Karlsruhe Institute of Technology

Worteinbettungen für die Anforderungsdomäne

Bachelorarbeit
von

Tobias Telge

An der Fakultät für Informatik
Institut für Programmstrukturen
und Datenorganisation (IPD)

Erstgutachter:	Prof. Dr. Walter F. Tichy
Zweitgutachter:	Prof. Dr. Ralf H. Reussner
Betreuender Mitarbeiter:	M.Sc. Tobias Hey

Bearbeitungszeit: 25.12.2019 – 24.04.2020

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Die Regeln zur Sicherung guter wissenschaftlicher Praxis im Karlsruher Institut für Technologie (KIT) habe ich befolgt.

Karlsruhe, 24.04.2020

(**Tobias Telge**)

Publikationsgenehmigung

Melder der Publikation

Hildegard Sauer

Institut für Programmstrukturen und Datenorganisation (IPD)
Lehrstuhl für Programmiersysteme
Leiter Prof. Dr. Walter F. Tichy

+49 721 608-43934
hildegard.sauer@kit.edu

Erklärung des Verfassers

Ich räume dem Karlsruher Institut für Technologie (KIT) dauerhaft ein einfaches Nutzungsrecht für die Bereitstellung einer elektronischen Fassung meiner Publikation auf dem zentralen Dokumentenserver des KIT ein.

Ich bin Inhaber aller Rechte an dem Werk; Ansprüche Dritter sind davon nicht berührt.

Bei etwaigen Forderungen Dritter stelle ich das KIT frei.

Eventuelle Mitautoren sind mit diesen Regelungen einverstanden.

Der Betreuer der Arbeit ist mit der Veröffentlichung einverstanden.

Art der Abschlussarbeit:	Bachelorarbeit
Titel:	Worteinbettungen für die Anforderungsdomäne
Datum:	24.04.2020
Name:	Tobias Telge

Karlsruhe, 24.04.2020

(**Tobias Telge**)

Kurzfassung

Worteinbettungen werden in Aufgaben aus der Anforderungsdomäne auf vielfältige Weise eingesetzt. In dieser Arbeit werden Worteinbettungen für die Anforderungsdomäne gebildet und darauf geprüft, ob sie in solchen Aufgaben bessere Ergebnisse als generische Worteinbettungen erzielen. Dafür wird ein Korpus von in der Anforderungsdomäne üblichen Dokumenten aufgebaut. Er umfasst 21 458 Anforderungsbeschreibungen und 1680 Anwendererzählungen. Verschiedene Worteinbettungsmodelle werden auf ihre Eignung für das Training auf dem Korpus analysiert. Mit dem `fastText-Modell`, das durch die Berücksichtigung von Teilwörtern seltene Wörter besser darstellen kann, werden die domänenspezifischen Worteinbettungen gebildet. Sie werden durch Untersuchung von Wortähnlichkeiten und Clusteranalysen intrinsisch evaluiert. Die domänenspezifischen Worteinbettungen erfassen einige domänenspezifische Feinheiten besser, die untersuchten generischen Worteinbettungen hingegen stellen manche Wörter besser dar. Um die Vorteile beider Worteinbettungen zu nutzen, werden verschiedene Kombinationsverfahren analysiert und evaluiert. In einer Aufgabe zur Klassifizierung von Sätzen aus Anforderungsbeschreibungen erzielt eine gewichtete Durchschnittsbildung mit einer Gewichtung von 0,7 zugunsten der generischen Worteinbettungen die besten Ergebnisse. Ihr bester Wert ist eine Genauigkeit von 0,83 mittels eines LSTMs als Klassifikator und der Training-Test-Teilung als Testverfahren. Die domänenspezifischen, bzw. generischen Worteinbettungen liefern dabei hingegen lediglich 0,75, bzw. 0,72.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

In der Verarbeitung natürlicher Sprache (engl. *natural language processing*) spielen Worteinbettungen, die Wörter auf Wortvektoren abbilden, eine immer größere Rolle. Sie werden in vielen verschiedenen Aufgaben wie zum Beispiel der Klassifizierung von Texten eingesetzt. Worteinbettungen können durch das Training eines Worteinbettungsmodells auf einem Textkorpus gebildet werden. Dabei wird der Kontext, in dem Wörter vorkommen, berücksichtigt. So sollen Wörter mit ähnlicher Bedeutung auf ähnliche Wortvektoren abgebildet werden.
Worteinbettungen können auch in Aufgaben aus der Anforderungsdomäne auf vielfältige Weise eingesetzt werden. Sie können zum Beispiel bei der Erkennung von äquivalenten Anforderungen oder bei der Klassifizierung von Anforderungen als funktional oder nichtfunktional helfen. Auch bei der Erkennung von nicht eindeutigen Formulierungen oder von Widersprüchen in Anforderungen können sie verwendet werden. Ein konkretes Beispiel ist das Projekt `INDIRECT` [Hey19]. Hier werden Worteinbettungen zur Bestimmung der semantischen Funktion von Sätzen in Anforderungsbeschreibungen verwendet.

Es gibt verschiedene vortrainierte Worteinbettungsmodelle. Sie werden auf umfangreichen generischen Korpora wie zum Beispiel Google-Nachrichten oder Wikipedia-Artikeln trainiert. In diesen Textkorpora kommen manche Begriffe aus der Anforderungsdomäne selten oder mit einer anderen Bedeutung vor. Daher können domänenspezifische Feinheiten in der Verteilung und Bedeutung von Wörtern von diesen Worteinbettungsmodellen nicht besonders gut erfasst werden. Speziell für die Anforderungsdomäne gebildete Worteinbettungen können dafür besser geeignet sein. Ein Beispiel für eine domänenspezifische Feinheit ist das Wort *Fenster*, das in Anforderungen im Vergleich zu generischen Texten häufiger mit der Bedeutung *Zeitfenster* als mit der Bedeutung *Lichtöffnung* vorkommt. Die Art und Weise, wie sich dieser Bedeutungsunterschied in generischen und domänenspezifischen Worteinbettungen manifestieren kann, ist in Abbildung 1.1 schematisch dargestellt. Es sind jeweils die Worteinbettungen zum Wort *Fenster* und seinen in den Worteinbettungen am ähnlichsten dargestellten Wörtern abgebildet. Aus den ähnlichsten Wörtern lassen sich die unterschiedlichen Bedeutungen von *Fenster* als *Lichtöffnung* oder *Zeitfenster*, die in den unterschiedlichen Worteinbettungen erfasst wurden, ableiten. Infolge der besseren Erfassung domänenspezifischer Eigenheiten können domänenspezifische Worteinbettungen in Aufgaben aus der Anforderungsdomäne bessere Ergebnissen erzielen als existierende vortrainierte Worteinbettungen.

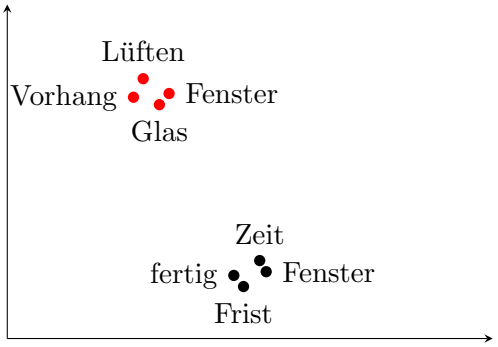

Abbildung 1.1: Schematische Darstellung von beispielhaften domänenspezifischen (schwarz) und generischen (rot) Worteinbettungen im Vektorraum, die das Wort *Fenster* unterschiedlichen erfassen

1.1 Zielsetzung

Diese Arbeit hat das Ziel, Worteinbettungen für die Anforderungsdomäne zu bilden. Es sollen Worteinbettungen entstehen, die auf englischsprachigen Anforderungsdokumenten trainiert wurden. Dazu soll zuerst ein Textkorpus von Anforderungsbeschreibungen und anderen in der Anforderungsdomäne üblichen Dokumenten aufgebaut werden. Dann sollen verschiedene Worteinbettungsmodelle auf ihre Vor- und Nachteile für das Training auf Dokumenten aus der Anforderungsdomäne untersucht und das geeignetste ausgewählt werden. Zudem soll überprüft werden, wie die mit dem ausgewählten Worteinbettungsmodell auf dem aufgebauten Textkorpus trainierten Worteinbettungen in geeigneter Form mit bestehenden auf umfangreichen generischen Korpora trainierten Worteinbettungen kombiniert werden können. Wurde eine Kombinationsmethode gewählt, werden bestehende generische Worteinbettungen auf ihre Eignung zur Kombination untersucht und ausgewählt. Abschließend sollen die auf dem aufgebauten Textkorpus trainierten sowie die kombinierten Worteinbettungen auf ihre Eignung in Aufgaben aus der Anforderungsdomäne evaluiert werden.

1.2 Aufbau

Zuerst werden in Kapitel 2 die Grundlagen, die zum Verständnis der Inhalte dieser Bachelorarbeit erforderlich sind, vorgestellt. Dann wird in Kapitel 3 das Projekt `INDIRECT` eingeführt, in dessen Rahmen diese Bachelorarbeit stattfindet. In Kapitel 4 werden die verwandten Arbeiten vorgestellt. In Kapitel 5 wird zuerst ein Korpus von Dokumenten aus der Anforderungsdomäne gebildet und seine Eigenschaften untersucht. Dann werden verschiedene Worteinbettungsmodelle analysiert und das für das Training auf dem aufgebauten Korpus geeignetste ausgewählt. Anschließend wird in diesem Kapitel untersucht, wie die auf dem aufgebauten Textkorpus trainierten Worteinbettungen mit bestehenden auf umfangreichen generischen Korpora trainierten Worteinbettungen kombiniert werden können. In Kapitel 6 wird dann die Implementierung, also das genaue Vorgehen bei der Aufbereitung des Textkorpus, beim Training der Worteinbettungen und bei der Kombination beschrieben. Anschließend werden in Kapitel 7 die auf dem aufgebauten Textkorpus trainierten sowie die kombinierten Worteinbettungen evaluiert. Abschließend erfolgt in Kapitel 8 eine Zusammenfassung der Ergebnisse dieser Bachelorarbeit.

2 Grundlagen

In diesem Kapitel werden grundlegende Informationen dargestellt, die beim Verständnis dieser Bachelorarbeit helfen. Zu Beginn werden die Verarbeitung natürlicher Sprache und das maschinelle Lernen näher erläutert. Anschließend folgen Grundlagen zu Worteinbettungen sowie einige grundlegende Worteinbettungsmodelle. Zum Schluss werden verschiedene Arten von Dokumenten in der Softwaretechnik wie zum Beispiel Anforderungsbeschreibungen eingeführt.

2.1 Verarbeitung natürlicher Sprache

Die Verarbeitung natürlicher Sprache beschreibt die Methoden, die natürliche menschliche Sprachen für Computer zugänglich machen [Eis19]. Dieser Bereich konzentriert sich auf die Entwicklung von Repräsentationen natürlicher Sprache sowie den Entwurf und die Analyse von Algorithmen zur Verarbeitung natürlicher Sprache. In den Algorithmen können dabei die Formen der Repräsentation verwendet werden.

Sogenannte Textkorpora können in der Verarbeitung natürlicher Sprache als Grundlage verwendet werden. Ein Textkorpus bezeichnet eine Sammlung von Texten, wobei ein Text unter anderem nur ein Satz oder ein ganzes Dokument sein kann. Ein Beispiel für ein Textkorpus wäre eine Sammlung bestehend aus allen an einem Institut entstandenen Arbeiten.

In der Verarbeitung natürlicher Sprache werden auf Texte meist mehrere Algorithmen nacheinander angewandt, die Verarbeitung umfasst also mehrere Stufen. Beispiele für solche Stufen sind die Zerteilung in Wörter oder die Feststellung von zusätzlichen Informationen zu diesen Wörtern, wie die Wortart.

Das Ziel der Verarbeitung natürlicher Sprache ist es, neue Möglichkeiten für Computer bereitzustellen, mit natürlicher Sprache umzugehen. Dies umfasst zum Beispiel die Informationsgewinnung aus einem Textkorpus, Übersetzungen zwischen verschiedenen Sprachen oder die Kommunikation des Benutzers mit dem Computer. Letzteres kann sich so darstellen, dass der Computer in natürlicher Sprache gegebene Befehle ausführt oder in natürlicher Sprache gestellte Fragen in natürlicher Sprache beantwortet.

Im Folgenden werden einige Formen der Repräsentation natürlicher Sprache vorgestellt. Diese Repräsentation können zum Beispiel als Eingabe von Algorithmen zur Verarbeitung natürlicher Sprache verwendet werden.

Tabelle 2.1: Darstellung im `Bag-of-Words-Modell`

Wort	Häufigkeit
Tom	1
Tina	1
Pizza	1
Spaghetti	1
und	2
mögen	1

2.1.1 Bag-of-Words-Modell

Das `Bag-of-Words-Modell` beschreibt eine vereinfachte Darstellung von Texten aus einem Textkorpus. Der Text wird dabei in seine Wörter zerlegt. Diese Wörter bilden dann eine Multimenge, die jedes Wort genau so oft enthält, wie es im Text vorkommt [Ska18]. Die Multimenge kann durch die verschiedenen Wörter zusammen mit den Häufigkeiten, in denen sie vorkommen, dargestellt werden. Ein Beispiel dafür findet sich in Beispiel 2.1. Im `Bag-of-Words-Modell` werden also nur die verschiedenen Wörter und ihre Häufigkeiten erfasst. Die Reihenfolge der Wörter wird hingegen nicht berücksichtigt.

Beispiel 2.1:

Der Text

Tom und Tina mögen Pizza und Spaghetti.

hat im `Bag-of-Words-Modell` eine Darstellung wie in Tabelle 2.1.

2.1.2 N-Gramme

Eine Multimenge von N-Grammen stellt eine Erweiterung des `Bag-of-Words-Modells` dar [Ska18]. Um N-Gramme zu bilden, werden die Texte aus einem Textkorpus in einzelne Bestandteile zerlegt. Diese Bestandteile können zum Beispiel die einzelnen Buchstaben oder ganze Wörter sein. Jeweils N direkt aufeinander folgende Bestandteile bilden ein N-Gramm. 1-Gramme werden auch Monogramme und 2-Gramme auch Bigramme genannt. Ein Beispiel findet sich in Beispiel 2.2.

Beispiel 2.2:

Der Text

Sie geht ins Theater.

lässt sich, wenn als Bestandteile die einzelnen Wörter gewählt werden, in die Bigramme

(Sie, geht), *(geht, ins)* und *(ins, Theater)*

zerlegen.

2.1.3 Skip-Gramme

Skip-Gramme ähneln N-Grammen, aber die einzelnen Bestandteile, in die die Texte aus einem Textkorpus zerlegt wurden, müssen nicht direkt aufeinander folgen. Stattdessen kann

es zwischen den Bestandteilen auch Lücken geben, die übersprungen werden [GAL+06]. Ein K-Skip-N-Gramm enthält N Bestandteile, die entweder direkt aufeinander folgen oder höchstens K Bestandteile zwischen sich haben. Ein Beispiel findet sich in Beispiel 2.3.

Beispiel 2.3:

Werden Wörter als Bestandteile gewählt, so lässt sich der Text

Er geht ins Kino.

in die 1-Skip-Bigramme

(Er, geht), *(Er, ins)*, *(geht, ins)*, *(geht, Kino)* und *(ins, Kino)*

zerlegen.

2.1.4 Wortvektoren

Ein Text kann durch all seine Wortvektoren beschrieben werden. Hat man eine Abbildung *abb*, die Wörter aus einem Vokabular auf Vektoren aus dem Vektorraum $\mathbb{R}^n$ abbildet, so wird das Bild eines Wortes als dessen Wortvektor unter *abb* bezeichnet. Eine mögliche Abbildung wäre die One-Hot-Kodierung [CA18]. Bei ihr ist die Dimensionalität n des Wortvektorraumes gleich der Anzahl der Wörter im Vokabular. Jedem Wort wird eine eigene Dimension zugewiesen. Jeder Wortvektor enthält also eine Eins an der der Dimension des Wortes entsprechenden Stelle und sonst nur Nullen. Diese Wortvektoren haben somit eine sehr hohe Dimension und sind dünnbesetzt.

2.2 Maschinelles Lernen

Beim maschinellen Lernen sollen Algorithmen von bestehenden Daten lernen. Mitchell [Mit97] beschreibt das Lernen wie in Definition 2.1.

Definition 2.1:

Ein Computerprogramm lernt aus Erfahrung, wenn sich seine Leistung in einer Klasse von Aufgaben mit Erfahrung verbessert.

Die Daten, von denen im maschinellen Lernen verwendete Algorithmen lernen, sind in einzelne Trainingsbeispiele aufgeteilt. Welche Form diese Beispiele haben und auf welche Art und Weise die Algorithmen von ihnen lernen, hängt vom Algorithmus ab.

Die Algorithmen lassen sich abhängig von der Form der verwendeten Beispiele in zwei Bereiche einteilen, in das überwachte und in das unüberwachte Lernen [GBC16]. Beim überwachten Lernen wird die Funktion mit Beispielen bestehend aus Eingaben und den zugehörigen Ausgaben gelernt. Beim unüberwachten Lernen hingegen werden nur Beispiele bestehend aus Eingaben verwendet, in denen der Algorithmus nach Regelmäßigkeiten sucht.

Viele Algorithmen im maschinellen Lernen nutzen Modelle mit verschiedenen Parametern. Mit festgelegten Parametern kann man so ein Modell als Funktion $g(x)$ mit Eingabe x und Ausgabe $g(x)$ darstellen. Ändern sich die Parameter ändert sich auch die Funktion. Das Ziel des Modells ist es, eine Zielfunktion $f(x)$ zu approximieren. Die Parameter des Modells sollen also möglichst solche Werte haben, dass $g(x)$ die Zielfunktion $f(x)$ möglichst gut approximiert. Die Parameter verbessern sich beim Lernen selbstständig anhand von

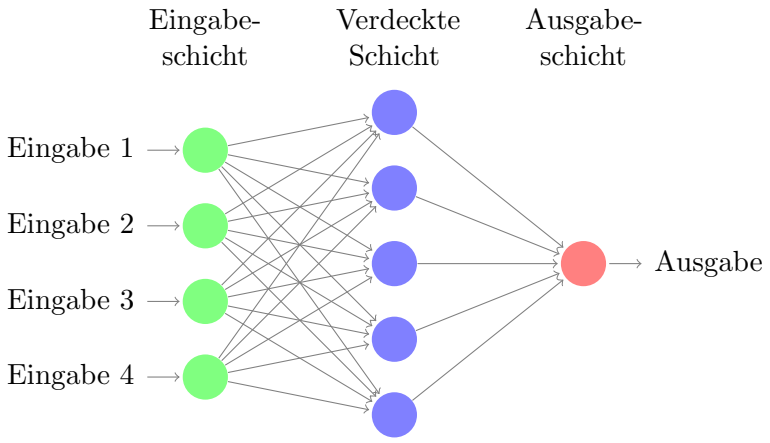

Abbildung 2.1: Schematischer Aufbau eines neuronalen Netzes mit einer verdeckten Schicht

Trainingsbeispielen. Die Trainingsbeispiele bestehen aus je einem Eingabewert x und einem Ausgabewert $y \approx f(x)$. Zu diesen Modellen gehören die sogenannten neuronalen Netze.

Neuronale Netze bestehen aus verschiedenen Schichten, die bei festgelegten Parametern je als eine Funktion $g^{(i)}(x)$ dargestellt werden können. Die durch das neuronale Netz dargestellte Funktion $g(x)$ entspricht dann der Verkettung der durch die einzelnen Schichten dargestellten Funktionen $g^{(i)}(x)$. Hat das neuronale Netz zum Beispiel drei Schichten, so stellt die erste Schicht $g^{(1)}(x)$ dar, die zweite Schicht $g^{(2)}(x)$ und die dritte und letzte Schicht $g^{(3)}(x)$. Das neuronale Netz stellt dann die Funktion $g(x) = g^{(3)}(g^{(2)}(g^{(1)}(x)))$ dar. Die erste Schicht wird auch Eingabeschicht und die letzte auch Ausgabeschicht genannt. Alle Schichten dazwischen werden auch verdeckte Schichten genannt. Die Trainingsbeispiele zeigen direkt den gewünschten Ausgabewert y der Ausgabeschicht. Die gewünschten Ausgabewerte der anderen Schichten sind hingegen nicht direkt aus den Trainingsbeispielen ersichtlich.

Eine von einer Schicht dargestellte Funktion kann als parallel angeordnete künstliche Neuronen veranschaulicht werden. Der Eingabewert der Schicht, ein Vektor, wird dabei an jedes der künstlichen Neuronen weitergeleitet. Jedes der künstlichen Neuronen gewichtet dann individuell entsprechend seiner Gewichte die einzelnen Vektoreinträge. Diese Gewichte stellen die Parameter des neuronalen Netzes dar. Entsprechend der gewichteten Vektoreinträge gibt jedes der künstlichen Neuronen dann einen bestimmten Ausgangswert ab. Die Ausgangswerte bilden dann die Eingabe der nächsten Schicht, beziehungsweise im Fall der letzten Schicht die Ausgabe. Das ganze wird in Abbildung 2.1 nach Fauske[1] dargestellt.

Ein rekurrentes neuronales Netz bezeichnet ein neuronales Netz, bei dem die Neuronen einer Schicht auch mit Neuronen derselben oder vorherigen Schichten verbunden sein können. Ein rekurrentes neuronales Netzes hat also Rückkopplungen, wodurch beim Lernen frühere Informationen verwendet werden können.

Aktuelle Methoden in der Verarbeitung natürlicher Sprache setzen stark auf maschinelles Lernen [Eis19]. Um natürliche Sprache in die Algorithmen einzubinden, werden Repräsentationen natürlicher Sprache, wie oben beispielhaft eingeführt, verwendet.

2.2.1 Klassifikatoren

Ein Entscheidungsbaum dient der Lösung eines Entscheidungsproblems und kann somit zur Klassifizierung von Datenobjekten genutzt werden. Seine Knoten stellen logische Regeln

[1] Quelle: `http://www.texample.net/tikz/examples/neural-network/`, zuletzt besucht am 23.04.2020

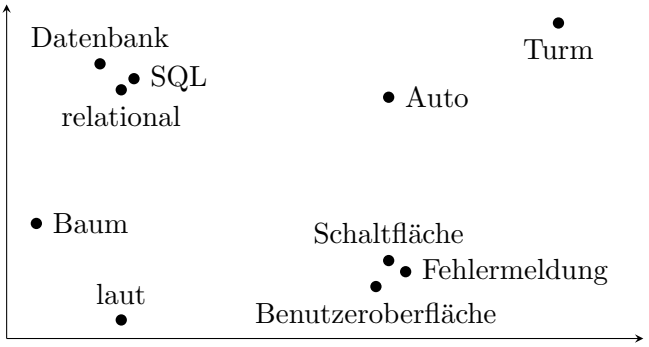

Abbildung 2.2: Schematische Darstellung von Worteinbettungen im Vektorraum

und seine Blätter Antworten auf das Entscheidungsproblem dar [She17]. Ein Zufallswald (engl. *random forest*) kombiniert mehrere Entscheidungsbäume. Diese werden getrennt auf einer jeweils zufällig ausgewählten Teilmenge von Datenobjekten trainiert.

Eine Stützvektormaschine (engl. *support vector machine*) kann als Vektoren dargestellte Datenobjekte klassifizieren, indem sie im Vektorraum eine Hyperebene definiert, die zu den nächstliegenden Vektoren einen möglichst großen Abstand hat [SC08]. Die nächstliegenden Vektoren werden Stützvektoren genannt. Die Seite von der Hyperebene, in der sich ein Vektor befindet, definiert seine Klasse. Sind die Trainingsdatenobjekte nicht linear trennbar, können sie durch den Kernel-Trick in eine höhere Dimension, in der sie linear trennbar sind, überführt werden.

Die logistische Regression ist eine Regressionsanalyse, sie modelliert also den Zusammenhang zwischen mehreren unabhängigen Variablen und einer abhängigen [Osb16]. Wird logistische Regression zur Klassifizierung verwendet, beschreibt die abhängige Variable die Klasse in Abhängigkeit von den unabhängigen Variablen.

2.3 Worteinbettungen

Worteinbettungen werden in der Verarbeitung natürlicher Sprache verwendet. Sie beschreiben eine Abbildung von Wörtern auf Wortvektoren, wobei diese Wortvektoren im Gegensatz zum Beispiel zu den Wortvektoren der One-Hot-Kodierung eine niedrigere Dimension haben und dichtbesetzt sein sollen [CA18]. Sie stellen somit eine Repräsentation natürlicher Sprache dar und die Wortvektoren werden unter anderem als Eingabe von Algorithmen aus den Bereichen Verarbeitung natürlicher Sprache und maschinelles Lernen eingesetzt.

Laut der Verteilungshypothese gibt es eine Korrelation zwischen dem Kontext, in dem Wörter vorkommen, und ihrer Bedeutung [Sah08]. Worteinbettungen nutzen diese Hypothese, indem sie den Kontext von Wörtern berücksichtigen. Dadurch sollen Wörter mit ähnlicher Bedeutung auf ähnliche Wortvektoren abgebildet werden. Das wird in Abbildung 2.2 schematisch dargestellt. Die Wortvektoren von Wörtern mit ähnlicher Bedeutung sollen also einen geringen Abstand zueinander haben. Gleichfalls sollen Wortvektoren von Wörtern, die sehr unterschiedliche Bedeutungen haben, weit voneinander entfernt sein.

Worteinbettungen können auch sinnvolle syntaktische und semantische Regelmäßigkeiten erfassen [MYZ13]. Betrachtet man eine bestimmte Beziehung zwischen je zwei Wörtern, zum Beispiel Singular und Plural, so sind die Wortvektoren dieser Wortpaare ähnlich angeordnet. So zeichnen sich die Wortvektorpaare dadurch aus, dass Richtung und Abstand vom einen zum anderen Wortvektor bei allen Paaren ähnlich ist. In Abbildung 2.3 wird die Beziehung von männlich zu weiblich mit Wortvektoren schematisch dargestellt. Richtung und Abstand von Mann zu Frau entsprechen Richtung und Abstand von König zu Königin.

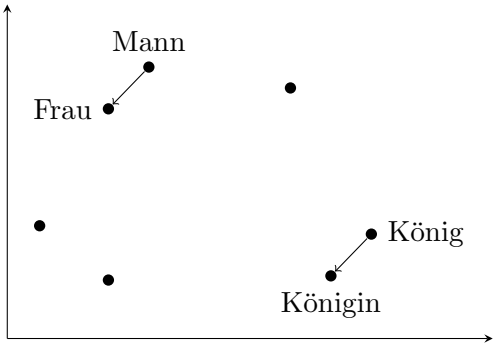

Abbildung 2.3: Schematische Darstellung linguistischer Regelmäßigkeiten in Wortvektoren

Abbildung 2.4: Trainingsfenster mit Größe zwei

Damit lassen sich auch mathematische Operationen auf Wortvektoren durchführen. Im in der Abbildung dargestellten Beispiel gilt für die entsprechenden Wortvektoren $König - Mann + Frau = Königin$.

Um Worteinbettungen zu generieren, können zum Beispiel neuronale Netze eingesetzt werden. Dabei wird ein neuronales Netz auf einem Textkorpus an einer Pseudoaufgabe wie der Vorhersage des nächsten Wortes trainiert. Die Wortvektoren können dann aus den Gewichten der verdeckten Schicht gewonnen werden.

2.3.1 word2vec

Eines der verbreitetsten Worteinbettungsmodelle ist `word2vec` [MCCD13, MSC+13]. Es lässt sich in den Bereich des unüberwachten Lernens einordnen und nutzt ein neuronales Netz. Das Modell hat zwei verschiedene Varianten, das `Continuous Bag-of-Words-Modell` und das `Skip-Gram-Modell`.

Beim Training wird der lokale Kontext eines Wortes berücksichtigt. Dafür wird ein sogenanntes Trainingsfenster mit festgelegter Größe genutzt. Das Wort in der Mitte des Trainingsfensters ist das aktuell betrachtete Wort, die benachbarten Wörter sind der betrachtete Kontext dieses Wortes. In Abbildung 2.4 ist ein beispielhaftes Trainingsfenster mit Größe zwei dargestellt. Das aktuell betrachtete Wort ist *gerne* und der Kontext sind die Wörter *geht*, *samstags*, *auf* und *den*. Das Modell basiert auf dem `Bag-of-Words-Modell`, das heißt die Reihenfolge der Wörter im Kontext wird nicht berücksichtigt. Das Trainingsfenster wird für alle Wörter im Trainingskorpus betrachtet.

Das `Continuous Bag-of-Words-Modell` nutzt nun ein neuronales Netz mit dem Kontext als Eingabe und dem aktuell betrachteten Wort als Ausgabe. Es soll also das aktuell betrachtete Wort anhand des Kontexts vorhersagen.

Das `Skip-Gram-Modell` hingegen verwendet für sein neuronales Netz das aktuell betrachtete Wort als Eingabe und je ein Wort aus dem Kontext als Ausgabe. Es soll also den Kontext anhand des aktuell betrachteten Wortes vorhersagen.

2.3.2 GloVe

Ein weiteres etabliertes Worteinbettungsmodell ist `GloVe` [PSM14]. Auch dieses Modell lässt sich dem Bereich des unüberwachten Lernens zuordnen. Jedoch wird kein neuronales Netz eingesetzt, sondern ein Problem der kleinsten Quadrate gelöst. `GloVe`, das für *Global*

Vectors steht, nutzt die Kookkurenz von Wörtern, also wie oft in den Trainingsdaten ein Wort im Kontext eines anderen Wortes vorkommt. Während bei `word2vec` also nur der lokale Kontext eines Wortes beim Trainieren der Worteinbettungen berücksichtigt wird, wird bei `GloVe` auch die globale Kookurenz berücksichtigt.

2.3.3 fastText

Das Worteinbettungsmodell `fastText` [BGJM17] verwendet ebenfalls ein neuronales Netz, basierend auf dem `Skip-Gram-Modell` oder dem `Continuous Bag-of-Words-Modell`. Bei `fastText` werden jedoch die Wörter als mehrere N-Gramme, die je aus mehreren zusammenhängenden Zeichen des Wortes bestehen, sowie dem Wort selbst dargestellt. Dabei werden zuvor dem Wort die Zeichen „<“ und „>“ hinzugefügt. So wird das Wort *gerne* bei $N = 3$ durch die 3-Gramme <*ge, ger, ern, rne, ne*> sowie das Wort <*gerne*> selbst dargestellt. Diese Darstellungen der Wörter werden dann wie beim `Skip-Gram-Modell`, beziehungsweise wie beim `Continuous Bag-of-Words-Modell` auf Vektoren abgebildet. Der Wortvektor eines Wortes ist dann durch die Summe der Vektoren seiner wie oben eingeführten Darstellung definiert.

2.3.4 ELMo

Wörter können in unterschiedlichen Kontexten unterschiedliche Dinge bedeuten. Mit dem Wort *Pferd* kann zum Beispiel je nach Kontext das Tier oder die Schachfigur gemeint sein. Das Worteinbettungsmodell `ELMo` [PNI+18] hat zum Ziel, dass beim Training der Worteinbettungen die unterschiedlichen Bedeutungen eines Wortes in unterschiedlichen Kontexten berücksichtigt werden. Dafür benutzt `ELMo` rekurrente neuronale Netz mit langem Kurzzeitgedächtnis (engl. *long short term memory*, LSTM), die den Kontext von Wörtern lernen. Durch die Rückkopplungen eines rekurrenten neuronalen Netzes kann dieses eine Art Gedächtnis haben, es können beim Lernen frühere Informationen verwendet werden. Ein LSTM ist eine spezielle Form eines rekurrenten neuronalen Netzes, das ein besseres Gedächtnis hat und somit mehr Informationen von früheren Zeitpunkten beim Lernen berücksichtigen kann. `ELMo` verwendet LSTMs mit mehreren Schichten, die die Eingaben bidirektional, das heißt sowohl in normaler als auch in umgekehrter Reihenfolge durchlaufen. Die Wortvektoren werden dann als Linearkombinationen der internen Zustände der LSTMs gebildet.

