0,767), lenovo (0,736), motherboards (0,714)
cookies	Gesamt	baked (0,666), cakes (0,663), muffins (0,654)
dell	Gesamt	hewlettpackard (0,583), cisco (0,548), lenovo (0,515), compaq (0,506)
limewire	Gesamt	bittorrent (0,703), filesharing (0,702), kazaa (0,689) rapidshare (0,668), gnutella (0,653)
photoshop	Technologie	illustrator (0,781), lightroom (0,758), cs3 (0,710)

Interpretation/Ergebnis

Im Gesamtmodell weisen bei 134 Testbegriffen mindestens drei der dazugehörigen ähnlichen Worte eine Verallgemeinerung des Testwortes auf. Im Technologiemodell ist dies nur bei 67 Begriffen der Fall. Somit bestätigt sich die Vermutung, dass im Gesamtmodell mehr Verallgemeinerungen zu finden sind. Dies hängt mit der Tatsache zusammen, dass das Gesamtmodell mit Daten, die nicht auf ein spezielles Thema beschränkt sind, trainiert wurde und diese Begriffe in vielen unterschiedlichen Zusammenhängen im Trainingskorpus enthalten sind. Die allgemeineren ähnlichen Worte müssen in den Trainingsdaten also häufiger im gleichen Kontext wie die Testbegriffe vorgekommen sein, als beispielsweise Worte, die den Testbegriff konkretisieren.

5.4 Unterschiedliche Beziehungen

Beschreibung

Die inhaltlichen Beziehungen zwischen Worten können von unterschiedlicher Art sein. So können die Worte ***gleiches*** darstellen, beispielsweise stellen sowohl „dropbox“, als auch „icloud“ oder „onedrive“ Cloud-Speicher dar. Außerdem können Worte auch ***gegenteiliges*** darstellen, wie z.B. „heiß“ und „kalt“. ***Syntaktisch*** ist eine weitere Beziehungsart, wie bei „Verschlüsselung“ und „verschlüsseln“. „Automobil“ und „KFZ“ haben eine ***synonyme*** Beziehung untereinander. Trifft keine der bisher genannten Arten auf die Beziehung zwischen zwei Worten, diese haben aber trotzdem etwas miteinander zu tun, wie beispielsweise „Motor“ und „Benzin“, dann

stehen diese Begriffe ***in Beziehung*** miteinander. Diese fünf Beziehungsarten sollen als Kategorien für dieses Experiment dienen.
Ziel dieses Experiments ist festzuhalten, ob in den beiden Modellen die Trainingsbegriffe zu ihren ähnlichen Worten unterschiedliche Arten von Beziehungen haben. Die Auswertung der Testbegriffe und ihrer ähnlichen Worte erfolgt hier wieder von Hand. Sobald eine Beziehungsart in mindestens drei der fünf ähnlichen Worte abgebildet ist, wird dieser Testbegriff mit der jeweiligen Beziehungsart markiert. Erreicht keine Art drei Vorkommen, wird dieser Begriff mit ***verschiedenes*** markiert, da hier eine einzige Art nicht eindeutig zuordenbar ist. Somit ergeben sich die sechs endgültigen, unterschiedlichen Beziehungskategorien *syntaktisch, synonym, gleiches, gegenteiliges, in Beziehung* und *verschiedenes*. Es werden alle 236 Testbegriffen für dieses Experiment verwendet.

Durchführung

Für dieses Experiment werden beide Modelle untersucht, um eine Aussage zu treffen, ob sich die Beziehungsarten unterscheiden. Auch hier wird von Hand die Art, die die Beziehung zwischen den einzelnen Testbegriffen und den dazugehörigen ähnlichen Worten beschreibt, dem jeweiligen Testbegriff zugeordnet.
In der Tabelle 5.7 werden die Beziehungsarten und die Anzahl der Testbegriffe, die mit der jeweiligen Art markiert sind, nach Modelltyp getrennt, dargestellt. In der darauffolgenden Tabelle 5.8 sind Beispiele zu den einzelnen Beziehungsarten aufgelistet. Neben der Art der Beziehungen ist der Suchbegriff, Modelltyp und die ähnlichen Worte mit den dazugehörigen Kosinusähnlichkeiten abgebildet.

