Karlsruhe Institute of Technology

Bestimmung der semantischen Funktion von Quelltextabschnitten

Bachelorarbeit
von

Timo Januschke

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

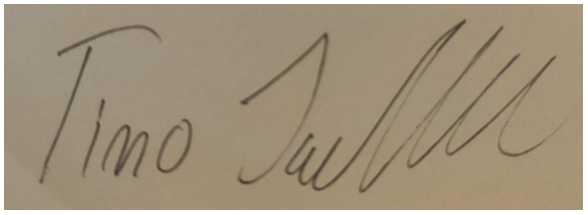
Erstgutachter:	Prof. Dr. Walter F. Tichy
Zweitgutachter:	Prof. Dr. Ralf H. Reussner
Betreuender Mitarbeiter:	M.Sc. Tobias Hey
Zweiter betr. Mitarbeiter:	

Bearbeitungszeit: 26.5.2020 – 25.9.2020

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, 25.9.2020



(**Timo Januschke**)

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.

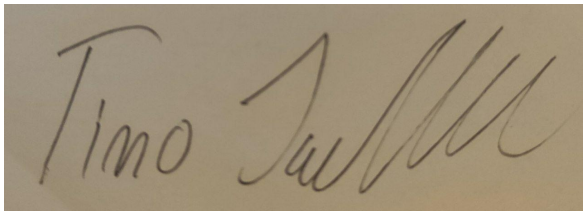
Bei etwaigen Forderungen Dritter stelle ich das KIT frei.

Eventuelle Mitautoren sind mit diesen Regelungen einverstanden.

Der Betreuer der Arbeit ist mit der Veröffentlichung einverstanden.

Art der Abschlussarbeit:	Bachelorarbeit
Titel:	Bestimmung der semantischen Funktion von Quelltextabschnitten
Datum:	25.9.2020
Name:	Timo Januschke

Karlsruhe, 25.9.2020



(**Timo Januschke**)

Kurzfassung

Rückverfolgbarkeitsinformationen zwischen Quelltext und Anforderungen ermöglichen es Werkzeugen Programmierer besser bei der Navigation und der Bearbeitung von Quelltext zu unterstützen. Um solche Verbindungen automatisiert herstellen zu können, muss die Semantik der Anforderungen und des Quelltextes verstanden werden. Im Rahmen dieser Arbeit wird ein Verfahren zur Beschreibung der geteilten Semantik von Gruppierungen von Programmelementen entwickelt. Das Verfahren basiert auf dem statistischen Themenmodell LDA und erzeugt eine Menge von Schlagwörtern als Beschreibung dieser Semantik. Es werden natürlichsprachliche Inhalte im Quelltext der Gruppierungen analysiert und genutzt um das Modell zu trainieren. Um Unsicherheiten in der Wahl der Paramter von LDA auszugleichen und die Robustheit der Schlagwortmenge zu verbessen, werden mehrere LDA-Modelle kombiniert. Das entwickelte Verfahren wurde im Rahmen einer Nutzerstudie evaluiert. Insgesamt wurde eine durchschnittliche Ausbeute von 0.73 und ein durchschnittlicher F1-Wert von 0.56 erreicht.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Große Softwareprojekte werden vor der Entwicklung meist formal durch Anforderungen beschrieben. Diese Anforderungen werden von den Entwicklern implementiert und die Dokumentation des Quelltextes geschieht durch Quelltextkommentare. Somit sind sowohl die Anforderungs- als auch Quelltextseite dokumentiert, allerdings besteht keine direkte Möglichkeit von den Anforderungen auf den implementierenden Quelltext oder von einem Quelltextfragment auf die beschreibende Anforderung zu schließen. Diese Verbindungen werden als Rückverfolgbarkeitsinformationen bezeichnet und sind hilfreich für Quelltextnavigation oder Fehlerlokalisierung. Solche Informationen sind allerdings in den meisten Fällen nicht vorhanden, da ihre manuelle Erstellung sowie Pflege einen erheblichen Aufwand mit sich bringt. Im Rahmen des Forschungsprojekts INDIRECT wurde eine semantische Repräsentation von Quelltext entwickelt, welche als Ausgangspunkt dient um solche Informationen aus dem Quelltext eines Softwareprojekts zu gewinnen. Diese Repräsentation gruppiert Quelltextabschnitte anhand ihrer semantischen Ähnlichkeit. Um verstehen zu können welche Absicht diese Gruppierungen erfüllen, wird im Rahmen dieser Arbeit ein Verfahren zur Beschreibung der geteilten Semantik von Quelltextabschnitten entwickelt.

2 Grundlagen

Themenmodelle sind statistische oder maschinelle Lernmodelle, welche Dokumenten Themen zuordnen. In diesem Kapitel werden die Grundlagen, welche zum Verständnis dieser Arbeit notwendig sind, eingeführt. Zuerst werden einige Begriffe formal definiert. Danach werden verschiedene Verfahren zur Themenmodellierung kurz vorgestellt.

2.1 Natürliche Sprachverarbeitung

Die natürliche Sprachverarbeitung (engl. *natural language processing*, kurz **NLP**) ist ein Bereich der Informatik, der sich mit der Verarbeitung von geschriebenen oder gesprochenen natürlichsprachlichen Inhalten befasst.

2.1.1 Tokenisierung

Die Tokenisierung (engl. *tokenization*) ist ein Prozess, durch den ein Text in eine Sequenz von Token zerlegt wird [CEE+10]. Ein Token ist die kleinste betrachtete Einheit eines Textes und wird meist als eine Folge von alphanumerischen Zeichen, welche von Leerzeichen umgeben ist, definiert. Der Begriff Token wird benötigt, da weder „Wort“ (ein Token kann auch ein einzelner Buchstabe sein), noch „Term“ (ein Term kann aus mehreren Token bestehen) dieselbe Bedeutung besitzen. Definition 2.1 führt den Begriff eines Token formal ein. Besonders bei der Tokenisierung von Quelltext kann es aber auch vorkommen, dass mehrere Wörter, welche als einzelne Token betrachtet werden sollen, zu einem Bezeichner kombiniert wurden. Die Aufspaltung eines Wortes in mehrere Token nennt man Subwort-Tokenisierung (engl. *subword tokenization*). Beispiel 2.1 zeigt die Subwort-Tokenisierung am Beispiel eines Java-Bezeichners.

Definition 2.1: Token

Ein Token t wird als die kleinste betrachtete Einheit eines Textes definiert. Dabei kann es sich um ein Wort $t =$ **contract** oder einen Buchstaben $t =$ **c** handeln. Außerdem ist es egal um welche Art von Text es sich handelt: Token können aus natürlichsprachlichen Dokumenten, sowie Quelltext gewonnen werden.

Beispiel 2.1: Subworttokenisierung

Gegeben sei der Java Methodenbezeichner `createAndStoreContract`. Er wird durch ein einzelnes Token repräsentiert, obwohl er vier Wörter enthält. Durch Subwort-Tokenisierung an den Großschreibungsgrenzen des Token erhält man die Subtoken: {`create`, `and`, `store`, `contract`}

2.1.2 Syntax und Semantik

Syntax und Semantik sind zwei Bereiche der Linguistik und Computerlinguistik [CEE+10]. Der Bereich der **Syntax** befasst sich mit der Struktur von Sätzen und im Fall von Programmiersprachen mit der Struktur von Quelltext. Konkret kann zwischen zwei Arten von Syntaxdarstellungen unterschieden werden: Die Dependenz- und Determinationssyntax betrachtet Relationen zwischen Wörtern als syntaktische Struktur. Die Konstituentensyntax ergänzt diese Darstellung um Konstituenten, komplexere Satzeinheiten, welche ebenfalls Relationen zu Wörtern oder anderen Konstituenten besitzen können. Syntaktische Strukturen können in beiden Fällen durch Bäume dargestellt werden. Im Bezug auf Programmiersprachen bezeichnet Syntax die Regeln, welche die Struktur eines validen Programms definieren. Der hierbei entstehende Baum wird abstrakter Syntaxbaum (engl. *abstract syntax tree*, kurz **AST**) genannt.

Der Bereich der **Semantik** befasst sich anders als die Syntax mit der Bedeutung und den Aussagen von Sätzen [CEE+10]. Die syntaktische Korrektheit trägt zur Semantik eines Satzes bei, ist allerdings nicht notwendig um einen semantisch sinnvollen Satz zu bilden. Die Semantik eines Satzes oder eines Dokuments ergibt sich meist nur durch eine Betrachtung als Gesamtheit. Semantische Bausteine können deshalb auch komplexer sein, als nur einzelne Wörter. Die Semantik von Quelltext beschreibt das Konzept, welches ein Quelltextabschnitt implementiert, also die Funktionalität, welche durch seine Ausführung realisiert wird. Eine korrekte Syntax ist oft ausschlaggebend für das Verständnis der Semantik von Quelltext, da Programmiersprachen strengeren Regeln unterliegen als natürliche Sprache.

2.1.3 Stammformreduktion und Lemmatisierung

Wörter können je nach Kontext in verschiedenen Formen auftreten. Um verschiedenen Wortformen dieselbe Bedeutung zuordnen zu können, müssen diese auf ihren Wortstamm zurückgeführt werden [JM09]. Die **Stammformreduktion** (engl. *stemming*) nach Porter nutzt eine Menge von Regeln, um iterativ Suffixe von Wörtern zu entfernen [Wil06]. Aufgrund dieser Regeln kann es vorkommen, dass zwei unverwandte Wörter fälschlicherweise auf denselben Wortstamm reduziert werden. Um solche Fehler zu vermeiden, nutzt die **Lemmatisierung** Informationen über die Wortart, die Bedeutung und den Kontext des zu normalisierenden Wortes, sowie ein Wörterbuch um ein Wort auf seine Grundform (*Lemma*) zu reduzieren [KLJJ04].

2.1.4 Stoppwortentfernung

Stoppwörter sind meist kurze, hochfrequente Wörter, welche kaum einen Beitrag zur Semantik eines Textes erbringen [JM09]. Sie selbst tragen keine Informationen über das Thema des Textes, in dem sie auftreten und können somit im Rahmen der Vorverarbeitung eines Textes entfernt werden. Die Stoppwortentfernung erlaubt es den weiteren Verarbeitungsschritten sich auf die Wörter zu konzentrieren, welche relevant für die Semantik des Textes sind. Beispiele für häufig gewählte Stoppworte sind Artikel oder Präpositionen. Alle Stoppworte werden in einer Liste festgehalten und die entsprechenden Token werden durch

einen Abgleich mit der Liste entfernt. Beispiel 2.2 zeigt die Stoppwortentfernung anhand einer Menge von Token.

Beispiel 2.2: Stoppwortentfernung

Gegeben sei die Token-Menge {`create`, `and`, `store`, `contract`} und angenommen `and` sei ein Stoppwort.
Nach der Stoppwortentfernung bleibt die Token-Menge: {`create`, `store`, `contract`}

2.2 Wort- und Dokumentrepräsentationen

Sowohl für die Verarbeitung durch statistische Themenmodelle, als auch für die Verarbeitung durch maschinelle Lernmodelle müssen Wörter und Dokumente in einer Zahlenrepräsentation vorliegen. Die konkret gewählte Repräsentation kann sich von Modell zu Modell unterscheiden. Allerdings kann sich die Wahl einer geeigneten Repräsentation auch auf die Effizienz des Modells auswirken. In diesem Abschnitt wird deshalb ein kurzer Überblick über verschiedene Formen von Wort- und Dokumentrepräsentationen gegeben.

2.2.1 Einführung grundlegender Begriffe

Themenanalyseverfahren beschäftigen sich mit der Findung und Zuordnung von Themen zu Dokumenten. Diese Begriffe müssen jedoch erst konkretisiert werden bevor genauer auf die Themenmodellierung eingegangen werden kann. In dieser Arbeit werden Wörter, Themen und Dokumente, angelehnt an Blei *et al.* [BNJ03] wie in Definition 2.2 eingeführt. Der Begriff „Dokument“ mag zwar den Eindruck erwecken, dass es sich um natürlichsprachliche Dokumente handeln muss, allerdings kann auch beispielsweise eine Quelltextdatei als Dokument dienen. Die Darstellung von Dokumenten durch Multimengen wird auch Wortbeuteldarstellung genannt. Die Token eines Dokuments in Wortbeutelrepräsentation unterliegen keiner festen Reihenfolge. Somit können Verfahren, welche Wortbeutel als Eingaberepräsentation nutzen, die Reihenfolge von Wörtern in Dokumenten nicht zur Informationsgewinnung nutzen. Die Repräsentation von Token durch Basisvektoren hat allerdings ebenfalls einen Nachteil: Durch sie werden Terme sowie Dokumente durch sehr dünn besetzte Vektoren von hoher Dimensionalität repräsentiert [CZYZ12]. Dieses Phänomen wird auch als „Fluch der Dimensionalität“ bezeichnet. Dünn besetzte Vektoren und Matrizen benötigen speziell auf sie zugeschnittene Algorithmen und Speicherformate, da sie sonst einen hohen Speicherbedarf und eine hohe rechnerische Komplexität mit sich bringen.

Definition 2.2: Dokumente, Vokabular, Token

Ein Dokument $D = \{t_1, \ldots, t_n\}$ ist eine Multimenge von Token t_i oder Wörtern w_i. Im Kontext von natürlichsprachlichen Dokumenten wird meist der Begriff „Wort“ verwendet aber handelt es sich nicht um natürlichsprachliche Wörter ist „Token“ eindeutiger. Ein Korpus $K = \{D_1, \ldots, D_m\}$ ist eine Menge von Dokumenten. Ein Vokabular $V = \{t_1, \ldots, t_k\}$ ist die Menge, die alle einzigartigen Token oder Wörter der Dokumente im Korpus K beinhaltet. Ein Token kann dabei sowohl durch seinen textuellen Inhalt $t =$ **someToken**, als auch durch einen Einheitsvektor $t = (0, \ldots, 1, \ldots, 0)^T$ beschrieben werden. In der Darstellung als Einheitsvektor entspricht der Index mit Wert 1 gleich dem Index i des Token t im Vokabular V und ein Dokument kann als Summe all seiner Einheitsvektoren dargestellt werden.

Beispiel 2.3: Darstellung durch Einheitsvektoren

Es sei $\mathbf{V} = \{\texttt{quelltext}, \texttt{anforderungen}, \texttt{sprache}\}$ ein Vokabular. Das Wort `anforderungen` entspricht Index 2 in $\mathbf{V}$ und wird durch den Vektor $\mathbf{w} = (0, 1, 0)^T$ repräsentiert.

2.2.2 Term-Dokument Matrix

Eine Möglichkeit um Token und Dokumente miteinander zu Verbinden ist die Term-Dokument Matrix. Diese $M \times N$ Matrix setzt die $M = |K|$ Dokumente eines Korpus K mit den $N = |V|$ Token des ihnen zugrunde liegenden Vokabulars V in Verbindung. Ihre Einträge geben an mit welcher Frequenz ein Term in einem Dokument auftritt. Die genauen Werte der Einträge hängen dabei von dem gewählten Gewichtungsschema der Matrix ab. Ein beliebtes Schema für Term-Dokument Matrizen ist *tf-idf* (engl. *term frequency-inverse document frequency*). Hierbei wird die Anzahl der Vorkommnisse eines Terms innerhalb eines Dokuments invers mit der Anzahl der Dokumente, die diesen Term beinhalten, gewichtet. Dieses Schema erlaubt die Gewichtung der Wichtigkeit eines Terms für ein Dokument mit der Relevanz des Terms im Vergleich zu allen anderen Dokumenten des Korpus. Diese Matrix bildet den Ausgangspunkt für viele statistische, sowie durch maschinelles lernen unterstützte Themenanalyseverfahren.

2.2.3 n-Gramme

n-Gramme sind eine weitere Möglichkeit, neben dem Wortbeutelmodell, um die Token in einem Dokument zu repräsentieren. n-Gramme können sowohl auf Wort-, als auch auf Zeichenebene gebildet werden. Die Komponenten der n-Gramme erhalten die Reihenfolge der Token im Dokument. Somit kodieren sie eine Nachbarschaftsinformation der Token, beispielweise welche Token häufig nahe beieinander vorkommen. 1-Gramme werden auch Unigramme genannt, 2-Gramme werden Bigramme genannt und 3-Gramme werden Trigramme genannt. Beispiel 2.4 zeigt die Unigramme und Bigramme eines Beispielsatzes.

Beispiel 2.4: n-Gramme

Gegeben sei der Satz „Er ging zur Uni“.
Seine 1-Gramme sind: $\{(\texttt{Er}), (\texttt{ging}), (\texttt{zur}), (\texttt{Uni})\}$
Seine 2-Gramme sind: $\{(\texttt{Er}, \texttt{ging}), (\texttt{ging}, \texttt{zur}), (\texttt{zur}, \texttt{Uni})\}$

2.2.4 Skip-Gramme

Skip-Gramme sind eine Generalisierung der n-Gramme. Anders als n-Gramme erlauben sie, dass die Token nicht direkt aufeinander folgend sein müssen [GAL+06]. Ein k-Skip-n-Gramm erlaubt insgesamt bis zu k Tokenüberspringungen um ein n-Gramm zu erzeugen. Beispiel 2.5 zeigt die Bildung von Skip-Grammen anhand eines Beispielsatzes.

Beispiel 2.5: Skip-Gramme

Gegeben sei der Satz „Sie ist seine Mutter“.
Ein 2-Skip-2-Gram dieses Satzes ist $(\texttt{Sie}, \texttt{Mutter})$.
Ohne das Überspringen von Token hätte dieses 2-Gramm, welches das Token „Sie“ mit „Mutter“ in Verbindung bringt, nicht gebildet werden können.

2.3 Unterscheidung von Themenmodellierung und Themenbeschriftung

Obwohl wir bisher nur von Themenmodellen gesprochen haben, gibt es eine wichtige Unterscheidung zwischen zwei eng verwandten Aufgabenstellungen in der natürlichen Sprachverarbeitung zu beachten. Die Themenmodellierung beschäftigt sich damit, innerhalb eines Korpus Mengen von Token zu finden, welche zusammen ein Thema charakterisieren [WKHT19]. Das Modell weiß zwar, dass eine Menge von Token miteinander verwandt ist, allerdings kann es keine Aussage darüber treffen, wodurch sich diese Verwandtschaft konkret auszeichnet und wie sie beschrieben werden kann.

Die Themenbeschriftung folgt in der Regel der Themenmodellierung und soll den identifizierten Mengen eine natürlichsprachliche Repräsentation zuordnen. Eine solche Repräsentation kann zum Beispiel ein unsortierte Menge von Termen oder ein natürlichsprachlicher Fließtextsatz sein.

Häufig können Themenmodellierungsverfahren auch als Themenbeschriftungsverfahren genutzt werden: LDA von Blei *et al.* ist ein generatives Themenmodell, es ordnet Dokumenten eine Wahrscheinlichkeitsverteilung von Themen zu. Diese Themen sind wiederum über Token charakterisiert, die häufig im Kontext des Themas auftreten. Betrachtet man das dominante Thema eines Dokuments und die dominanten Token des Themas, so kann man diese als Beschriftung für das Thema ansehen.

Im Rahmen dieser Arbeit wird der Begriff „Themenmodell“ als eine solche Zusammenfassung der beiden Teilaufgaben genutzt.

2.4 Statistische Themenanalyse für Fließtextkorpora

Das Feld der statistischen Themenanalyse ist von großer Bedeutung für weiterführende Arbeiten in den Bereichen der durch maschinelles lernen unterstützten Themenanalyse und Quelltextzusammenfassung. Verfahren aus diesem Feld nutzen statistischen Modelle um Beziehungen zwischen Token und Dokumenten in Korpora herzustellen.

2.4.1 Latente semantische Indizierung

Die latente semantische Indizierung (engl. *latent semantic indexing*) [DDF$^+$90] ist ein Themenmodellierungsverfahren von Deerwester *et al.*

Das Verfahren benötigt als Ausgangspunkt eine $M \times N$ Matrix, deren Reihen jeweils einem der $M = |K|$ Dokumente und deren Spalten jeweils einem der $N = |V|$ Token des Vokabulars entsprechen. Die Singulärwertzerlegung erlaubt es eine beliebige $m \times n$ Matrix X in ein Produkt aus drei Matrizen:

$$X = USV^T$$

zu zerlegen. Die hierbei entstandenen Matrizen haben die folgenden Eigenschaften: U ($m \times m$) und V^T ($n \times n$) sind orthogonal (also $UU^T = I = V^TV$) und S ($m \times n$) ist eine Diagonalmatrix. Die Elemente auf der Hauptdiagonalen von S sind die *Singulärwerte* der Matrix X und sie sind ihrem Betrag nach absteigend sortiert.

Nun legt man eine Anzahl von k Themen fest und reduziert S auf eine $k \times k$ Matrix S', welche nur die k höchstwertigen Diagonaleinträge von S enthält. Die Dimensionen von U und V^T werden entsprechend angepasst sie werden von nun an mit T und D bezeichnet:

$$X' = TS'D$$

Multipliziert man S' mit D erhält man eine $k \times n$ Matrix, deren Spalten die Stärke der Zugehörigkeit eines Dokuments zu den k Themen kodieren. Analog erhält man durch Multiplikation von T mit S' eine $m \times k$ Matrix, deren Reihen kodieren, wie relevant ein Token für eines der k Themen ist. Betrachtet man die Reihen- und Spaltenvektoren der entstandenen Matrizen als Vektoren des k-Dimensionalen Raums erhält man eine Dimensionalitätsreduzierung auf k Dimensionen.

Durch ähnliche Rechenschritte ist es außerdem möglich Token mit Token, Token mit Dokumenten und Dokumente mit Dokumenten zu vergleichen. Für die Themenmodellierung ist hierbei interessant, dass die Reihen von TS' genutzt werden können, um den abstrakten Themen Terme zuzuordnen und sie somit zu beschreiben.

2.4.2 Latente Dirichlet Allokation

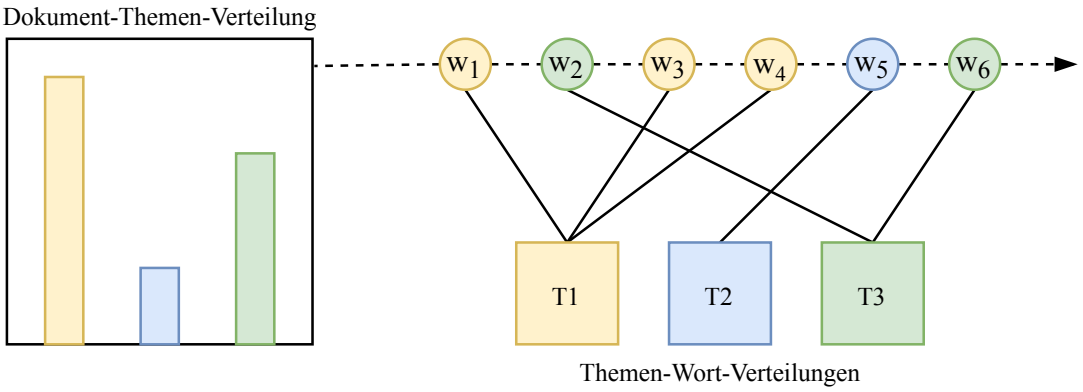


Abbildung 2.1: Schematische Darstellung der generativen Dokumenterzeugung.

Die latente Dirichlet Allokation (engl. *latent dirichlet allocation*) ist ein Verfahren von Blei *et al.*, welches ein dreistufiges bayessches Modell benutzt um einem Dokument mehrere Themen zuordnen zu können [BNJ03]. Der Name LDA leitet sich von der genutzten Dirichletverteilung ab, welche in Definition 2.3 kurz eingeführt wird. LDA basiert auf der Annahme, dass Dokumente als zufällige Mischungen von Themen und Themen als Wahrscheinlichkeitsverteilung über die Token des zugrundeliegenden Vokabulars modelliert werden können. Der generative Prozess zur Erzeugung von Dokumenten wird im Folgenden kurz eingeführt und ist in Abbildung 2.1 dargestellt.

LDA basiert auf drei Komponenten. Dokumente werden als Menge von Wörtern angesehen, wobei jedem Dokument eine Wahrscheinlichkeitsverteilung von Themen zugeordnet wird. Themen sind wiederum Wahrscheinlichkeitsverteilungen über die Wörter, welche das Thema charakterisieren. Definition 2.4 führt diesen Begriff eines Themas im Kontext von LDA formal ein. Wörter sind die kleinste Einheit in einem Dokument und kann als äquivalent zu dem in dieser Arbeit eingeführten Begriff eines Token angesehen werden. LDA nutzt aus, dass Dokumente, welche ähnliche Themen behandeln, häufig auch ähnliche Wörter beinhalten. LDA nimmt den folgenden Erzeugungsprozess von Dokumenten an:

Definition 2.3: Dirichletverteilung

Die Dirichletverteilung ist eine merdimensionale Verallgemeinerung der Beta-Verteilung. Während multinomiale Verteilungen die Wahrscheinlichkeit für das auftreten von K aufeinanderfolgenden Ereignissen modellieren, liefern Dirichletverteilungen die Wahrscheinlichkeit, dass eine solche Multinomialverteilung auftritt.

1. Wähle eine Anzahl an Wörtern N aus einer Poissionverteilung
2. Wähle eine Verteilung θ aus einer Dirichletverteilung
3. Für jedes der N Wörter:
   a) Wähle ein Thema aus einer Multinomialverteilung über θ
   b) Wähle ein Belegung für das Wort aus der Wortverteilung des Themas

Definition 2.4: LDA Thema

Ein LDA-Thema $T = \{(t_1, p_1), \ldots, (t_n, p_n)\}$ ist eine Menge von Paaren bestehend aus Token und dem Anteil des Token an dem Thema. Da sich diese Anteile zu 1 aufsummieren handelt es sich bei einem LDA-Thema um eine Wahrscheinlichkeitsverteilung und sie können wie solche behandelt werden. Formal gilt $n = |V|$ da die Themen Wahrscheinlichkeitsverteilungen über das gesamte genutzte Vokabular des LDA-Modells sind. Da jedoch die meisten dieser Wahrscheinlichkeiten sehr nahe bei 0 liegen, geben viele LDA-Implementierungen nur Wörter über einem bestimmten Schwellwert aus.

LDA nutzt mehrere Parameter um das Verhalten des Modells zu beeinflussen. Der α Parameter steuert die Dokument-Themen-Verteilung. Ein kleiner Wert für α bedeutet, dass einem Dokument wenige bis ein Thema zugeordnet werden und ein hohes α bedeutet, dass die Dokumente eher aus einer Mischung vieler Themen bestehen. β verhält sich ähnlich, jedoch hat dieser Parameter Einfluss auf die Verteilung der Wörter pro Thema. Ein kleines β bedeutet, dass Themen eher aus wenigen verschiedenen Wörtern bestehen. Diese Parameter werden im Allgemeinen nur ein mal pro Dokumentkorpus festgelegt. Zu beachten ist, dass die Bestimmung von Optimalwerten für diese Parameter ein offenes Problem ist. Es gibt keine allgemeinen Optimalwerte, welche für jeden Datensatz genutzt werden können. Somit müssen die Parameter approximiert werden.

Ein weiterer wichtiger Parameter des LDA Modells ist k. Er gibt an, wie viele Themen insgesamt von LDA erzeugt werden und muss fest und im Voraus bekannt sein. Der Optimalwert für k ist stark vom Anwendungsfall und den zugrundeliegenden Daten abhängig. Liegt ein Datensatz von Dokumenten vor, welcher eine bekannte Anzahl an Themen beinhaltet, so kann man den Parameter direkt auf die erwartete Anzahl an Themen setzen. Sollte man an einer unbekannten Anzahl an Themen interessiert sein, so muss dieser Parameter ebenfalls approximiert werden. Weiß man beispielsweise, dass jedes Dokument jeweils ein paarweise disjunktes Thema beinhaltet, so kann man die Anzahl an Dokumenten als Ausgangspunkt für k nutzen.

LDA ist ein wichtiges Ausgangsverfahren, auch über die Themenmodellierung hinaus [WG08].

2.4.3 Hierarchischer Dirichlet Prozess

Der hierarchische Dirichlet Prozess (engl. *hierarchical dirichlet process*, kurz **HDP**) ist ein von Teh *et al.* [TJBB05] entwickeltes hierarchisches Themenmodell. Anders als LDA, welches eine im Voraus festgelegte Anzahl an Themen benötigt, kann HDP die Anzahl an Themen aus den Daten eines Korpus inferieren und selbst festlegen. Außerdem können Token zwischen Themen geteilt werden, wodurch Abhängigkeiten zwischen Themen modelliert werden können. HDP kann in Kombination mit LDA eingesetzt werden, um den Parameter k des LDA Modells zu approximieren.

2.5 Wissensbasierte Themenmodelle

Wissensbasierte Themenmodelle nutzen externe Wissensquellen, um Dokumenten Themen zuzuordnen.

Ontologien sind Sammlungen von Konzepten, Entitäten und Beziehungen zwischen diesen. Sie können genutzt werden um Themen durch Konzepte anstatt nur durch Wörter zu charakterisieren [AK15]. Dokumenten wird anhand der in ihnen enthaltenen Entitäten und Beziehungen aus der Ontologie ein Thema zugeordnet. Durch die Nutzung von Ontologieinformationen kann semantische Ähnlichkeit auf Konzeptebene berechnet werden.

Gabrilovich und Markovich stellten bereits 2007 ein Verfahren vor, welches Wikipedia als externe Wissensquelle nutzt [GM07]. Wikipedia ist eine der größten menschengemachten Wissensdatenbanken der Welt und enthält als solche Definitionen von hunderttausenden Konzepten. Um diese expliziten Hintergrundinformationen zu nutzen, verwenden auch Gabrilovich und Markovich Wortvektoren in einen hochdimensionalen semantischen Vektorraum. Mithilfe einer invertierten Indexstruktur ordnen sie Wörtern aus dem Vokabular eine gewichtete Liste von Wikipedia-Artikeln zu, durch die sie charakterisiert werden. Dokumenten kanna nun anhand ihrer enthaltenen Wörter ein gewichteter Vektor von Wikipedia-Konzepten zugeordnet werden.

Das Internet-Wörterbuch Wiktionary[1] kann als Quelle von lexikalischen Informationen genutzt werden [ZMG08]. Es enthält Millionen von kollaborativ beigetragenen Wortdefinitionen in über einhundert Sprachen. Für die natürliche Sprachverarbeitung interessant sind unter anderem Angaben über Synonyme oder Sinnverwandte Wörter, Übersetzungen oder Beispielsätze. Zesch *et al.* nutzen wie auch Grabrilovich und Markovich einen konzeptvektorbasierten Ansatz um Wörter durch gewichtete Vektoren von Wiktionary-Konzepten darzustellen.

2.6 Hellingerabstand

Der Hellingerabstand ist eine Ähnlichkeitsmetrik für Wahrscheinlichkeitsverteilungen. Für zwei diskrete Wahrscheinlichkeitsverteilungen P und Q wird der Hellingerabstand gemäß Formel 2.1 definiert.

$$H(P,Q) = \frac{1}{\sqrt{2}}||\sqrt{P} - \sqrt{Q}||_2 \tag{2.1}$$

Der Wertebereich von H liegt zwischen 0 und 1. Ein Wert von 0 wird durch zwei exakt übereinstimmende Wahrscheinlichkeitsverteilungen erreicht, während ein Wert von 1 durch zwei exakt komplementäre Wahrscheinlichkeitsverteilungen erreicht wird. Je näher er bei 0 liegt, desto ähnlicher sind die Verteilungen. Beispiel 2.6 zeigt diesen Zusammenhang anhand einiger einfacher Wahrscheinlichkeitsverteilungen.

Beispiel 2.6: Hellingerabstand

Gegeben seien die Wahrscheinlichkeitsverteilungen $P = \{1.0, 0.0\}$, $Q = \{1.0, 0.0\}$ und $R_1 = \{0.0, 1.0\}$. Da P und Q übereinstimmen gilt $H(P,Q) = 0.0$, für P und R gilt hingegen $H(P,R) = 1.0$. Nun sei eine weitere Wahrscheinlichkeitsverteilung $S = \{0.5, 0.5\}$ gegeben. Für P, S und R gilt dann $H(P,S) \approx 0.54 \approx H(S,R)$.

[1] https://www.wiktionary.org

2.7 Agglomeratives Clustering

Agglomerative Clusteringverfahren bilden iterativ aus einer Grundmengen Gruppierungen (engl. *Cluster*) der in ihr enthaltenen Elemente [AR13]. Hierzu beginnt das Verfahren mit einer Gruppierung je Element. Die Gruppierungen werden basierend auf einer Distanzmetrik miteinander verglichen und zu größeren Gruppierungen verschmolzen. Jede symmetrische Distanzmetrik kann mit agglomerativen Clusteringverfahren genutzt werden. Wie genau die Gruppierungen miteinander verschmolzen werden ist von der gewählten Verschmelzungsstrategie abhängig. Einige beliebte Strategien werden im Folgenden kurz vorgestellt:

1. Die **average-linkage** Strategie berechnet den Durchschnittswert der Distanzen zwischen den Elementen zweier Gruppierungen.
2. Die **complete-linkage** Strategie berechnet den Maximalwert der Distanzen zwischen den Elementen zweier Gruppierungen.
3. Die **single-linkage** Strategie berechnet den Minimalwert der Distanzen zwischen den Elementen zweier Gruppierungen.

Das Verfahren minimiert in jedem Schritt diesen durch die genutzte Strategie berechneten Wert und verschmilzt die zwei Gruppierungen die ihn erreicht haben. Dem Verfahren kann außerdem ein Schwellwert übergeben werden, ab dem zwei Gruppierungen nicht mehr verschmolzen werden.

2.8 Maschinelles Lernen

Maschinelles Lernen (engl. *machine learning*) ist ein Teilbereich der Informatik. Er beschäftigt sich mit Algorithmen, welche in der Lage sind anhand von Merkmalen (engl. *features*) Muster in großen Datenbeständen zu erkennen, anhand welcher Entscheidungen für unbekannte Daten getroffen werden können [FAP18].

2.8.1 Überwachtes und unüberwachtes Lernen

Lernverfahren können allgemein in zwei Kategorien eingeteilt werden: **Überwachte Lernverfahren** erhalten als Eingabe Paare von Datenpunkten und Beschriftungen (engl. *labels*). Anhand dieser beschrifteten Datenpunkte kann das Verfahren lernen, welche Beschriftung einem unbekannten Datenpunkt zugeordnet werden soll.

Unüberwachte Lernverfahren erhalten lediglich Datenpunkte als Eingabe und lernen Zusammenhänge zwischen diesen. Beispielsweise können unüberwachte Verfahren genutzt werden, um Datenpunkte anhand ihrer Ähnlichkeit zueinander zu gruppieren. Auch LDA 2.4.2 kann als ein unüberwachtes Lernverfahren angesehen werden, da es keine beschrifteten Trainingsdaten benötigt.

2.8.2 Neuronale Netze

Neuronale Netze sind eine konkrete Möglichkeit maschinelle Lernverfahren zu implementieren. Wie ihr Name verrät, ist ihr Aufbau schematisch an das menschliche Gehirn angelehnt [Gur97]. Sie bestehen aus einer Vielzahl von untereinander verbundenen Knoten, sogenannten künstlichen Neuronen. Ein Neuron erhält eine Liste von Eingaben, welche jeweils mit einem Gewicht multipliziert und dann an eine Aktivierungsfunktion (engl. *activation function*) weitergereicht werden. Diese Funktion kombiniert die Eingabewerte und erzeugt eine Ausgabe. Ein Beispiel für eine solche Aktivierungsfunktion ist eine Schwellwertfunktion: Die Eingaben werden gewichtet und summiert, liegt das Ergebnis über einem bestimmten Schwellwert, so gibt das Neuron ein hohes Signal aus, ansonsten eine Null.

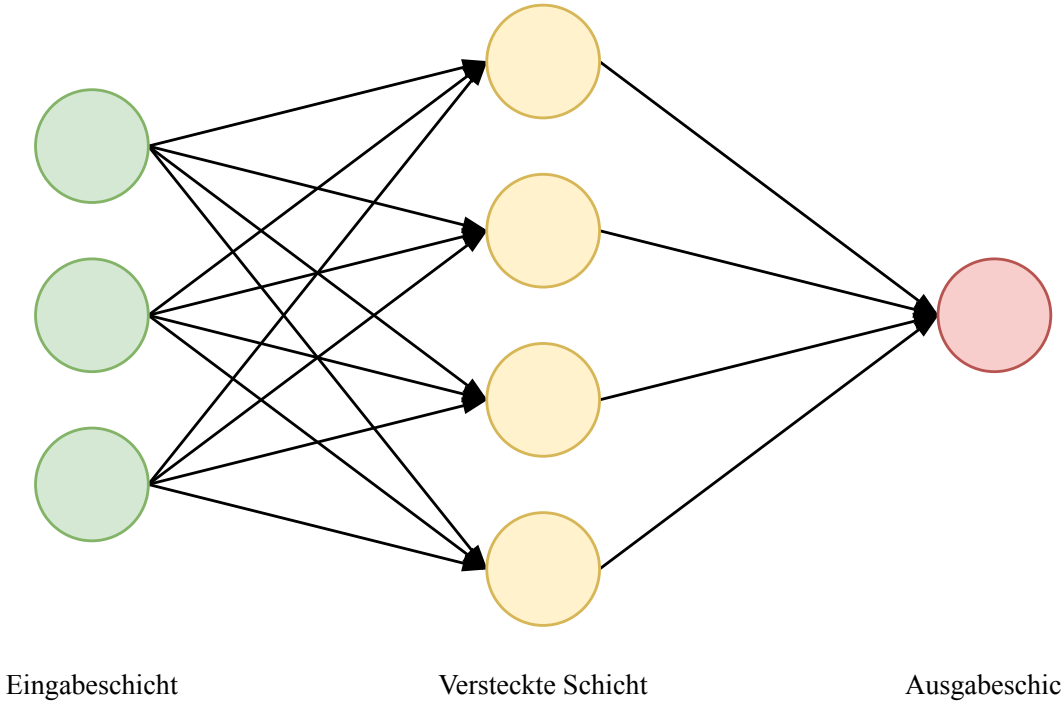


Abbildung 2.2: Schematische Darstellung eines neuronalen Netzes.

Werden Neuronen zu untereinander verbundenen Schichten kombiniert, spricht man von einem neuronalen Netz. Diese Netze haben meist eine Eingabe- und eine Ausgabeschicht, sowie eine oder mehrere verdeckte Schichten dazwischen. Abbildung 2.2 zeigt den schematischen Aufbau eines einfachen neuronalen Netzes.

Das neuronale Netz erhält als Eingabe einen Vektor x von Merkmalen. Nun werden Schicht für Schicht alle Aktivierungsfunktionen der Neuronen auf die Eingabewerte angewandt und an die nächste Schicht weitergereicht. Das neuronale Netz implementiert also eine gewisse Funktion $y = f(x)$, wobei f die Komposition der Schichten angewandt auf den Eingabevektor bezeichnet.

Um nun aus den Daten zu lernen wird für jeden Eingabevektor eine Fehlerfunktion berechnet. Sie gibt an, wie stark sich der vom neuronalen Netz vorhergesagte Wert vom idealen Wert unterscheidet. Mithilfe von Rückpropagierung (engl. *backpropagation*), einer wiederholten der Kettenregel für Ableitungen, können die partiellen Ableitungen dieser Fehlerfunktion im Bezug auf die Parameter des neuronalen Netzes berechnet werden. Diese partiellen Ableitungen können dann mit einem Algorithmus wie dem Gradientenabstieg verwendet werden, um die Parameter des Netzes zu optimieren.

2.8.3 Kodierer-Dekodierer Architekturen

Kodierer-Dekodierer Architekturen (engl. *encoder decoder architecture* oder *autoencoder*) beschreiben einen zweistufigen Aufbau von neuronalen Netzen. Der **Kodierer** erhält eine Eingabe x und erzeugt eine Ausgabe x', welche meist von niedrigerer Dimension ist, als die Eingabe. Der **Dekodierer** erhält als Eingabe die vom Kodierer erzeugte Repräsentation x' der ursprünglichen Eingabe und erzeugt eine Ausgabe y, sodass gilt:

$$x' = enc(x), y = dec(x') \Rightarrow y = dec(enc(x))$$

Diese Beschreibung von Kodierer-Dekodierer Architekturen ist bewusst sehr abstrakt, da sich die konkreten Ein- und Ausgabeformate, sowie die konkret genutzten Kodierer und

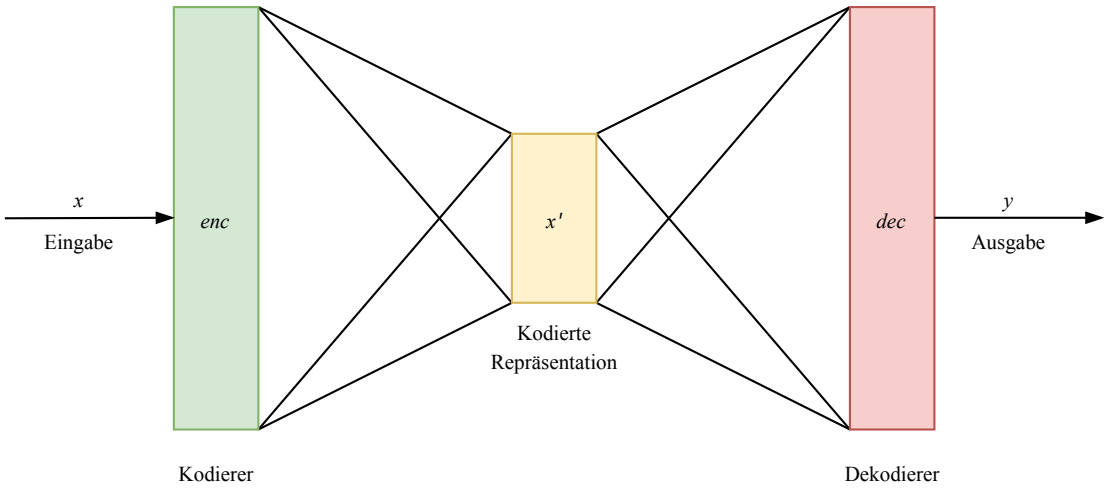


Abbildung 2.3: Schematische Darstellung einer Kodierer-Dekodierer Architektur.

Dekodierer je nach Anwendungsfall unterscheiden können. Abbildung 2.3 zeigt eine abstrakte Darstellung eines Kodierer-Dekodierer Netzwerks.

2.8.4 word2vec

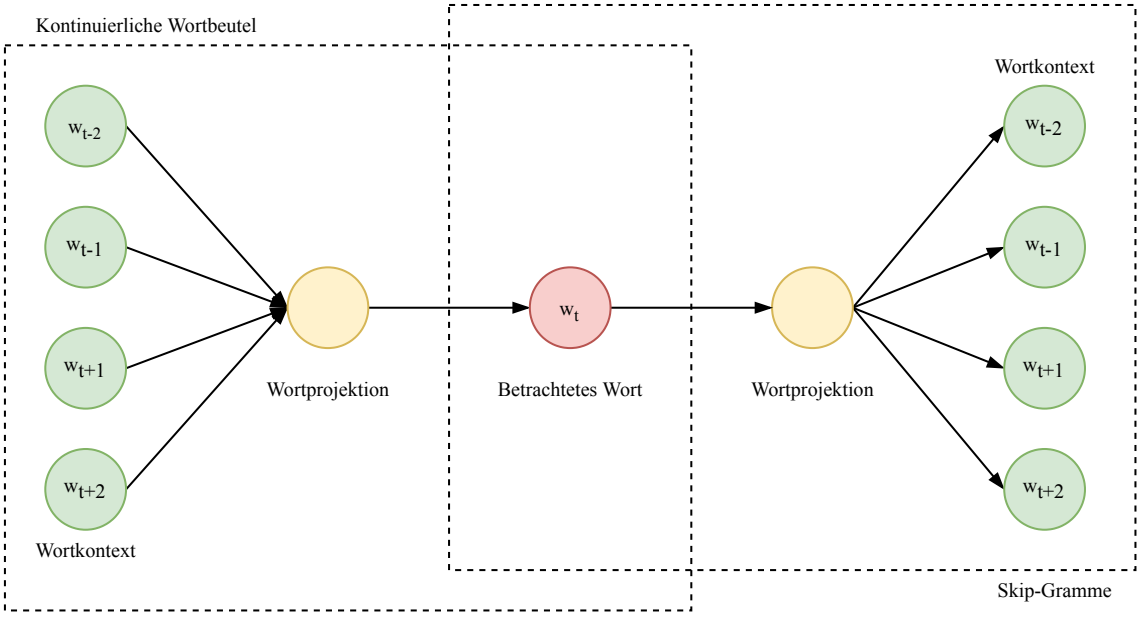


Abbildung 2.4: Schematische Darstellung des word2vec Modells.

Eine bekannte Implementierung eines Worteinbettungsmodells ist word2vec von Mikolov *et al.* [MCCD13] [MSC+13]. Das word2vec-Modell besteht aus zwei Teilmodellen: Zur Kodierung der Eingabesequenz wird ein kontinuierliches Wortbeutelmodell, eine Erweiterung normaler Wortbeutelmodelle, eingesetzt. Der Dekodierer-Teil besteht aus einem Skip-Gramm Modell.

Das kontinuierliche Wortbeutelmodell betrachtet einen Term und erhält als Kontext ein $2n$-Gramm der n Terme vor und nach diesem. Mit jedem Schritt wird dieses Kontextfenster weiter verschoben, sodass jedes Wort in der Eingabesequenz durch seine Nachbarschaft charakterisiert wird. Dieses Modell wird darauf trainiert Terme anhand ihres Kontextes vorherzusagen.

Das Skip-Gramm Modell bildet die Umkehrfunktion zum kontinuierlichen Wortbeutelmodell. Es wird darauf trainiert anhand eines Terms die n Worte vor und nach ihm vorherzusagen, also ein $2n$-Gramm zu erzeugen.

Die erste Stufe des word2vec Modells kann eingesetzt werden um einem Term und seinen umliegenden Kontexttermen einen reellwertigen Vektor aus einem gelernten semantischen Vektorraum zuzuordnen. Diese Vektoren können genutzt werden, um arithmetische Operationen durchzuführen, welche bei einem gut trainierten Modell intuitiv erscheinen. Am Beispiel $vec(Paris) - vec(Frankreich) + vec(Deutschland) \approx vec(Berlin)$ wird verdeutlicht, dass semantisch ähnliche Terme im Vektorraum nahe beieinander liegen und einfache mathematische Operationen genutzt werden können um Muster zwischen ihnen darzustellen.

2.8.5 Rekurrente neuronale Netze

Klassische neuronale Netze setzen voraus, dass die Dimensionalitäten ihrer Ein- und Ausgaben bekannt und fest sind. Im Bereich der maschinellen Übersetzung sind die Eingaben und Ausgaben meist natürlichsprachliche Sätze. Die Länge der Eingabesätze ist im Allgemeinen nicht bekannt und vor allem nicht fest und die Länge des Ausgabesatzes kann sich je nach Eingabe ebenfalls unterscheiden.