2.4 Dokumente in der Softwaretechnik

In der Softwaretechnik spielen Dokumente, die ein Softwaresystem beschreiben, eine wichtige Rolle. Wenn die Entwicklung eines Softwaresystems in Auftrag gegeben wird, kann durch solche Dokumente das zu entwickelnde System vorab zwischen Auftraggeber und Entwicklungsteam genau definiert werden. Bei der Entwicklung, dem Testen und der späteren Wartung und Pflege eines Softwaresystems kann es sehr hilfreich sein, solche Dokumente als Grundlage zu haben. Im Folgenden werden verschiedene Arten dieser Dokumente definiert.

Am wichtigsten für diese Bachelorarbeit sind die in natürlicher Sprache verfassten Anforderungsbeschreibungen. Eine Anforderung bezeichnet eine Bedingung oder Fähigkeit, die ein System erfüllen muss, um eine Vereinbarung einzuhalten [ISO17]. Um erfolgreich ein Softwaresystem zu entwickeln, ist die Definition und Verwaltung von Anforderungen von zentraler Bedeutung. Anforderungen lassen sich in zwei Arten einteilen. Funktionale Anforderungen werden spezifiziert, um die Funktionalität des Systems zu definieren. Nichtfunktionale Anforderungen werden spezifiziert, um die qualitativen Merkmale des Systems wie Performanz und Zuverlässigkeit zu definieren.

Für den Aufbau des Korpus, auf dem Worteinbettungen für die Anforderungsdomäne trainiert werden sollen, werden neben Anforderungsbeschreibungen auch Anwendererzählungen (engl. *user stories*) und Spezifikationen berücksichtigt. In ihnen werden üblicherweise ebenfalls in Anforderungsbeschreibungen vorkommende Formulierungen und Begriffe verwendet.

Anwendererzählungen beschreiben gewünschte Interaktionen des Benutzers mit einem Softwaresystem [Ins13]. So sollen Funktion oder qualitative Merkmale eines Softwaresystems beschrieben werden.

Eine Spezifikation ist eine detaillierte und vollständige Beschreibung eines Systems [ISO17]. Die Beschreibung hat zum Ziel dieses System zu entwickeln oder zu validieren.

Um das Trainingskorpus aufzubauen, könnten Fragen und Antworten zur Softwaretechnik, wie sie sich zum Beispiel auf der Plattform `Stack Overflow` finden lassen, ebenfalls berücksichtigt werden, da auch in ihnen Begriffe aus der Anforderungsdomäne vorkommen können.

3 INDIRECT

Dieses Kapitel führt das Projekt `INDIRECT` (*Intent-driven Requirements-to-Code Traceability*) [Hey19] ein, in dessen Rahmen diese Bachelorarbeit stattfindet. Die Worteinbettungen für die Anforderungsdomäne sollen später in diesem Projekt verwendet werden.

3.1 Zielsetzung

Während der Entwicklung von Softwareprojekten werden zuerst die Anforderungen definiert. Anschließend werden diese Anforderungen im Quelltext implementiert. Dabei ist aber meistens nicht klar ersichtlich, welcher Quelltextteil welche Anforderung implementiert. In der Wartung und Pflege von Software hat die Rückverfolgbarkeit zwischen Quelltext und den entsprechenden Anforderungen jedoch eine große Bedeutung. Das Projekt `INDIRECT` hat zum Ziel, automatisiert Informationen zu dieser Rückverfolgbarkeit zu generieren.

3.2 Lösungsansatz

Im Projekt `INDIRECT` wird das Ziel zur automatisierten Generierung von Rückverfolgbarkeitsinformationen nicht erreicht, indem direkt zwischen Quelltext und Anforderungen eine Zuordnung gebildet wird. Stattdessen basiert der in Abbildung 3.1 veranschaulichte Lösungsansatz darauf, dass für den Quelltext und für die Anforderungen je ein Graph modelliert wird. Die Graphen repräsentieren dabei die Absichten der einzelnen Anforderungen bzw. Quelltextteile. Sie werden deshalb Absichtsmodelle genannt.

Die Absichtsmodelle werden auf verschiedene Weisen erzeugt. Das Anforderungsabsichtmodell entsteht inkrementell durch Verarbeitung natürlicher Sprache. Dabei werden jedem Wort Informationen wie die Wortart zugeordnet. Das Quelltextabsichtsmodell entsteht ebenfalls inkrementell durch Analyse von Quelltext, Tests und weiteren Artefakten wie Kommentaren.

Zwischen den beiden Graphen wird dann eine Zuordnung gebildet. Dafür werden Muster im Anforderungsabsichtmodell und im Quelltextabsichtsmodell gelernt. Der Gedanke dahinter ist, dass die Muster Hinweise auf die Zuordnung geben. Diese Zuordnung repräsentiert dann die Rückverfolgbarkeitsinformationen zwischen Quelltext und Anforderungen.

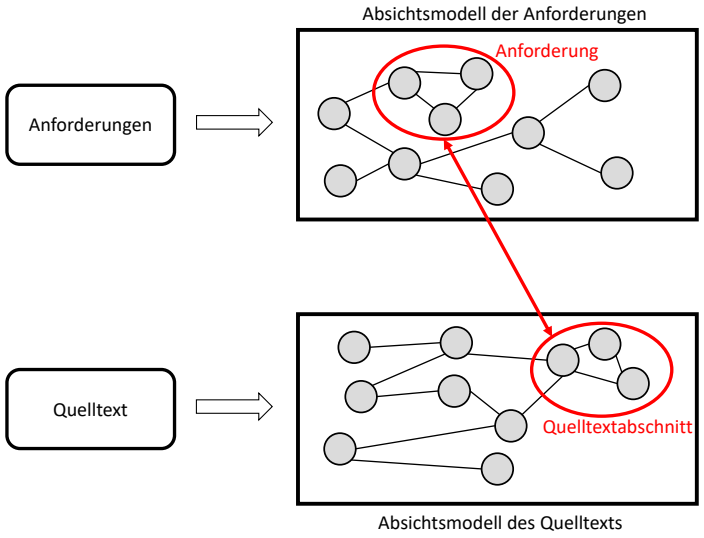

Abbildung 3.1: Veranschaulichung des Lösungsansatzes von `INDIRECT`

4 Verwandte Arbeiten

Dieses Kapitel stellt die verwandten Arbeiten vor. Diese lassen sich in vier Bereiche gliedern, die für die Bildung von Worteinbettungen für die Anforderungsdomäne relevant sind. Zu Beginn werden Arbeiten präsentiert, die den Aufbau eines Textkorpus untersuchen. Anschließend werden Arbeiten vorgestellt, die sich mit der Analyse von Worteinbettungen auseinandersetzen. Dann werden Arbeiten beschrieben, die von domänenspezifischen Worteinbettungen handeln. Zum Schluss werden Arbeiten zur Kombination von Worteinbettungen vorgestellt.

4.1 Aufbau eines Textkorpus

Im Buch **„The Routledge Handbook of Corpus Linguistics“** von O'Keeffe und McCarthy [OM10] wird beschrieben, was beim Aufbau eines Textkorpus beachtet werden sollte. Vor dem Aufbau eines Korpus sollte festgelegt werden, wie die Texte benannt und gespeichert werden. Der Grad der Spezialisierung eines Korpus hängt von der Aufgabe ab. Bei der Größe des Korpus sollte beachtet werden, dass genug Texte gesammelt wurden, um die untersuchte Domäne genau zu repräsentieren. Beim Sammeln der Texte spielt der Grad der Repräsentation eine große Rolle. Soll zum Beispiel ein Textkorpus für eine bestimmte Domäne aufgebaut werden, aber die gesammelten Texte stammen alle von einer Organisation, so wird nur die in dieser Organisation verwendete Sprache repräsentiert, und nicht die Sprache der Domäne allgemein. Es sollte also darauf geachtet werden, dass das Korpus alle verschiedenen Seiten und Auftreten der Domäne repräsentiert und nicht nur einige wenige berücksichtigt. Das Wichtigste, was beim Aufbau eines Textkorpus beachtet werden sollte, ist, dass die Art der Aufgabe, für die das Korpus angefertigt wird, berücksichtigt wird.

Die Arbeit **„PURE: a Dataset of Public Requirements Documents“** von Ferrari et al. [FSG17] beschreibt den Aufbau und die Struktur eines Korpus von Anforderungsdokumenten. Um die Anforderungsdokumente zu erhalten, wurde eine Google-Suche mit Schlüsselwörtern wie *Requirements Documents* oder *Requirements Specification* durchgeführt. Es wurden insgesamt 79 verschiedene Anforderungsdokumente im Internet gefunden. Die Arbeit erhebt keinen Anspruch auf Vollständigkeit, sondern sieht sich als Stichprobe der öffentlich im Internet zugänglichen Anforderungsdokumente. Zudem kann sich die Menge der im Internet verfügbaren Anforderungsdokumente mit der Zeit ändern. Die im `PURE`-Korpus am häufigsten vorkommenden Wörter sind *system*, *shall* und *data*. Ein Vergleich mit dem Brown-Korpus, der als stichprobenhafte Repräsentation der generischen englischen Sprache angesehen werden kann, ergibt, dass 62% der Wörter im `PURE`-Korpus nicht

im Brown-Korpus enthalten sind. Die Arbeit analysiert, dass die Sprache in Anforderungsdokumente sich erheblich von der Sprache in generischen englischen Texten unterscheidet und dass in der Verarbeitung natürlicher Sprache Werkzeuge für die Anforderungsanalyse entsprechend angepasst sein müssen.

4.2 Analyse von Worteinbettungen

Die Arbeit **„*Don't count, predict!* A systematic comparison of context-counting vs. context-predicting semantic vectors“** von Baroni et al. [BDK14] vergleicht die Wortvektoren von kontextzählenden Modellen mit denen von kontextvorhersagenden Modellen, da ein systematischer Vergleich in bestehenden Arbeiten nicht durchgeführt wurde. Als kontextzählende Modelle werden Modelle bezeichnet, die Wortvektoren mit der Kookkurenz von Wörtern, also wie oft ein Wort im Kontext eines anderen Wortes vorkommt, initialisieren. Dann werden auf diese Wortvektoren Transformationen wie Gewichtungen oder Dimensionsreduktionen angewandt. Kontextvorhersagende Modelle bezeichnen Modelle, die an der Pseudoaufgabe der Vorhersage des Kontexts eines Wortes trainiert werden. In der Arbeit werden kontextzählenden Modelle erstellt, die unterschiedliche Parameter haben. Von diesen Modellen werden insgesamt 36 evaluiert. Als kontextvorhersagende Modelle werden `Continuous Bag-of-Words-Modelle` mit unterschiedlichen Parametern verwendet. Davon werden insgesamt 48 evaluiert. Die Modelle werden an einer Vielzahl von Benchmarks getestet. Die Benchmarks testen die semantische Verwandtschaft, die Erkennung von Synonymen, die Kategorisierung von Konzepten, Auswahlpräferenzen und Analogien. Die kontextvorhersagenden Modelle erzielen klar bessere Ergebnisse, in den meisten Benchmarks mit großem Abstand. Zum Beispiel erzielt in einem auf einer Wortähnlichkeitsaufgabe basierenden Benchmark das beste kontextzählende Modell ein Ergebnis von 70%, das beste kontextvorhersagende Modell ein Ergebnis von 80%. Der einzige Benchmark, bei dem die Ergebnisse von vergleichbarer Qualität sind, ist der Test von Auswahlpräferenzen. Anhand der Ergebnisse beschreibt die Arbeit zudem, welche Parameter die besten Ergebnisse erzielen, zum Beispiel erzielt bei kontextvorhersagenden Modellen die Unterabtastung von häufigen Wörtern bessere Ergebnisse. Allgemein sind die Ergebnisse der kontextvorhersagenden Modelle jedoch robuster, sie hängen weniger stark von der Wahl der Parameter ab. Die Arbeit merkt auch noch an, dass es sowohl bei kontextzählenden als auch bei kontextvorhersagenden Modellen sehr viele mögliche Parameter gibt, die nicht alle ausprobiert wurden. Es besteht also durchaus die Möglichkeit, dass es noch Parameter gibt, die bessere Ergebnisse erzielen.

Bakarov stellt in seiner Arbeit **„A Survey of Word Embeddings Evaluation Methods“** [Bak18] Methoden zur Evaluation von Worteinbettungen vor. Es gibt extrinsische Evaluationsmethoden von Worteinbettungen, bei denen zum Beispiel die zu evaluierenden Worteinbettungen in Aufgaben aus dem Bereich der Verarbeitung natürlicher Sprache eingesetzt werden. Die Leistung der Aufgabe mit den eingesetzten Worteinbettungen wird dann als Maß für die Qualität der Worteinbettungen angesehen. Ein Beispiel für eine solche Aufgabe aus dem Bereich der Verarbeitung natürlicher Sprache ist die Stimmungsanalyse, also die Klassifizierung, ob ein Text einen positiven oder negativen Grundtenor hat. Die Anwendung von Evaluationsmethoden, bei denen Worteinbettungen in Aufgaben eingesetzt werden, ist vor allem dann sinnvoll, wenn die zu evaluierenden Worteinbettungen später nur in einer speziellen Aufgabe eingesetzt werden. Da die Leistungen von verschiedenen Aufgaben mit den eingesetzten Worteinbettungen durchaus variieren können, stellen solche Evaluationsmethoden jedoch im Allgemeinen kein ausreichendes Qualitätsmaß für Worteinbettungen dar. Es gibt auch intrinsische Evaluationsmethoden von Worteinbettungen, bei denen zum Beispiel die Worteinbettungen mit vorhandenem Wissen von Wortbeziehungen verglichen werden. Beispiele für solche Wortbeziehungen sind die seman-

tische Ähnlichkeit von Wörtern und Analogien zwischen Wörtern wie zwischen Land und Hauptstadt.

In der Arbeit **„Problems With Evaluation of Word Embeddings Using Word Similarity Tasks“** [FTRD16] von Faruqui et al. werden Probleme, die bei der Evaluation von Worteinbettungen mithilfe von Aufgaben zu semantischen Wortähnlichkeiten auftreten, analysiert sowie bestehende Lösungen vorgestellt. Bei der Evaluation mittels Wortähnlichkeiten wird die Kosinus-Ähnlichkeit von Wortvektoren mit der Einschätzung der Ähnlichkeit der entsprechenden Wörter durch Menschen verglichen. Je geringer die Unterschiede bei diesem Vergleich sind, desto höher wird die Qualität der Worteinbettungen bewertet. Probleme dabei sind die Subjektivität der Einschätzung der Wortähnlichkeit sowie die Nichtberücksichtigung der Mehrdeutigkeit von Wörtern. Letzteres Problem kann gelöst werden, indem für die Einschätzung der Wortähnlichkeiten sowie für die Berechnung der Ähnlichkeit der entsprechenden Wortvektoren eine der Wortbedeutungen ausgewählt wird. Dazu wird allerdings für jede Wortbedeutung ein eigener Wortvektor benötigt, was auf die meisten der häufig benutzten Worteinbettungsmodelle nicht zutrifft. Ein weiteres Problem ist die geringe Korrelation mit extrinsischer Evaluation. Faruqui et al. empfehlen eine solche extrinsische Evaluation, die vergleicht, wie gut die Leistungen von Aufgaben aus dem Bereich der Verarbeitung natürlicher Sprache mit verschiedenen eingesetzten Worteinbettungen sind. Die Variation der Leistungen von verschiedenen Aufgaben mit denselben eingesetzten Worteinbettungen wird in der Arbeit nicht notwendigerweise als Problem angesehen, da die Art der Information, die die eingesetzten Worteinbettungen erfassen, je nach Aufgabe unterschiedlich hilfreich sein kann.

Die Arbeit **„Traversal-Free Word Vector Evaluation in Analogy Space“** von Che et al. [CRR+17] stellt eine alternative Form der Evaluation von Worteinbettungen mittels Analogien zwischen Wörtern vor. Die klassische Form der Evaluation mittels Analogien nutzt die Aufgabe, zu drei gegebene Wörter A, B und C ein Wort D zu finden, dass zu C ist, wie B zu A. Ein Problem der Evaluation durch Analogientests ist, dass wenn die gegebenen drei Wörter bei der Suche nach D nicht ausgeschlossen werden, in 91% der Fälle C als Antwort gefunden wird. Es wird also nicht das Wort gefunden, dass zu C analog ist wie B zu A, sondern das zu C ähnlichste Wort gefunden. In der alternativen Form von Che et al. wird ein Analogienraum mit derselben Dimension wie der Wortvektorraum erstellt. Für je zwei Wörter einer Analogie wird der Relationsvektor gebildet. Der Relationsvektor wird berechnet, indem der erste vom zweiten Wortvektor subtrahiert wird. Der Analogienraum enthält alle Relationsvektoren. Die Evaluation erfolgt dann, indem die Ähnlichkeiten zwischen den Relationsvektoren ermittelt werden. Dafür kann die Kosinus-Ähnlichkeit oder die euklidische Distanz zwischen je zwei Relationsvektoren berechnet werden. Die Arbeit stellt fest, dass diese alternative Form der Evaluation zwar keine besseren Ergebnisse erzielt als die klassische, dafür allerdings eine geringere Berechnungskomplexität hat.

Schnabel et al. untersuchen in ihrer Arbeit **„Evaluation Methods for unsupervised word embeddings“** [SLMJ15] Evaluationsmethoden für Worteinbettungen. Sie stellen eine neue Evaluationsmethode mittels einer Kohärenzaufgabe vor. Anstatt wie bei der klassischen Wortähnlichkeitsaufgabe zu untersuchen, ob semantisch ähnliche Wörter auf Wortvektoren mit geringem Abstand abgebildet werden, wird dabei untersucht, ob Wortgruppen in einer kleinen Nachbarschaft im Wortvektorraum miteinander verwandt sind. Hochqualitative Worteinbettungen sollten für jedes Wort eine kohärente Nachbarschaft haben, wird also ein neues nicht verwandtes Wort in die Nachbarschaft eingefügt, sollte das leicht zu erkennen sein. Die Kohärenzaufgabe besteht daraus, so ein neu hinzugefügtes nicht verwandtes Wort zu erkennen. Je besser die Erkennungsrate ist, desto höher wird die Qualität der Worteinbettungen bewertet. Werden die Ergebnisse der Evaluation mittels der Kohärenzaufgabe mit den Ergebnissen der Evaluation mittels der klassischen Wortähnlichkeitsaufgabe verglichen, so lassen sich zwar Unterschiede feststellen, aber auch einige

Gemeinsamkeiten, woraus sich eine gewisse Korrelation der beiden Evaluationsmethoden schließen lässt.

4.3 Domänenspezifische Worteinbettungen

In der Arbeit **„Detecting Domain-specific Ambiguities: an NLP Approach based on Wikipedia Crawling and Word Embeddings“** von Ferrari et al. [FDG17] soll der Grad der Mehrdeutigkeit von Begriffen aus der Informatik in verschiedenen Anwendungsdomänen untersucht werden. Es werden die Anwendungsdomänen Elektrotechnik, Maschinenbau, Medizin, Literatur und Sport untersucht. Dazu wird für die Domäne der Informatik sowie für jede der Anwendungsdomänen ein Textkorpus aus domänenspezifischen Wikipedia-Artikeln erstellt. Die Korpora werden vorverarbeitet, indem alles kleingeschrieben wird, Stoppwörter entfernt werden und die Wörter lemmatisiert werden. Für jede der Anwendungsdomänen wird ihr Korpus mit dem der Informatikdomäne kombiniert. Dabei werden jedoch zuvor die Wörter im Korpus der Anwendungsdomäne modifiziert, indem ihnen ein Unterstrich vorangestellt wird. So ist es später möglich zu erkennen, aus welchem Korpus ein Wort stammt. Anschließend wird für jede Anwendungsdomäne auf ihrem kombinierten Korpus das `Skip-Gram-Modell` trainiert. Für jeden der Vektorräume werden dann die Ähnlichkeiten zwischen den Wortvektoren eines Wortes und seiner modifizierten Version mithilfe der Kosinus-Ähnlichkeit berechnet. Bei Wörtern mit hoher Ähnlichkeit, also domänenunabhängigen Wörtern, handelt es sich um Begriffe wie *year* und *company*, also nicht besonders informatikspezifische Wörter. Die Wörter hingegen, bei denen die Ähnlichkeit niedrig ist, die also je nach Domäne eine unterschiedliche Bedeutung haben, sind klar informatikspezifisch. Dazu gehören zum Beispiel die Wörter *code* und *database*. Die Arbeit schlägt vor, die durchschnittliche Ähnlichkeit als Maß für die Mehrdeutigkeit von Begriffen aus der Informatikdomäne in anderen Domänen sowie für den technischen Abstand zwischen Domänen zu verwenden. Die ungefähre durchschnittliche Ähnlichkeit liegt in der Elektrotechnikdomäne bei 84,8%, in der Maschinenbaudomäne bei 70,3%, in der Medizindomäne bei 64,3%, in der Literaturdomäne bei 59,5% und in der Sportdomäne bei 51,4%.

In der Arbeit **„Word Embeddings for the Software Engineering Domain“** von Efstathiou et al. [ECS18] werden Worteinbettungen für die Domäne der Softwareentwicklung trainiert, da es an bestehenden Arbeiten zu solchen Worteinbettungen mangelt. Die weit verbreiteten bestehenden Worteinbettungsmodelle wurden auf Textkorpora trainiert, die nicht spezifisch für die Domäne der Softwareentwicklung angefertigt wurden. Für das Training der domänenspezifischen Worteinbettungen wird ein Textkorpus angelegt, der Beiträge enthält, die auf der Plattform `Stack Overflow` veröffentlicht wurden. Diese Auswahl wird in der Arbeit damit begründet, dass die Beiträge kleine Bedeutungsunterschiede in Begriffen der Softwaretechnik erfassen und trotzdem die verschiedene Themen der Softwaretechnik repräsentiert werden. Zudem enthalten viele Beiträge Umgangssprache, die in der Softwareentwicklung ebenfalls häufig verwendet wird. Nach einer mehrschrittigen Vorverarbeitung, die zum Beispiel Quellcodeausschnitte und Stoppwörter entfernt und alle Wörter kleinschreibt, enthält das entstandene Textkorpus über sechs Milliarden Wörter. Als Worteinbettungsmodell wird `word2vec` verwendet. Als Fenstergröße wird fünf gewählt und die Dimension des Wortvektorraumes auf 200 gesetzt. Zur Evaluation werden die domänenspezifischen Worteinbettungen mit den Worteinbettungen aus dem auf Google-News vortrainierten `word2vec-Modell` verglichen. Mehrdeutige Wörter werden im Vergleich unterschiedlich repräsentiert. Zum Beispiel enthalten die zum Wort *virus* über die Abstände der Wortvektoren berechneten ähnlichsten Wörter in den domänenspezifischen Worteinbettungen das Wort *malware*, in den anderen Worteinbettungen den Begriff *flu_virus*. Auch Analogien wie Programmiersprache zu Entwicklungsumgebung werden von den domänenspezifischen Worteinbettungen besser erfasst.

Risch und Krestel stellen in ihrer Arbeit **„Domain-Specific Word Embeddings for Patent Classification“** [RK19] Worteinbettungen für die Patentdomäne vor. Patente müssen von Patentbüros nach einem standardisierten Schema klassifiziert werden. Das Ziel der Automatisierung wird durch die domänenspezifische Sprache erschwert. Deshalb werden in dieser Arbeit domänenspezifische Worteinbettungen trainiert. Das Textkorpus wird dabei aus mehr als fünf Millionen Patentdokumenten gebildet. Als Worteinbettungsmodell wird `fastText` verwendet. Die Fenstergröße ist fünf und Wörter, die weniger als zehnmal vorkommen, werden nicht beachtet. Es werden insgesamt drei Modelle mit den Wortvektorraumdimensionen 100, 200 und 300 trainiert. Die entstandenen Worteinbettungen werden mit auf Wikipedia-Artikeln trainierten Worteinbettungen mit der Dimension 300 verglichen. Es wird die Genauigkeit in der Klassifizierung von Patenten untersucht. Wenn nur die Klasse, die mit der höchsten Wahrscheinlichkeit vorhergesagt wird, berücksichtigt wird, haben die auf Wikipedia-Artikeln trainierten Worteinbettungen eine Genauigkeit von 42%. Die domänenspezifischen Worteinbettungen mit Dimension 100 haben eine Genauigkeit von 45%, diejenigen mit Dimension 200 haben eine Genauigkeit von 48% und diejenigen mit Dimension 300 haben eine Genauigkeit von 49%.

In der Arbeit **„Evaluation of Domain-specific Word Embeddings using Knowledge Resources“** von Nooralahzadeh et al. [NØL18] werden Worteinbettungen für die Öl- und Gasdomäne trainiert und evaluiert. Das Textkorpus, auf dem die Worteinbettungen später mit `word2vec` trainiert werden sollen, wird aus technischen Dokumenten und wissenschaftlichen Artikeln aus der Öl- und Gasdomäne gebildet. Das Korpus enthält über acht Millionen Sätze und wird in mehreren Schritten vorverarbeitet. Unter anderem wird der Text gemischt. Da die Lernrate mit zunehmendem Training sinkt, hätte sonst früherer Text größere Auswirkungen auf die Worteinbettungen. Es werden sowohl `Skip-Gram-Modelle` als auch `Continuous Bag-of-Words-Modelle` mit verschiedenen Parametern trainiert. Die entstandenen Worteinbettungen werden mit intrinsischen Evaluationsmethoden, die auf Aufgaben zu Synonymen, zu Gegenteilen und zu alternativen Wortformen basieren, ausgewertet und die Ergebnisse verglichen. Mit den Standardparametern liefert das `Continuous Bag-of-Words-Modell` im Allgemeinen bessere Ergebnisse als das `Skip-Gram-Modell`. Die Qualität der Ergebnisse des `Continuous Bag-of-Words-Modells` steigt mit der Dimensionalität des Wortvektorraumes, bis die beste Leistung je nach Aufgabe bei Dimension 400 oder 500 erreicht ist. Werden verschiedene Parameter zur Fenstergröße und zum Ausschluss seltener Wörter betrachtet, so ist die Qualität der Ergebnisse je nach Aufgabe nicht konsistent.

4.4 Kombination von Worteinbettungen

Die Arbeit **„Word Embeddings Evaluation and Combination“** von Ghannay et al. [GFEC16] vergleicht die Qualität verschiedener Worteinbettungen und untersucht mögliche Kombinationsmethoden. Es werden Worteinbettungen, die unter anderem aus dem Training des `Continuous Bag-of-Words-Modells`, des `Skip-Gram-Modells` und von `GloVe` auf demselben Textkorpus entstanden, evaluiert und verglichen. Bei allen Modellen wird dabei fünf als Fenstergröße und 200 als Dimension des Wortvektorraumes verwendet. Zur Evaluation werden die Worteinbettungen in vier Aufgaben aus der Verarbeitung natürlicher Sprache eingesetzt. Dabei erzielen in zwei Aufgaben das `Continuous Bag-of-Words-Modell` mit 90,48% und 78,32% und in zwei Aufgaben das `Skip-Gram-Modell` mit 96,43% und 57,80% die besten Ergebnisse. Des Weiteren werden die Leistungen in Aufgaben zu Wortanalogien untersucht. Hierbei hat das `Continuous Bag-of-Words-Modell` eine Gesamtgenauigkeit von 57,2%, das `Skip-Gram-Modell` eine Gesamtgenauigkeit von 62,3% und `GloVe` eine Gesamtgenauigkeit von 65,5%. Zudem werden die Leistungen in drei Aufgaben zu Wortähnlichkeiten untersucht. Hierbei zeigen einmal das `Continuous Bag-of-Words-Modell`

mit 59,0% und zweimal das `Skip-Gram-Modell` mit 50,2% und 66,2% die besten Ergebnisse. Die verschiedenen Worteinbettungsmodelle zeigen also je nach Aufgabe unterschiedlich gute Ergebnisse, kein Worteinbettungsmodell ist klar das beste. Um die Vorteile der verschiedenen Modelle je nach Aufgabe zu vereinen, werden verschiedene Methoden zur Kombination der drei Worteinbettungsmodelle untersucht. Bei der ersten Methode `Concat` werden die Worteinbettungen konkateniert, sodass der resultierende Wortvektorraum eine Dimension von 600 hat. Die zweite Methode `PCA` wendet die Hauptkomponentenanalyse auf die konkatenierten Worteinbettungen an. Die dritte Methode `AutoE` nutzt einen Autoencoder, also ein neuronales Netz mit den konkatenierten Worteinbettungen als Eingabe. Die Werte der verdeckten Schicht bilden dann die kombinierten Worteinbettungen. Die verschiedenen kombinierten Worteinbettungen werden wieder auf dieselbe Weise evaluiert. In den Aufgaben aus der Verarbeitung natürlicher Sprache erzielen in einer Aufgabe `PCA` mit 96,39% und in drei Aufgaben `Concat` mit 91,24%, 79,43% und 57,86% die besten Ergebnisse. In den Aufgaben zu Wortanalogien hat `Concat` eine Gesamtgenauigkeit von 62,9%, `PCA` eine Gesamtgenauigkeit von 62,8% und `AutoE` eine Gesamtgenauigkeit von 56,0%. In den Aufgaben zu Wortähnlichkeiten zeigen einmal `Concat` mit 60,2% und zweimal `PCA` mit 49,6% und 66,9% die besten Ergebnisse.

In der Arbeit **„Learning Word Meta-Embeddings“** von Yin und Schütze [YS16] werden verschiedene Methoden zur Kombination von Worteinbettungen vorgestellt. Die aus der Kombination resultierenden Worteinbettungen werden als Metaeinbettungen bezeichnet. Die Ziele der Kombination sind bessere Repräsentationen der Wörter, die Metaeinbettungen sollen also bessere Leistungen erbringen als die zu kombinierenden Worteinbettungen, und eine größere Abdeckung, die Metaeinbettungen sollen also zu allen Wörtern Worteinbettungen enthalten, zu denen mindestens eine Worteinbettung in den zu kombinierenden Worteinbettungen existiert. Es werden insgesamt die vier Kombinationsmethoden `CONC`, `SVD`, `1TON` und `1TON`$^+$ vorgestellt. Nur die Kombinationsmethode `1TON`$^+$ hat dabei den Vorteil der größeren Abdeckung. Bei der Kombinationsmethode `CONC` werden die Worteinbettungen konkateniert, wobei die einzelnen Mengen der zu kombinierenden Worteinbettungen unterschiedlich stark gewichtet werden können. Die Kombinationsmethode `SVD` verwendet zur Berechnung der Metaeinbettungen eine Singulärwertzerlegung $C = USV^T$ der Matrix C mit den gewichteten konkatenierten Wortvektoren als Zeilen. Aus der Matrix U werden dann die Metaeinbettungen entnommen. Bei der Kombinationsmethode `1TON` werden die Metaeinbettungen von einem neuronalen Netz gelernt. Das neuronale Netz erhält die Metaeinbettungen als Eingabe und die einzelnen Mengen der zu kombinierenden Worteinbettungen als Ausgabe. Dabei werden die Metaeinbettungen zu Beginn zufällig initialisiert. Zudem werden nur die Wörter betrachtet, zu denen es in jeder Menge der zu kombinierenden Worteinbettungen eine entsprechende Worteinbettung gibt. Die Pseudoaufgabe, an der das neuronale Netz trainiert wird, ist die Vorhersage der einzelnen Elemente der Mengen der zu kombinierenden Worteinbettungen aus den entsprechenden Metaeinbettungen. Die Kombinationsmethode `1TON`$^+$ stellt eine Erweiterung der Kombinationsmethode `1TON` dar, bei der Metaeinbettungen für alle Wörter gebildet werden, zu denen mindestens eine Worteinbettung in den zu kombinierenden Worteinbettungen existiert. Mithilfe der Methode `MUTUALLEARNING` können `CONC`, `SVD` und `1TON` so angepasst werden, dass auch sie den Vorteil der größeren Abdeckung haben. Die Kombinationsmethoden werden anhand Aufgaben zu Wortähnlichkeiten, zu Wortanalogien und zur Wortartmarkierung (engl. *part-of-speech tagging*), also der Zuordnung von Wörtern zu Wortarten, evaluiert. Je nach Aufgabe liefert eine andere Kombinationsmethode die besten Ergebnisse. Bei den Aufgaben zur Wortartmarkierung liefern alle Kombinationsmethoden bis auf `SVD` bessere Ergebnisse als die einzelnen Mengen zu kombinierender Worteinbettungen.