Tabelle 5.7: Experiment 4: Arten von Beziehungen

Beziehungsart	**Gesamtmodell**	**Technologiemodell**
syntaktisch	0	0
gegenteilig	0	0
gleiches	81	38
synonym	0	0
in Beziehung	118	126
verschiedenes	37	47
nicht im Modell enthaltene Worte	0	25

Tabelle 5.8: Beispiele zur den verschiedenen Beziehungsarten

Art	**Suchbegriff**	**Modell**	**Ähnliche Worte**
gleiches	chrome	Gesamt Technologie	firefox (0,784), safari (0,686), ie8 (0,674) firefox (0,733), safari (0,769), flock (0,667), seamonkey (0,650)
in Beziehung	browser	Gesamt	browsers (0,882), firefox (0,812), desktop (0,755), javascript (0,751)
verschiedenes	encryption	Gesamt	encrypted (0,833), authentication(0,815), encrypts (0,798), decryption (0,789)

Interpretation/Ergebnis

Wie aus Tabelle 5.7 ersichtlich ist, wurden keine Testbegriffe mit *synonym*, *syntaktisch* oder *gegenteilig* markiert. Das hängt damit zusammen, dass die Bedingung für die Markierung war, dass diese Beziehungsart in mindestens drei der fünf ähnlichen Worte vorhanden sein muss. Zwar traten diese Arten der Beziehung vereinzelt auf, aber die Bedingung konnten diese drei Arten bei keinem Testbegriff erfüllen. Meist fielen diese Begriffe in die Kategorie *verschiedenes*, da auch noch andere Arten vorhanden waren, aber keine dominierte. Mit einer Anzahl von 118 im Gesamtmodell und 126 im Technologiemodell war die Beziehungsart *in Beziehung* am häufigsten vorhanden. Dies kann damit erklärt werden, dass im Word2Vec Modell Worte die in ähnlichem Kontext stehen, auch nahe in der Vektorrepräsentation zusammen stehen. Die Art *gleiches* war mit 81 Markierungen im Gesamtmodell am zweithäufigsten vorhanden, auch hier liegt nahe, dass es daran liegt, dass Worte mit gleichem Kontext eine ähnliche Vektorrepräsentation haben. Im Technologiemodell tritt diese Beziehungsart nur 38 Mal auf, was daher kommen könnte, dass dieses Modell mit weniger Daten trainiert wurde und deswegen die Anzahl an Worte, die im gleichen Kontext stehen, auch geringer ist. Im Gesamtmodell steht die Art *verschiedenes* mit 37 Vorkommen an dritter Stelle, beim Technologiemodell mit 47 an zweiter. Diese Art sagt nichts über die tatsächlichen Beziehungen aus, außer dass nicht eindeutig eine Art dominant vorhanden ist.
In beiden Modellen sind die gefundenen Arten die gleichen. Das Gesamtmodell stellt verhältnismäßig mehr *gleiches* als das Technologiemodell dar, das Technologiemodell jedoch mehr *verschiedenes* und *in Beziehung*. Der Unterschied in der Anzahl der vorkommenden Beziehungsarten ist erkennbar, allerdings ist er in den unterschiedlichen Modellen nicht sehr groß.

5.5 Erkennen von Mehrdeutigkeit

Beschreibung

Das letzte Experiment beschäftigt sich mit der Mehrdeutigkeit von Worten. Es wird untersucht, ob im Gesamtmodell die ähnlichen Worte eines mehrdeutigen Suchbegriffs, dessen unterschiedliche Bedeutungen repräsentieren. Oder ob dieses Verhalten häufiger im Technologiemodell aufzufinden ist. Hier werden nur die 66 mehrdeutigen der insgesamt 236 Testbegriffen verwendet[8].