Rekurrente neuronale Netze (engl. *recurrent neural networks*, kurz **RNN**) sind eine Verallgemeinerung von klassischen neuronalen Netzen, welche mit Sequenzen $x = (x_1, \ldots, x_n)$ als Eingabe umgehen können [SVL14]. Diese Sequenzen können beispielsweise als Zeitschritte, Momentaufnahmen von kontinuierlichen Daten oder wie im Fall der maschinellen Übersetzung als die Wörter eines Satzes interpretiert werden. Da klassische RNNs allerdings ebenfalls voraussetzen, dass die Längen der Ein- und Ausgabesequenzen bekannt sind, sind sie für maschinelle Übersetzung nur bedingt geeignet.

Lange Kurzzeitgedächtnis Netzwerke (engl. *long short-term memory*, kurz **LSTM**) sind eine Art von RNN und schätzen eine bedingte Wahrscheinlichkeit:

$$p(y_1, \ldots, y_n | x_1, \ldots, x_m)$$

Entscheidend ist hierbei, dass sich die Längen der Eingabesequenz m und die Länge der Ausgabesequenz n unterscheiden können. Das von Sutskever *et al.* implementierte Modell [SVL14] nutzt zwei LSTMs, eines für die Eingabesequenz und eines für die Ausgabesequenz. Dies macht ihr Modell, sowie fast alle Sequenz-zu-Sequenz Modelle, zu einer Kodierer-Dekodierer-Architektur (siehe 2.8.3).

Eine weitere RNN Architektur sind die sogenannten beschränkten rekurrenten Einheiten (engl. *gated recurrent unit*, kurz **GRU**) [CGCB14]. GRUs erfüllen weitestgehend dieselben Aufgaben wie LSTMs, besitzen allerdings weniger trainierbare Parameter, da ihnen ein Ausgabegatter fehlt, welches bei LSTMs vorhanden ist. Dadurch sind sie im Allgemeinen leichter zu trainieren, ohne erheblich an Genauigkeit einzubüßen.

2.8.6 Transformer-Modelle

Transformer-Modelle sind sowohl Sequenz-zu-Sequenz Modelle, als auch Kodierer-Dekodierer Architekturen. Sie nutzen Aufmerksamkeitsmechanismen (engl. *attention mechanisms*) um weitläufige Abhängigkeiten innerhalb einer Sequenz zu berücksichtigen [VSP+17]. Diese Mechanismen erlauben es dem Dekodierer in jedem Schritt unterschiedliche Elemente der Eingabesequenz besonders zu berücksichtigen. Transformer sind eine gänzlich neue Architektur für neuronale Netze. Durch die Rückverbindungen in den rekurrenten Schichten von RNNs, welche für viele Abhängigkeiten zwischen Neuronen aufeinanderfolgender Schichten verantwortlich sind, sind RNNs schwer zu parallelisieren. Da Transformer gänzlich

ohne rekurrente Schichten auskommen, sondern lediglich Aufmerksamkeitsmechanismen nutzen, können diese einfacher parallelisiert werden. Dies hat zur Folge, dass Transformer im allgemeinen deutlich schneller zu trainieren sind als RNNs.

2.9 Evaluationsmetriken

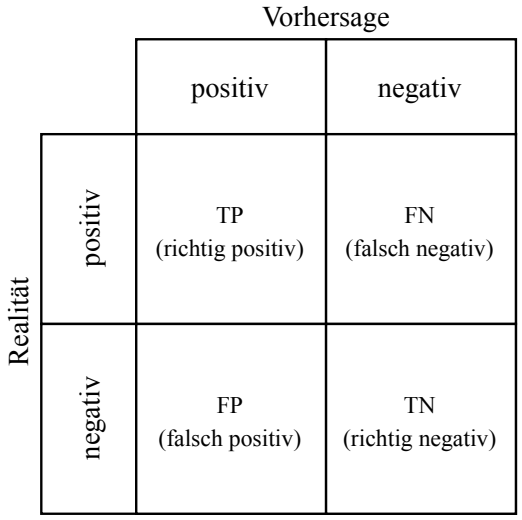


Abbildung 2.5: Schematische Darstellung der Wahrheitsmatrix und ihrer vier möglichen Fälle

Für einen Menschen ist es oft relativ einfach die Sinnhaftigkeit eines Satzes oder einer Beschreibung zu beurteilen. Unsere Erfahrung und unsere Fähigkeit Konzepte zu identifizieren und zu vergleichen ermöglichen es uns eine solche Entscheidung auf einer Verständnisebene zu fällen. Maschinen hingegen müssen ohne ein solches Verständnis von abstrakten Konzepten Entscheidungen treffen. Um dieses Problem zu umgehen, können deterministische Evaluationsmetriken eingesetzt werden.

Die Wahrheitsmatrix (engl. *confusion matrix*) ist eine Möglichkeit die Entscheidungen eines Klassifizierers anhand der von ihm vorhergesagten und den tatsächlichen Klassenzuordnungen zu bewerten [Tha18]. Handelt es sich um ein binäres Klassifizierungsproblem, so ist die Wahrheitsmatrix eine 2×2 Matrix deren Reihen die vorhergesagten Klassen und deren Spalten die tatsächlichen Klassen repräsentieren. Diese Aufteilung ergibt vier unterschiedliche Fälle, welche in Abbildung 2.5 dargestellt sind, wie ein Klassifizierer entscheiden kann:

1. **Richtig positiv (TP):** Der Klassifizierer sagt ein positives Ergebnis hervor und der tatsächliche Wert ist ebenfalls positiv.
2. **Falsch positiv (FP):** Der Klassifizierer sagt ein positives Ergebnis hervor, der tatsächliche Wert ist allerdings negativ.
3. **Falsch negativ (FN):** Der Klassifizierer sagt ein negatives Ergebnis hervor und der tatsächliche Wert ist ebenfalls negativ.
4. **Richtig negativ (TN):** Der Klassifizierer sagt ein negatives Ergebnis hervor, der tatsächliche Wert ist allerdings positiv.

Die Fälle entlang der Hauptdiagonalen der Matrix (*Richtig positiv* und *Richtig negativ*) entsprechen korrekten Klassifizierungen. Anhand der Wahrheitsmatrix lassen sich unter anderem folgende Metriken definieren:

- Die Trefferquote (engl. *recall*) eines Modells gibt das Verhältnis der richtig positiv klassifizierten Datenpunkte zu der Gesamtanzahl der tatsächlich positiven Datenpunkte an.
- Die Präzision (engl. *precision*) entspricht dem Verhältnis der richtig positiv klassifizierten Datenpunkte zu der Gesamtzahl der positiv vorhergesagten Datenpunkte.
- Die Genauigkeit (engl. *accuracy*) entspricht dem Verhältnis aller richtig klassifizierten Datenpunkte zu der Anzahl aller Datenpunkte.
- Der F1-Wert (engl. *F1-Score*) ist eine Metrik, welche die Trefferquote und Präzision als harmonisches Mittel kombiniert.

$$precision = \frac{TP}{TP + FP} \tag{2.2a}$$

$$recall = \frac{TP}{TP + FN} \tag{2.2b}$$

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{2.2c}$$

3 INDIRECT

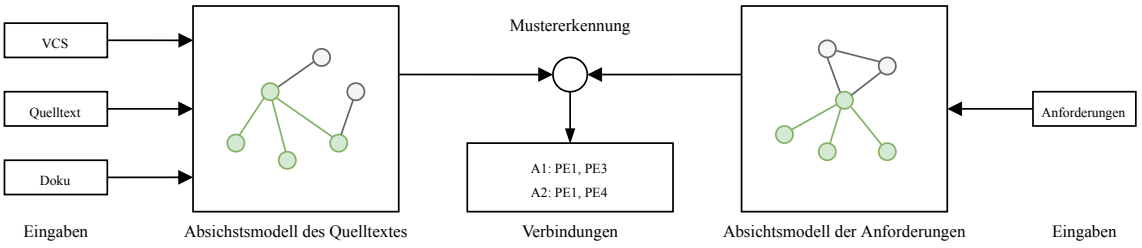


Abbildung 3.1: Schematische Darstellung der allgemeinen Architektur von INDIRECT.

In diesem Kapitel wird ein Überblick und eine Einführung in das Projekt INDIRECT (*Intent-driven Requirements-to-Code Traceability*) [Hey19] gegeben. Das Ziel von INDIRECT ist es automatisiert Rückverfolgbarkeitsinformationen aus den Anforderungen und dem Quelltext eines Softwareprojekts zu erzeugen. Da sich natürliche Sprache und Quelltext stark unterscheiden können Zuordnungen zwischen ihnen nicht direkt aufgebaut werden. Um dieses Problem zu lösen, verwendet INDIRECT sowohl auf der Anforderungs- als auch Quelltextseite ein graphenbasiertes Absichtsmodell.

Abbildung 3.1 zeigt eine Darstellung der von INDIRECT genutzten Architektur. Sowohl das Absichtsmodell für den Quelltext, als auch das Absichtsmodell für die Anforderungen werden unabhängig voneinander erzeugt. In das Quelltextabsichtsmodell fließen Informationen aus dem Quelltext, der Dokumentation oder der Versionshistorie des zu analysierenden Projekts ein. Das Absichtsmodell für die Anforderungsseite wird basierend auf den Anforderungsschreiben des Projekts erzeugt. In einem weiteren Schritt werden ähnliche Muster in beiden Absichtsmodellen identifiziert. Anhand dieser Muster kann dann eine Zuordnung von Anforderungen zu Programmelementen geschehen.

3.0.1 Absichtsmodell des Quelltextes

Das Quelltextabsichtsmodell wurde im Rahmen einer Masterarbeit von Eurich erarbeitet und stellt Beziehungen, Ähnlichkeiten und Abhängigkeiten zwischen Programmelementen und Themen dar. Dieses Modell wird in einer mehrstufigen Quelltextanalyse erzeugt: Zuerst wird der Quelltext sowohl syntaktisch, als auch lexikalisch analysiert. Diese Analysen liefern Zusammenhänge und semantische Ähnlichkeiten zwischen Programmelementen. Die Programmelemente bilden dann die Knoten eines Graphen und die gefundenen Zusammenhänge die Kanten zwischen ihnen.

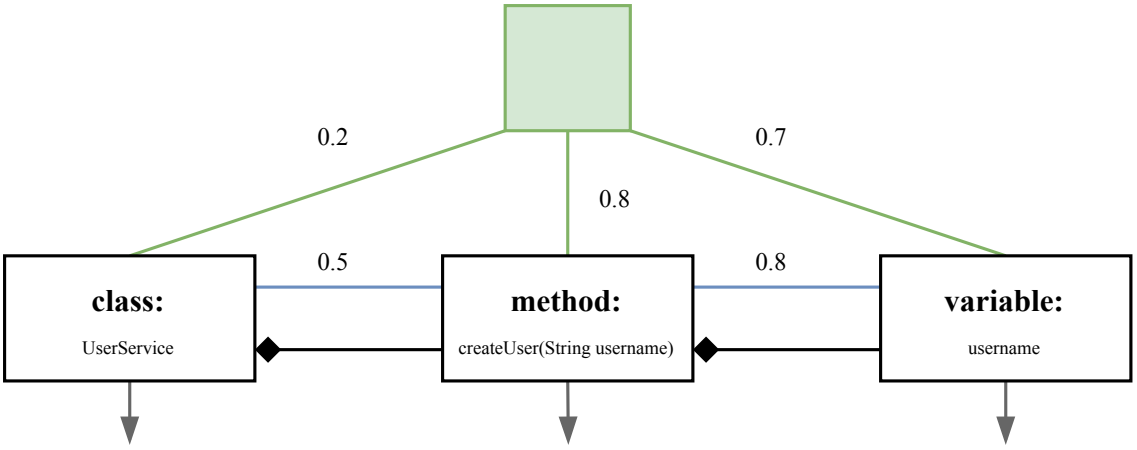


Abbildung 3.2: Schematische Darstellung des Quelltextabsichtsmodells.

Nun wird anhand dieses Graphen eine hierarchische Clusteranalyse durchgeführt, welche Gruppierungen von semantisch zusammenhängenden Programmelementen erzeugt. Jede dieser Gruppierungen enthält außerdem einen sogenannten Absichtsknoten. Er ist mit jedem Programmelement der Gruppierung verbunden und die Kantengewichte dieser Verbindungen geben an wie wichtig ein Element für die Semantik der Gruppierung als ganzes ist. Somit ist es möglich zu erkennen welche Programmelemente eine gemeinsame Absicht implementieren.

Abbildung 3.2 zeigt den Zusammenhang zwischen diesen Komponenten: Der grüne Absichtsknoten besitzt Verbindungen zu den drei Programmelementen der Gruppierung, die Gewichtung dieser Kanten ist die semantische Relevanz des Programmelements für die Absicht. Die Programmelemente verbinden untereinander Eltern-Kind-Kanten, sowie blaue Kanten, welche den semantischen Zusammenhang zwischen Programmelementen darstellen. Außerdem können die Programmelemente noch AST-Beziehungen zu anderen Programmelementen aufweisen, diese sind als graue Pfeile eingezeichnet. Basierend auf diesen Zusammenhängen können die Quelltextgruppierungen von INDIRECT wie in Definition 3.1 formal eingeführt werden.

Definition 3.1: Programmelemente und Quelltextgruppierungen

Ein Programmelement e repräsentiert entweder eine Methode, eine Klasse oder eine Schnittstelle aus einem Softwareprojekt. Wie diese Programmelemente konkret repräsentiert werden spielt zu diesem Zeitpunkt keine Rolle, sie müssen jedoch eindeutig identifizierbar sein. Konkrete Repräsentationen von Quelltext werden im weiteren Verlauf der Arbeit betrachtet. Eine Quelltextgruppierung $G = \{e_1, \ldots, e_n\}$ ist dann eine Menge von Programmelementen e_i.

4 Verwandte Arbeiten

In diesem Kapitel werden die verwandten Arbeiten aus den Bereichen der Themenmodellierung und insbesondere Quelltextzusammenfassung präsentiert. Diese Bereiche sind für diese Arbeit von besonderem Interesse, da sie sich mit ähnlichen Problemstellungen befassen. Zunächst werden Arbeiten vorgestellt, welche auf statistischen Themenmodellen basieren. Danach wird der Fokus auf Arbeiten gelegt, die verschiedene Verfahren des maschinellen Lernens nutzen.

4.1 Statistische Verfahren

Movshovitz *et al.* analysieren in **„Natural Language Models for Predicting Programming Comments“** [MC13] verschiedene Modelle zur Erzeugung von Klassenkommentaren. Sie vergleichen sowohl n-Gramm basierte, als auch LDA basierte Modelle, welche auf denselben Daten trainiert wurden. Um die Modelle zu trainieren wird Quelltext von neun quelloffenen Java-Projekten genutzt. Außerdem wird ein Datensatz von Java-bezogenen Stackoverflow-Fragen [1] genutzt, um mehr natürlichsprachlichen Text mit in das Training einzubeziehen. Sie kommen zu dem Ergebnis, dass ein LDA basiertes Modell häufiger korrekte Kommentare vorhersagt. Darüber hinaus stellen sie fest, dass Modelle, welche auf Daten aus dem Projekt, für das Kommentare erzeugt werden sollen, trainiert wurden, ebenfalls erfolgreicher sind.

In **„Using Topic Model to Suggest Fine-Grained Source Code Changes“** [NNN16] von Nguyen *et al.* wird ein auf LDA basierendes Themenmodell vorgestellt (**TasC**), welches den Kontext von Quelltextänderungen nutzt um Vorschläge zur Änderung und Fehlerbehebung von Quelltext zu machen. Das Verfahren kann außerdem die semantische Ähnlichkeit von Quelltextabschnitten basierend auf ihrem Änderungskontext bestimmen. Um eine Änderung zu definieren, nutzen die Autoren aufeinanderfolgende Einbuchungen der Versionskontrolle eines Projektes. Konkret betrachten sie Paare von Teilbäumen des abstrakten Syntaxbaums eines Programms (s, t), wobei s den Teilbaum eines Quelltextabschnitts vor einer Veränderung und t den Teilbaum für denselben Quelltextabschnitt nach einer Veränderung repräsentieren. Handelt es sich bei einer Änderung um das Hinzufügen von neuem Quelltext, so wird s durch einen leeren Baum repräsentiert. Beschreibt eine Änderung das Löschen eines Quelltextabschnitts, so wird t durch einen leeren Baum repräsentiert. Basierend auf dieser Definition wird der Änderungskontext an einer Quelltextstelle als eine

[1] https://stackoverflow.com/

Menge von Änderungen definiert, welche dieselbe Änderungsabsicht haben und deshalb häufig zusammen auftreten. Um LDA auf dieses Problem anwenden zu können wird eine Änderung durch seine Quelltexttokens als Satz aufgefasst. Eine Einbuchung in ein Versionskontrollsystem ist eine Menge von Änderungen und kann somit als Dokument modelliert werden. Das Modell wurde mithilfe eines Datensatzes von 88 quelloffenen Java-Projekten mit insgesamt 435 Tausend eingebuchten Methodenänderungen evaluiert. Die Autoren vergleichen ihr Modell mit zwei weiteren Modellen, welche keinen Änderungskontext zur Findung von ähnlichen Änderungen nutzen. Das erste Modell (*Exact*) nutzt einen exakten Abgleich, um für eine gegebene Änderung alle Änderungen mit einem isomorphen AST zu finden. Das zweite Modell (*Similar*) nutzt das Verhältnis zwischen der Länge der längsten gemeinsamen Teilsequenz und der Gesamtlänge der Quelltexttokensequenzen zweier Änderungen um Ähnlichkeit zu definieren. Unter Betrachtung der besten Vorhersagen der Modelle (Top-1 bei einem Ähnlichkeitsschwellwert von 80%) innerhalb eines Projektes erreicht Exact eine Genauigkeit von $0,20\%$, Similar erreicht eine Genauigkeit von $0,32\%$ und TasC erreicht eine Genauigkeit von $0,51\%$.

Zhang *et al.* stellen in **„Mining Source Code Topics Through Topic Model and Words Embedding“** [ZSA+16] das **EmbTE** (engl. *Embedded Topic Extraction*) Verfahren vor, welches Worteinbettungen basierend auf dem word2vec Modell nutzt, um Themen in Quelltext zu identifizieren. Die Autoren sprechen in ihrer Arbeit lediglich von Dokumenten, da sie jedoch sowohl Informationen auf Klassenebene, als auch auf Methodenebene nutzen scheint ein Dokument einer Java Quelltextdatei zu entsprechen. Zunächst wird jeder Term eines Dokuments durch ein word2vec Modell als reellwertiger Vektor repräsentiert. Ein Dokument entspricht somit einer Menge von Vektoren. Dann wird der Schwerpunkt (engl. *centroid*) der Menge bestimmt, welcher das Hauptthema des Dokuments repräsentiert. Anschließend werden die n nächsten Nachbarn des Schwerpunkts bestimmt, welche die Top-n Schlagwörter für das Dokument repräsentieren. Die Autoren nutzen einen heuristischen Suchalgorithmus, welcher den Zusammenhang der Schlagwortmenge nutzt, um zu identifizieren, welche Teile des Quelltextes (bspw. Bezeichner, Kommentare) die meisten semantischen Informationen beinhalten. In einer Evaluation auf 100 quelloffenen Java-Projekten stellen sie fest, dass Methodenbezeichner, Methodenkommentare, Klassenbezeichner und Klassenkommentare die größten positiven Auswirkungen auf ihr Modell haben. Diese Evaluation wurde sowohl für eine Variante des EmbTE Modells, welche auf einem CBOW-word2vec Modell basiert, als auch für eine Variante, welche auf einem Skip-Gram-word2vec Modell basiert, sowie für ein LDA und NMF (engl. *non-negative matrix factorization*, ein Verfahren statistisches Verfahren ähnlich wie LSI) Modell durchgeführt. Die Autoren geben die Zusammenhangswerte der von den Modellen erzeugten Schlagwortmengen lediglich in Diagrammen wieder, was einen direkten Vergleich erschwert. Sie geben jedoch an, dass das NMF Modell durchgehend am schlechtesten abschneidet, während das EmbTE-CBOW Modell am besten und das EmbTE-Skip-Gram Modell ähnlich wie LDA abschneidet. Des Weiteren stellen sie fest, dass das Erhöhen der Anzahl an Schlagwörter für ein Dokument den Zusammenhang der Schlagwortmenge verringert.

Mahmoud und Bradshaw stellen in **„Semantic topic models for source code analysis“** [MB17] ein Verfahren zur Quelltextzusammenfassung vor, welches anders als LDA und darauf basierende Themenmodelle, explizit zur Themenmodellierung in Quelltext konzipiert ist und somit einige ihrer Probleme beheben soll. Die Autoren identifizieren zunächst Probleme mit Verfahren wie LDA, wenn sie auf Quelltext angewandt werden: Viele Themenmodelle benötigen eine genaue Einstellung ihrer Parameter, um ein gewünschtes Ergebnis zu erzielen, was es schwer macht sie automatisiert anzuwenden. Aufgrund der statistischen Eigenschaften von Verfahren wie LDA und LSI kann es vorkommen, dass sukzessive Ausführungen der Verfahren auf denselben Daten unterschiedliche Themenverteilungen erzeugen. Außerdem leidet Quelltext oft unter einer Datenknappheit, da das genutzte Vo-

kabular meist klein ist und Quelltext sich oft wiederholt. Das Verfahren beginnt mit der Tokenisierung und Vorverarbeitung der Java-Klassen eines Softwareprojekts. Anschließend wird die paarweise Ähnlichkeit aller Quelltexttokens basierend auf dem NGD (engl. *normalized google distance*) Ähnlichkeitsmaß berechnet. Dieses Maß basiert auf der Anzahl an Google-Ergebnissen zweier Terme, wird in der Arbeit von Mahmoud und Bradshaw jedoch basierend auf den Kookkurenzinformationen von Termen berechnet. Das Maß ordnet Termen, welche lediglich gemeinsam Auftreten, eine perfekte Ähnlichkeit (100%) zu und trägt somit zur Lösung des Datenknappheitsproblems bei. Die Autoren nutzen ein auf dem NGD Maß basierendes hierarchisches agglomeratives Gruppierungsverfahren, um Termgruppen zu bilden, welche den Themen des Quelltextes entsprechen. Um die Zugehörigkeit eines Terms zu einem Thema zu bestimmen berechnen die Autoren den Durchschnitt der paarweisen NGD-Ähnlichkeit des Terms zu allen anderen Termen des Themas. Anschließend berechnen sie die Zugehörigkeiten zwischen den Java-Klassen des Projekts und den identifizierten Themen basierend auf den Ähnlichkeiten der Terme einer Java-Klasse und den Termen eines Themas. Zur Evaluation ihres Verfahrens nutzen Mahmoud und Bradshaw drei Java-Projekte aus unterschiedlichen Anwendungsdomänen. Die Evaluation wird in Form einer Nutzerstudie durchgeführt und vergleicht STM mit LDA und einer Abwandlung von LSI Im ersten Teil der Studie müssen die Teilnehmer feststellen, welcher Term am unähnlichsten zu den restlichen Termen eines Themas ist. Dieser Teil prüft die Fähigkeit des Modells Themen mit einem starken Zusammenhang zu erzeugen. Im zweiten Teil der Studie müssen die Teilnehmer feststellen, welches Thema aus einer Menge an Themen für eine gegebene Java-Klasse am irrelevantesten ist. Dieser Teil prüft die Fähigkeit des Modells Themen korrekt zu Dokumenten zuzuordnen. In der Nutzerstudie schneidet STM auf allen getesteten Projekten am besten ab. Die Autoren merken an, dass die Evaluation mittels einer Nutzerstudie die Validität ihrer Ergebnisse gefährden kann, da sie von dem Domänenwissen ihrer Teilnehmer abhängig ist.

4.2 Verfahren des maschinellen Lernens

In **„Suggesting Accurate Method and Class Names“** [ABBS15] von Allamanis *et al.* wird ein Modell zur automatischen Erzeugung von Variablen-, Methoden- oder Klassenbezeichnern vorgestellt. Sie nutzen die Tokens aus dem Methodenrumpf zusammen mit weiteren expliziten Merkmalen (bspw. der Rückgabetype der Methode oder die Subtokens der Felder der umschließenden Klasse), um ein logbilineares Sprachmodell zu trainieren. Konkret nutzen sie die Subtokens von Bezeichnern um unbekannten, zusammengesetzten Bezeichnern einen Vektor zuordnen zu können. Dieser Vektor entspricht der Summe aller Wortvektoren der Subtokens. Die Nutzung von Subtokens ermöglicht die Bildung von unendlich vielen neuen Bezeichnern für eine Methode. Dadurch wird eine lineare Suche nach dem besten Bezeichner unmöglich. Aus diesem Grund nutzen Allamanis *et al.* eine Strahlensuche (engl. *beam search*), einen Suchalgorithmus der in jedem Schritt das Element mit der höchsten Wahrscheinlichkeit wählt, um die maximal B Subtokens eines Bezeichners zu wählen. Das Modell wurde mithilfe von 20 quelloffenen Java-Projekten aus verschiedenen Anwendungsbereichen evaluiert. Sie berechnen den F1-Wert und die Genauigkeit des Modells. Diese Berechnungen basieren auf den Subtokens des erzeugten Methodennamen, sodass Vorhersagen nicht exakt mit dem vorgegebenen Methodennamen übereinstimmen müssen. Dadurch werden nur teilweise übereinstimmende Methodennamen nicht direkt ausgeschlossen. Statt eines Schwellwertes berichten Allamanis *et al.* eine Vorschlagsrate, eine niedrige Vorschlagsrate bedeutet, dass das Modell nur Bezeichner mit einer hohen Zuversicht zurückgibt. In ihrer Evaluation vergleichen Allamanis *et al.* ein Subtoken- und ein Token-basiertes Modell, jeweils bei 5% und 20% Vorschlagsrate. Um zu erkennen, wie stark sich spezielle explizite Merkmale auf die Leistung des Modells auswirken trainieren sie pro Merkmal ein Modell, welches zusätzlich zu den Tokens einer Methode exakt dieses

Merkmal als Eingabe erhält. Anschließend vergleichen sie die Abweichungen der erzielten F1 und Genauigkeitswerte für jedes dieser Modelle mit einem Modell, welches lediglich die Tokens einer Methode als Eingabe erhält. Bei der Erzeugung von Methodenbezeichnern mit einer Vorschlagsrate von 5% wirken sich die Subtokens des Namen der definierenden Klasse, ihrer Superklasse und der von ihr implementierten Schnittstellen am stärksten auf das Modell aus (+11,7 PP F1). Der Rückgabetyp der Methode wird als zweitwichtigstes Merkmal identifiziert (+10,0 PP F1). Ein Modell, welches alle betrachteten Merkmale einbezieht erreicht eine Abweichung von +17,0 PP im Bezug auf den F1-Wert des Modells ohne explizite Merkmale. Die Autoren geben zwar absolute Abweichungen zwischen den von ihnen evaluierten Modellen an, jedoch keine absoluten Werte. Dies ermöglicht zwar einen Vergleich zwischen der Güte der von ihnen gewählten expliziten Merkmalen, allerdings keinen Vergleich ihres Modells mit den Modellen anderer Autoren.

Allamanis *et al.* stellen in **„A Convolutional Attention Network for Extreme Summarization of Source Code“** [APS16] ein weiteres Modell vor, welches Faltungen (engl. *convolutions*) und Aufmerksamkeitsmechanismen nutzt, um Quelltext zusammenzufassen. Die Eingabe an das Modell, die Subtokens einer Java Methode, wird zuerst durch ein rekurrentes neuronales Netz kodiert. Anschließend wird diese kodierte Repräsentation gemeinsam mit den gefalteten Eingabetokens zu einer Aufmerksamkeitsmatrix kombiniert. Sie enthält für jeden Schritt in der Eingabesequenz einen Vektor, der kodiert welche Stellen in der Sequenz für das aktuelle Token von Bedeutung sind. Außerdem erweitern die Autoren diesen faltungsbasierten Aufmerksamkeitsmachanismus durch einen Kopie-Aufmerksamkeitsmechanismus. Dieser ermöglicht es dem Modell Subtokens, welche nicht in dem Vokabular des Modells vorkommen aus der Eingabesequenz zu kopieren. Um einen vollständigen Methodennamen vorherzusagen, nutzen die Autoren eine Mischung aus einer Breitensuche und einer Strahlensuche ausgehend von dem speziellen Starttoken $< s >$. Das Modell schlägt nun iterativ Subtokens vor und hängt sie an die Ausgabesequenz an, bis es das Schlusstoken $< /s >$ vorschlägt. Das Modell wird auf 10 quelloffenen Java-Projekten evaluiert. Für jedes Projekt werden drei Aufmerksamkeitsmechanismen (faltungsbasierte Aufmerksamkeit, Kopie-Aufmerksamkeit und eine standard Aufmerksamkeit nach Bahdanau *et al.* [BCB16]) verglichen und es werden die F1-Werte der Top-1 und der Top-5 Vorhersagen des Modells berechnet, wobei unter den Top-n der beste F1-Wert gewählt wird. Die F1-Werte werden durch den vorhergesagten und den tatsächlichen Methodennamen auf Subtoken-Ebene berechnet. Analog werden Präzision und Sensitivität, sowie die Quote exakt korrekt vorhergesagter Bezeichner berechnet. Die Evaluation zeigt, dass das Modell, welches den Kopie-Aufmerksamkeitsmechanismus verwendet, auf beinahe allen Testprojekten die besten F1-Werte erreicht (Durchschnitt auf allen Projekten 44,7% F1 unter den Top-1 und 59,6% unter den Top-5).

Iyer *et al.* stellen in **„Summarizing Source Code using a Neural Attention Model“** [IKCZ16] das **CODE-NN** Modell zur Zusammenfassung von C#-Quelltext sowie SQL-Anfragen vor. Es basiert auf einer rekurrenten Architektur mit LSTMs und Aufmerksamkeitsmechanismen. Die Eingabetokens werden durch Worteinbettungen repräsentiert. Anschließend werden sie durch die LSTM-Schichten kodiert. Das Modell erzeugt ausgehend von einem START Token iterativ die Tokens der Ausgabesequenz, bis es ein END Token wählt. Hierzu verwenden die Autoren eine Strahlensuche. Iyer *et al.* trainieren ihr Modell auf einem automatisch aus Stackoverflow-Beiträgen erzeugten Datensatz trainiert. Stackoverflow-Beiträge wurden als Datenquelle genutzt, da die akzeptierte Antwort in einem Beitrag meist Quelltext mit einer hochqualitativen Beschreibung dessen Funktion kombiniert. Das Modell wird sowohl automatisiert, durch Metriken, als auch durch eine Nutzerstudie evaluiert. Die Autoren berechnen jeweils die *METEOR* und die *BLEU* Metriken für ihr Modell und für drei weitere Vergleichsmodelle (MOSES [KHB+07], SUM-NN [RCW15] und ein Modell, welches den Namen der Methode aus dem Trainingsdatensatz, deren Tokens die

Levenshtein-Distanz zu den Tokens der Eingabe minimieren). CODE-NN erreicht auf dem Evaluationsdatensatz einen METEOR-Wert von 12,3% (Abweichung von +1,7 PP zum zweitbesten Modell SUM-NN) und einen BLEU-4-Wert von 20,5% (Abweichung von +1,5 PP zum zweitbesten Modell SUM-NN).

„Automatically Generating Commit Messages from Diffs using Neural Machine Translation“ [JAM17] von Jiang *et al.* behandelt ein Modell zur automatischen Erzeugung von Einbuchungsnachrichten (engl. *commit messages*) basierend auf dem Unterschieden zweier Einbuchungszeitpunkte. Das Modell basiert auf einer rekurrenten Kodierer-Dekodierer Architektur mit Aufmerksamkeitsmechanismen. Jiang *et al.* nutzen sogenannte *diffs*, also die Änderungen zwischen zwei Einbuchungen in der Versionskontrollhistorie eines Projekts. Aus diesen *diffs* geht hervor, welche Zeilen eines Quelltextes neu hinzugekommen, entfernt oder geändert wurden. Für das Training und zur Evaluation wird ein Datensatz bestehend aus über zwei Millionen Einbuchungen von quelloffenen Java-Projekten genutzt. Zunächst werden Datenpunkte mit zu langen *diff* Sequenzen (> 100 Tokens) und zu langen Beschreibungen (> 30 Tokens) aussortiert, der Datensatz umfasst danach noch 75 Tausend Einbuchungen. Die Autoren implementieren einen Filter, welcher die Einbuchungsnachrichten erhält, welche mit einem Verb direkt gefolgt von einem Objekt beginnen (bspw. „Implementiere neue Testfälle“, wobei *Testfälle* das direkte Objekt des Verbs *Implementiere* ist). Nach Anwenden dieses Filters umfasst der Datensatz noch 32 Tausend Einbuchungen. Hiervon werden 3000 Einbuchungen als Testdatensatz, 3000 Einbuchungen als Evaluationsdatensatz und 26 Tausend Einbuchungen als Trainingsdatensatz verwendet. Das Vokabular der Beschreibungen umfasst 16 Tausend Terme, das der *diffs* 50 Tausend Terme. Das Modell basiert auf der Nematus [SFC+17] Werkzeugkasten (engl. *toolkit*). Zur Evaluation werden MOSES [KHB+07] (ein phrasenbasiertes Übersetzungssystem) und zwei Varianten des Modells von Jiang *et al.* verglichen. Sie trainieren ein Modell, welches einen Verb-Objektsatz Filter verwendet und eines ohne einen solchen Filter. Alle Modelle werden auf einem Verb-Objektsatz gefilterten Datensatz evaluiert. MOSES erreicht einen BLEU-Wert von 3,63%. Das Modell mit dem Filter erreicht auf dem Test-Datensatz einen BLEU-Wert von 31,92%. Dasselbe Modell ohne diesen Filter erreicht einen Wert von 32,81%. Wird das nicht gefilterte Modell auf einem nicht gefilterten Datensatz evaluiert erreicht es einen BLEU-Wert von 23,10. Die Autoren merken an, dass das Modell sehr gute Beschreibungen liefert, wenn es ähnliche Quelltextänderungen bereits zuvor gesehen hat. Allerdings stellen sie auch fest, dass oft Beschreibungen von niedriger Qualität erzeugt werden, besonders wenn das Modell Änderungen beschreiben soll, die es in dieser Form noch nie gesehen hat.

Hu *et al.* stellen in **„Deep code comment generation“** [HLX+18] das **DeepCom** Modell zur Erzeugung von Kommentaren für Java-Methoden vor. Das Modell nutzt anders als die meisten vorherigen Arbeiten nicht die Tokens des Quelltextes, sondern die SBT-Darstellung (eine flache Repräsentation eines AST) des abstrakten Syntaxbaums der zu beschreibenden Methode als Eingabe. Es basiert auf LSTMs und Aufmerksamkeitsmechanismen, welche mithilfe des AST Zusammenhänge zwischen der Struktur und der Semantik des Quelltextes lernen sollen. Das Modell wird auf einem Datensatz von Methoden und deren JavaDoc-Kommentaren aus über 9000 quelloffenen Java-Projekten evaluiert. Die Autoren vergleichen ihr Modell basierend auf der SBT-Darstellung mit CODE-NN von Iyer *et al.* und drei weiteren Modellen, darunter ein weiteres Modell von Hu *et al.*, welches lediglich eine andere Form der AST-Traversierung nutzt. DeepCom mit der SBT-Darstellung erreicht einen BLEU-Wert von 38,17%. CODE-NN von Iyer *et al.* erreicht auf demselben Datensatz einen BLEU-Wert von 25,30%. Ein standard seq2seq-Modell erreicht einen Wert von 34,87%. Die Autoren argumentieren, dass das CODE-NN Modell es nicht schaffe die Semantik des Quelltextes zu verstehen, da es direkt auf den eingebetteten Quelltexttokens arbeitet und nicht auf einer kodierten AST Repräsentation. Zusätzlich vergleichen sie CODE-NN und das seq2seq-Modell auf dem von Iyer *et al.* genutzten Datensatz. Auf dem C# Datensatz

erreicht das seq2seq-Modell einen BLEU-Wert von 30,0%. CODE-NN erreicht einen Wert von 20,4%. Die Autoren räumen jedoch ein, dass sowohl die automatische Evaluation als auch die Qualität der für das Training gesammelten Quelltextkommentare dieses Ergebnis beeinflussen kann.

LeClair *et al.* stellen in **„A Neural Model for Generating Natural Language Summaries of Program Subroutines“** [LJM19] das **ast-attendgru** Modell zu Beschreibung von Methoden vor, das Informationen aus dem AST einer Methode mit Informationen aus ihren Tokens kombiniert. Das Modell von LeClair *et al.* ist eine GRU basierte Kodierer-Dekodierer Architektur. Zur Nutzung von AST Informationen wird der AST der zu beschreibenden Methode in die von Hu *et al.* vorgestellte SBT-Darstellung [HLX+18] transformiert. Sowohl die Tokens der Methode, als auch ihr SBT-kodierte AST werden einzeln durch Kodierer verarbeitet. Anschließend werden Aufmerksamkeitsmechanismen auf die Sequenzen angewandt und in einer kombinierten Form an den Dekodierer weitergereicht. Die Dekodierung in eine Tokensequenz wird um Variablen zu minimieren und um Rechenleistung einzusparen, durch einen gierigen Algorithmus (engl. *greedy algorithm*), statt durch eine Strahlensuche Für das Training sowie für die Evaluation wird ein Datensatz aus über zwei Millionen Java Methoden mit dazugehörigen Kommentaren und AST-Sequenzen genutzt. ast-attendgru erreicht einen BLEU-Wert von 19,6%. CODE-NN von Iyer *et al.* erreicht auf demselben Datensatz einen Wert von 9,95%. Die Autoren kommen zu dem Schluss, dass das Einbeziehen von AST-Informationen hilfreich für das Zusammenfassen von Java Methoden sein kann, wenn die Quelltext-Tokens nicht genügend Informationen über die Semantik der Methode in sich tragen.

„code2seq: Generating Sequences from Structured Representations of Code“ [ABLY19] von Alon *et al.* beschreibt ein Modell zu Zusammenfassung von Java- und C#-Methoden. Das von ihnen implementierte LSTM basierte Kodierer-Dekodierer Verfahren unterscheidet sich durch die gewählte Quelltextrepräsentation von anderen Arbeiten. Eine Java-Methode wird demnach durch eine Menge von Pfaden, welche je zwei der Blattknoten des AST der Methode verbinden charakterisiert. Diese Darstellung durch AST Pfade erlaubt es dem Modell Muster in den Beziehungen von Programmelementen zu lernen, welche nicht oberflächlich in den Tokens der Methode vorliegen. Die Autoren evaluieren ihr Modell auf drei Java-Datensätzen von verschiedenen Größen durch Berechnung des F1-Wertes, der Sensitivität und der Präzision des Modells. Auf dem mittelgroßen Datensatz erreicht das code2seq Modell einen F1-Wert von 53,23%. Das Modell von Allamanis *et al.* [APS16] erreicht einen Wert von 37,16%. code2seq wird außerdem auf dem Datensatz von Hu *et al.* [HLX+18] evaluiert und mit DeepCom verglichen. Hu *et al.* gaben in ihrer Arbeit einen BLEU-Wert von 38,17% an, Berechnungen von Alon *et al.* zeigen jedoch einen BLEU-Wert von 8,97% für DeepCom. Auf diesem Datensatz erreicht code2seq einen BLEU-Wert von 14,53%. Die Autoren weisen außerdem darauf hin, dass sich dieses Verfahren nicht auf eine bestimmte Programmiersprache beschränkt, sondern auf den Quelltext einer jeden Programmiersprache anwendbar ist, deren Quelltext sich als AST darstellen lässt.

Haque *et al.* stellen in **„Improved Automatic Summarization of Subroutines via Attention to File Context“** [HLWM20] eine Weiterentwicklung des Modells von LeClair *et al.* [LJM19] vor. Das Modell basiert auf dem Kodierer-Dekodierer Modell von LeClair *et al.* und erweitert dieses um einen dritten Kodierer, welcher den Dateikontext der zu beschreibenden Methode als Eingabe erhält. Der Dateikontext wird dabei als eine Matrix aus n Tokens für m andere Methoden in derselben Datei definiert. Auf diesen Kontext werden ebenfalls Aufmerksamkeitsmechanismen angewandt und er wird mit den Ausgaben der beiden bereits vorhandenen Kodierern kombiniert. Zur erzeugung der Ausgabesequenzen wird wie auch bei **ast-attendgru** ein gieriger Algorithmus verwendet. Der genutzte Datensatz ist eine um Dateikontext erweiterte Version des Datensatzes aus [LJM19]. Die Autoren vergleichen ihr neues **ast-attendgru-fc** Modell mit den ast-attendgru [LJM19] und code2seq [ABLY19]

Modellen. Außerdem vergleichen sie die Auswirkungen der Nutzung von Dateikontextinformationen und verschiedenen AST-Kodierungsverfahren. Das ast-attendgru Modell erreicht einen BLEU-Wert von 18,69%, ast-attendgru-fc erreicht einen Wert von 19,95%. code2seq erreicht einen BLEU-Wert von 18,84%, eine erweiterte Version, welche Dateikontextinformationen nutzt erreicht einen Wert von 19,11%. Die Autoren zeigen, dass Modelle durch Nutzung von Dateikontextinformationen bessere Beschreibungen erzeugen können, als äquivalente Modelle ohne diese Informationen. Außerdem stellen sie fest, dass dieser Verbesserungsansatz orthogonal zu anderen Verfahren ist und somit potenziell auch zur Erweiterung anderer Verfahren geeignet ist.

Ahmad *et al.* stellen in **„A Transformer-based Approach for Source Code Summarization“** [ACRC20] einen Transformer basierten Ansatz zur Zusammenfassung von Java- und Python-Methoden vor. Die Tokens der zu beschreibenden Methode werden eingebettet und zur Dekodierung der Ausgabesequenz wird eine Strahlensuche verwendet. Das Modell wird auf Datensätzen aus Java- sowie Python-Methoden mit ihren entsprechenden Quelltextkommentaren trainiert und evaluiert. Außerdem wird das Modell mit anderen Modellen verglichen, darunter *CODE-NN* [IKCZ16] und *DeepCom* [HLX+18]. Die Modelle werden mit den BLEU, METEOR und ROUGE Metriken evaluiert. CODE-NN erreicht auf dem Java-Datensatz einen BLEU-Wert von 27,60%. DeepCom erreicht einen Wert von 39,75%. Das Modell von Ahmad *et al.* erreicht einen BLEU-Wert von 44,58%. Die Autoren stellen fest, dass das Modell trotz seiner simplen Architektur bessere Zusammenfassungen erzeugt, als die Modelle auf Basis von rekurrenten Netzen. Des Weiteren verweisen sie darauf, dass die Struktur des Quelltextes ebenfalls explizit in ein solches Modell miteinbezogen werden könnte.

5 Analyse und Entwurf

Das Ziel dieser Arbeit ist es Beschreibungen für die Quelltextgruppierungen von INDIRECT zu erarbeiten um Verbindungen zwischen den Anforderungen und dem Quelltext eines Softwareprojekts zu finden. Quelltext zu verstehen ist selbst für Menschen keine einfache Aufgabe. Um zusammenhängende Quelltextabschnitte maschinell verstehen zu können, wird eine Repräsentation ihrer geteilten Semantik benötigt. Hierzu muss zunächst untersucht werden, wie Quelltext repräsentiert werden kann, wie Beschreibungen von Semantik repräsentiert werden können und welche semantischen Informationen in Quelltext vorkommen. Im Rahmen der Analyse wird außerdem betrachtet, wie natürlichsprachliche Elemente in Quelltext vorverarbeitet werden können, um ihren semantischen Informationsgehalt zu erhöhen. Mithilfe einer Vorstudie werden mehrere Verfahren zur Themenmodellierung miteinander verglichen. Anschließend wird auf Basis der Ergebnisse der Vorstudie ein Verfahren zur Beschreibung der geteilten Semantik von Programmelementen entworfen.

5.1 Quelltextrepräsentationen

In diesem Abschnitt werden verschiedene Repräsentationen von Quelltext analysiert und hingehend ihrer Eignung als Ausgangspunkt für die Themenmodellierung in INDIRECT-Quelltextgruppierungen bewertet. Dabei wird betrachtet, welche Informationen aus den einzelnen Repräsentationen hervorgehen. Außerdem wird bewertet, ob die Repräsentationen sinnvoll auf Gruppierungen von Programmelementen angewandt werden können.

5.1.1 Abstrakter Syntaxbaum

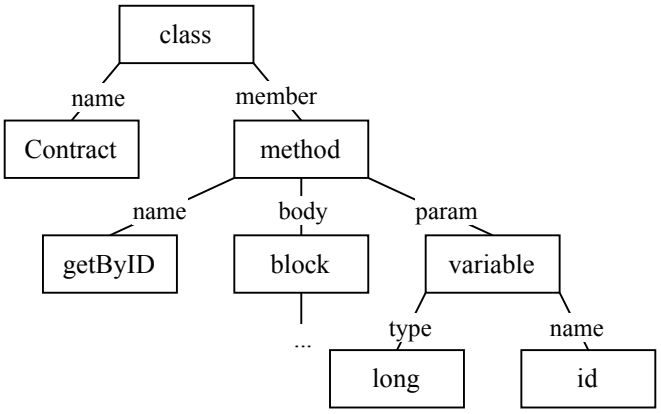


Abbildung 5.1: Schematische Darstellung eines abstrakten Syntaxbaums.