Die Arbeit **„Frustratingly Easy Meta-Embeddings-Computing Meta-Embeddings by Averaging Source Word-Embeddings“** von Coates und Bollegala [CB18] untersucht die Kom-

bination von Worteinbettungen durch Durchschnittsbildung. Die aus der Kombination resultierenden Worteinbettungen werden wieder als Metaeinbettungen bezeichnet. Vor der Durchschnittsbildung werden die jeweiligen Worteinbettungen normalisiert. Coates und Bollegala zeigen, dass, obwohl die Wortvektorräume der zu kombinierenden Worteinbettungen nicht vergleichbar sind, semantische Informationen der zu kombinierenden Worteinbettungen bei der Durchschnittsbildung erhalten bleiben. Die relativen Entfernungen zwischen den Wortvektoren bleiben nämlich bei der Durchschnittsbildung erhalten. Die Metaeinbettungen werden anhand Aufgaben zu Wortähnlichkeiten und zu Wortanalogien evaluiert. Dabei werden die Ergebnisse mit den Ergebnissen der zu kombinierenden Worteinbettungen sowie der durch Konkatenation erhaltenen Worteinbettungen verglichen. Insgesamt sind die Ergebnisse der durch Durchschnittsbildung und der durch Konkatenation erhaltenen Worteinbettungen recht ähnlich, in manchen Fällen sind die Ergebnisse der durch Durchschnittsbildung erhaltenen Metaeinbettungen etwas besser. Zusätzlich hat die Durchschnittsbildung den Vorteil der im Vergleich zur Konkatenation geringeren Dimensionalität der kombinierten Wortvektoren. Bei Kombination von einem `Continuous Bag-of-Words-Modell` und einem `GloVe-Modell` liefern die durch Durchschnittsbildung erhaltenen Metaeinbettungen in einer Aufgabe zu Wortähnlichkeiten ein Ergebnis von 46,5%. Die durch Konkatenation erhaltenen Worteinbettungen erzielen ein Ergebnis von 46,3%. Die zur Kombination verwendeten Modelle, das `Continuous Bag-of-Words-Modell` und das `GloVe-Modell` liefern 44,2%, beziehungsweise 45,3%.

Kameswara Sarma et al. stellen in ihrer Arbeit **„Domain Adapted Word Embeddings for Improved Sentiment Classification“** [KSLS18] eine Methode zur Kombination von Worteinbettungen vor. Als generische Worteinbettungen werden Worteinbettungen bezeichnet, die auf einem umfangreichen generischen Korpus trainiert wurden. Domänenspezifische Worteinbettungen sind Worteinbettungen, die auf einem Korpus, der Dokumente aus der Domäne enthält, trainiert wurden. Die Arbeit stellt eine Methode zur Kombination von generischen und domänenspezifischen Worteinbettungen vor. Die aus der Kombination entstehenden Worteinbettungen werden domänenadaptierte (engl. *domain adapted*) Worteinbettungen genannt. Das Ziel der Kombination ist, dass die domänenadaptierten Worteinbettungen sowohl den Umfang der generischen als auch die Spezifität der domänenspezifischen Worteinbettungen als Eigenschaften aufweisen. Die generischen und die domänenspezifischen Worteinbettungen werden kombiniert, indem eine kanonische Korrelationsanalyse (engl. *canonical correlation analysis*, CCA) eingesetzt wird. Dabei wird entweder eine lineare CCA oder eine nichtlineare Kern CCA (engl. *kernel CCA*, KCCA) verwendet. So werden für ein Wort die entsprechenden generischen und domänenspezifischen Wortvektoren in die Richtung der höchsten Korrelation projiziert. Die erhaltenen projizierten Worteinbettungen werden dann nach einem Optimierungsansatz linear kombiniert. Zur Evaluation werden mehrere generische, domänenspezifische und domänenadaptierte Worteinbettungen in einer Aufgabe zur Stimmungsanalyse eingesetzt. Die domänenadaptierten Worteinbettungen erzielen dabei bessere Ergebnisse als die generischen oder die domänenspezifischen Worteinbettungen. Zum Beispiel liefern die generischen Worteinbettungen, die durch das Training von `GloVe` auf einem Common Crawl-Datensatz entstanden, bei der Stimmungsanalyse von Amazon-Reviews eine durchschnittliche Genauigkeit von 79,91%. Die domänenspezifischen Worteinbettungen, die durch latente semantische Analyse entstanden, erzielen eine durchschnittliche Genauigkeit von 82,65%. Die domänenadaptierten Worteinbettungen, die durch Kombination dieser generischen und domänenspezifischen Worteinbettungen mithilfe einer KCCA entstanden, liefern eine durchschnittliche Genauigkeit von 89,73%.

5 Analyse und Entwurf

In diesem Kapitel werden zuerst die Problemstellung und die Zielsetzung erläutert. Anschließend werden für jedes der einzelnen Ziele eine Analyse durchgeführt und Entwurfsentscheidungen getroffen. Zum Schluss wird der Entwurf zusammengefasst.

5.1 Problemstellung

Worteinbettungen spielen in der Verarbeitung natürlicher Sprache eine immer größere Rolle. Sie werden in vielen verschiedenen Aufgaben wie zum Beispiel der Klassifizierung von Texten eingesetzt.

Worteinbettungen können auch in Aufgaben aus der Anforderungsdomäne auf vielfältige Weise eingesetzt werden. Sie können zum Beispiel bei der Erkennung von äquivalenten Anforderungen oder bei der Klassifizierung von Anforderungen als funktional oder nichtfunktional helfen. Auch bei der Erkennung von nicht eindeutigen Formulierungen oder von Widersprüchen in Anforderungen können sie verwendet werden. Ein konkretes Beispiel ist das Projekt `INDIRECT` [Hey19]. Hier werden Worteinbettungen zur Bestimmung der semantischen Funktion von Sätzen in Anforderungsbeschreibungen verwendet.

Existierende vortrainierte Worteinbettungsmodelle wurden häufig auf umfangreichen generischen Korpora wie zum Beispiel Google-Nachrichten oder Wikipedia-Artikeln trainiert. In diesen Textkorpora kommen manche Begriffe aus der Anforderungsdomäne selten oder mit einer anderen Bedeutung vor. Daher können domänenspezifische Feinheiten in der Verteilung und Bedeutung von Wörtern von diesen Worteinbettungsmodellen nicht besonders gut erfasst werden. Infolgedessen kann die Verwendung von nicht speziell auf die Anforderungsdomäne angepassten Worteinbettungen in Aufgaben aus der Anforderungsdomäne zu Problemen und Ergebnissen von schlechter Qualität führen.

Es kann zum Beispiel passieren, dass anforderungsspezifische Begriffe nicht in dem generischen Textkorpus, auf dem ein Worteinbettungsmodell trainiert wurde, vorkommen. Infolgedessen kann das Worteinbettungsmodell auch nicht den Kontext dieser dem Modell nicht bekannten Wörter erfassen. Typischerweise werden diese Wörter in den aus dem Modell gewonnenen Worteinbettungen alle auf den Nullvektor abgebildet. In Abbildung 5.1 ist ein Ausschnitt einer Abbildung einiger Wörter auf Wortvektoren schematisch dargestellt. Die Wörter *SQL* und *Benutzerschnittstelle* kamen offenbar nicht im Textkorpus, auf dem diese Worteinbettungen trainiert wurden, vor und werden somit auf denselben Wortvektor, den Nullvektor, abgebildet, was nicht besonders aussagekräftig oder hilfreich ist.

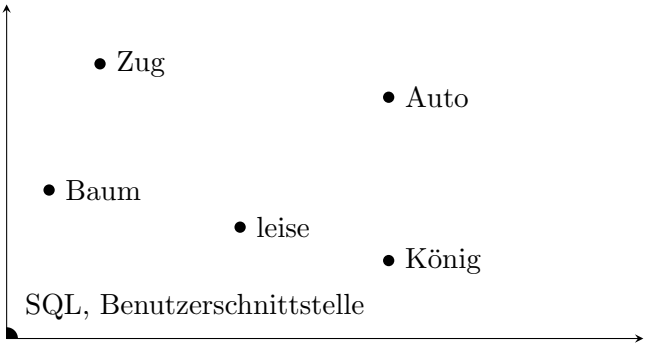

Abbildung 5.1: Schematische Darstellung von der Abbildung mehrerer Wörter auf den Nullvektor

Ein weiteres Problem ist die Mehrdeutigkeit von Begriffen. Ferrari et al. [FDG17] haben festgestellt, dass Begriffe aus der Informatik in verschiedenen Domänen unterschiedliche Bedeutungen haben. Wörter können im Kontext von Anforderungen eine andere Bedeutung haben als in einem anderen Kontext wie Nachrichten. Ein Beispiel dafür ist das Wort *Fenster*, womit sowohl das Zeitfenster als auch die Lichtöffnung gemeint sein kann.

Worteinbettungen, die speziell für die Anforderungsdomäne gebildet wurden, könnten diese Probleme beheben und in Aufgaben aus der Anforderungsdomäne bessere Ergebnisse erzielen.

5.2 Zielsetzung

Diese Arbeit hat das Ziel, Worteinbettungen für die Anforderungsdomäne zu bilden.

Dazu soll zuerst ein Korpus von Anforderungsbeschreibungen und anderen in der Anforderungsdomäne üblichen Dokumenten wie Anwendererzählungen gebildet werden, auf denen die Worteinbettungen trainiert werden können.

Die Qualität von Worteinbettungen hängt nicht nur von den Trainingsdaten, sondern auch vom Trainingsverfahren ab. Es sollen daher bestehende Worteinbettungsmodelle, wie zum Beispiel `word2vec`, `GloVe` oder `fastText`, mit Augenmerk darauf untersucht werden, welche Vor- und Nachteile diese für die Anwendung auf Anforderungen haben. Dann wird ein geeignetes Worteinbettungsmodell ausgewählt. Dabei werden die zuvor untersuchten Vor- und Nachteile für die Anwendung auf Anforderungen sowie Eigenschaften des Korpus, wie zum Beispiel die Quantität der Anforderungen, berücksichtigt.

Um auch den größeren Umfang der auf umfangreichen generischen Korpora trainierten bestehenden Worteinbettungen nutzen zu können, soll abschließend untersucht werden, wie die domänenspezifischen Worteinbettungen in geeigneter Form mit diesen Worteinbettungen kombiniert werden können. Dabei sollen die domänenspezifischen Feinheiten und Bedeutungen der domänenspezifischen Worteinbettungen erhalten bleiben. Nach der Wahl der Kombinationsmethode werden bestehende Worteinbettungen auf ihre Eignung zur Kombination untersucht und ausgewählt.

5.3 Aufbau des Textkorpus

Das erste Ziel dieser Arbeit ist es, ein englischsprachiges Textkorpus von Anforderungsbeschreibungen und anderen Dokumenten aus der Anforderungsdomäne zu bilden.

Am wichtigsten für den Aufbau des Korpus sind die Anforderungsbeschreibungen. Ein Datensatz von öffentlich im Internet zugänglichen Anforderungsdokumenten ist das `PURE`-Korpus von Ferrari et al. [FSG17]. Er enthält insgesamt 79 verschiedene Anforderungsdokumente. 62% der Wörter im `PURE`-Korpus sind nicht im Brown-Korpus enthalten. Da

das Brown-Korpus als stichprobenhafte Repräsentation der generischen englischen Sprache angesehen werden kann, bedeutet das, dass das PURE-Korpus viele domänenspezifische Begriffe enthält.

38% der Anforderungen stammen von Universitäten, der Rest stammt von privaten Unternehmen oder Organisationen wie Standardisierungsgruppen. Da in jedem Unternehmen ein eigener Sprachstil vorherrscht, wäre es von Vorteil, weitere Anforderungsdokumente aus anderen Organisationen zu sammeln, um alle möglichen Sprachstile abzudecken.

Um die Anforderungsdokumente im PURE-Korpus zu erhalten, wurde eine Google-Suche mit den durch eine Oder-Verknüpfung verbundenen Schlüsselwörtern *Requirements Documents*, *Requirements Specification*, *System Specification*, *Software Specification* und *SRS* durchgeführt. Dann wurden die Links, die zu Anforderungsdokumenten führen, ausgewählt und die entsprechenden Webseiten nach etwaigen weiteren Anforderungsdokumenten durchsucht.

Um zu überprüfen, ob das PURE-Korpus noch aktuell ist, oder ob inzwischen neue Anforderungsdokumente zu finden sind, wurde eine erneute Google-Suche mit denselben Schlüsselwörtern durchgeführt. Dabei wurden zwar keine weiteren einzelnen Anforderungsdokumente gefunden, jedoch einige Datensätze, die im Folgenden aufgeführt werden.

Datensätze mit weiteren Anforderungsdokumenten sind der öffentlich zugängliche NFR-Datensatz [CMLP07], der 625 Anforderungsbeschreibungen enthält, sowie der Software Requirement Risk Prediction-Datensatz [SNZ18], der 299 Anforderungsbeschreibungen enthält. Von PROMISE [SSM05] stammen die Datensätze MODIS mit 68 Anforderungsbeschreibungen und CM1 mit 455 Anforderungsbeschreibungen. Der EBT-Datensatz [CCC03] enthält 41, der GANNT-Datensatz [HHD09] 86 und der Ice Breaker-Datensatz [CSB+05] 201 Anforderungsbeschreibungen. Von CoEST[1] werden der iTrust-Datensatz mit 131 Anforderungsbeschreibungen und der WARC-Datensatz mit 152 Anforderungsbeschreibungen angeboten. All diese Datensätze wurden ursprünglich für Aufgaben aus dem Bereich der Generierung von Rückverfolgbarkeitsinformationen erstellt und enthalten somit noch weitere Daten, die für dieses Korpus irrelevant sind und gelöscht werden können.

All diese gefundenen Anforderungsbeschreibungen werden beim Aufbau des Textkorpus verwendet.

Zusätzlich werden auch andere Dokumente aus der Anforderungsdomäne berücksichtigt, in denen ebenfalls in Anforderungsbeschreibungen vorkommende Formulierungen und Begriffe verwendet werden. Durch eine Google-Suche wurde dafür der Datensatz von Dalpiaz [Dal18] gefunden, der Anwendererzählungen enthält und beim Aufbau des Textkorpus verwendet wird.

In der Arbeit von Efstathiou et al. [ECS18] werden Worteinbettungen für die Domäne der Softwareentwicklung trainiert. Für das Training der domänenspezifischen Worteinbettungen wird ein Textkorpus angelegt, der Beiträge enthält, die auf der Plattform Stack Overflow veröffentlicht wurden. Es gibt allerdings einige Punkte, die gegen eine Verwendung der Stack Overflow-Beiträge beim Aufbau des Textkorpus für die Anforderungsdomäne sprechen. So weist die Anforderungsdomäne einige Unterschiede zur Softwareentwicklungsdomäne auf und die Sprache in Stack Overflow-Beiträgen ist zudem deutlich informeller als in typischen Anforderungsbeschreibungen. Deswegen werden Stack Overflow-Beiträge beim Aufbau des Textkorpus nicht berücksichtigt.

5.3.1 Eigenschaften des Textkorpus

Um später eine fundierte Wahl eines Worteinbettungsmodells zum Training der Worteinbettungen für die Anforderungsdomäne treffen zu können, wird das aufgebaute Textkorpus

[1] Quelle: www.coest.org, zuletzt besucht am 23.04.2020

Tabelle 5.1: Eigenschaften des aufgebauten Textkorpus

Bestandteil	Anzahl
Projekte	65
Anforderungsbeschreibungen	21 458
Anwendererzählungen	1680
Sätze	15 388
Wörter	346 256
Wörter ohne Stoppwörter	214 542
Vokabular	16 129

von Anforderungsbeschreibungen und anderen Dokumenten aus der Anforderungsdomäne im Folgenden auf seine Eigenschaften untersucht. Die wichtigsten Eigenschaften des Textkorpus sind in Tabelle 5.1 dargestellt.

Das Textkorpus wurde aus insgesamt 65 verschiedenen Projekten aufgebaut. Die einzelnen Projekte und ihre jeweiligen Eigenschaften sind in Abschnitt A im Anhang aufgeführt. Die verschiedenen Projekte unterscheiden sich stark in ihrem Umfang. So enthält das kleinste Projekt, `Libra`, nur 25 Anforderungsbeschreibungen, während das größte Projekt, `NPAC SMS`, 3570 Anforderungsbeschreibungen enthält.

Das aufgebaute Textkorpus umfasst insgesamt 23 138 Anforderungen mit 21 458 Anforderungsbeschreibungen und 1680 Anwendererzählungen. Eine Anforderung hat eine durchschnittliche Länge von knapp 15 Wörtern, beziehungsweise ungefähr 93 Zeichen. Die Anzahl der Sätze im Textkorpus beträgt 15 388. Somit besteht eine Anforderung aus durchschnittlich etwa eineinhalb Sätzen.

Insgesamt beinhaltet das Textkorpus 346 256 Wörter. Das Google-Nachrichten-Korpus, für den ein vortrainiertes `word2vec-Modell` existiert, enthält im Vergleich ungefähr 100 Milliarden Wörter [MSC+13], ist also deutlich umfangreicher. Die Größe des Vokabulars des aufgebauten Textkorpus, also die Anzahl der voneinander verschiedenen Wörter, beträgt 16 129. Die im Textkorpus am häufigsten vorkommenden Wörter sind in Abbildung 5.2 dargestellt. Die fünf häufigsten Wörter sind *the* mit 25 561, *to* mit 12 569, *of* mit 9087, *a* mit 9028 und *and* mit 7625 Vorkommen. Der Anteil dieser fünf Wörter am Textkorpus beträgt knapp 18,4%. Es lässt sich erkennen, dass die häufigsten Wörter recht allgemein sind und vor allem Begriffe wie Artikel oder Konjunktionen, sogenannte Stoppwörter, enthalten. Um aussagekräftigere Daten zu erhalten, können solche Begriffe aus dem Korpus entfernt werden.

Nach der Entfernung von Stoppwörtern beinhaltet das Textkorpus insgesamt 214 542 Wörter. Es wurden somit 131 714 Stoppwörter entfernt. Die Größe des Vokabulars ohne Stoppwörter beträgt 16 085 verschiedene Wörter. Die im Textkorpus ohne Stoppwörter am häufigsten vorkommenden Wörter sind in Abbildung 5.3 dargestellt. Die fünf häufigsten Wörter im Textkorpus ohne Stoppwörter sind *shall* mit 4912, *data* mit 2813, *system* mit 2443, *npac* mit 2431 und *sms* mit 2095 Vorkommen. Der Anteil dieser fünf Wörter am Textkorpus ohne Stoppwörter beträgt ungefähr 6,8%. Es lässt sich feststellen, dass die häufigsten Wörter ohne Stoppwörter weniger allgemein sind, sondern Begriffe wie *data* und *system* enthalten, die typischerweise oft in Anforderungen vorkommen. Auch projektspezifische Begriffe wie *npac* und *sms* kommen aufgrund des Umfangs des Projektes `NPAC SMS` häufig vor.

Die häufigsten Trigramme von Wörtern im aufgebauten Textkorpus ohne das Projekt `NPAC SMS` sind in Abbildung 5.4 dargestellt. Dabei kommt *I want to* 1608 mal, *so that I* 700

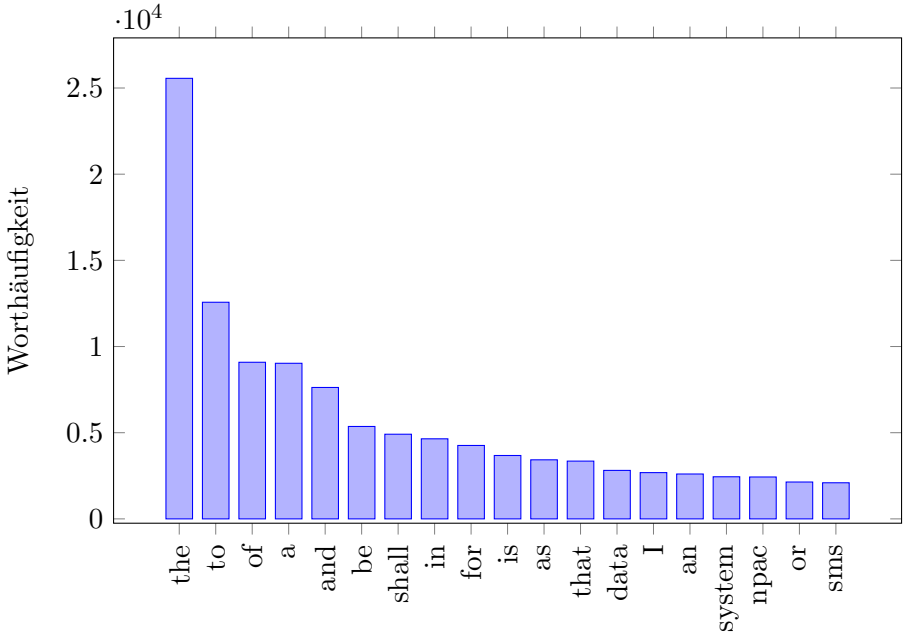

Abbildung 5.2: Worthäufigkeiten im Textkorpus

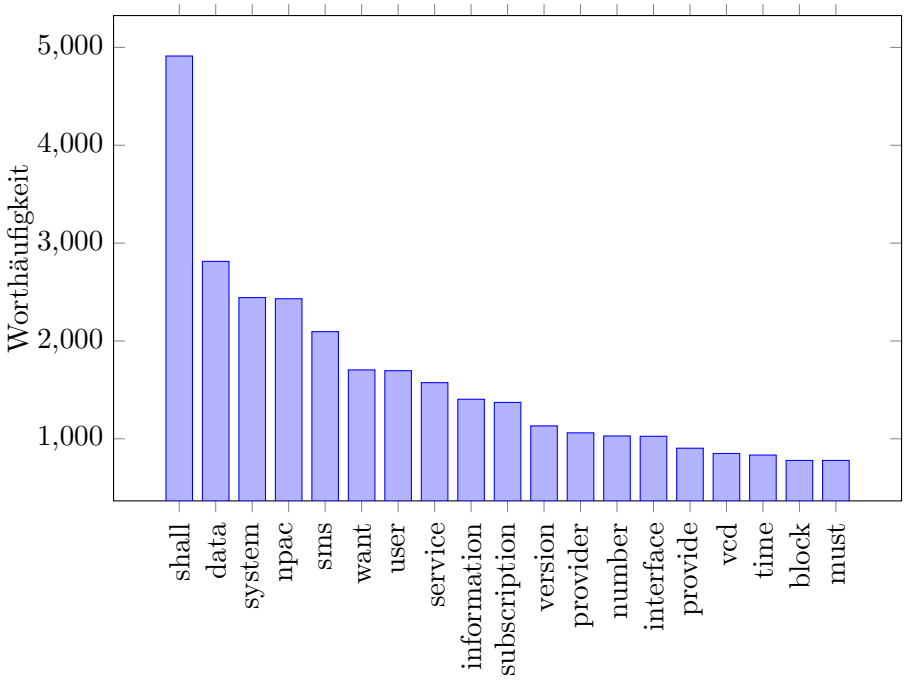

Abbildung 5.3: Worthäufigkeiten im Textkorpus ohne Stoppwörter

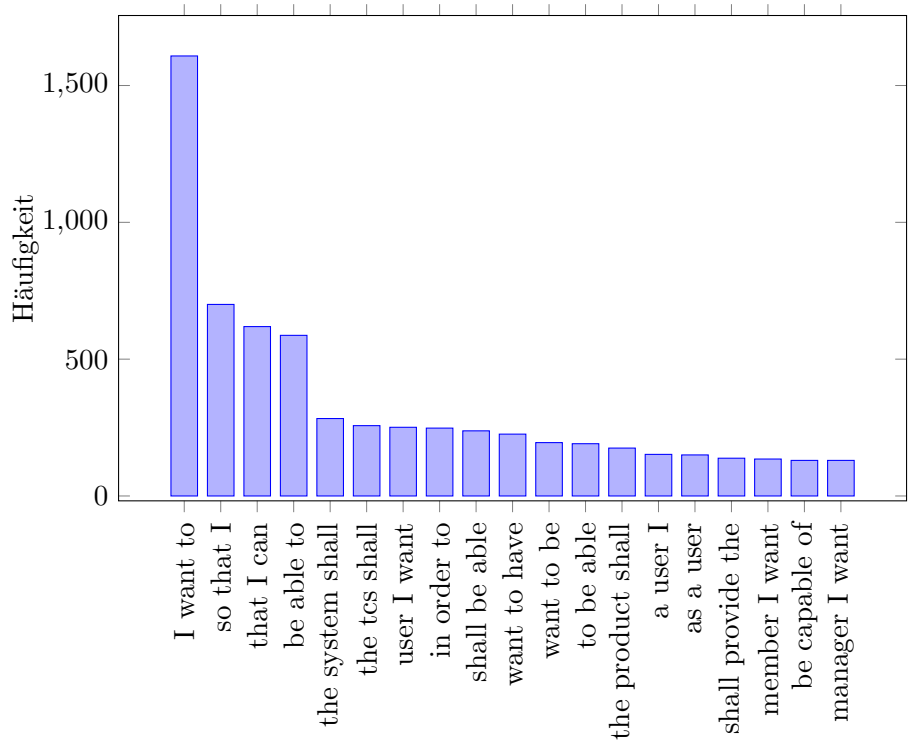

Abbildung 5.4: Häufigkeiten der Trigrammen von Wörtern im Textkorpus ohne das Projekt `NPAC SMS`

mal, *that I can* 619 mal, *be able to* 587 mal und *the system shall* 283 mal vor. Anforderungen haben oft eine ähnliche Struktur und folgen einem bestimmten Schema mit sich in verschiedenen Anforderungen immer wieder wiederholenden Formulierungen. Das lässt sich auch in den häufigsten Trigrammen erkennen, die typische Formulierungen wie *the system shall* enthalten. Besonders innerhalb eines Projektes ist die Struktur der einzelnen Anforderungen oft sehr ähnlich und bestimmte Formulierungen kommen sehr oft vor. Deshalb wurde auch das Projekt `NPAC SMS` hier ausgeklammert, da es sehr umfangreich ist und viele Anforderungen enthält, die immer dem selben Schema folgen. Dadurch würden bestimmte Trigramme aus diesem Projekt, wie zum Beispiel *npac sms shall*, überrepräsentiert werden. Die häufigsten Trigramme von Wörtern im Textkorpus mit dem Projekt `NPAC SMS` finden sich in Abbildung B.1 im Anhang.

Zusammenfassend lässt sich feststellen, dass das aufgebaute Textkorpus dafür, dass er nur Anforderungsbeschreibungen und andere Dokumente aus der Anforderungsdomäne enthält, recht umfangreich und vielfältig ist. Generische Korpora, wie exemplarisch am Google-Nachrichten-Korpus gezeigt, sind aber, wie zu erwarten war, noch einmal deutlich umfangreicher. Dieser Umstand kann dann später in der Kombination der domänenspezifischen mit generischen Worteinbettungen ausgenutzt werden.

5.4 Worteinbettungsmodelle

Im Folgenden werden verschiedene Worteinbettungsmodelle auf ihre Vor- und Nachteile für das Training auf Dokumenten aus der Anforderungsdomäne untersucht.

Eines der verbreitetsten kontextvorhersagenden Worteinbettungsmodelle ist `word2vec` [MCCD13, MSC+13], das zum Lernen von Worteinbettungen ein neuronales Netz verwendet. Es hat zwei verschiedene Varianten, das `Continuous Bag-of-Words-Modell` und das `Skip-Gram-Modell`, die verschiedene Pseudoaufgaben nutzen, an denen das neuronale Netz trainiert

wird. Das `Continuous Bag-of-Words-Modell` soll ein Wort anhand seines Kontexts vorhersagen. Das `Skip-Gram-Modell` soll den Kontext eines Wortes anhand dieses Wortes vorhersagen.

Laut Mikolov[2] hat das `Continuous Bag-of-Words-Modell` die Vorteile, deutlich schneller trainiert werden zu können und ein bisschen genauer bei im Trainingskorpus häufig auftretenden Wörtern zu sein. Das `Skip-Gram-Modell` hingegen hat die Vorteile, auch mit einem kleinen Trainingskorpus gute Resultate zu liefern und im Trainingskorpus selten auftretende Begriffe und Ausdrücke gut zu repräsentieren. Demnach ist das `Skip-Gram-Modell` für das Training auf Dokumenten aus der Anforderungsdomäne besser geeignet, da das Trainingskorpus im Vergleich zu großen generischen Korpora wie dem Google-Nachrichten-Korpus eher geringen Umfang hat und die Trainingsgeschwindigkeit nicht so wichtig ist. Zudem ist es sehr gut, dass auch seltene Wörter gut repräsentiert werden, da in Anforderungen viele seltene Fachbegriffe vorkommen.

Nooralahzadeh et al. [NØL18] trainieren das `Continuous Bag-of-Words-Modell` und das `Skip-Gram-Modell` auf technischen Dokumenten und wissenschaftlichen Artikeln aus der Öl- und Gasdomäne und evaluieren anschließend beide. Die Evaluation erfolgt dabei mit intrinsischen Methoden, die auf den semantischen Beziehungen von Wörtern basieren. Nach ihren Ergebnissen liefert das `Continuous Bag-of-Words-Modell` mit den Standardparametern im Allgemeinen bessere Ergebnisse als das `Skip-Gram-Modell`. Da allerdings nur die Standardparameter verwendet wurden und somit etwaige Optimierungsmöglichkeiten nicht berücksichtigt wurden, sind diese Ergebnisse nicht sehr aussagekräftig.

Im Worteinbettungsmodell `GloVe` [PSM14] wird kein neuronales Netz eingesetzt. Im Gegensatz zu `word2vec` wird keine Pseudoaufgabe zur Vorhersage des Kontexts eines Wortes oder zur Vorhersage des Wortes aus einem Kontext eingesetzt. Stattdessen nutzt `GloVe` die Kookkurenz von Wörtern. Während `word2vec` also zu den kontextvorhersagenden Modellen gehört, kann man `GloVe` den kontextzählenden Modellen zuordnen.

Baroni et al. [BDK14] vergleichen die Wortvektoren von kontextzählenden Modellen mit denen von kontextvorhersagenden Modellen. Dabei werden viele mögliche Parameter berücksichtigt. Die Modelle werden an einer Vielzahl von Benchmarks getestet. Die kontextvorhersagenden Modelle erzielen in diesen Benchmarks mit meist großem Abstand klar bessere Ergebnisse. Obwohl nicht alle möglichen Parameter ausprobiert wurden, wurden genug Parameter berücksichtigt, um vertrauenswürdige Ergebnisse zu erhalten.

Auf Basis dieser Ergebnisse ist das kontextvorhersagende Modell `word2vec` dem kontextzählenden Modell `GloVe` vorzuziehen.

Wörter können in unterschiedlichen Kontexten unterschiedliche Dinge bedeuten. Das Worteinbettungsmodell `ELMo` [PNI+18] hat daher zum Ziel das Worteinbettungen die unterschiedlichen Kontexte der Wörter berücksichtigen. Dafür benutzt `ELMo` LSTMs mit mehreren Schichten, die die Eingaben bidirektional durchlaufen. Es ist jedoch zu berücksichtigen, dass dieser zentrale Vorteil von `ELMo`, die Berücksichtigung der unterschiedlichen Bedeutungen eines Wortes in unterschiedlichen Kontexten, beim Training auf einem Korpus bestehend aus Dokumenten aus der Anforderungsdomäne nur bedingt zum Tragen kommt. Da der Trainingskorpus nämlich nicht, wie es zum Beispiel bei einem generischen Trainingskorpus der Fall wäre, Texte aus vielen verschiedenen Domänen enthält, sondern sich auf die Anforderungsdomäne beschränkt, kommen Wörter oft nicht mit so vielen verschiedenen Bedeutungen vor.