Durchführung

Wie in den vorherigen Experimenten, werden in diesem Experiment die Testbegriffe und ihre fünf ähnlichen Worte von Hand analysiert. Sobald zwei der ähnlichen Worte unterschiedliche Bedeutungen eines mehrdeutigen Testbegriffes darstellen, wird dieser Begriff als *Mehrdeutigkeit erkannt* markiert. Sollte dies nicht der Fall sein, werden sie mit *Mehrdeutigkeit nicht erkannt* markiert. Es werden beide Modelle untersucht.
In Tabelle 5.9 werden die Anzahl der mehrdeutigen Testbegriffe, die Anzahl der nicht mehrdeutigen Testbegriffe und wie viele der Testbegriffe jeweils nicht im Modell vorhanden sind, nach Modelltyp getrennt, aufgelistet. Die Tabelle 5.10 bezieht sich komplett auf die mehrdeutigen Testbegriffe. Hier sind die Anzahl der Testbegriffe, die mit *Mehrdeutigkeit erkannt*, sowie mit *Mehrdeutigkeit nicht erkannt* markiert sind und die Anzahl der mehrdeutigen Testworte, die nicht im Modell enthalten sind, dargestellt. Einige Beispiele der erkannten und nicht erkannten Mehrdeutigkeiten sind in Tabelle 5.11 aufgelistet. Neben dem Suchbegriff, Modelltyp ist auch angegeben, ob die Begriffe als mehrdeutig erkannt wurden und es sind die jeweiligen ähnlichen Worte mit ihren Kosinusähnlichkeiten aufgeführt.

Tabelle 5.9: Experiment 5: Aufteilung mehrdeutiger Worte

Worte	**Gesamtmodell**	**Technologiemodell**
nicht mehrdeutig	170	170
davon nicht im Korpus	0	24
mehrdeutig	66	66
davon nicht im Korpus	0	1

[8]Siehe Auflistung der mehrdeutigen Testworte unter 7.4 .

Tabelle 5.10: Experiment 5: Erkennen von mehrdeutigen Worten

Mehrdeutigkeit	**Gesamtmodell**	**Technologiemodell**
nicht erkannt	59	58
erkannt	7	7
Worte nicht im Korpus	0	1

Tabelle 5.11: Beispiele zur Erkennung von Mehrdeutigkeiten

Suchbegriff	**Modell**	**Erkannt**	**Ähnliche Worte**
trojan	Gesamt	ja	rsplug (Kosinusähnlichkeit: 0,472), athena(0,462),bundestrojaner (0,496), centaurs (0,475)
	Technologie	ja	horse (0,586), malware (0,560), zinaps (0,476)
raspberry	Gesamt	nein	apple (0,657), apricot (0,629), blackcurrant (0,586),
	Technologie	nein	pi (0,677), cubieboard (0,524), trimeslice (0,519)

Interpretation/Ergebnis

Im beiden Modellen werden von den 66 mehrdeutigen Testbegriffen nur sieben erkannt. Jedoch sind dies 13 unterschiedliche Testbegriffe, denn die Modelle erkennen immer bei unterschiedlichen Begriffen die Mehrdeutigkeit und nur beim Begriff *trojan* erkennen beide Modelle die Mehrdeutigkeit. Ein mehrdeutiger Testbegriff kommt im Technologiemodell gar nicht vor und 59 im Gesamtmodell beziehungsweise 58 im Technologiemodell mehrdeutige Begriffe werden nicht erkannt. Die sehr geringe Anzahl an erkannten Mehrdeutigkeiten kann auch damit zusammen hängen, dass diese Begriffe in den unterschiedlichen Bedeutungen oft sehr unterschiedliche Dinge beschreiben und somit eher selten im gleichen Kontext in den Trainingsdaten vorkommen. Und da Word2Vec die Ähnlichkeiten über den gleichen Kontext bildet, kann die Mehrdeutigkeit schlecht damit abgebildet werden.
Allerdings werden bei 35 Testbegriffen in den unterschiedlichen Modellen je eine unterschiedliche Bedeutung der mehrdeutigen Begriffe in den ähnlichen Worten dargestellt. Siehe dazu Tabelle 5.12, dort sind einige Beispiele aufgelistet. Dargestellt sind die Testbegriffe, Modelltyp und die erhaltenen ähnlichen Worte mit Kosinusähnlichkeiten.

Tabelle 5.12: Beispiele Mehrdeutigkeiten in unterschiedlichen Modellen

Suchbegriff	**Modell**	**Ähnliche Worte**
architecture	Gesamt	revival (Kosinusähnlichkeit: 0,774), neoclassical (0,677), gothic (0,666),
	Technologie	armv7a (0,514), multicore (0,494), xmos (0,477)
fat	Gesamt	calories (0,615), milk (0,598), fatfree (0,595)
	Technologie	ntfs (0,780), exfat(0,751), fat32 (0,709)
keyboard	Gesamt	synthesizer (0,769), accordion (0,768), guitar (0,752)
	Technologie	qwerty (0,672), mouse (0,667), keypad(0,641)
pi	Gesamt	theta (0,754), tau (0,729), rho (0,729), mu (0,695)
	Technologie	raspberry (0,677), cubieboard (0,478), iyonix (0,455)
ps2	Gesamt	ps3 (0,795), gamecube(0,764), n64 (0,756), snes (0,748)
	Technologie	keyboardmouse (0,764), centronics (0,676), rs232 (0,664)
windows	Gesamt	sunhoods (0,745), sixoverone (0,743), nineovernine (0,736), sixoversix (0,721)
	Technologie	vistawindows (0,699), vista (0,683), xp (0,656)