Eine Möglichkeit Quelltext zu repräsentieren ist durch einen abstrakten Syntaxbaum (AST). Dieser ist eine baumförmige Repräsentation des Quelltextes, welcher syntaktische Informationen über Beziehungen zwischen Elementen wie Klassen, Methoden oder Anweisungen kodiert [Com08].
Abbildung 5.1 zeigt einen abstrakten Syntaxbaum für eine Klasse `Contract`. Der Knoten an der Wurzel des Baumes entspricht der Klasse als Ganzes. Über Kanten werden weitere Elemente, wie Methoden oder Felder innerhalb der Klasse mit der Wurzel verbunden. Typen, Bezeichner oder konstante Werte im Quelltext werden durch Blattknoten im AST repräsentiert. Anhand des AST der Klasse `Contract` lässt sich ablesen, dass diese eine Methode namens `getByID` beinhaltet. Aus den Kindknoten des Methodenknotens lassen sich die Parameter der Methode ablesen: Im Fall von `getByID` handelt es sich um einen einzigen Parameter vom Typen `long` mit dem Bezeichner `id`.
Der AST bietet sich an, wenn man Informationen über die Beziehungen zwischen Quelltextelementen gewinnen möchte. Für diese Zwecke ist eine AST-Repräsentation von Quelltext jedoch bereits im Rahmen von Eurichs Arbeit [Eur] in die Bildung der INDIRECT-Quelltextgruppierungen eingeflossen. Somit ist uns bereits bekannt, dass die Quelltextelemente einer Gruppierung eine gemeinsame Absicht verfolgen. Im Rahmen dieser Arbeit müssen wir uns also nicht mehr mit der Ähnlichkeit der Elemente einer Gruppierung beschäftigen. Da die abstrakten Syntaxbäume der Quelltextelemente einer Gruppierung nicht notwendigerweise direkt durch AST-Kanten verbunden sind, ist außerdem nicht klar wie eine Quelltextgruppierung als Ganzes sinnvoll repräsentiert werden soll. Eine solche Möglichkeit ist es die ASTs der einzelnen Quelltextelemente zu bilden und die Gruppierung als Menge dieser ASTs zu repräsentieren. Allerdings erscheint diese Repräsentation einer Gruppierung als Ganzes nicht sehr sinnvoll, da die ASTs einzelner Programmelemente lediglich semantische Informationen über die einzelnen Elemente beinhalten. Zusammenfassend bietet sich der AST also als Repräsentation an, wenn man Informationen über die Beziehungen von Quelltextelementen gewinnen möchte. Da wir jedoch bereits eine Menge von zusammengehörigen Quelltextelementen vorliegen haben, sind wir nicht an dieser Art von Information interessiert.

5.1.2 Kontrollflussgraph

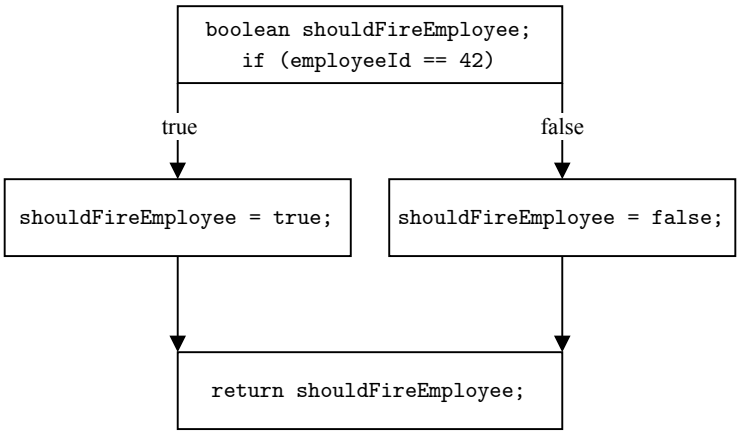


Abbildung 5.2: Schematische Darstellung eines Kontrollflussgraphen.

Der Kontrollflussgraph einer Methode ist ein gerichteter Graph, der alle möglichen Ausführungspfade durch den Rumpf der Methode abbildet. Die Knoten des Kontrollflussgraphen sind Blöcke von aufeinanderfolgende Zeilen von Quelltext. Diese Quelltextrepräsentation kodiert wie eine Methode ihre Funktionalität erfüllt, und wie die möglichen Pfade durch ihren Rumpf verlaufen.

Tufano *et al.* [TWB+18] nutzen unter anderem eine Quelltextrepräsentation durch Kontrollflussgraphen um Ähnlichkeiten zwischen Quelltextelementen festzustellen. Dazu trainieren sie ein maschinelles Lernmodell, welches den Kontrollflussgraphen einer Methode in eine kontinuierliche Vektorrepräsentation überführt. Anschließend nutzen sie diese Repräsentation um die paarweise Ähnlichkeit zwischen den Elementen des Quelltextes zu bestimmen. Eine solche Repräsentation durch Vektoren ist jedoch nicht nur für den Kontrollflussgraphen möglich und wird im weiteren Verlauf genauer betrachtet.

Der Kontrollflussgraph ermöglicht es also Informationen über den Fluss von Daten im Inneren einer Methode, sowie in Kombination mit maschinellen Lernverfahren, Informationen über die Ähnlichkeit von Methoden zu gewinnen. Problematisch an dieser Repräsentation ist, dass nicht ersichtlich ist wie die Klassen- oder Schnittstellenelemente einer Gruppierung behandelt werden sollen, da Kontrollfluss lediglich in Methodenrümpfen vorkommt. Tufano *et al.* umgehen dieses Problem, indem sie eine Klasse als Menge der Kontrollflussgraphen der in ihr enthaltenen Methoden betrachten. Da eine Quelltextgruppierung jedoch auch eine Klasse, allerdings keine ihrer Methoden, enthalten kann ist diese Repräsentation für Klassenelemente im Fall von Quelltextgruppierungen nicht sinnvoll. Zusätzlich ist auch unklar wie Quelltextgruppierungen als Ganzes repräsentiert werden sollen, da zwischen den einzelnen Elementen kein Kontrollfluss stattfindet. Eine Möglichkeit ist es auch hier Kontrollflussgraphen für die einzelnen Elemente einer Gruppierung zu erzeugen und eine Menge von diesen zu betrachten. Da die Kontrollflussgraphen der einzelnen Quelltextelemente jedoch unabhängig sind, die Repräsentation von Klassen durch Kontrollflussgraphen fragwürdig ist und sie lediglich kodieren wie eine Methode ihre Absicht ausführt, jedoch nicht was diese Absicht ist, bieten sie sich zur Repräsentation von Quelltextgruppierungen nicht an.

5.1.3 Aufrufgraph

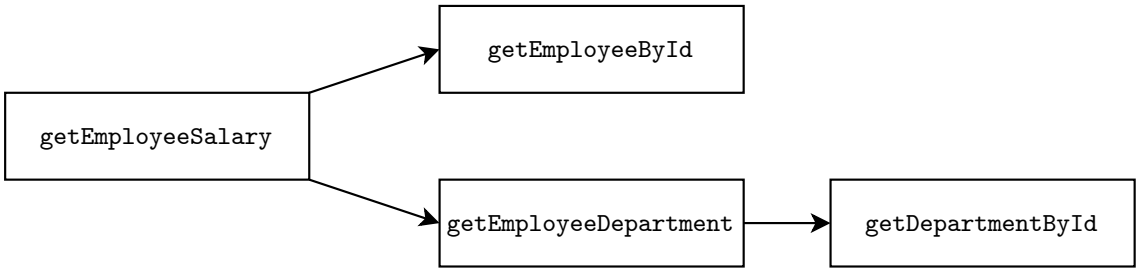


Abbildung 5.3: Schematische Darstellung eines Aufrufgraphen.

Der Aufrufgraph (engl. *call graph*) einer Methode gibt an, welche anderen Methoden des Quelltextes durch sie aufgerufen werden. Ähnlich wie beim AST handelt es sich um einen Graphen der Quelltextelemente, in diesem Fall lediglich Methoden, durch Kanten verbindet. Aus dem Aufrufgraphen geht hervor welche Methoden wie häufig von einander aufgerufen werden. Erzeugt man den Aufrufgraphen ausgehend der `main` Methode in Java, so erhält man eine Repräsentation des gesamten Quelltextes. McBurney *et al.* [MM16] nutzen den Aufrufgraphen von Java-Projekten um festzustellen in welchem Kontext diese genutzt werden und um die Wichtigkeit der Methoden zu bewerten. Anschließend nutzen sie die Bezeichner von Methoden aus dem Kontext einer Zielmethode um deren Funktionalität anhand ihrer Nutzung zu beschreiben. Der Aufrufgraph bietet sich in erster Linie an um die Wichtigkeit einer Methode für eine andere zu bewerten und Ähnlichkeiten zwischen ihnen zu erkennen. Durch die Nutzung der INDIRECT-Quelltextgruppierungen von Eurich [Eur], wissen wir jedoch bereits, dass die Programmelemente einer Gruppierung zu einander ähnlich sind. Der Aufrufgraph bietet uns also ebenfalls keinen Vorteil auf der Suche nach Beschreibung der Semantik von Quelltextgruppierungen. Außerdem stellt sich die Frage wie Klassenelemente oder ganze Quelltextgruppierungen mithilfe des Aufrufgraphen dargestellt werden sollen. Wie auch bereits für Kontrollflussgraphen behandelt, können Klassen als Menge der Aufrufgraphen ihrer Methoden betrachtet werden. Allerdings besteht auch hier das Problem, dass ein Klasse, jedoch keine ihrer Methoden, in einer Gruppierung enthalten sein kann und die Repräsentation der Klasse durch ihre Methoden somit wenig Sinn macht. Wie auch die Repräsentation durch Kontrollflussgraphen leidet die Repräsentation durch Aufrufgraphen darunter, dass Klassen und ganze Gruppierungen nicht sinnvoll repräsentiert werden können und, dass der Aufrufgraph selbst für einzelne Methoden lediglich Ähnlichkeits- und Abhängigkeitsinformationen kodiert.

5.1.4 Repräsentation durch Quelltexttoken

Die offensichtlichste Möglichkeit Quelltext zu repräsentieren ist durch die Token, die er beinhaltet. Einen großen Teil davon machen sprachspezifische Schlüsselwörter, Sonderzeichen oder benutzerdefinierte Bezeichner aus. Über die Token des Quelltextes kann auf in ihm enthaltene natürlichsprachliche Elemente, wie Bezeichner, Kommentare oder Zeichenketten zugegriffen werden.

Betrachtet man Bezeichner von Quelltextelementen als Beschreibungen ihrer Semantik, so können aus dieser Repräsentation einfache Beschreibungen einzelner Programmelemente gewonnen werden. Da wir uns jedoch mit Quelltextgruppierungen und nicht mit einzelnen Elementen beschäftigen stellt sich die Frage wie aus diesen einzelnen Beschreibungen die Beschreibung einer Gruppierung entsteht. Wir wissen, dass die Elemente in einer Gruppierung eine ähnliche Absicht implementieren und somit sollte auch die Semantik ihrer

Bezeichner ähnlich sein. Da die Informationen im Quelltext eines einzelnen Programmelements meist ausreichend sind um seine Semantik zu beschreiben, nehmen wir an, dass der in einer Gruppierung enthaltene Quelltext ausreichend ist, um die Semantik der Gruppierung selbst zumindest grob zu beschreiben. Somit ist diese Form der Repräsentation sehr einfach auf Quelltextgruppierungen anzuwenden, da lediglich die Quelltexte der einzelnen Programmelemente konkateniert werden müssen. Das Ergebnis dieser Konkatenation muss nicht zwangsläufig gültiger Quelltext sein, da wir nicht daran interessiert sind diesen auszuführen. Diese Quelltexttoken können dann durch Vorverarbeitunggsschritte aufbereitet werden um ihren Informationsgehalt zu erhöhen.

Die Repräsentation von Quelltextgruppierungen durch die Token der in ihnen enthaltenen Programmelemente ist sehr flexibel. Sie ermöglicht es uns sowohl maschinelle Lernverfahren, als auch statistische Verfahren für natürlichsprachliche Dokumente anzuwenden und ist leicht nachvollziehbar und interpretierbar. Außerdem können andere Repräsentationen, wie der AST oder CFG, aus den Token der Programmelemente abgeleitet werden, was sie zu einer mächtigen Ausgangsrepräsentation macht.

5.1.5 Repräsentation durch kontinuierliche Vektoren

Maschinelle Lernmodelle wie word2vec (siehe Abschnitt 2.8.4) können eingesetzt werden um Eingabetexte in eine Vektorraumrepräsentation zu überführen. Hierzu lernt das Modell die Bedeutung der Wörter in den Texten anhand ihres Kontext. Im Falle von natürlichsprachlichen Dokumenten werden die Vektorrepräsentationen von Wörtern auch Worteinbettungen genannt. Eine solche Repräsentation ermöglicht es die Wörter eines Dokuments wie reguläre Vektoren, wie sie aus der linearen Algebra bekannt sind, zu behandeln. Beispielsweise können dann Distanz- und Ähnlichkeitsmaße, wie die Kosinusähnlichkeit, genutzt werden, um Ähnlichkeiten zwischen Wortvektoren und somit Wörtern zu messen. Mithilfe neuronaler Netzer kann auch Quelltext in eine kontinuierliche Vektorrepräsentation überführt werden. Um solche Repräsentationen von Quelltext zu erstellen können die Token eines Programmelements wie natürlichsprachliche Wörter behandelt und eingebettet werden. Die entstehenden Vektoren repräsentieren dann die einzelnen Quelltexttoken und können zur weiteren Verarbeitung durch neuronale Netze verwendet werden. Da unser Ziel jedoch die Beschreibung der Semantik von ganzen Quelltextgruppierungen ist, wären Vektorrepräsentationen für ganze Methoden oder Klassen für uns interessanter. Solche Repräsentationen können erstellt werden, indem man nicht die einzelnen Token des Quelltextelements, sondern eine Repräsentation des Quelltextelements insgesamt einbettet. Hierzu können der Kontrollflussgraph [TWB+18], der AST [AZLY19] oder, mithilfe von LSTMs oder GRUs, eine Sequenz der Token des Elements [LJM19] verwendet werden. Außerdem können mehrere dieser Ausgangsrepräsentationen zu einer einzelnen Vektorrepräsentation für ein Quelltextelement kombiniert werden. Auch hier ist es jedoch nicht trivial, wie man von diesen Repräsentationen für einzelne Quelltextelemente zu einer Repräsentation für ganze Quelltextgruppierungen gelangen soll. Eine Repräsentation durch Vektorraumeinbettungen bietet also die Möglichkeit viele verschiedene Informationen über Quelltext kompakt darzustellen. Allerdings können diese Vektoren nur durch maschinelle Lernmodelle weiterverarbeitet werden, da es nicht möglich ist die Funktion oder Semantik eines solchen Vektors, ohne das gelernte Wissen des Modells das ihn erzeugt hat, zu bestimmen. Die Repräsentation durch eingebettete Vektoren würde es uns also verwehren textbasierte Verfahren wie LDA (siehe Abschnitt 2.4.2) anzuwenden.

5.1.6 Entwurfsentscheidung: Quelltextrepräsentation

Wie die eben durchgeführte Analyse verschiedener Quelltextrepräsentationen gezeigt hat, bietet jede Repräsentation ihre eigenen Vor- und Nachteile. Ebenso eignen sich die Repräsentationen unterschiedlich gut zur Gewinnung von bestimmten Informationen aus dem

Quelltext. Sowohl der Aufrufgraph, als auch der Kontrollflussgraph eignen sich um Informationen über die Zusammenhänge und Ähnlichkeiten zwischen Quelltextelementen zu finden. Für die Findung von Gruppierungsbeschreibungen spielt diese Art von Information nur eine untergeordnete Rolle, da Ähnlichkeitsinformationen zwischen Quelltextelementen bereits beim Aufbau der Gruppierungen mit eingeflossen ist. Wir wissen also bereits, dass die Elemente einer Quelltextgruppierung zu einander ähnlich sind. Der AST kodiert in erster Linie zwar auch Beziehungen zwischen Programmelementen, da dieser allerdings eine vollständige Repräsentation von Quelltext ist, kann er auch genutzt werden um auf natürlichsprachliche Elemente wie Bezeichner zuzugreifen. Dennoch ist der größte Vorteil der Repräsentation durch ASTs die aus den strukturellen Beziehungen gewonnene Information über den Zusammenhang von Programmelementen. Durch die Baumstruktur des AST sind wir außerdem in der Wahl der Verfahren zur Weiterverarbeitung eingeschränkt, da etwa textbasierte Verfahren wie LDA nicht trivial mit ASTs funktionieren. Demnach bleibt noch die Abwägung der Repräsentationen durch die Quelltexttoken und durch eingebettete Vektoren übrig. Eingebettete Vektoren ermöglichen eine kompakte Repräsentation von Quelltext, welche jedoch für Menschen quasi unmöglich zu interpretieren ist und nur durch maschinelle Lernverfahren weiter genutzt werden kann. Existierende Verfahren [ACRC20] [ABLY19] können Methoden oder Klassen zunächst in eine Vektorrepräsentation überführen und im Anschluss natürlichsprachliche Beschreibung für deren Semantik liefern. Bisherige Arbeiten haben sich immer auf das Beschreiben der Semantik einzelner Methoden oder Klassen konzentriert. Wir sind jedoch an Beschreibungen für die geteilte Semantik mehrerer Quelltextelemente interessiert. Ein solches Modell selbst zu implementieren ist auf Grund des enormen Zeitaufwands im Rahmen dieser Arbeit nicht möglich. Die Repräsentation des Quelltextes durch seine Token hingegen ist sowohl leicht nachvollziehbar, flexibel und ermöglicht die Nutzung etablierter textbasierter Verfahren. Außerdem bietet sich diese Repräsentation an, da sie allgemein genug ist um die anderen Repräsentationen aus ihr zu gewinnen, falls sie später benötigt werden sollten. Im der weiteren Analyse gehen wir nun also davon aus, dass die Quelltextgruppierungen als Menge der Quelltexttoken der einzelnen Programmelemente vorliegen.

5.2 Beschreibungsrepräsentationen

Nachdem im vorherigen Abschnitt die verschiedenen Repräsentationsformen von Quelltext analysiert wurden, soll nun analysiert werden, welche Repräsentationen für Beschreibungen der Semantik von Quelltextgruppierungen möglich sind.

5.2.1 Natürlichsprachliche Sätze

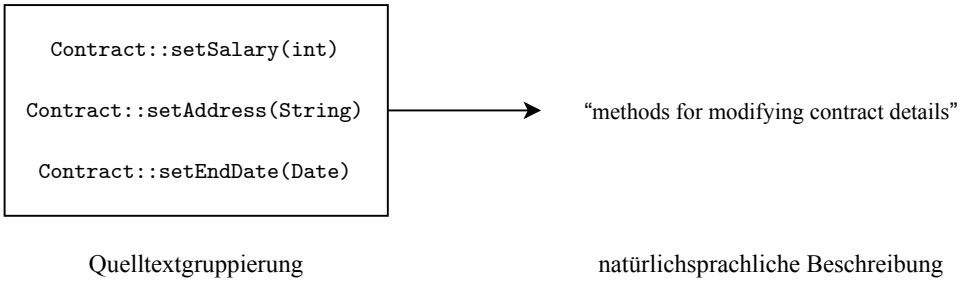


Abbildung 5.4: Schematische Darstellung einer Quelltextgruppierung und einer entsprechenden Beschreibung in natürlicher Sprache.

Beschreibungen der Semantik von Quelltextgruppierungen können als natürlichsprachliche Sätze repräsentiert werden. Diese Repräsentationsform setzt ein Verständnis der grammatikalischen Regeln der Zielsprache voraus, da sonst keine korrekten natürlichsprachlichen

Sätze gebildet werden können. Abbildung 5.4 zeigt am Beispiel einer Quelltextgruppierung wie eine Beschreibung in Form eines natürlichsprachlichen Satzes aussehen könnte.

Natürlichsprachliche Sätze werden häufig als Ausgaberepräsentation von Verfahren genutzt, welche auf Techniken der maschinellen Übersetzung, wie etwa Sequenz-zu-Sequenz- oder Transformer-Modellen, aufbauen. Die Ausgaberepräsentation durch natürlichsprachliche Sätze findet sich am häufigsten in Arbeiten, welche auf Verfahren aus der maschinellen Übersetzung aufbauen. Verfahren wie Sequenz-zu-Sequenz-Modelle oder Transformer-Modelle werden dort genutzt um Sätze aus einer Sprache in eine andere Sprache zu übersetzen. In diesem Fall handelt es sich bei Eingabe und Ausgabe um natürlichsprachliche Sätze in zwei verschiedenen Sprachen. Wählt man Quelltext als Ausgangssprache und Englisch als Zielsprache, so kann man allerdings auch Modelle trainieren, welche natürlichsprachliche Beschreibungen des Quelltextes erzeugen. Diese Verfahren benötigen große Mengen von Trainingsdaten um Quelltext adäquat und durch korrekte natürliche Sprache zu beschreiben.

5.2.2 Schlagworte

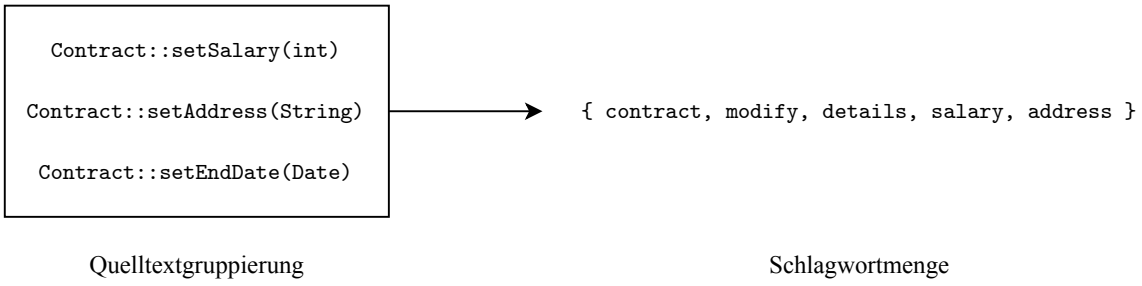


Abbildung 5.5: Schematische Darstellung einer Quelltextgruppierung und einer entsprechenden Beschreibung in natürlicher Sprache.

Eine weitere Möglichkeit Beschreibungen der Semantik von Quelltext zu repräsentieren sind Schlagwortmengen. Abbildung 5.5 zeigt eine solche Schlagwortmenge am Beispiel einer Quelltextgruppierung. Anders als natürlichsprachliche Sätze setzen sie kein Verständnis von Grammatik voraus. Das macht die Erzeugung von Schlagwortmengen um einiges einfacher, da die einzelnen Schlagwörter voneinander grammatikalisch unabhängig sind. Da Grammatik in Schlagwortmengen keine Rolle spielt, sinkt allerdings auch ihr Informationsgehalt.

LDA oder Verfahren welche darauf aufbauen und somit als Ausgabe Themen erzeugen können leicht genutzt werden um Schlagwortmengen zu bilden. Schlagwortmengen können aus diesen Themen gewonnen werden, indem eine beliebige Anzahl an Wörtern des Themas ausgewählt werden. An dieser Stelle stellt sich die Frage welche Wörter erhalten bleiben sollten. Solche Auswahlstrategien werden im späteren Verlauf der Analyse konkret betrachtet.

Schlagwortmengen können als Ausgangspunkt genutzt werden um mithilfe von Spracherzeugungsverfahren natürlichsprachliche Sätze zu konstruieren. Außerdem können sie in Kombination mit Themenbeschriftungsverfahren oder wissensbasierten Themenmodellen genutzt werden um ihren Informationsgehalt weiter zu erhöhen.

5.2.3 Entwurfsentscheidung: Beschreibungsrepräsentation

Nun muss abgewägt werden, welche dieser Repräsentationen sich besser eignet um Quelltextgruppierungen zu beschreiben. Natürlichsprachliche Sätze können auf grammatikali-

sche Konstrukte zurückgreifen, welche den Informationsgehalt von Beschreibungen erhöhen. Allerdings stellt diese Abhängigkeit auf eine korrekte Grammatik eine Herausforderung dar, da die Erzeugung grammatikalisch korrekter Sätze nicht trivial ist. Eine Repräsentation durch Schlagwortmengen ist frei von grammatikalischen Einschränkungen und Regeln oder Reihenfolgeabhängigkeiten. Da die einzelnen Wörter der Schlagwortmenge somit in erster Linie unabhängig voneinander sind, sind sie nicht so einfach für Menschen interpretierbar als natürlichsprachliche Sätze. Dennoch bietet sich diese einfache Repräsentation an, da sie leichter zu erzeugen ist als korrekte natürlichsprachliche Sätze und dennoch einen guten Informationsgehalt aufweist. Außerdem können Schlagwortmengen genutzt werden um in einem Nachbearbeitungsschritt aus ihnen natürlichsprachliche Sätze zu generieren. Durch diese Kombination aus einfacher Erzeugung und Verarbeitung, einem guten Informationsgehalt und der Möglichkeit die Schlagwortmengen weiterzuverarbeiten, werden im weiteren Verlauf der Analyse Quelltextbeschreibungen in Form von Schlagwortmengen angenommen.

5.3 Natürliche Sprache in Quelltextgruppierungen

Natürliche Sprache kommt in Quelltext (und somit in den Quelltextgruppierungen) hauptsächlich in Form von Bezeichnern, Kommentaren oder konstanten Zeichenketten vor. Allerdings machen diese natürlichsprachlichen Elementen nur einen geringen Teil der Token des Quelltextes insgesamt aus. Java-Quelltext besteht zum einem großen Teil aus Schlüsselwörtern (bspw. `class`, `interface`, `int`) und Sonderzeichen wie Klammern oder Semikola. Viele dieser Elemente werden jedoch nur benötigt um gültigen Java-Quelltext zu erzeugen und helfen nicht dabei die Semantik eines Quelltextelements oder einer Gruppierung von Quelltextelementen zu verstehen.

Rodeghero *et al.* [RMM+14] untersuchen in einer durch Blickerfassung (engl. *eye tracking*) gestützten Studie, wie Menschen Quelltext verstehen. In ihrer Studie lesen zehn professionelle Java Programmierer den Quelltext von Java Methoden und erstellen Beschreibungen für diese. Mittels Blickerfassung beobachten sie, welche Teile des Quelltextes die Programmierer besonders lange betrachten und identifizieren so Stellen im Quelltext, welche auch für die automatische Quelltextzusammenfassung interessant von besonderem Interesse sind. Insgesamt besteht der Datensatz für die Studie aus 67 zufällig ausgewählten Methoden aus mehreren Java Projekten verschiedener Domänen. Einer der Forschungsfragen von Rodeghero *et al.* ist es, ob Programmierer sich beim Verständnis einer Methode eher auf die Signatur, als den restlichen Quelltext fokussieren. In der Tat zeichnet sich über alle Teilnehmer und Methoden hinweg ein Muster ab das bestätigt, dass die Programmierer überproportional lange die Signatur einer Methode betrachten. Eine weitere Forschungsfrage beschäftigt sich mit dem Kontrollfluss von Methoden und soll zeigen, wie stark Programmierer den Kontrollfluss einer Methode für ihr Verständnis der Methode nutzen. Die Studie zeigt, dass etwa Kontrollflussschlüsselwörter durchschnittlich weniger von den Programmierern gelesen werden, als andere Token im Quelltext.

Allamanis *et al.* vergleichen im Rahmen von **„Suggesting Accurate Method and Class Names“** [ABBS15] verschiedene Informationsquellen für Typdeklarationen, Methodendeklarationen und Variablendeklarationen. Bei dem von Allamanis genutzten Modell, welches Methodennamen basierend auf dem Kontext der Methode vorschlagen soll, handelt es sich um ein neuronales Sprachmodell. Für Typdeklarationen betrachten die Autoren die Subtoken der Felder des Typs, die Subtoken der Bezeichner von Superklasse und implementierten Schnittstellen und die Subtoken der Bezeichner der Methoden des Typs. Für Methodendeklarationen werden eine Vielzahl von Informationsquellen betrachtet: Um den Kontrollfluss innerhalb einer Methode zu berücksichtigen wird die zyklomatische Komplexität (eine Metrik der Komplexität eines Programms, basierend auf der Anzahl an unabhängigen Pfaden

durch den Quelltext des Programms) der Methode bestimmt und als Eingabe an das neuronale Sprachmodell hinzugefügt. Des weiteren fließen AST-Abhängigkeiten, Subtoken der Felder der umschließenden Klasse, der Superklassen, der Methodenimplementierung selbst, sowie Deklarationsmodifikatoren (bspw. `static` oder `private`), Rückgabetyp, Geschwistermethoden, Anzahl an Parametern und deklarierte Ausnahmen in das Sprachmodell mit ein. Die Autoren identifizieren für ihr Modell die Subtoken der Bezeichner der Superklassen einer Klasse, den Rückgabetyp einer Methode und die Subtoken der Felder einer Klasse als wichtigste Informationsquellen.

Diese Ergebnisse bestätigen, dass nicht alle Quelltexttoken den gleichen semantischen Wert besitzen und somit nicht alle Quelltexttoken für die Erzeugen von Gruppierungsbeschreibungen betrachtet werden müssen.

Im Folgenden wird daher zunächst analysiert, welche Stellen im Quelltext von Methoden und Klassen natürliche Sprache beinhalten. Dabei wird untersucht, wie die natürlichsprachlichen Informationen an diesen Stellen genutzt werden können um die Semantik von Quelltextelementen und Quelltextgruppierungen zu verstehen. Zunächst werden Elemente betrachtet welche sowohl im Kontext von Methodenelementen, als auch Klassenelementen auftreten und zur Gewinnung von natürlichsprachlichen Inhalten aus dem Quelltext genutzt werden können. Danach werden spezielle Informationsquellen für Klassenelemente und Methodenelemente betrachtet.

5.3.1 Geteilte Quellen natürlicher Sprache

Im Folgenden werden Informationsquellen betrachtet, die sowohl für Methoden als auch für Klassen vorliegen. Diese stellen für beide Arten von Elementen wichtige Quellen natürlicher Sprache dar.

5.3.1.1 Bezeichner

Bezeichner werden von Entwicklern bewusst eingesetzt um semantische Informationen über die Funktion eines Programmelements im Quelltext zu kodieren. Klassen, Methoden und Variablen tragen Bezeichner, welche die Quelltextelemente identifizieren. Somit treten Bezeichner an unterschiedlichen Stellen des Quelltextes eines Programmelements auf, diese speziellen Quelltextstellen und ihre Bedeutung für das Verständnis der Semantik von Quelltext werden im weiteren Verlauf genauer betrachtet. Das konkrete Format eines Bezeichners ist abhängig von den Konventionen der Programmiersprache des zu analysierenden Projekts. Bezeichner in Java folgen der Binnenmajuskelschreibweise (engl. *camel case*). Dies bedeutet, dass mehrere Wörter groß geschrieben aneinandergereiht werden um sie zu einem Bezeichner zu kombinieren. Zu beachten ist hierbei, dass der erste Buchstabe von Klassen- und Schnittstellenbezeichnern immer groß geschrieben wird (bspw. `EmployeeContractService`), während der erste Buchstabe von Methoden- oder Variablenbezeichnern immer klein geschrieben wird (bspw. `getContractByID(int id)`). Durch Kombination mehrerer Wörter zu einem Bezeichner lässt sich sein semantischer Informationsgehalt erhöhen. Die Bezeichner in Binnenmajuskelschreibweise müssen in einem späteren Vorverarbeitungsschritt aufgespalten werden um die einzelnen Token zu gewinnen, welche den Bezeichner ausmachen.

5.3.1.2 Kommentare

```
/**
*  This class represents an identifiable employee in a company
*
*  @author Timo
*  @version 1
*/
public class Employee {
    private long id;

    /**
    *  Gets this employee's ID
    *
    *  @return ID of this employee
    */
    public long getID() { ... }

    /**
    *  Sets this employee's ID
    *
    *  @param new ID for this employee
    */
    public long setID(long id) { ... }

}
```

Quelltextausschnitt 5.1: Beispiel von Javadoc-Kommentaren anhand einer Java-Klasse `Employee` und ihrer Methoden

Quelltextkommentare werden von Entwicklern genutzt um Quelltextelemente zu Dokumentieren und ein späteres Verständnis des Quelltextes zu erleichtern. Kommentare können an beliebigen Stellen im Quelltext auftreten um ihn zu Dokumentieren. Java unterscheidet zwischen zwei Arten von Kommentaren:

Normale Kommentare können an beliebigen Stellen im Quelltext platziert werden, um ihn zu beschreiben. Einzeilige Kommentare werden mit zwei Schrägstrichen (`//`) eingeleitet und markieren den Rest der Quelltextzeile als Kommentar. Zeilenüberspannende Kommentare werden mit einem Schrägstrich und einem Stern (`/*`) eingeleitet und umgekehrt (`*/`) terminiert. Dies markiert sämtlichen Text zwischen der Start- und Endmarkierung als Kommentar. Javadoc-Kommentare sind eine spezielle Form von Kommentaren, welche mithilfe des Javadoc Werkzeugs zu HTML Dokumentationsdateien umgewandelt werden können. Diese Kommenatare werden mit einem Schrägstrich und zwei Sternen (`/**`) eingeleitet und mit einem Stern und einem Schrägstrich terminiert (`*/`). Javadoc-Kommentare dürfen unmittelbar vor Klassen-, Feld-, Methoden- oder Konstruktordefinitionen platziert werden. Üblicherweise beginnen Javadoc-Kommentare mit einer natürlichsprachlichen Beschreibung des annotierten Quelltextelements. Diese Beschreibung darf unter anderem HTML-Elemente oder Verweise auf andere Quelltextelemente enthalten. Der zweite Teil eines Javadoc-Kommentars enthält sogenannte Javadoc-Markierungen (engl. *Javadoc-Tags*). Diese Markierungen können genutzt werden, um Metadaten wie Autor oder Version des Quelltextelements festzuhalten. Außerdem sind spezielle Markierungen vorhanden, welche genutzt werden können um die Parameter, den Rückgabetyp oder mögliche Ausnahmen des annotierten Quelltextelements natürlichsprachlich zu beschreiben. Ein Beispiel von Javadoc-Kommentaren in Quelltext ist in Quelltextausschnitt 5.1 dargestellt. Sowohl die Klasse, als auch ihre Methoden besitzen Javadoc-Kommentare. Im Kommentar der Klasse `Employee` selbst sind neben einer Beschreibung ihrer Absicht noch Markierungen enthalten um Metadaten über ihren Autor und ihre Version festzuhalten. Die Kommentare der Me-

thoden enthalten Markierungen um den Rückgabewert und die Parameter der Methoden zu beschreiben. Zu beachten ist, dass Quelltextkommentare und Javadoc-Kommentare optional sind und somit nicht in jedem Softwareprojekt vorhanden sind. Das Vorhandensein von deskriptiven und semantisch sinnvollen Quelltextkommentaren ist ein Gütemerkmal für die Qualität von Quelltext. Da hochqualitative Quelltextkommentare eine große Menge an natürlichsprachlichen Informationen über die Semantik des kommentierten Elements beinhalten, stellen sie für uns eine wichtige Informationsquelle dar.

5.3.2 Quellen natürlicher Sprache für Klassen

```
package com.company.models;

public class Employee extends AbstractEmployee {
    private long employeeId;

    public Employee(long employeeId) {
        this.employeeId = employeeId;
    }

    public long getEmployeeId() {
        return this.employeeId;
    }
}
```

Quelltextausschnitt 5.2: Eine fiktive Java-Klasse, welche einen Mitarbeiter in einer Firma repräsentiert.

Bestimmte Quellen natürlicher Sprache sind nur für Klassen definiert und können somit nur auf Klassenelemente angewandt werden. Dieser Abschnitt gibt einen Überblick über solche Informationsquellen. Da ein beachtlicher Teil der natürlichen Sprache in Quelltext von Bezeichnern stammt, stellen handelt es sich bei vielen dieser Informationsquellen um spezielle Instanzen von Bezeichnern. Quelltextausschnitt 5.2 zeigt ein Beispiel einer Java-Klasse `Employee`. Natürliche Sprache kommt etwa in der Paketdeklaration, dem Bezeichner der Methode oder dem Bezeichner ihrer Oberklasse vor. Diese Quellen natürlicher Sprache werden im Folgenden genauer betrachtet.

5.3.2.1 Superklasse und implementierte Schnittstellen

Java Klassen können exakt eine Superklasse erweitern und beliebig viele Schnittstellen implementieren. Die Bezeichner der Superklasse und der implementierten Schnittstellen einer Klasse helfen es Programmierern das Verhalten der Klasse nachzuvollziehen. Implementiert eine Klasse die Java Standardschnittstelle `List<T>`, erkennt man direkt an der Klassendefinition, dass sich auch die implementierende Klasse wie eine Liste verhält. Für die Semantik der Klasse bedeutet dies, dass jede Instanz der Klasse auch eine Instanz von `List<T>` ist und das von `List<T>` definierte Verhalten implementiert. Die Unterklasse erfüllt somit außerdem die Bedingungen des liskovschen Substitutionsprinzips und eine Instanz der Klasse kann an jeder Stelle genutzt werden, wo eine Instanz von `List<T>` erwartet wird. Die Bezeichner der Supertypen erlauben es also nachzuvollziehen welche Absichten von einer Klasse implementiert werden. Im Bezug auf Quelltextgruppierungen bedeutet das, dass die Bezeichner der Supertypen enthaltener Klassen genutzt werden können um geteilte Informationen über die Semantik der Elemente zu gewinnen. Angenommen eine Quelltextgruppierung enthält nur Klassenelemente und diese implementieren alle dieselbe Schnittstelle. Somit implementieren alle Elemente die von der Schnittstelle vorgegebene Absicht, wodurch sich dann auch die Absicht der Gruppierung insgesamt charakterisieren lässt. Beispiel 5.1 zeigt diesen Zusammenhang anhand einer fiktiven Quelltextgruppierung.

Beispiel 5.1: Superklasse

Gegeben sei eine Quelltextgruppierung G, welche die beiden Programmelemente `class User extends DatabaseEntity` und `class Document extends DatabaseEntity` enthält. Anhand der gemeinsamen Oberklasse `DatabaseEntity` kann man erkennen, dass es sich bei beiden Klassen um Entitäten einer Datenbank handelt. Da die Gruppierung lediglich aus diesen zwei Elementen besteht, hat die Semantik der Gruppierung also sehr wahrscheinlich etwas mit Datenbankentitäten zu tun.

5.3.2.2 Felder

Die Felder (auch Attribute genannt) einer Klasse definieren ihren Zustand. Die Definition eines Feldes innerhalb einer Klasse besteht aus drei Teilen. Der erste Teil der Definition ist der Typ des Feldes, er legt fest welche Objekte oder Primitive in dem Feld gespeichert werden können. Darauf folgt ein vom Programmierer festgelegter Bezeichner. Ähnlich wie die Bezeichner von Klassen oder Methoden selbst werden Feldbezeichner in hochqualitativen Softwareprojekten semantisch sinnvoll gewählt. Optional kann ein Feld mit einem Ausdruck initialisiert werden. Von diesen drei Teilen ist der Bezeichner von größtem Interesse, da er zum einen Teil einer jeden Felddefinition ist und zum anderen meist aus natürlichsprachlichen Wörtern oder Abkürzungen solcher besteht. Klassen, welche eine ähnliche Absicht implementieren nutzen dazu häufig ähnliche Daten. Diese Daten werden in den Feldern der Klasse gespeichert. Häufig tragen Felder, welche in einem ähnlichen Kontext verwendet werden, ähnliche Bezeichner. Für die Quelltextgruppierungen bedeutet das, dass die Bezeichner der Felder von Klassen Hinweise auf deren geteilte Semantik liefern können. Beispiel 5.2 zeigt diesen Zusammenhang anhand einer fiktiven Quelltextgruppierung.

Beispiel 5.2: Felder

Gegeben sei eine Quelltextgruppierung G welche mehrere Klassenelemente enthält. Alle dieser Klassenelemente definieren ein Feld *privateLoggerlogger*;. Dieses Feld ist das einzige das von allen Klassenelementen definiert wird, die Bezeichner keiner anderen Felder stimmen überein. Die Übereinstimmung dieses Feldes deutet darauf hin, dass die Gruppierung die Absicht „Logging“ abbildet.

5.3.2.3 Konstruktoren

Konstruktoren sind spezielle Methoden, welche genutzt werden um Klassen zu instanziieren. Java-Klassen besitzen immer einen Standardkonstruktor. Dieser instanziiert die Klasse, jedoch ohne ihre Felder zu initialisieren oder weitere Logik auszuführen. Programmierer können weitere Konstruktoren hinzufügen um ihre Klasse zu initialisieren. Meist werden Konstruktorparameter definiert, welche dann innerhalb des Konstruktors genutzt werden um Felder der Klasse zu initialisieren. Diese im Konstruktor initialisierten Felder spielen meist eine wichtige Rolle für die Funktionalität der Klasse. Somit können die Parameter der Klassenkonstruktoren genutzt werden, um wichtige Felder der Klasse zu identifizieren. Da Klassen, welche eine ähnliche Absicht implementieren auch oft ähnliche Konstruktorsignaturen aufweisen, können ihre Parameter oder Javadoc-Kommentare genutzt werden um nachzuvollziehen welche Daten für die Semantik der Klassen wichtig sind und weshalb. Beispiel 5.3 zeigt diesen Zusammenhang anhand einer fiktiven Quelltextgruppierung.

Beispiel 5.3: Konstruktoren

Gegeben sei eine Quelltextgruppierung G, welche mehrere Klassen enthält. Jede dieser Klassen definiert einen Konstruktor, welcher einen Parameter `DatabaseConnection databaseConnection` übergeben bekommt. Die Semantik der Gruppierung hat somit vermutlich etwas mit Datenbankverbindungen zu tun.

5.3.2.4 Pakethierarchie

Pakete sind ineinander geschachtelte Ordner, welche genutzt werden, um Klassen bezüglich ihrer Funktionalität zu gruppieren. Eine gut gepflegte Pakethierarchie ist ein weiteres Zeichen für hochqualitativen Quelltext. Die Namen dieser Ordner werden durch Punkte konkateniert und treten etwa in vollqualifizierten Bezeichnern oder Import-Ausdrücken auf. Je weiter man die Pakethierarchie zur Wurzel hin traversiert, desto allgemeiner wird die Funktionalität der darin enthaltenen Pakete und Klassen. Deshalb ist das Paket welches eine Klasse umschließt lokal von größtem Interesse. Ist bekannt, dass mehrere Klassen in einer Quelltextgruppierung dem gleichen Paket entstammen, so kann daraus abgeleitet werden, dass diese eine Absicht verfolgen. Der Bezeichner des Pakets liefert dann eine kurze Beschreibung dieser Absicht nach der sie gruppiert wurden. Beispiel 5.4 zeigt diesen Zusammenhang anhand einer fiktiven Quelltextgruppierung.

Beispiel 5.4: Pakete

Gegeben sei eine Quelltextgruppierung G, welche mehrere Klassen enthält. Alle dieser Klassen entstammen dem `com.company.entities` Paket. Die Semantik der Gruppierung hat somit vermutlich etwas mit Datenbankentitäten zu tun.

5.3.3 Quellen natürlicher Sprache für Methoden

```java
/**
*  Calculates the salary for an employee
*
*  @param the employee whose salary to calculate
*  @return the calculated salary
*/
public int calculateEmployeeSalary(Employee employee) {
    int salary;

    if (employee.isStudent()) {
        salary = 0;
    } else {
        salary = 5000;
    }

    return salary;
}
```

Quelltextausschnitt 5.3: Beispiel einer Java-Methode

Java-Methoden können grob durch drei Bestandteile beschrieben werden. Ein optionaler Quelltextkommentar kann, wie bereits eingeführt, an die Methode angeheftet werden um ihre Funktionalität zu beschreiben. Danach folgt die Signatur der Methode, diese beinhaltet den Zugriffsmodifikator, den Rückgabetyp, den Bezeichner und die Parameter der Methode. Der Rumpf der Methode befindet sich zwischen den zwei geschwungenen Klammern

nach der Signatur einer Methode. Er beinhaltet die Anweisungen der Methode, welche genutzt werden um ihre Funktionalität zu implementieren. Außerdem finden sich in seinem inneren Kontrollflussstrukturen wie Schleifen oder Verzweigungen. Quelltextausschnitt 5.1 zeigt ein Beispiel einer Java-Methode `calculateEmployeeSalary`, welche den monatlichen Lohn eines Mitarbeiters in einer Firma berechnet. Im folgenden werden die Signatur und der Rumpf von Methoden genauer betrachtet und es wird analysiert, welche Informationen aus ihnen gewonnen werden können und inwiefern diese hilfreich sind um die Semantik von einzelnen Methoden und Quelltextgruppierungen insgesamt zu verstehen.

5.3.3.1 Rückgabetyp

Der Rückgabetyp einer Methode gibt Programmierern einen Hinweis darauf ob und welche Objekte eine Methode zurückgeben kann. Allerdings reicht er alleine nicht aus, um die Semantik einer Methode zu verstehen, da er lediglich Aussagen über die letzte Anweisung einer Methode, der Rückgabeanweisung, macht. Gibt eine Methode ein Benutzerdefiniertes Objekt zurück, so kann der Bezeichner dieses Typen betrachtet werden um nachzuvollziehen, welche Semantik das zurückgegebene Objekt besitzt, jedoch nicht was die Semantik der Methode ist die es zurückgibt. Sollte der Rückgabetyp einer Methode ein primitiver Typ sein, macht es wenig Sinn ihn zu betrachten, da die Bezeichner von primitiven Typen keinen Hinweis auf eine spezielle Semantik zulassen. Lediglich im Falle von Zugriffsmethoden haben sie direkt etwas mit der Semantik der Methode zu tun. Allerdings wird diese Semantik auch meist im Bezeichner der Methode festgehalten. Beispiel 5.5 zeigt diesen Zusammenhang anhand einer fiktiven Quelltextgruppierung.

Beispiel 5.5: Rückgabetyp

Gegeben sei eine Quelltextgruppierung G, welche zwei Methoden enthält. Der Rückgabetyp beider Methoden ist `int`. Alleine anhand der Rückgabetypen lässt sich kein Schluss auf die Semantik der Methoden ziehen. Angenommen die Bezeichner der Methoden sind `getEmployeeSalary` und `getInternSalary` und sind ebenfalls bekannt. Durch die Bezeichner wird deutlich, dass die Semantik der Gruppierung mit dem Lohn von Mitarbeitern zu tun hat.

5.3.3.2 Parameter

Die Parameter von Java-Methoden bestehen aus Paaren von jeweils einem Typ und einem Bezeichner. Der Typ gibt an welche Objekte der Methode übergeben werden können, während der Bezeichner lediglich während der Entwicklung eine Rolle spielt, um die Parameter zu unterscheiden und ihre semantische Rolle für die Methode festzuhalten. Die Entwicklungszeit eines Projekts ist jedoch genau die Zeit, in der Semantik und ein einfaches Verständnis der Semantik von Quelltextelementen die größte Rolle spielt. Somit werden Bezeichner von Parametern mit einer möglichst hohen Aussagekraft gewählt. Aus diesem Grund sind die Bezeichner der Parameter einer Methode von größerem Interesse als ihre Typen. Die Parameter von Methoden, und besonders ihre Bezeichner, ermöglichen es uns den Zweck eines Objektes für die Semantik der Methode nachzuvollziehen. Mehrere Methoden, deren Parameter ähnliche Bezeichner aufweisen, verfolgen somit auch häufig eine ähnliche Absicht. Dies kann zur Findung einer Beschreibung ihrer geteilten Semantik hilfreich sein. Beispiel 5.6 zeigt diesen Zusammenhang anhand einer fiktiven Quelltextgruppierung.

Beispiel 5.6: Parameter

Gegeben sei eine Quelltextgruppierung G, welche zwei Methoden enthält. Beide Methoden definieren einen Parameter `Contract contractToUpdate`. Anhand des Bezeichners der Parameter kann man erkennen, dass die Absicht beider Methoden das aktualisieren eines Vertrags ist.