Das Worteinbettungsmodell `fastText` [BGJM17] verwendet ein neuronales Netz, basierend auf dem `Skip-Gram-Modell` oder dem `Continuous Bag-of-Words-Modell`. Es kann

[2] Quelle: `https://groups.google.com/forum/#!searchin/word2vec-toolkit/c-bow/word2vec-toolkit/NLvYXU99cAM/E5ld8LcDxlAJ`, zuletzt besucht am 23.04.2020

somit als Erweiterung von `word2vec` betrachtet werden. Bei `fastText` werden jedoch die Wörter als mehrere N-Gramme sowie dem Wort selbst dargestellt, wobei zuvor die Zeichen „<“ und „>“ hinzugefügt werden. Diese Darstellungen der Wörter werden dann wie beim `Skip-Gram-Modell`, beziehungsweise wie beim `Continuous Bag-of-Words-Modell`, auf Vektoren abgebildet. Der Wortvektor eines Wortes ist dann durch die Summe der Vektoren seiner wie oben eingeführten Darstellung definiert.

Dadurch hat `fastText` den Vorteil, bessere Worteinbettungen für im Textkorpus selten vorkommende Wörter zu erzeugen. Denn schließlich können N-Gramme von selten vorkommenden Wörtern häufiger im Textkorpus auftreten und diese N-Gramme und ihre Kontexte werden beim Training der Worteinbettungen von `fastText` berücksichtigt.

Ebenso kann `fastText` durch seine Vorgehensweise im Vergleich zu den anderen Worteinbettungsmodellen aussagekräftigere Worteinbettungen für Wörter erzeugen, die im Trainingskorpus gar nicht auftreten. Der Wortvektor eines solchen Wortes wird einfach aus den Wortvektoren seiner N-Gramme, die im Gegensatz zum Wort im Trainingskorpus auftreten, gebildet. Zum Beispiel kann für das Wort *Fußball*, auch wenn es nicht im Trainingskorpus vorkommt, durch die Berücksichtigung von Teilwortinformationen ein relativ sinnvoller Wortvektor erzeugt werden, wenn die Wörter *Fuß* und *Ball* im Trainingskorpus auftreten.

Selten oder gar nicht im Trainingskorpus vorkommende Wörter sind genau eines der zentralen Probleme, wegen dem in dieser Bachelorarbeit überhaupt Worteinbettungen für die Anforderungsdomäne gebildet werden sollen. In Anforderungen treten nämlich häufig Fachbegriffe und sehr domänenspezifische Wörter auf. Diese Wörter kommen dann in einem generischen Textkorpus nur sehr selten oder gar nicht vor. Aber auch in einem domänenspezifischen Korpus, und sei er noch so umfangreich, kann dieses Problem nicht gänzlich ausgemerzt werden. Viele der Begriffe kommen nämlich nur in ein paar wenigen Anforderungen einer Subdomäne vor. Das sind zum Beispiel Begriffe, die nur in einem Unternehmen, oder nur in Anforderungen zu einem bestimmten Softwaretyp verwendet werden. Das domänenspezifische Korpus kann trotz großem Umfang nicht alle möglichen Bereiche abdecken.

Da der wichtigste Vorteil von `fastText` ist, genau das Problem der selten oder gar nicht im Trainingskorpus vorkommenden Wörter abzumildern, bietet sich das Worteinbettungsmodell sehr für das Training auf dem domänenspezifischen Korpus an.

Beim Training von Worteinbettungen mit `fastText` gibt es einige Parameter, die man anpassen kann, um Worteinbettungen von möglichst hoher Qualität zu erhalten.

Risch und Krestel [RK19] trainieren Worteinbettungen für die Patentdomäne mithilfe von `fastText`. Als Fenstergröße wird fünf gewählt und Wörter, die weniger als zehnmal vorkommen, werden beim Training nicht beachtet. Es werden insgesamt drei Modelle mit den Wortvektorraumdimensionen 100, 200 und 300 trainiert. Für die entstandenen Worteinbettungen wird die Genauigkeit in der Klassifizierung von Patenten untersucht. Dabei liefern die Worteinbettungen mit einer Dimension von 300 die besten Ergebnisse.

Je höher die Dimension, desto mehr Informationen können in den Wortvektoren abgebildet werden. Allerdings ist die benötigte Trainingszeit länger und der benötigte Speicherplatz größer. Da das aufgebaute Textkorpus nicht so groß ist, dass die Trainingszeit oder der Speicherplatz eine entscheidende Rolle spielen, bietet es sich an, entsprechend der Ergebnisse von Risch und Krestel als Wortvektorraumdimension 300 zu wählen. Dieser Wert wurde auch von Bojanowski et al. bei der Vorstellung des `fastText-Modells` [BGJM17] verwendet. Auch bei der Fenstergröße bietet es sich an, den von Risch und Krestel, sowie von Bojanowski et al. verwendeten Wert von fünf zu übernehmen.

Eines der Ziele dieser Bachelorarbeit ist, dass die Worteinbettungen für die Anforderungsdomäne auch sehr seltene Begriffe gut repräsentieren. Daher bietet es sich an, eine geringere Häufigkeit, mit der ein Wort im Textkorpus auftreten muss, um beim Training beachtet zu werden, zu wählen, als die Häufigkeit zehn von Risch und Krestel. Es kann nämlich durchaus sein, dass projektspezifische Begriffe nur in einem der Projekte geringeren Umfangs auftreten und somit bei Risch und Krestel nicht beim Training beachtet würden. Als mögliche geringere Häufigkeit bietet sich die von Bojanowski et al. gewählte Häufigkeit von fünf an.

Bevor das aufgebaute Textkorpus dem Worteinbettungsmodell übergeben wird, bietet es sich an, den Text vorzuverarbeiten. Es gibt verschiedene Methoden der Vorverarbeitung, die im Folgenden kurz vorgestellt werden.

Eine grundlegende Methode, deren Anwendung vor dem Training der Worteinbettungen häufig durchgeführt wird, ist das Kleinschreiben des gesamten Textes. Würde diese Methode nicht durchgeführt, würden Wörter, die im Korpus mal klein- und mal großgeschrieben werden, zum Beispiel weil sie am Satzanfang stehen, auf unterschiedliche Wortvektoren abgebildet werden. Dabei repräsentieren diese Wortvektoren jedoch eigentlich dasselbe Wort.

Eine weitere Vorverarbeitungsmethode, deren Anwendung vor dem Training der Worteinbettungen häufig durchgeführt wird, ist die Rauschentfernung (engl. *noise removal*). Dabei wird der Text gesäubert, indem für das Training der Worteinbettungen unnötige Textbestandteile wie zum Beispiel Kommas, Ausrufezeichen, Fragezeichen und Klammern entfernt werden.

Die Entfernung von Stoppwörtern wie Artikeln und Konjunktionen ist eine weitere Methode der Vorverarbeitung. Da solche Stoppwörter beim Training von Worteinbettungen meistens nicht hilfreich sind, im Gegenteil sogar ein unerwünschtes Rauschen darstellen, ist ihre Entfernung während der Vorverarbeitung oft ein sinnvoller Schritt.

Bei der Lemmatisierung werden Wörter auf ihre Grundform zurückgeführt. Ein Beispiel dafür wäre die Grundform *gehen* des Wortes *gehst*. Der Vorteil des Einsatzes dieser Methode vor dem Training der Worteinbettungen ist, dass das Problem von nur selten im Textkorpus auftretenden Begriffen abgemildert werden kann. Schließlich werden Begriffe, die dieselbe Grundform besitzen, im Textkorpus aber in unterschiedlichen Formen auftreten, ohne vorherige Lemmatisierung auf unterschiedliche Wortvektoren abgebildet. Wird während der Vorverarbeitung hingegen die Lemmatisierung durchgeführt, werden diese Begriffe auf dieselbe Grundform zurückgeführt, die anschließend im Trainingskorpus öfter vorkommt und auf einen Wortvektor abgebildet wird. Die Lemmatisierung hat allerdings den Nachteil, dass syntaktische Feinheiten verloren gehen können. So würden zum Beispiel die Wörter *gut* und *besser* bei Einsatz der Lemmatisierung während der Vorverarbeitung später auf denselben Wortvektor abgebildet. Zudem werden Worteinbettungen zu Wörtern mit derselben Grundform von Worteinbettungsmodellen so gelernt, dass die entsprechenden Wortvektoren nah beieinander liegen, was eine vorherige Lemmatisierung weniger notwendig macht. Der Einsatz der Lemmatisierung ist also nur dann sinnvoll, wenn das Trainingskorpus sehr klein ist und syntaktische Feinheiten keine wichtige Rolle spielen. Da aber das aufgebaute Textkorpus nicht sehr klein ist und syntaktische Feinheiten durchaus eine Rolle spielen, bietet sich die Vorverarbeitungsmethode der Lemmatisierung nicht besonders an.

5.4.1 Entwurf

Auf Basis der durchgeführten Analyse werden die Worteinbettungen für die Anforderungsdomäne mit dem Worteinbettungsmodell `fastText` trainiert. Dabei wird die `fastText`-Implementierung, die auf dem `Skip-Gram-Modell` basiert, verwendet.

Der Grund für diese Entscheidung ist, dass die Vorteile von `fastText` am besten zu den Problemen passen, die mit den domänenspezifischen Worteinbettungen behoben werden sollen, nämlich das der selten oder überhaupt nicht im Trainingskorpus vorkommenden Wörter. Der zentrale Vorteil von `ELMo`, die Berücksichtigung der unterschiedlichen Bedeutungen eines Wortes in unterschiedlichen Kontexten, ist hingegen für die Bildung von domänenspezifischen Worteinbettungen nicht ganz so relevant. Da `GloVe` als unter den betrachteten Modellen einziges kontextzählendes Worteinbettungsmodell nicht ganz so gute Resultate wie die kontextvorhersagenden Modelle rund um `fastText` liefert, wird es genau wie `ELMo` nicht verwendet. Weil `fastText` als Erweiterung von `word2vec` betrachtet werden kann, werden auch die Vor- und Nachteile des Modells übernommen. Deswegen, wird auch die `fastText`-Implementierung, die auf dem `Skip-Gram-Modell` basiert, verwendet. Schließlich hat die Analyse gezeigt, dass das `Skip-Gram-Modell` besser für das Training auf den Dokumenten aus der Anforderungsdomäne geeignet ist als das `Continuous Bag-of-Words-Modell`.

Beim Training der domänenspezifischen Worteinbettungen mit `fastText` gibt es einige Parameter, die man anpassen kann, um möglichst gute Worteinbettungen zu erhalten. Um möglichst viele Informationen in den Wortvektoren abzubilden, und da das aufgebaute Textkorpus nicht so groß ist, dass die Trainingszeit oder der Speicherplatz eine signifikante Rolle spielen, wird die Dimension der Wortvektoren auf 300 festgelegt. Für die Fenstergröße, genau wie für die Häufigkeit, mit der ein Wort im Textkorpus auftreten muss, um beim Training beachtet zu werden, liefert der Wert fünf gute Ergebnisse und wird daher für beides ausgewählt.

Bevor die Worteinbettungen trainiert werden, wird das aufgebaute Textkorpus vorverarbeitet. Da das aufgebaute Textkorpus zwar für ein domänenspezifisches Korpus recht umfangreich ist, aber trotzdem deutlich kleiner als große generische Korpora wie das Google-Nachrichten-Korpus, werden viele Vorverarbeitungsmethoden auf das Textkorpus angewandt. Das sind das Kleinschreiben aller Wörter, sowie das Entfernen von Rauschen und Stoppwörtern. Da jedoch das aufgebaute Textkorpus nicht sehr klein ist und syntaktische Feinheiten durchaus eine Rolle spielen, wird keine Lemmatisierung auf das Textkorpus angewandt.

5.5 Kombination von Worteinbettungen

Im Folgenden wird untersucht, welche Möglichkeiten es gibt, um die domänenspezifischen auf dem aufgebauten Textkorpus trainierten Worteinbettungen mit anderen auf umfangreichen generischen Korpora trainierten Worteinbettungen zu kombinieren. Die verschiedenen Möglichkeiten werden auf ihre Vor- und Nachteile und bestehende Worteinbettungen auf ihre Eignung untersucht.

Das Ziel der Kombination ist es, den größeren Umfang der generischen Worteinbettungen zu nutzen. Dabei besteht jedoch die Gefahr, dass domänenspezifische Feinheiten und Bedeutungen, die in den auf dem aufgebauten Textkorpus trainierten Worteinbettungen erfasst wurden, verloren gehen. Ein Wort wie *Fenster* zum Beispiel kann im domänenspezifischen Korpus eine andere Bedeutung haben als im generischen, schließlich kann mit dem Begriff sowohl das Zeitfenster als auch die Lichtöffnung gemeint sein. Eine mögliche Eigenschaft von Kombinationsmethoden, die dabei hilft, dieses Ziel möglichst gut zu erreichen, ist die Möglichkeit der unterschiedlichen Gewichtung der domänenspezifischen und generischen Worteinbettungen. Die Gewichtung kann dann so gewählt werden, dass der größere Umfang der generischen Worteinbettungen genutzt wird, ohne zu viel domänenspezifische Feinheiten und Bedeutungen zu verlieren. Die aus der Kombination von generischen und domänenspezifischen Worteinbettungen entstehenden Worteinbettungen werden im Folgenden domänenadaptierte Worteinbettungen genannt.

Kameswara Sarma et al. [KSLS18] stellen eine Methode zur Kombination von generischen und domänenspezifischen Worteinbettungen vor. Die domänenadaptierten Worteinbettungen sollen sowohl den Umfang der generischen als auch die Spezifität der domänenspezifischen Worteinbettungen als Eigenschaften aufweisen. Die domänenadaptierten Worteinbettungen erzielen in Aufgaben aus der Domäne bessere Ergebnisse als die generischen oder die domänenspezifischen Worteinbettungen. Somit bietet sich an, die auf Dokumenten aus der Anforderungsdomäne trainierten Worteinbettungen nach dem Ansatz von Kameswara Sarma et al. mit generischen Worteinbettungen zu kombinieren. Die generischen und die domänenspezifischen Worteinbettungen werden kombiniert, indem eine lineare CCA oder eine nichtlineare KCCA eingesetzt wird. So werden für ein Wort die entsprechenden generischen und domänenspezifischen Wortvektoren in die Richtung der höchsten Korrelation projiziert. In der Arbeit von Kameswara Sarma et al. liefert die nichtlineare KCCA im Allgemeinen bessere Ergebnisse und bietet sich daher besonders an. Die durch die CCA erhaltenen projizierten Worteinbettungen werden nach einem Optimierungsansatz linear kombiniert. In der Arbeit von Kameswara Sarma et al. wird dabei der Ansatz des geringsten Abstandes der domänenadaptierten Worteinbettungen zu sowohl den domänenspezifischen als auch den generischen Worteinbettungen verfolgt. Somit werden die projizierten Worteinbettungen linear kombiniert, indem sie halbiert und addiert werden. Es ist allerdings leicht möglich, stattdessen andere Gewichtungen der domänenspezifischen und der generischen Worteinbettungen vorzunehmen. So können später verschiedene Gewichtungen ausprobiert und die beste ausgewählt werden.

Neben der Kombinationsmethode von Kameswara Sarma et al. gibt es noch einige andere mögliche Kombinationsmethoden, die jedoch nicht speziell zur Kombination von domänenspezifischen und generischen Worteinbettungen geschaffen wurden, sondern zum Beispiel zur Kombination von verschiedenen generischen Worteinbettungen. Trotzdem können auch solche Kombinationsmethoden für die Kombination domänenspezifischer und generischer Worteinbettungen verwendet werden.

Ghannay et al. [GFEC16] vergleichen die Qualität verschiedener Worteinbettungen und untersuchen mögliche Methoden zur Kombination. Sie stellen fest, dass die verschiedenen Worteinbettungsmodelle je nach Aufgabe unterschiedlich gute Ergebnisse liefern. Um die Vorteile der verschiedenen Modelle je nach Aufgabe zu vereinen, werden die Kombinationsmethoden `Concat`, `PCA` und `AutoE` untersucht. Die Kombinationsmethode `Concat` konkateniert die Worteinbettungen. Bei der Kombinationsmethode `PCA` wird eine Hauptkomponentenanalyse auf die konkatenierten Worteinbettungen angewendet. Die Kombinationsmethode `AutoE` nutzt einen Autoencoder, also ein neuronales Netz mit den konkatenierten Worteinbettungen als Eingabe. Die Kombinationsmethode `Concat` geht sehr unkompliziert vor, die zu kombinierenden Wortvektoren werden einfach aneinandergehängt. Ein Nachteil davon ist, dass die Dimension der konkatenierten Wortvektoren dem Ergebnis der Summe der Dimensionen der zu kombinierenden Wortvektoren entspricht und damit sehr hoch werden kann. Die Kombinationsmethoden `PCA` und `AutoE` nutzen die konkatenierten Worteinbettungen als Eingabe und liefern Wortvektoren mit geringerer Dimension als die konkatenierten Worteinbettungen. Allerdings liefert die Kombinationsmethode `PCA`, trotz des sehr simplen Ansatzes der Kombinationsmethode `Concat`, keine besseren Ergebnisse als ebenjene, und die Kombinationsmethode `AutoE` liefert sogar klar schlechtere Ergebnisse. Allgemein sind die Ergebnisse aller drei Kombinationsmethoden nicht signifikant besser als die der ursprünglichen Worteinbettungen. Eine mögliche unterschiedliche Gewichtung der zu kombinierenden Worteinbettungen ist bei den drei Kombinationsmethoden nicht vorgesehen.

Yin und Schütze [YS16] stellen die Kombinationsmethoden `CONC`, `SVD`, `1TON` und `1TON`$^+$ vor. Bei der Kombinationsmethode `CONC` werden die Worteinbettungen konkateniert, wobei die einzelnen Mengen der zu kombinierenden Worteinbettungen unterschiedlich stark

gewichtet werden können. Die Kombinationsmethode `SVD` verwendet zur Kombination eine Singulärwertzerlegung. Bei der Kombinationsmethode `1TON` werden die kombinierten Worteinbettungen von einem neuronalen Netz gelernt. Das neuronale Netz erhält die kombinierten Worteinbettungen als Eingabe und die einzelnen Mengen der zu kombinierenden Worteinbettungen als Ausgabe. Die Kombinationsmethode `1TON`$^+$ stellt eine Erweiterung der Kombinationsmethode `1TON` dar, bei der kombinierte Worteinbettungen für alle Wörter gebildet werden, zu denen mindestens eine Worteinbettung in den zu kombinierenden Worteinbettungen existiert. Die Kombinationsmethoden werden anhand verschiedener Aufgaben zu Wortähnlichkeiten, zu Wortanalogien und zur Wortartmarkierung evaluiert. Je nach Aufgabe liefert eine andere Kombinationsmethode die besten Ergebnisse. Werden die Ergebnisse der Kombinationsmethoden in einer Aufgabe mit den Ergebnissen der Menge zu kombinierender Worteinbettungen, die in dieser Aufgabe die besten Ergebnisse liefert, verglichen, so lassen sich für die meisten Aufgaben keine deutlichen Verbesserungen feststellen. Eine mögliche unterschiedliche Gewichtung der zu kombinierenden Worteinbettungen ist im Gegensatz zu den Kombinationsmethoden `Concat`, `PCA` und `AutoE` von Ghannay et al. vorgesehen.

Coates und Bollegala [CB18] untersuchen die Kombination von Worteinbettungen durch Bildung des Durchschnitts. Vor der Durchschnittsbildung werden die jeweiligen Worteinbettungen normalisiert. Coates und Bollegala zeigen, dass, obwohl die Wortvektorräume der zu kombinierenden Worteinbettungen nicht vergleichbar sind, semantische Informationen der zu kombinierenden Worteinbettungen bei der Durchschnittsbildung erhalten bleiben. Die relativen Entfernungen zwischen den Wortvektoren bleiben nämlich bei der Durchschnittsbildung erhalten. Die kombinierten Worteinbettungen werden anhand Aufgaben zu Wortähnlichkeiten und zu Wortanalogien evaluiert. Dabei werden die Ergebnisse mit den Ergebnissen der durch Konkatenation erhaltenen Worteinbettungen verglichen. Insgesamt sind die Ergebnisse der durch Durchschnittsbildung und der durch Konkatenation erhaltenen Worteinbettungen recht ähnlich, in manchen Fällen sind die Ergebnisse der durch Durchschnittsbildung erhaltenen Worteinbettungen etwas besser. Statt der ursprünglich verwendeten gleich starken Gewichtung der domänenspezifischen und generischen Worteinbettungen nach der Normalisierung ist es auch möglich, andere Gewichtungen vorzunehmen und die Wortvektoren dann zu addieren, also den gewichteten Durchschnitt zu berechnen.

Weder bei den drei Kombinationsmethoden `Concat`, `PCA` und `AutoE` von Ghannay et al., noch bei den vier Kombinationsmethoden `CONC`, `SVD`, `1TON` und `1TON`$^+$ von Yin und Schütze kann sich eine Methode konsequent von der Konkatenation absetzen. Wie von Coates und Bollegala beschrieben liefert die Durchschnittsbildung ähnliche oder bessere Ergebnisse als die Konkatenation und hat zudem den Vorteil der geringeren Dimensionalität. Laut Coates und Bollegala sind die durch Durchschnittsbildung erhaltenen Worteinbettungen vergleichbar oder besser als die durch komplexere Kombinationsmethoden entstehenden Worteinbettungen. Ein Beispiel für eine komplexere Kombinationsmethode ist die von Kameswara Sarma et al. vorgestellte Methode. Daher bietet es sich an, neben der Methode von Kameswara Sarma et al. auch die Methode von Coates und Bollegala zu verwenden und dann zu vergleichen, welche Methode die besseren Ergebnisse liefert.

Für die generischen Worteinbettungen bietet sich an, wie auch bei den domänenspezifischen Worteinbettungen das `fastText-Modell` zu verwenden. So ist es durch Kombination der Teilwortvektoren möglich, dass auch mit den domänenadaptierten Worteinbettungen Wortvektoren für nicht im Vokabular enthaltene Wörter gebildet werden können. Zudem gilt, dass wenn für die generischen und die domänenspezifischen Worteinbettungen dasselbe Worteinbettungsmodell verwendet wird, die Wortvektorräume in ihrer Struktur oft mehr übereinstimmen, als wenn verschiedene Worteinbettungsmodelle verwendet wurden. Das kann die Kombination verbessern.

Es gibt verschiedene vortrainierte `fastText-Modelle`, die unterschiedlich groß sind und auf unterschiedlich großen Textkorpora trainiert wurden. Das Modell, das am größten ist und das auf dem größten Textkorpus trainiert wurde, bietet zwei Millionen Wortvektoren mit 300 Dimensionen [MGB+18]. Es wurde auf einem Korpus von `Common Crawl` bestehend aus insgesamt 630 Milliarden Wörtern trainiert. Dabei wurde das `Continuous Bag-of-Words-Modell` mit Berücksichtigung von Teilwortinformationen verwendet. Da die Qualität der generischen Worteinbettungen für die Kombination wichtig und die Berechnungszeit für diese Arbeit zu vernachlässigen ist, bietet es sich an, das Modell, das am größten ist und das auf dem größten Textkorpus trainiert wurde, zur Kombination zu verwenden.

5.5.1 Entwurf

Entsprechend der Analyse zur Kombination von Worteinbettungen werden die zuvor auf dem aufgebauten Textkorpus von Dokumenten aus der Anforderungsdomäne trainierten Worteinbettungen mit generischen Worteinbettungen kombiniert. Dabei werden zwei verschiedene Kombinationsmethoden ausprobiert.

Die erste Methode von Kameswara Sarma et al. ist im Gegensatz zur zweiten Methode komplexer und wurde speziell zur Kombination von domänenspezifischen und generischen Worteinbettungen geschaffen. Es wird zuerst eine CCA und anschließend eine lineare Kombination der projizierten Worteinbettungen durchgeführt. Aufgrund der laut der Arbeit von Kameswara Sarma et al. besseren Ergebnisse wird für die CCA eine nichtlineare KCCA eingesetzt. Die lineare Kombination erfolgt bei Kameswara Sarma et al. durch Halbieren und Addieren der projizierten Worteinbettungen. Es können jedoch auch andere Gewichtungen vorgenommen werden. Das Ziel bei der Festlegung der Gewichtung ist, dass der größere Umfang der generischen Worteinbettungen genutzt wird, ohne zu viel domänenspezifische Feinheiten und Bedeutungen zu verlieren.

Bei der zweiten Methode werden die Wortvektoren entsprechend der Arbeit von Coates und Bollegala normalisiert und der Durchschnitt gebildet. Statt der gleich starken Gewichtung der domänenspezifischen und generischen Worteinbettungen nach der Normalisierung ist es auch möglich, andere Gewichtungen vorzunehmen und den gewichteten Durchschnitt zu berechnen.

In der Evaluation wird eine der zwei Methoden ausgewählt, sowie die genaue Gewichtung festgelegt, indem die aus den zwei Methoden und verschiedenen Gewichtungen entstandenen Worteinbettungen evaluiert und die Ergebnisse verglichen werden.

Für die generischen Worteinbettungen wird das vortrainierte `fastText-Modell`, das am größten ist und das auf dem größten Textkorpus trainiert wurde, verwendet, da die Berechnungszeit für diese Arbeit zu vernachlässigen ist und die Qualität der Worteinbettungen eine höhere Bedeutung hat. Dieses Modell mit zwei Millionen 300-dimensionalen Wortvektoren wurde auf einem `Common Crawl`-Korpus bestehend aus insgesamt 630 Milliarden Wörtern trainiert. Dabei wurde das `Continuous Bag-of-Words-Modell` mit Berücksichtigung von Teilwortinformationen verwendet.

5.6 Zusammenfassung des Entwurfs

Nachdem das Textkorpus von Dokumenten aus der Anforderungsdomäne aufgebaut wurde, wird er vorverarbeitet. Dafür werden Methoden zum Kleinschreiben aller Wörter, sowie zum Entfernen von Rauschen und Stoppwörtern eingesetzt.

Dann werden die Worteinbettungen für die Anforderungsdomäne mit dem Worteinbettungsmodell `fastText` auf dem vorverarbeiteten Textkorpus trainiert. Dabei wird die

fastText-Implementierung, die auf dem Skip-Gram-Modell basiert, verwendet. Als Dimension der Wortvektoren wird 300 gewählt. Die Fenstergröße wird auf fünf gesetzt, genau wie die Häufigkeit, mit der ein Wort im Textkorpus auftreten muss, um beim Training beachtet zu werden.

Die auf dem Textkorpus trainierten Worteinbettungen werden mit generischen Worteinbettungen kombiniert. Dabei werden zwei verschiedene Kombinationsmethoden ausprobiert. Bei der ersten Methode wird nach der Arbeit von Kameswara Sarma et al. zuerst eine nichtlineare KCCA und anschließend eine lineare Kombination der projizierten Worteinbettungen durchgeführt, wobei verschiedene Gewichtungen vorgenommen werden können. Bei der zweiten Methode werden die Wortvektoren nach der Arbeit von Coates und Bollegala normalisiert, gewichtet und addiert. In der Evaluation wird eine der zwei Methoden ausgewählt, sowie die genaue Gewichtung festgelegt, indem die aus den zwei Methoden und verschiedenen Gewichtungen entstandenen Worteinbettungen evaluiert und die Ergebnisse verglichen werden.

Für die generischen Worteinbettungen wird das vortrainierte fastText-Modell, das am größten ist und das auf dem größten Textkorpus trainiert wurde, verwendet. Es hat zwei Millionen Wortvektoren und wurde auf einem Korpus von Common Crawl bestehend aus insgesamt 630 Milliarden Wörtern trainiert.

6 Implementierung

Das aufgebaute Textkorpus von Anforderungsbeschreibungen und anderen Dokumenten aus der Anforderungsdomäne liegt als CSV-Datei in UTF-8-Kodierung vor. Er wird im ersten Schritt mit mehreren Methoden vorverarbeitet. So werden alle Wörter kleingeschrieben und Rauschen, also für das Training der Worteinbettungen unnötige Textbestandteile, entfernt. Das Rauschen, das gelöscht wird, umfasst Sonderzeichen wie Punkte, Kommas, Fragezeichen und Klammern. Zudem werden Stoppwörter wie Artikel und Konjunktionen entfernt. Zur Vorverarbeitung wird das `Natural Language Tool Kit` (NLTK) [BL04] eingesetzt. Das NLTK ist eine Python-Bibliothek für Aufgaben aus dem Bereich der Verarbeitung natürlicher Sprache. Die Liste von Stoppwörtern, die vom NLTK bereitgestellt wird und die aus dem Textkorpus entfernt wird, findet sich in Abschnitt C des Anhangs. Sie enthält Begriffe wie *the*, *and* und *to*. In Beispiel 6.1 wird die Vorverarbeitung eines beispielhaften Satzes aus dem aufgebautem Textkorpus gezeigt. Alle Wörter aus dem Satz wurden dabei kleingeschrieben. Zudem wurden die Sonderzeichen „-“, „/“ und „.“ gelöscht und die Stoppwörter *the*, *a* und *of* entfernt.

Beispiel 6.1:

Der beispielhafte Satz

The DPU-BOOT CSC shall include a DRAM BIT consisting of two write/read/compare tests.

aus dem Textkorpus wird durch die Vorverarbeitung in den Text

dpu boot csc shall include dram bit consisting two write read compare tests

umgewandelt.

Zum Training der Worteinbettungen auf dem vortrainierten Textkorpus wird das Worteinbettungsmodell `fastText` [BGJM17] verwendet. Dabei wird das offizielle `fastText`-Modul[1] eingesetzt. Es handelt sich hierbei um eine Python-Bibliothek. Das `fastText`-Modul verlangt, dass das Textkorpus, auf dem das Modell trainiert werden soll, in UTF-8-Kodierung vorliegen muss. Das ist für das aufgebaute Textkorpus schon der Fall, womit eine Konvertierung nicht notwendig ist. Beim Training wird die auf dem `Skip-Gram-Modell`

[1] Quelle: `https://github.com/facebookresearch/fastText`, zuletzt besucht am 23.04.2020

basierende `fastText`-Implementierung verwendet. Die Fenstergröße wird auf fünf und die Dimension der Wortvektoren auf 300 festgelegt. Zudem wird die minimale Anzahl von Vorkommen eines Wortes, sodass es beim Training berücksichtigt wird, auf fünf gesetzt. Die Größe des Vokabulars der trainierten domänenspezifischen Worteinbettungen, also die Anzahl der Wortvektoren, beträgt 4230. Allerdings kann das `fastText-Modell` wie eingeführt durch die Verwendung von Teilwörtern auch Wortvektoren für Wörter außerhalb des Vokabulars bilden.

Zum Schluss werden die domänenspezifischen Worteinbettungen mit generischen Worteinbettungen kombiniert. Für die generischen Worteinbettungen wird das vortrainierte `fastText-Modell`, das am größten ist und das auf dem größten Textkorpus trainiert wurde, verwendet. Dieses Modell[2] mit zwei Millionen 300-dimensionalen Wortvektoren wurde auf einem `Common Crawl`-Korpus bestehend aus insgesamt 630 Milliarden Wörtern trainiert [MGB+18]. Dabei wurde das `Continuous Bag-of-Words-Modell` mit Berücksichtigung von Teilwortinformationen verwendet. Es werden zwei verschiedene Kombinationsmethoden ausprobiert.