Aus der Tatsache, dass nur eine so geringe Menge von sieben Begriffen als mehrdeutig erkannt werden, kann geschlossen werden, dass sich beide Modelle nicht dazu eignen, Mehrdeutigkeiten in den ähnlichen Worten abzubilden. Im folgenden Kapitel wird unter der Rubrik Ausblick ein Vorschlag gemacht, wie eine Mehrdeutigkeit von Begriffen mit mehreren Word2Vec Modellen erkannt werden könnte.

Kapitel 6

Fazit und Ausblick

6.1 Fazit

Nach einer ausführlichen Vorverarbeitung können die Artikel aus Wikipedia gut als Trainingsdaten für Word2Vec Modelle dienen. Dank der Gensim-Implementierung ist die Benutzung von Word2Vec sehr vereinfacht und kann komfortabel bedient werden.
Die Experimente liefern interessante Resultate im Bezug auf die Problemstellung dieser Arbeit. Durch die Experimente können die unterschiedlichen semantischen Beziehungen etwas aufgeschlüsselt werden. So wird durch die Experimente deutlich, dass durch einmalige Rekursion der Suchbegriffe, nicht auf Synonyme des ursprünglichen Begriffs geschlossen werden kann. Des Weiteren eignet sich der domänenspezifische Teilkorpus eher für eine Konkretisierung der Suchbegriffe, wobei dies nur in ca. einem Drittel der Testdaten nachgewiesen werden konnte.
Sollen die Testdaten verallgemeinert werden, eignet sich der allgemeine komplette Korpus viel besser als der domänenspezifische Teilkorpus. Soll also ein Überblick über ein gewisses Thema oder Suchbegriff erhalten werden, wird empfohlen das Modell, welches auf dem allgemeinen kompletten Korpus trainiert wurde, zu verwenden.
Die unterschiedlichen Arten von Beziehung zwischen den Testdaten und ihren ähnlichen Worten unterscheiden sich nicht grundlegend zwischen den beiden Modellen. Das allgemeine komplette Modell hat einen etwas Größeren Anteil an *gleichen* Worten, hingegen hat das domänenspezifische Teilmodell einen etwas größeren Anteil an *verschiedenen* und *in Beziehung* stehenden Beziehungen. *Synonyme*, *syntaktische* oder *gegenteilige* Arten an Beziehungen sind in beiden Modellen nicht vorhanden.
Soll auf Mehrdeutigkeit untersucht werden, ist die Vorgehensweise nur die ähnlichen Worte zu vergleichen und zu analysieren, nicht zielführend, da die Modelle die Mehrdeutigkeit in den ähnlichen Worten nicht gut abbilden. Im Ausblick wird ein

Vorschlag gemacht, wie die Mehrdeutigkeit eventuell besser erfasst werden könnte.

Es kann also festgehalten werden, dass sich das allgemeine komplette Modell dann besser eignet, wenn ein genereller Überblick über die Testdaten erhalten werden soll. Denn in diesem Modell werden die Suchbegriffe eher verallgemeinert und die ähnlichen Worte repräsentieren *Gleiches* im Bezug zu den Testdaten.
Soll aber ein Suchbegriff genauer untersucht, bzw. sehr fokussiert betrachtet werden, eignet sich ein domänenspezifisches Modell besser. Dieses konkretisiert die Testdaten nicht nur besser, sondern liefert auch fachspezifischere ähnliche Worte.

Die Qualität der Beziehungen hängt zudem stark von den verwendeten Trainingsdaten ab. Es sollte gewährleistet sein, dass ausreichend viele und auch qualitativ gute Daten zum Training verwendet werden.
Soll ein domänenspezifisches Modell erstellt werden, sollte hier auch darauf geachtet werden, dass nicht nur ein Teil dieser Domäne in den Trainingsdaten abgebildet ist, außer es soll nur dieser spezielle Teil der Domäne im Modell dargestellt werden.

