Karlsruhe Institute of Technology

Anforderung-zu-Quelltextrückverfolgbarkeit mittels Wort- und Quelltexteinbettungen

Masterarbeit
von

Fei Chen

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

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

Bearbeitungszeit: 1.2.2020 – 7.9.2020

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, 7.9.2020

(**Fei Chen**)

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:	Masterarbeit
Titel:	Anforderung-zu-Quelltextrückverfolgbarkeit mittels Wort- und Quelltexteinbettungen
Datum:	7.9.2020
Name:	Fei Chen

Karlsruhe, 7.9.2020

(**Fei Chen**)

Kurzfassung

Rückverfolgbarkeitsinformationen helfen Entwickler beim Verständnis von Softwaresystemen und dienen als Grundlage für weitere Techniken wie der Abdeckungsanalyse. In dieser Arbeit wird untersucht, wie Einbettungen für die automatische Rückverfolgbarkeit zwischen Anforderungen und Quelltext eingesetzt werden können. Dazu werden verschiedene Möglichkeiten betrachtet, die Anforderungen und den Quelltext mit Einbettungen zu repräsentieren und anschließend aufeinander abzubilden, um Rückverfolgbarkeitsverbindungen zwischen ihnen zu erzeugen. Für eine Klasse existieren beispielsweise viele Optionen, welche Informationen bzw. welche Klassenelemente zur Berechnung einer Quelltexteinbettung berücksichtigt werden. Für die Abbildung werden zwischen den Einbettungen durch eine Metrik Ähnlichkeitswerte berechnet, mit deren Hilfe Aussagen über die Existenz einer Rückverfolgbarkeitsverbindung zwischen ihren repräsentierten Artefakten getroffen werden können.
In der Evaluation wurden die verschiedenen Möglichkeiten für die Einbettung und Abbildung untereinander und mit anderen Arbeiten verglichen. Bezüglich des F1-Wertes erzeugen Quelltexteinbettungen mit Klassennamen, Methodensignaturen und -kommentaren sowie Abbildungsverfahren, die die Word Mover's Distance als Ähnlichkeitsmetrik nutzen, die besten projektübergreifenden Ergebnisse. Das beste Verfahren erreicht auf dem Projekt „LibEST“, welches aus 14 Quelltext- und 52 Anforderungsartefakten besteht, einen F1-Wert von 60,1%. Die beste projektübergreifende Konfiguration erzielt einen durchschnittlichen F1-Wert von 39%.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Softwareprojekte werden seit Jahren größer und komplexer. Für die Entwickler wird die Handhabung der Komplexität immer schwieriger. Anforderung-zu-Quelltextrückverfolgbarkeit ist hierbei eines der Mittel, das dabei helfen kann, den Überblick über das Projekt beizubehalten. Hierbei werden Rückverfolgbarkeitsverbindungen zwischen Artefakten hergestellt, wie etwa zwischen Anforderungen und den Quelltextteilen, die die Anforderungen implementieren. Durch das Vorhandensein solcher Verbindungen gewinnen Entwickler mehr Erkenntnis über die Zusammenhänge des Projekts, wodurch auch das Verständnis steigt und dadurch Kosten reduziert werden können [DP98].
Außerdem ermöglichen Rückverfolgbarkeitsinformationen die Anwendung anderer Verfahren wie zum Beispiel die Auswirkungsanalyse: Wird ein Artefakt geändert, können durch die Rückverfolgbarkeitsverbindungen alle anderen Artefakte identifiziert werden, die von der Änderung potentiell ebenfalls betroffen sind. Auch können Rückverfolgbarkeitsverbindungen genutzt werden, um eine Abdeckungsanalyse durchzuführen, das heißt um zu überprüfen, ob alle Anforderungen (vollständig) umgesetzt wurden. Dies ist besonders bei sicherheitskritischen Systemen wichtig.