Bei der ersten Kombinationsmethode wird nach der Arbeit von Kameswara Sarma et al. [KSLS18] eine nichtlineare KCCA und anschließend eine lineare Kombination durchgeführt. Kameswara Sarma et al. haben für ihre Kombinationsmethode das Grundgerüst eines Python-Skripts[3] bereitgestellt, dass im Rahmen der Implementierung vervollständigt und anschließend eingesetzt wird. Der Teil des Skripts zur nichtlinearen KCCA basiert dabei auf `Pyrcca` [BG16]. Das ist eine Python-Bibliothek, die zur Durchführung einer CCA genutzt werden kann.

Bei der zweiten Kombinationsmethode werden die Wortvektoren nach der Arbeit von Coates und Bollegala L2-normalisiert, gewichtet und addiert. Für die L2-Normalisierung wird dabei die Python-Bibliothek `Scikit-learn` [PVG+11] verwendet.

Die domänenadaptierten Worteinbettungen haben eine Vokabulargröße von 2 000 128, zu den zwei Millionen Wortvektoren der generischen sind also noch einmal 128 neue dazugekommen.

[2] Quelle: `https://fasttext.cc/docs/en/english-vectors.html`, zuletzt besucht am 23.04.2020
[3] Quelle: `https://github.com/prtsk/DomainAdaptedWordEmbeddings`, zuletzt besucht am 23.04.2020

7 Evaluation

In Abschnitt 7.1 sollen die Unterschiede sowie die Vor- und Nachteile der domänenspezifischen, also der auf dem Textkorpus von Dokumenten aus der Anforderungsdomäne trainierten Worteinbettungen gegenüber generischen Worteinbettungen bemessen werden. Dazu werden die domänenspezifischen Worteinbettungen analysiert, mit generischen Worteinbettungen verglichen und evaluiert. Als generische Worteinbettungen werden dabei die vortrainierten Worteinbettungen verwendet, mit denen die domänenspezifischen kombiniert werden.

In Abschnitt 7.2 soll die beste Kombination der domänenspezifischen und generischen Worteinbettungen gefunden werden und ihre Qualität im Vergleich zu den domänenspezifischen sowie den generischen Worteinbettungen bemessen werden. Dazu werden zuerst verschiedene Kombinationen der domänenspezifischen und generischen Worteinbettungen anhand einer Aufgabe zur Klassifizierung von Sätzen in Anforderungsbeschreibungen extrinsisch evaluiert und die beste ausgewählt. Diese domänenadaptierten Worteinbettungen bilden somit die Worteinbettungen für die Anforderungsdomäne. Die Qualität ihrer in der Klassifizierungsaufgabe erhaltenen Ergebnisse wird mit der Qualität der Ergebnisse verglichen, die durch den Einsatz der generischen, beziehungsweise der domänenspezifischen Worteinbettungen in derselben Aufgabe entstehen.

7.1 Domänenspezifische Worteinbettungen

Die domänenspezifischen Worteinbettungen werden im Folgenden mithilfe einer Analyse von Wortähnlichkeiten und Clustern intrinsisch evaluiert und mit den generischen Worteinbettungen verglichen.

7.1.1 Wortähnlichkeiten

Eine Möglichkeit, die Qualität von Worteinbettungen einzuschätzen, ist die Betrachtung der Wortvektoren, die dem Wortvektor zu einem ausgewählten Wort am ähnlichsten sind. So lässt sich stichprobenhaft feststellen, welcher Art die in den Worteinbettungen gespeicherten semantischen Regelmäßigkeiten sind. Es soll für ausgewählte Begriffe analysiert werden, welche Unterschiede es in den ähnlichsten Wortvektoren zwischen den domänenspezifischen und den generischen Worteinbettungen gibt, und wie sinnvoll die ähnlichsten Wortvektoren in den domänenspezifischen Worteinbettungen sind, mit besonderem Augenmerk auf die Anwendung der Worteinbettungen in Aufgaben aus der Anforderungsdomäne.

Tabelle 7.1: Die zehn Wörter, die *system* in den generischen Worteinbettungen ohne Vorauswahl am ähnlichsten sind

Wort	Ähnlichkeit
sytem	0,8590
systems	0,8127
sysytem	0,7529
system-and	0,7448
system--and	0,7333
system.It	0,7237
systen	0,7175
system--a	0,7094
system.That	0,7058
systme	0,7014

Als Maß für die Ähnlichkeit von Wortvektoren wird die Kosinus-Ähnlichkeit verwendet. Die Ähnlichkeit zweier Vektoren wird dabei durch den Kosinus des Winkels zwischen ihnen bestimmt. Im Folgenden ist mit der Ähnlichkeit von Wörtern die Ähnlichkeit ihrer Wortvektoren gemeint.

Es werden Wörter untersucht, die häufig im aufgebauten Textkorpus auftreten und somit typisch für die Anforderungsdomäne sind, und bei denen man davon ausgehen kann, dass sie sich in ihrer Bedeutung in Anforderungen und in generischen Texten unterscheiden. Der Grund dafür ist, dass es für die Anwendung der domänenspezifischen Worteinbettungen in Aufgaben aus der Anforderungsdomäne besonders wichtig ist, dass Wörter, die typisch für die Anforderungsdomäne sind, gut repräsentiert werden. Zudem sollen so Unterschiede zwischen den domänenspezifischen und den generischen Worteinbettungen deutlich gemacht werden.

Für jedes der ausgewählten Wörter werden die zehn ähnlichsten Wörter in den domänenspezifischen und die zehn ähnlichsten Wörter in den generischen Worteinbettungen betrachtet. Da das Trainingskorpus der generischen Worteinbettungen weniger umfassend vorverarbeitet wurde als das Trainingskorpus der domänenspezifischen Worteinbettungen, sind in den zehn ähnlichsten Wörtern in den generischen Worteinbettungen häufig falsch geschriebene oder Satzzeichen enthaltende Wörter vorhanden. In Tabelle 7.1 sind zum Beispiel die zehn ähnlichsten Wörter in den generischen Worteinbettungen zum Wort *system* aufgeführt. Um einen aussagekräftigeren Vergleich durchführen zu können, wird im Folgenden bei den ähnlichsten Wörtern in den generischen Worteinbettungen eine entsprechende Vorauswahl durchgeführt. Dabei werden die Begriffe ausgewählt, die aus einem, möglicherweise zusammengesetzten Wort bestehen, richtig geschrieben sind und keine Großbuchstaben oder Satzzeichen wie Punkte enthalten.

Die Richtung von einem Wort zu einem anderen kann in Worteinbettungen einen Anhaltspunkt dafür liefern, in welcher Beziehung diese Wörter zueinander stehen [MYZ13]. Haben daher in Worteinbettungen für einen ausgewählten Begriff zwei Wörter mit gleicher Beziehung zum Begriff ungefähr dieselbe Richtung, kann das als positive Aussage über die Qualität der Worteinbettungen verstanden werden. Um die Richtungen der ausgewählten Begriffe zu ihren ähnlichsten Wörtern darzustellen, wird eine Hauptkomponentenanalyse durchgeführt, die die Wortvektoren in einen zweidimensionalen Vektorraum abbildet.

Das erste Wort, das im Folgenden betrachtet wird, ist das Wort *data*. Die zehn Wörter, die dem Wort *data* in den domänenspezifischen, beziehungsweise in den generischen Wor-

teinbettungen am ähnlichsten sind, sind in Tabelle 7.2 und Abbildung 7.1 dargestellt. Dabei entsprechen die Richtungen des Wortes *data* zu seinen ähnlichsten Wörtern in Abbildung 7.1 den entsprechenden Richtungen im durch die Hauptkomponentenanalyse erhaltenen zweidimensionalen Vektorraum. In Tabelle 7.2 sind zudem die Unterschiede der Ähnlichkeit im Vergleich mit den jeweils anderen Worteinbettungen dargestellt, das heißt für positive Differenzwerte ist die Ähnlichkeit im Vergleich zu den anderen Worteinbettungen größer. Ist ein Wort dabei nicht im Vokabular der anderen Worteinbettungen enthalten, so ist der entsprechende Wert eingeklammert. Es fällt auf, dass die Ähnlichkeit von *data* zum ähnlichsten Wort in den domänenspezifischen Worteinbettungen mit ungefähr 0,54 relativ gering ist. Das liegt daran, dass das Wort *data* in Anforderungen recht konsequent verwendet wird, was heißt, dass im Trainingskorpus keine Synonyme gleichwertig verwendet werden. Zudem wird der Singular *datum*, der sonst wohl eine relativ große Ähnlichkeit hätte, nur sehr selten verwendet. Dass die ähnlichsten Wörter die Wörter *metadata*, *database* und *databases* sind, ist nachvollziehbar. Hier zeigt sich der Vorteil von `fastText`. Diese drei Wörter enthalten nämlich alle das Teilwort *data*, was beim Training der Worteinbettungen berücksichtigt wurde. In den generischen Worteinbettungen werden diese drei Wörter sogar noch mit etwas größeren Ähnlichkeiten dargestellt. Die Wortpaare bestehend aus *database* und *databases*, sowie *datapackage* und *datasets* aus den domänenspezifischen Worteinbettungen stehen jeweils in derselben Beziehung zu *data*, sollten also jeweils ungefähr dieselbe Richtung haben. Das ist allerdings nur für das letztere Paar der Fall. Relativierend muss angemerkt werden, dass die durch die Hauptkomponentenanalyse erhaltenen Richtungen nur eine Annäherung sind. In zwei Dimensionen können schließlich nicht alle Informationen aus den 300 Dimensionen der Wortvektoren erfasst werden. Das in den generischen Worteinbettungen ähnlichste Wort ist *information*. Es wird mit einer um ungefähr 0,36 größeren Ähnlichkeit in den generischen Worteinbettungen besser dargestellt. Die ähnlichsten Wörter in den generischen Worteinbettungen, die Sonderzeichen wie einen Bindestrich enthalten, sind aufgrund der Vorverarbeitung des Trainingskorpus nicht im Vokabular der domänenspezifischen Worteinbettungen enthalten. Ein Beispiel dafür ist das Wort *user-data*, das im Trainingskorpus der domänenspezifischen Worteinbettungen in die Wörter *user* und *data* aufgeteilt wurde. Für solche zusammengesetzte Wörter kann das domänenspezifische Modell aber naturgemäß recht gute Wortvektoren aus der Summe der Vektoren der Teilwörter bilden. Somit ist die Ähnlichkeit für *user-data* in den generischen Worteinbettungen nur um ungefähr 0,02 größer. Es lässt sich für das Wort *data* insgesamt feststellen, dass es in den domänenspezifischen Worteinbettungen zwar sinnvoll dargestellt wird, die generischen Worteinbettungen es aber noch etwas besser erfassen. Zudem wird das Wort in Anforderungen und generischen Texten recht ähnlich verwendet, weswegen sich zwischen den generischen und den domänenspezifischen Worteinbettungen keine größeren Unterschiede in der erfassten Bedeutung des Wortes feststellen lassen.

Als Nächstes wird das Wort *system* betrachtet. Die Wörter, die dem Wort *system* am ähnlichsten sind, sind in Tabelle 7.3 und Abbildung 7.2 dargestellt. Mit Abstand am ähnlichsten in den domänenspezifischen Worteinbettungen ist der Plural *systems* mit einer Ähnlichkeit von ungefähr 0,74. Da die Ähnlichkeit von Singular und Plural keine besonders domänenspezifische Eigenschaft ist, sind sich *system* und *systems* auch in den generischen Worteinbettungen ähnlich. Es ist sogar genau wie in den domänenspezifischen Worteinbettungen das mit Abstand ähnlichste Wort. Aufgrund des größeren Trainingskorpus, in dem *system* und *systems* oft in einem ähnlichen Kontext vorkommen, ist die Ähnlichkeit in den generischen Worteinbettungen etwas größer, um ungefähr 0,08. Dann folgen in den domänenspezifischen Worteinbettungen mit dem Begriff *subsystem*, seinem Plural, und dem Begriff *filesystem* verschiedene Arten und nähere Bezeichnungen von Systemen. Um zu überprüfen, wie gut die Beziehung von System zu Systemarten in den Worteinbettungen dargestellt wird, werden die Richtungen der drei Wörter betrachtet. Die drei Begriffe befinden sich alle in einer ähnlichen Richtung des Wortes *system*, was darauf hindeutet,

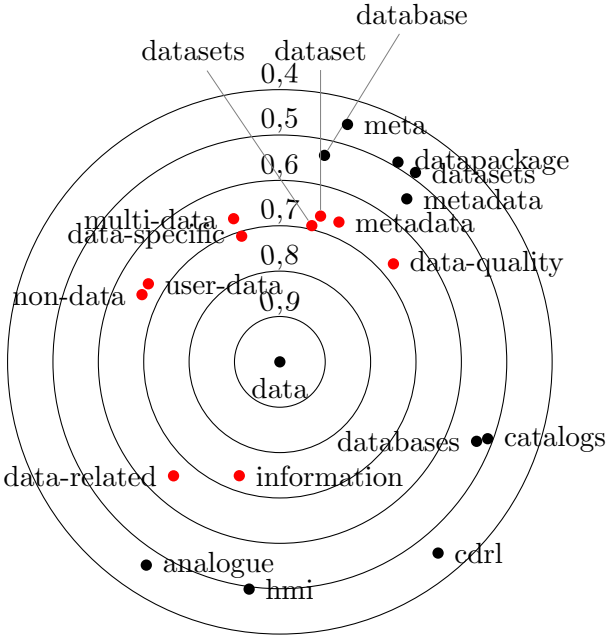

Abbildung 7.1: Darstellung der Ähnlichkeiten und Richtungen der zehn Wörter, die *data* in den domänenspezifischen (schwarz), bzw. generischen (rot) Worteinbettungen am ähnlichsten sind

Wort	Ähnl.	Diff.
metadata	0,5447	−0,1209
database	0,5343	−0,1144
databases	0,5325	−0,1225
catalogs	0,5117	0,0824
hmi	0,4945	0,3037
datapackage	0,4886	(0,0125)
datasets	0,4866	−0,2050
analogue	0,4639	0,0648
meta	0,4559	0,0110
cdrl	0,4530	(0,2124)
information	0,7336	0,3624
data-specific	0,7105	(0,2433)
datasets	0,6916	0,2050
data-quality	0,6694	(0,1551)
multi-data	0,6684	(0,0649)
dataset	0,6664	0,2180
metadata	0,6655	0,1209
user-data	0,6630	(0,0189)
non-data	0,6620	(−0,1069)
data-related	0,6565	(0,1041)

Tabelle 7.2: Die zehn Wörter, die *data* in den domänenspezifischen (oben), bzw. generischen (unten) Worteinbettungen am ähnlichsten sind, mit der Ähnlichkeit sowie dem Ähnlichkeitsunterschied zu den jeweils anderen Worteinbettungen

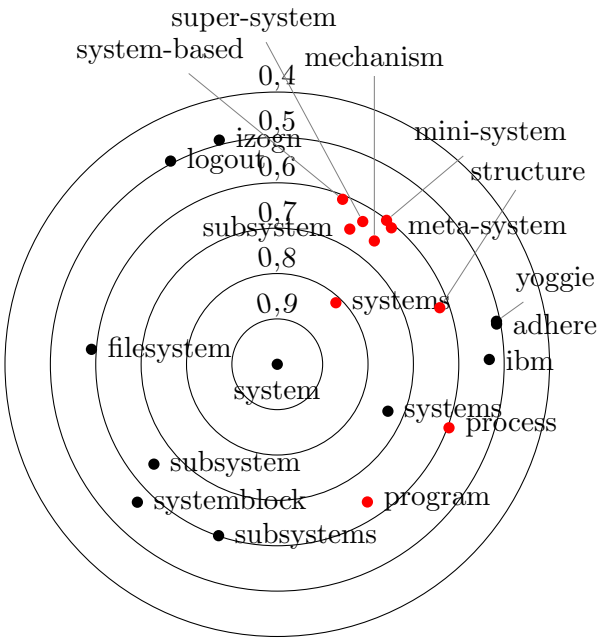

Abbildung 7.2: Darstellung der Ähnlichkeiten und Richtungen der zehn Wörter, die *system* in den domänenspezifischen (schwarz), bzw. generischen (rot) Worteinbettungen am ähnlichsten sind

Wort	Ähnl.	Diff.
systems	0,7352	−0,0775
subsystem	0,6502	−0,0117
subsystems	0,6007	0,02724
filesystem	0,5892	0,1870
systemblock	0,5670	(0,1358)
ibm	0,5326	0,3534
adhere	0,5087	0,2472
yoggie	0,5077	(0,1947)
logout	0,4941	0,1799
izogn	0,4896	(0,2308)
systems	0,8127	0,0775
subsystem	0,6619	0,0117
mechanism	0,6541	0,2836
program	0,6373	0,3798
super-system	0,6333	(−0,0570)
structure	0,6209	0,4477
system-based	0,6089	(−0,1650)
meta-system	0,6079	(−0,2136)
mini-system	0,6017	(−0,2591)
process	0,5965	0,3039

Tabelle 7.3: Die zehn Wörter, die *system* in den domänenspezifischen (oben), bzw. generischen (unten) Worteinbettungen am ähnlichsten sind, mit der Ähnlichkeit sowie dem Ähnlichkeitsunterschied zu den jeweils anderen Worteinbettungen

dass sie in den Worteinbettungen sinnvollerweise in einer ähnlichen Beziehung zu *system* dargestellt werden. Das Wort *subsystem* ist auch in den generischen Worteinbettungen das zweitähnlichste Wort. Während dieses Wort in den generischen Worteinbettungen eine mit ungefähr 0,01 geringfügig größere Ähnlichkeit hat, werden *subsystems* und *filesystem* in den domänenspezifischen Worteinbettungen um 0,03, beziehungsweise 0,02 ähnlicher und somit besser dargestellt. Das Wort *yoggie* aus den domänenspezifischen Worteinbettungen ist der Name eines Unternehmens für Sicherheitssysteme, das vollständig *Yoggie Security Systems* heißt. Auch das Wort *logout* gehört zu den ähnlichsten Wörtern. Damit ist das Abmelden aus einem System gemeint, was in Anforderungen häufig formuliert wird. Bei den ähnlichsten Wörtern in den generischen Worteinbettungen, die einen Bindestrich enthalten, fällt auf, dass diese alle in den domänenspezifischen Worteinbettungen noch ähnlicher dargestellt werden, obwohl sie aufgrund der Vorverarbeitung des Textkorpus nicht im Vokabular sind.

Die Wörter, die dem Wort *user* am ähnlichsten sind, sind in Tabelle 7.4 und Abbildung 7.3 dargestellt. Die meisten der zehn Wörter in den domänenspezifischen und in den generischen Worteinbettungen haben *user* als Teilwort. Am ähnlichsten sind in den domänenspezifischen Worteinbettungen die Wörter *superuser* und *username*, wobei die Ähnlichkeit des ähnlichsten Wortes mit ungefähr 0,61 nicht besonders hoch ist. Diese Wörter werden mit einer um ungefähr 0,1, beziehungsweise 0,07 größeren Ähnlichkeit als in den generi-

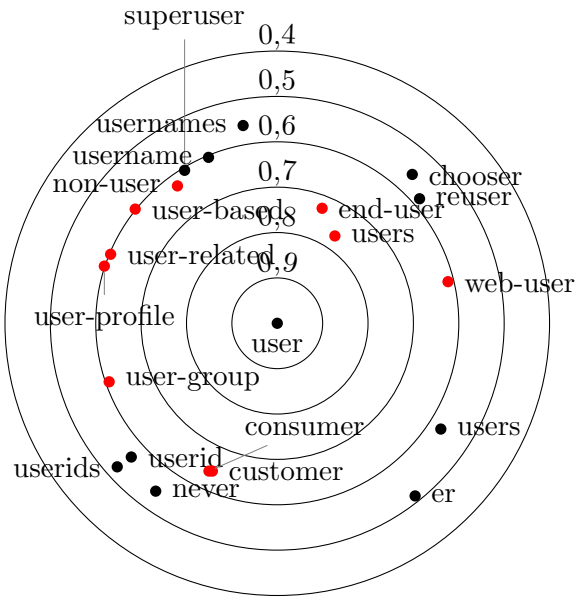

Abbildung 7.3: Darstellung der Ähnlichkeiten und Richtungen der zehn Wörter, die *user* in den domänenspezifischen (schwarz), bzw. generischen (rot) Worteinbettungen am ähnlichsten sind

Wort	Ähnl.	Diff.
superuser	0,6058	0,0993
username	0,6041	0,0741
reuser	0,5832	(0,1305)
users	0,5708	−0,1986
userid	0,5637	−0,0243
usernames	0,5577	0,1279
chooser	0,5571	0,1401
never	0,5432	0,2937
userids	0,5262	−0,0154
er	0,5130	0,3732
users	0,7694	0,1986
end-user	0,7279	(−0,0421)
customer	0,6442	0,3657
consumer	0,6411	0,1326
non-user	0,6257	(−0,1452)
web-user	0,6127	(−0,1310)
user-group	0,6077	(−0,0203)
user-related	0,6025	(0,1460)
user-profile	0,5986	(0,0565)
user-based	0,5983	(−0,0696)

Tabelle 7.4: Die zehn Wörter, die *user* in den domänenspezifischen (oben), bzw. generischen (unten) Worteinbettungen am ähnlichsten sind, mit der Ähnlichkeit sowie dem Ähnlichkeitsunterschied zu den jeweils anderen Worteinbettungen

schen Worteinbettungen dargestellt. Auch *usernames*, der Plural von *username*, befindet sich unter den zehn ähnlichsten Wörtern und wird um einem Differenzwert von ungefähr 0,13 ähnlicher dargestellt als in den generischen. Zudem findet sich unter den ähnlichsten Wörtern sinnvollerweise der Plural *users*, der aber in den generischen Worteinbettungen noch um ungefähr 0,19 ähnlicher dargestellt wird, da wie erwähnt die Beziehung von Singular und Plural nicht besonders domänenspezifisch ist. Des Weiteren befinden sich unter den zehn ähnlichsten Wörtern die Begriffe *userid* und sein Plural *userids*. Diese zwei Begriffe werden in den generischen Worteinbettungen aber mit einer jeweils um ungefähr 0,02 größeren Ähnlichkeit noch geringfügig besser dargestellt. Die zwei Wörter haben fast die gleiche Richtung, was sinnvoll ist, da sie dieselbe Beziehung zum Wort *user* haben. Das Wort *end-user* hingegen wird, obwohl es nicht im Vokabular vorkommt, in den domänenspezifischen Worteinbettungen mit einer um ungefähr 0,04 größeren Ähnlichkeit besser dargestellt.

Nun wird das Wort *service* betrachtet. Die Wörter, die *service* am ähnlichsten sind, sind in Tabelle 7.5 und Abbildung 7.4 dargestellt. Der Plural *services* ist in den domänenspezifischen Worteinbettungen das zweitähnlichste Wort mit einer Ähnlichkeit von ungefähr 0,73. In den generischen Worteinbettungen ist die Ähnlichkeit sogar geringfügig größer, um ungefähr 0,04. Dort ist es das mit Abstand ähnlichste Wort. Die abgesehen vom Plural in den domänenspezifischen Worteinbettungen ähnlichsten Wörter sind *provider* und *pro-*

viders. Das ist sinnvoll und bildet die Tatsache ab, dass Services von etwas oder jemandem angeboten werden. Die Ähnlichkeiten sind bei diesen zwei Wörtern mit Differenzwerten von jeweils ungefähr 0,2 auch deutlich größer als in den generischen Worteinbettungen. Dass der Begriff *serving* als entsprechendes Adjektiv zu den ähnlichsten Wörtern gehört, ist logisch und wird auch in den generischen Worteinbettungen so dargestellt, wenn auch um ungefähr 0,09 weniger ähnlich und somit nicht mehr in den zehn ähnlichsten Wörtern. Als weiteres Adjektiv befindet sich *old* unter den ähnlichsten Wörtern. Es tritt in Anforderungen häufig zusammen mit dem Wort *service* auf, wenn ein alter Service erwähnt wird. Die ähnlichsten Wörter in den generischen Worteinbettungen haben bis auf den Plural *services* keine Überschneidungen mit den ähnlichsten Wörtern in den domänenspezifischen Worteinbettungen. Zusätzlich zu *services* sind die Wörter *delivery*, *customer* und *maintenance* in den generischen Worteinbettungen ähnlicher. Die restlichen der in den generischen Worteinbettungen ähnlichsten Wörter wie *service-only* werden allerdings in den domänenspezifischen Worteinbettungen ähnlicher dargestellt. Unter den in den domänenspezifischen Worteinbettungen ähnlichsten Wörtern befinden sich auch einige Akronyme, deren jeweilige Ähnlichkeit deutlich größer ist als in den generischen. So ist das Akronym *spid* in den domänenspezifischen Worteinbettungen ungefähr 0,49 ähnlicher. Die Vollform dieses Akronyms ist *service profile identifier*. Die Ähnlichkeit zum Wort *service* ist also offensichtlich sinnvoll. Das Akronym *nxxs* ist eine Beschreibung der nordamerikanischen Vorwahl und *lrns* ist der Plural von *location routing number*, einer Identifikationsnummer für Vermittlungsstellen. Sie treten somit beide häufig im Kontext von Anbietern von Kommunikationsdiensten, übersetzt *service provider*, auf. Mit einer Ähnlichkeit von ungefähr 0,58 ist das Wort *soas* in den domänenspezifischen Worteinbettungen ungefähr 0,41 ähnlicher. Es ist der Plural des Akronyms *soa* und steht somit für serviceorientierte Architekturen. Dieses Wort ist also eindeutig der Anforderungsdomäne zuzuordnen und die deutlich größere Ähnlichkeit des Wortes in den domänenspezifischen Worteinbettungen als sehr gut zu bewerten. Es handelt sich hierbei um eine domänenspezifische Feinheit, die die generischen Worteinbettungen nicht erfassen. Die Richtungen der Akronyme unterscheiden sich nicht besonders. Über die Richtungen wird also die Beziehung von Wort zu mit ihm in Verbindung stehendem Akronym dargestellt.

Die Wörter, die dem Wort *information* am ähnlichsten sind, sind in Tabelle 7.6 und Abbildung 7.5 dargestellt. Alle der zehn in den domänenspezifischen ähnlichsten Wörter werden dort als ähnlicher dargestellt als in den generischen. Die meisten der ähnlichsten Wörter, wie *creation*, *deletion*, *aggregation* oder *examination*, beschreiben Verfahren zur Ansammlung und Verarbeitung von Informationen. Zudem befindet sich das dem Nomen *information* entsprechende Verb *informs* unter den ähnlichsten Wörtern. Da in Anforderungen auch oft beschrieben wird, in welchem Format Informationen vorliegen, gehört auch das Wort *format* zu den ähnlichsten Wörtern. Auch die Ähnlichkeit des Wortes *origination* ist sinnvoll. Es tritt in Anforderungen bei der Beschreibung des Ursprungs von Informationen auf. In den generischen Worteinbettungen sind die ähnlichsten Wörter alternative Begriffe zu *information*, wie *info* oder *data*. Diese Wörter werden in den generischen Worteinbettungen im Vergleich zu den domänenspezifischen als ähnlicher dargestellt, was für die generischen Worteinbettungen spricht.

Das nächste Wort, das betrachtet wird, ist das Wort *subscription*. Die Wörter, die *subscription* am ähnlichsten sind, sind in Tabelle 7.7 und Abbildung 7.6 dargestellt. Unter ihnen befinden sich in den domänenspezifischen Worteinbettungen logischerweise das entsprechende Verb *subscribe* mit einer Ähnlichkeit von ungefähr 0,72 und der Plural *subscriptions* als ähnlichstes Wort mit einer Ähnlichkeit von ungefähr 0,88. Diese Wörter werden, wie alle anderen acht auch, mit einer größeren Ähnlichkeit dargestellt als in den generischen Worteinbettungen. Auch das Wort *version* gehört mit einer Ähnlichkeit von ungefähr 0,79 sinnvollerweise zu den ähnlichsten Wörtern. In Anforderungen werden häufig

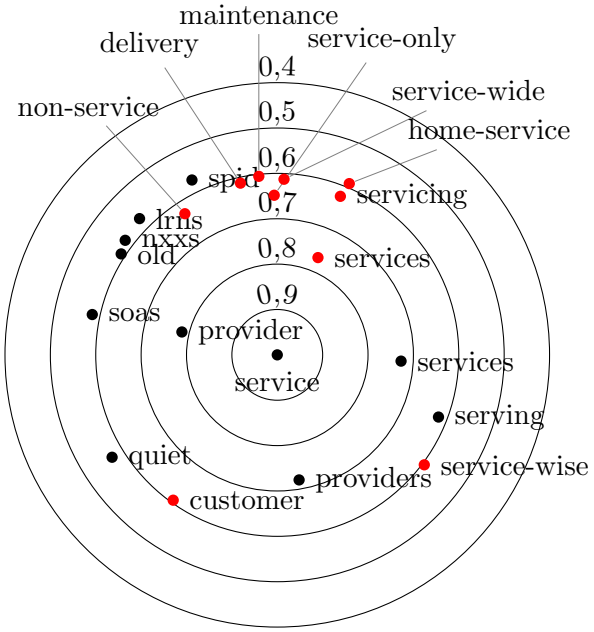

Abbildung 7.4: Darstellung der Ähnlichkeiten und Richtungen der zehn Wörter, die *service* in den domänenspezifischen (schwarz), bzw. generischen (rot) Worteinbettungen am ähnlichsten sind

Wort	Ähnl.	Diff.
provider	0,7838	0,1962
services	0,7269	−0,0408
providers	0,7202	0,1997
serving	0,6192	0,0867
old	0,5903	0,2641
soas	0,5827	0,4092
nxxs	0,5804	(0,3197)
lrns	0,5730	(0,2853)
quiet	0,5717	0,2196
spid	0,5711	0,3852
services	0,7676	0,0408
service-only	0,6481	(−0,2486)
non-service	0,6282	(−0,3224)
servicing	0,6239	(−0,1272)
delivery	0,6129	0,3523
service-wide	0,6126	(−0,2591)
customer	0,6060	0,1880
maintenance	0,6044	0,4033
service-wise	0,5954	(−0,3080)
home-service	0,5905	(−0,3259)

Tabelle 7.5: Die zehn Wörter, die *service* in den domänenspezifischen (oben), bzw. generischen (unten) Worteinbettungen am ähnlichsten sind, mit der Ähnlichkeit sowie dem Ähnlichkeitsunterschied zu den jeweils anderen Worteinbettungen

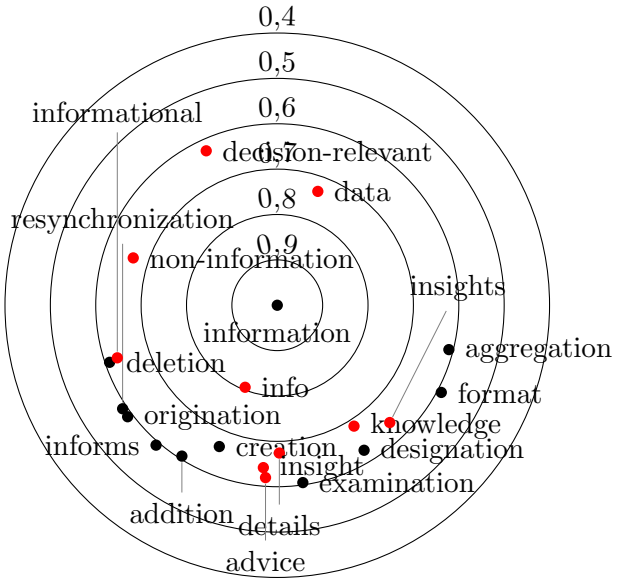

Abbildung 7.5: Darstellung der Ähnlichkeiten und Richtungen der zehn Wörter, die *information* in den domänenspezifischen (schwarz), bzw. generischen (rot) Worteinbettungen am ähnlichsten sind

Wort	Ähnl.	Diff.
creation	0,6634	0,2826
designation	0,6275	0,2386
deletion	0,6097	0,2575
aggregation	0,6094	0,1457
addition	0,6066	0,2183
examination	0,6050	0,1327
informs	0,5920	0,1533
format	0,5906	0,1642
resynchronization	0,5899	0,2743
origination	0,5887	0,2129
info	0,8058	0,3616
data	0,7336	0,3624
knowledge	0,6844	0,5241
details	0,6742	0,3711
non-information	0,6659	(−0,3044)
insights	0,6418	(0,2996)
insight	0,6408	0,4131
informational	0,6289	(−0,1655)
decision-relevant	0,6253	(0,2756)
advice	0,6192	(0,2658)

Tabelle 7.6: Die zehn Wörter, die *information* in den domänenspezifischen (oben), bzw. generischen (unten) Worteinbettungen am ähnlichsten sind, mit der Ähnlichkeit sowie dem Ähnlichkeitsunterschied zu den jeweils anderen Worteinbettungen

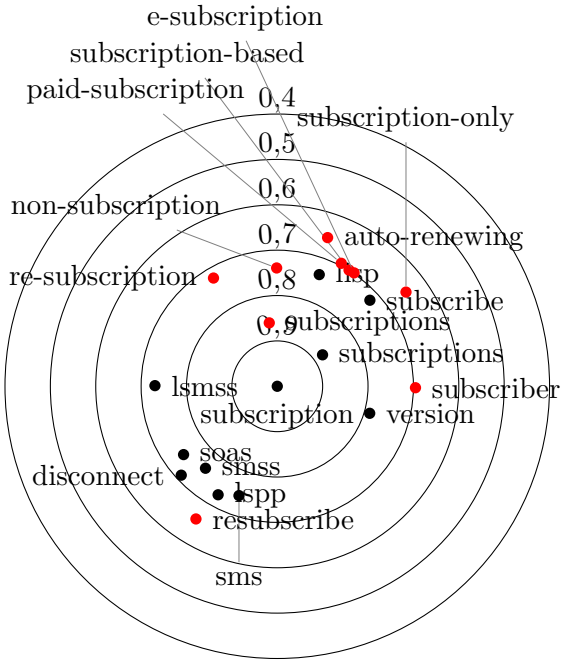

Abbildung 7.6: Darstellung der Ähnlichkeiten und Richtungen der zehn Wörter, die *subscription* in den domänenspezifischen (schwarz), bzw. generischen (rot) Worteinbettungen am ähnlichsten sind

Wort	Ähnl.	Diff.
subscriptions	0,8785	0,0194
version	0,7878	0,3587
smss	0,7598	(0,5275)
sms	0,7447	0,4430
soas	0,7444	0,5877
lisp	0,7370	0,5798
lsmss	0,7304	(0,5662)
lspp	0,7276	(0,6815)
subscribe	0,7214	0,1017
disconnect	0,7116	0,4220
subscriptions	0,8591	−0,0194
non-subscription	0,7392	(−0,2455)
re-subscription	0,7231	(−0,2604)
subscription-based	0,6990	(−0,2629)
e-subscription	0,6981	(−0,2880)
subscriber	0,6954	0,1009
paid-subscription	0,6944	(−0,2801)
resubscribe	0,6570	(−0,0726)
auto-renewing	0,6540	(0,2871)
subscription-only	0,6485	(−0,3167)

Tabelle 7.7: Die zehn Wörter, die *subscription* in den domänenspezifischen (oben), bzw. generischen (unten) Worteinbettungen am ähnlichsten sind, mit der Ähnlichkeit sowie dem Ähnlichkeitsunterschied zu den jeweils anderen Worteinbettungen

die verschiedenen Versionen von Abonnements oder die verschiedenen Software-Versionen, die abonniert werden können, beschrieben. Die Akronyme *lisp* und *lspp* beschreiben zwei verschiedene Arten von Ports, die als Eigenschaft eine *subscription version*, also eine bestimmte Version eines Abonnements haben. Die zehn in den generischen Worteinbettungen ähnlichsten Wörter werden fast alle in den domänenspezifischen Worteinbettungen als ähnlicher dargestellt, was klar für die domänenspezifischen Worteinbettungen spricht.