5.3.3.3 Anweisungen

Ein großer Teil des Quelltextes im Rumpf von Methoden besteht aus Anweisungen. Die Anweisungen im inneren einer Methode geben an, wie ihre Funktionalität realisiert wird. Diese Implementierungsnähe macht sie allerdings für unseren Zweck der Findung von Beschreibungen von Semantik eher uninteressant. Der Kontrollfluss innerhalb der Anweisungen einer Methode kodiert Informationen darüber, wie Daten fließen und wie verschiedene Anweisungen interagieren um eine bestimmte Semantik zu implementieren. Für die Beschreibung geteilter Semantik ist diese Information allerdings nicht sehr interessant, da wir viel mehr daran interessiert sind welche Daten durch den Rumpf einer Methode fließen und wie sich diese Interaktion konkret ausprägt. Solche Interaktionen geschehen beispielsweise durch Methodenaufrufe. Anhand dieser Methodenaufrufe können Methoden identifiziert werden, welche relevant für die Semantik der vorliegenden Methode sind. Da eine solche Analyse jedoch bereits in den Aufbau der semantischen Quelltextrepräsentation von Eurich [Eur] eingeflossen ist, wissen wir, dass die Quelltextgruppierungen bereits zusammengehörige Methoden beinhalten. Eine Betrachtung der Bezeichner, etwa von Variablendefinitionen, im Rumpf einer Methode kann hilfreich sein um gemeinsame natürlichsprachliche Informationen von Quelltextgruppierungen zu erhalten. Allerdings macht es keinen Sinn alle Anweisungen aus den Methodenrümpfen zu betrachten, da sie nicht zwangsläufig dazu beitragen die Semantik einer Methode zu verstehen und im schlimmsten Fall durch das Hinzufügen von Rauschen andere Informationsquellen negativ beeinflussen können. Beispiel 5.7 zeigt diesen Zusammenhang anhand einer fiktiven Quelltextgruppierung.

Beispiel 5.7: Anweisungen

Gegeben sei eine Quelltextgruppierung G, welche mehrere Methodenelemente enthält. Alle diese Methoden definieren in ihrem Rumpf exakt eine Variable `Contract newContract;`. Somit hat die Semantik der Gruppierung vermutlich etwas mit dem Anlegen neuer Verträge zu tun.

5.3.3.4 Überschriebene Methoden und Annotationen

Java-Klassen können nicht-finale Methoden aus ihren Superklassen überschreiben. Die überschriebene Methode kann entweder abstrakt, Teil einer Schnittstelle oder eine konkrete Methode einer Klasse sein. Der Bezeichner einer überschriebenen Methode ist identisch mit dem der zu überschreibenden Methode. Allerdings können sich die JavaDoc-Kommentare oder im Falle einer konkreten Methode die Anweisungen in ihrem Inneren unterscheiden. Häufig werden abstrakte Methoden oder Methodendefinitionen in Schnittstellen dokumentiert, um es Programmierern leichter zu machen sie korrekt zu implementieren. Die konkreten Implementierungen stellen jedoch häufig Spezialisierungen der überschriebenen Methoden dar und somit können auch die Quelltextkommentare der beiden Elemente unterschiedliche Semantiken beschreiben. Überschriebene Methoden aus den Supertypen sollten daher nur als Informationsquelle mit einbezogen werden, sofern sie selbst Teil der Quelltextgruppierung sind. Methoden, welche Methoden aus Supertypen überschreiben,

werden in Java mit einer `@Override` Annotation versehen. Diese signalisiert dem Leser des Quelltextes, dass diese Methode eine das Verhalten einer Oberklassenmethode überschreibt und spezialisiert. Die `@Override` Annotation ist nicht sehr interessant, da es für das Verständnis der Methode selbst egal ist, ob es sie eine Oberklassenmethode überschreibt oder nicht. Allerdings kann Java-Quelltext auch Nutzerdefinierte Annotationen enthalten. Diese werden genutzt, um Metadaten an Quelltextelemente anzuheften. Im Allgemeinen sind solche Annotationen nur von geringem Interesse, da sie lediglich Aufschluss über Metadaten der Methode geben aber nicht über die Semantik der Methode selbst.

5.4 Vorverarbeitung von natürlicher Sprache

Die Vorverarbeitung von natürlichsprachlichen Dokumenten dient der Steigerung des aus ihnen gewonnenen Informationsgehalts. Ein Vorverarbeitungsschritt stellt dabei einen unabhängigen Teil der Vorverarbeitung eines Dokuments dar. Vorverarbeitungsschritte können grob in zwei Klassen eingeteilt werden. Die erste Klasse entfernt Token aus Dokumenten, während die zweite Klasse Token in andere Token transformiert. Beide Arten von Vorverarbeitungsschritten verfolgen das Ziel die Informationsgewinnung aus Dokumenten zu maximieren, gehen dabei allerdings unterschiedlich vor.

5.4.1 Entfernende Vorverarbeitungsschritte

Die erste Klasse von Vorverarbeitungsschritten entfernt Token mit geringem Informationsgehalt aus Dokumenten, wodurch das Vokabular der Dokumente kleiner wird. Eine solche Vorverarbeitung ist sinnvoll, da Token mit geringem semantischen Wert aus dem Vokabular entfernt werden und der durchschnittliche Informationsgehalt in den Dokumenten somit steigt. Im Folgenden werden einige Vorverarbeitungsschritte, welche dieser Klasse zuzordnen sind genauer vorgestellt.

5.4.1.1 Entfernen von Paketpfaden

Typen liegen in der Quelltextrepräsentation von INDIRECT immer mit einem vollqualifiziertem Bezeichner vor. Dieser enthält neben dem Typen selbst auch den Paketpfad, welcher zu diesem Typ führt. Wie bereits eingeführt, stellen Pakete nahe der Wurzel keine wichtige Informationsquelle dar, da die Semantik der Pakete zur Wurzel hin immer allgemeiner wird. Somit ist es sinnvoll Paketpfade zu entfernen, da diese durch eine große Menge an wiederholten Token das Verständnis der Semantik von Quelltext erschweren. Beispiel 5.8 zeigt auf, warum das Entfernen von Paketpfaden sinnvoll ist.

Beispiel 5.8: Entfernen von Paketpfaden

Eine Klasse `DatabaseConnection` liegt im Paket `com.company.util`. Jedes Vorkommnis dieser Klasse im Quelltext enthält dann auch den Paketpfad `com.company.util`. Dadurch nehmen die Token `com`, `company` und `util` einen nicht vernachlässigbaren Anteil des Quelltextes an. Da Top-Level-Domänen wie `com` häufig als Wurzel von Paketpfaden genutzt werden, tritt dieses Verhalten auch bei der Nutzung fremder Typen auf.

5.4.1.2 Entfernen von HTML und Markierungen in Javadoc

Programmierer nutzen HTML und spezielle Markierungen um ihre Javadoc-Kommentare aufzuwerten und sie leichter verständlich zu machen. Diese Aufwertungen sind meist nicht von inhaltlicher Natur, sondern fügen Metadaten über das Quelltextelement hinzu oder

verbessern die Stilisierung innerhalb des Kommentars. Somit können sie zwar den Lesefluss von Programmierern unterstützen, zur maschinellen Analyse sind sie allerdings nicht hilfreich, da sie keine semantischen Informationen kodieren und zur einer Steigerung des Rauschens in den Ausgangsdaten führen. Aus diesem Grund werden Javadoc-Kommentare von HTML und Javadoc-Markierungen bereinigt, bevor ihre Token weiter verarbeitet werden. Beispiel 5.9 zeigt am Beispiel von HTML-Markierungen, weshalb Javadoc-Kommentare im Rahmen der Vorverarbeitung bereinigt werden sollten.

Beispiel 5.9: Entfernen von HTML

Gegeben sei ein Ausschnit aus der Beschreibung eines Javadoc-Kommentars: `This class is <b>very<\b> important`. Hier werden HTML-Markierungen verwendet, welche dem Nutzer verdeutlichen sollen, dass es sich um eine **sehr** wichtige Klasse handelt. Für die maschinelle Verarbeitung sind diese Markierungen jedoch hinderlich, da sie wie der Rest der Beschreibung als Token aufgefasst werden, jedoch alleinstehend nicht zur Semantik beitragen.

5.4.1.3 Entfernen von Sonderzeichen

Quelltext enthält eine große Menge von Sonderzeichen. Jede Anweisung in Java-Quelltext wird mit einem Semikolon beendet, Parameter von Methoden werden von Klammern umschlossen und durch Kommas getrennt. All diese Sonderzeichen sind wichtige strukturelle Elemente von Quelltext, welche es einem Programmierer leichter machen Teile von Quelltext abzugrenzen und zu unterscheiden. Einige dieser Sonderzeichen, wie Semikolons oder Kommas, tragen keinen Beitrag zur Semantik von Quelltext. Andere jedoch, wie etwa mathematische Operationen, sind durchaus wichtig für die Semantik eines Quelltextelements. Diese Sonderzeichen sind allerdings sehr implementierungsnah und sind für uns nicht interessant. Zur Beschreibung von Semantik ist viel mehr ein abstrakteres Verständnis der Funktion des Quelltextes notwendig. Beispiel 5.10 zeigt, dass Sonderzeichen keinen Beitrag zur Semantik von Quelltext leisten und somit entfernt werden können.

Beispiel 5.10: Entfernen von Sonderzeichen

Gegeben sei die Methodensignatur `public void setEmployeeSalary(int salary)`. Die Klammern, welche die Parameter umschließen, sind offensichtlich nicht für die Semantik der entsprechenden Methode relevant. Bei ihnen handelt es sich lediglich um syntaktische Elemente der Java-Programmiersprache, welche keinen Einfluss auf die Semantik der Methode haben.

5.4.1.4 Längenfilter

Kurze Wörter tragen im Allgemeinen einen sehr geringen Anteil an Informationen. Beispielsweise stellt man fest, dass fast alle Wörter von Länge 2 Präpositionen oder Verbindungswörter sind, welche quasi keinen Einfluss auf die Semantik von natürlicher Sprache haben. Wörter von Länge 3 hingegen können Verben (bspw. *get, set*) oder Substantive (bspw. *set*) sein. Somit können Wörter von Länge 2 und kleiner direkt entfernt werden, da sie im Allgemeinen keinen Einfluss auf die Semantik von Text und Quelltext haben.

5.4.1.5 Stoppwortentfernung

Viele Wörter, deren Länge größer als 2 ist, tragen trotzdem quasi keinen Beitrag zur Semantik von natürlicher Sprache. Diese Wörter werden häufig in Listen gesammelt, welche

dann in ihrer Gesamtheit entfernt werden. Es gibt verschiedene solcher Listen, viele Wörter überschneiden sich jedoch zwischen ihnen. Manuelles betrachten solcher Stoppwortlisten zeigt, dass Wörter, welche in der natürlichen Sprache nur einen geringen semantischen Wert haben, in Quelltext von großer Bedeutung sein können. Dazu zählen unter anderem die Wörter „get“ und „set“, welche in Zugriffsmethoden auftreten, oder das Wort „value“. Diese Listen müssen also manuell überprüft werden, dass nicht zu viele Token entfernt werden, die für die Semantik des Quelltextes eine Rolle spielen. Schlüsselwörter von Programmiersprachen können hingegen unbedenklich entfernt werden, da es sich bei ihnen um Teile der Syntax der Programmiersprache handelt und diese keinen Einfluss auf die Semantik von Quelltext hat. Hierzu zählen auch Token, welche in einer Programmiersprache auf Grund von Konventionen häufig als Präfix oder Suffix von Bezeichnern verwendet werden. Im Fall von Java-Quelltext ist ein solches Token „Bean“, worauf die Bezeichner vieler Typen in Java-Projekten enden.

5.4.2 Transformierende Vorverarbeitungsschritte

Die zweite Art von Vorverarbeitungsschritten transformiert ein Token in ein anderes oder mehrere andere Token. Eine solche Vorverarbeitung ist sinnvoll, da oft mehrere Wörter eine ähnliche oder gleiche Semantik besitzen. Durch Transformation dieser Wörter können sie durch ein einzelnes Wort ersetzt werden, das dennoch die gleiche Semantik besitzt. Solche Vorverarbeitungsschritte verringern ebenfalls die Größe des genutzten Vokabulars und wirken sich somit positiv auf das Rauschen in den Daten aus. Anders als die zuvor eingeführten Vorverarbeitungsschritte beschäftigen sie sich jedoch mit der Aufbereitung wichtiger Token und nicht mit dem Entfernen unwichtiger Token.

5.4.2.1 Spalten von Bezeichnern (Subworttokenisierung)

Java-Bezeichner bestehen häufig aus mehreren konkatenierten natürlichsprachlichen Wörtern oder Abkürzungen. Gemäß der Java-Konvention werden solche zusammengesetzten Bezeichner an den Wortgrenzen durch Großschreibung getrennt (Binnenmajuskelschreibweise). Demnach können Bezeichner an diesen Grenzen auch wieder gespalten werden. Eine solches Aufspalten der Bezeichner ist sinnvoll, da aus ihnen Wörter gewonnen werden können, die wichtig für die Semantik einer Gruppierung sind, aber nicht einzeln in ihr vorkommen. Zu beachten ist, dass Abkürzungen, also mehrere aufeinanderfolgenden Großbuchstaben nicht zu einzelnen Buchstaben aufgespalten werden, sondern als solche erhalten bleiben. Abkürzungsauflösung ist Thema vieler Arbeiten [PP17] [CAHV15] und im Allgemeinen schwer zu lösen. Für unsere Zwecke ist es ausreichend Abkürzungen als solche zu erhalten und als eigenständige Wörter zu betrachten. Beispiel 5.11 zeigt die Subworttokenisierung anhand eines Java-Bezeichners.

Beispiel 5.11: Subworttokenisierung

Gegeben sei ein Token `EmployeeContractDAO`. Das Token wird an den Großschreibungsgrenzen aufgespalten und man erhält die Token `Employee`, `Contract` und `DAO`. Da DAO eine Abkürzung ist bleibt sie erhalten.

5.4.2.2 Lemmatisierung und Stammformreduktion

Sowohl die Lemmatisierung, als auch die Stammformreduktion, werden häufig genutzt, um viele verschiedene Wortformen auf wenige Stammformen zu reduzieren. Wie auch die Entfernung von Stoppwörtern und Sonderzeichen dienen diese Verfahren der Reduzierung der Anzahl an verschiedenen Wörtern im Quelltextvokabular und somit einer Verringerung

von Rauschen. Beide Verfahren gehen dabei grundlegen verschieden vor. Die Stammformreduktion ist eine Folge von angewandten Regeln, welche die Endung eines Token nach und nach verkürzen, bis keine Regel mehr anwendbar ist. Da sie nur auf Regeln basiert, ist die Stammformreduktion immer und auch auf einzelne Token anwendbar, die entstehenden Token sind jedoch nicht immer natürlichsprachliche Wörter. Die Lemmatisierung ist eine Alternative zur Stammformreduktion. Sie nutzt ein Wörterbuch, um Wortformen auf einen grammatikalisch korrekten Wortstamm abzubilden. Allerdings benötigt die Lemmatisierung dafür im allgemeinen ganze natürlichsprachliche Sätze, da sie von Wortartmarkierungen abhängig ist um die richtige Stammform für ein Token zu wählen. Da ganze natürlichsprachliche Sätze grundsätzlich nur in Kommentaren auftreten ist unklar, ob die Lemmatisierung auch auf einzelne Token angewandt werden kann. Besonders im Englischen existieren viele Wörter, welche je nach ihrem Verwendungskontext verschiedene Wortformen annehmen können (bspw. „set“ als Substantiv „Menge“ und als Verb „setzen“). Diese Mehrdeutigkeit erschwert es einzelne Token korrekt mit ihrer Wortart zu markieren. Da die Lemmatisierung darüber hinaus auf einem Wörterbuch basiert, ist es nicht unwahrscheinlich, dass Quelltext Token enthält, welche zu domänenspezifisch oder Wortneuschöpfungen durch Programmierer sind und nicht in dem genutzten Wörterbuch vorhanden sind. Eine Mischung beider Verfahren, Lemmatisierung für Kommentare und Stammformreduktion für andere Token, erscheint nicht sinnvoll, da dies im schlimmsten Fall die Anzahl der Wörter im Vokabular des Modells erhöht, statt sie zu verringern, da beide Verfahren verschiedene Stammformen erzeugen. Beispiel 5.12 zeigt die Unterschiede zwischen der Stammformreduktion und der Lemmatisierung anhand eines Beispiels.

Beispiel 5.12: Stammformen

Gegeben seien die Wörter „competing“ und „computation“. Ein regelbasiertes Verfahren zur Stammformreduktion könnte beiden Wörtern den Stamm „comp“ zuordnen, obwohl ihre Semantik unterschiedlich ist. Ein Lemmatisierungsverfahren hingegen könnte „competing“ auf „compete“ und „computation“ auf „compute“ abbilden, wodurch deren unterschiedliche Semantik erhalten bleibt.

5.4.2.3 Abbildung auf Kleinbuchstaben

Für die Semantik eines Textes ist es unwichtig, welcher Groß- und Kleinschreibung die Wörter des Textes folgen. Dies hat lediglich einen Einfluss auf die Grammatik des Textes, welche für uns uninteressant ist. Tatsächlich hat die Abbildung auf Kleinbuchstaben einen großen Vorteil, da sie die Anzahl verschiedener Wörter im Vokabular, welche sich nur durch ihre Groß- und Kleinschreibung unterscheiden, verringert. Somit stellt sie eine sehr einfache und sehr effektive Möglichkeit dar die Anzahl an Wörtern im Vokabular zu vermindern.

5.5 Vorstudie: Auswahl eines Ausgangsverfahrens

Im Rahmen einer Vorstudie sollen mehrere Verfahren zur Themenmodellierung im Hinblick auf ihre Eignung zur Beschreibung der Semantik von Quelltextgruppierungen verglichen werden. In diesem Abschnitt wird zunächst der Korpus für dieser Vorstudie eingeführt. Danach wird eine Auswahl von Kandidatenverfahren getroffen, welche miteinander verglichen werden sollen. Dabei wird erläutert, wie die Ergebnisse der Analyse die Festlegung der Kandidatenverfahren mitbestimmt haben. Anschließend werden das Studiendesign und die gewählten Mittel zur Bewertung der Leistung der einzelnen Modelle beschrieben. Zuletzt werden die Ergebnisse der Vorstudie ausgewertet.

5.5.1 Vorstudienkorpus

Der Korpus der Vorstudie besteht aus fünf zufällig ausgewählten Quelltextgruppierungen des Gesundheitsverwaltungssystems iTrust. Insgesamt wurde der Quelltext des iTrust-Projekts durch INDIRECT in 120 Gruppierungen unterteilt. iTrust wurde ausgewählt, da es eine voll funktionsfähige Anwendung mit einer starken Domänenbindung ist. Außerdem sind für iTrust Anforderungsdokumente verfügbar, sodass evaluiert werden kann ob es sich lohnt solche Dokumente zusätzlich zum regulären Quelltext von Softwareprojekten mit in die Erzeugung von Themenbeschreibungen einfließen zu lassen. Auf Grund von technischen Limitierungen wurde das Gruppierungsverfahren von INDIRECT ohne den linguistischen Analyseschritt durchgeführt. Die Vorstudie basiert somit auf der Annahme, dass die linguistische Analyse die Güte der erzeugten Quelltextgruppierungen nur positiv beeinflussen kann. Die Festlegung auf fünf zufällige Quelltextgruppierungen stellt einen Kompromiss aus Umfang der Vorstudie und der Aussagekraft ihrer Ergebnisse dar. Da wir im Rahmen der Vorstudie mehrere Verfahren und mehrere verschiedene Eingaben an diese Verfahren evaluieren wollen können wir nur eine Stichprobe der Quelltextgruppierungen betrachten, da sonst die Anzahl an auszuwertenden Datenpunkten zu groß ist. Die Gruppierungen wurden zufällig ausgewählt, um eine Beeinflussung der Ergebnisse durch Auswahl besonders einfach zu beschreibender Gruppierungen zu vermeiden. Um die ausgewählten Gruppierungen in eine textuelle Repräsentation durch ihre Token zu überführen, werden die eben vorgestellten Quellen natürlicher Sprache und Vorverarbeitungsschritte genutzt.

5.5.2 Kandidatenverfahren

Im Rahmen der Vorstudie sollen mehrere Modelle und Verfahren aus verwandten Arbeiten miteinander verglichen und auf ihre Eignung zur Themenmodellierung in Quelltext überprüft werden. Die ausgewählten Verfahren sind die statistischen Themenmodelle LDA und HDP, ein auf [ZSA+16] basierendes Verfahren, welches den Schwerpunkt von word2vec und fastText Worteinbettungen nutzt und ein Verfahren, welches die Token einer Gruppierung basierend auf der NGD-Norm aus [MB17] bewertet. Diese Verfahren wurden ausgewählt, da es sich bei allen von ihnen um unüberwachte Verfahren handelt, welche Token als Eingabe verwenden und eine Menge von Token als Ausgabe produzieren. Da diese Verfahren nicht speziell auf Quelltext oder noch spezieller nur auf Methoden oder Klassen, sondern auf Dokumente allgemein ausgelegt sind, können sie speziell an die Quelltextgruppierungen angepasst werden.

Sowohl das NGD-Verfahren, als auch das HDP-Verfahren, wurden bereits vor dieser formalen Vorstudie auf anderen Quelltextgruppierungen evaluiert. Die dort erzielten Ergebnisse sollten jedoch direkt auf die Quelltextgruppierungen des Vorstudienkorpus übertragbar sein, weshalb sie nicht erneut auf den 5 zufällig ausgewählten Gruppierungen berechnet wurden.

Verfahren, welche auf Techniken der maschinellen Übersetzung basieren, werden in der Vorstudie nicht berücksichtigt. Dies hat den Grund, dass solche auf neuronalen Netzen basierende Modelle der Klasse der überwachten Lernverfahren zuzuordnen sind. Überwachte Lernmodelle benötigen während ihres Trainings beschriftete Datenpunkte. Im Fall von INDIRECT Gruppierungen setzt dies also einen Datensatz von Mengen von Quelltextelementen und einer dazugehörigen natürlichsprachlichen Beschreibung der Semantik der Quelltextelemente. Ein solcher Datensatz sollte im idealfall aus mehreren tausenden Datenpunkten bestehen. Da das Erstellen eines solchen Datensatzes ein sehr komplexes und zeitintensives Thema ist, ist es nicht realistisch im Rahmen dieser Arbeit einen solchen Datensatz zu erstellen. Es existieren zwar Datensätze, welche Java-Methoden und zugehörige Beschreibungen enthalten, jedoch keine Datensätze, welche ganze Gruppierungen

verschiedener Quelltextelemente enthalten. Potentiell könnte ein Datensatz für Quelltextgruppierungen aus solchen Datensätzen für Methoden oder Klassen approximiert werden. Allerdings ist nicht offensichtlich wie die natürlichsprachlichen Beschreibungen für einzelne Programmelemente zu Beschreibungen für mehrere Elemente kombiniert werden sollen. Außerdem ist fragwürdig, wie es um die Übertragbarkeit der Lernergebnisse auf einem solchen Datensatz steht, da Projekte aus verschiedenen Domänen beispielsweise verschiedene Fachbegriffe beinhalten können, die im Trainingsdatensatz nicht vorkamen. Das Verfahren zur Beschreibung von INDIRECT-Gruppierungen soll jedoch unabhängig von der Domäne des gruppierten Softwareprojekts funktionieren. Sollte solch ein Datensatz in Zukunft erstellt werden, ist es durchaus realistisch, dass LSTM basierte [HLWM20] oder Transformer basierte [ACRC20] Modelle trainiert werden können, um das Beschreiben solcher Gruppierungen zu lernen.

Wissensbasierte Themenmodelle [HHKG13] [AK15] nutzen externe Wissensquellen, wie Ontologien, um das in den Daten kodierte Wissen anzureichern und zu ergänzen. Solche Themenmodelle eignen sich besonders gut um Verbindungen zwischen Themen aus den vorliegenden Daten und neuen, in den Wissensquellen enthaltenen, Konzepten herzustellen. Wissensbasierte Themenmodelle funktionieren sehr gut, wenn die gewählte Wissensquelle zum Anwendungsfall des Themenmodells passt. Quelltext von Softwareprojekten hat die Eigenschaft sehr domänenspezifisch zu sein. Will man also wissensbasierte Themenmodelle auf Quelltext anwenden gibt es zwei Möglichkeiten dieses Problem zu lösen: Die erste Lösung ist es eine domänenübergreifende Ontologie anzulegen, welche Konzepte aus allen möglichen Domänen beinhaltet. Solche Welt-Ontologien hätten eine enorme Größe und würden vermutlich trotzdem nicht alle nötigen Konzepte abdecken. Um sie zu approximieren können Wissensquellen wie Wikipedia, DBPedia oder WordNet genutzt werden. Außerdem enthält Quelltext oft Wortneuschöpfungen und Abkürzungen, welche in solchen Ontologien mit hoher Wahrscheinlichkeit fehlen. Die zweite Lösung ist es für jede Domäne eine eigene Ontologie anzulegen. Dieser spezialisiertere Ansatz ist für eine kleine Anzahl an Domänen sehr gut geeignet, da sich die Ontologien für jede Domäne spezialisieren lassen. Für einen Anwendungsfall wie INDIRECT, welcher domänenübergreifend funktionieren soll ist auch dieser Ansatz nicht realistisch. Des Weiteren kann es sein, dass eine solche Wissensquelle nicht genügend Details der Domäne abdeckt, um von Nutzen zu sein, da sie bestimmte spezialisierte Entitäten und Beziehungen nicht kennt. Solche Verfahren können jedoch genutzt werden, um andere Verfahren zu ergänzen, indem man etwa Überbegriffe oder Konzepte für die Wörter einer Schlagwortmenge bildet. Da wir jedoch an einem Ausgangsverfahren interessiert sind, werden diese Verfahren und Aufbereitungstechniken hier nicht betrachtet.

5.5.3 Studiendesign

Die Vorstudie wird lediglich durch einen Teilnehmer, dem Autor dieser Arbeit, durchgeführt. Da im Rahmen der Vorstudie lediglich ein geeignetes Verfahren unter den Kandidatenverfahren ausgewählt werden soll, ist eine große Teilnehmerzahl nicht notwendig. Die durch die einzelnen Modelle erzeugten Themen werden in eine Tabelle überführt. Hierbei werden nur die 10 höchstbewerteten Wörter aus den Themen Für jede Quelltextgruppierung werden manuell Schlagworte ausgewählt, welche die Semantik der Gruppierung charakterisieren.

Als Basismetrik und zur ersten Quantifizierung der Leistung der Modelle wurde berechnet wie viele der Wörter eines erzeugten Themas mit den Token der betrachteten Quelltextgruppierung übereinstimmen. Diese Metrik ist nicht sehr genau, lässt jedoch eine erste Einschätzung der verschiedenen Modelle zu.

Zur Auswertung wird außerdem die Annahme getroffen, dass sich eine Quelltextgruppierung mit maximal fünf Schlagwörtern beschreiben lässt. Diese Annahme stellt einen

Erfahrungswert aus der Arbeit mit den iTrust-Quelltextgruppierungen dar. Zu beachten gilt, dass diese Annahme nicht unbedingt auf andere Softwareprojekte übertragbar ist. Im Rahmen der Vorstudie wird sie genutzt um zu bewerten wie häufig die manuell als wichtig identifizierten Wörter unter den Top-5 Wörtern der erzeugten Themen liegen. Diese Bewertung erscheint sinnvoll, da für perfekte Themen die wichtigsten Wörter des Themas mit den manuell als wichtig identifizierten Wörtern übereinstimmen sollten.

Im folgenden wird kurz eingeführt welche Konfigurationen der verschiedenen Kandidatenverfahren getestet werden:

Um zu überprüfen wie sich die Repräsentation einzelner Dokumente während des Trainings der Modelle auswirkt werden jeweils die Auswahlstrategien DPC (engl. „doc per cluster“, Dokument pro Gruppierung) und DPE (engl. *doc per element*, Dokument pro Element) getestet. DPC behandelt dabei jede der 120 Quelltextgruppierungen als ein Dokument und reicht diese als Eingabe an die Modelle weiter. DPE hingegen betrachtet jedes Quelltextelement einzeln als Dokument. Die Annahme ist hier, dass DPC eine bessere Leistung erzielen sollte, da das Modell alle semantische ähnlichen Elemente einer Gruppierung gleichzeitig erhält.

Wie in Abschnitt 2.4.2 deutlich wurde ist die Auswahl des Parameters k für LDA-Modelle nicht trivial. Für die Vorstudie nehmen wir an, dass die Quelltextgruppierungen von INDIRECT möglichst optimal sind und somit alle Gruppierungen eine einzige Semantik beschreiben. Somit sollte für perfekte Gruppierungen jeweils ein Thema pro Gruppierung erzeugt werden. Basierend auf dieser Annahme werden Werte für $k \in 90, 120, 150$ getestet, um einzuschätzen ob die Annahme sinnvoll ist und wie sich Abweichungen von diesem Ausgangswert verhalten.

Außerdem soll die Auswirkung zusätzlicher Trainingsdaten, wie Anforderungen oder den restlichen Klassen des Softwareprojekts, evaluiert werden. Zusätzlich soll getestet werden ob die Modelle auch mit dem gesamten Quelltext des Projekts trainiert werden können, anstatt nur auf dem Quelltext der Gruppierungen. Hierzu wird eine Reihe von Modellen ausschließlich auf dem vorverarbeiteten Projektquelltext trainiert. Eine weitere Reihe an Modellen wird lediglich auf dem Gruppierungsquelltext trainiert. Außerdem werden Reihen von Modellen trainiert, welche zusätzlich zu dem Gruppierungsquelltext die Anforderungen von iTrust, den Quelltext der restlichen Klassen von iTrust und eine Kombination all dieser Zusatzdaten als Trainingsgrundlage erhält.

Eine solche Reihe von Modellen enthält also jede mögliche Kombination von Trainingsstrategie, k-Parameter und zusätzlicher Trainingsdaten. Für die Einbettungsmodelle wird nur zwischen den Trainingsstrategien und den zusätzlichen Trainingsdaten unterschieden, da diese keinen k-Parameter besitzen. Allerdings werden ein vortrainiertes word2vec-Modell [ECS18] und ein vortrainiertes fastText-Modell [GBG+18] miteinander verglichen. Da für das HDP-Modell ebenfalls kein k-Parameter existiert, wird für dieses nur zwischen DPC und DPE unterschieden. Da die Leistung des Modells in vorherigen Tests sehr schlecht war wird außerdem auf die Überprüfung zusätzlicher Trainingsdaten verzichtet, da keiner Verbesserung der Leistung zu erwarten ist. Die Bewertung einer Gruppierung durch die NGD-Metrik wird ebenfalls gesondert behandelt, da sie keine Themen erzeugt sondern lediglich die vorhandenen Token einer Quelltextgruppierung bewertet.

Im Rahmen der Vorstudie sollen folgende Fragen beantwortet werden:

(RQ1) Wie wirkt sich das Hinzufügen von zusätzlichen Trainingsdokumenten, wie Anforderungen, auf die Leistung der einzelnen Modelle aus?

(RQ2) Welche der Trainingsstrategien (Dokument pro Gruppierung, Dokument pro Element und Dokument pro Klasse) auf die Leistung der Modelle aus?

(RQ3) Wie wirkt sich die Festlegung der Anzahl an Themen auf das LDA-Modell aus und approximiert HDP eine sinnvolle Anzahl an Themen?

(RQ4) Eignet sich der Schwerpunkt einzelner Worteinbettungen als Repräsentation geteilter Semantik?

(RQ5) Helfen vortrainierte Worteinbettungsmodelle diese Leistung zu verbessern?

(RQ6) Eignet sich die NGD-Metrik um wichtige Token in Gruppierungen zu identifizieren?

5.5.4 Auswertung

Im Folgenden werden die Ergebnisse der Vorstudie ausgewertet. Auf Grund der hohen Anzahl an getesteten Konfigurationen der Kandidatenmodelle können die Rohdaten der Vorstudie nicht in den Anhang dieser Arbeit angehängt werden. Sie können jedoch im Versionskontrolldepot dieser Arbeit eingesehen werden.

5.5.4.1 (RQ1): Auswirkung zusätzlicher Trainingsdaten

Name	Token				
LDA 120 DPC Klassen	get	**bean**	personnel	**monitor**	data
LDA 120 DPC Cluster	**bean**	**set**	get	mid	**monitor**
LDA 120 DPC K+C	**bean**	load	loader	data	**set**
LDA 120 DPC R+C	**bean**	**set**	referr	advers	event
LDA 120 DPC All	data	**remot**	report	**monitor**	**bean**

Tabelle 5.1: Themen der LDA-Modelle mit $k = 120$ trainiert auf allen Klassen, den Quelltextgruppierungen, den Klassen und Gruppierungen (K + C), den Anforderungen und Gruppierungen (K + C), sowie allen zusätzlichen Daten

Im Rahmen dieser Forschungsfrage soll evaluiert werden auf welchen Trainingsdaten die Modelle die beste Leistung erzielt haben. Tabelle 5.1 zeigt die Top-5 Wörter der erzeugten Themen der LDA-Modelle mit verschiedenen zusätzlichen Trainingsdaten. Sollte ein Modell mehr als ein Thema erzeugt haben, so wird lediglich das Thema mit dem höchsten Anteil dargestellt. Die erwarteten Schlagwörter der Gruppierung sind fett hervorgehoben. In diesem Fall sind die Schlagwörter „bean“, „remot“, „monitor“ und „set“. Weder die Übereinstimmungsmetrik der Token noch die Auswertung anhand der manuell festgelegten wichtigen Wörter einer Gruppierung lassen eine eindeutige Aussage zu. Somit scheinen die notwendigen Informationen zur Erzeugung von Themen für Quelltextgruppierungen auch bereits in diesen Gruppierungen vorzuliegen. Das Hinzufügen weiterer Trainingsdaten wirkt sich im Allgemeinen weder positiv noch negativ auf die Leistung der Modelle aus.

5.5.4.2 (RQ2): Auswirkung der Trainingsstrategien

Tabelle 5.2 zeigt die Top-5 Wörter der Top-Themen verschiedener LDA-Modelle mit den DPC- und DPE-Strategien Betrachtet man die Themen der Modelle, welche mit der DPC- und DPE-Trainingsstrategie trainiert wurden, kann man Unterschiede zwischen den beiden Ansätzen zu erkennen. Am auffälligsten ist, dass die DPE-Modelle häufiger mehrere Themen pro Gruppierung ausgeben. Allerdings verteilen sich die manuell als wichtig identifizierten Wörter dann ebenfalls auf diese mehreren Themen und häufig tragen sie einen niedrigeren Anteil an dem Thema insgesamt, als in den Themen der DPC-Modelle. Ein möglicher Grund hierfür ist, dass das Modell im Trainingsschritt jedes Element einzeln lernt und somit Themen für die Semantiken der einzelnen Elemente, jedoch nicht für

Name	Token				
LDA 90 DPC Cluster	**bean**	**set**	referr	get	email
LDA 120 DPC Cluster	**bean**	**set**	get	mid	**monitor**
LDA 150 DPC Cluster	**bean**	**set**	visit	prescript	office
LDA 90 DPE Cluster	**set**	appt	**bean**	report	role
LDA 120 DPE Cluster	bodi	**set**	messag	mid	**bean**
LDA 150 DPE Cluster	mid	report	**bean**	data	**set**

Tabelle 5.2: Themen der LDA-Modelle mit den Trainingsstrategien DPC und DPE, trainiert auf den Token der Quelltextgruppierungen.

die Gruppierungen als ganzes lernt. Diese einzelnen Themen werden dann von dem Modell kombiniert um die Semantik der abgefragten Gruppierung zu approximieren. Somit scheint die DPC-Trainingsstrategie eher Themen zu erzeugen, welche die geteilte Semantik der abgefragten Gruppierungen abbilden.

5.5.4.3 (RQ3): Auswirkungen des k-Parameters und HDP

Die Wahl des k-Parameters der LDA-Modelle legt fest, wie viele Themen insgesamt durch sie erzeugt werden. Das festlegen eines Optimalwerts für diesen Parameter ist allgemein nicht möglich (siehe Abschnitt 2.4.2). Im Rahmen der Vorstudie wurde dieser Wert anhand der Anzahl an Quelltextgruppierungen von iTrust approximiert. Alle Konfigurationen des HDP-Modells erzeugen 150 Themen. Dies entspricht 125% der Anzahl an Quelltextgruppierungen. Allerdings sind die von HDP erzeugten Themen von sehr niedriger Qualität: Sie stimmen fast nie mit den erwarteten Beschreibungen der Gruppierungen überein und die einzelnen Terme der Themen machen meist nur einen einstelligen Anteil des Themas aus. Dies bedeutet, dass HDP sehr unspezifische Themen inferiert, welche für diesen Anwendungsfall quasi unbrauchbar sind. Zwischen den LDA-Modellen mit $k = 90, 120, 150$ fallen nur geringfügige Unterschiede auf. Dies kann man auch in Tabelle 5.2 erkennen. Alle der Modelle erzeugen sinnvolle Themen, welche eine gute Übereinstimmung mit den manuell festgelegten Gruppierungsbeschreibungen aufweisen. Somit scheint es einen gewissen Spielraum bei der Festlegung des k-Parameters zu geben. Gleichzeitig zeigt dieses Ergebnis, dass ein Ausgangswert für k gleich der Anzahl an Gruppierungen für ein Softwareprojekt sinnvoll erscheint.

5.5.4.4 (RQ4): Schwerpunkt von Worteinbettungen

Die Worteinbettungsmodelle word2vec und FastText erreichen eine durchschnittliche Leistung. Häufig sind die erwarteten Wörter in den von den Modellen erzeugten Themen enthalten, jedoch enthalten sie auch viele irrelevante Wörter. word2vec scheint mit dieser Art von Daten besser umgehen zu können, als FastText. Außerdem scheinen die Worteinbettungsmodelle anders als die anderen getesteten Modelle von zusätzlichen Daten aus anderen Klassen oder den Anforderungen zu profitieren. Die Ähnlichkeiten der gefundenen Wörter zum Schwerpunkt des Dokuments im Vektorraum ist bei allen Wörtern sehr hoch (meist $> 99\%$). Dies lässt darauf schließen, dass die eingebetteten Vektoren sehr nahe bei einander liegen und somit keine all zu gute Trennung der Absichten im Vektorraum erreicht wurde. Eine Nutzung von Einbettungen, welche lediglich auf den Gruppierungen selbst trainiert wurden scheint also nicht sinnvoll.

5.5.4.5 (RQ5): Vortrainierte Einbettungsmodelle

Sowohl für word2vec, als auch für fastText, existieren vortrainierte Modelle. Das vortrainierte word2vec-Modell, das im Rahmen der vorstudie genutzt wurde, wurde speziell

auf Dokumenten aus der Domäne der Softwaretechnik trainiert. Bei dem vortrainierten fastText-Modell handelt es sich um ein Modell das auf Daten aus dem Common Crawl Datensatz [1] und von Wikipedia [2] trainiert wurde. Die ursprüngliche Dimension dieser Einbettungen betrugt 300, um den Speicherverbrauch des Modells zu reduzieren wurden sie jedoch auf 100 reduziert. Dies kann sich auf die Leistung des Modells auswirken, schien in der Vorstudie jedoch keine Auswirkung zu haben. Sowohl das vortrainierte word2vec Modell, als auch das vortrainierte fastText Modell neigen dazu Abwandlungen eines Wortes der Gruppierung zu finden. Eine Gruppierung des iTrust-Projektes beschäftigt sich mit Emails, sowie deren Betreff und Inhalt. In den bereinigten Tokens der Gruppierung kommen „str“ für „String“ (Zeichenkette) und „email“ vor. Das vortrainierte word2vec Modell liefert Wörter wie: „my_string“, „new_str“ oder „your_string“. Das vortrainierte fastText Modell liefert Wörter wie: „emailing“, „e-mail“ oder „e-email“. Das fastText Modell liefert darüber hinaus Wörter, welche keine natürlichsprachlichen Wörter sind. Dies liegt daran, dass fastText auf Subtokens arbeitet und somit gänzlich neue Wörter erzeugen kann. Der Schwerpunkt einzelner Wortvektoren eignet sich also offensichtlich nicht als Repräsentation der Semantik eines Dokuments insgesamt.

5.5.4.6 (RQ6): NGD-Metrik zur Bewertung von Token

Die NGD Metriken sind grundsätzlich auch auf die Gruppierungen von INDIRECT anwendbar. Sie können genutzt werden, um die Güte der Gruppierungen von INDIRECT zu bewerten und die Wichtigkeit von Token innerhalb einer Gruppierung zu bestimmen. Letzteres funktioniert teilweise sehr gut aber auch teilweise sehr schlecht. Allgemein scheint NGD seltene oder einzigartige Wörter viel zu stark zu gewichten, weshalb diese häufig Themen dominieren. Außerdem können mit der NGD Metrik selbst nur Token bewertet werden, welche bereits innerhalb der Gruppierung vorkommen. Token, welche außerhalb einer Gruppierung auftreten und dennoch für ein Thema wichtig sind können nicht berücksichtigt werden. Die NGD-Metrik ist also eher ungeeignet um Themen zu bestimmen, kann aber potentiell genutzt werden um bestehende Themen zu bewerten.

5.5.4.7 Zusammenfassung der Vorstudie

Das Hinzufügen von zusätzlichen Dokumenten, wie den anderen Klassen des Softwareprojekts oder den zugehörigen Anforderungen zeigt keine erheblichen Auswirkungen auf die Leistung der Modelle. Die Verfahren basierend auf den Worteinbettungsmodellen word2vec und fastText haben bestenfalls durchschnittliche Ergebnisse erreicht. Auffällig ist, dass word2vec den meisten Token eine Relevanz von $> 99\%$ zuordnet. Dies lässt darauf schließen, dass die Elemente des Einbettungsraums sehr dicht beieinander liegen. Das ist ein Indiz darauf, dass nicht genügen Trainingsdaten vorhanden waren um genügende Kookkurrenzinformationen von Wörtern zu sammeln. Das vortrainierte word2vec Modell enthält in der Nähe des Schwerpunktes der getesteten Gruppierungen häufig verschiedene Formen eines Token aus einer Gruppierung. Diese Zuordnungen sind zwar theoretisch korrekt, stellen allerdings keine adequate Repräsentation des Schwerpunkts der Gruppierung dar. Das vortrainierte fastText Modell zeigt ein ähnliches Verhalten, allerdings erzeugt es häufig Wörter, die keinen natürlichsprachlichen Wörtern entsprechen. Somit scheint der Schwerpunkt der Wortvektoren der Token einer Gruppierung keine sinnvolle Repräsentation der Semantik der Gruppierung als ganze zu sein.

Die NGD-Metrik wird verworfen, da sie häufig unwichtige Wörter zu gut und wichtige Wörter zu niedrig bewertet hat.

[1] https://commoncrawl.org/
[2] https://www.wikipedia.org/

Insgesamt gehen die auf den Gruppierungen von iTrust trainierten LDA-Modelle als die Sieger der Vorstudie hervor. Die von ihnen erzeugten Themen erreichen in einer Stichprobe von 5 Gruppierungen immer eine gute Überlappung der erwarteten Token mit den Token der von den Modellen erzeugten Themen. Außerdem zeichnet sich ein Wert für k gleich der Anzahl an Gruppierungen für ein Softwareprojekt als ein guter Ausgangspunkt ab. Aufgrund dieser Ergebnisse wird die weitere Analyse auf Basis von LDA-Modellen, einem k gleich der Anzahl an Quelltextgruppierungen und einem Trainingsdokument pro Quelltextgruppierung ablaufen.

5.6 Entwurf des Gruppierungsthemenmodells

Da die Ergebnisse der Vorstudie LDA als Ausgangsverfahren nahelegen soll nun ein Themenmodell für Quelltextgruppierungen auf Basis von LDA entworfen werden. Hierzu wird betrachtet wie man wichtige Wörter in Themen identifizieren kann, um die Länge der daraus entstehenden Schlagwortmenge einzugrenzen. Außerdem wird untersucht ob mehrere LDA-Modelle in Kombination genutzt werden können um Unsicherheiten, etwa in der Wahl des Parameters k, auszugleichen.

5.6.1 Struktur von LDA-Themen

Das LDA-Verfahren wird auf einer Menge von Dokumenten, in unserem Fall den Token von Quelltextgruppierungen, trainiert und kann anschließend mit einzelnen Dokumenten abgefragt werden. Bei den Abfragedokumenten handelt es sich in unserem Fall ebenfalls um die Token einer Quelltextgruppierung. Das LDA-Verfahren liefert dann eine Liste von Themen, welche die Quelltextgruppierung charakterisieren. Wie viele Themen pro Abfrage erzeugt werden hängt von der abgefragten Quelltextgruppierung, sowie den Parametern des LDA-Modells (siehe Abschnitt 2.4.2) ab.

LDA-Themen besitzen meist einige wenige Wörter mit einem hohen Anteil an dem gegebenen Thema und viele Wörter mit einem sehr niedrigen Anteil. Da ein hoher Anteil eine hohe Sicherheit des Modells im Bezug auf die Zugehörigkeit eines Wortes zu einem Thema bedeutet, sollten diese Wörter besonders betrachtet werden. Im Folgenden werden hierzu zwei mögliche Auswahlstrategien vorgestellt.

5.6.2 Identifikation wichtiger Wörter in Themen

Ein LDA-Modell erzeugt als Ausgabe eine Liste von Themen, welche selbst wieder Listen von Paaren aus Wörtern und einem zugehörigen Anteil des Wortes am Thema bestehen. Da sich die Anteile der Wörter zu 1 summieren müssen, bestehen Themen meist aus wenigen Wörtern mit hohen Anteilen und vielen Wörtern mit kleinen Anteilen. Dies wirft die Frage auf, wie viele Wörter eines Themas wirklich von Interesse sind. Im Allgemeinen möchte man die Wörter mit den höchsten Anteilen erhalten, allerdings ist unklar wo die Grenze zwischen relevant und irrelevant liegt. Es ergeben sich zwei Möglichkeiten zur Auswahl von Wörtern aus LDA-Themen. Einer dieser Ansätze extrahiert eine statische Anzahl an Wörtern, während der andere ein dynamisches Auswahlverfahren nutzt. Im folgenden werden beide dieser Verfahren eingeführt und gegeneinander abgewägt.