6.2 Ausblick

In dieser Arbeit wurden Beziehungen zwischen den Testdaten und ihren ähnlichen Worten analysiert. Es wurden jedoch bei weitem nicht alle Beziehungen untersucht, weshalb der Fokus weiterführender Arbeiten andersartige Beziehungen zwischen den Worten sein könnten. Dies bedeutet, dass noch mehrere Experimente durchgeführt werden müssten. Zum Beispiel könnte die Mehrdeutigkeit noch anders untersucht werden, indem man mehrere unterschiedliche Modelle hat und dann die ähnlichen Worte eines gleichen Testwortes vergleicht und sollten sich diese ähnlichen Worte unterscheiden, könnte mit hoher Wahrscheinlichkeit von einer Mehrdeutigkeit gesprochen werden.
Ein weiterer Ansatzpunk für zukünftige Arbeiten ist, wie sich subsampling[1] im Word2Vec Modell, im Blick auf die in dieser Arbeit analysierten Fragestellungen, verhält. Es wäre auch möglich das Word2Vec Modell mit N-Grammen, auch Phrases genannt, zu lernen, dann könnten auch Mehrwortbegriffe abgebildet und gesucht werden. Allerdings erhöhen N-Gramme die Trainingszeiten und den benötigten Speicher sehr[MSC+13], weshalb sie nicht in dieser Arbeit verwendet wurden.

[1] Beim subsampling wird das Ungleichgewicht von sehr häufig und sehr selten vorkommenden Worten im Trainingskorpus verringert[MSC+13].

Literaturverzeichnis

[BDVJ03] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155, 2003.

[DDL+90] Scott C. Deerwester, Susan T Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *JAsIs*, 41(6):391–407, 1990.

[Elm90] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[GH12] Michael U Gutmann and Aapo Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *The Journal of Machine Learning Research*, 13(1):307–361, 2012.

[HMR86] G. E. Hinton, J. L. McClelland, and D. E. Rumelhart. Distributed representations. *Parallel distributed processing: Explorations in the microstructure of cognition. Volume 1: Foundations*, pages 503–527, 1986.

[MB05] Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In *Proceedings of the international workshop on artificial intelligence and statistics*, pages 246–252. Citeseer, 2005.

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

[Mik07] Tomas Mikolov. Language Modeling for Speech Recognition in Czech. Master's thesis, Brno University of Technology, 2007.

[MKB+09] Tomáš Mikolov, Jiří Kopeckỳ, Lukáš Burget, Ondřej Glembek, and Jan Honza Černockỳ. Neural network based language models for highly

inflective languages. In *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, pages 4725–4728. IEEE, 2009.

[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.

[MT12] Andriy Mnih and Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. *arXiv preprint arXiv:1206.6426*, 2012.

[MYZ13] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 9-14, 2013, Westin Peachtree Plaza Hotel, Atlanta, Georgia, USA*, pages 746–751, 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.

[WH86] DE Rumelhart GE Hinton RJ Williams and GE Hinton. Learning representations by back-propagating errors. *Nature*, pages 323–533, 1986.

Tabellenverzeichnis

Abbildungsverzeichnis

Kapitel 7

Anhang

7.1 Testdatenvergleich

Auflistung der Testdaten, die zum Vergleich der Testdomänen benutzt wurden.

Tabelle 7.1: Testdaten aus der Domäne Mode/Fashion

apron	babydoll	bandana	belt
beret	bermuda	bikini	blazer
blouse	boot	boxer	cap
cargo	chaps	chesterfield	coat
cufflink	cummerbund	fashion	fedora
gaiters	handbag	handkerchief	hat
headband	helmet	hood	hoodie
jacket	jersey	jewellery	muff
necktie	negligee	pajamas	pantsuit
sash	shawl	shirt	shrug
sleeveless	slip	sweater	trunks
turban	umbrella	undershirt	veil
wallet	wetsuit		

Tabelle 7.2: Testdaten aus der Domäne Medizin

allergy	anatomy	arthritis	aseptic
cardiology	cell	clinic	cornea
dental	disability	disease	ect
excision	finger	fistula	gynaecology
hemostat	hernia	hospital	immunology
iris	knee	limb	lip
mammography	medicine	mouth	mri
nephrology	oncology	otoplasty	pain
prolapse	prosthesis	radiology	radiotherapy
retina	scalpel	scapula	skeleton
skull	sterile	surgery	symptom
tomography	transplant	ultrasound	urology
wrist	xray		