Die Wörter, die *version* am ähnlichsten sind, sind in Tabelle 7.8 und Abbildung 7.7 dargestellt. Das in den domänenspezifischen Worteinbettungen ähnlichste Wort, *subscription*, und seine Beziehung zum Wort *version* wurde oben schon betrachtet. Das zweitähnlichste Wort ist der Plural *versions* mit einer Ähnlichkeit von ungefähr 0,75. Er wird in den generischen Worteinbettungen, wo es das ähnlichste Wort ist, mit einem Unterschied von ungefähr 0,07 ein wenig besser dargestellt, da wie bereits erwähnt die Beziehung von Singular und Plural nicht besonders domänenspezifisch ist. Dass das Wort *old* als beschreibendes Adjektiv unter den ähnlichsten Wörtern ist, ist logisch, genau wie die Ähnlichkeit der passenden Präposition *upon*. Viele der in den generischen Worteinbettungen ähnlichsten Wörter wie *sub-version*, *web-version* und *pre-version* werden in den domänenspezifischen Worteinbettungen noch ähnlicher dargestellt.

Als Nächstes wird das Wort *provider* betrachtet. Die Wörter, die dem Wort *provider* am ähnlichsten sind, sind in Tabelle 7.9 und Abbildung 7.8 dargestellt. Das in den domänenspe-

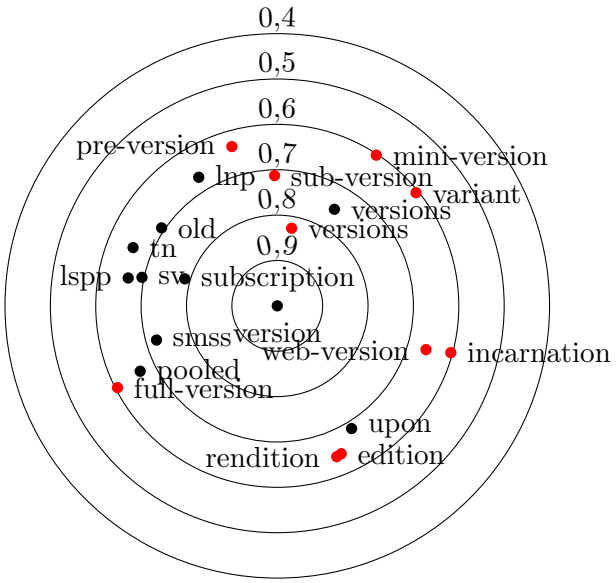

Abbildung 7.7: Darstellung der Ähnlichkeiten und Richtungen der zehn Wörter, die *version* in den domänenspezifischen (schwarz), bzw. generischen (rot) Worteinbettungen am ähnlichsten sind

Wort	Ähnl.	Diff.
subscription	0,7878	0,3587
versions	0,7527	−0,0734
smss	0,7234	(0,4002)
sv	0,6953	0,4858
old	0,6925	0,3276
upon	0,6838	0,5350
lnp	0,6679	0,6639
lspp	0,6658	(0,5568)
pooled	0,6653	0,4359
tn	0,6573	0,5198
versions	0,8261	0,0734
sub-version	0,7128	(−0,1694)
web-version	0,6583	(−0,2474)
edition	0,6449	(0,3120)
rendition	0,6427	(0,1944)
pre-version	0,6349	(−0,2399)
variant	0,6054	(0,2236)
full-version	0,6045	(−0,2874)
incarnation	0,6039	(0,3465)
mini-version	0,6026	(−0,2735)

Tabelle 7.8: Die zehn Wörter, die *version* in den domänenspezifischen (oben), bzw. generischen (unten) Worteinbettungen am ähnlichsten sind, mit der Ähnlichkeit sowie dem Ähnlichkeitsunterschied zu den jeweils anderen Worteinbettungen

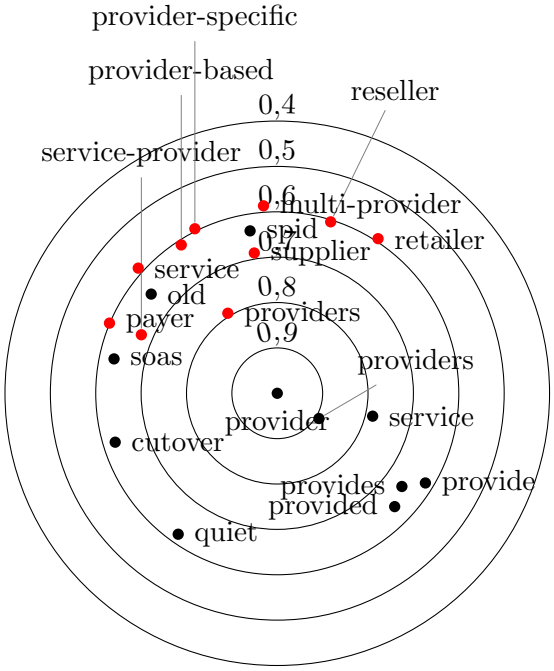

Abbildung 7.8: Darstellung der Ähnlichkeiten und Richtungen der zehn Wörter, die *provider* in den domänenspezifischen (schwarz), bzw. generischen (rot) Worteinbettungen am ähnlichsten sind

Wort	Ähnl.	Diff.
providers	0,8926	0,0999
service	0,7838	0,1962
provides	0,6570	0,3763
old	0,6465	0,4266
provided	0,6404	0,2643
spid	0,6370	0,4033
soas	0,6324	0,5070
cutover	0,6269	0,3923
quiet	0,6203	0,4394
provide	0,6184	0,2899
providers	0,7926	−0,0999
supplier	0,6866	(0,2281)
service-provider	0,6739	(−0,2505)
provider-based	0,6105	(−0,3096)
reseller	0,6042	(0,3014)
payer	0,5992	(0,2106)
provider-specific	0,5944	(−0,1650)
retailer	0,5932	(0,1511)
service	0,5877	−0,1962
multi-provider	0,5857	(−0,2813)

Tabelle 7.9: Die zehn Wörter, die *provider* in den domänenspezifischen (oben), bzw. generischen (unten) Worteinbettungen am ähnlichsten sind, mit der Ähnlichkeit sowie dem Ähnlichkeitsunterschied zu den jeweils anderen Worteinbettungen

zifischen ähnlichste Wort ist der Plural *providers* mit einer hohen Ähnlichkeit von ungefähr 0,89, die sogar um ungefähr 0,1 größer ist als in den generischen Worteinbettungen, wo das Wort ebenfalls das ähnlichste ist. Das in den domänenspezifischen Worteinbettungen zweitähnlichste Wort ist *service*, was sinnvoll ist, da der Anbieter einen Service bereitstellt. Mit *provide*, *provides* und *provided* sind auch sinnvollerweise drei Formen des dem Wort *provider* entsprechenden Verbes unter den zehn ähnlichsten Wörtern. Die Ähnlichkeiten sind für die drei Begriffe im Vergleich zu den generischen Worteinbettungen sogar deutlich größer, um ungefähr 0,29, 0,38, beziehungsweise 0,26. Die Ähnlichkeit der Verben zu *provide* wird in den domänenspezifischen Worteinbettungen also deutlich besser dargestellt. Einige der in den generischen Worteinbettungen ähnlichsten Wörter wie *service-provider* und *provider-based* werden in den domänenspezifischen Worteinbettungen noch ähnlicher dargestellt. Für die Wörter *supplier*, *reseller*, *payer* und *retailer*, übersetzt Lieferant, Wiederverkäufer, Zahler und Einzelhändler, ist das hingegen nicht der Fall. Das liegt daran, dass diese Begriffe nicht aus der Anforderungsdomäne, sondern aus der Wirtschaftsdomäne stammen. In der Anforderungsdomäne kommen diese Wörter nicht so häufig im Kontext von *provider* vor. Hier wird wieder deutlich, wie sich domänenspezifische Feinheiten in Worteinbettungen manifestieren.

Die Wörter, die dem Wort *interface* am ähnlichsten sind, sind in Tabelle 7.10 und Abbildung 7.9 dargestellt. Am ähnlichsten sind in den domänenspezifischen Worteinbettungen

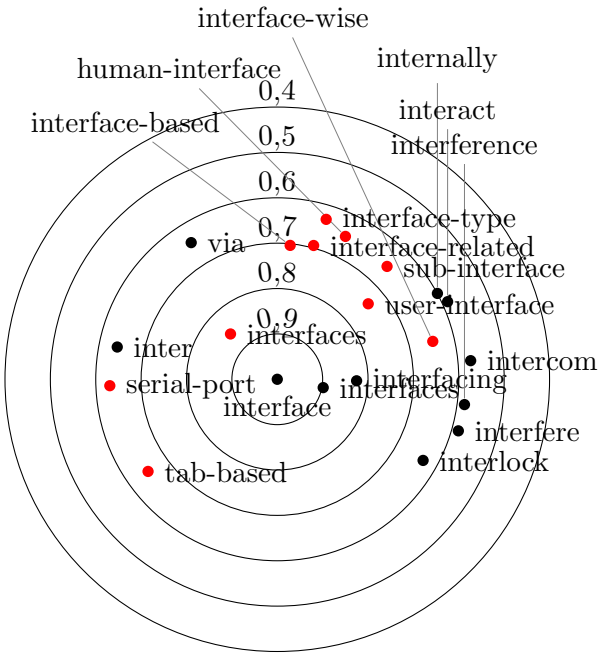

Abbildung 7.9: Darstellung der Ähnlichkeiten und Richtungen der zehn Wörter, die *interface* in den domänenspezifischen (schwarz), bzw. generischen (rot) Worteinbettungen am ähnlichsten sind

Wort	Ähnl.	Diff.
interfaces	0,8971	0,0407
interfacing	0,8252	0,2270
via	0,6440	0,2999
inter	0,6402	0,3198
interlock	0,6322	0,3149
internally	0,5996	0,2448
interact	0,5875	0,1600
interfere	0,5851	0,3308
interference	0,5837	0,2282
intercom	0,5719	0,2036
interfaces	0,8565	−0,0407
user-interface	0,7395	(−0,1857)
interface-based	0,7035	(−0,2388)
interface-related	0,6944	(−0,1499)
sub-interface	0,6530	(−0,2549)
human-interface	0,6511	(−0,2246)
tab-based	0,6502	(0,3094)
interface-wise	0,6473	(−0,2966)
interface-type	0,6314	(−0,3181)
serial-port	0,6307	(0,2197)

Tabelle 7.10: Die zehn Wörter, die *interface* in den domänenspezifischen (oben), bzw. generischen (unten) Worteinbettungen am ähnlichsten sind, mit der Ähnlichkeit sowie dem Ähnlichkeitsunterschied zu den jeweils anderen Worteinbettungen

die mit *interface* direkt verwandten Begriffe *interfaces* und *interfacing* mit hohen Ähnlichkeiten von ungefähr 0,9, beziehungsweise 0,83. Mit *interfacing* ist dabei die Kopplung von Systemen gemeint. Diese zwei Begriffe werden in den domänenspezifischen Worteinbettungen besser dargestellt als in den generischen mit Differenzwerten von ungefähr 0,04, beziehungsweise 0,23. Das drittähnlichste Wort ist die passende Präposition *via*. Die restlichen Wörter haben eine eher geringe Ähnlichkeit mit höchstens 0,63. Sie zeichnen sich durch die Gemeinsamkeit des Teilwortes *inter* aus, und sind meistens sinnvoll wie der Begriff *interact*, der sich auf das interagieren mit einem Software-System über seine Schnittstelle bezieht. Der von den zehn Wörtern am wenigsten ähnliche Begriff *interlock* erscheint hingegen weniger im Zusammenhang mit *interface* zu stehen. Die meisten der zehn in den generischen Worteinbettungen ähnlichsten Wörter sind in den domänenspezifischen Worteinbettungen ähnlicher dargestellt. Neben dem bereits betrachteten Plural ist zum Beispiel das Wort *user-interface* in den domänenspezifischen Worteinbettungen um ungefähr 0,2 ähnlicher, obwohl es wegen des Sonderzeichens nicht im Vokabular vorkommt.

Das letzte Wort, das betrachtet wird, ist das Wort *window*. Die Wörter, deren Wortvektoren dem Wortvektor zum Wort *window* am ähnlichsten sind, sind in Tabelle 7.11 und Abbildung 7.10 dargestellt. Die in den domänenspezifischen Worteinbettungen ähnlichsten Wörter sind die direkt verwandten Begriffe *windowing* und *windows*. Sie werden genau wie alle anderen der zehn Wörter in den domänenspezifischen Worteinbettungen im Ver-

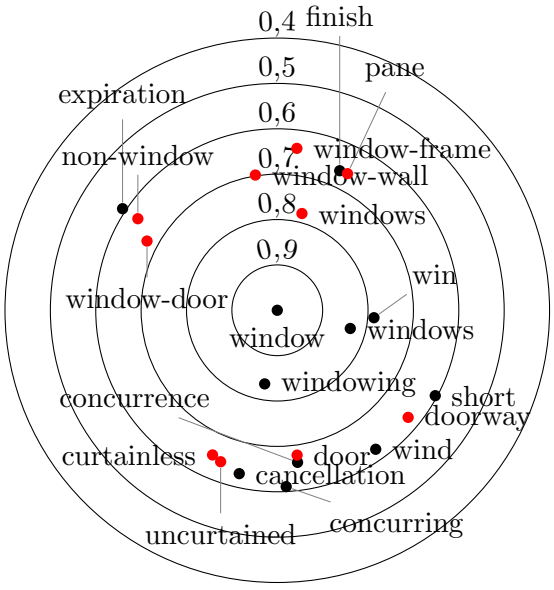

Abbildung 7.10: Darstellung der Ähnlichkeiten und Richtungen der zehn Wörter, die *window* in den domänenspezifischen (schwarz), bzw. generischen (rot) Worteinbettungen am ähnlichsten sind

Wort	Ähnl.	Diff.
windowing	0,8354	0,3272
windows	0,8342	0,0543
win	0,7863	0,5025
finish	0,6630	0,3616
concurrence	0,6618	0,4419
cancellation	0,6303	0,3576
wind	0,6246	0,2210
concurring	0,6107	0,4590
short	0,6042	0,2335
expiration	0,5921	0,2638
windows	0,7799	−0,0543
window-wall	0,6980	(−0,2408)
door	0,6778	(0,3837)
window-door	0,6749	(−0,2771)
pane	0,6614	(0,3483)
curtainless	0,6507	(0,2468)
uncurtained	0,6436	(0,3051)
window-frame	0,6405	(−0,2543)
non-window	0,6324	(−0,3352)
doorway	0,6273	(0,3919)

Tabelle 7.11: Die zehn Wörter, die *window* in den domänenspezifischen (oben), bzw. generischen (unten) Worteinbettungen am ähnlichsten sind, mit der Ähnlichkeit sowie dem Ähnlichkeitsunterschied zu den jeweils anderen Worteinbettungen

gleich zu den generischen Worteinbettungen als ähnlicher dargestellt. Interessant ist die Ähnlichkeit von *window* zu den Wörtern *finish*, *cancellation*, *short* und *expiration* in den domänenspezifischen Worteinbettungen, beziehungsweise zu den Wörtern *door*, *pane*, *curtainless* und anderen in den generischen Worteinbettungen. Diese Beziehungen stimmen damit überein, dass das Wort *window* in Anforderungen meistens nicht die Bedeutung der Lichtöffnung hat, sondern die Bedeutung als Zeitfenster, in dem zum Beispiel etwas beendet werden muss oder in dem etwas abläuft. Die Konsequenz der Erfassung dieser domänenspezifischen Feinheit für die Anwendung kann zum Beispiel eine bessere Leistung der domänenspezifischen Worteinbettungen in Aufgaben zur Klassifizierung von Anforderungen in ihre Art sein. Die Erwähnung eines Zeitfensters in einer Anforderung deutet nämlich darauf hin, dass es sich um eine nichtfunktionale Anforderung handelt. Schließlich wird zum Beispiel bei der Beschreibung der Performanz eines Systems oft ein Zeitfenster erwähnt. Die generischen Worteinbettungen erfassen im Vergleich zu den domänenspezifischen die Bedeutung von *window* als Lichtöffnung anstatt als Zeitfenster, und können den Indikator, den *window* bei der Klassifizierung darstellt, nicht so gut nutzen.

Zusammenfassend lässt sich feststellen, dass die generischen Worteinbettungen zwar aufgrund ihres umfangreicheren Korpus manche Wörter wie *data* etwas besser erfassen, die domänenspezifischen Worteinbettungen aber gut mithalten können und sich aus ihnen sinnvolle Wortähnlichkeiten ableiten lassen. Der große Vorteil, den die domänenspezifi-

schen Worteinbettungen bieten, ist die Erfassung domänenspezifischer Feinheiten wie der größeren Ähnlichkeit von *service* und dem Akronym von serviceorientierter Architektur, der geringeren Ähnlichkeit von *provider* und Begriffen aus der Wirtschaftsdomäne, sowie der Erfassung der Bedeutung von *window* als Zeitfenster anstatt als Lichtöffnung.

7.1.2 Clusteranalysen

Um die Struktur der Wortvektoren der domänenspezifischen Worteinbettungen im Gesamten zu analysieren, werden Clusteranalysen durchgeführt.

Dabei wird zuerst der k-Means-Algorithmus eingesetzt. Mit ihm soll überprüft werden, wie gleichmäßig sich die Wortvektoren auf eine zuvor festgelegte Anzahl k von Clustern verteilen.

Der Algorithmus initialisiert zuerst zufällig k Mittelwerte. Jeder Wortvektor wird seinem ähnlichsten Mittelwert zugeordnet. Sie bilden damit einen temporären Cluster. Dann wird der Mittelwert für jeden der Cluster neu berechnet. Das ganze wird durchgeführt, bis sich die Zugehörigkeit der Wortvektoren zu den Clustern nicht mehr ändert. Der Gesamtvorgang wird mehrmals wiederholt, wobei die Anzahl der Wiederholungen zuvor festgelegt wurde. So sollen verschiedene zufällige Initialisierungen genutzt werden. Die Mittelwerte, die am häufigsten auftreten, werden am Ende ausgewählt und ihre jeweiligen Wortvektoren bilden die k Cluster. Als Anzahl der Cluster k wird die Zahl elf gewählt und als Anzahl der Wiederholungen die Zahl 35. Die Zahlen stellen einen Kompromiss zwischen ihrer Größe und dem benötigten Rechenaufwand dar. Es wird die Kosinus-Ähnlichkeit als Ähnlichkeitsmaß verwendet.

Die Anzahl der Wortvektoren pro Cluster wird in Tabelle 7.12 für alle Wörter und für die 50 häufigsten im Trainingskorpus vorkommenden Wörter dargestellt. Dabei fällt auf, dass bei Betrachtung aller Wörter die Wortvektoren relativ gleichmäßig auf die Cluster verteilt sind. Werden stattdessen nur die 50 häufigsten Wörter betrachtet, so ist die Verteilung deutlich ungleicher. Mit 24 sind fast die Hälfte der Wortvektoren der 50 häufigsten Wörter im selben Cluster. Die Worteinbettungen stellen also viele der häufig vorkommenden Wörter als ähnlich dar. Das liegt daran, dass sich in Anforderungen bestimmte Formulierungen häufig wiederholen. Die Wörter aus diesen Formulierungen gehören damit zu den am häufigsten vorkommenden Wörtern und treten gleichzeitig in den Anforderungen in unmittelbarer Nachbarschaft zueinander auf. Somit werden diese häufigen Wörter auch von den Worteinbettungen als ähnlich dargestellt. Es lässt sich durch diese Clusteranalyse also feststellen, dass die sich in Anforderungen häufig wiederholenden Formulierungen von den Worteinbettungen gut erfasst werden. Für ein Wort aus einer Formulierung werden andere Wörter aus dieser Formulierung häufig als ähnlich dargestellt.

Mithilfe einer agglomerativen Clusteranalyse soll nun überprüft werden, ob die Wörter in einem Cluster auch tatsächlich in ihrem Sinn zusammenhängend sind und zu einer Klasse gehören. Ein Wort ist in einem Cluster enthalten, wenn sein entsprechender Wortvektor im Cluster enthalten ist.

Der Algorithmus fasst zuerst jeden Wortvektor als einen Cluster auf. Dann werden Schritt für Schritt jeweils zwei Cluster nach einem Verknüpfungskriterium zusammengeführt. Das Verknüpfungskriterium ist ein Wert für die Distanz zwischen je zwei Clustern, den der Algorithmus minimieren soll. Das ganze wird durchgeführt, bis der minimale Distanzwert einen zuvor festgelegten Schwellwert erreicht. Als Verknüpfungskriterium wird der Durchschnitt aller Ähnlichkeiten zwischen je zwei Wörtern aus den zwei Clustern verwendet. Dabei wird als Ähnlichkeitsmaß wieder die Kosinus-Ähnlichkeit verwendet. Der Schwellwert, über dem Cluster nicht mehr zusammengeführt werden, wird auf 0,7 gesetzt. Dies ist in aufgrund des Rechenaufwandes gewählten Zehntelschritten der erste Schwellwert,

Tabelle 7.12: Die Anzahl der Wortvektoren für jeden der durch den k-Means-Algorithmus erhaltenen Cluster bei Betrachtung aller Wörter, beziehungsweise der 50 häufigsten

Cluster	Gesamtzahl	Anzahl aus 50
1	309	1
2	383	1
3	332	1
4	401	24
5	311	3
6	602	4
7	436	1
8	362	7
9	413	0
10	360	4
11	321	4

bei dem am Ende mehr als ein Cluster übrigbleibt. Das heißt beim Schwellwert 0,8 bleibt nur ein Cluster übrig. Beim Schwellwert 0,7 sind es hingegen 83 Cluster. Demnach könnte man die Worteinbettungen in 83 verschiedene Klassen einteilen. Für jeden Cluster wurde als Repräsentant das Wort aus dem jeweiligen Cluster gewählt, das am häufigsten im Trainingskorpus auftritt. Die Anzahl der Wortvektoren pro Cluster sowie der jeweilige Repräsentant sind in Tabelle D.3 im Anhang aufgeführt. Die Anzahl der Wortvektoren pro Cluster unterscheidet sich je nach Cluster deutlich. So sind im kleinsten Cluster drei und im größten 556 Wortvektoren enthalten. Der Grund für die sehr unterschiedlich großen Cluster ist eine offensichtlich ungleiche Verteilung der Wortvektoren.

Drei Cluster werden beispielhaft in Tabelle 7.13 vollständig aufgeführt. Dafür wird einmal ein Cluster mit minimaler Anzahl von Wortvektoren sowie jeweils ein Cluster mit zehn beziehungsweise zwanzig Wortvektoren ausgewählt. Die Nummerierung der Cluster wird aus Tabelle D.3 übernommen. Die Wörter sind von oben nach unten entsprechend ihrer Häufigkeit im Trainingskorpus sortiert. In zweispaltigen Fall wird mit der linken Spalte begonnen. Um eine Aussage über die Qualität der Worteinbettungen treffen zu können, wird nun überprüft, wie zusammenhängend die in den drei beispielhaften Clustern enthaltenen Wörter in ihrem Sinn sind.

In Cluster 58 sind die drei Wörter *second*, *seconds* und *sec* enthalten. Diese drei Wörter sind offensichtlich sehr stark zusammenhängend und lassen sich klar als eine Klasse auffassen. Mit der Bedeutung *Sekunde* sind für *second* der Plural und die Abkürzung im Cluster enthalten. Interessant ist, dass *second* auch *zweite* bedeuten kann. Diese Bedeutung ist in diesem Cluster offenbar nicht beinhaltet, da das Wort *first* in einem anderen Cluster enthalten ist. Die Wörter *second* und *first* sind sich also nicht so ähnlich wie die Wörter im Cluster. Ein Grund dafür ist die Berücksichtigung von Teilwörtern durch `fastText`. Während *second*, *seconds* und *sec* in vielen Teilwörtern übereinstimmen, ist das für *second* und *first* nicht der Fall.

In Cluster 43 sind zehn Begriffe enthalten, die sich allerdings nicht in einen, sondern in zwei Sinnzusammenhangskomponenten aufteilen lassen. Zum einen sind das Wort *format* und verschiedene Formen des Wortes wie zum Beispiel *form* und *formats* im Cluster enthalten. Zum anderen sind die Wörter *inform*, *informs* und *informed* im Cluster enthalten. Diese zwei Wortgruppen wären in zwei verschiedenen Clustern sinnvoller aufgeteilt. Der Grund für ihre Ähnlichkeit ist das gemeinsame Teilwort *form*. Um zu überprüfen, ob die zwei

Tabelle 7.13: Alle Wörter zu den Wortvektoren in drei beispielhaften durch die agglomerative Clusteranalyse erhaltenen Clustern

Cluster 58	Cluster 43	Cluster 25	
second	format	include	fully
seconds	form	including	uploaded
sec	formats	download	ftp
	forms	full	downloading
	inform	includes	downloaded
	informed	upload	loaded
	formatted	bulk	loading
	informs	load	broad
	formed	included	uploading
	formatting	downloads	loads

Wortgruppen in den Worteinbettungen tatsächlich nicht gut getrennt sind, oder ob das nur durch die Clusteranalyse so dargestellt wird, werden die Ähnlichkeiten berechnet. Das Wort *inform* hat zu den Wörtern seiner Wortgruppe, *informs* und *informed*, Ähnlichkeiten von ungefähr 0,84, beziehungsweise 0,73. Zum Wort *format* beträgt die Ähnlichkeit hingegen nur ungefähr 0,23. Somit sind die Wortgruppen in den Worteinbettungen gut getrennt.

In Cluster 25 sind zwanzig Begriffe zum Thema Laden und Einbinden enthalten. Es sind verschiedene Formen der Wörter *load*, wie zum Beispiel *download* und *upload*, sowie verschiedene Formen des Wortes *include* im Cluster enthalten. Zudem sind andere Wörter aus diesem Sinnzusammenhang im Cluster enthalten, wie *ftp*, das für *File Transfer Protocol* steht, oder *full* und *fully*, die im Kontext vom vollständigen Laden oder Einbinden von etwas auftreten.

Zu beachten ist, dass das Verfahren der Einteilung in Cluster nur verwendet wurde, um eine Analyse möglich zu machen. Wie anhand der Wortgruppen in Cluster 43 gezeigt, können schließlich trotz der Einteilung in denselben Cluster bestimmte Wörter ähnlicher zueinander sein als zu anderen.

7.2 Domänenadaptierte Worteinbettungen

Um unter verschiedenen Kombinationen der domänenspezifischen und generischen Worteinbettungen die beste zu finden, werden sie in Abschnitt 7.2.1 extrinsisch evaluiert. Die Kombination mit den besten Ergebnissen wird dann ausgewählt. Als Kombinationsmethoden werden dabei die nichtlineare KCCA mit anschließender linearen Kombination nach Kameswara Sarma et al. [KSLS18], beziehungsweise die Durchschnittsbildung nach Coates und Bollegala [CB18] verwendet. Erstere wird im Folgenden mit `PROJ` und letztere mit `AVG` bezeichnet. Die Ergebnisse der zwei Kombinationsmethoden mit jeweils gleichen Gewichtungen der domänenspezifischen und der generischen Worteinbettungen werden in einer Aufgabe zur Klassifizierung von Sätzen in Anforderungsbeschreibungen eingesetzt und die Qualität der Ergebnisse verglichen. Die Kombinationsmethode mit den besseren Ergebnissen wird ausgewählt. Dann werden mehrere mit dieser Kombinationsmethode erhaltene Worteinbettungen, die sich durch verschiedenen Gewichtungen der domänenspezifischen und der generischen Worteinbettungen unterscheiden, anhand derselben Klassifizierungsaufgabe evaluiert. Auf Grundlage der daraus gewonnen Erkenntnisse wird eine Gewichtung ausgewählt. Dies sind dann die finalen domänenadaptierten Worteinbettungen.

Zum Schluss werden in Abschnitt 7.2.2 die generischen, die domänenspezifischen und die domänenadaptierten Worteinbettungen in Bezug auf ihre Leistung verglichen. Dazu wird die Qualität der Ergebnisse in der Klassifizierungsaufgabe verglichen, die durch Einsetzen der generischen, der domänenspezifischen, beziehungsweise der domänenadaptierten Worteinbettungen in die Aufgabe entstehen.