5.6.2.1 Auswahl einer fixen Anzahl an Wörtern

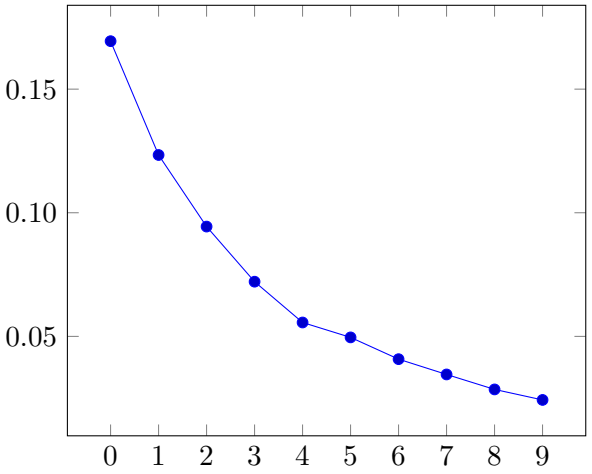


Abbildung 5.6: Darstellung der Position eines Token in einem Thema gegen den Anteil des Token am Thema. Berechnet für ein LDA-Modell und Gruppierungen des iTrust-Softwareprojekts

Betrachtet man die von den LDA-Modellen erzeugten Themen, so stellt man fest, dass häufig nur sehr wenige Wörter in einem Thema einen hohen Anteil ($> 10\%$) haben und die einzelnen Anteile danach sehr schnell abnehmen. Die Genauere Betrachtung mehrerer Themen, in Abbildung 5.6, zeigt, dass bereits das fünfte Wort in einem Thema durchschnittlich nur einen Anteil von etwa 5% besitzt. Ein Problem mit dieser statischen Auswahl an Token zeichnet sich ab, wenn ein Thema sehr ausgeglichen ist und die Anteile der Wörter nahe bei einander liegen.

Beispiel 5.13:

Gegeben sei ein Thema mit 10 Wörtern, welche jeweils einen Anteil von 10% besitzen. Das statische Auswahlverfahren mit 5 Wörtern würde hier nur die hälfte der Wörter auswählen, obwohl alle Wörter gleich wichtig sind.

5.6.2.2 Dynamische Auswahl von Wörtern

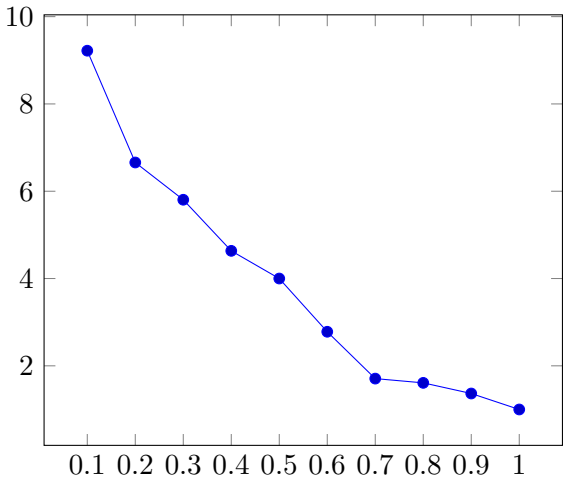


Abbildung 5.7: Darstellung der Anzahl an Token nach Auswahlfaktor. Berechnet für ein LDA Modell und Gruppierungen des iTrust-Softwareprojekts

Die statische Natur des ersten Ansatzes macht ihn zu unflexibel um zuverlässig die wichtigsten Token eines Themas auszuwählen. Ein weiterer Ansatz basiert auf dem maximalen Anteil eines Wortes in einem Thema. Durch Multiplikation des maximalen Anteils mit einem Wert p zwischen 0 und 1 erhält man einen Bereich, aus dem Token ausgewählt werden können. Es werden dann alle Token zwischen dem maximalen Anteil max und der unteren Grenze $max * p$ ausgewählt.

Beispiel 5.14:

Gegeben sei ein Thema, dessen maximaler Wortanteil 15% beträgt. Auswahl mit $p = 0.5$ liefert Token bis zu einem Anteil von 7.5%. Auswahl mit $p = 0.3$ liefert Token bis zu einem Anteil von 5%.

Die Wahl eines optimalen Parameters p ist nicht-trivial. Versuche mit den Gruppierungen des iTrust Projekts (siehe Abbildung 5.7) haben gezeigt, dass ein Wert zwischen 0.3 und 0.5 eine akzeptable Anzahl an Wörtern pro Thema liefert.

5.6.3 Themenstabilität durch Kombination mehrerer Modelle

Wie bereits angesprochen stellt die Wahl des richtigen Parameters k für LDA Modelle eine offene im Bereich der Themenmodellierung dar. Es ist also wahrscheinlich, dass die Wahl von k als 100% der Anzahl an Quelltextgruppierungen nicht perfekt ist. Aufgrund unserer getroffenen Annahme, dass sich der optimale Parameter k in der Nähe dieses idealisierten Parameters befindet, stellt sich die Frage, ob mehrere Modelle mit verschiedenen (ähnlichen) Werten für k kombiniert werden können, um die Stabilität der erzeugten Themen zu erhöhen. Um dieser Frage nachzugehen, werden fünf LDA Modelle auf denselben Dokumenten trainiert. Der Wert k der Modelle wird auf 70%, 85%, 100%, 115% und 130% der Anzahl an Themen festgelegt. Da sich diese Werte relativ Nahe bei einander liegen, sollten die erzeugten Themen für die selbe Quelltextgruppierung ähnlich sein. Das bedeutet, dass die Wörter, welche für die Quelltextgruppierung am wichtigsten sind, in den Themen aller Modelle enthalten sein sollten. Angenommen zu jeder Quelltextgruppierung wird exakt ein Thema erzeugt, dann kann ein Wort dahingehend bewertet werden in wie vielen der Themen es vorkommt.

Beispiel 5.15:

Gegeben sei eine Quelltextgruppierung G. Es werden drei LDA Modelle trainiert und es werden die zu der Gruppierung G gehörigen Themen abgefragt. Zwei der Themen enthalten das Wort „employee“, eines mit einem Anteil von 15% und ein anderes mit einem Anteil von 8%. Der Gesamtanteil des Wortes „employee“ beträgt also $\frac{0.15+0.08}{3} \approx 0.07$ bzw. 7%.

Häufig kommt es jedoch vor, dass zu einer Quelltextgruppierung von einem Modell mehrere Themen erzeugt werden oder seltener kann es vorkommen, dass nur ein Modell ein Thema erzeugt, welches jedoch wichtig für die Semantik der Gruppierung ist.

Beispiel 5.16:

Gegeben sei eine Quelltextgruppierung G, welche Quelltextelemente beinhaltet die Mitarbeiterverträge verarbeiten. Angenommen die drei LDA Modelle erzeugen jeweils ein Thema, welches das Wort „employee“ beinhaltet und nur eines der Modelle erzeugt ein Thema welches das Wort „contract“ beinhaltet. Da das Wort „employee“ in $\frac{3}{4}$ Themen vorkommt, das Wort „contract“ jedoch nur in $\frac{1}{4}$ Themen wird das Wort „employee“ als wichtiger als das Wort „contract“ identifiziert, obwohl beide wichtig für die Semantik der Gruppierung sind.

Um solche Themen berücksichtigen zu können, müssen Themen miteinander verglichen werden können. Die Themen von LDA Modellen sind grundsätzlich nur Wahrscheinlichkeitsverteilungen über das Vokabular des Modells. Das bedeutet, dass Distanzmetriken für Wahrscheinlichkeitsverteilungen ebenfalls für LDA Themen genutzt werden können. Eine solche Distanzmetrik ist die Hellinger-Distanz (siehe Abschnitt 2.6). Die Hellinger-Distanz liefert einen kontinuierlichen Wert zwischen 0 und 1 der angibt, wie ähnlich zwei Wahrscheinlichkeitsverteilungen sind. Zu beachten gilt hierbei, dass ein Wert von 0 eine sehr hohe Ähnlichkeit und ein Wert von 1 eine sehr niedrige Ähnlichkeit repräsentiert.

Beispiel 5.17:

Gegeben seien zwei Themen $t_1 = \{(employee, 1.0)\}, t_2 = \{(contract, 1.0)\}$. Da sich keine der Wörter der Themen überschneiden beträgt die Hellinger-Distanz $hellinger(t_1, t_2) = 1.0$ und ist somit maximal.

Um ähnliche Themen zu gruppieren, kann ein hierarchisches agglomeratives Clustering genutzt werden. Die Verknüpfungsstrategie des Clusterings wird als „complete“ gewählt (siehe Abschnitt 2.7). Das bedeutet, dass immer der Maximalwert der paarweisen Distanzen zwischen zwei Clustern als Kriterium genutzt wird, ob zwei Cluster weiter verschmolzen werden sollen. Der Schwellwert, bis zu dem Cluster verschmolzen werden wird auf 0,5 gesetzt, da die Hellinger-Distanz im Bereich von 0 bis 1 liegt und dies somit intuitiv der Punkt ist, ab dem sich zwei Themen nicht mehr ähnlich sind.

Das Verfahren beginnt mit einer Gruppierung pro Thema. In jedem Schritt werden die Gruppierungen paarweise miteinander verglichen und die zwei ähnlichsten Gruppierungen werden miteinander verschmolzen. Dies wird so lange wiederholt, bis es keine zwei Gruppierungen mehr gibt, deren Distanz niedriger als ein festgelegter Schwellwert ist. Jede dieser Gruppierungen beinhaltet dann ähnliche Themen, was in diesem Fall bedeutet, dass

in den Themen einer Gruppierung jeweils den gleichen Wörtern ähnliche Wahrscheinlichkeiten zugeordnet sind.

Bildet man nun den Durchschnitt aller Wahrscheinlichkeiten für ein Wort in den Themen einer solchen Gruppierung erhält man ein Maß für die Relevanz des Wortes für die ganze Gruppierung. Um zu verhindern, dass Aureißer einen zu großen Einfluss auf diesen Durchschnittswert auswirken, werden lediglich die Top-N Wörter eines Themas betrachtet. Bestimmt man diesen Wert für jedes Wort in den Themen einer Gruppierung, so ergibt dies eine neue Wahrscheinlichkeitsverteilung. Diese Verteilung hat die Eigenschaft den Wörtern, denen in allen Themen hohe Wahrscheinlichkeiten zugeordnet sind, ebenfalls hohe Wahrscheinlichkeiten zuzuordnen. Außerdem ordnet sie Wörtern, welche in den Themen stark schwankende Wahrscheinlichkeiten annehmen, eher niedrige Wahrscheinlichkeiten zu. Somit charakterisieren die am höchsten gewichteten Wörter dieser Verteilung die Themengruppierung. Berechnet man diese neue Verteilung für alle Gruppierungen, so erhält man eine Menge von Wahrscheinlichkeitsverteilungen, was nach Definition nichts anderes ist als eine Menge von

Mantyla *et al.* nutzen in „Measuring LDA Topic Stability from Clusters of Replicated Runs“ [MCF18] eine Vorgehensweise, welche auf derselben Idee der Kombination mehrerer Modelle zur Stabilisierung der Erzeugten Themen basiert, jedoch grundsätzlich anders implementiert wird. Die Autoren führen ihr LDA-Modell mit k Gruppierungen n mal aus und erhalten somit insgesamt $t = nk$ Themen. Anschließend wollen sie diese t Themen mithilfe eines Clustering Verfahrens wieder auf k Themen reduzieren. Zunächst bestimmen sie GloVE-Worteinbettungen für ihr gesamtes Vokabular. Anschließend bilden sie eine $w \times v$ Matrix von Wörtern zu Worteinbettungen, wobe v die Dimension der Einbettungen ist. Durch Multiplikation der Themen-Wort-Matrix und der Wort-Einbettungs-Matrix entsteht eine Themen-zu-Worteinbettungsmatrix mit Dimension $t \times v$. Anschließend führen die Autoren ein KMedoids-Clustering aus um daraus wieder k Themen zu erzeugen. Zur Evaluation nutzen sie die sogenannte RBO (rank-biased overlap) Metrik, welche sowohl die Überlappung, als auch den Anteil eines Themas selbst nutzt um einen Ähnlichkeitswert für sie zu bestimmen. Diese RBO-Metrik kann potentiell an Stelle der vorgestellten Hellinger-Distanz genutzt werden um die Ähnlichkeit von Themen währen des agglomerativen Clusterings zu bewerten. Aus Zeitgründen konnte eine solche Evaluation allerdings nicht vorgenommen werden. Die Ergebnisse von Mantyla *et al.* zeigen jedoch, dass die Kombination mehrerer Modelle durchaus zu stabileren LDA-Themen führen kann.

5.6.4 Durschnitt von Themen und Gruppierungen

Betrachtet ein Mensch Gruppierungen von Quelltextelementen, so liegt sein Fokus zunächst auf den Token die innerhalb der Gruppierung enthalten sind. Diese Beobachtung erlaubt es uns die Menge an potenziell wichtigen Token der Quelltextgruppierung weiter einzuschränken. Durch bilden des Durchschnitts der Token der Quelltextgruppierung und den gefundenen Themen können unter dieser Annahme also wichtige Token leicht identifziert werden. Token, welche nicht Teil dieses Durchschnitts sind, sollten jedoch nicht verworfen werden, da diese weiteren Kontext für das Thema bilden.

5.6.5 Rücktransformation der Stammformreduktion

Die zur vorverarbeitung von Token genutzte Stammformreduktion nutzt nacheinander angewandte Reduktionsregeln. Dies führt dazu, dass die Ausgabe der Stammformreduktion meist kein valides natürlichsprachliches Wort ist. Eine Rücktransformation von einer reduzierten Form zu einem natürlichsprachlichen Wort kann nützlich sein, um eine spätere Interpretation der Themen zu erleichtern. Um dies zu ermöglichen, kann eine Zuordnung von reduzierter Form zu einer Liste von natürlichsprachlichen Wörtern genutzt werden. In

dieser Liste wird festgehalten, wie häufig ein natürlichsprachliches Wort auf eine bestimmte Stammform reduziert wurde. Naheliegend ist, dass das am häufigsten reduzierte Wort auch wieder das Ziel dieser Rücktransformation sein sollte.

5.7 Entwurfszusammenfassung

Im Rahmen der Analyse wurden mehrere Repräsentationsformen für Quelltext und für Quelltextbeschreibungen, sowie Quellen natürlicher Sprache in Quelltextgruppierungen betrachtet. Um Token aus den Quelltextgruppierungen zu extrahieren werden für Klassen die Informationsquellen **Javadoc-Kommentar**, **einfacher Bezeichner**, **Feldbezeichner**, **Konstruktor-Javadoc** und **Konstruktorparameter** verwendet. Für Methodenelemente werden die Informationsquellen **Klassenname**, **Javadoc**, **einfacher Bezeichner**, **Parameterbezeichner** und **Variablenbezeichner** genutzt. Die Token, welche aus diesen Quellen gewonnen werden, werden im Anschluss durch ein Verarbeitungsfließband bestehend aus den Schritten **HTML-Entfernung**, **Paketpfadentfernung**, **Subwort-Tokenisierung**, **Stopwortentfernung** und **Längenfilterung** vorverarbeitet. Basierend auf diesen vorverarbeiteten Quelltextgruppierungen werden LDA-Modelle trainiert, deren k Parameter (Anzahl an Themen) auf 70%, 85%, 100%, 115% und 130% der Anzahl an Gruppierungen gesetzt werden. Um die Stabilität der von den Modellen erzeugten Themen zu erhöhen werden sie mithilfe eines agglomerativen Clusterings kombiniert. In einem zweistufigen Rücktransformationsschritt werden die daraus entstehenden Gruppierungen von Themen zu einem einzelnen Thema zusammengefasst. Durch bilden des Durchschnitts der Token der Themen und den Token der Gruppierung werden wichtige und zusätzliche Token identifiziert. In einem letzten Schritt werden die im Rahmen der Vorverarbeitung in Stammformen transformierten Token mithilfe eines Wörterbuches rücktransformiert.

6 Implementierung

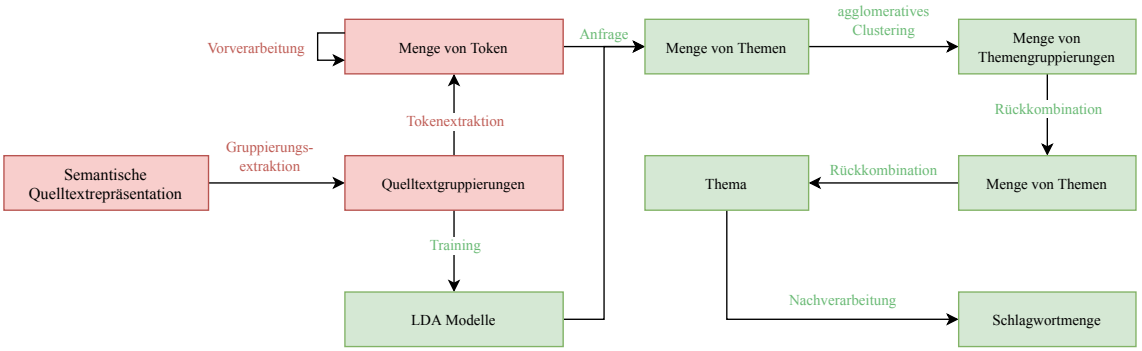


Abbildung 6.1: Überblick über das Zusammenspiel der verschiedenen Komponenten der Implementierung.

In diesem Kapitel wird die Implementierung der bereits getroffenen Entwurfsentscheidungen diskutiert. Die Implementierung selbst und auch diese Kapitel besteht aus zwei Teilen. Der erste Teil der Implementierung ist in Java geschrieben und beschäftigt sich mit der Interaktion mit der graphenbasierten Quelltextrepräsentation von INDIRECT, sowie der Vorverarbeitung von Quelltexttoken. Der zweite Teil ist in Python geschrieben und ist für das Training und die Abfrage der LDA Modelle, sowie der Nachverarbeitung der Themen verantwortlich. Abbildung 6.1 zeigt schematisch das Zusammenspiel der in Java und in Python implementierten Komponenten des Themenmodells. Die rot hinterlegten Knoten- und Kantenbeschriftungen entsprechen den in Java implementierten und aufgerufenen Komponenten, während die grünen Teile des Graphen der Python-Implementierung entsprechen.

6.1 Extraktion von Gruppierungen aus dem PARSE Graph

Das INDIRECT Projekt baut auf dem PARSE Framework auf und nutzt dessen agentenbasierte Architektur zur Interaktion mit dem Quelltextgraphen. Der im Rahmen dieser Arbeit implementierte PARSE Agent erweitert den von PARSE bereitgestellten `AbstractAgent`, um später gemeinsam mit anderen Agenten auf dem Quelltextgraphen arbeiten zu können. Zunächst werden alle Absichtsknoten des Graphen in einer Liste gespeichert. Diese Absichtsknoten und eine Instanz der `IGraph` Schnittstelle werden genutzt um eine

Instanz der `Corpus` Klasse zu erzeugen. Diese `Corpus` Klasse dient der Kapselung aller Quelltextgruppierungen des Softwareprojekts, sowie der Vorverarbeitung der Token der Gruppierung. Außerdem stellt sie Informationen über die Gruppierungen, wie etwa die minimale, durchschnittliche oder maximale Anzahl an Token der Gruppierungen bereit.

6.2 Von Absichtsknoten zu Quelltextelementen

Die Klasse `Cluster` repräsentiert eine Quelltextgruppierung innerhalb der `Corpus` Klasse. Für jeden Absichtsknoten des Korpus wird eine Cluster-Instanz erzeugt.

Um Quelltextduplikation zu vermeiden, wurde eine abstrakte Klasse `ClusterElement` eingeführt, welche gemeinsame Eigenschaften von Methoden- und Klassenelementen beinhaltet. Diese gemeinsamen Informationen sind der vollqualifizierte bzw. einfache Bezeichner und ein optionaler Javadoc Kommentar. Die `ClusterElement` Klasse implementiert die `IClusterElement` Schnittstelle, welche die zwei Methoden `toDTO`, zur Erzeugung eines Datentransferobjekts und `getProcessedTokens`, welche die verarbeiteten Token der Gruppierung zurück gibt, bereit stellt.

Die konkreten Implementierungen `ClassElement` und `MethodElement` nutzen die in Kapitel 5 identifizierten Tokenquellen. Diese Tokenquellen sind zur einfachen Wiederverwendung als statische Methoden in der Klasse `NodeParser` enthalten. Eine solche Methode erhält immer exakt einen Parameter, nämlich den PARSE-Knoten des Quelltextelements, aus dem die Token extrahiert werden sollen.

6.3 Vorverarbeitung der Token

Innerhalb der `Corpus` Klasse wird die bereits aus der von Eurich im Rahmen der linguistischen Quelltextanalyse implementierte `ProcessPipeline` Klasse genutzt um die Token des Korpus zu verarbeiten. Diese Klasse implementiert das Fließband Entwurfsmuster. Durch implementieren der `IPreprocessStep` Schnittstelle kann die Vorverarbeitung durch eigene Schritte ergänzt werden. Einige Verarbeitungsschritte, wie das Entfernen von HTML, Wort-Tokenisierung oder Abbildung auf Kleinbuchstaben wurden bereits im Rahmen früherer Arbeiten am INDIRECT Projekt implementiert und können wiederverwendet werden. Die Klasse `PackagePathRemover` wurde hinzugefügt, um Paketpfade aus den vollqualifizierten Namen von Typen zu entfernen. Dieser Schritt wurde mithilfe eines regulären Ausdrucks implementiert, welcher den Paketpfad akzeptiert, jedoch nicht den Bezeichner. Die Klasse `PreservingStopWordRemover` wurde als Erweiterung des vorhandenen `StopWordRemover` Verarbeitungsschrittes implementiert und erlaubt es nutzerdefinierte Wörter, welche eigentlich Teil der genutzten Stopwortlisten sind, zu erhalten. Anders als die Klasse `StopWordRemover`, welche als schwarze Liste fungiert, fungiert sie als weiße Liste und kann genutzt werden um Wörter explizit zu erlauben, ohne die Stoppwortlisten manuell bearbeiten zu müssen. Außerdem wurde die Klassen `CustomStopWordRemover` zur Entfernung von Wörtern, welche nicht in den Stopwortlisten enthalten sind und die `LengthFilter` zur Entfernung von Token mit einer Länge $< n$ implementiert.

6.4 Implementierung des Themenmodells auf Basis von LDA

Zur Implementierung des Themenmodells selbst wird Python genutzt. Es kommt die Bibliothek **gensim** [ŘS10] zum Einsatz, da diese bereits eine Implementierung des LDA Modells besitzt und eine der meistgenutzten Bibliotheken für Themenmodelle ist. Zur einfacheren Handhabung der Daten werden auf der Python Seite ebenfalls Klassen implementiert, welche `Corpus` und `Cluster` der Java-Seite repräsentieren. Zusätzlich wird eine Klasse `Topic` implementiert, welche die von LDA erzeugten Themen kapselt. Da im

Rahmen der Vorstudie mehrere Modelle verglichen und in Zukunft eventuell noch andere Modelle getestet werden sollen, wird eine Oberklasse `BaseModel` implementiert. Diese definiert lediglich die abstrakten Methoden `get_name` und `train`, sowie `query` zum Trainieren und Abfragen eines Modells. Die konkrete Implementierung `LDAModel` implementiert diese Funktionen um ein gensim `LdaModel` zu trainieren und abzufragen. Für die Parameter des LDA Modells 2.4.2 werden die gensim Standardeinstellungen verwendet.
Das eigentliche Verfahren, welches die Ausgabe mehrere LDA-Modelle kombiniert, wird durch die Klasse **KeywordSummarizer** implementiert, welche ebenfalls von `BaseModel` erbt. Um die Kombination der Themen mehrerer LDA Modelle zu implementieren werden die Bibliotheken **gensim** (Bereitstellung der Hellinger-Distanz), sowie **sklearn** [PVG+11] (Bereitstellung des agglomerativen Clusterings) genutzt. Zunächst werden n LDA Modelle mit verschiedenen Werten für k (Anzahl an Themen) trainiert und in der `KeywordSummarizer` Klasse gespeichert. Der Wert n ist frei wählbar, er muss jedoch ungerade sein, da immer mindestens ein Modell mit $k = \#Gruppierungen$ trainiert wird. Die Themenanzahl der einzelnen Modelle wird gleichmäßig über ein Intervall um die Anzahl der Quelltextgruppierungen verteilt. Die untere und obere Grenze dieses Intervalls kann im Voraus festgelegt werden. Da die `KeywordSummarizer` Klasse von `BaseModel` erbt, stellt sie ebenfalls eine `query` Methode bereit, welche eine Liste aller durch die n Modelle erzeugten Themen zurückgibt. Die Methode `get_keywords_for_cluster` berechnet eine Schlagwortmenge für eine gegebene Quelltextgruppierung und geht dabei wie folgt vor: Berechne für jedes der n Modelle die Themen der Gruppierung. Füge alle gefundenen Themen zu einer Liste hinzu und berechne die paarweisen Hellinger-Distanzen zwischen allen Themen. Anschließend wird über die Funktion `cluster_topics` ein agglomeratives Clustering initialisiert und auf die Liste der Themen angewandt. Das Clustering erzeugt Gruppierungen ähnlicher Themen aus den Ausgaben der einzelnen LDA Modelle. Diese Kombination dient wie in Abschnitt 5.6.3 beschrieben einer höheren Stabilität der Themen. Anschließend werden die Themen einer jeden solchen Gruppierung mittels der Funktion `calculate_token_overlap` und dem dynamischen Tokenauswahlverfahren (Funktion `select_percentage`) aus Abschnitt 5.6.2 kombiniert. Diese kombinierten Themen werden dann auf dieselbe Weise zu einem einzigen Thema kombiniert, welches die Schlagwörter für eine Quelltextgruppierung repräsentiert. In einem weiteren Schritt wird diese Schlagwortmenge in eine Menge von direkten Schlagwörtern und eine Menge von Kontextwörtern aufgespalten. Die Aufspaltung geschieht wie in Abschnitt 5.6.4 beschrieben durch bilden des Durchschnitts der Token der Gruppierung und der Schlagwortmenge. Nun werden die Token dieser Schlagwortmengen von ihrer Stammform wieder auf natürlichsprachliche Wörter rücktransformiert. Die Rücktransformation der Stammformreduktion wird ermöglicht, da bei der Stammformreduktion eines Tokens immer ein Eintrag in ein Wörterbuch gemacht wird, welcher festhält, wie häufig dieses Token bereits auf eine bestimmte Stammform reduziert wurde. Hierzu stellt die `Cluster` Klasse eine Methode namens `unstem_token` bereit. Die Parameter für all diese Schritte können über eine Konfigurationsdatei angepasst werden.

6.5 Datentransfer zwischen Java und Python

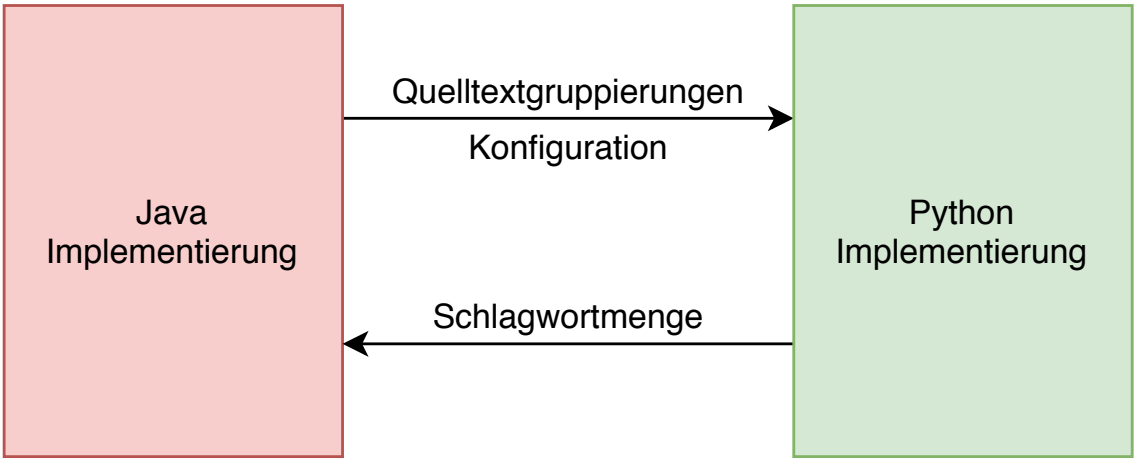


Abbildung 6.2: Schematische Darstellung des Datentransfers zwischen den zwei Implementierungen.

Der Datentransfer zwischen der Java-Implementierung und der Python-Implementierung geschieht über eine Json-serialisierte Repräsentation der `Corpus` Klasse der Java implementierung. Sie besteht aus eine Liste von Gruppierungen, welche jeweils aus einem eindeutigen Identifizierer, dem Zusammenhang der Gruppierung, der enthaltenen Programmelemente und deren verarbeitete Token. Diese Json Repräsentation wird über eine HTTP Anfrage an die Python-Implementierung geschickt. Zusätzlich schickt die Java-Implementierung eine Konfiguration an die Python-Implementierung über welche bestimmte Parameter des Themenmodells angepasst werden können. Konfiguriert werden können die obere und untere Grenze für den Skalierungsfaktor des k Parameters der einzelnen Modelle, die Anzahl an intern genutzten LDA-Modellen, die Schwellwerte für die Tokenauswahl bei der Rückkombination und der Gesamtschwellwert für die Auswahl der Schlagwörter aus dem entstehenden kombinierten Thema. Nachdem auf der Python Seite die Schlagwortmenge berechnet wurde, wird diese ebenfalls Json-serialisiert an die Java-Implementierung zurückgeschickt, wo die Informationen über die Schlagworte in den Quelltextgraphen übernommen werden. Abbildung 6.2 zeigt schematisch diesen Transfer und die transferierten Daten.

6.6 Schlagworte im Quelltextgraphen

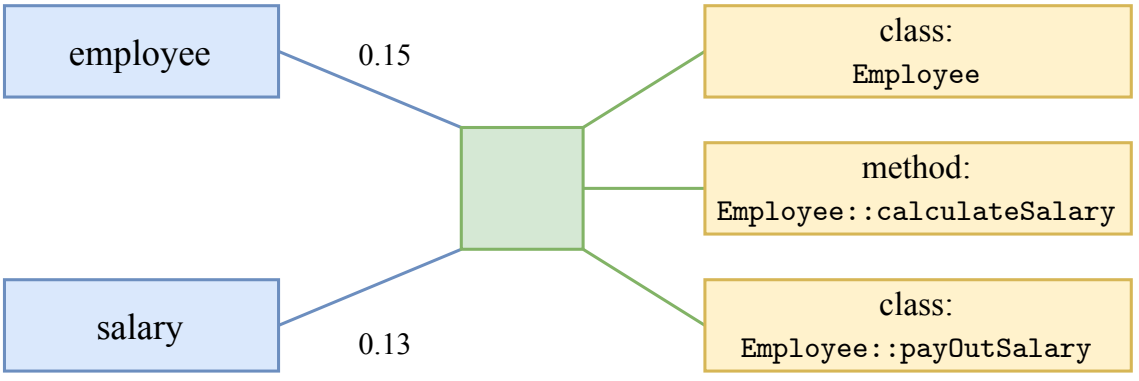


Abbildung 6.3: Schematische Darstellung des Datentransfers zwischen den zwei Implementierungen.

In einem letzten Schritt müssen die Ergebnisse der Python implementierung noch in den Quelltextgraphen eingetragen werden. Hierzu wird dem PARSE-Graphen ein Knotentyp `keyword` hinzugefügt. Dieser Knotentyp besitzt eine Eigenschaft (engl. *property*) namens `value`, welche dem Schlagwort entspricht. Alle Schlagwörter werden dann mittels einer Kante (engl. *Arc*) vom Typ `hasKeyword` mit dem entsprechenden Absichtsknoten verbunden, der die Quelltextgruppierung repräsentiert für die das Schlagwort gefunden wurde. Diese Kanten besitzen eine Eigenschaft namens `keywordProportion`, welche die Relevanz des Schlagwortes für die Quelltextgruppierung abbildet. Abbildung 6.3 zeigt diesen Sachverhalt anhand einer reduzierten Form der Bereits in Kapitel 3 eingeführten Darstellung des Quelltextabsichtsmodells. Der grün hinterlegte Knoten entspricht dem Absichtsknoten der Quelltextgruppierung. Die Programmelemente der Gruppierung sind in gelb dargestellt. Neu hinzugekommen sind die blau hinterlegten Knoten und Kanten, welche den `keyword` Knoten und `hasKeyword` Kanten entsprechen. Die Gruppierung beschäftigt sich mit dem Auszahlen von Gehältern an angestellte einer Firma. Die zwei gefundenen Schlüsselwörter „employee“ und „salary“ haben Anteile von je 0.15 und 0.13 an der Semantik der Gruppierung. Zu beachten ist, dass sich diese Anteile nicht zu 1 aufsummieren müssen, da nur Schlüsselwörter über einem vorher festgelegten Schwellwert übernommen werden.

7 Evaluation

In diesem Kapitel wird das in Kapitel 6 implementierte Themenmodell zur Erzeugung von Schlagwortmengen für Quelltextgruppierungen evaluiert. Um die Qualität der Schlagwortmengen zu evaluieren wird eine Nutzerstudie, bestehend aus zwei Aufgaben, durchgeführt. Hierzu werden Quelltextgruppierungen für zwei Java-Projekte erzeugt, welche die Datengrundlage für die Studie bilden. Die Evaluation findet als Nutzerstudie statt, da wir keinen Goldstandard für Beschreibungen der durch INDIRECT erzeugten Quelltextgruppierungen vorliegen haben und somit viele Standardmetriken nicht direkt angewandt werden können. Die manuelle Erstellung eines solchen Goldstandards ist schwierig, da nicht offensichtlich ist welche Beschreibung für die Semantik einer Quelltextgruppierung eindeutig und optimal ist. Um den Goldstandard für einzelne Quelltextgruppierungen zu approximieren werden sich die Nutzer im Rahmen der ersten Evaluationsaufgabe in die INDIRECT Quelltextgruppierungen einarbeiten und diese anschließend beschreiben. In der zweiten Aufgabe bewerten sie die Gruppierungsbeschreibungen, welche von dem in Kapitel 6 implementierten Modell erzeugt wurden. In diesem Kapitel wird zunächst der Evaluationskorpus, bestehend aus zufällig ausgewählten Quelltextgruppierung zweier Java-Projekte, genauer vorgestellt. Anschließend wird auf das Studiendesign, besonders im Hinblick auf den Aufbau der zwei Aufgaben, sowie eine Strategie zur Vermeidung von Reihenfolgeeffekten eingegangen. Dann werden die Ergebnisse der Evaluation ausgewertet und vorgestellt. Hierbei werden die Ergebnisse der gesamten Nutzerstudie ausgewertet und es werden die folgenden Evaluationsfragen beantwortet:

(EF1) Wie hoch ist die Übereinstimmung der von unserem Modell erzeugten Gruppierungsbeschreibungen mit den nutzergenerierten Beschreibungen? Diese Evaluationsfrage soll klären, wie ähnlich unsere Gruppierungsbeschreibungen zu den durch die Nutzer generierten Beschreibungen sind.

(EF2) Wie passend sind die Gruppierungsbeschreibungen für die getesteten Quelltextgruppierungen? Im Rahmen dieser Frage soll geklärt werden, ob Abweichungen zwischen den Nutzerbewertungen für Gruppierungsbeschreibungen und den Bewertungen der Beschreibungen durch Metriken feststellbar sind.

(EF3) Gibt es einen Zusammenhang zwischen der Anzahl an Programmelementen einer Gruppierung und der Qualität der Beschreibungen der Nutzer und unseres Modells? Auf Grund der hohen kognitiven Last, die mit langen Quelltextabschnitten einhergeht, ist zu erwarten, dass die Bewertungsleistung der Nutzer mit der Anzahl an Programmelementen und Token in einer Gruppierung abnimmt. Im Rahmen dieser

Evaluationsfrage soll sowohl diese Vermutung, als auch die Leistung unseres Modells für Quelltextgruppierungen verschiedener Größen überprüft werden.

Anschließend wird auf einige ausgewählte Quelltextgruppierungen und Erkenntnisse, welche aus deren Auswertung hervorgingen, eingegangen. Zuletzt wird die Evaluation zusammengefasst.

7.1 Evaluationskorpus

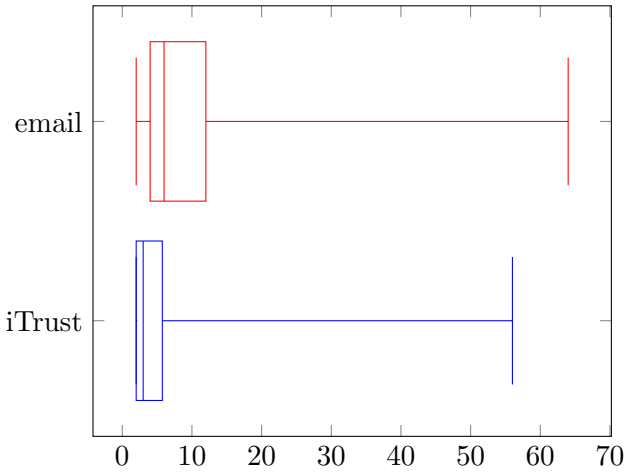


Abbildung 7.1: Darstellung der Anzahl an Programmelementen in den Quelltextgruppierungen von iTrust und apache-commons-email als Kastendiagramme.

Der Evaluationskorpus besteht aus den Quelltextgruppierungen für die zwei Java-Projekte **iTrust**[1] und **apache-commons-email**[2]. iTrust ist ein Verwaltungssystem für Gesundheitsdaten in Form einer Webanwendung. apache-commons-email ist eine Java-Bibliothek zum Versenden von E-mails.

iTrust wurde ausgewählt, da das Projekt gut dokumentiert ist und eine starke Domänenzugehörigkeit hat. apache-commons-emails wurde ausgewählt, da Apache[3] Projekte einem hohen Qualitätsstandard unterliegen und der Quelltext, sowie seine Dokumentation, somit von hoher softwaretechnischer Qualität sind. Gute Dokumentation ist ein wichtiges Merkmal da Quelltextkommentare, neben Bezeichnern, einen großen Anteil der natürlichen Sprache im Quelltext ausmachen. Durch das Testen von Projekten aus unterschiedlichen Domänen und mit starker Domänenbindung kann festgestellt werden, wie gut das Modell mit domänenspezifischen Begriffen umgehen kann und ob der gewählte Ansatz domänenübergreifend anwendbar ist..

Um den Evaluationskorpus zu erstellen, wurden beide Softwareprojekte mithilfe des Quelltextgruppierungsverfahrens von Eurich [Eur] gruppiert. Anschließend wurden die in Kapitel 6 implementierten Vorverarbeitungsschritte auf die Quelltextgruppierungen angewandt. Aus den Quelltextgruppierungen beider Projekte werden jeweils 9 Gruppierungen zufällig ausgewählt. Die Festlegung auf 9 Gruppierungen pro Projekt stellt eine Abwägung des Bearbeitungsaufwands der Studie für die Nutzer und der Aussagekraft des Evaluationskorpus dar. Diese Gruppierungen werden den Nutzern im Rahmen der Evaluationsaufgaben präsentiert. Für beide Projekte wurden je drei Gruppierungen aus den unteren 10%, den

[1] https://sourceforge.net/projects/itrust/
[2] https://commons.apache.org/proper/commons-email/
[3] https://www.apache.org/

Projekt	l_{code}	l_{com}	$\|V\|$	n_{stems}	p_{min}	p_{max}	p_{mean}	p_{med}
iTrust	14573 (62%)	6338 (27%)	939	708 (-25%)	2	56	5.01	3
email	6776 (54%)	4177 (33%)	624	483 (-23%)	2	64	8.99	6

Tabelle 7.1: Überblick über die Projekte im Evaluationskorpus.
l_{code} und l_{com}: Anzahl an Quelltext- und Kommentarzeilen des Projekts.
$|V|$: Anzahl Wörter im Vokabular der Gruppierungsmenge.
n_{stems}: Anzahl einzigartiger Wortstämme.
$p_{min}, p_{max}, p_{mean}$ und p_{med}: minimale, maximale und durchschnittliche Anzahl an Programmelementen und Median.

	G_i	$\|G_i\|$	$\|token(G_i)\|$	$\|supp\ token(G_i)\|$	$\|supp\ stems(G_i)\|$
klein	iTrust 1	2	15	10	9
	iTrust 2	2	15	11	10
	iTrust 3	2	21	12	11
mittel	iTrust 4	4	24	12	12
	iTrust 5	3	13	7	7
	iTrust 6	3	40	20	19
groß	iTrust 7	34	188	23	23
	iTrust 8	13	86	16	15
	iTrust 9	15	165	30	28

Tabelle 7.2: Überblick über die Metadaten der iTrust-Gruppierungen.
G_i: Gruppierung des iTrust-Projekts.
$|token(G_i)|$: Anzahl Token in der Gruppierung G_i.
$|supp\ token(G_i)|$: Anzahl einzigartiger Token in G_i.
$|supp\ stems(G_i)|$ Anzahl einzigartiger Wortstämme G_i.

	G_i	$\|G_i\|$	$\|token(G_i)\|$	$\|supp\ token(G_i)\|$	$\|supp\ stems(G_i)\|$
klein	email 1	3	21	11	11
	email 2	3	24	7	6
	email 3	3	40	22	17
mittel	email 4	7	59	19	18
	email 5	5	40	13	12
	email 6	6	203	82	74
groß	email 7	18	303	39	33
	email 8	16	357	79	74
	email 9	16	162	55	50

Tabelle 7.3: Überblick über die Metadaten der apache-commons-email-Gruppierungen.
G_i: Gruppierung des iTrust-Projekts.
$|token(G_i)|$: Anzahl Token in der Gruppierung G_i.
$|supp\ token(G_i)|$: Anzahl einzigartiger Token in G_i.
$|supp\ stems(G_i)|$ Anzahl einzigartiger Wortstämme G_i.

mittleren 50% und den oberen 10% der Gruppierungsmenge nach der Anzahl an Programmelementen in den Gruppierungen ausgewählt. In der Abbildung 7.1 sind die Quelltextgruppierungen beider Projekte als Kastendiagramme dargestellt. Anhand der Diagramme kann man erkennen, dass der Mittelwert der Anzahl an Programmelementen bei beiden Projekten unter 10 liegt. Außerdem liegen das obere Quartil und der Maximalwert beider Kastendiagramme weit auseinander, was darauf schließen lässt, dass nur wenige Quelltext-

gruppierungen eine sehr hohe Anzahl an Programmelementen enthält. Insgesamt bedeutet das, dass die Anzahl an Programmelementen in den Quelltextgruppierungen sehr ungleichmäßig verteilt ist: Der Großteil der Quelltextgruppierungen weist eine ähnlich hohe Anzahl an Programmelementen auf, allerdings beinhalten beide Gruppierungsmengen einige wenige Ausreißer mit einer verhältnismäßig sehr hohen Anzahl an Programmelementen. Durch das gewählte Aufteilungs- und Auswahlverfahren wird sichergestellt, dass die in der Nutzerstudie präsentierten Quelltextgruppierungen diese Eigenschaften der Gruppierungsmengen wiederspiegeln. Außerdem kann evaluiert werden, wie sich die Größe der Quelltextgruppierungen auf die Qualität der Nutzerbeschreibungen, aber auch auf die Beschreibungen des im Rahmen dieser Arbeit implementierten Themenmodells auswirkt. Metadaten der Gruppierungen des iTrust Projekts finden sich in Tabelle 7.2 und Metadaten für die Gruppierungen von apache-commons-email finden sich in Tabelle 7.3. Tabelle 7.1 zeigt Metadaten für beide Softwareprojekte und deren Quelltextgruppierungen. Unter anderem fällt auf, dass die Stammformreduktion das Vokabular beider Projektgruppierungen um etwa 25% reduziert hat.

Der Tabelle lässt sich entnehmen, dass das iTrust Projekt aus fast doppelt so vielen Quelltextzeilen wie das apache-commons-email Projekt besteht. Ein möglicher Grund hierfür ist die Tatsache, dass es sich bei iTrust um eine komplette Anwendung zur Verwaltung von Gesundheitsdaten handelt, welche eine Vielzahl an Funktionalitäten und eine Nutzeroberfläche implementiert. apache-commons-email hingegen ist eine stark spezialisierte Softwarebibliothek, welche lediglich Funktionalität für das Verarbeiten und Versenden von E-Mails bereitstellt. Das Verhältnis von Kommentarzeilen zu Quelltextzeilen ist bei apache-commons-email höher, als bei iTrust. Ein Grund hierfür könnte sein, dass es sich bei iTrust um eine Anwendung und bei apache-commons-email um eine Bibliothek handelt. Softwarebibliotheken sind meist besser und vollständiger dokumentiert als Anwendungen, da sie an viele Entwickler verteilt werden und diese sich in den Quelltext einarbeiten müssen. Sowohl für iTrust, als auch apache-commons-email beträgt $p_{min} = 2$, da eine Quelltextgruppierung immer aus mindestens zwei Programmelementen besteht. Die maximale Anzahl an Programmelementen p_{max} beträgt je 56 und 64. Die Tatsache, dass in beiden Fällen $p_{med} < p_{mean}$ gilt bestätigt, dass ein großer Teil der Gruppierungen wenige Elemente beinhaltet, es allerdings einige Ausreißer mit sehr vielen Programmelementen gibt, die den Durchschnittswert anheben.

7.2 Studiendesign

Die Evaluation des im Rahmen dieser Arbeit implementierten Themenmodells wird in Form einer Nutzerstudie durchgeführt. Die Nutzer werden je zwei Aufgaben bearbeiten, wobei in beiden Aufgaben je 9 Quelltextgruppierungen bearbeitet werden. Im Rahmen der ersten Aufgabe müssen Nutzer Quelltextgruppierungen durch eigene Schlagwörter beschreiben, in der zweiten Aufgabe bewerten sie die von INDIRECT erzeugten Schlagwortmengen. Die Nutzer werden für die Evaluation in zwei Gruppen eingeteilt, welche dieselben Aufgaben, jedoch auf Basis unterschiedlicher Quelltextgruppierungen erhalten. Im Folgenden werden zunächst beide Aufgaben detailliert vorgestellt. Danach wird darauf eingegangen, weshalb die Nutzer in zwei Gruppen aufgeteilt werden und weshalb diese Aufteilung sinnvoll ist.

7.2.1 Durchführung der Studie

Die Evaluationsstudie wird über ein Google-Formular[4] durchgeführt. Sowohl Gruppe A, als auch Gruppe B haben 8 Teilnehmer. Die Studie wurde sowohl an Studenten der Informatik,

[4] https://www.google.com/forms

als auch berufstätige Softwareentwickler, welche sich in ihrem Beruf mit Java beschäftigen, verteilt. Im Anhang unter Abschnitt A befinden sich schematische Darstellungen beider Formulare.