Die manuelle Durchführung der Rückverfolgbarkeit durch Entwickler ist sehr aufwändig. Existierende Werkzeuge für die automatische Rückverfolgbarkeit besitzen nicht den erforderlichen Automatisierungsgrad für eine produktive Anwendung. Daher wird eine vollautomatische Lösung benötigt.
Die automatische Rückverfolgbarkeit von Quelltext und Anforderungen bringt jedoch auch Herausforderungen mit sich. Quelltext ist im Gegensatz zu Anforderungen nicht vollständig in natürlicher Sprache formuliert, was die Rückverfolgbarkeit erschwert. Außerdem befinden sich Anforderungen und Quelltext inhaltlich nicht unbedingt auf gleichen Abstraktionsebenen. Dieser Umstand führt dazu, dass korrespondierende Anforderungen und Quelltextteile das System mit unterschiedlichem Detailgrad und verschiedener Wortwahl beschreiben können. Existierende Rückverfolgbarkeitsverfahren, die zum Beispiel auf das Vorkommen des gleichen Wortes in den zu verbindenden Artefakten basieren, würden hierdurch viele Rückverfolgbarkeitsverbindungen nicht identifizieren können.
Eine Verbesserung verspricht der Einsatz von Worteinbettungen. Eine Einbettung kann die Semantik eines Wortes als einen Vektor repräsentieren. Die Vektoren besitzen die Eigenschaft, dass die semantische Ähnlichkeit durch die Distanz im Vektorraum kodiert wird. Vektoren von Wörtern, die sich semantisch ähnlich sind, liegen nah beieinander. Es wird von der konkreten Wortform abstrahiert und die Überwindung des Abstraktionsunterschieds zwischen den Artefakten wird erleichtert.

1.1 Ziele

In dieser Arbeit soll eine Rückverfolgbarkeit zwischen Anforderungen und Quelltext unter Einsatz von Einbettungen entworfen und durchgeführt werden. Dazu gehört auch der Entwurf von Anforderungs- und Quelltexteinbettungen, die ganze Anforderungen oder Quelltextartefakte repräsentieren. Schließlich soll ein Abbildungsverfahren zwischen diesen Einbettungen entwickelt werden, um Rückverfolgbarkeitsverbindungen zu erzeugen.

1.2 Aufbau der Arbeit

In Kapitel 2 werden die Grundlagen erklärt. Dazu gehören zum Beispiel existierende Einbettungsverfahren. Anschließend werden in Kapitel 3 verwandte Arbeiten vorgestellt. Kapitel 4 befasst sich mit der Analyse der Möglichkeiten, Einbettungs- und Abbildungsverfahren zu gestalten sowie mit dem Entwurf dieser Verfahren. Die Beschreibung in Kapitel 5 befasst sich mit der Implementierung des Entwurfs und in Kapitel 6 werden die Verfahren evaluiert und verglichen. Schließlich fasst Kapitel 7 die Arbeit zusammen und gibt einen Ausblick auf mögliche zukünftige Arbeiten.

2 Grundlagen

In diesem Kapitel werden benötigte Grundlagen erläutert. Abschnitt 2.1 erklärt, was unter Anforderungsrückverfolgbarkeit verstanden wird. Informationsrückgewinnung ist eine Möglichkeit zur Anforderungsrückverfolgbarkeit und wird in Abschnitt 2.2 definiert. Natürliche Sprache und Quelltext enthalten nicht ausschließlich Informationen, die für die Rückverfolgbarkeit nützlich sind. Eingabedaten müssen je nach Verfahren auch in bestimmte Formaten vorliegen (zum Beispiel ausschließlich Kleinschreibung). Aus diesen Gründen ist häufig eine Vorverarbeitung sinnvoll, bevor die eigentlichen Verfahren und Algorithmen darauf angewendet werden. Abschnitt 2.3 erklärt einige dieser Vorverarbeitungsschritte. Die meisten Einbettungsverfahren werden durch Techniken des maschinellen Lernens erzeugt. Abschnitt 2.4 definiert zunächst diesen Begriff. (Künstliche) neuronale Netze sind ein zentraler Baustein im maschinellen Lernen und sind ein wichtiges Werkzeug, mit dem man Rückverfolgbarkeitsverbindungen aufstellen kann. Abschnitt 2.5 gibt dazu einen Überblick über die verschiedenen Arten von neuronalen Netzen. In Abschnitt 2.6 und Abschnitt 2.7 sind verschiedene Methoden zur numerischen Darstellung von Anforderungen und Quelltext aufgezählt und beschrieben. Schließlich werden in Abschnitt 2.8 diverse Metriken erläutert, die zur Bemessung der Ähnlichkeit zwischen zwei Einbettungen verwendet werden können.