Als Klassifizierungsaufgabe wird die Klassifizierung von Sätzen aus Anforderungsbeschreibungen in eine der vier Klassen *Aktion*, *Aggregation*, *Ereignis* und *Zustand* verwendet. Diese Aufgabe wurde aus dem Projekt `INDIRECT` [Hey19] übernommen. Sie dient der Bestimmung der semantischen Funktion der Sätze in Anforderungsbeschreibungen [Tom19]. Der Datensatz, der in der Klassifizierungsaufgabe verwendet wird, enthält insgesamt 700 annotierte Sätze aus Anforderungsbeschreibungen. Der Datensatz wurde aus den schon im Aufbau des Textkorpus verwendeten Datensätzen `WARC` und `iTrust` von CoEST[1], sowie `NFR` [CMLP07], `Software Requirement Risk Prediction` [SNZ18] und `Ice Breaker` [CSB+05] aufgebaut. Die Anzahl der einzelnen Klassen in den Datensätzen ist in Tabelle 7.14 aufgeführt. Es werden die vier Klassifikatoren Zufallswald, Stützvektormaschine, logistische Regression und LSTM eingesetzt. Die ersten drei nutzen dabei für die einzelnen Sätze in der Trainings-, beziehungsweise Testmenge den jeweiligen Durchschnitt der entsprechenden Wortvektoren als Eingabe. Das LSTM nutzt die Wortvektoren der untersuchten Worteinbettungen zur Initialisierung der Gewichte. Dann wird es auf den Sätzen in der Trainingsmenge mit Verwendung der entsprechenden Worteinbettungen trainiert. Mit dem trainierten LSTM werden dann die Sätze aus Anforderungsbeschreibungen in der Testmenge wieder mit Verwendung der entsprechenden Worteinbettungen klassifiziert. Somit ist das LSTM der einzige der vier Klassifikatoren, der die Reihenfolge der Wörter berücksichtigt. Alle vier Klassifikatoren werden in drei verschiedenen Testverfahren eingesetzt. Das erste Testverfahren wird im Folgenden Training-Test-Teilung genannt. Dabei werden 80% des Datensatzes zum Training des Klassifikators und die restlichen 20% zum Testen genutzt. Die Aufteilung erfolgt, damit die Testmenge nicht zum Training verwendet wird. Schließlich ist das Ziel die Prüfung, wie der Klassifikator ihm unbekannte Sätze aus Anforderungsbeschreibungen klassifiziert. Das zweite Testverfahren ist eine zehnfache Kreuzvalidierung. Dabei wird der Datensatz in zehn gleich große Teilmengen aufgeteilt. In zehn Durchläufen wird dann jeweils eine andere Teilmenge getestet, während die restlichen neun Teilmengen davor beim Training des Klassifikators zum Einsatz kamen. Das Endergebnis bildet sich aus dem Durchschnitt der Ergebnisse der einzelnen Durchläufe. So wird im Gegensatz zur Training-Test-Teilung der gesamte Datensatz in mehreren Schritten zum Testen verwendet. Dadurch sind die Ergebnisse aussagekräftiger als in der Training-Test-Teilung. Das dritte Testverfahren wird im Folgenden projektspezifische Kreuzvalidierung genannt. Ähnlich wie bei der zehnfachen Kreuzvalidierung wird in mehreren Durchläufen jeweils ein anderes Projekt getestet, während die restlichen Projekte davor beim Training des Klassifikators zum Einsatz kamen. Das Endergebnis bildet sich wieder aus dem Durchschnitt der Ergebnisse der einzelnen Durchläufe. So wird wie bei der zehnfachen Kreuzvalidierung der gesamte Datensatz in mehreren Schritten zum Testen verwendet, allerdings erfolgt die Aufteilung entlang der einzelnen Projekte.

Als Maße werden im Folgenden die Genauigkeit und der gewichtete Durchschnitt des F_1-Maßes betrachtet. Bei beiden Maßen ist null das schlechteste und eins das beste Ergebnis. Die Genauigkeit ist die Anzahl der richtig klassifizierten Sätze geteilt durch die Anzahl aller Sätze. Sie ist bei einer möglichst gleichmäßigen Verteilung der Klassen besonders aussagekräftig. Da dies, wie in Tabelle 7.14 aufgelistet ist, nicht in hohem Maße der Fall ist, wird zusätzlich das F_1-Maß verwendet. Das F_1-Maß ist das harmonische Mittel aus Präzision und Ausbeute. Die Präzision für eine Klasse ist der Anteil der Sätze, die dieser Klasse richtigerweise vom Klassifikator zugeordnet werden, an allen Sätzen, die dieser

[1] Quelle: `www.coest.org`, zuletzt besucht am 23.04.2020

Tabelle 7.14: Anzahl der einzelnen Klassen in den Datensätzen

Datensatz	Aktion	Aggregation	Ereignis	Zustand	Gesamt
iTrust	88	22	83	37	230
NFR	139	15	43	14	202
Ice Breaker	32	6	55	7	100
Risk Prediction	56	17	16	6	98
WARC	24	8	23	15	70

Klasse vom Klassifikator zugeordnet werden. Die Ausbeute für eine Klasse ist der Anteil der Sätze, die dieser Klasse richtigerweise vom Klassifikator zugeordnet werden, an allen Sätzen, die tatsächlich zu dieser Klasse gehören. Beim gewichteten Durchschnitt des F_1-Maßes wird zuerst für jede Klasse das F_1-Maß berechnet. Für die F_1-Maße wird dann der Durchschnitt berechnet, wobei der Wert für eine Klasse mit dem Anteil dieser Klasse am Datensatz gewichtet wird. Durch die Gewichtung soll verhindert werden, dass besonders schlechte oder gute Ergebnisse in einer seltenen Klasse übermäßig stark gewichtet werden und das Gesamtergebnis verfälschen.

7.2.1 Auswahl der Kombination

Nun werden zuerst die Ergebnisse der Kombinationsmethoden `AVG` und `PROJ` in der Klassifizierungsaufgabe verglichen, wobei bei der Kombination die domänenspezifischen und die generischen Worteinbettungen gleich gewichtet wurden. Die Ergebnisse in der Klassifizierungsaufgabe sind in Tabelle 7.15 dargestellt. Die Kombinationsmethode `AVG` liefert für alle Testverfahren, Klassifikatoren und Maße klar bessere Ergebnisse. Das beste Ergebnis ist eine Genauigkeit von 0,81 und ein gewichteter Durchschnitt des F_1-Maßes von 0,80 für `AVG` in der Training-Test-Teilung mit der logistischen Regression. Die Kombinationsmethode `PROJ` liefert hier im Vergleich nur eine Genauigkeit von 0,65 und einen gewichteten Durchschnitt des F_1-Maßes von 0,62. Die Werte für `AVG` in der Training-Test-Teilung mit dem LSTM sind mit nur minimalem Abstand die zweitbesten Ergebnisse. Hier ist für `AVG`, beziehungsweise `PROJ` die Genauigkeit 0,81, beziehungsweise 0,63 und der gewichtete Durchschnitt des F_1-Maßes 0,79, beziehungsweise 0,57. Auf Basis dieser Ergebnisse wird die Kombinationsmethode `AVG` für das weitere Vorgehen ausgewählt.

Da nun die Kombinationsmethode festgelegt wurde, werden für `AVG` verschiedene Gewichtungen der generischen und domänenspezifischen Worteinbettungen in der Klassifizierungsaufgabe ausprobiert und die Ergebnisse verglichen. So soll überprüft werden, ob eine Alternative zur gleichen Gewichtung, wie sie im Vergleich mit `PROJ` verwendet wurde, noch bessere Ergebnisse liefert. Neben der gleichen Gewichtung von je 0,5 werden Gewichtungen der generischen Worteinbettungen von 0,3 und 0,7 ausprobiert. Die Ergebnisse in der Klassifizierungsaufgabe sind in Tabelle 7.16 aufgelistet. Dabei sind bei `AVG`-0,3 die generischen Worteinbettungen zu 0,3 gewichtet und die domänenspezifischen zu 0,7. Bei `AVG`-0,7 ist die Gewichtung umgekehrt und bei `AVG`-0,5 ist sie gleich.

Vergleicht man die Ergebnisse von `AVG`-0,3 und `AVG`-0,5, fällt auf, dass die Ergebnisse von `AVG`-0,3 nie besser sind. Bis auf einen Wert liefert `AVG`-0,3 sogar immer schlechtere Ergebnisse, mit einem Differenzwert von bis zu 0,06. Dieser eine Wert ist der gewichtete Durchschnitt des F_1-Maßes in der Training-Test-Teilung mit der logistischen Regression. Er beträgt sowohl für `AVG`-0,3 als auch für `AVG`-0,5 0,80. Das beste Ergebnis für `AVG`-0,3 ist eben jene Training-Test-Teilung mit der logistischen Regression. Dasselbe gilt, wie schon im Vergleich mit `PROJ` festgestellt, für `AVG`-0,5.

Beim Vergleich von `AVG`-0,7 und `AVG`-0,5 lässt sich feststellen, dass `AVG`-0,7 insgesamt betrachtet bessere Ergebnisse liefert. Die Unterschiede zwischen den Werten sind höchstens

Tabelle 7.15: Die Ergebnisse in der Klassifizierungsaufgabe für die Kombinationsmethoden `AVG` und `PROJ`; für jede Kombination aus Testverfahren, Klassifikator und Maß ist jeweils das beste Ergebnis markiert

		T.-T.-Teilung		Zehnf. Kreuzv.		Projektsp. Kreuzv.	
Klass.	Maß	`AVG`	`PROJ`	`AVG`	`PROJ`	`AVG`	`PROJ`
ZW	Genauigkeit	**0,68**	0,59	**0,61**	0,49	**0,66**	0,56
	g. D. F_1-Maß	**0,61**	0,53	**0,55**	0,45	**0,58**	0,50
SVM	Genauigkeit	**0,77**	0,68	**0,69**	0,59	**0,69**	0,60
	g. D. F_1-Maß	**0,77**	0,64	**0,69**	0,56	**0,69**	0,58
LR	Genauigkeit	**0,81**	0,65	**0,74**	0,58	**0,71**	0,59
	g. D. F_1-Maß	**0,80**	0,62	**0,73**	0,55	**0,71**	0,57
LSTM	Genauigkeit	**0,81**	0,63	**0,77**	0,56	**0,72**	0,60
	g. D. F_1-Maß	**0,79**	0,57	**0,76**	0,53	**0,69**	0,57

0,04. `AVG`-0,7 ist in 18 der insgesamt 24 betrachteten Werte besser als `AVG`-0,5, in drei schlechter und in drei gleich gut. Das unter allen Gewichtungen und allen Werten beste Ergebnis ist eine Genauigkeit von 0,83 und ein gewichteter Durchschnitt des F_1-Maßes von 0.81 für `AVG-0,7` in der Training-Test-Teilung mit dem LSTM. Bemerkenswert ist, dass `AVG`-0,7 für jedes der drei Testverfahren die besten Ergebnisse mit dem LSTM liefert. In den anderen Gewichtungen und `PROJ` lässt sich so eine klare Absetzung eines Klassifikators nicht beobachten.

Zusammenfassend liefert `AVG`-0,7 bessere Ergebnisse als `AVG`-0,5, und `AVG`-0,5 wiederum bessere Ergebnisse als `AVG`-0,3. Es lässt sich also feststellen, dass eine stärkere Gewichtung der generischen Worteinbettungen mindestens bis zum Gewicht 0,7 besser ist als eine stärkere Gewichtung der domänenspezifischen oder eine gleichstarke Gewichtung. Eine mögliche Feineinstellung der Gewichtung um 0,7 herum durch die Evaluation verschiedener kleinschrittig ausgewählter Gewichtungen könnte wohl noch etwas bessere Ergebnisse liefern, würde aber den Umfang dieser Bachelorarbeit überschreiten. Als finale Gewichtungen werden somit diejenigen aus `AVG`-0,7 festgelegt. Die Worteinbettungen in `AVG`-0,7 sind also das Endergebnis der Kombination und bilden die finalen domänenadaptierten Worteinbettungen.

7.2.2 Vergleich der Ergebnisse

Da nun die finalen domänenadaptierten Worteinbettungen gefunden wurden, werden ihre Leistung und die Leistungen der generischen und domänenspezifischen Worteinbettungen in der Klassifizierungsaufgabe miteinander verglichen. Die Ergebnisse in der Klassifizierungsaufgabe werden in Tabelle 7.17 gezeigt. Zudem sind in Abschnitt E des Anhangs alle Klassifizierungsberichte gesammelt.

Zuerst fällt im Vergleich der generischen und der domänenspezifischen Worteinbettungen auf, dass erstere insgesamt bessere Werte erzielen. Für das LSTM ist der Unterschied sehr gering, beziehungsweise sind die domänenspezifischen Worteinbettungen sogar besser. Die größte Verbesserung der domänenspezifischen Worteinbettungen ist dabei der gewichtete Durchschnitt des F_1-Maßes in der Training-Test-Teilung mit einem Differenzwert von 0,08.

Nimmt man die domänenadaptierten Worteinbettungen hinzu, lässt sich feststellen, das die domänenadaptierten Worteinbettungen deutlich die besten Ergebnisse liefern. Nur

Tabelle 7.16: Die Ergebnisse in der Klassifizierungsaufgabe für `AVG` mit Gewichtungen der generischen Worteinbettungen von 0,3, 0,5 und 0,7 (v. l. n. r.); für jede Kombination aus Testverfahren, Klassifikator und Maß ist jeweils das beste Ergebnis markiert

		T.-T.-Teilung			Zehnf. Kreuzv.			Projektsp. Kreuzv.		
Klass.	Maß	0,3	0,5	0,7	0,3	0,5	0,7	0,3	0,5	0,7
ZW	Genauigkeit	0,65	0,68	**0,71**	0,55	0,61	**0,65**	0,60	**0,66**	**0,66**
	g. D. F_1-Maß	0,58	0,61	**0,65**	0,50	0,55	**0,59**	0,53	**0,58**	**0,58**
SVM	Genauigkeit	0,74	0,77	**0,78**	0,67	0,69	**0,70**	0,64	0,69	**0,70**
	g. D. F_1-Maß	0,73	0,77	**0,78**	0,67	0,69	**0,70**	0,65	0,69	**0,70**
LR	Genauigkeit	0,80	**0,81**	0,78	0,68	**0,74**	0,73	0,69	0,71	**0,72**
	g. D. F_1-Maß	0,80	**0,81**	0,78	0,68	**0,73**	**0,73**	0,69	0,71	**0,72**
LSTM	Genauigkeit	0,80	0,81	**0,83**	0,71	0,77	**0,78**	0,70	0,72	**0,75**
	g. D. F_1-Maß	0,77	0,79	**0,81**	0,70	0,76	**0,78**	0,68	0,69	**0,73**

in der zehnfachen Kreuzvalidierung mit der Stützvektormaschine können die generischen Worteinbettungen gleich gute Werte aufweisen. Ansonsten haben die domänenadaptierten Worteinbettungen für alle Testverfahren, Klassifikatoren und Maße bessere Ergebnisse als die domänenspezifischen und die generischen Worteinbettungen. Wie schon im Vergleich der Gewichtungen festgestellt, liefern die domänenadaptierten Worteinbettungen mit dem LSTM als Klassifikator die besten Werte. Diese Eigenschaft haben sie bei der Kombination von den domänenspezifischen Worteinbettungen übernommen. Für das LSTM ist auch die Verbesserung zu den generischen Worteinbettungen besonders groß. In der Training-Test-Teilung verbessert sich die Genauigkeit um 0,11 auf 0,83 und im gewichteten Durchschnitt des F_1-Maßes sogar um 0,15 auf 0,81. In der zehnfachen Kreuzvalidierung ist die Genauigkeit mit 0,78 um 0,07 besser und der gewichtete Durchschnitt des F_1-Maßes mit ebenfalls 0,78 um 0,11. In der projektspezifischen Kreuzvalidierung ist die Genauigkeit mit 0,75 um 0,10 besser und der gewichtete Durchschnitt des F_1-Maßes mit 0,73 um 0,15.

Zusammenfassend lässt sich feststellen, dass sich die Kombination der generischen und der domänenspezifischen Worteinbettungen offensichtlich gelohnt hat. Die domänenadaptierten Worteinbettungen liefern klar bessere Ergebnisse als die generischen und die domänenspezifischen, bei umfassender Betrachtung vieler Testverfahren, Klassifikatoren und Maße. Anhand der Klassifizierungsaufgabe wurde gezeigt, dass die Verwendung der domänenadaptierten Worteinbettungen anstatt generischer in Aufgaben aus der Anforderungsdomäne Vorteile bringen kann. Es ist allerdings zu beachten, dass nur eine Aufgabe in der extrinsischen Evaluation verwendet wurde. Andere Aufgaben könnten möglicherweise andere Ergebnisse liefern.

Tabelle 7.17: Die Ergebnisse in der Klassifizierungsaufgabe für die generischen, die domänenspezifischen und die domänenadaptierten Worteinbettungen; für jede Kombination aus Testverfahren, Klassifikator und Maß ist jeweils das beste Ergebnis markiert

		T.-T.-Teilung			Zehnf. Kreuzv.			Projektsp. Kreuzv.		
Klass.	Maß	gen.	sp.	ad.	gen.	sp.	ad.	gen.	sp.	ad.
ZW	Genauigkeit	0,68	0,59	**0,71**	0,61	0,49	**0,65**	0,62	0,58	**0,66**
	g. D. F_1-Maß	0,62	0,54	**0,65**	0,55	0,45	**0,59**	0,55	0,51	**0,58**
SVM	Genauigkeit	0,74	0,66	**0,78**	**0,70**	0,57	**0,70**	0,69	0,56	**0,70**
	g. D. F_1-Maß	0,73	0,65	**0,78**	**0,70**	0,57	**0,70**	0,69	0,58	**0,70**
LR	Genauigkeit	0,77	0,72	**0,78**	0,71	0,58	**0,73**	0,70	0,58	**0,72**
	g. D. F_1-Maß	0,76	0,72	**0,77**	0,71	0,59	**0,73**	0,70	0,60	**0,72**
LSTM	Genauigkeit	0,72	0,75	**0,83**	0,71	0,67	**0,78**	0,65	0,62	**0,75**
	g. D. F_1-Maß	0,66	0,74	**0,81**	0,67	0,66	**0,78**	0,58	0,62	**0,73**

8 Zusammenfassung und Ausblick

Diese Bachelorarbeit hatte das Ziel der Bildung von Worteinbettungen für die Anforderungsdomäne. Sie fand im Rahmen des Projektes `INDIRECT` [Hey19] statt, das sich mit der automatisierten Generierung von Informationen zur Rückverfolgbarkeit zwischen Quelltext und Anforderungen beschäftigt. Die gebildeten Worteinbettungen sollten darauf überprüft werden, ob sie in Aufgaben aus der Anforderungsdomäne, wie sie zum Beispiel in `INDIRECT` vorkommen, bessere Ergebnisse liefern, als generische Worteinbettungen, die domänenspezifische Feinheiten und Besonderheiten weniger gut erfassen.

Im ersten Schritt wurde dafür ein Textkorpus aus Anforderungsbeschreibungen und anderen in der Anforderungsdomäne üblichen Dokumenten aufgebaut. Er umfasst insgesamt 21 458 Anforderungsbeschreibungen und 1680 Anwendererzählungen aus 65 verschiedenen Projekten. Nach einer Analyse verschiedener Worteinbettungsmodelle auf ihre Vor- und Nachteile für die Anwendung auf Anforderungen wurde das `fastText-Modell` ausgewählt. Zudem wurden in der Analyse verschiedene Kombinationsverfahren der domänenspezifischen mit generischen Worteinbettungen untersucht, um die Vorteile der domänenspezifischen und der generischen Worteinbettungen zu vereinen. Die daraus entstehenden domänenadaptierten Worteinbettungen sollten also zusätzlich zur Angepasstheit an die Anforderungsdomäne auch den Umfang der generischen Worteinbettungen nutzen. Es wurden die Methoden `PROJ`, welche eine CCA und anschließende lineare Kombination nutzt, und `AVG`, die eine Durchschnittsbildung vorsieht, für eine weitergehende Betrachtung in der Evaluation ausgewählt.

In der Implementierung wurde `fastText` auf dem vorverarbeiteten aufgebauten Textkorpus trainiert und verschiedene Kombinationen der domänenspezifischen und der generischen Worteinbettungen gebildet.

In der Evaluation wurden zu Beginn die domänenspezifischen Worteinbettungen intrinsisch evaluiert. Dabei wurden Wortähnlichkeiten untersucht und Clusteranalysen durchgeführt. Es wurde festgestellt, dass die domänenspezifischen Worteinbettungen manche domänenspezifische Feinheiten besser als die generischen Worteinbettungen erfassen. Zum Beispiel kann man auf Grundlage der ähnlichsten Wörter schließen, dass die domänenspezifischen Worteinbettungen das Wort *window* in seiner in Anforderungen vorherrschenden Bedeutung als Zeitfenster erfassen. In den generischen Worteinbettungen hingegen wird die in Anforderungen wenig vorkommende Bedeutung als Lichtöffnung erfasst. Aufgrund des umfassenderen Trainingskorpus besitzen die generischen Worteinbettungen allerdings ein umfassenderes Vokabular und stellen einige, meistens weniger häufig in der Anforderungsdomäne auftretende Wörter besser dar.

In der Evaluation der domänenadaptierten Worteinbettungen wurden verschiedene Kombinationen der generischen und domänenspezifischen Worteinbettungen extrinsisch evaluiert. Dabei wurde eine aus dem Projekt `INDIRECT` übernommene Aufgabe zur Klassifizierung von Sätzen aus Anforderungsbeschreibungen verwendet. Um ein möglichst umfassendes Bild der Qualität der Worteinbettungen zu erhalten, wurden verschiedene Testverfahren, Klassifikatoren und Maße untersucht. Eine Kombination mit der Methode `AVG` und einer Gewichtung der generischen Worteinbettungen von 0,7 erzielte die besten Ergebnisse. Der beste Wert dieser finalen domänenadaptierten Worteinbettungen ist eine Genauigkeit von 0,83 in der Training-Test-Teilung mit Verwendung eines LSTMs. Die domänenspezifischen Worteinbettungen liefern hier 0,75 und die generischen nur 0,72. Aus den Werten kann man schließen, das die domänenadaptierten Worteinbettungen in Aufgaben aus der Anforderungsdomäne Vorteile bringen können. Es wurde jedoch nur eine Aufgabe in der extrinsischen Evaluation verwendet, andere Aufgaben könnten möglicherweise andere Ergebnisse liefern.

Für die Zukunft gilt, dass die gebildeten Worteinbettungen noch auf anderen Aufgaben extrinsisch evaluiert werden können. Zudem kann ein größeres Textkorpus, der dann neu erschienene Anforderungsdokumente integriert, aufgebaut werden. Auf dem größeren Textkorpus trainierte Worteinbettungen könnten manche Wörter möglicherweise noch besser darstellen. Weiterhin könnte eine mögliche Feineinstellung der Gewichtungen bei der Kombination eventuell noch bessere Ergebnisse liefern.

Literaturverzeichnis

[Bak18] Bakarov, Amir: A Survey of Word Embeddings Evaluation Methods. (2018), Januar (zitiert auf Seite 14).

[BDK14] Baroni, Marco ; Dinu, Georgiana ; Kruszewski, Germán: Don't Count, Predict! A Systematic Comparison of Context-Counting vs. Context-Predicting Semantic Vectors. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Baltimore, Maryland : Association for Computational Linguistics, Juni 2014, S. 238–247 (zitiert auf den Seiten 14 und 27).

[BG16] Bilenko, Natalia Y. ; Gallant, Jack L.: Pyrcca: Regularized Kernel Canonical Correlation Analysis in Python and Its Applications to Neuroimaging. In: *Frontiers in Neuroinformatics* 10 (2016). `http://dx.doi.org/10.3389/fninf.2016.00049`. – DOI 10.3389/fninf.2016.00049. – ISSN 1662–5196 (zitiert auf Seite 36).

[BGJM17] Bojanowski, Piotr ; Grave, Edouard ; Joulin, Armand ; Mikolov, Tomas: Enriching Word Vectors with Subword Information. In: *Transactions of the Association for Computational Linguistics* 5 (2017), S. 135–146. `http://dx.doi.org/10.1162/tacl_a_00051`. – DOI 10.1162/tacl_a_00051 (zitiert auf den Seiten 9, 27, 28 und 35).

[BL04] Bird, Steven ; Loper, Edward: NLTK: The Natural Language Toolkit. In: *Proceedings of the ACL Interactive Poster and Demonstration Sessions*. Barcelona, Spain : Association for Computational Linguistics, Juli 2004, S. 214–217 (zitiert auf Seite 35).

[CA18] Chollet, François ; Allaire, J. J.: *Deep Learning mit R und Keras: Das Praxis-Handbuch von den Entwicklern von Keras und RStudio*. MITP-Verlags GmbH & Co. KG, 2018. – ISBN 978–3–95845–895–6 (zitiert auf den Seiten 5 und 7).

[CB18] Coates, Joshua ; Bollegala, Danushka: Frustratingly Easy Meta-Embedding – Computing Meta-Embeddings by Averaging Source Word Embeddings. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. New Orleans, Louisiana : Association for Computational Linguistics, Juni 2018, S. 194–198 (zitiert auf den Seiten 18, 32 und 53).

[CCC03] Cleland-Huang, J. ; Chang, C.K. ; Christensen, M.: Event-Based Traceability for Managing Evolutionary Change. In: *IEEE Transactions on Software Engineering* 29 (2003), September, Nr. 9, S. 796–810. `http://dx.doi.org/10.1109/TSE.2003.1232285`. – DOI 10.1109/TSE.2003.1232285. – ISSN 2326–3881 (zitiert auf Seite 23).

[CMLP07] Cleland-Huang, Jane ; Mazrouee, Sepideh ; Liguo, Huang ; Port, Dan: Nfr. (2007), März. http://dx.doi.org/10.5281/ZENODO.268542. – DOI 10.5281/ZENODO.268542 (zitiert auf den Seiten 23 und 54).

[CRR+17] Che, Xiaoyin ; Ring, Nico ; Raschkowski, Willi ; Yang, Haojin ; Meinel, Christoph: Traversal-Free Word Vector Evaluation in Analogy Space. In: *Proceedings of the 2nd Workshop on Evaluating Vector Space Representations for NLP*. Copenhagen, Denmark : Association for Computational Linguistics, 2017, S. 11–15 (zitiert auf Seite 15).

[CSB+05] Cleland-Huang, Jane ; Settimi, Raffaella ; BenKhadra, Oussama ; Berezhanskaya, Eugenia ; Christina, Selvia: Goal-Centric Traceability for Managing Non-Functional Requirements. In: *Proceedings of the 27th International Conference on Software Engineering*. St. Louis, MO, USA : Association for Computing Machinery, Mai 2005 (ICSE '05). – ISBN 978–1–58113–963–1, S. 362–371 (zitiert auf den Seiten 23 und 54).

[Dal18] Dalpiaz, Fabiano: Requirements Data Sets (User Stories). (2018), Juli. http://dx.doi.org/10.17632/7ZBK8ZSD8Y.1. – DOI 10.17632/7ZBK8ZSD8Y.1 (zitiert auf Seite 23).

[ECS18] Efstathiou, Vasiliki ; Chatzilenas, Christos ; Spinellis, Diomidis: Word Embeddings for the Software Engineering Domain. In: *Proceedings of the 15th International Conference on Mining Software Repositories - MSR '18*. Gothenburg, Sweden : ACM Press, 2018. – ISBN 978–1–4503–5716–6, S. 38–41 (zitiert auf den Seiten 16 und 23).

[Eis19] Eisenstein, Jacob: *Introduction to Natural Language Processing*. MIT Press, 2019. – ISBN 978–0–262–04284–0 (zitiert auf den Seiten 3 und 6).

[FDG17] Ferrari, Alessio ; Donati, Beatrice ; Gnesi, Stefania: Detecting Domain-Specific Ambiguities: An NLP Approach Based on Wikipedia Crawling and Word Embeddings, 2017, S. 393–399 (zitiert auf den Seiten 16 und 22).

[FSG17] Ferrari, Alessio ; Spagnolo, Giorgio O. ; Gnesi, Stefania: PURE: A Dataset of Public Requirements Documents. In: *2017 IEEE 25th International Requirements Engineering Conference (RE)*. Lisbon, Portugal : IEEE, September 2017. – ISBN 978–1–5386–3191–1, S. 502–505 (zitiert auf den Seiten 13 und 22).

[FTRD16] Faruqui, Manaal ; Tsvetkov, Yulia ; Rastogi, Pushpendre ; Dyer, Chris: Problems With Evaluation of Word Embeddings Using Word Similarity Tasks. In: *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*. Berlin, Germany : Association for Computational Linguistics, 2016, S. 30–35 (zitiert auf Seite 15).

[GAL+06] Guthrie, David ; Allison, Ben ; Liu, Wei ; Guthrie, Louise ; Wilks, Yorick: A Closer Look at Skip-Gram Modelling. In: *LREC*, 2006 (zitiert auf Seite 5).

[GBC16] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep Learning*. MIT Press, 2016 (zitiert auf Seite 5).

[GFEC16] Ghannay, Sahar ; Favre, Benoit ; Estève, Yannick ; Camelin, Nathalie: Word Embedding Evaluation and Combination. In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*. Portorož, Slovenia : European Language Resources Association (ELRA), Mai 2016, S. 300–305 (zitiert auf den Seiten 17 und 31).

[Hey19] Hey, Tobias: INDIRECT: Intent-Driven Requirements-to-Code Traceability. In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. Montreal, QC, Canada : IEEE, Mai 2019. – ISBN 978–1–72811–764–5, S. 190–191 (zitiert auf den Seiten 1, 11, 21, 54 und 59).

[HHD09] Holbrook, Elizabeth A. ; Hayes, Jane H. ; Dekhtyar, Alex: Toward Automating Requirements Satisfaction Assessment. In: *2009 17th IEEE International Requirements Engineering Conference* (2009), S. 149–158. `http://dx.doi.org/10.1109/RE.2009.10`. – DOI 10.1109/RE.2009.10 (zitiert auf Seite 23).

[Ins13] Institute, Project M.: *Software Extension to the PMBOK® Guide Fifth Edition*. Project Management Institute, 2013. – ISBN 978–1–62825–041–1 (zitiert auf Seite 10).

[ISO17] ISO/IEC/IEEE International Standard - Systems and Software Engineering–Vocabulary. In: *ISO/IEC/IEEE 24765:2017(E)* (2017), August, S. 1–541. `http://dx.doi.org/10.1109/IEEESTD.2017.8016712`. – DOI 10.1109/IEEESTD.2017.8016712 (zitiert auf den Seiten 9 und 10).

[KSLS18] Kameswara Sarma, Prathusha ; Liang, Yingyu ; Sethares, Bill: Domain Adapted Word Embeddings for Improved Sentiment Classification. In: *Proceedings of the Workshop on Deep Learning Approaches for Low-Resource NLP*. Melbourne : Association for Computational Linguistics, 2018, S. 51–59 (zitiert auf den Seiten 19, 31, 36 und 53).

[MCCD13] Mikolov, Tomas ; Corrado, G.s ; Chen, Kai ; Dean, Jeffrey: Efficient Estimation of Word Representations in Vector Space, 2013, S. 1–12 (zitiert auf den Seiten 8 und 26).

[MGB+18] Mikolov, Tomas ; Grave, Edouard ; Bojanowski, Piotr ; Puhrsch, Christian ; Joulin, Armand: Advances in Pre-Training Distributed Word Representations. In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Miyazaki, Japan : European Language Resources Association (ELRA), Mai 2018 (zitiert auf den Seiten 33 und 36).

[Mit97] Mitchell, Tom M.: *Machine Learning*. McGraw-Hill, 1997. – ISBN 978–0–07–115467–3 (zitiert auf Seite 5).