7.2.2 Repräsentation von Quelltextgruppierungen

```
/**
* Class representing an employee in a company
*/
public class Employee {
    private long employeeID;

    /**
    * Constructs a new employee with a given ID
    */
    public Employee(long id);

    /**
    * Gets the contract associated with this employee
    */
    public Contract getContract();
}

public class Contract {
    /**
    * Gets the start date for this contract
    */
    public Date getStartDate();

    /**
    * Gets the end date for this contract
    */
    public Date getEndDate();
}
```

Quelltextausschnitt 7.1: Beispiel der in der Evaluation genutzten Repräsentation von Quelltextgruppierungen

Für die Bearbeitung beider Evaluationsaufgaben müssen die Nutzer die Möglichkeit haben den Quelltext der Gruppierung die sie bearbeiten zu lesen und zu verstehen. Es muss also eine geeignete Repräsentation gefunden werden, um die Quelltextgruppierungen dem Nutzer vorzulegen. Wichtig ist dabei, dass möglichst alle Merkmale, welche von unserem Modell zur Erzeugung von Beschreibungen genutzt werden auch dem Nutzer vorliegen sollen. Da die Teilnehmer der Nutzerstudie mit dem Lesen von Quelltext vertraut sind, bietet sich eine Repräsentation durch verkürzten Java Quelltext an. Eine starke Reduzierung der Länge des Quelltextes ohne Verlust zu vieler Informationen kann durch weglassen der Methodenrümpfe erreicht werden. Wie in Kapitel 5 bereits festgelegt, sind die Bezeichner von Variablendefinitionen die einzige Tokenquelle aus dem inneren von Methodenrümpfen, welche in die Erzeugung von Beschreibungen einfließt. Um besonders Gruppierungen mit vielen Quelltextelementen überschaubar zu halten, werden Methoden ohne ihren Rumpf und somit ohne die enthaltenen Variablendefinitionen präsentiert. Hierbei ist zu beachten, dass die verringerte kognitive Last des Quelltextes auf den Nutzer dazu führen kann, dass der Nutzer bessere Beschreibungen erzeugt. Gleichzeitig stellt das Entfernen des Methodenrumpfs auch eine Gefahr dar, da die Nutzer durch den resultierenden Informationsverlust eventuell Gruppierungen nicht mehr adäquat beschreiben können. Quelltextausschnitt 7.1 zeigt ein Beispiel einer Quelltextgruppierung in dieser verkürzten Form. Ein Methodenelement in einer Gruppierung wird durch eine Methodensignatur

in ihrer umschließenden Klasse, sowie dem optionalen Javadoc-Kommentar der Methode repräsentiert. Liegt zusätzlich die umschließende Klasse selbst als Element in der Gruppierung vor, so werden ihre Konstruktoren und deren Javadoc-Kommentare, die Felder der Klasse, und der Javadoc-Kommentar der Klasse selbst ebenfalls in die Repräsentation übernommen. Die Dargestellte Gruppierung enthält also die Klasse `Employee`, sowie deren Methode `Employee::getContract` und die Methoden `Contract::getStartDate` und `Contract::getEndDate`, jedoch nicht die Klasse `Contract` selbst.

7.2.3 Aufbau der Aufgaben

Im Rahmen beider Aufgaben arbeiten die Nutzer mit Quelltextgruppierungen. In diesem Abschnitt soll zunächst geklärt werden, in welcher Form die Gruppierungen den Nutzern präsentiert werden können. Hierbei sind Informationsgehalt, Lesbarkeit und einfaches Verständnis der Gruppierungen wichtige Eigenschaften einer guten Repräsentation. Anschließend werden beide Aufgaben im Detail eingeführt.

7.2.3.1 Beschreibungsaufgabe

Im Rahmen der ersten Aufgabe sollen die Nutzer Beschreibungen für Quelltextgruppierungen finden. Dazu erhalten sie insgesamt 9 Quelltextgruppierungen in der eben eingeführten Repräsentation. Die Nutzer sollen bis zu 10 Schlagwörter identifizieren, welche ihrer Meinung nach die geteilte Semantik der Quelltextgruppierung am besten beschreiben. Die Wörter, welche die Nutzer hierfür verwenden, können aus dem Quelltext der Gruppierungen stammen, müssen dies aber nicht. Die Schlagwörter, welche die Nutzer erarbeiten, werden als kommaseparierte Liste angegeben. Diese Aufgabe wurde ausgewählt, da die von den Nutzern erzeugten Beschreibungen verwendet werden können um Musterlösungen für Gruppierungsbeschreibungen zu approximieren. Die Idee dahinter ist es die übereinstimmenden Wörter aller eingereichten Beschreibungen für eine Gruppierung als Musterlösung zu verwenden. Eine solche Approximation kann dann genutzt werden um zu vergleichen wie ähnlich die von unserem Modell erzeugten Beschreibungen zu den Beschreibungen der Nutzer sind. Da die Nutzer ihre Beschreibungen als Freitext formulieren, müssen die Nutzerantworten zuerst normalisiert werden, bevor ein Konsens daraus gebildet werden kann. Das bedeutet, dass beispielsweise verschiedene Wortformen, welche die selbe Semantik ausdrücken, zusammengefasst werden müssen. Genaueres zu diesem Normalisierungsschritt folgt im Abschnitt Auswertung 7.3. Definition 7.1 führt Beispiel 7.3 zeigt anhand der bereits eingeführten Quelltextgruppierung 7.1 wie dieser Konsens aus den Nutzerantworten gewonnen wird.

Als Teil der Beschreibungsaufgabe müssen die Nutzer ihre Antworten als Freitext angeben. Diese Tatsache führt dazu, dass die Antworten der Nutzer ein gewisses Rauschen aufweisen, da verschiedene Nutzer unterschiedliche Schlagwörter für das gleiche Konzept angeben können. Beispiel 7.1 zeigt an konkreten Beispielen aus der Evaluationsstudie wieso eine solche Normalisierung sinnvoll ist und wie sie abläuft. Auch Rechtschreibfehler können die Nutzerantworten voneinander abweichen lassen. Hier gilt zu beachten, dass Rechtschreibfehler sowohl in den Freitextantworten der Nutzer, aber auch im Quelltext, welcher Ausgangspunkt für die Gruppierungen war, auftreten können. In Beispiel 7.2 wird verdeutlicht, dass auch Fehler im Ausgangsquelltext die Leistung des Modells beeinflussen können. Die Normalisierung ist also ein wichtiger Schritt um das Rauschen in den Nutzerantworten zu reduzieren und das Bilden eines Konsens zu ermöglichen. Gleichzeitig stellt sie allerdings auch eine Gefahr dar, da durch das Normalisieren eventuell ein Teil der Absicht des Nutzers verloren gehen kann. Aus diesem Grund muss die Normalisierung sehr konservativ vorgenommen werden um das Ergebnis der Evaluation nicht zu beeinflussen.

Abbildung A.1 im Anhang zeigt den schematischen Aufbau des Fragebogens dieser Aufgabe.

Beispiel 7.1: Normalisieren von Nutzerantworten

Ein Teilnehmer der Nutzerstudie verwendete die Schreibweise „e-mail“, während die anderen Nutzer die Schreibweise „email“ nutzten. Da der Großteil der Nutzer die zweite Schreibweise nutze, wird „e-mail“ zu „email“ normalisiert. Ein weiteres Beispiel für das Rauschen, welches in den Nutzerantworten vorliegt sind die Wörter „resolve“, „resolver“ und „resolution“. Diese Wörter treten häufig gemeinsam in den Antworten für Gruppierungen auf, deren Semantik das Auflösen (engl. „resolution“) von Pfaden oder Dateien ist. In diesem Fall ist nicht offensichtlich, welches dieser Wörter korrekt ist, da sie alle dieselbe Semantik beschreiben.

Beispiel 7.2: Rechtschreibfehler in Quelltext

Die Entwickler des apache-commons-email Projekts nutzen in ihren Bezeichner die Schreibweise `ClassPath`. Auf Grund dieser Binnenmajuskelschreibweise wird das Wort während der Vorverarbeitung 5.4 in `class` und `path` aufgespalten. Die korrekte Schreibweise ist jedoch nicht „class path“ sondern „classpath“, da es sich um ein Eigenwort handelt. Durch diesen Fehler der Entwickler ist es dem Modell nicht möglich „classpath“ als Schlagwort zu erzeugen, sondern lediglich „class“ und „path“ getrennt.

Definition 7.1: Entstehung des Nutzerkonsens

Die Menge $R_{proj,i} = \{(t_1, p_1), \ldots, (t_n, p_n)\}$ beinhaltet die Wörter aller Nutzerantworten für die i-te Gruppierung des Projekts $proj$, sowie deren Nennungsrate. Die Menge $N_{proj,i} = normalize(R_{proj,i})$ entsteht durch Normalisierung der Menge $R_{proj,i}$. Die Menge $C_{proj,i} = \{(t, p) \in N_{proj,i} | p >= 0.5\}$ bezeichnet den Nutzerkonsens, welcher aus der Menge der normalisierten Nutzerantworten entsteht.

Beispiel 7.3: Nutzerkonsens

Gegeben sei die Quelltextgruppierung aus Quelltextausschnitt 7.1. Tabelle 7.4 zeigt die Wörter, welche die Nutzer zur Beschreibung der Semantik der Gruppierung angegeben haben. Die Wörter, welche von mehr als 50% der Teilnehmer angegeben wurden sind fett hervorgehoben und werden als Konsens angesehen. Unter dieser Annahme können die Wörter **employee**, **contract**, **start** und **end** für die weitere Evaluation als Approximation einer Musterlösung angesehen werden.

Wort	Anzahl
employee	**6/6**
contract	**5/6**
start	**3/6**
end	**3/6**
id	2/6
get	1/6

Tabelle 7.4: Beispiel normalisierter Nutzerantworten für die Gruppierung aus 7.1. Der Konsens ist fett hervorgehoben.

7.2.3.2 Bewertungsaufgabe

In der zweiten Aufgabe müssen die Nutzer Quelltextgruppierungen bewerten. Hierzu erhalten sie wie in der ersten Aufgabe 9 Quelltextgruppierungen in der eingeführten Repräsentation. Sollten die Nutzer in der vorherigen Aufgabe die Quelltextgruppierungen von iTrust bearbeitet haben, so bearbeiten sie in den zweiten Aufgabe die Gruppierungen von apache-commons-email und umgekehrt. Der Grund für diese Aufteilung ist wie bereits erwähnt die Vermeidung von Reihenfolgeeffekten und wird später in einem eigenen Abschnitt thematisiert. Anstatt nun eigene Beschreibungen zu finden, erhalten die Nutzer zusätzlich zu jeder Gruppierung die von unserem Themenmodell erzeugte Schlagwortmenge und sollen deren Güte bewerten. Zunächst sind wird an der Relevanz der einzelnen Wörter für die Semantik der Gruppierung interessiert. Hierzu bewerten die Nutzer die einzelnen Schlagwörter mit einer von drei Bewertungen. Sie haben die Möglichkeit die Relevanz eines Schlagwortes als „gut“ (es besteht ein direkter Zusammenhang zwischen dem Wort und der Semantik der Gruppierung), „neutral“ (es besteht ein indirekter Zusammenhang zur Semantik der Gruppierung) oder „schlecht“ (das Wort steht in keinem Zusammenhang mit der Semantik der Gruppierung) zu bewerten. Zusätzlich sollen die Teilnehmer die Schlagwortmenge in ihrer Gesamtheit dahingehend bewerten, wie stark die Übereinstimmung der Semantik der Gruppierung und der Schlagwortmenge ist. Hierzu können die Nutzer die Übereinstimmung mit 1 (keinerlei Übereinstimmung) bis 5 (perfekte Übereinstimmung) bewerten. Diese Aufgabe wurde ausgewählt um nicht nur die Beschreibungen der Nutzer mit unseren eigenen vergleichen zu können, sondern um zusätzlich eine direkte Bewertung unserer Beschreibungen durch die Nutzer zu erhalten.

Abbildung A.2 im Anhang zeigt den schematischen Aufbau des Fragebogens dieser Aufgabe.

7.2.4 Aufteilung der Nutzer in zwei Gruppen

Für die Evaluation werden die Nutzer in zwei Gruppen aufgeteilt. Gruppe A bearbeitet in der ersten Aufgabe 9 Gruppierungen des iTrust-Projekts und in der zweiten Aufgabe 9 Gruppierungen des apache-commons-email-Projekts. Gruppe B bearbeitet dieselben Gruppierungen wie Gruppe A, jedoch werden die apache-commons-email-Gruppierungen für die erste und die iTrust-Gruppierungen für die zweite Aufgabe genutzt. Durch das Bearbeiten der Beschreibungsaufgabe können die Nutzer in der zweiten Aufgabe beeinflusst werden. Durch die eigenständige Auswahl wichtiger Wörter wird der Nutzer unterbewusst dazu verleitet in der Bewertungsaufgabe Wörter, welcher er selbst zuvor gewählt hat, als wichtiger einzustufen, als Wörter, welcher er nicht gewählt hat. Umkehren der Reihenfolge der Aufgaben würde dieses Problem nur umkehren, es jedoch noch verschlimmern, da sich der Nutzer dann unterbewusst von INDIRECT erzeugte Wörter merken kann und eventuell bei der Wortfindung in seiner Entscheidungsfreiheit eingeschränkt wird. Dieses Problem wird durch die Aufteilung in zwei Gruppen gelöst, da keine Gruppe dieselben Quelltextgruppierungen in beiden Aufgaben bearbeitet und somit eine Beeinflussung durch vorheriges Arbeiten mit der Gruppierung ausgeschlossen ist.

7.3 Auswertung

In diesem Kapitel wird auf die konkreten Ergebnisse der Evaluationsstudie eingegangen und diese werden ausgewertet. Zunächst werden die Metriken festgelegt, mit denen die Studie ausgewertet wird. Anschließend wird auf ausgewählte Quelltextgruppierungen und deren Evaluationsergebnisse eingegangen. Diese Gruppierungen wurden ausgewählt, da sie besondere Einblicke in die Erzeugung der Themenbeschreibungen ermöglichen. Danach wird die Studie als Ganzes ausgewertet und es werden die zu Beginn definierten Evaluationsfragen beantwortet.

7.3.1 Festlegung der Evaluationsmetriken

In diesem Abschnitt werden die Metriken zur Evaluation der Nutzerstudie festgelegt. Hierbei kann man zwischen Metriken unterscheiden, welche die Güte der Nutzerbeschreibungen, die Güte der Beschreibungen unseres Modells und den Grad der semantischen Übereinstimmung zwischen den beiden bewerten.

7.3.1.1 Bewertung der Schlagwortmengen

Durch die Bildung eines Konsens mithilfe der normalisierten Nutzerantworten können wir die aus Abschnitt 2.9 bekannten Metriken auf die von unserem Modell erzeugten Gruppierungsbeschreibungen anwenden. Insbesondere werden die Präzision, die Ausbeute und der F1-Wert berechnet. Diese Metriken werden sehr häufig zur Evaluation von Sprach- und Themenmodellen eingesetzt. Um diese zu erläutern werden die vier möglichen Fälle der Wahrheitsmatrix noch ein mal konkret für Schlagwortmengen eingeführt:

1. **Richtig positiv (TP):** Das Wort ist sowohl Teil der von unserem Modell erzeugten Schlagwortmenge, als auch dem Konsens der Nutzer.
2. **Falsch positiv (FP):** Das Wort ist Teil der von unserem Modell erzeugten Schlagwortmenge, jedoch nicht des Konsens der Nutzer.
3. **Falsch negativ (FN):** Das Wort ist Teil des Konsens der Nutzer, jedoch nicht der von unserem Modell erzeugten Schlagwortmenge.
4. **Richtig negativ (TN):** Das Wort ist weder Teil der von unserem Modell erzeugten Schlagwortmenge, noch des Konsens der Nutzer.

Die Anzahl an richtig negativen Wörtern kann berechnet werden, indem man alle nicht durch das Modell genannten Worte des Vokabulars betrachtet, die auch nicht im Nutzerkonsens vorkommen. Da die Anzahl von vorhergesagten Wörtern pro Gruppierung im Vergleich zu der Anzahl an Wörtern im Vokabular insgesamt sehr klein ist, würde der **TN** Fall die Wahrheitsmatrix dominieren und die Genauigkeitsmetrik würde ihre Aussagekraft verlieren. Aus diesem Grund werden in der Evaluation nur die Präzision, die Ausbeute, sowie ihr harmonisches Mittel, der F1-Wert, betrachtet.

Definition 7.2: TP und FP Mengen

Gegeben sei eine Gruppierungsbeschreibung $T = \{t_1, \ldots, t_n\}$ mit n Wörtern. Die Mengen $T_{TP} = \{t \in T | t \in K\}$ und $T_{FP} = \{t \in T | t \notin K\}$ beinhalten dann respektive die TP und FP Wörter der Beschreibung.

Zusätzlich zu diesen Metriken können aus den Evaluationsdaten noch einige andere Kennzahlen abgelesen werden:

Mithilfe der Bewertungen für die einzelnen Wörter einer Gruppierungsbeschreibung aus Abschnitt 7.2.3.2 kann quantifiziert werden, wie gut die Nutzer die Wörter im Schnitt bewertet haben. Hierzu wird zunächst jeder schlechten Bewertung eine 0, jeder neutralen Bewertung eine 1 und jeder guten Bewertung eine 2 zugeordnet. Die Funktion *tokenScore* aus Formel 7.2 ordnet einer Nutzerbewertung von einem Wort diesen Wert zu. Anschließend werden diese Werte aufsummiert und es wird ihr Durchschnitt gebildet. Ein Schnitt nahe bei 0 bedeutet eine durchschnittlich schlechte, bei 1 eine neutrale und bei 2 eine gute Bewertung.

Weiter kann diese Metrik genutzt werden um die durchschnittliche Bewertung für alle Wörter der Gruppierung, nur für die richtig positiven Wörter oder nur für die falsch

$$tokenScore(t_{i,n}) = \begin{cases} 0, & \text{schlechte Bewertung} \\ 1, & \text{neutrale Bewertung} \\ 2, & \text{gute Bewertung} \end{cases} \tag{7.1}$$

Abbildung 7.2: Funktion zur Zuordnung von Werten zu Bewertungen. Der Index i entspricht dem i-ten Wort der Quelltextbeschreibung und n der Bewertung durch den n-ten Nutzers.

positiven Wörter berechnet werden. Durch diese Aufteilung kann quantifiziert werden, ob die falsch positiven Wörter dennoch für die Semantik der Gruppierung relevant sind.

Mithilfe dieser Metriken können dann ebenfalls Durchschnittswerte für die Gruppierungen von iTrust und apache-commons-email, den drei Abstufungen nach Anzahl der Programmelemente oder dem gesamten Evaluationskorpus berechnet werden. Die Ergebnisse für die Gruppierungen des iTrust Projekts sind in Tabelle 7.9 eingetragen und in Abbildung 7.3 visualisiert. Die Werte für apache-commons-email finden sich in Tabelle 7.10 und sind in Abbildung 7.4 dargestellt. Schließlich sind die Ergebnisse für den gesamten Evaluationskorpus in Tabelle 7.11 vermerkt.

7.3.1.2 Gütemetrik für den Nutzerkonsens

Mithilfe der bereits definierten Metriken können wir Aussagen über die Güte von Quelltextgruppierungen treffen. Es ist uns möglich, sowohl die Zugehörigkeit von Wörtern zu einer Gruppierung zu bewerten, als auch die Übereinstimmung der Semantik der Beschreibung und der Gruppierung als Ganzes. Im Rahmen dieser Metriken spielt der Nutzerkonsens eine wichtige Rolle, da er es uns erst ermöglicht die bekannten Metriken wie Präzision oder Ausbeute zu berechnen. Folgende Überlegungen führen zu den Metriken für den Nutzerkonsens:

Je größer die Anzahl an insgesamt genannten Wörtern im Vergleich zu der Anzahl an Wörtern im Konsens ist, desto schwieriger ist es den Nutzern gefallen sich auf den Konsens zu einigen, hieraus ergibt sich Formel 7.2a. Je häufiger die Wörter im Konsens genannt wurden, desto leichter fiel es allen Nutzern sich auf den Konsens zu einigen. Hier gilt zu beachten, dass der Wertebereich für die durschnittliche Bewertung innerhalb des Konsens durch die Festlegung der Untergrenze des Konsens zwischen 0.5 und 1 liegt. Um eine korrekte Kombination mit dem *outerScore* zu ermöglichen muss dieser Durchschnittswert erst normiert werden. Diese Normierung führt zu der in Formel 7.2b dargestellten Konsensmetrik *innerScore*. Die Gesamtbewertung *totalScore* wird als harmonisches Mittel der Metriken *outerScore* und *innerScore* definiert. Da die Nutzer Freitextantworten einreichen und das Verständnis der Semantik der Gruppierungen von Nutzer zu Nutzer unterschiedlich sein kann, ist es sehr unwahrscheinlich, dass für *totalScore* ein Wert von 1 erreicht wird. Dies würde voraussetzen, dass zum einen alle eingereichten Wörter teil des Konsens sind und dieser Konsens darüber hinaus einstimmig ist.

$$outerScore_{p,i} = \frac{|C_{p,i}|}{|N_{p,i}|} \tag{7.2a}$$

$$innerScore_{p,i} = 2(\frac{\sum_{(t,p)\in C_{p,i}} p}{|C_{p,i}|} - 0.5) \tag{7.2b}$$

$$totalScore = 2 \cdot \frac{outerScore \cdot innerScore}{outerScore + innerScore} \tag{7.2c}$$

7.3.2 Gesamtergebnis

Im folgenden Abschnitt werden die Ergebnisse der Evaluation ausgewertet. Es werden nun der Reihe nach die bereits zu Beginn dieses Kapitels definierten Evaluationsfragen beantwortet. Abschließend wird auf einige Quelltextgruppierungen genauer eingegangen, bevor die Evaluation zusammengefasst wird.

7.3.2.1 (EF1): Übereinstimmung von Nutzern und Modell

Zunächst ist für uns interessant, inwiefern die von unserem Modell erzeugten Beschreibungen mit den von den Nutzern erzeugten Beschreibungen übereinstimmen. Hierzu werden zum einen die aus dem Nutzerkonsens berechneten Metriken (Präzision, Ausbeute und F1-Wert), sowie die Konsensbewertung für die Nutzerantworten verwendet.

	G_i	pr_i	pr_i o. K.	re_i	re_i o. K.	$F1_i$	$F1_i$ o. K.	*cons*
klein	iTrust 1	0.60	**1.00**	**1.00**	0.71	0.75	**0.83**	0.58
	iTrust 2	0.66	**1.00**	**0.66**	0.30	**0.66**	0.46	0.38
	iTrust 3	0.43	**0.57**	**0.75**	0.44	**0.54**	0.50	0.59
mittel	iTrust 4	0.11	**0.44**	**0.50**	0.33	0.18	**0.38**	0.27
	iTrust 5	0.40	0.40	**1.00**	0.18	**0.57**	0.25	0.26
	iTrust 6	0.25	**0.50**	**0.50**	0.40	0.33	**0.44**	0.42
groß	iTrust 7	0.20	**0.60**	**1.00**	0.16	**0.33**	0.26	0.11
	iTrust 8	0.16	**0.33**	**1.00**	0.14	**0.29**	0.20	0.13
	iTrust 9	0.25	**0.50**	**1.00**	0.29	**0.40**	0.36	0.22

Tabelle 7.5: Präzision (pr_i), Ausbeute (re_i) und F1-Wert ($F1_i$) der iTrust Gruppierungen mit und ohne Einbeziehung des Konsens (o. K.). Der höhere beider Werte ist jeweils fett hervorgehoben. *cons* bezeichnet die Konsensbewertung für die jeweilige Gruppierung

G_i	pr_i	pr_i o. K.	re_i	re_i o. K.	$F1_i$	$F1_i$ o. K.	*cons*
klein	0.53	**0.80**	**0.80**	0.46	**0.64**	0.59	0.58
mittel	0.23	**0.45**	**0.63**	0.30	0.33	**0.36**	0.38
groß	0.21	**0.47**	**1.00**	0.20	**0.35**	0.28	0.38
Gesamt	0.30	**0.55**	**0.77**	0.30	**0.44**	0.39	0.38

Tabelle 7.6: Präzision (pr_i), Ausbeute (re_i) und F1-Wert ($F1_i$) der iTrust Gruppierungen mit und ohne Einbeziehung des Konsens (o. K.). Der höhere beider Werte ist jeweils fett hervorgehoben. *cons* bezeichnet die Konsensbewertung für die jeweilige Gruppierung

Durch betrachten von Tabelle 7.5 ist leicht zu erkennen, dass die Präzision des Modells ohne den Nutzerkonsens um einiges höher ist, die Ausbeute jedoch stark sinkt. Anhand von Tabelle 7.7 kann man erkennen, dass sich die Präzision ohne Konsens lediglich für eine der apache-commons-email-Gruppierung von der Präzision mit Konsens unterscheidet. Für Ausbeute und F1-Wert ist das Verhalten für die apache-commons-email-Gruppierungen ähnlich zu dem für die iTrust-Gruppierungen. Die Erklärung für diese Abweichungen ist recht simpel: Ohne den Nutzerkonsens fließen auch Worte, welche von den Nutzern lediglich ein einziges mal genannt wurden in die Berechnung der Metriken ein. Hierdurch steigt die Anzahl an Wörtern in der Menge der Nutzerantworten, die Anzahl an Wörtern in der erzeugten Schlagwortmenge bleibt jedoch konstant. Die Wahrscheinlichkeit, dass ein Wort aus den Nutzerantworten nicht in der Schlagwortmenge liegt steigt also und folglich sinkt die Präzision. Auf die Ausbeute wirkt sich dieser Anstieg an Wörtern in den

	G_i	pr_i	pr_i o. K.	re_i	re_i o. K.	$F1_i$	$F1_i$ o. K.	*cons*
mittel	email 1	1.00	1.00	**1.00**	0.60	**1.00**	0.75	0.73
	email 2	0.80	0.80	**0.80**	0.66	**0.80**	0.72	0.54
	email 3	0.66	0.66	**1.00**	0.57	**0.80**	0.62	0.69
mittel	email 4	1.00	1.00	**0.50**	0.20	**0.66**	0.33	0.49
	email 5	0.38	0.38	**0.75**	0.43	**0.50**	0.40	0.60
	email 6	0.38	0.38	**0.75**	0.33	**0.50**	0.35	0.52
mittel	email 7	0.50	**1.00**	**0.20**	0.13	**0.29**	0.24	0.29
	email 8	1.00	1.00	**1.00**	0.42	**1.00**	0.59	0.43
	email 9	1.00	1.00	**0.50**	0.14	**0.66**	0.25	0.36

Tabelle 7.7: Präzision (pr_i), Ausbeute (re_i) und F1-Wert ($F1_i$) der apache-commons-email Gruppierungen mit und ohne Einbeziehung des Konsens (o. K.). Der höhere beider Werte ist jeweils fett hervorgehoben. *cons* bezeichnet die Konsensbewertung für die jeweilige Gruppierung

G_i	pr_i	pr_i o. K.	re_i	re_i o. K.	$F1_i$	$F1_i$ o. K.	*cons*
klein	0.79	**0.79**	**0.92**	0.61	0.85	**0.69**	0.58
mittel	0.44	**0.44**	**0.67**	0.31	**0.53**	0.36	0.38
groß	0.89	**1.00**	**0.57**	0.22	**0.70**	0.36	0.38
Gesamt	0.66	**0.68**	**0.71**	0.33	**0.68**	0.44	0.38

Tabelle 7.8: Präzision (pr_i), Ausbeute (re_i) und F1-Wert ($F1_i$) der iTrust Gruppierungen mit und ohne Einbeziehung des Konsens (o. K.). Der höhere beider Werte ist jeweils fett hervorgehoben. *cons* bezeichnet die Konsensbewertung für die jeweilige Gruppierung

Nutzerantworten genau umgekehrt aus, da die Wahrscheinlichkeit ein Wort aus den Nutzerantworten vorherzusagen steigt. Das Einbeziehen eines Nutzerkonsens wirkt sich also insgesamt positiv auf die Aussagekraft der Metriken aus, da sich die Antworten eines einzelnen Nutzers nicht so stark auf sie auswirken. In der weiteren Auswertung werden daher nur noch Metriken betrachtet, welche unter Einbeziehung des Nutzerkonsens berechnet wurden.

Da der F1-Wert das harmonische Mittel aus Präzision und Ausbeute ist, bietet er sich besonders an um die Nutzerantworten mit unseren Beschreibungen zu vergleichen. Der Grund hierfür ist, dass in ihn sowohl Wörter die zwar von den Nutzern aber nicht dem Modell, als auch Wörter die von dem Modell aber nicht von den Nutzern genannt wurden, einfließen. Tabelle 7.11 kann entnommen werden, dass der durchschnittlich erreichte F1-Wert auf den iTrust-Gruppierungen 0.44 und auf den apache-commons-email-Gruppierungen 0.68 beträgt. Das lässt sich so erklären, dass auf den Gruppierungen beider Projekte zwar eine sehr hohe Ausbeute, jedoch nur auf den apache-commons-email-Gruppierungen eine gute Präzision erreicht wird. Im Schnitt haben die Nutzer für die iTrust-Gruppierungen einen Konsens der Länge 2.45 erzeugt, für apache-commons-email-Gruppierungen wurden durchschnittlich 4.22 Wörter im Konsens erzeugt. Wie wir bereits festgestellt haben, wirkt sich ein kleiner Konsens im Allgemeinen negativ auf die Präzision (und folglich den F1-Wert) aus, weshalb dieses Ergebnis sinnvoll erscheint. Betrachtet man darüber hinaus die Konsensbewertungen für die iTrust-Gruppierungen und die apache-commons-email-Gruppierungen, so fällt auf, dass diese für iTrust-Gruppierungen deutlich niedriger ausfallen, als für apache-commons-email-Gruppierungen. Durch die direkte Koppelung des F1-Wertes an den Nutzerkonsens macht es Sinn, dass eine niedrige Konsensbewertung mit einem niedrigen F1-Wert einhergeht. Die Konsensbewertungen und die Länge des Konsens

liefern ein Indiz dafür, dass es den Nutzern leichter gefallen ist apache-commons-email-Gruppierungen zu beschreiben, als iTrust-Gruppierungen. Ein möglicher Grund hierfür ist, dass die Anwendungsdomäne von iTrust um einiges komplexer und spezifischer ist, als die von apache-commons-email. Somit finden sich im Quelltext von iTrust viele medizinische Fachbegriffe und Abkürzungen, welche die Nutzer eventuell nicht verstehen. apache-commons-email hingegen behandelt einen Anwendungsfall, mit dem die meisten Teilnehmer der Studie vertraut sein sollten.

Insgesamt lässt sich für unser Modell damit sagen, dass die Übereinstimmung der erzeugten Schlagwortmengen mit den Nutzerantworten im Bezug auf die Ausbeute gut ist. Sowohl den Teilnehmern, als auch unserem Modell, fiel es schwerer die Gruppierungen von iTrust als die Gruppierungen von apache-commons-email zu beschreiben. Im Rahmen der nächsten Evaluationsfrage soll geklärt werden, ob und wie sich das auf die Güte der erzeugten Beschreibungen auswirkt.

7.3.2.2 (EF2): Güte der Gruppierungsbeschreibungen

G_i	pr_i	re_i	$F1_i$	$rating$	$score_{TP}$	$score_{FP}$	$score_{avg}$	$\|C_i\|$	$\|K_i\|$
iTrust 1	0.60	1.00	0.75	4.25	2.00	1.13	1.65	3	5
iTrust 2	0.67	0.67	0.67	4.25	1.88	1.25	1.67	3	3
iTrust 3	0.43	0.75	0.55	3.50	1.79	1.32	1.52	4	7
klein	0.53	0.80	0.64	4.00	1.89	1.23	1.61	3.34	5.00
iTrust 4	0.11	0.50	0.18	3.50	2.00	1.21	1.29	2	9
iTrust 5	0.40	1.00	0.57	3.75	1.75	1.17	1.40	2	5
iTrust 6	0.25	0.50	0.33	3.13	1.69	1.05	1.21	4	8
mittel	0.22	0.63	0.33	3.45	1.81	1.14	1.30	2.67	7.34
iTrust 7	0.20	1.00	0.33	3.88	1.88	1.44	1.53	1	5
iTrust 8	0.17	1.00	0.29	3.63	1.88	1.28	1.38	1	6
iTrust 9	0.25	1.00	0.40	3.13	1.63	1.15	1.27	2	8
groß	0.21	1.00	0.35	3.54	1.80	1.29	1.39	1.34	6.34
Gesamt	0.30	0.77	0.44	3.67	1.83	1.22	1.44	2.45	6.23

Tabelle 7.9: Überblick über die Auswertung der iTrust Gruppierungen.
pr_i: Präzision des Modells.
re_i: Ausbeute des Modells.
$F1_i$: F1-Wert des Modells.
$rating$: Bewertung der Übereinstimmung von Schlagwortmenge und Gruppierung.
$score_{TP}$: Durchschnittliche Einzelwortbewertung für TPs.
$score_{FP}$: Durchschnittliche Einzelwortbewertung für FPs.
$score_{avg}$: Durchschnittliche Einzelwortbewertung für alle Wörter.
$|C_i|$: Anzahl Wörter im Nutzerkonsens.
$|K_i|$: Anzahl Wörter in der Schlagwortmenge

Nachdem im Rahmen der vorangegangenen Evaluationsfrage geklärt wurde, wie gut die Übereinstimmung der Nutzerantworten mit den Beschreibungen unseres Modells ist, muss nun die absolute Güte der Quelltextbeschreibungen unseres Modells evaluiert werden. Diese Form der Bewertung wird größtenteils durch die Antworten der Nutzer auf die zweite Aufgabe der Studie ermöglicht. In den Tabellen 7.9 und 7.10 finden sich zusätzlich zur Präzision, der Ausbeute und dem F1-Wert die Übereinstimmungsbewertung (im Bezug auf die Semantik der Gruppierung und unserer erzeugten Beschreibung), sowie die durchschnittlichen Einzelwortbewertungen für alle Wörter, die korrekt vorhergesagten Wörter

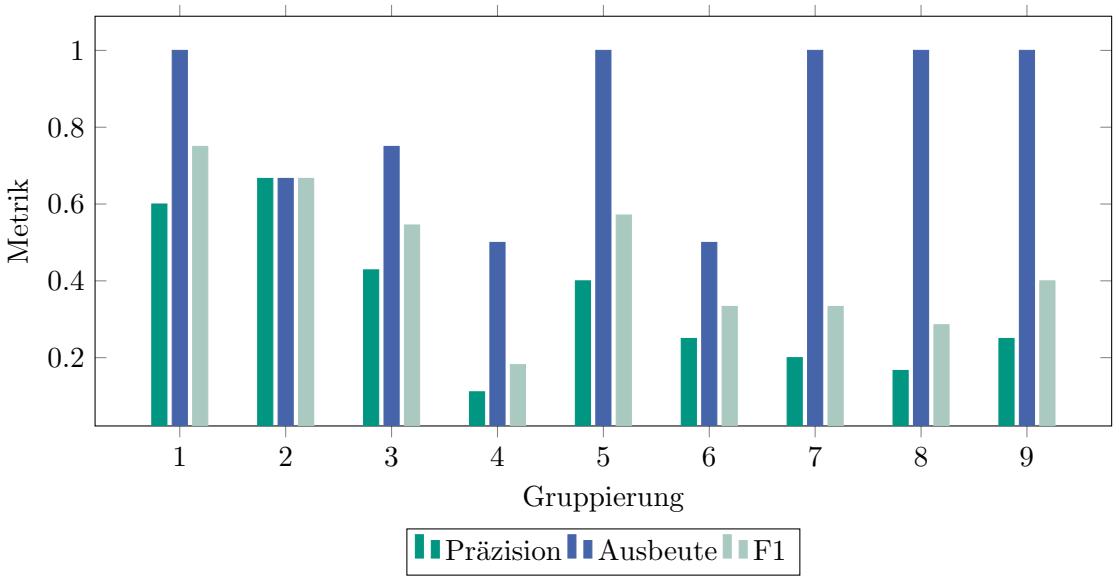


Abbildung 7.3: Überblick über die Präzision, die Ausbeute und den F1-Wert der Gruppierungen von iTrust.

G_i	pr_i	re_i	$F1_i$	$rating$	$score_{TP}$	$score_{FP}$	$score_{avg}$	$\|C_i\|$	$\|K_i\|$
email 1	1.00	1.00	1.00	4.75	2.00	-	2.00	3	3
email 2	0.80	0.80	0.80	4.25	1.72	1.88	1.75	5	5
email 3	0.67	1.00	0.80	4.25	1.88	1.13	1.63	4	6
klein	0.79	0.92	0.85	4.42	1.87	1.50	1.79	4.00	4.67
email 4	1.00	0.50	0.67	4.13	1.94	-	1.94	4	2
email 5	0.33	0.50	0.40	3.13	2.00	0.78	1.94	4	8
email 6	0.38	0.75	0.50	3.63	1.79	1.23	1.57	4	8
mittel	0.44	0.67	0.53	3.63	1.91	1.00	1.54	4.00	6.00
email 7	0.50	0.20	0.29	2.88	2.00	1.13	1.57	5	2
email 8	1.00	1.00	1.00	4.50	2.00	-	2.00	5	5
email 9	1.00	0.50	0.67	4.50	2.00	-	2.00	4	2
groß	0.89	0.57	0.69	3.96	2.00	1.13	1.86	4.67	3.00
Gesamt	0.66	0.71	0.68	4.00	1.93	1.22	1.73	4.22	4.56

Tabelle 7.10: Überblick über die Auswertung der apache-commons-email Gruppierungen.
pr_i: Präzision des Modells.
re_i: Ausbeute des Modells.
$F1_i$: F1-Wert des Modells.
$rating$: Bewertung der Übereinstimmung von Schlagwortmenge und Gruppierung.
$score_{TP}$: Durchschnittliche Einzelwortbewertung für TPs.
$score_{FP}$: Durchschnittliche Einzelwortbewertung für FPs.
$score_{avg}$: Durchschnittliche Einzelwortbewertung für alle Wörter.
$|C_i|$: Anzahl Wörter im Nutzerkonsens.
$|K_i|$: Anzahl Wörter in der Schlagwortmenge

Projekt	pr	re	$F1$	$rating$	$score_{TP}$	$score_{FP}$	$score_{avg}$	$\|C_i\|$	$\|K_i\|$
iTrust	0.65	0.86	0.75	3.67	1.83	1.22	1.44	2.45	6.23
email	0.33	0.65	0.43	4.00	1.93	1.22	1.73	4.22	4.56
Gesamt	0.45	0.73	0.56	3.83	1.88	1.22	3.17	3.34	5.40

Tabelle 7.11: Überblick über die Auswertung des gesamten Evaluationskorpus

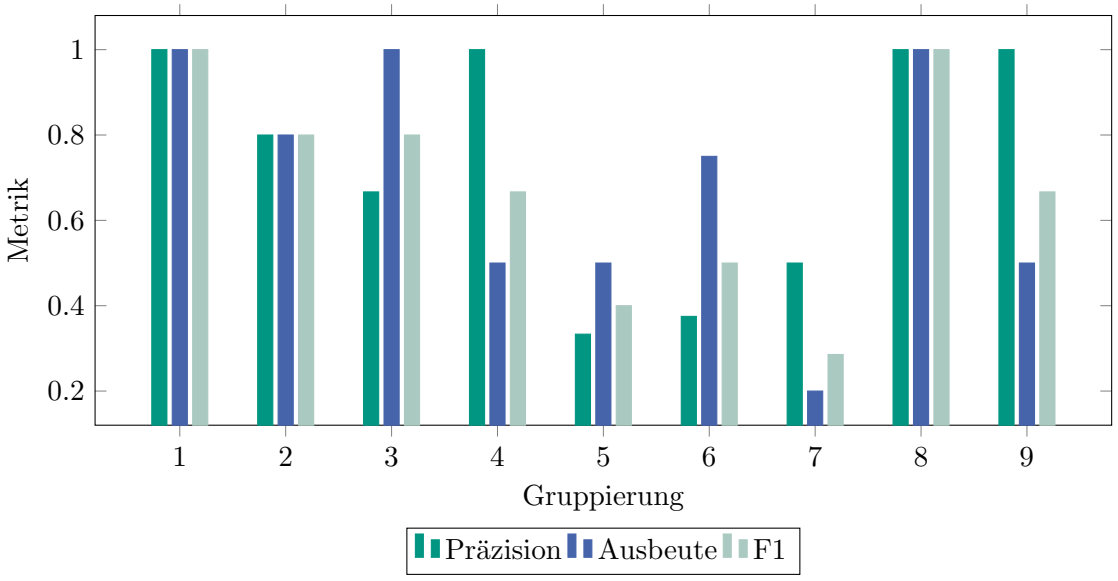


Abbildung 7.4: Überblick über die Präzision, die Ausbeute und den F1-Wert der Gruppierungen von iTrust.

und die fälschlich vorhergesagten Wörter. In den Abbildungen 7.3 und 7.4 sind außerdem die Präzision, die Ausbeute und der F1-Wert der Gruppierungen beider Projekte visualisiert. Insgesamt zeigt sich, dass das implementierte Modell auf dem Evaluationskorpus mit 76% eine gute Ausbeute und mit 54% eine durchschnittliche Präzision aufweist. Dies bedeutet vereinfacht, dass das Modell häufig dieselben Wörter als Teil einer Gruppierungsbeschreibung erzeugt, die auch von den Nutzern als Teil ihrer Beschreibungen verwendet werden. Wollen wir nun die Güte der Beschreibungen messen, so können wir dazu die Übereinstimmungsbewertung und die Einzelwortbewertungen durch die Nutzer verwenden. Betrachtet man die Übereinstimmungsbewertungen für beide Projekte, so fällt auf dass dieser Wert bei den iTrust-Gruppierungen durchschnittlich bei 3.67 und bei den apache-commons-email-Gruppierungen bei 4.00 liegt. Da die Nutzer die Übereinstimmung der Semantik zwischen 1 und 5 bewerten konnten, liegen beide dieser Werte im überdurchschnittlich guten Bereich. Zieht man nun die Einzelwortbewertungen hinzu kann man noch einige weitere Feststellungen machen: Die durchschnittliche Bewertung für ein einzelnes Wort in den iTrust-Gruppierungen liegt bei 1.44, in den apache-commons-email-Gruppierungen liegt sie bei 1.73. Man stellt fest, dass diese Metriken genau wie Präzision, Ausbeute oder F1 für iTrust-Gruppierungen niedriger sind als für apache-commons-email-Gruppierungen. Der Grund hierfür liegt vermutlich wie bereits erwähnt an der höheren Komplexität des iTrust-Projekts. Außerdem kann man Tabelle 7.11 entnehmen, dass für die iTrust-Gruppierungen ungefähr 27% größere Schlagwortmengen erzeugt wurden. Je mehr Schlagwörter erzeugt werden, desto höher ist die Wahrscheinlichkeit ein Wort falsch vorherzusagen, was mit den Bewertungen für iTrust übereinstimmen zu scheint. Betrachtet man deshalb nur die Bewertungen für die Wörter in den T_{TP} und T_{FP} Mengen, so kann man feststellen, wie relevant die mit den Nutzerkonsens übereinstimmenden oder nicht übereinstimmenden Wörter einer Gruppierung sind. Man stellt fest, dass $score_{TP}$ für die iTrust-Gruppierungen bei 1.83 und für die apache-commons-email-Gruppierungen bei 1.93 liegt. Betrachtet man hingegen $score_{FP}$ der komplementären T_{FP} Menge so stellt man fest, dass dieser sowohl für iTrust, als auch für apache-commons-email bei 1.22 liegt. Diese große Differenz zwischen $score_{TP}$ und $score_{FP}$ für beide Projekte ist ein Indiz dafür, dass der Nutzerkonsens tatsächlich hauptsächlich aus für die Gruppierung relevanten Wörtern besteht. Da die Erstellung des Nutzerkonsens und der Einzelwortbewertungen darüber hinaus auf Basis der Antworten unterschiedlicher Nutzer geschah, ist ausgeschlos-

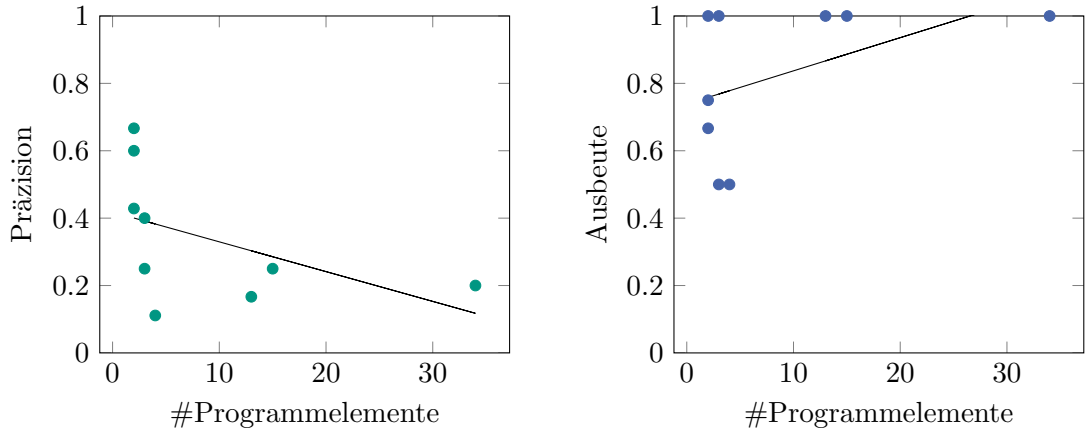


Abbildung 7.5: Streudiagramme der Präzision und Ausbeute für iTrust Gruppierungen gegen die Anzahl der Programmelemente in den Gruppierungen

sen, dass Nutzer ihre selbst genannten Wörter höher bewertet haben. Da ein Wert von 1 einer durchschnittlich neutralen Bewertung entspricht, stellt man fest, dass selbst die fälschlich vorhergesagten Wörter noch eine besser als neutrale Bewertung erhalten haben. Dieses Verhalten ist auffällig, da es entweder darauf hindeutet, dass der Nutzerkonsens im Allgemeinen zu klein ist und unser Modell mehr wichtige Wörter erkennt als die Nutzer oder, dass die Nutzer selbst Wörter, welche eigentlich keine wirkliche Relevanz für die Gruppierung besitzen, noch überdurchschnittlich gut bewerten. Aus diesem Grund sollte der Wert $score_{FP}$ vorsichtig behandelt werden, da er eventuell nicht repräsentativ für die Gruppierungen insgesamt ist.

Insgesamt bedeutet dies für unser Modell, dass eine Kombination aus hohem F1-Wert, sowie hohem $score_{TP}$ ein Indiz für hochwertige Gruppierungsbeschreibungen ist. Für iTrust ist der F1-Wert mit 0.44 zwar recht niedrig, die durchschnittliche Einzelwortbewertung $score_{TP}$ ist mit 1.83 jedoch sehr gut und auch die Übereinstimmung der Semantik der Gruppierung und unserer Beschreibung ist mit 3.67 überdurchschnittlich gut. Auf den apache-commons-email-Gruppierungen ist der F1-Wert mit 0.68 deutlich höher, $score_{TP}$ ist auch hier mit 1.93 sehr gut und die Bewertung der semantischen Übereinstimmung ergab einen Durchschnittswert von 4.00. Aus diesen Daten geht hervor, dass die Güte der von unserem Modell erzeugten Gruppierungsbeschreibungen auf dem Evaluationskorpus hoch ist. Der Evaluationskorpus besteht aus zwei Projekten gänzlich verschiedener Domänen, die Leistung unseres Modells ist jedoch im Hinblick auf F1-Wert, $score_{TP}$ und semantischer Übereinstimmung für beide Projekte vergleichbar. Das deutet darauf hin, dass die Qualität der Kommentare, der Bezeichner und des Quelltextes allgemein für die Übertragbarkeit der Ergebnisse eine viel größere Rolle spielt, als die Domäne des Softwareprojekts.