2.1 Anforderungsrückverfolgbarkeit

Rückverfolgbarkeit (engl. traceability) stellt logische Verbindungen zwischen Artefakten aus dem Softwareentwicklungsprozess her [MG12] oder wird als der Grad, mit dem man Beziehungen zwischen Artefakten, insbesondere Vorgänger-Nachfolger- und Auftraggeber-Auftragnehmer-Beziehungen, aufstellen kann [NdS13] bezeichnet. Anforderungsrückverfolgbarkeit (engl. requirements traceability) ist die Rückverfolgbarkeit zwischen Anforderungen und anderen Artefakten; Anforderung-zu-Quelltext-Rückverfolgbarkeit ist dementsprechend die Rückverfolgbarkeit zwischen den Artefakten „Anforderungen“ und „Quelltext“. Das Ziel ist das Extrahieren von Rückverfolgbarkeitsverbindungen (engl. trace links) zwischen den Artefakten. Bei der Anforderung-zu-Quelltext-Rückverfolgbarkeit bestehen folglich Rückverfolgbarkeitsverbindungen zwischen Anforderungen und den Quelltextteilen, welche die Anforderung implementieren. Häufig werden Rückverfolgbarkeitsverbindungen in einer zweidimensionalen Rückverfolgbarkeitsmatrix zusammengefasst, in der dargestellt ist, welche Anforderung zu welchem Quelltextartefakt korrespondiert.

2.2 Informationsrückgewinnung

Die Extraktion von Rückverfolgbarkeitsverbindungen wurde in der Vergangenheit oft als ein Problem der Informationsrückgewinnung (engl. information retrieval, kurz IR) formuliert [BRA14]. Nach Manning et al. [MRS08] ist die Informationsrückgewinnung ist wie in Definition 2.1 definiert :

Definition 2.1: Informationsrückgewinnung

Informationsrückgewinnung ist das Auffinden von Material (meistens Dokumente), die unstrukturiert vorliegen (meistens in Textform), um ein Informationsbedürfnis zu einer großen Sammlung von Daten (die meistens auf Computern gespeichert sind) zu befriedigen.

Das Informationsbedürfnis liegt in Form einer Anfrage vor und ist oft komplexer als eine einfache Stichwortsuche. Häufig werden daher Techniken verwendet, die alle vorliegenden Dokumente und die Anfrage auf Vektoren abbilden, die danach mit einer Ähnlichkeitsmetrik verglichen werden (siehe dazu Abschnitt 2.6.3). Es gibt viele Variationen, wie die Abbildung auf die Vektoren verläuft und wie die Ähnlichkeit gemessen wird, in Abschnitt 3.1 werden dazu einige Methoden aus verwandten Arbeiten vorgestellt.

2.3 Vorverarbeitung von natürlicher Sprache und Quelltext

Natürliche Sprache kann unpräzise und redundant sein. Außerdem setzen manche Einbettungsverfahren bestimmte Formate für die Eingabedaten voraus. Die im Folgenden vorgestellten Vorverarbeitungstechniken sollen dabei helfen, die natürliche Sprache effektiver für den Computer zu erfassen und sie in die richtigen Eingabeformate zu transformieren.

2.3.1 Stoppwortentfernung

Stoppwörter sind Wörter, die oft vorkommen, aber nur wenig relevante Informationen besitzen [JM09]. In Beispiel 2.1 sind einige Stoppwörter aufgeführt. Die Entfernung dieser Wörter verringert das Rauschen für die weitere Verarbeitung.

Beispiel 2.1: Stoppwörter

der, ein, eure, wie

Stoppwörter können anhand von vorgefertigten Listen identifiziert werden oder man sortiert alle vorkommende Wörter des Textes nach ihrer Häufigkeit und entfernt die häufigsten n Wörter.

2.3.2 Lemmatisierung und Stemming

Bei der Lemmatisierung wird die Grundform eines Wortes bestimmt [JM09]. Beispiel 2.2 zeigt einige Beispiele dazu.