Tabelle 7.3: Testdaten aus der Domäne Technologie/Pc/Internet

acer	amazon	apple	arcade
arpanet	biometrics	blogging	chrome
cookies	cyberwar	dell	drone
dropbox	events	filesharing	firefox
foursquare	gameplay	groupon	hacking
instagram	lenovo	macintosh	malware
microsoft	mobile	motoring	mouse
myspace	paypal	pc	photography
playstation	processor	ram	raspberry
samsung	software	sony	starcraft
steam	surface	tablet	tetris
tumblr	twitter	warcraft	windows
yahoo	youtube		

7.2 Testdaten

Tabelle 7.4: Vollständige 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
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

Tabelle 7.5: Vollständige Testdaten Teil 2

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

7.3 Reinigungsskript

Das Originalskript von Matt Mahoney kann unter
http://mattmahoney.net/dc/textdata.html gefunden werden.
Geändertes Skript:

```
#!/usr/bin/perl

# Program to filter Wikipedia XML dumps to "clean" text consisting
# only of lowercase letters (a-z, converted from A-Z), and spaces
# (never consecutive).
# All other characters are converted to spaces.  Only text which
# normally appears in the web browser is displayed.  Tables are removed.
# Image captions are
# preserved.  Links are converted to normal text.
# Digits are spelled out.

# Written by Matt Mahoney, June 10, 2006.  This program is
# released to the public domain.

$/=">";                     # input record separator
while (<>) {
  if (/<text /) {$text=1;}  # remove all but between <text> ... </text>
  if (/#redirect/i) {$text=0;}  # remove #REDIRECT
  if ($text) {

    # Remove any text not normally visible
    if (/<\/text>/) {$text=0;}
    s/<.*>//;               # remove xml tags
    s/&amp;/&/g;            # decode URL encoded chars
    s/&lt;/</g;
    s/&gt;/>/g;
    s/<ref[^<]*<\/ref>//g;  # remove references <ref...> ... </ref>
    s/<[^>]*>//g;           # remove xhtml tags
    s/\[http:[^] ]*/[/g;    # remove normal url, preserve visible text
    s/\|thumb//ig;          # remove images links, preserve caption
    s/\|left//ig;
```

```
    s/\|right//ig;
    s/\|\d+px//ig;
    s/\[\[image:[^\[\]]*\|//ig;
    # show categories without markup
    s/\[\[category:([^|\]]*)[^]]*\]\]/[[$1]]/ig;
    s/\[\[[a-z\-]*:[^\]]*\]\]//g;  # remove links to other languages
    s/\[\[[^\|\]]*\|/[[/g;  # remove wiki url, preserve visible text
    s/{{[^}]*}}//g;         # remove {{icons}} and {tables}
    s/{[^}]*}//g;
    s/\[//g;                # remove [ and ]
    s/\]//g;
    s/&[^;]*;/ /g;          # remove URL encoded chars

    $_=" $_ ";
    ###### begin changed lines ######
    s/Ä/Ae/g;
    s/ä/ae/g;
    s/Ö/Oe/g;
    s/ö/oe/g;
    s/Ü/Ue/g;
    s/ü/ue/g;
    s/ß/ss/g;
    s/-//g;
    #removes everything else than this characters
    tr/0-9A-Za-z,.!?;\r\n / /csd;
    ###### end changed lines ######
    chop;
    print $_;
  }
}
```

7.4 Mehrdeutige Testbegriffe

Aus allen Testbegriffen wurden von Hand diejenigen ausgewählt, die mehrere Bedeutungen haben.

Tabelle 7.6: Vollständige Auflistung der mehrdeutigen Testbegriffe

3ds	acer	acta	amazon
android	anonymous	apple	arcade
architecture	augmented	auto	battlefield
bing	blackberry	blizzard	casual
chrome	cookies	dell	diablo
doodle	dotcom	drone	e3
engine	events	fat	halo
i	ie	indie	keyboard
link	mario	mobile	monitor
mouse	net	oracle	pi
ps	ps2	python	raider
ram	raspberry	rpg	rts
safari	seo	sonic	spam
steam	stream	sun	surface
ted	tomb	trojan	twitch
vine	virus	web	windows
worms	wow		