7.3.2.3 (EF3): Auswirkungen der Größe einer Gruppierung

Die Streudiagramme in den Abbildungen 7.5 und 7.6 stellen jeweils die erreichte Präzision und Ausbeute für die iTrust-Gruppierungen und apache-commons-email-Gruppierungen im Vergleich zu der Anzahl an Programmelementen in den Gruppierungen dar. Zusätzlich wurden Trendlinien hinzugefügt um Änderungen am Verhalten des Modells, bei Variation der Anzahl an Programmelementen, leichter erkennbar zu machen. Zunächst fällt auf, dass das Auswahlverfahren für Quelltextgruppierungen, wie es in Abschnitt 7.1 beschrieben wurde, tatsächlich die Verteilung von Programmelemente pro Gruppierung der Gesamtgruppierungsmenge aufrecht erhält und somit nur wenige Gruppierungen mit sehr vielen Programmelementen teil des Evaluationskorpus sind. Der Evaluationskorpus sollte

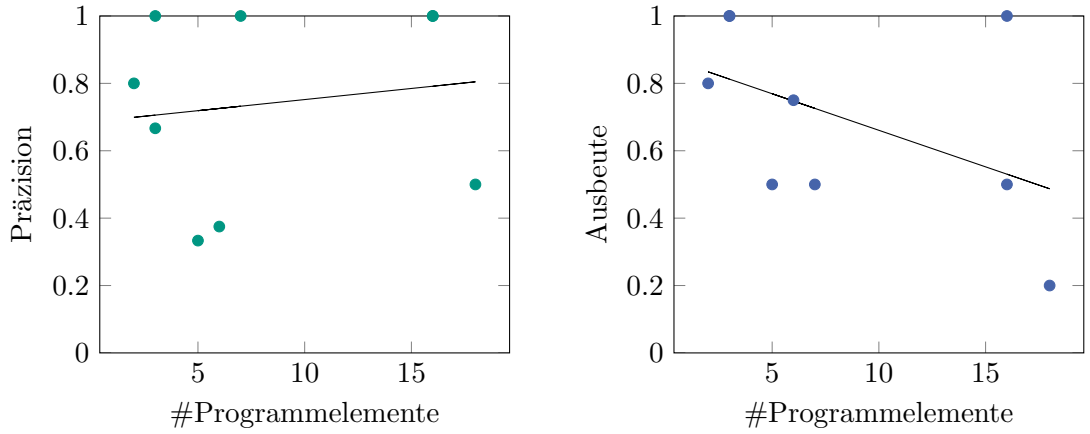


Abbildung 7.6: Streudiagramme der Präzision und Ausbeute für email Gruppierungen gegen die Anzahl der Programmelemente in den Gruppierungen

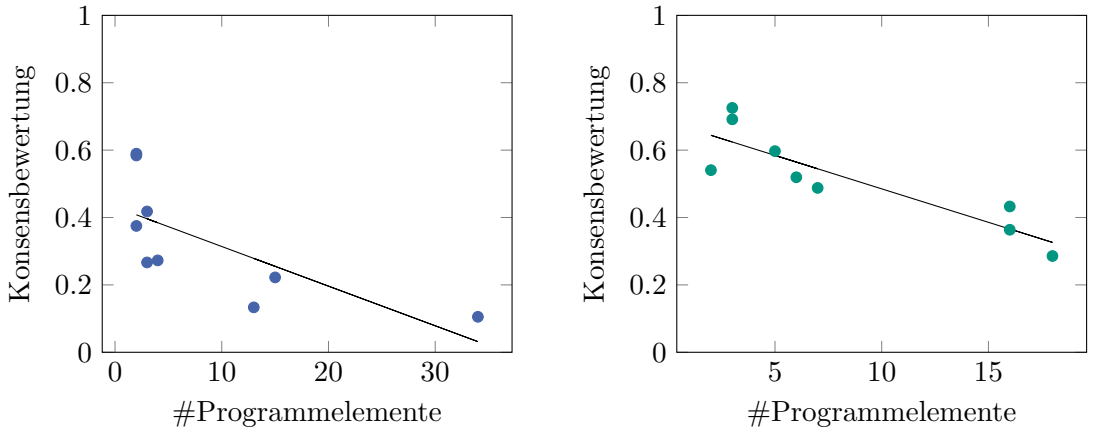


Abbildung 7.7: Streudiagramme der Konsensbewertungen für iTrust-Gruppierungen (links) und apache-commons-email-Gruppierungen (rechts) verglichen mit der Anzahl an Programmelementen.

in dieser Hinsicht also repräsentativ für die Gesamtmenge der Quelltextgruppierungen beider Projekte sein. Man stellt jedoch fest, dass die Trendlinien, sowohl für die Präzision als auch für die Ausbeute, nicht konsistent sind. Auf den iTrust-Gruppierungen zeichnet sich ein negativer Trend für die Präzision ab, auf den apache-commons-email-Gruppierungen ist der Trend jedoch positiv. Für die Ausbeute ist das Verhalten genau umgekehrt: Der Trend auf den iTrust-Gruppierungen ist positiv, während er auf den apache-commons-email-Gruppierungen negativ ist. Dieser Vergleich der Metriken anhand der Anzahl an Programmelementen in den Gruppierungen lässt also keine zuverlässige Aussage über das Verhalten des Modells bei steigender Anzahl an Programmelementen zu.

Als nächstes wird die Auswirkung der Anzahl an Programmelementen einer Gruppierung auf ihre Konsensbewertung (siehe Abschnitt 7.3.1.2) betrachtet. Dadurch soll bestätigt oder widerlegt werden, ob es den Nutzern schwerer fällt Gruppierungen mit vielen Programmelementen zu beschreiben. Abbildung 7.7 stellt die Konsensbewertungen für die iTrust- bzw. apache-commons-email-Gruppierungen im Vergleich zu ihrer Anzahl an Programmelementen dar. Wie auch zuvor wurde eine Trendlinie hinzugefügt um Muster in den Daten leichter erkennen zu können. Anders als bei den zuvor getesteten Metriken zeichnet sich hier ein eindeutiger Trend ab. Sowohl für die iTrust-Gruppierungen, als auch für die apache-commons-email-Gruppierungen nehmen die Konsensbewertungen mit der

Anzahl an Programmelementen ab. Den Nutzern fällt es also offensichtlich leichter kleine Quelltextgruppierungen zu beschreiben, als Große.

Aus diesen beiden Ergebnissen kann man letztendlich ableiten, dass die Leistung unseres Modells nicht direkt von der Anzahl an Programmelementen in den Quelltextgruppierungen abhängt. Allerdings kann man feststellen, das die Bewertungsleistung und Konsensbildung der Nutzer bei steigender Anzahl an Programmelementen abnimmt, was auch intuitiv richtig erscheint. Da die Metriken zur Bewertung der Gruppierungen direkt vom Nutzerkonsens abhängig sind, nimmt die Aussagekraft der Metriken für Gruppierungen mit vielen Programmelementen ab.

7.3.3 Betrachtung ausgewählter Quelltextgruppierungen

In diesem Abschnitt werden einige ausgewählte Quelltextgruppierung genauer betrachtet. Diese Gruppierungen lassen entweder interessante Einblicke auf das Verhalten der Teilnehmer der Nutzerstudie oder das Verhalten unseres implementierten Modells zu.

7.3.3.1 Gruppierung iTrust 1

Nutzerkonsens	Erz. Beschr.
HTML (8/8)	html
encoding/encode(r) (8/8)	encoder
url (7/8)	url
	checks
	input

Tabelle 7.12: Nutzerkonsens und erzeugte Beschreibung für Gruppierung iTrust 1

Die erste Gruppierung des iTrust Projekts kann genutzt werden um den Zweck und den Prozess der Konsensbildung noch ein mal zu verdeutlichen. Tabelle 7.12 zeigt sowohl den Nutzerkonsens, als auch die von unserem Modell erzeugte Schlagwortmenge. Unter Abschnitt B.1 finden sich alle Nutzerantworten und Bewertungen für diese Gruppierung. Jeder Nutzer hat entweder „encoder“, „encoding“ oder „encode“ als Schlagwort angegeben. Insgesamt wurde drei mal „encoder“, zwei mal „encoding“ und drei mal „encode“ verwendet. Im Kontext dieser Gruppierung ist zu erkennen, dass die Nutzer vermutlich dieselbe Intention hatten (ein Wort zur Beschreibung der Semantik von „kodieren“ zu finden), jedoch auf Grund verschiedener Faktoren, wie Programmiererfahrung, Englischkentniss oder Verständnis der Aufgabe unterschiedliche Wortformen gewählt haben um diese auszudrücken. Fasst man diese drei Schlagworte zu einem kombinierten Schlagwort „encoder/encoding/encode“ zusammen, so wird damit eine Abdeckung über 8/8 der Nutzerantworten erreicht und das Wort fließt in den Konsens ein. Würde man die Wörter nicht zusammenfassen, so würde der Schwellwert von 50% der Nutzerantworten nicht erreicht werden und es würde kein Wort, welches die Semantik „encode“ repräsentiert als Konsens angesehen werden. In der durch unser Modell erzeugten Beschreibung findet sich das Wort „encoder“, durch die Normalisierung stimmt das Wort mit dem Konsens überein uns es wird als richtig positiv (TP) gewertet. Tabelle 7.9 kann entnommen werden, dass für diese Gruppierung eine Präzision von 0.60 und eine Ausbeute von 1.00 erreicht wurde. Das Modell hat also dieselben Schlagwörter erzeugt, wie sich auch im Konsens der Nutzer vorliegen. Die durchschnittliche Bewertung der übereinstimmenden Wörter $score_{TP}$ liegt bei 2.00, was darauf hindeutet, dass diese Wörter auch tatsächlich für die Gruppierung relevant sind. Die Bewertung der nicht übereinstimmenden Wörter liegt bei 1.13 und ist somit eher neutral, dennoch haben die Nutzer sie als indirekt zusammenhängend mit der

Semantik der Gruppierung bewertet. In der Bewertungsaufgabe hat diese Gruppierung eine Bewertung von 4.25 erhalten und ist somit gut. Dieses Beispiel zeigt noch einmal, dass die Normalisierung und Konsensbildung für die Evaluation von großer Bedeutung ist, da sie die Aussagekraft unserer genutzten Metriken verbessert. Aus Kapitel 6 wissen wir, dass die Stammformreduktion zur Vorverarbeitung von Quelltexttoken das Rauschen im Trainingsdatensatz für unser Modell verringert. Analog dazu verringert die Konsensbildung das rauschen in den Nutzerantworten.

7.3.3.2 Gruppierung iTrust 7

Nutzerkonsens	Erz. Beschr.
patient (8/8)	patient
	bean
	set
	phone
	message

Tabelle 7.13: Nutzerkonsens und erzeugte Beschreibung für Gruppierung iTrust 7

Die Gruppierung iTrust 7 ist ein gutes Beispiel, welches zeigt, dass das Bilden eines Konsens und das Zuordnen von Wörtern zu Konzepten selbst für Menschen nicht immer einfach ist. Tabelle 7.13 zeigt sowohl den Nutzerkonsens, als auch die von unserem Modell erzeugte Schlagwortmenge. Unter Abschnitt B.7 finden sich alle Nutzerantworten und Bewertungen für diese Gruppierung. Bei der Gruppierung handelt es sich um Zugriffsmethoden aus der Klasse `PatientBean`. Die Klasse speichert eine Vielzahl an Patienteninformationen, wie etwa den Namen, die Adresse oder die Telefonnummer. Wie Tabelle 7.2 zu entnehmen ist, handelt es sich bei dieser Gruppierung zusätzlich um die Gruppierung mit den meisten Programmelementen (34). Außerdem enthält die Gruppierung überdurchschnittlich viele einzigartige Token (188). Jeder Teilnehmer hat das Wort „patient“ als Schlagwort angegeben. Dies ist offensichtlich sinnvoll, da es sich bei der Gruppierung um Methoden zum Modifizieren von Patienteninformationen handelt. Allerdings haben lediglich drei Nutzer eine Form von „patient information“ angegeben. Zwei weitere Nutzer haben „personal data“ angegeben. Die restlichen Nutzer haben die Felder, wie Name, Adresse oder Telefonnummer nicht zu einem der übergeordneten Konzepte „information“ oder „data“ zusammengefasst, sondern einzeln angegeben. Da die Normalisierung, wie bereits eingeführt, möglichst konservativ durchgeführt werden soll, werden weder die einzelnen Wörter, welche Teil der Patienteninformationen sind, noch „personal data“ und „patient information“ zusammengefasst. Somit schafft es lediglich das Wort „patient“ in den Konsens. Ein Grund für die Unsicherheit der Nutzer kann die hohe Anzahl an Programmelementen und die damit einhergehende kognitive Belastung beim Lesen des Quelltextes sein. Darüber hinaus ist es auch möglich, dass die Fragestellung der Beschreibungsaufgabe zu undeutlich formuliert war und es den Nutzern nicht bewusst war bis zu welchem Grad sie Konzepte zusammenfassen dürfen. Insgesamt erreicht diese Gruppierung eine Präzision von nur 0.2, da in der Beschreibung Wörter wie „phone“ oder „message“ enthalten sind. Allgemein ist es dem Modell nicht möglich solche einzelnen Token zu einem übergreifenden Konzept zusammenzufassen, da LDA auf Kookkurenz von Wörtern basiert und kein Wissen über das Zusammenspiel dieser Wörter hat. Das Finden und Generalisieren von zusammengehörigen Wörtern in einer Gruppierungsbeschreibung kann als Ausgangspunkt für weiterführende Arbeiten genutzt werden, worauf in Kapitel 8 genauer eingegangen wird. Die Präzision für diese Gruppierung liegt nur bei 0.20, da lediglich eines der erzeugten Wörter mit dem Nutzerkonsens übereinstimmt. Gleichzeitig liegt die Sensitivität bei 1.00, da das Modell dennoch das einzige Wort aus dem Konsens als Teil seiner Beschreibung erzeugt hat. Die

durchschnittliche Bewertung für die erzeugte Beschreibung der Gruppierung als ganzes liegt bei 3.88. Insgesamt zeigt diese Gruppierung ein Problem und einen Vorteil unseres Verfahrens auf. Aus Abschnitt 7.3.2.3 wissen wir, dass die Anzahl an Programmelementen einer Gruppierung zwar einen Einfluss auf die Qualität des Nutzerkonsens hat, ein solcher Zusammenhang zur Güt der erzeugten Beschreibungen jedoch nicht hergestellt werden kann. Obwohl diese Gruppierung mit 0.10 die niedrigste Konsensbewertung aller iTrust-Gruppierungen erhalten hat, wurde die Übereinstimmung der Semantik gut bewertet. Unser Modell erzeugt also selbst für durch Menschen schwer beschreibbare Quelltextgruppierungen noch gute Beschreibungen. Allerdings zeigt dieses Beispiel auch auf, dass unser Modell keine Möglichkeit hat Konzepte zu generalisieren und somit die Größe und Aussagekraft der Schlagwortmenge zu verbessern.

7.3.3.3 Gruppierung email 2

Nutzerkonsens	Erz. Beschr.
test (8/8)	resolver
classpath (6/8)	data
data (5/8)	source
resolution/resolve(r) (5/8)	path
source (4/8)	test

Tabelle 7.14: Nutzerkonsens und erzeugte Beschreibung für Gruppierung email 2

Das Beispiel 7.2 basiert auf Erkenntnissen welche aus der zweiten Gruppierung des apache-commons-email Projekts hervorgingen. Die Gruppierung zeigt auf, weshalb qualitativ hochwertiger Quelltext eine Voraussetzung für gute Gruppierungsbeschreibungen ist. Tabelle 7.14 zeigt sowohl den Nutzerkonsens, als auch die von unserem Modell erzeugte Schlagwortmenge. Unter Abschnitt B.11 finden sich alle Nutzerantworten und Bewertungen für diese Gruppierung. Im Quelltext der Gruppierung findet sich das Token `ClassPath`. Die korrekte Schreibweise für dieses Wort ist jedoch `Classpath`. Durch diese falsche Schreibweise wird das Token während der Vorverarbeitung in die Subtoken „class“ und „path“ zerlegt. Wegen der Filterung von Java-Schlüsselwörtern wird anschließend das Wort „class“ aus dem Vokabular entfernt. Somit ist es dem Modell weder möglich „class“ und „path“ einzeln, noch „classpath“ zusammen als Teil der Beschreibung zu erzeugen. Da die Entfernung von Java-Schlüsselwörtern einen wichtigen Vorverarbeitungsschritt zur Reduktion von Rauschen in den Trainingsdaten darstellt, kann dieser Fehler der Programmierer von apache-commons-email nicht trivial kompensiert werden. Anhand der Gruppierung lässt sich noch ein weiteres Beispiel aufzeigen: Im Quelltext findet sich das Token `DataSource`. Anders als bei „classpath“ ist es hier nicht eindeutig, ob „Datasource“ oder „Data source“ korrekt ist. In der Tat sind auch die Antworten der Nutzer nicht eindeutig, es finden sich sowohl die zusammengeschriebene, als auch die getrennte Form. Durch die Binnenmajuskelschreibweise im Quelltext wurde auch `DataSource` zu „data“ und „source“ zerlegt, weshalb die Nutzerantworten auf die beiden getrennten Token normalisiert wurden. Insgesamt gaben fünf von acht Teilnehmern das Wort „data“ und vier von acht Teilnehmern das Wort „source“ an, weshalb beide Wörter teil des Konsens sind. Für diese Gruppierung wurden eine Präzision, sowie eine Ausbeute von 0.8 erreicht. Da beide dieser Werte nahe bei 1 liegen, hat das Modell sowohl einen Großteil der Wörter des Konsens erzeugt und gleichzeitig nur wenige konsensfremde Wörter miteinbezogen. Die Nutzerbewertungen für diese Gruppierung liegt durchschnittlich bei 4.25. Für unser Modell bedeutet das, dass Fehler im Ausgangsquelltext, auf die wir keinen Einfluss haben, die Leistung unseres Modells beeinflussen können. Außerdem kann es vorkommen, dass durch eine Kombination aus Subworttokenisierung und Stoppwortentfernung potentiell wichtige Wörter nicht in den

Schlagwortmengen unseres Modells auftauchen können. Auf das erste Problem können wir nur wenig Einfluss nehmen, da wie bereits angemerkt die Leistung unseres Modells von der Qualität des Ausgangsquelltextes abhängt. Eine Idee zur Behebung des zweiten Problems ist es Token in Binnenmajuskelschreibweise sowohl aufgespalten, als auch in ihrer Gesamtheit in die Beschreibungen einfließen zu lassen. Dadurch erhöht sich zwar die Anzahl an Wörtern im Vokabular und somit das rauschen leicht, allerdings können so auch zusammengehörige Wörter, wie `DataSource`, teil der erzeugten Beschreibungen werden. Ein ähnlicher Ansatz, ohne die Veränderung des Ausgangsvokabulars, ist es die Wörter in der Schlagwortmenge und die zusammengesetzten Bezeichner im Quelltext zu betrachten und diese Bezeichner zu rekonstruieren, sollten ihre Subtoken in den Gruppierungen enthalten sein.

7.3.3.4 Gruppierung email 7

Nutzerkonsens	Erz. Beschr.
email (8/8)	email
reply (5/8)	set
recipient/receiver (4/8)	
cc (4/8)	
sender (4/8)	

Tabelle 7.15: Nutzerkonsens und erzeugte Beschreibung für Gruppierung email 7

Die siebte Quelltextgruppierung des apache-commons-email Projekts sticht hervor, da sie im gesamten Evaluationsdatensatz die niedrigste Ausbeute erreicht. Tabelle 7.15 zeigt sowohl den Nutzerkonsens, als auch die von unserem Modell erzeugte Schlagwortmenge. Unter Abschnitt B.16 finden sich alle Nutzerantworten und Bewertungen für diese Gruppierung. Der Nutzerkonsens besteht für diese Gruppierung aus den Wörtern „email“, „reply“, „recipient/receiver“, „cc“ und „sender“. Unser Modell liefert als Schlagwortmenge lediglich die Wörter „email“ und „set“. Hier raus ergibt sich eine Ausbeute von 0.2, eine Präzision von 0.5 und ein F1-Wert von 0.29. Das Wort „cc“ kann von unserem Modell nicht vorhergesagt werden, da es eine Länge von 2 besitzt und somit bereits aus den Trainingsdaten herausgefiltert wurde. Betrachtet man die Metadaten der Gruppierung, so fällt auf, dass sie aus 18 Programmelementen, 303 Token insgesamt, allerdings nur 39 einzigartigen Token besteht. Wie bereits in Abschnitt 7.3.2.3 angemerkt, wirkt sich die Größe der Gruppierungen negativ auf die Bewertungsleistung der Nutzer und somit auf die Aussagekraft der berechneten Metriken aus.

7.4 Gefährdung der Validität

In diesem Abschnitt werden einige Entscheidungen, die während der Evaluation getroffen wurden, kritisch hinterfragt und es wird geklärt inwiefern sie die Validität der Ergebnisse der Evaluation gefährden können. Außerdem wird darauf eingegangen, wie diese Fehler hatten vermieden werden können.

7.4.1 Keine Methodenrümpfe in der Gruppierungsrepräsentation

Die gewählte Repräsentationsform für Quelltextgruppierungen aus Abschnitt 7.2.2 enthält keine Methodenrümpfe. Diese Entscheidung wurde getroffen um den Nutzern das Bearbeiten der Evaluationsaufgaben zu erleichtern, da die Methodenrümpfe im Vergleich zu den Signaturen meist sehr viel länger sind. Gerade bei Quelltextgruppierungen mit sehr vielen Programmelementen ist diese Abwägung sinnvoll, da ansonsten Nutzerantworten

unter der kognitiven Last der Gruppierungsrepräsentation leiden können. Allerdings gilt auch zu beachten, dass dies eigentlich im Widerspruch zu unserem Ziel steht, dem Nutzer alle Informationen zugänglich zu machen, die auch das Modell zur Verfügung hatte um Beschreibungen zu erzeugen. Da ein großer Teil der Programmelemente in den Quelltextgruppierungen des Evaluationskorpus aus Zugriffsmethoden besteht, welche in den meisten Fällen in ihrem Rumpf ohnehin keine Variablen definieren, erschien diese Abwägung sinnvoll. Dennoch kann es sein, dass den Nutzern wichtige Informationen verwehrt blieben und deshalb die Evaluationsergebnisse verfälscht wurden.

7.4.2 Testquelltext im Datensatz

Sowohl bei der Gruppierung des iTrust-Projekts, als auch bei der Gruppierung des apache-commons-email-Projekts, wurden der normale Anwendungs- bzw. Bibliotheksquelltext aber auch der Testquelltext der Projekte miteinbezogen. Aus diesem Grund beinhalten viele Gruppierungen des Evaluationskorpus sowohl Programmelemente aus dem normalen Quelltext, als auch aus dem Testquelltext. Es können zwar semantische Beziehungen zwischen Tests und dem normalen Quelltext bestehen, allerdings verfolgt Testquelltext eine andere Absicht als normaler Quelltext. Tests beschäftigen sich mit der Überprüfung von korrektem spezifizierten Verhalten, die Semantik von Testklassen und Methoden hat also im allgemeinen wenig mit der Semantik der entsprechenden Anwendungs- bzw. Bibliotheksmethoden zu tun. INDIRECT verfolgt die Aufgabe Rückverfolgbarkeitsinformationen zwischen Anforderungen und Quelltext zu fördern. Im Rahmen dieser Aufgabe spielt der Testquelltext jedoch keine Rolle, da er nicht in den Anforderungen spezifiziert ist. Um aussagekräftigere Ergebnisse zu erhalten, hätte man den Testquelltext verwerfen und nur den normalen Quelltext erhalten sollen. Allerdings ermöglichen die Evaluationsergebnisse dennoch einen Einblick in die Beschreibungsleistung unseres Modells, da diese in erster Linie von den Anforderungen unabhängig ist.

7.4.3 Studiendesign

An unserer Evaluationsstudie haben insgesamt 16 Nutzer teilgenommen, welche in zwei Gruppen von je 8 Teilnehmern aufgespalten wurden. Es wurden je 9 Quelltextgruppierungen aus zwei verschiedenen Java-Projekten in zwei Aufgaben betrachtet. Sowohl die Anzahl an Teilnehmern pro Softwareprojekt, als auch die Anzahl an betrachteten Quelltextgruppierungen stellt eine Gefahr für die Validität unserer Ergebnisse dar. Es ist nicht klar, ob die Anzahl an Gruppierungen ausreicht um die Leistung des Modells allgemein zu bewerten. Obwohl sich bereits bei dieser kleinen Anzahl an Gruppierungen einige Muster erkennen lassen, sollte das Modell auf einem größeren Korpus evaluiert werden um sicherzustellen, dass die Ergebnisse übertragbar sind. Im Rahmen dieser Arbeit war dies nicht möglich, da wir von der Nutzerstudie als Approximation für unseren Goldstandard abhängig sind. Die Bearbeitungszeit für das Formular einer Evaluationsgruppe betrug bereits in der jetzigen Form etwa 20 Minuten. Da es sich bei den Teilnehmern um freiwillige handelte, wirkt sich ein hoher Arbeitsaufwand negativ auf die Motivation der Nutzer aus. Somit wurde mehr Wert auf eine hohe Teilnehmerzahl, als auf eine hohe Abdeckung an Gruppierungen gelegt. Sollte eine Möglichkeit gefunden werden den Goldstandard für Gruppierungsbeschreibungen ohne eine Nutzerstudie zu approximieren, so sollte die Leistung des Modells auf einer größeren Datengrundlage erneut evaluiert werden.

8 Zusammenfassung und Ausblick

Das Ziel dieser Arbeit war es eine Beschreibungsrepräsentation für die geteilte Absicht der INDIRECT-Quelltextgruppierungen zu entwerfen und zu implementieren. Dazu wurden zunächst verschiedene Repräsentationen, sowohl für Quelltext als auch für Absichtsbeschreibungen, analysiert. Es wurde festgestellt, dass Quelltextelemente die eine gemeinsame Semantik besitzen auch häufig ähnliche natürliche Sprache in ihren Bezeichnern und Kommentaren verwenden. Für den Quelltext der Gruppierungen von INDIRECT wurde deshalb eine Repräsentation durch die im Quelltext enthaltenen Token ausgewählt. Außerdem wurde die Repräsentation von Absichtsbeschreibungen durch Fließtextsätze und durch Schlagwortmengen miteinander verglichen. Die Repräsentation durch Schlagwortmengen hat sich durchgesetzt, da sie kein grammatikalisches Verständnis voraussetzen und dennoch einen guten Informationsgehalt besitzen. Darüber hinaus können Schlagwortmengen durch weitere Verfahren, wie der Erzeugung von Fließtextsätzen, weiterverarbeitet werden, um ihren Informationsgehalt zu erhöhen. Basierend auf dieser Erkenntnis und der Repräsentation durch Token wurde eine Vorstudie durchgeführt in der verschiedene Verfahren zur Themenmodellierung auf Textdokumenten miteinander verglichen wurden. Von den in der Vorstudie getesteten Verfahren hat LDA sich als ein guter Ausgangspunkt zur Erzeugung von Themen in Quelltextgruppierungen herausgestellt. Außerdem hat die Vorstudie einige wichtige Erkenntnisse über den Trainingsprozess und die Konfiguration von LDA ermöglicht. Anschließend wurde diskutiert wie die von LDA erzeugten Themen genutzt und aufbereitet werden können um aus ihnen aussagekräftige Schlagwortmengen zu gewinnen. Da eine Festlegung des k-Parameters (Anzahl an erzeugten Themen) von LDA nicht trivial und auch nicht allgemeingültig möglich ist werden mehrere LDA-Modelle mit leicht abweichenden k-Parametern trainiert. Zur Erzeugung einer Schlagwortmengen wird zunächst durch jedes dieser LDA-Modelle eine Menge von Themen für eine Quelltextgruppierung erzeugt. Diese Mengen werden dann zu einer einzelnen Menge kombiniert, welche alle erzeugten Themen enthält. Diese Themen werden mithilfe eines agglomerativen Clusterings mit der Hellinger-Distanz als Abstandsmaß zu einer Menge von Themengruppierungen kombiniert. In einem zweistufigen Rückkombinationsverfahren werden diese Themengruppierungen zunächst zu einer einzelnen Menge von Themen und anschließend zu einem einzelnen Thema zusammengefasst. Durch Bilden des Durchschnitts zwischen den Wörtern in diesem Thema und den Token aus der entsprechenden Quelltextgruppierung werden die Wörter identifiziert, welche wichtig für die Semantik der Gruppierung sind. Dieser letzte Schritt basiert auf der Annahme, dass eine Quelltextgruppierung bereits einen großen Teil der Token beinhaltet, die benötigt werden um ihre Semantik zu beschreiben.

Im Rahmen der Evaluation wurde die Leistung des implementierten Verfahren mithilfe der Quelltextgruppierungen zweier Softwareprojekte bewertet. Die Evaluation lief im Rahmen einer Nutzerstudie ab. In der ersten Aufgabe mussten die Teilnehmer Quelltextgruppierungen mit Schlagwörtern beschreiben. Basierend auf den Nutzerantworten wurde ein Konsens gebildet um einen Goldstandard für Gruppierungsbeschreibungen zu approximieren. Im Rahmen der zweiten Aufgabe haben die Nutzer die von dem in dieser Arbeit implementierten Verfahren bewertet. Insgesamt erreicht das Verfahren Durchschnittswerte von 0.55 für die Präzision, 0.77 für die Ausbeute und 0.57 für den F1-Wert. Für die zweite Aufgabe haben die Nutzer die Zugehörigkeit der einzelnen Wörter der Schlagwortmenge zur Semantik der Quelltextgruppierung und die Schlagwortmenge selbst bewertet. Insgesamt bewerteten die Teilnehmer die erzeugten Schlagworte mit 1.58, was zwischen neutral und gut liegt, allerdings leicht zu gut tendiert. Wörter die sowohl in der erzeugten Schlagwortmenge, als auch im Nutzerkonsens vorkamen wurden durchschnittlich mit 1.88 bewertet. Da selbst die Wörter aus der Schlagwortmenge, welche nicht teil des Nutzerkonsens waren noch mit 1.22 (leicht besser als neutral) bewertet wurden, zeigt dies, dass der niedrige Präzisionswert unseres Verfahrens nicht unbedingt eine schlechte Beschreibungsleistung bedeutet. Eine Bewertung der Übereinstimmung der Semantik der Schlagwortmengen und der Semantik der Quelltextgruppierungen ergab einen Durchschnittswert von 3.88. Diese Ergebnisse zeigen, dass das implementierte Verfahren häufig dieselben Wörter erzeugt die auch von Teilnehmern der Studie genannt wurden. Außerdem besitzen diese Wörter eine hohe Relevanz für die Semantik der Quelltextgruppierung.

Trotz dieser guten Ergebnisse bestehen Möglichkeiten das Verfahren noch weiter zu verbessern: Ein möglicher Verbesserungspunkt stellt die Kombination der Themen mehrerer LDA-Modelle dar. Hierzu wurde in dieser Arbeit ein agglomeratives Clustering mit der Hellinger-Distanz als Abstandsmaß genutzt. Die Arbeit von Mantyla *et al.* [MCF18] zeigt jedoch, dass auch andere Clusteringverfahren und Abstandsmaße genutzt werden können. Unter diesem Aspekt könnte noch untersucht werden, ob ein anderes Clusteringverfahren oder Abstandsmaß die Ergebnisse unseres Verfahrens noch weiter verbessern kann.

Eine weitere Verbesserungsmöglichkeit besteht darin die erzeugten Schlagwortmengen weiterzuverarbeiten. So können etwa Themenbeschriftungsverfahren (siehe Abschnitt 2.3) eingesetzt werden um die Schlagwortmengen noch zu präzisieren und ihr abstraktere Konzepte zuzuordnen. Allerdings wird hierfür weiterhin eine geeignete externe Wissensquelle benötigt, deren Domäne eine Abstraktion der Schlagwortmenge zulässt. Ebenso könnten die Schlagwortmengen als Ausgangspunkt zur Erzeugung von natürlichsprachlichen Sätzen, mithilfe von neuronalen Sprachmodellen, genutzt werden.

Da zum aktuellen Zeitpunkt kein Datensatz bestehend aus Quelltextgruppierungen und deren natürlichsprachlichen Beschreibungen vorliegt, konnten Verfahren zur Quelltextzusammenfassung welche auf überwachten maschinellen Lernverfahren aufbauen nicht genutzt werden. Mithilfe des hier implementierten unüberwachten Verfahren zur Beschreibung von Quelltextgruppierungen könnte jedoch ein solcher Datensatz aufgebaut und dessen Goldstandard approximiert werden. Ein solcher Ansatz könnte besonders in Kombination mit der Erzeugung von natürlichsprachlichen Sätzen aus den Schlagwortmengen interessant sein, da dies das Training eines maschinellen Übersetzungsmodells erlauben würde.

Literaturverzeichnis

[ABBS15] Allamanis, Miltiadis ; Barr, Earl T. ; Bird, Christian ; Sutton, Charles: Suggesting Accurate Method and Class Names. In: *ESEC/FSE 2015*, 2015 (zitiert auf den Seiten 21 und 34).

[ABLY19] Alon, Uri ; Brody, Shaked ; Levy, Omer ; Yahav, Eran: Code2seq: Generating Sequences from Structured Representations of Code. In: *arXiv:1808.01400 [cs, stat]* (2019), Februar (zitiert auf den Seiten 24 und 32).

[ACRC20] Ahmad, Wasi ; Chakraborty, Saikat ; Ray, Baishakhi ; Chang, Kai-Wei: A Transformer-Based Approach for Source Code Summarization. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online : Association for Computational Linguistics, Juli 2020, S. 4998–5007 (zitiert auf den Seiten 25, 32 und 47).

[AK15] Allahyari, Mehdi ; Kochut, Krys: Automatic Topic Labeling Using Ontology-Based Topic Models. In: *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, 2015, S. 259–264 (zitiert auf den Seiten 10 und 47).

[APS16] Allamanis, Miltiadis ; Peng, Hao ; Sutton, Charles: A Convolutional Attention Network for Extreme Summarization of Source Code. In: *ICML* (2016) (zitiert auf den Seiten 22 und 24).

[AR13] Aggarwal, Charu C. ; Reddy, Chandan K.: *Data Clustering: Algorithms and Applications*. 1st. Chapman & Hall/CRC, 2013. – ISBN 978–1–4665–5821–2 (zitiert auf Seite 11).

[AZLY19] Alon, Uri ; Zilberstein, Meital ; Levy, Omer ; Yahav, Eran: Code2vec: Learning Distributed Representations of Code. In: *Proceedings of the ACM on Programming Languages* 3 (2019), Januar, Nr. POPL, S. 40:1–40:29. `http://dx.doi.org/10.1145/3290353`. – DOI 10.1145/3290353 (zitiert auf Seite 31).

[BCB16] Bahdanau, Dzmitry ; Cho, Kyunghyun ; Bengio, Yoshua: Neural Machine Translation by Jointly Learning to Align and Translate. In: *arXiv:1409.0473 [cs, stat]* (2016), Mai (zitiert auf Seite 22).

[BNJ03] Blei, David M. ; Ng, Andrew Y. ; Jordan, Michael I.: *Latent Dirichlet Allocation*. März 2003 (zitiert auf den Seiten 5, 8 und 124).

[CAHV15] Carvalho, Nuno R. ; Almeida, José João ; Henriques, Pedro R. ; Varanda, Maria J.: From Source Code Identifiers to Natural Language Terms. In: *Journal of Systems and Software* 100 (2015), Februar, S. 117–128. `http://dx.doi.org/10.1016/j.jss.2014.10.013`. – DOI 10.1016/j.jss.2014.10.013. – ISSN 0164–1212 (zitiert auf Seite 44).

[CEE+10] Carstensen, Kai-Uwe (Hrsg.) ; Ebert, Christian (Hrsg.) ; Ebert, Cornelia (Hrsg.) ; Jekat, Susanne (Hrsg.) ; Langer, Hagen (Hrsg.) ; Klabunde, Ralf (Hrsg.): *Computerlinguistik und Sprachtechnologie: Eine Einführung*. Third. Springer Spektrum, 2010. http://dx.doi.org/10.1007/978-3-8274-2224-8. http://dx.doi.org/10.1007/978-3-8274-2224-8. – ISBN 978–3–8274–2023–7 (zitiert auf den Seiten 3 und 4).

[CGCB14] Chung, Junyoung ; Gulcehre, Caglar ; Cho, KyungHyun ; Bengio, Yoshua: Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. In: *arXiv:1412.3555 [cs]* (2014), Dezember (zitiert auf Seite 14).

[Com08] *Compiler: Prinzipien, Techniken und Werkzeuge*. Pearson Deutschland GmbH, 2008. – ISBN 978–3–8273–7097–6 (zitiert auf Seite 28).

[CZYZ12] Crain, Steven P. ; Zhou, Ke ; Yang, Shuang-Hong ; Zha, Hongyuan: Dimensionality Reduction and Topic Modeling: From Latent Semantic Indexing to Latent Dirichlet Allocation and Beyond. In: Aggarwal, Charu C. (Hrsg.) ; Zhai, ChengXiang (Hrsg.): *Mining Text Data*. Boston, MA : Springer US, 2012. – ISBN 978–1–4614–3223–4, S. 129–161 (zitiert auf Seite 5).

[DDF+90] Deerwester, Scott ; Dumais, Susan T. ; Furnas, George W. ; Landauer, Thomas K. ; Harshman, Richard: Indexing by Latent Semantic Analysis. In: *Journal of the American Society for Information Science* 41 (1990), Nr. 6, S. 391–407. http://dx.doi.org/10.1002/(SICI)1097-4571(199009)41:6<391::AID-ASI1>3.0.CO;2-9. – DOI 10.1002/(SICI)1097–4571(199009)41:6<391::AID–ASI1>3.0.CO;2–9. – ISSN 1097–4571 (zitiert auf den Seiten 7 und 122).

[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*. Gothenburg, Sweden : Association for Computing Machinery, Mai 2018 (MSR '18). – ISBN 978–1–4503–5716–6, S. 38–41 (zitiert auf Seite 48).

[Eur] Eurich, Felix: *Entwurf Und Aufbau Einer Semantischen Repräsentation von Quelltext*, Karlsruher Institut für Technologie (KIT) – IPD Tichy, Master's Thesis (zitiert auf den Seiten 28, 30, 41 und 66).

[FAP18] Fernandes de Mello, Rodrigo ; Antonelli Ponti, Moacir: A Brief Review on Machine Learning. In: Fernandes de Mello, Rodrigo (Hrsg.) ; Antonelli Ponti, Moacir (Hrsg.): *Machine Learning: A Practical Approach on the Statistical Learning Theory*. Cham : Springer International Publishing, 2018. – ISBN 978–3–319–94989–5, S. 1–74 (zitiert auf Seite 11).

[GAL+06] Guthrie, David ; Allison, Ben ; Liu, Wei ; Guthrie, Louise ; Wilks, Yorick: A Closer Look at Skip-Gram Modelling. In: *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*. Genoa, Italy : European Language Resources Association (ELRA), Mai 2006 (zitiert auf Seite 6).

[GBG+18] Grave, Edouard ; Bojanowski, Piotr ; Gupta, Prakhar ; Joulin, Armand ; Mikolov, Tomas: Learning Word Vectors for 157 Languages. In: *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018 (zitiert auf Seite 48).

[GM07] Gabrilovich, Evgeniy ; Markovitch, Shaul: Computing Semantic Relatedness Using Wikipedia-Based Explicit Semantic Analysis. In: *Proceedings*

of the 20th International Joint Conference on Artifical Intelligence. Hyderabad, India : Morgan Kaufmann Publishers Inc., Januar 2007 (IJCAI'07), S. 1606–1611 (zitiert auf Seite 10).

[Gur97] Gurney, Kevin: *An Introduction to Neural Networks*. USA : Taylor & Francis, Inc., 1997. – ISBN 978–1–85728–673–1 (zitiert auf Seite 11).

[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 Seite 17).

[HHKG13] Hulpus, Ioana ; Hayes, Conor ; Karnstedt, Marcel ; Greene, Derek: Unsupervised Graph-Based Topic Labelling Using Dbpedia. In: *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*. Rome, Italy : Association for Computing Machinery, Februar 2013 (WSDM '13). – ISBN 978–1–4503–1869–3, S. 465–474 (zitiert auf Seite 47).

[HLWM20] Haque, Sakib ; LeClair, Alexander ; Wu, Lingfei ; McMillan, Collin: Improved Automatic Summarization of Subroutines via Attention to File Context. In: *arXiv:2004.04881 [cs]* (2020), April. `http://dx.doi.org/10.1145/3379597.3387449`. – DOI 10.1145/3379597.3387449 (zitiert auf den Seiten 24 und 47).

[HLX+18] Hu, Xing ; Li, Ge ; Xia, Xin ; Lo, David ; Jin, Zhi: Deep Code Comment Generation. In: *ICPC '18: Proceedings of the 26th Conference on Program Comprehension*, 2018. – ISBN 978–1–4503–5714–2, S. 200–210 (zitiert auf den Seiten 23, 24 und 25).

[IKCZ16] Iyer, Srinivasan ; Konstas, Ioannis ; Cheung, Alvin ; Zettlemoyer, Luke: Summarizing Source Code Using a Neural Attention Model. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany : Association for Computational Linguistics, August 2016, S. 2073–2083 (zitiert auf den Seiten 22 und 25).

[JAM17] Jiang, Siyuan ; Armaly, Ameer ; McMillan, Collin: Automatically Generating Commit Messages from Diffs Using Neural Machine Translation. In: *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)* (2017). `http://dx.doi.org/10.1109/ASE.2017.8115626`. – DOI 10.1109/ASE.2017.8115626 (zitiert auf Seite 23).

[JM09] Jurafsky, Daniel ; Martin, James H.: *Speech and Language Processing (2nd Edition)*. Upper Saddle River, NJ, USA : Prentice-Hall, Inc., 2009. – ISBN 0–13–187321–0 (zitiert auf Seite 4).

[KHB+07] Koehn, Philipp ; Hoang, Hieu ; Birch, Alexandra ; Callison-Burch, Chris ; Federico, Marcello ; Bertoldi, Nicola ; Cowan, Brooke ; Shen, Wade ; Moran, Christine ; Zens, Richard ; Dyer, Chris ; Bojar, Ondřej ; Constantin, Alexandra ; Herbst, Evan: Moses: Open Source Toolkit for Statistical Machine Translation. In: *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*. Prague, Czech Republic : Association for Computational Linguistics, Juni 2007, S. 177–180 (zitiert auf den Seiten 22 und 23).

[KLJJ04] Korenius, Tuomo ; Laurikkala, Jorma ; Järvelin, Kalervo ; Juhola, Martti: Stemming and Lemmatization in the Clustering of Finnish Text Documents. In: *Proceedings of the Thirteenth ACM International Conference*

on Information and Knowledge Management. Washington, D.C., USA : Association for Computing Machinery, November 2004 (CIKM '04). – ISBN 978–1–58113–874–0, S. 625–633 (zitiert auf Seite 4).

[LJM19] LeClair, Alexander ; Jiang, Siyuan ; McMillan, Collin: A Neural Model for Generating Natural Language Summaries of Program Subroutines. In: *arXiv:1902.01954 [cs]* (2019), Februar (zitiert auf den Seiten 24 und 31).

[MB17] Mahmoud, Anas ; Bradshaw, Gary: Semantic Topic Models for Source Code Analysis. In: *Empirical Software Engineering* 22 (2017), August, Nr. 4, S. 1965–2000. http://dx.doi.org/10.1007/s10664-016-9473-1. – DOI 10.1007/s10664–016–9473–1. – ISSN 1382–3256, 1573–7616 (zitiert auf den Seiten 20 und 46).

[MC13] Movshovitz-Attias, Dana ; Cohen, William W.: Natural Language Models for Predicting Programming Comments. In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Sofia, Bulgaria : Association for Computational Linguistics, August 2013, S. 35–40 (zitiert auf Seite 19).

[MCCD13] Mikolov, Tomas ; Chen, Kai ; Corrado, Greg ; Dean, Jeffrey: Efficient Estimation of Word Representations in Vector Space. In: *arXiv:1301.3781 [cs]* (2013), September (zitiert auf Seite 13).

[MCF18] Mäntylä, Mika ; Claes, Maëlick ; Farooq, Umar: Measuring LDA Topic Stability from Clusters of Replicated Runs. In: *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (2018), Oktober, S. 1–4. http://dx.doi.org/10.1145/3239235.3267435. – DOI 10.1145/3239235.3267435 (zitiert auf den Seiten 56 und 88).

[MM16] McBurney, Paul W. ; McMillan, Collin: Automatic Source Code Summarization of Context for Java Methods. In: *IEEE Transactions on Software Engineering* 42 (2016), Februar, Nr. 2, S. 103–119. http://dx.doi.org/10.1109/TSE.2015.2465386. – DOI 10.1109/TSE.2015.2465386. – ISSN 1939–3520 (zitiert auf Seite 30).

[MSC+13] Mikolov, Tomas ; Sutskever, Ilya ; Chen, Kai ; Corrado, Greg ; Dean, Jeffrey: Distributed Representations of Words and Phrases and Their Compositionality. In: *arXiv:1310.4546 [cs, stat]* (2013), Oktober (zitiert auf Seite 13).

[NNN16] Nguyen, Hoan A. ; Nguyen, Anh T. ; Nguyen, Tien N.: Using Topic Model to Suggest Fine-Grained Source Code Changes. In: *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2016, S. 200–210 (zitiert auf Seite 19).

[PP17] Pirapuraj, P. ; Perera, Indika: Analyzing Source Code Identifiers for Code Reuse Using NLP Techniques and WordNet. In: *2017 Moratuwa Engineering Research Conference (MERCon)*, 2017, S. 105–110 (zitiert auf Seite 44).

[PVG+11] Pedregosa, F. ; Varoquaux, G. ; Gramfort, A. ; Michel, V. ; Thirion, B. ; Grisel, O. ; Blondel, M. ; Prettenhofer, P. ; Weiss, R. ; Dubourg, V. ; Vanderplas, J. ; Passos, A. ; Cournapeau, D. ; Brucher, M. ; Perrot, M. ; Duchesnay, E.: Scikit-Learn: Machine Learning in Python. In: *Journal of Machine Learning Research* 12 (2011), S. 2825–2830 (zitiert auf Seite 61).