[MSC+13] Mikolov, Tomas ; Sutskever, Ilya ; Chen, Kai ; Corrado, Greg ; Dean, Jeffrey: Distributed Representations of Words and Phrases and Their Compositionality. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. USA : Curran Associates Inc., 2013 (NIPS'13), S. 3111–3119 (zitiert auf den Seiten 8, 24 und 26).

[MYZ13] Mikolov, Tomas ; Yih, Wen-tau ; Zweig, Geoffrey: Linguistic Regularities in Continuous Space Word Representations. In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Atlanta, Georgia : Association for Computational Linguistics, Juni 2013, S. 746–751 (zitiert auf den Seiten 7 und 38).

[NØL18] Nooralahzadeh, Farhad ; Øvrelid, Lilja ; Lønning, Jan T.: Evaluation of Domain-Specific Word Embeddings Using Knowledge Resources. In: *Proceedings of the Eleventh International Conference on Language Resources and*

Evaluation (LREC 2018). Miyazaki, Japan : European Language Resources Association (ELRA), Mai 2018 (zitiert auf den Seiten 17 und 27).

[OM10] O'Keeffe, Anne ; McCarthy, Michael: *The Routledge Handbook of Corpus Linguistics*. Routledge, 2010. – ISBN 978–1–135–15363–2 (zitiert auf Seite 13).

[Osb16] Osborne, Jason W.: *Regression & Linear Modeling: Best Practices and Modern Methods*. SAGE Publications, 2016. – ISBN 978–1–5063–0276–8 (zitiert auf Seite 7).

[PNI+18] Peters, Matthew ; Neumann, Mark ; Iyyer, Mohit ; Gardner, Matt ; Clark, Christopher ; Lee, Kenton ; Zettlemoyer, Luke: Deep Contextualized Word Representations. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana : Association for Computational Linguistics, 2018, S. 2227–2237 (zitiert auf den Seiten 9 und 27).

[PSM14] Pennington, Jeffrey ; Socher, Richard ; Manning, Christopher: Glove: Global Vectors for Word Representation. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar : Association for Computational Linguistics, 2014, S. 1532–1543 (zitiert auf den Seiten 8 und 27).

[PVG+11] Pedregosa, Fabian ; Varoquaux, Gaël ; Gramfort, Alexandre ; Michel, Vincent ; Thirion, Bertrand ; Grisel, Olivier ; Blondel, Mathieu ; Prettenhofer, Peter ; Weiss, Ron ; Dubourg, Vincent ; Vanderplas, Jake ; Passos, Alexandre ; Cournapeau, David ; Brucher, Matthieu ; Perrot, Matthieu ; Duchesnay, Édouard: Scikit-Learn: Machine Learning in Python. In: *Journal of Machine Learning Research* 12 (2011), Nr. 85, S. 2825–2830. – ISSN 1533–7928 (zitiert auf Seite 36).

[RK19] Risch, Julian ; Krestel, Ralf: Domain-Specific Word Embeddings for Patent Classification. In: *Data Technologies and Applications* 53 (2019), Februar, Nr. 1, S. 108–122. http://dx.doi.org/10.1108/DTA-01-2019-0002. – DOI 10.1108/DTA–01–2019–0002. – ISSN 2514–9288 (zitiert auf den Seiten 17 und 28).

[Sah08] Sahlgren, Magnus: The Distributional Hypothesis. In: *Italian Journal of Linguistics* 20 (2008), Januar (zitiert auf Seite 7).

[SC08] Steinwart, Ingo ; Christmann, Andreas: *Support Vector Machines*. Springer Science & Business Media, 2008. – ISBN 978–0–387–77242–4 (zitiert auf Seite 7).

[She17] Sheppard, Clinton: *Tree-Based Machine Learning Algorithms: Decision Trees, Random Forests, and Boosting*. CreateSpace Independent Publishing Platform, 2017. – ISBN 978–1–975860–97–4 (zitiert auf Seite 7).

[Ska18] Skansi, Sandro: *Introduction to Deep Learning: From Logical Calculus to Artificial Intelligence*. Springer, 2018. – ISBN 978–3–319–73004–2 (zitiert auf Seite 4).

[SLMJ15] Schnabel, Tobias ; Labutov, Igor ; Mimno, David ; Joachims, Thorsten: Evaluation Methods for Unsupervised Word Embeddings. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal : Association for Computational Linguistics, September 2015, S. 298–307 (zitiert auf Seite 15).

[SNZ18] Shaukat, Zain ; Naseem, Rashid ; Zubair, Muhammad: Software Requirement Risk Prediction Dataset. (2018), März. `http://dx.doi.org/10.5281/ZENODO.1209601`. – DOI 10.5281/ZENODO.1209601 (zitiert auf den Seiten 23 und 54).

[SSM05] Sayyad Shirabad, J. ; Menzies, T.J.: The PROMISE Repository of Software Engineering Databases. (2005) (zitiert auf Seite 23).

[Tom19] Tomova, Dana: *Bestimmung Der Semantischen Funktion von Sätzen in Anforderungsbeschreibungen*, Karlsruher Institut für Technologie (KIT) – IPD Tichy, Bachelor's Thesis, Oktober 2019 (zitiert auf Seite 54).

[YS16] Yin, Wenpeng ; Schütze, Hinrich: Learning Word Meta-Embeddings. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany : Association for Computational Linguistics, 2016, S. 1351–1360 (zitiert auf den Seiten 18 und 31).

Anhang

A Projekte im Textkorpus

Tabelle A.1: Anzahl der Anwendererzählungen und Wörter sowie Vokabulargröße der einzelnen im Korpus enthaltenen Projekte mit Anwendererzählungen

Projekt	Anwendererz.	Wörter	Vokabulargr.
federal spending	98	2086	480
loudoun	58	1579	350
recycling	51	1287	340
open spending	53	1641	336
frictionless	66	1749	342
srum alliance	97	2571	481
nsf	73	1753	390
camperplus	55	1397	264
planning poker	53	1460	365
datahub	67	1841	381
mis	68	1536	425
cask	64	1627	240
neurohub	102	2198	500
alfred	138	2441	489
bad camp	69	1885	402
rdadmp	83	2246	399
archives space	57	868	222
unibath	53	1462	364
duraspace	100	2015	305
racdam	100	2122	443
culrepo	115	3320	779
zooniverse	60	1061	268

Tabelle A.2: Anzahl der Anforderungsbeschreibungen und Wörter sowie Vokabulargröße der einzelnen im Korpus enthaltenen Projekte mit Anforderungsbeschreibungen

Projekt	Anforderungsb.	Wörter	Vokabulargr.
CCTNS	182	4782	1241
GAMMA-J	410	4398	797
Gemini Project	2103	21 413	2923
THEMAS	440	3681	613
DII	585	4388	1291
TCS	954	15 600	2645
Qheadache	102	1652	391
Microcare	704	5642	1082
PHIN	35	4992	1007
EIRENE 15	1195	15 470	2304
EIRENE 7	1864	17 727	1738
ERTMS/ETCS	274	4275	692
Get Real	172	2030	739
KeePass Password Safe	373	5820	1005
CDN Peering	294	4602	1127
PEPPOL	3098	39 523	3397
Video Search Engine	179	1977	521
BLIT	73	1449	438
UUIS	56	498	213
BEYOND	620	6665	1489
C2C	134	873	234
Elforsk	335	4113	923
APAF	109	497	141
Space Fractions	40	458	217
XRT	76	823	322
Libra	25	760	288
EVLA Backend	282	2532	691
EVLA Monitor & Control	163	1411	508
agentMom	34	278	74
TACHOnet	114	987	343
ColorKast	119	1415	457
Nenios	129	851	259
NPAC SMS	3570	61 640	2740
HATS	557	6913	888
Risk Prediction	299	4832	980
NFR	625	12 134	1756
CM1	455	22 322	2441
MODIS	68	1321	388
EBT	41	559	183
GANNT	86	2127	362
Ice Breaker	201	1738	423
iTrust	131	7862	1174
WARC	152	3081	730

B Trigramme im Textkorpus

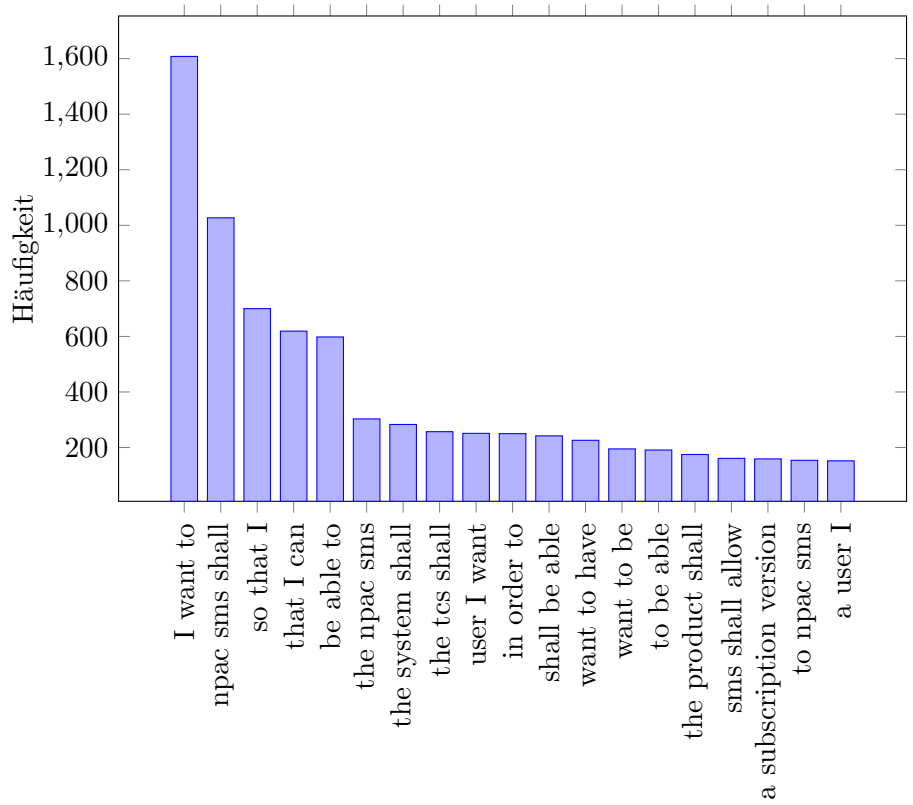

Abbildung B.1: Häufigkeiten der Trigrammen von Wörtern im Textkorpus

C Stoppwörter

Die folgende Liste von Stoppwörtern wird vom NLTK bereitgestellt:

its, weren, yourself, doesn, on, from, do, mustn't, again, between, m, when, herself, didn, where, needn't, it's, under, isn't, while, aren't, you'd, ourselves, o, weren't, most, them, hers, hasn, isn, d, above, doesn't, should've, too, an, he, wasn, won, both, y, did, are, before, won't, themselves, more, does, here, you're, with, up, about, through, after, why, t, by, the, yours, into, we, should, shouldn't, itself, these, don't, off, will, him, has, for, been, your, is, so, down, ve, haven't, s, shouldn, his, very, those, and, didn't, our, you've, over, such, no, some, yourselves, mustn, she, i, further, now, than, ain, couldn, or, other, whom, mightn, theirs, ours, her, needn, if, was, in, few, be, any, as, being, ma, you, out, hadn, me, that'll, all, that, re, their, but, then, shan, myself, ll, which, a, couldn't, mightn't, once, against, they, because, himself, hasn't, at, just, wouldn't, of, am, my, nor, were, she's, shan't, what, wasn't, haven, doing, had, who, there, don, having, have, during, until, it, same, wouldn, only, to, not, hadn't, how, each, you'll, can, below, this, aren, own.

D Cluster

Tabelle D.3: Die Anzahl der Wortvektoren und der Repräsentant für jeden der durch die agglomerative Clusteranalyse erhaltenen Cluster

Cluster	Anzahl	Repräsentant
1	66	interface
2	169	number
3	287	shall
4	336	dpu
5	38	enter
6	217	user
7	40	service
8	13	due
9	556	want
10	498	vcd
11	26	id
12	17	range
13	22	given
14	21	allow
15	52	control
16	22	may
17	28	response
18	87	operator
19	469	system
20	15	first
21	82	using
22	28	date
23	53	authority
24	24	reference
25	20	include
26	11	data
27	9	example
28	7	test
29	15	see
30	15	requirements
31	8	users
32	19	collection
33	11	period
34	41	map
35	7	set
36	11	plan
37	10	usage
38	21	read
39	19	solution
40	27	section
41	12	event
42	15	mission
43	10	format
44	5	code
45	22	either
46	24	etc

47	7	type
48	16	public
49	13	change
50	40	language
51	15	record
52	13	active
53	27	capability
54	24	maintenance
55	16	support
56	16	configuration
57	8	automatically
58	3	second
59	33	results
60	42	use
61	24	presented
62	8	restriction
63	20	status
64	15	appropriate
65	19	task
66	49	information
67	9	indication
68	7	order
69	14	list
70	7	defined
71	18	back
72	11	general
73	7	stored
74	3	one
75	18	scheduled
76	7	selection
77	11	sequence
78	16	power
79	17	access
80	24	functions
81	28	related
82	54	communication
83	66	able

E Klassifizierungsberichte

E.1 Generische Worteinbettungen

Tabelle E.4: 80 zu 20 Training-Test-Teilung: Zufallswald: Klassifizierungsbericht mit den generischen Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,67416	0,95238	0,78947	63
Aggregation	0,00000	0,00000	0,00000	13
Ereignis	0,70732	0,70732	0,70732	41
Zustand	0,66667	0,12500	0,21053	16
Genauigkeit			0,68421	133
Makro-Durchschnitt	0,51204	0,44617	0,42683	133
gewichteter Durchschnitt	0,61758	0,68421	0,61733	133

Tabelle E.5: 80 zu 20 Training-Test-Teilung: Stützvektormaschine: Klassifizierungsbericht mit den generischen Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,78261	0,85714	0,81818	63
Aggregation	0,47059	0,61538	0,53333	13
Ereignis	0,79487	0,75610	0,77500	41
Zustand	0,62500	0,31250	0,41667	16
Genauigkeit			0,73684	133
Makro-Durchschnitt	0,66827	0,63528	0,63580	133
gewichteter Durchschnitt	0,73693	0,73684	0,72873	133

Tabelle E.6: 80 zu 20 Training-Test-Teilung: Logistische Regression: Klassifizierungsbericht mit den generischen Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,81818	0,85714	0,83721	63
Aggregation	0,57143	0,61538	0,59259	13
Ereignis	0,77778	0,85366	0,81395	41
Zustand	0,62500	0,31250	0,41667	16
Genauigkeit			0,76692	133
Makro-Durchschnitt	0,69810	0,65967	0,66511	133
gewichteter Durchschnitt	0,75837	0,76692	0,75554	133

Tabelle E.7: 80 zu 20 Training-Test-Teilung: LSTM-Klassifizierungsbericht mit den generischen Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,73750	0,93651	0,82517	63
Aggregation	0,25000	0,07692	0,11765	13
Ereignis	0,72917	0,85366	0,78652	41
Zustand	1,00000	0,06250	0,11765	16
Genauigkeit			0,72180	133
Makro-Durchschnitt	0,67917	0,48240	0,46175	133
gewichteter Durchschnitt	0,71886	0,72180	0,65898	133

Tabelle E.8: Zehnfache Kreuzvalidierung: Zufallswald: Klassifizierungsbericht mit den generischen Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,64165	0,84665	0,73003	313
Aggregation	1,00000	0,06061	0,11429	66
Ereignis	0,54772	0,64078	0,59060	206
Zustand	1,00000	0,06410	0,12048	78
Genauigkeit			0,61237	663
Makro-Durchschnitt	0,79734	0,40303	0,38885	663
gewichteter Durchschnitt	0,69029	0,61237	0,55370	663

Tabelle E.9: Zehnfache Kreuzvalidierung: Stützvektormaschine: Klassifizierungsbericht mit den generischen Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,77812	0,79553	0,78673	313
Aggregation	0,53333	0,48485	0,50794	66
Ereignis	0,71090	0,72816	0,71942	206
Zustand	0,50000	0,46154	0,48000	78
Genauigkeit			0,70437	663
Makro-Durchschnitt	0,63059	0,61752	0,62352	663
gewichteter Durchschnitt	0,70015	0,70437	0,70198	663

Tabelle E.10: Zehnfache Kreuzvalidierung: Logistische Regression: Klassifizierungsbericht mit den generischen Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,78816	0,80831	0,79811	313
Aggregation	0,50000	0,34848	0,41071	66
Ereignis	0,71300	0,77184	0,74126	206
Zustand	0,52055	0,48718	0,50331	78
Genauigkeit			0,71342	663
Makro-Durchschnitt	0,63043	0,60395	0,61335	663
gewichteter Durchschnitt	0,70464	0,71342	0,70720	663

Tabelle E.11: Zehnfache Kreuzvalidierung: LSTM: Klassifizierungsbericht mit den generischen Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,74074	0,89457	0,81042	313
Aggregation	0,47368	0,13636	0,21176	66
Ereignis	0,69710	0,81553	0,75168	206
Zustand	0,64000	0,20513	0,31068	78
Genauigkeit			0,71342	663
Makro-Durchschnitt	0,63788	0,51290	0,52114	663
gewichteter Durchschnitt	0,68874	0,71342	0,67378	663

Tabelle E.12: Projektspezifische Kreuzvalidierung: Zufallswald: Klassifizierungsbericht mit den generischen Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,64965	0,89457	0,75269	313
Aggregation	0,00000	0,00000	0,00000	66
Ereignis	0,59375	0,64563	0,61860	206
Zustand	1,0000	0,01282	0,02532	78
Genauigkeit			0,62443	663
Makro-Durchschnitt	0,56085	0,38826	0,34915	663
gewichteter Durchschnitt	0,60883	0,62443	0,55053	663

Tabelle E.13: Projektspezifische Kreuzvalidierung: Stützvektormaschine: Klassifizierungsbericht mit den generischen Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,78914	0,78914	0,78914	313
Aggregation	0,42424	0,42424	0,42424	66
Ereignis	0,72277	0,70874	0,71569	206
Zustand	0,42683	0,44872	0,43750	78
Genauigkeit			0,68778	663
Makro-Durchschnitt	0,59075	0,59271	0,59164	663
gewichteter Durchschnitt	0,68957	0,68778	0,68862	663

Tabelle E.14: Projektspezifische Kreuzvalidierung: Logistische Regression: Klassifizierungsbericht mit den generischen Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,83333	0,78275	0,80725	313
Aggregation	0,43077	0,42424	0,42748	66
Ereignis	0,71493	0,76699	0,74005	206
Zustand	0,40964	0,43590	0,42236	78
Genauigkeit			0,70136	663
Makro-Durchschnitt	0,59717	0,60247	0,59928	663
gewichteter Durchschnitt	0,70662	0,70136	0,70328	663

Tabelle E.15: Projektspezifische Kreuzvalidierung: LSTM: Klassifizierungsbericht mit den generischen Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,62449	0,97764	0,76214	313
Aggregation	0,57143	0,05405	0,09877	74
Ereignis	0,71069	0,55941	0,62604	202
Zustand	0,85714	0,08108	0,14815	74
Genauigkeit			0,64706	663
Makro-Durchschnitt	0,69094	0,41804	0,40877	663
gewichteter Durchschnitt	0,67080	0,64706	0,57810	663

E.2 Domänenspezifische Worteinbettungen

Tabelle E.16: 80 zu 20 Training-Test-Teilung: Zufallswald: Klassifizierungsbericht mit den domänenspezifischen Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,62651	0,82540	0,71233	63
Aggregation	0,00000	0,00000	0,00000	13
Ereignis	0,55556	0,60976	0,58140	41
Zustand	0,50000	0,12500	0,20000	16
Genauigkeit			0,59398	133
Makro-Durchschnitt	0,42052	0,39004	0,37343	133
gewichteter Durchschnitt	0,52818	0,59398	0,54071	133

Tabelle E.17: 80 zu 20 Training-Test-Teilung: Stützvektormaschine: Klassifizierungsbericht mit den domänenspezifischen Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,70667	0,84127	0,76812	63
Aggregation	0,42857	0,46154	0,44444	13
Ereignis	0,73529	0,60976	0,66667	41
Zustand	0,40000	0,25000	0,30769	16
Genauigkeit			0,66165	133
Makro-Durchschnitt	0,56763	0,54064	0,54673	133
gewichteter Durchschnitt	0,65142	0,66165	0,64982	133

Tabelle E.18: 80 zu 20 Training-Test-Teilung: Logistische Regression: Klassifizierungsbericht mit den domänenspezifischen Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,73913	0,80952	0,77273	63
Aggregation	0,72727	0,61538	0,66667	13
Ereignis	0,77500	0,75610	0,76543	41
Zustand	0,46154	0,37500	0,41379	16
Genauigkeit			0,72180	133
Makro-Durchschnitt	0,67574	0,63900	0,65465	133
gewichteter Durchschnitt	0,71563	0,72180	0,71693	133

Tabelle E.19: 80 zu 20 Training-Test-Teilung: LSTM-Klassifizierungsbericht mit den domänenspezifischen Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,76000	0,90476	0,82609	63
Aggregation	0,50000	0,30769	0,38095	13
Ereignis	0,81081	0,73171	0,76923	41
Zustand	0,69231	0,56250	0,62096	16
Genauigkeit			0,75188	133
Makro-Durchschnitt	0,69078	0,62667	0,64924	133
gewichteter Durchschnitt	0,74211	0,75188	0,74034	133

Tabelle E.20: Zehnfache Kreuzvalidierung: Zufallswald: Klassifizierungsbericht mit den domänenspezifischen Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,54663	0,67412	0,60372	313
Aggregation	0,57143	0,06061	0,10959	66
Ereignis	0,39847	0,50485	0,44540	206
Zustand	0,55556	0,06410	0,11494	78
Genauigkeit			0,48869	663
Makro-Durchschnitt	0,51802	0,32592	0,31841	663
gewichteter Durchschnitt	0,50411	0,48869	0,44783	663

Tabelle E.21: Zehnfache Kreuzvalidierung: Stützvektormaschine: Klassifizierungsbericht mit den domänenspezifischen Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,68525	0,66773	0,67638	313
Aggregation	0,30769	0,30303	0,30534	66
Ereignis	0,60190	0,61650	0,60911	206
Zustand	0,28049	0,29487	0,28750	78
Genauigkeit			0,57164	663
Makro-Durchschnitt	0,46883	0,47053	0,46958	663
gewichteter Durchschnitt	0,57415	0,57164	0,57279	663

Tabelle E.22: Zehnfache Kreuzvalidierung: Logistische Regression: Klassifizierungsbericht mit den domänenspezifischen Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,71223	0,63259	0,67005	313
Aggregation	0,35484	0,33333	0,34375	66
Ereignis	0,60435	0,67476	0,63761	206
Zustand	0,29032	0,34615	0,31579	78
Genauigkeit			0,58220	663
Makro-Durchschnitt	0,49043	0,49671	0,49180	663
gewichteter Durchschnitt	0,59350	0,58220	0,58581	663

Tabelle E.23: Zehnfache Kreuzvalidierung: LSTM: Klassifizierungsbericht mit den domänenspezifischen Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,76087	0,78275	0,77165	313
Aggregation	0,42857	0,31818	0,36522	66
Ereignis	0,62705	0,74272	0,68000	206
Zustand	0,50000	0,30769	0,38095	78
Genauigkeit			0,66817	663
Makro-Durchschnitt	0,57912	0,53784	0,54946	663
gewichteter Durchschnitt	0,65552	0,66817	0,65675	663

Tabelle E.24: Projektspezifische Kreuzvalidierung: Zufallswald: Klassifizierungsbericht mit den domänenspezifischen Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,58475	0,88179	0,70318	313
Aggregation	0,36364	0,06061	0,10390	66
Ereignis	0,58333	0,50971	0,54404	206
Zustand	0,0000	0,0000	0,0000	78
Genauigkeit			0,58069	663
Makro-Durchschnitt	0,38293	0,36303	0,33778	663
gewichteter Durchschnitt	0,49350	0,58069	0,51135	663

Tabelle E.25: Projektspezifische Kreuzvalidierung: Stützvektormaschine: Klassifizierungsbericht mit den domänenspezifischen Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,78455	0,61661	0,69052	313
Aggregation	0,21101	0,34848	0,26286	66
Ereignis	0,63184	0,61650	0,62408	206
Zustand	0,24299	0,33333	0,28108	78
Genauigkeit			0,55656	663
Makro-Durchschnitt	0,46760	0,47873	0,46463	663
gewichteter Durchschnitt	0,61630	0,55656	0,57913	663

Tabelle E.26: Projektspezifische Kreuzvalidierung: Logistische Regression: Klassifizierungsbericht mit den domänenspezifischen Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,81624	0,61022	0,69835	313
Aggregation	0,28571	0,42424	0,34146	66
Ereignis	0,62896	0,67476	0,65105	206
Zustand	0,25455	0,35897	0,29787	78
Genauigkeit			0,58220	663
Makro-Durchschnitt	0,49636	0,51705	0,49719	663
gewichteter Durchschnitt	0,63916	0,58220	0,60101	663

Tabelle E.27: Projektspezifische Kreuzvalidierung: LSTM: Klassifizierungsbericht mit den domänenspezifischen Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,78777	0,69968	0,74112	313
Aggregation	0,34375	0,29730	0,31884	74
Ereignis	0,55474	0,75248	0,63866	202
Zustand	0,42553	0,27027	0,33058	74
Genauigkeit			0,62293	663
Makro-Durchschnitt	0,52795	0,50493	0,50730	663
gewichteter Durchschnitt	0,62678	0,62293	0,61695	663

E.3 Domänenadaptierte Worteinbettungen

Tabelle E.28: 80 zu 20 Training-Test-Teilung: Zufallswald: Klassifizierungsbericht mit den domänenadaptierten Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,71765	0,96825	0,82432	63
Aggregation	0,00000	0,00000	0,00000	13
Ereignis	0,68889	0,75610	0,72093	41
Zustand	1,00000	0,18750	0,31579	16
Genauigkeit			0,71429	133
Makro-Durchschnitt	0,60163	0,47796	0,46526	133
gewichteter Durchschnitt	0,67260	0,71429	0,65070	133

Tabelle E.29: 80 zu 20 Training-Test-Teilung: Stützvektormaschine: Klassifizierungsbericht mit den domänenadaptierten Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,83582	0,88889	0,86154	63
Aggregation	0,62500	0,76923	0,68966	13
Ereignis	0,83333	0,73171	0,77922	41
Zustand	0,57143	0,50000	0,53333	16
Genauigkeit			0,78195	133
Makro-Durchschnitt	0,71640	0,72246	0,71594	133
gewichteter Durchschnitt	0,78264	0,78195	0,77988	133

Tabelle E.30: 80 zu 20 Training-Test-Teilung: Logistische Regression: Klassifizierungsbericht mit den domänenadaptierten Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,81690	0,92063	0,86567	63
Aggregation	0,69231	0,69231	0,69231	13
Ereignis	0,80000	0,78049	0,79012	41
Zustand	0,55556	0,31250	0,40000	16
Genauigkeit			0,78195	133
Makro-Durchschnitt	0,71619	0,67648	0,68703	133
gewichteter Durchschnitt	0,76807	0,78195	0,76942	133

Tabelle E.31: 80 zu 20 Training-Test-Teilung: LSTM-Klassifizierungsbericht mit den domänenadaptierten Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,79487	0,98413	0,87943	63
Aggregation	0,75000	0,46154	0,57143	13
Ereignis	0,92308	0,87805	0,90000	41
Zustand	0,75000	0,37500	0,50000	16
Genauigkeit			0,82707	133
Makro-Durchschnitt	0,80449	0,67468	0,71272	133
gewichteter Durchschnitt	0,82461	0,82707	0,81002	133

Tabelle E.32: Zehnfache Kreuzvalidierung: Zufallswald: Klassifizierungsbericht mit den domänenadaptierten Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,66098	0,86581	0,74965	313
Aggregation	0,80000	0,06061	0,11268	66
Ereignis	0,61983	0,72816	0,66964	206
Zustand	0,83333	0,06410	0,11905	78
Genauigkeit			0,64857	663
Makro-Durchschnitt	0,72854	0,42967	0,41276	663
gewichteter Durchschnitt	0,68231	0,64857	0,58720	663

Tabelle E.33: Zehnfache Kreuzvalidierung: Stützvektormaschine: Klassifizierungsbericht mit den domänenadaptierten Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,76147	0,79553	0,77812	313
Aggregation	0,52830	0,42424	0,47059	66
Ereignis	0,73430	0,73786	0,73608	206
Zustand	0,47368	0,46154	0,46753	78
Genauigkeit			0,70136	663
Makro-Durchschnitt	0,62444	0,60479	0,61308	663
gewichteter Durchschnitt	0,69596	0,70136	0,69791	663

Tabelle E.34: Zehnfache Kreuzvalidierung: Logistische Regression: Klassifizierungsbericht mit den domänenadaptierten Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,80247	0,83067	0,81633	313
Aggregation	0,64444	0,43939	0,52252	66
Ereignis	0,73423	0,79126	0,76168	206
Zustand	0,47222	0,43590	0,45333	78
Genauigkeit			0,73303	663
Makro-Durchschnitt	0,66334	0,62431	0,63847	663
gewichteter Durchschnitt	0,72668	0,73303	0,72740	663

Tabelle E.35: Zehnfache Kreuzvalidierung: LSTM: Klassifizierungsbericht mit den domänenadaptierten Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,82972	0,85623	0,84277	313
Aggregation	0,65306	0,48485	0,55652	66
Ereignis	0,76250	0,88835	0,82063	206
Zustand	0,72549	0,47436	0,57364	78
Genauigkeit			0,78431	663
Makro-Durchschnitt	0,74269	0,67595	0,69839	663
gewichteter Durchschnitt	0,77899	0,78431	0,77573	663

Tabelle E.36: Projektspezifische Kreuzvalidierung: Zufallswald: Klassifizierungsbericht mit den domänenadaptierten Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,68106	0,90735	0,77808	313
Aggregation	0,75000	0,04545	0,08571	66
Ereignis	0,61570	0,72330	0,66518	206
Zustand	0,0000	0,0000	0,0000	78
Genauigkeit			0,65762	663
Makro-Durchschnitt	0,51169	0,41903	0,38224	663
gewichteter Durchschnitt	0,58749	0,65762	0,58254	663

Tabelle E.37: Projektspezifische Kreuzvalidierung: Stützvektormaschine: Klassifizierungsbericht mit den domänenadaptierten Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,79479	0,77955	0,78710	313
Aggregation	0,53448	0,46970	0,50000	66
Ereignis	0,71889	0,75728	0,73759	206
Zustand	0,43210	0,44872	0,44025	78
Genauigkeit			0,70287	663
Makro-Durchschnitt	0,62007	0,61381	0,61623	663
gewichteter Durchschnitt	0,70263	0,70287	0,70233	663

Tabelle E.38: Projektspezifische Kreuzvalidierung: Logistische Regression: Klassifizierungsbericht mit den domänenadaptierten Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,84028	0,77316	0,80532	313
Aggregation	0,56364	0,46970	0,51240	66
Ereignis	0,72458	0,83010	0,77376	206
Zustand	0,41667	0,44872	0,43210	78
Genauigkeit			0,72247	663
Makro-Durchschnitt	0,63629	0,63042	0,63089	663
gewichteter Durchschnitt	0,72695	0,72247	0,72245	663

Tabelle E.39: Projektspezifische Kreuzvalidierung: LSTM: Klassifizierungsbericht mit den domänenadaptierten Worteinbettungen

Klasse	Präzision	Ausbeute	F_1-Maß	Anzahl
Aktion	0,84112	0,86262	0,85174	313
Aggregation	0,90909	0,27027	0,41667	74
Ereignis	0,68235	0,86139	0,76149	202
Zustand	0,49231	0,43243	0,46043	74
Genauigkeit			0,74811	663
Makro-Durchschnitt	0,73122	0,60668	0,62258	663
gewichteter Durchschnitt	0,76140	0,74811	0,73200	663