[RCW15] Rush, Alexander M. ; Chopra, Sumit ; Weston, Jason: A Neural Attention Model for Abstractive Sentence Summarization. In: *arXiv:1509.00685 [cs]* (2015), September (zitiert auf Seite 22).

[RMM+14] Rodeghero, Paige ; McMillan, Collin ; McBurney, Paul W. ; Bosch, Nigel ; D'Mello, Sidney: Improving Automated Source Code Summarization via an Eye-Tracking Study of Programmers. In: *Proceedings of the 36th International Conference on Software Engineering*. New York, NY, USA : Association for Computing Machinery, Mai 2014 (ICSE 2014). – ISBN 978–1–4503–2756–5, S. 390–401 (zitiert auf Seite 34).

[ŘS10] Řehůřek, Radim ; Sojka, Petr: Software Framework for Topic Modelling with Large Corpora. In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta : ELRA, Mai 2010, S. 45–50 (zitiert auf Seite 60).

[SFC+17] Sennrich, Rico ; Firat, Orhan ; Cho, Kyunghyun ; Birch, Alexandra ; Haddow, Barry ; Hitschler, Julian ; Junczys-Dowmunt, Marcin ; Läubli, Samuel ; Miceli Barone, Antonio V. ; Mokry, Jozef ; Nădejde, Maria: Nematus: A Toolkit for Neural Machine Translation. In: *Proceedings of the Software Demonstrations of the 15th Conference of the European Chapter of the Association for Computational Linguistics*. Valencia, Spain : Association for Computational Linguistics, April 2017, S. 65–68 (zitiert auf Seite 23).

[SVL14] Sutskever, Ilya ; Vinyals, Oriol ; Le, Quoc V.: Sequence to Sequence Learning with Neural Networks. In: *arXiv:1409.3215 [cs]* (2014), Dezember (zitiert auf Seite 14).

[Tha18] Tharwat, Alaa: Classification Assessment Methods. In: *Applied Computing and Informatics* (2018), August. `http://dx.doi.org/10.1016/j.aci.2018.08.003`. – DOI 10.1016/j.aci.2018.08.003. – ISSN 2210–8327 (zitiert auf Seite 15).

[TJBB05] Teh, Yee W. ; Jordan, Michael I. ; Beal, Matthew J. ; Blei, David M.: Sharing Clusters among Related Groups: Hierarchical Dirichlet Processes. In: *Advances in Neural Information Processing Systems*, 2005, S. 1385–1392 (zitiert auf Seite 9).

[TWB+18] Tufano, Michele ; Watson, Cody ; Bavota, Gabriele ; Di Penta, Massimiliano ; White, Martin ; Poshyvanyk, Denys: Deep Learning Similarities from Different Representations of Source Code. In: *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, 2018. – ISSN 2574–3864, S. 542–553 (zitiert auf den Seiten 29 und 31).

[VSP+17] Vaswani, Ashish ; Shazeer, Noam ; Parmar, Niki ; Uszkoreit, Jakob ; Jones, Llion ; Gomez, Aidan N. ; Kaiser, Lukasz ; Polosukhin, Illia: Attention Is All You Need. In: *arXiv:1706.03762 [cs]* (2017), Dezember (zitiert auf Seite 14).

[WG08] Wang, Xiaogang ; Grimson, Eric: Spatial Latent Dirichlet Allocation. In: Platt, J. C. (Hrsg.) ; Koller, D. (Hrsg.) ; Singer, Y. (Hrsg.) ; Roweis, S. T. (Hrsg.): *Advances in Neural Information Processing Systems 20*. Curran Associates, Inc., 2008, S. 1577–1584 (zitiert auf Seite 9).

[Wil06] Willett, Peter: The Porter Stemming Algorithm: Then and Now. In: *Program electronic library and information systems* 40 (2006), Juli. `http://dx.doi.org/10.1108/00330330610681295`. – DOI 10.1108/00330330610681295 (zitiert auf Seite 4).

[WKHT19] Weigelt, Sebastian ; Keim, Jan ; Hey, Tobias ; Tichy, Walter F.: Unsupervised Multi-Topic Labeling for Spoken Utterances. In: *Proceedings of The First IEEE International Conference on Humanized Computing and Communication (HCC) (to Appear)*. Laguna Hills, California, September 2019 (zitiert auf Seite 7).

[ZMG08] Zesch, Torsten ; Müller, Christof ; Gurevych, Iryna: Using Wiktionary for Computing Semantic Relatedness. In: *AAAI*, 2008 (zitiert auf Seite 10).

[ZSA+16] Zhang, Wei E. ; Sheng, Quan ; Abebe, Ermyas ; Ali Babar, Muhammad ; Zhou, Andi: Mining Source Code Topics Through Topic Model and Words Embedding, 2016. – ISBN 978–3–319–49585–9, S. 664–676 (zitiert auf den Seiten 20 und 46).

Anhang

A Evaluationsfragebogen

Im folgenden sind die Aufgabenstellungen der Nutzerstudie in den Abbildungen A.1 und A.2 schematisch dargestellt. Dabei geht es nicht um die konkret genutzten Daten, sondern darum wie die Aufgaben den Teilnehmern präsentiert wurden.

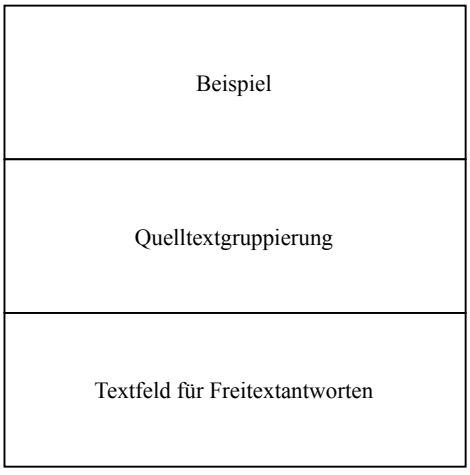


Abbildung A.1: Schematische Darstellung des Fragebogen der ersten Aufgabe. Die Teilnehmer haben je 9 Aufgaben in diesem Format bearbeitet.

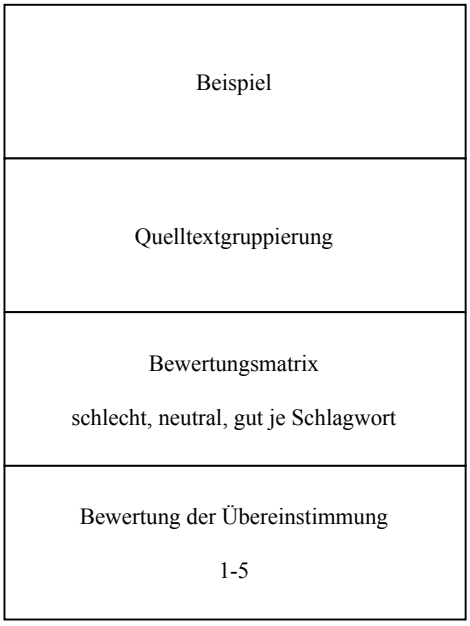


Abbildung A.2: Schematische Darstellung des Fragebogen der zweiten Aufgabe. Die Teilnehmer haben je 9 Aufgaben in diesem Format bearbeitet.

B Quelltextelemente im Evaluationskorpus

In diesem Abschnitt sind alle Quelltextgruppierungen aus dem Evaluationskorpus und deren Programmelemente zu finden. Aus Gründen der Übersichtlichkeit wurden die Paketpfade zu den Programmelementen selbst entfernt, da diese bei fast allen Programmelementen aus einem Projekt gleich sind. Die Paketpfade von Parametern wurden ebenfalls entfernt.

Der entfernte Basispfad für iTrust lautet `itrust`

B.1 iTrust 1

Programmelement
HtmlEncoder:encode(String)
HtmlEncoder:URLOnSite(String)

Tabelle B.1: Programmelemente der Gruppierung iTrust 1

B.2 iTrust 2

Programmelement
exception.DBException:getExtendedMessage()
exception.iTrustException:getExtendedMessage()

Tabelle B.2: Programmelemente der Gruppierung iTrust 2

B.3 iTrust 3

Programmelement
action.ViewExpiredPrescriptionsAction:getPatient(long)
action.ViewExpiredPrescriptionsAction:getPrescribingDoctor(PrescriptionBean)

Tabelle B.3: Programmelemente der Gruppierung iTrust 3

B.4 iTrust 4

Programmelement
beans.DiagnosisBean:setDescription(String)
beans.DiagnosisBean:setICDCode(String)
beans.DiagnosisBean:getFormattedDescription()
beans.DiagnosisBean:getOvDiagnosisID()

Tabelle B.4: Programmelemente der Gruppierung iTrust 4

B.5 iTrust 5

Programmelement
beans.MedicationBean:equals(MedicationBean)
beans.MedicationBean:getNDCodeFormatted()
beans.MedicationBean:hashCode()

Tabelle B.5: Programmelemente der Gruppierung iTrust 5

B.6 iTrust 6

Programmelement
exception.ErrorList:addIfNotNull(String)
validate.BeanValidator:checkFormat(String,String,ValidationFormat,boolean)
validate.BeanValidator:checkInt(jString,String,int,int,boolean)

Tabelle B.6: Programmelemente der Gruppierung iTrust 6

B.7 iTrust 7

Programmelement
beans.PatientBean:setMessageFilter(String)
beans.PatientBean:setIcPhone3(String)
beans.PatientBean:setIcPhone2(String)
beans.PatientBean:setIcPhone1(String)
beans.PatientBean:setEmergencyPhone3(String)
beans.PatientBean:setEmergencyPhone2(String)
beans.PatientBean:setEmergencyPhone1(String)
beans.PatientBean:setZip2(String)
beans.PatientBean:setZip1(String)
beans.PatientBean:setTopicalNotes(String)
beans.PatientBean:setStreetAddress2(String)
beans.PatientBean:setStreetAddress1(String)
beans.PatientBean:setState(String)
beans.PatientBean:setPhone3(String)
beans.PatientBean:setPhone2(String)
beans.PatientBean:setPhone1(String)
beans.PatientBean:setMotherMID(String)
beans.PatientBean:setIcState(String)
beans.PatientBean:setIcZip2(String)
beans.PatientBean:setIcZip1(String)
beans.PatientBean:setIcName(String)
beans.PatientBean:setCreditCardNumber(String)
beans.PatientBean:setCreditCardType(String)
beans.PatientBean:setIcID(String)
beans.PatientBean:setIcCity(String)
beans.PatientBean:setIcAddress2(String)
beans.PatientBean:setLastName(String)
beans.PatientBean:setIcAddress1(String)
beans.PatientBean:setFatherMID(String)
beans.PatientBean:setCity(String)
beans.PatientBean:setFirstName(String)
beans.PatientBean:setEmail(String)
beans.PatientBean:setCauseOfDeath(String)
beans.PatientBean:setEmergencyName(String)

Tabelle B.7: Programmelemente der Gruppierung iTrust 7

B.8 iTrust 8

Programmelement
beans.MessageBean:getFrom()
beans.MessageBean:getBody()
beans.MessageBean:getSubject()
beans.MessageBean:getSentDate()
beans.MessageBean:setParentMessageId(long)
beans.MessageBean:setMessageId(long)
beans.MessageBean:getRead()
beans.MessageBean:setSubject(String)
beans.MessageBean:setBody(String)
beans.MessageBean:setTo(long)
beans.MessageBean:setFrom(long)
beans.MessageBean:setSentDate(Timestamp)
beans.MessageBean:setRead(int)

Tabelle B.8: Programmelemente der Gruppierung iTrust 8

B.9 iTrust 9

Programmelement
dao.DAOFactory:getMessageDAO()
action.ViewMyMessagesAction:getName(long)
action.ViewMyMessagesAction:getPersonnelName(long)
action.ViewMyMessagesAction:filterMessages(List,String)
action.ViewMyMessagesAction:setRead(MessageBean)
action.ViewMyMessagesAction:validateAndCreateFilter(String)
action.ViewMyMessagesAction:getAllMyMessages()
action.ViewMyMessagesAction:getAllMySentMessages()
action.ViewMyMessagesAction:getAllMyMessagesTimeAscending()
action.ViewMyMessagesAction:getAllMySentMessagesTimeAscending()
action.ViewMyMessagesAction:getAllMyMessagesNameDescending()
action.ViewMyMessagesAction:getAllMySentMessagesNameDescending()
action.ViewMyMessagesAction:getAllMyMessagesNameAscending()
action.ViewMyMessagesAction:getAllMySentMessagesNameAscending()
action.ViewMyMessagesAction

Tabelle B.9: Programmelemente der Gruppierung iTrust 9

B.10 email 1

Programmelement
Email:getSubject()
EmailTest:testSetSubjectValid()
EmailTest:testEndOflineCharactersInSubjectAreReplacedWithSpaces()

Tabelle B.10: Programmelemente der Gruppierung email 1

B.11 email 2

Programmelement
resolver.DataSourceClassPathResolverTest:testResolvingClassPathLenient()
resolver.DataSourceClassPathResolverTest:testResolvingClassPathNonLenient()

Tabelle B.11: Programmelemente der Gruppierung email 2

B.12 email 3

Programmelement
resolver.DataSourceFileResolver:getBaseDir()
resolver.DataSourceFileResolver:resolve(jString,boolean)
resolver.DataSourceFileResolver

Tabelle B.12: Programmelemente der Gruppierung email 3

B.13 email 4

Programmelement
mocks.MockSimpleEmail:getMsg()
SimpleEmailTest:setUpSimpleEmailTest()
SimpleEmailTest:testGetSetMsg()
SimpleEmailTest:testSend()
SimpleEmailTest:testDefaultMimeCharset()
mocks.MockSimpleEmail
SimpleEmailTest

Tabelle B.13: Programmelemente der Gruppierung email 4

B.14 email 5

Programmelement
EmailAttachment:getDescription()
MultiPartEmail:attach(EmailAttachment)
MultiPartEmail:attach(URL,String,String,String)
EmailAttachment:getDisposition()
EmailAttachment:setDisposition(String)

Tabelle B.14: Programmelemente der Gruppierung email 5

B.15 email 6

Programmelement
HtmlEmail$InlineImage:getDataSource()
HtmlEmail$InlineImage:getCid()
HtmlEmail$InlineImage:getMbp()
HtmlEmail$InlineImage:equals(Object)
HtmlEmail$InlineImage:hashCode()
HtmlEmail

Tabelle B.15: Programmelemente der Gruppierung email 6

B.16 email 7

Programmelement
Email:checkSessionAlreadyInitialized()
Email:addTo(String[])
Email:addTo(String,String)
Email:addCc(String[])
Email:addCc(String,String)
Email:addBcc(String[])
Email:addBcc(String,String)
Email:createInternetAddress(String,String,String)
Email:setFrom(String,String,String)
Email:setFrom(String,String)
Email:addBcc(String,String,String)
Email:addTo(String,String,String)
Email:addCc(String,String,String)
Email:addCc(String)
Email:addBcc(String)
Email:addReplyTo(String,String,String)
Email:addReplyTo(String)
Email:addReplyTo(String,String)

Tabelle B.16: Programmelemente der Gruppierung email 7

B.17 email 8

Programmelement
util.MimeMessageParserTest:testParseInlineCID()
util.MimeMessageParserTest:testParseNoHeaderSeperatorWithOutOfMemory()
util.MimeMessageParserTest:testParseHtmlEmailWithAttachments()
util.MimeMessageParserTest:testParseSimpleReplyEmail()
util.MimeMessageParserTest:testAttachmentOnly()
util.MimeMessageParserTest:testParseMultipartReport()
util.MimeMessageParserTest:testParseSimpleEmail()
util.MimeMessageParserTest:testParseHtmlEmailWithAttachmentAndEncodedFilename()
util.MimeMessageParserTest:testParseHtmlEmailWithHtmlAttachment()
util.MimeMessageParserTest:testMultipartTextAttachment()
util.MimeMessageParserTest:testMultipartTextAttachmentOnly()
util.MimeMessageParserTest:testParseCreatedHtmlEmailWithNoContent()
util.MimeMessageParserTest:testParseCreatedHtmlEmailWithTextContent()
util.MimeMessageParserTest:testParseCreatedHtmlEmailWithMixedContent()
settings.EmailConfiguration
util.MimeMessageParser

Tabelle B.17: Programmelemente der Gruppierung email 8

B.18 email 9

Programmelement
EmailTest:testSendNoHostName()
EmailTest:testSendBadAuthSet()
EmailTest:testSendCorrectSmtpPortContainedInException()
EmailTest:testSendFromSetInSession()
EmailTest:testSendFromNotSet()
EmailTest:testSendDestinationNotSet()
EmailTest:testSetPopBeforeSmtp()
EmailTest:testSetContentObject()
EmailTest:testSendBadHostName()
EmailTest:testSupportForInternationalDomainNames()
EmailTest:testFoldingHeaders()
EmailTest:testCorrectContentTypeForPNG()
EmailTest:testDefaultCharsetIgnoredByNonTextContent()
EmailTest:testDefaultCharsetAppliesToTextContent()
EmailTest:testDefaultCharsetCanBeOverriddenByContentType()
EmailTest

Tabelle B.18: Programmelemente der Gruppierung email 9

C Evaluationsergebnisse

In diesem Abschnitt des Anhangs sind alle Evaluationsergebnisse für die Quelltextgruppierungen des Evaluationskorpus zu finden.

C.1 iTrust 1

Teilnehmer	Beschreibung	Übereinstimmung
1	HTML, Encoder, URL	4
2	html, encoding, escaping	4
3	HTML, encoder, URL	5
4	html, encode, url, check	4
5	encode, html, url	4
6	encoding, url, website, html	5
7	HTML, encoder, URL, Check	4
8	HTML, encode, input url	4

Tabelle C.19: Freitextantworten für Gruppierung iTrust 1 und Bewertung der semantischen Übereinstimmung je Teilnehmer

Normalisiertes Wort	Anzahl Nennungen
HTML	**8/8**
encoder/encoding/encode	**8/8**
url	**7/8**
check	2/8
website	1/8
input	1/8
escaping	1/8

Tabelle C.20: Normalisierte Nutzerantworten und ihre Nennungshäufigkeit für die Gruppierung iTrust 1

t \ s	checks	**encoder**	**html**	**url**	input
1	2	2	2	2	1
2	1	2	2	2	1
3	2	2	2	2	2
4	0	2	2	2	0
5	1	2	2	2	2
6	2	2	2	2	2
7	1	2	2	2	0
8	0	2	2	2	1

Tabelle C.21: Bewertung der Schlagworte (s) für Gruppierung iTrust 1 je Teilnehmer (t).

C.2 iTrust 2

Teilnehmer	Beschreibung	Übereinstimmung
1	Exception, Message, Database, iTrust	3
2	database, exception, interface, certificate, description	4
3	exception, message	4
4	database, exception, message, trust	5
5	Exception, detailed, message	5
6	message,	5
7	Database, Exception, Error Message,	5
8	Exception, message, get	3

Tabelle C.22: Freitextantworten für Gruppierung iTrust 2 und Bewertung der semantischen Übereinstimmung je Teilnehmer

Normalisiertes Wort	Anzahl Nennungen
exception	**7/8**
message	**7/8**
database	4/8
iTrust	2/8
interface	1/8
certificate	1/8
description	1/8
detailed	1/8
get	1/8
error	1/8

Tabelle C.23: Normalisierte Nutzerantworten und ihre Nennungshäufigkeit für die Gruppierung iTrust 2

t \ s	**message**	**exception**	get
1	1	2	0
2	2	2	0
3	2	2	1
4	2	2	2
5	2	2	2
6	2	2	2
7	2	2	2
8	2	1	1

Tabelle C.24: Bewertung der Schlagworte (s) für Gruppierung iTrust 2 je Teilnehmer (t).

C.3 iTrust 3

Teilnehmer	Beschreibung	Übereinstimmung
1	Prescription, Expired, Patient, Doctor	2
2	prescription,patient,doctor,retrieval	4
3	action, patient, doctor, prescription, expiration	3
4	patient, ”prescribing doctor”, prescription, expired	5
5	patient, doctor, prescription, expired	3
6	prescription, patient, hospital, doctor	3
7	Health Record,Prescription,Expired,Patient,Doctor	4
8	Prescription, patient, doctor, expired	4

Tabelle C.25: Freitextantworten für Gruppierung iTrust 3 und Bewertung der semantischen Übereinstimmung je Teilnehmer

Normalisiertes Wort	Anzahl Nennungen
prescription	**8/8**
patient	**8/8**
doctor	**8/8**
expired	**6/8**
retrieval	1/8
action	1/8
hospital	1/8
health	1/8
record	1/8

Tabelle C.26: Normalisierte Nutzerantworten und ihre Nennungshäufigkeit für die Gruppierung iTrust 3

t \ s	bean	**prescription**	get	view	**expired**	**patient**	action
1	0	2	0	0	0	2	0
2	2	2	0	2	1	2	2
3	2	2	1	1	1	2	1
4	0	2	2	2	2	2	2
5	0	2	2	2	2	2	2
6	1	2	2	1	1	2	2
7	2	2	2	2	2	2	2
8	2	2	1	1	2	2	1

Tabelle C.27: Bewertung der Schlagworte (s) für Gruppierung iTrust 3 je Teilnehmer (t).

C.4 iTrust 4

Teilnehmer	Beschreibung	Übereinstimmung
1	diagnosis, health, modify	4
2	diagnosis,persistence,id,description	3
3	diagnosis, code, description, formatted, ID	4
4	diagnosis, code	4
5	diagnosis. description	2
6	destiprion, medicine, diagnose, visits	4
7	Health Record,Diagnosis,ICDCode,Description,modify	4
8	Diagnosis, description	3

Tabelle C.28: Freitextantworten für Gruppierung iTrust 4 und Bewertung der semantischen Übereinstimmung je Teilnehmer

Normalisiertes Wort	Anzahl Nennungen
diagnosis	**8/8**
description	**6/8**
code	3/8
id	2/8
health	2/8
modify	2/8
persistence	1/8
formatted	1/8
medicine	1/8
visits	1/8
record	1/8
icd	1/8

Tabelle C.29: Normalisierte Nutzerantworten und ihre Nennungshäufigkeit für die Gruppierung iTrust 4

t \ s	visit	bean	editing	**diagnosis**	office	icd	get	set	code
1	1	0	0	2	1	2	1	1	1
2	0	2	0	2	1	2	0	0	0
3	1	2	1	2	1	2	1	1	2
4	1	1	0	2	0	2	2	2	2
5	0	0	1	2	0	2	2	2	2
6	2	2	1	2	2	2	2	2	2
7	1	2	1	2	1	2	2	2	2
8	0	2	1	2	0	1	1	1	1

Tabelle C.30: Bewertung der Schlagworte (s) für Gruppierung iTrust 4 je Teilnehmer (t).

C.5 iTrust 5

Teilnehmer	Beschreibung	Übereinstimmung
1	Medication, Compare	4
2	medication,persistence,id	3
3	medication, code	5
4	medication, code	5
5	medication	3
6	medication, patient, description, medicine code	5
7	Health Record,Medication,NDCode,compare	3
8	Medication	2

Tabelle C.31: Freitextantworten für Gruppierung iTrust 5 und Bewertung der semantischen Übereinstimmung je Teilnehmer

Normalisiertes Wort	Anzahl Nennungen
medication	**8/8**
compare	**4/8**
persistence	2/8
id	1/8
patient	1/8
description	1/8
medicine	1/8
health	1/8
record	1/8
ND	1/8

Tabelle C.32: Normalisierte Nutzerantworten und ihre Nennungshäufigkeit für die Gruppierung iTrust 5

t \ s	code	bean	equals	get	**medication**
1	2	0	0	0	2
2	0	2	0	0	2
3	2	2	2	1	2
4	2	1	2	2	2
5	2	0	1	2	2
6	2	2	2	2	2
7	2	2	2	2	2
8	0	1	0	0	2

Tabelle C.33: Bewertung der Schlagworte (s) für Gruppierung iTrust 5 je Teilnehmer (t).

C.6 iTrust 6

Teilnehmer	Beschreibung	Übereinstimmung
1	validation, error, checks	3
2	error,reporting,formating,validation	2
3	error, list, validator, check, format	3
4	error, validator, format	4
5	error, validate, check	2
6	form, website,	5
7	Error List,Validation,Checker,Format,Integer	4
8	Error, validate, format, message	2

Tabelle C.34: Freitextantworten für Gruppierung iTrust 6 und Bewertung der semantischen Übereinstimmung je Teilnehmer

Normalisiertes Wort	Anzahl Nennungen
error	**7/8**
validation/validator/validate	**7/8**
format/formating	**5/8**
check/checker/checks	**4/8**
list	2/8
website	1/8
integer	1/8
message	1/8
reporting	1/8
form	1/8

Tabelle C.35: Normalisierte Nutzerantworten und ihre Nennungshäufigkeit für die Gruppierung iTrust 6

t \ s	message	string	**check**	bean	value	**error**	list	nullable
1	1	1	2	0	0	2	2	1
2	1	0	1	2	0	2	0	0
3	2	1	2	1	1	2	0	2
4	2	1	2	2	1	2	1	1
5	2	0	2	0	2	2	2	0
6	2	1	2	2	2	2	1	1
7	2	1	2	2	1	2	2	2
8	0	0	0	2	0	0	0	0

Tabelle C.36: Bewertung der Schlagworte (s) für Gruppierung iTrust 6 je Teilnehmer (t).

C.7 iTrust 7

Teilnehmer	Beschreibung	Übereinstimmung
1	patient, modify, personal data	4
2	patient,persistence	3
3	patient, filter, death, address, contact, ID	3
4	”patient information”	5
5	patient, information	3
6	patient, patient personal data,	5
7	Health,Record,Patient,File,Information,Object	5
8	Patient, set, zip, phone, address	3

Tabelle C.37: Freitextantworten für Gruppierung iTrust 7 und Bewertung der semantischen Übereinstimmung je Teilnehmer

Normalisiertes Wort	Anzahl Nennungen
patient	**8/8**
information	3/8
personal	2/8
address	2/8
data	2/8
modify	1/8
filter	1/8
death	1/8
contact	1/8
id	1/8
record	1/8
file	1/8
object	1/8
set	1/8
zip	1/8
phone	1/8
persistence	1/8
health	1/8

Tabelle C.38: Normalisierte Nutzerantworten und ihre Nennungshäufigkeit für die Gruppierung iTrust 7

t \ s	bean	**patient**	set	phone	message
1	0	2	0	2	2
2	2	2	0	1	1
3	1	1	2	2	0
4	1	2	2	2	2
5	0	2	2	2	2
6	2	2	2	2	2
7	2	2	2	2	2
8	2	2	0	1	1

Tabelle C.39: Bewertung der Schlagworte (s) für Gruppierung iTrust 7 je Teilnehmer (t).

C.8 iTrust 8

Teilnehmer	Beschreibung	Übereinstimmung
1	message, modify, id,	4
2	message,persistence,subject,body,recipient	3
3	message, id, parent, to, from, subject, body, date, read	4
4	message	5
5	message	2
6	message data	4
7	Message,Object	4
8	Message, subject, body, from, to	3

Tabelle C.40: Freitextantworten für Gruppierung iTrust 8 und Bewertung der semantischen Übereinstimmung je Teilnehmer

Normalisiertes Wort	Anzahl Nennungen
message	**8/8**
subject	3/8
body	3/8
id	2/8
to	2/8
from	2/8
date	1/8
read	1/8
data	1/8
object	1/8
modify	1/8
recipient	1/8
parent	1/8
persistence	1/8

Tabelle C.41: Normalisierte Nutzerantworten und ihre Nennungshäufigkeit für die Gruppierung iTrust 8

s / t	**message**	bean	set	mid	date	get
1	2	0	0	2	2	0
2	2	2	0	1	1	0
3	2	2	2	0	1	2
4	2	1	2	2	2	2
5	2	0	2	0	2	2
6	2	2	2	0	2	2
7	2	2	2	2	2	2
8	1	2	0	0	1	0

Tabelle C.42: Bewertung der Schlagworte (s) für Gruppierung iTrust 8 je Teilnehmer (t).

C.9 iTrust 9

Teilnehmer	Beschreibung	Übereinstimmung
1	view, messages, sort, filter	2
2	view,messages	3
3	DAO, action, messages, order, filter, name, personnel, read	3
4	”message factory”, message, action	5
5	message, factory, message types, message actions	3
6	message metadata	4
7	Message,Factory,Actions	3
8	Message, get	2

Tabelle C.43: Freitextantworten für Gruppierung iTrust 9 und Bewertung der semantischen Übereinstimmung je Teilnehmer

Normalisiertes Wort	Anzahl Nennungen
message/messages	**8/8**
action	**4/8**
factory	3/8
view	2/8
filter	2/8
personnel	1/8
types	1/8
metadata	1/8
get	1/8
read	1/8
order	1/8
name	1/8
DAO	1/8
sort	1/8

Tabelle C.44: Normalisierte Nutzerantworten und ihre Nennungshäufigkeit für die Gruppierung iTrust 9

t \ s	**messages**	get	bean	**action**	view	set	mid	user
1	2	0	0	0	0	0	2	2
2	2	0	2	1	2	0	2	2
3	2	2	2	2	1	0	1	0
4	2	2	0	2	2	2	2	2
5	2	2	0	2	2	2	0	1
6	2	2	2	2	2	2	0	2
7	2	2	1	2	2	2	1	1
8	1	0	1	0	0	0	0	0

Tabelle C.45: Bewertung der Schlagworte (s) für Gruppierung iTrust 9 je Teilnehmer (t).

C.10 email 1

Teilnehmer	Beschreibung	Übereinstimmung
1	email, subject, validation	5
2	email, test, replace, subject	5
3	validation, email, subject, test	4
4	email,test,subject	5
5	email, subject, test	5
6	email, subject, test	5
7	e-mail, test, subject	4
8	email, test, validity, subject	5

Tabelle C.46: Freitextantworten für Gruppierung email 1 und Bewertung der semantischen Übereinstimmung je Teilnehmer

Normalisiertes Wort	Anzahl Nennungen
email	**8/8**
subject	**8/8**
test	**7/8**
validation/validity	3/8
replace	1/8

Tabelle C.47: Normalisierte Nutzerantworten und ihre Nennungshäufigkeit für die Gruppierung email 1

t \ s	**test**	**email**	**subject**
1	2	2	2
2	2	2	2
3	2	2	2
4	2	2	2
5	2	2	2
6	2	2	2
7	2	2	2
8	2	2	2

Tabelle C.48: Bewertung der Schlagworte (s) für Gruppierung email 1 je Teilnehmer (t).

C.11 email 2

Teilnehmer	Beschreibung	Übereinstimmung
1	classpath, resolution, test	5
2	Data, Resolver, test, lenient	5
3	test, resolver, classpath	5
4	data,source,classpath,test,lenient	4
5	datasource, resolve, test	4
6	class, path, data, source, test	4
7	test, data source, class path	4
8	test, classpath, resolver, leniency	3

Tabelle C.49: Freitextantworten für Gruppierung email 2 und Bewertung der semantischen Übereinstimmung je Teilnehmer

Normalisiertes Wort	Anzahl Nennungen
test	**8/8**
classpath	**6/8**
data	**5/8**
resolution/resolve/resolver	5/8
source	4/8
lenient/leniency	3/8

Tabelle C.50: Normalisierte Nutzerantworten und ihre Nennungshäufigkeit für die Gruppierung email 2

t \ s	**resolver**	**data**	**source**	path	**test**
1	2	2	2	2	2
2	2	2	2	2	2
3	2	2	2	2	2
4	1	1	1	2	2
5	2	1	1	2	2
6	2	2	1	2	2
7	2	2	2	2	2
8	2	1	1	1	1

Tabelle C.51: Bewertung der Schlagworte (s) für Gruppierung email 2 je Teilnehmer (t).

C.12 email 3

Teilnehmer	Beschreibung	Übereinstimmung
1	Datasource, file, resolution	5
2	Data, Resolver, File, lenient	5
3	data, source, resolver, file	5
4	data,source,file,resolver	4
5	datasource, resolve, file	3
6	data source, file, directory, resolve	4
7	data source, file, file path, resolver	4
8	data, source, resolver, directory	4

Tabelle C.52: Freitextantworten für Gruppierung email 3 und Bewertung der semantischen Übereinstimmung je Teilnehmer

Normalisiertes Wort	Anzahl Nennungen
data	**8/8**
resolver/resolve/resolution	**8/8**
source	**7/8**
file	**7/8**
directory	2/8
lenient	1/8
path	1/8

Tabelle C.53: Normalisierte Nutzerantworten und ihre Nennungshäufigkeit für die Gruppierung email 3

t \ s	url	**resolver**	**data**	**source**	resource	**file**
1	2	2	2	2	2	2
2	2	2	2	2	2	2
3	1	2	2	2	2	2
4	1	2	2	2	2	2
5	0	2	1	1	0	2
6	0	2	2	2	1	2
7	1	2	2	2	2	2
8	0	2	1	1	0	2

Tabelle C.54: Bewertung der Schlagworte (s) für Gruppierung email 3 je Teilnehmer (t).

C.13 email 4

Teilnehmer	Beschreibung	Übereinstimmung
1	email, mock, test, message, address	5
2	Mock, Email, mime, test, address	5
3	test, email, content,	2
4	mock,email,test,send,setup,message	5
5	email, test, send, message	4
6	email, simple, test, message	5
7	testing, e-mail, message, adress	3
8	mock, email, messages, test	4

Tabelle C.55: Freitextantworten für Gruppierung email 4 und Bewertung der semantischen Übereinstimmung je Teilnehmer

Normalisiertes Wort	Anzahl Nennungen
email	**8/8**
test/testing	**8/8**
message	**6/8**
mock	**4/8**
address	2/8
send	2/8
setup	1/8
simple	1/8
content	1/8
mime	1/8

Tabelle C.56: Normalisierte Nutzerantworten und ihre Nennungshäufigkeit für die Gruppierung email 4

t \ s	**test**	**email**
1	2	2
2	2	2
3	2	2
4	2	2
5	2	2
6	2	2
7	2	2
8	1	2

Tabelle C.57: Bewertung der Schlagworte (s) für Gruppierung email 4 je Teilnehmer (t).

C.14 email 5

Teilnehmer	Beschreibung	Übereinstimmung
1	email, attachment, description, disposition	3
2	Email, attachment, description	2
3	email, attachment, type, description	4
4	email,attachment,description,disposition	3
5	email, attachment, header	3
6	email, attachment, disposition, description	3
7	e-mail attachment, meta data	3
8	email, attachment, description, disposition	4

Tabelle C.58: Freitextantworten für Gruppierung email 5 und Bewertung der semantischen Übereinstimmung je Teilnehmer

Normalisiertes Wort	Anzahl Nennungen
email	**8/8**
attachment	**8/8**
description	**6/8**
disposition	**4/8**
type	1/8
header	1/8
metadata	1/8

Tabelle C.59: Normalisierte Nutzerantworten und ihre Nennungshäufigkeit für die Gruppierung email 5

t \ s	part	**email**	url	**disposition**	set	get	multi	**attachment**
1	0	2	0	2	2	2	0	2
2	0	2	1	2	0	0	0	2
3	0	2	0	2	2	2	0	2
4	1	2	1	2	2	2	0	2
5	0	2	1	2	2	2	0	2
6	0	2	0	2	2	2	0	2
7	0	2	1	2	2	2	0	2
8	0	2	0	2	1	1	0	2

Tabelle C.60: Bewertung der Schlagworte (s) für Gruppierung email 5 je Teilnehmer (t).

C.15 email 6

Teilnehmer	Beschreibung	Übereinstimmung
1	email, html, images, body, mime	3
2	html, email, mime,	4
3	email, html, image, message, inline	4
4	html,email,inline,image,message	2
5	email, body, image	4
6	email, html, cid, image, message	3
7	e-mail, html, image, message	5
8	email, verification, message, html	4

Tabelle C.61: Freitextantworten für Gruppierung email 6 und Bewertung der semantischen Übereinstimmung je Teilnehmer

Normalisiertes Wort	Anzahl Nennungen
email	**8/8**
html	**7/8**
image	**6/8**
message	**5/8**
mime	2/8
body	2/8
inline	2/8
cid	1/8
verification	1/8

Tabelle C.62: Normalisierte Nutzerantworten und ihre Nennungshäufigkeit für die Gruppierung email 6

t \ s	source	get	data	**email**	mail	msg	**message**	**html**
1	1	1	1	2	1	1	2	2
2	2	0	2	2	1	0	1	2
3	2	2	2	2	2	1	1	2
4	1	0	0	2	1	0	2	2
5	1	2	2	2	1	0	1	2
6	1	2	1	2	1	1	1	1
7	2	2	2	2	2	2	2	2
8	1	1	1	2	2	1	2	2

Tabelle C.63: Bewertung der Schlagworte (s) für Gruppierung email 6 je Teilnehmer (t).

C.16 email 7

Teilnehmer	Beschreibung	Übereinstimmung
1	email, header, modify, address	2
2	email, cc, reply	3
3	email, recipients, reply, session, sender	1
4	email,from,to,bcc,cc,reply,add,set	4
5	email, add, recipient, sender	2
6	email, to, from, cc, bcc	5
7	e-mail, functionality, sender, receiver, reply	3
8	email, modification, reply, bcc, cc, receiver, sender	3

Tabelle C.64: Freitextantworten für Gruppierung email 7 und Bewertung der semantischen Übereinstimmung je Teilnehmer

Normalisiertes Wort	Anzahl Nennungen
email	**8/8**
reply	**5/8**
recipient/receiver	**4/8**
cc	**4/8**
sender	**4/8**
bcc	3/8
add	3/8
modify/modification	2/8
from	2/8
to	2/8
address	1/8
session	1/8
set	1/8
header	1/8
functionality	1/8

Tabelle C.65: Normalisierte Nutzerantworten und ihre Nennungshäufigkeit für die Gruppierung email 7

t \ s	**email**	set
1	2	0
2	2	0
3	2	1
4	2	1
5	2	2
6	2	2
7	2	2
8	2	1

Tabelle C.66: Bewertung der Schlagworte (s) für Gruppierung email 7 je Teilnehmer (t).

C.17 email 8

Teilnehmer	Beschreibung	Übereinstimmung
1	email, mime, message parsing	5
2	mime, email, parser, attachment	5
3	test, parser, mimetype, email, configuration	4
4	mime,message,parser,test,attachment,email,html,text	5
5	email, multipart, test, parse	4
6	email, message, parser, content	5
7	test, parser, message, configuration, e-mail	4
8	message, parser, test, attachement	4

Tabelle C.67: Freitextantworten für Gruppierung email 8 und Bewertung der semantischen Übereinstimmung je Teilnehmer

Normalisiertes Wort	Anzahl Nennungen
parser/parsing/parse	**8/8**
email	**7/8**
test	**5/8**
message	**5/8**
mime	**4/8**
attachment	3/8
configuation	2/8
html	1/8
text	1/8
multipart	1/8
content	1/8
type	1/8

Tabelle C.68: Normalisierte Nutzerantworten und ihre Nennungshäufigkeit für die Gruppierung email 8

t \ s	**mime**	**message**	**test**	**email**	**parser**
1	2	2	2	2	2
2	2	2	2	2	2
3	2	2	2	2	2
4	2	2	2	2	2
5	2	2	2	2	2
6	2	2	2	2	2
7	2	2	2	2	2
8	2	2	2	2	2

Tabelle C.69: Bewertung der Schlagworte (s) für Gruppierung email 8 je Teilnehmer (t).

C.18 email 9

Teilnehmer	Beschreibung	Übereinstimmung
1	email, test, character set, content type, smtp	5
2	Email, Test	5
3	test, email, headers, sending, content	4
4	email,test,send,from,host,name,charset	5
5	email, test, send, receive, content	4
6	email, test, server, content	5
7	test, e-mail, send, char set	3
8	email, test, connection	5

Tabelle C.70: Freitextantworten für Gruppierung email 9 und Bewertung der semantischen Übereinstimmung je Teilnehmer

Normalisiertes Wort	Anzahl Nennungen
email	**8/8**
test	**8/8**
content	**4/8**
send/sending	**4/8**
character set/char set/charset	**3/8**
type	1/8
smtp	1/8
headers	1/8
from	1/8
host	1/8
name	1/8
receive	1/8
server	1/8
connection	1/8

Tabelle C.71: Normalisierte Nutzerantworten und ihre Nennungshäufigkeit für die Gruppierung email 9

t \ s	**email**	**test**
1	2	2
2	2	2
3	2	2
4	2	2
5	2	2
6	2	2
7	2	2
8	2	2

Tabelle C.72: Bewertung der Schlagworte (s) für Gruppierung email 9 je Teilnehmer (t).

D Literaturanalysen

D.1 Latent Semantic Indexing

[DDF+90] Deerwester, Scott ; Dumais, Susan T. ; Furnas, George W. ; Landauer, Thomas K. ; Harshman, Richard: Indexing by Latent Semantic Analysis. In: *Journal of the American Society for Information Science* 41 (1990), Nr. 6, S. 391–407. `http://dx.doi.org/10.1002/(SICI)1097-4571(199009)41:6<391::AID-ASI1>3.0.CO;2-9`. – DOI 10.1002/(SICI)1097–4571(199009)41:6<391::AID–ASI1>3.0.CO;2–9. – ISSN 1097–4571

Inhalte

Was sind die zentralen Inhalte (Themen, interessante Aussagen, Botschaften, Fragestellungen), die in der Arbeit (d.h., in der analysierten Literatur) behandelt werden?

- I 1 Informationsgewinnung auf Basis von Konzepten, anstatt rein Wortbasiert.
- I 2 Nutzung der latenten semantischen Struktur der Dokumente.
- I 3 Verwendet Singulärwertzerlegung um die Term-Dokument Matrix auf einen semantischen Raum abzubilden.

Defizite

Welche Defizite bestehender Arbeiten und Lösungen werden als Motivation der eigenen Lösungen genannt?

- D 1 Dokumente werden mit anderen Worten gesucht, als mit denen sie indiziert wurden.
- D 2 Synonymie: Konzepte können mit vielen verschiedenen Worten beschrieben werden.
- D 3 Polysemie: Ein Wort kann, je nach Kontext, ganz unterschiedliche Bedeutungen haben.
- D 4 Faktorenanalyse ist sehr teuer ($N^2 * k^3$, wobei N die Anzahl der Terme plus Dokumente und k die Anzahl an Faktoren ist). Vorherige Arbeiten waren deshalb durch die vorhandene Rechenleistung eingeschränkt.

Prämissen

Welche Einschränkungen und Vorgaben werden hinsichtlich der eigenen Lösungen gemacht?

- P 1 Anzahl der Themen (Parameter k) muss im Voraus bekannt sein.

Lösungen

Was sind die eigenen Lösungen der Autoren?

- L 1 Faktorenanalyse liefert eine Zerlegung der Term-Dokument Beziehungen in linear unabhängige Faktoren. Entfernen von sehr kleinen Faktoren erlaubt eine Dimensionalitätsreduktion.
- L 2 Dokumente mit verschieden genutzten Termen können auf den selben Faktorenvektor abgebildet werden, dies löst das Synonymieproblem.
- L 3 Anfragen werden als Pseudo-Dokumente aufgefasst und mit anderen Dokumenten im semantischen Raum verglichen.

Nachweise

Welche Nachweise (Evidence) werden hinsichtlich der Tragfähigkeit der eigenen Lösungen geliefert?

- N 1 Verwendet Metriken sind Sensitivität und Präzision.
- N 2 Vergleich der Ergebnisse mit direkter Termübereinstimmungsmethode, und den SMART und Voorhees Systemen.
- N 3 Nutzt zwei verschiedene Datensätze zur Evaluation (MED und CISI)
- N 4 Mindestens gleich gut wie die anderen Verfahren, teilweise besser.

Offene Fragen

Welche Fragen sind noch ungelöst geblieben beziehungsweise stellen sich dem Leser?

F 1 Sind andere Ähnlichkeitsmaße als die Kosinusähnlichkeit denkbar?
F 2 Wie soll die Anzahl an reduzierten Dimensionen (k) gewählt werden?
F 3 Wie würden sich Wortvorverarbeitungen wie Lemmatisierung auf das Modell auswirken?

Notizen

LSI löst das Synonymieproblem, das Polysemieproblem allerdings nur teilweise. Dies liegt daran, dass jeder Term nur einem Punkt im semantischen Raum zugeordnet ist und somit Worte mit mehreren verschiedenen Bedeutungen als gewichteter Durchschnitt dieser Bedeutungen dargestellt werden.

D.2 Latent Dirichlet Allocation

[BNJ03] Blei, David M. ; Ng, Andrew Y. ; Jordan, Michael I.: *Latent Dirichlet Allocation.* März 2003

Inhalte

Was sind die zentralen Inhalte (Themen, interessante Aussagen, Botschaften, Fragestellungen), die in der Arbeit (d.h., in der analysierten Literatur) behandelt werden?

- I 1 Nutzt ein dreistufiges bayessches Modell.
- I 2 Dokumente werden als endliche Wahrscheinlichkeitsverteilung über eine Menge von Themen aufgefasst.
- I 3 Themen werden als Mischungen über eine Menge von Themenwahrscheinlichkeiten aufgefasst.

Defizite

Welche Defizite bestehender Arbeiten und Lösungen werden als Motivation der eigenen Lösungen genannt?

- D 1 Unklar, warum LSI angewandt werden soll, wenn bayessche Methoden direkt angewandt werden können um ein generatives Modell anzupassen.
- D 2 Bei pLSI (probabilistisches LSI) wächst die Anzahl der Modellparameter mit der Größe des Korpus.

Prämissen

Welche Einschränkungen und Vorgaben werden hinsichtlich der eigenen Lösungen gemacht?

- P 1 Wortbeutelannahme: Basiert auf der Annahme, dass die Reihenfolge der Worte in einem Dokument, sowie die Reihenfolge der Dokumente in einem Korpus vernachlässigbar sind.
- P 2 Die idealen Parameter des Modells müssen geschätzt und approximiert werden.

Lösungen

Was sind die eigenen Lösungen der Autoren?

- L 1 Dokumente können mehreren Themen zugeordnet werden.
- L 2 LDA kann in anderen Modelle eingebettet werden.
- L 3 Ausnutzen des Satz von De Finetti.

Nachweise

Welche Nachweise (Evidence) werden hinsichtlich der Tragfähigkeit der eigenen Lösungen geliefert?

- N 1 Vergleich mit Unigrammmodel, Mischung von Unigrammen Modell und pLSI.
- N 2 Perplexität als Maß.
- N 3 LDA leidet nicht an Overfitting.

Offene Fragen

Welche Fragen sind noch ungelöst geblieben beziehungsweise stellen sich dem Leser?

- F 1 Inwiefern ist LDA durch die Wortbeutelannahme eingeschränkt?
- F 2 Kann LDA effizient und präzise auf Quelltext angewandt werden?

Notizen

LDA kann nicht nur auf Fließtextkorpora angewendet werden. Weitere Anwendungsfälle beschäftigen sich beispielsweise mit Bildern.