Beispiel 2.2: Grundformen bestimmen

Tische → Tisch
gesungen → singen
bist → sein

Für viele Anwendungen ist es nicht relevant, welche Konjugation/welcher Numerus/ etc. das Wort besitzt. Nach der Lemmatisierung werden die verschiedenen Formen eines Wortes nicht als verschiedene Wörter aufgefasst, wodurch sich das Rauschen reduziert. Lemmatisierung bestimmt die Grundform durch Nachschlagen in Wörterbüchern oder durch maschinelles Lernen. Stemming hat das gleiche Ziel wie Lemmatisierung, jedoch wird hier die Grundform an Hand von Regeln durch Wegschneiden des Suffix' bestimmt. Stemming ist im Vergleich zu Lemmatisierung schneller, aber dafür ungenauer, wie Beispiel 2.3 zeigt.

Beispiel 2.3: Lemmatisierung vs. Stemming

Lemmatisierung

Bibliotheken → Bibliothek
Museen → Museum

Stemming

Bibliotheken → Bibliothek (Korrekte Entfernung des „en"-Suffix)
Museen → Muse (Fehlerhafte Entfernung des „en"-Suffix)

2.3.3 Auflösung der Binnenmajuskelschreibweise

Quelltextbezeichner werden oft in der Binnenmajuskelschreibweise verfasst. Die Bezeichner müssen in einem Vorverarbeitungsschritt aufgetrennt werden, damit die einzelnen Wörter weiterverarbeitet werden können.

2.3.4 Transformation in Kleinbuchstaben

Im Englischen werden Wörter am Satzanfang und Eigennamen groß geschrieben. Für die Wörter am Satzanfang ist es sinnvoll, diese in Kleinbuchstaben zu transformieren, da sonst die groß und klein geschriebene Varianten in den späteren Weiterverarbeitungsschritten als verschiedene Wörter aufgefasst werden, obwohl ihre Bedeutung identisch ist. Dies ist meist nicht gewollt. Bei Quelltextbezeichnern ist diese Vorverarbeitung auch sinnvoll, da Wörter aufgrund von Benamungskonventionen groß geschrieben werden können.

2.3.5 Wortlängenfilter

Es kann sinnvoll sein, Wörter unter einer Mindestlänge an Buchstaben herauszufiltern. Heuristisch betrachtet können Wörter mit sehr wenigen Buchstaben als Wörter mit wenig Semantik angesehen werden, da sie oft Präpositionen oder Artikel sind. Daher kann ein Wortlängenfilter ergänzend zur Stoppwortentfernung eingesetzt werden. Weiterhin können kurze Wörter auch auch Abkürzungen darstellen. Falls Abkürzungen nicht aufgelöst werden, kann es das Rauschen verringern, wenn die Abkürzungen entfernen werden, da viele weiterverarbeitende Verfahren nicht mit Abkürzungen umgehen können. Im Falle von Quelltext werden zum Beispiel oft Variablennamen mit einem oder zwei Buchstaben verwendet. Auch solche Bezeichner tragen keine Semantik und würden nur Rauschen verursachen.

Die Mindestlänge für den Wortlängenfilter ist ausschlaggebend für die Nützlichkeit des Wortlängenfilters. Ist er zu groß gewählt, können unbeabsichtigt semantisch nützliche Wörter herausgefiltert werden.

2.3.6 Nichtbuchstabenfilter

Texte können Zeichen beinhalten, die keine Buchstaben sind. Dazu gehören zum Beispiel Zahlen, Klammern oder Satzzeichen. Je nachdem, ob das weiterverarbeitende Verfahren mit den Zeichen umgehen kann, ist eine Entfernung der Zeichen eventuell sinnvoll, um nicht verarbeitbare Zeichen zu entfernen und damit das Rauschen zu verringern.

2.3.7 N-Gramme

Ein N-Gramm ist eine Sequenz von N aufeinander folgenden Wörtern [JM09]. 1-Gramme heißen auch Unigramme, 2-Gramme nennt man Bigramme und 3-Gramme können auch als Trigramme bezeichnet werden. Manche Konzepte bestehen aus mehreren Wörtern, wie zum Beispiel „Vereinigtes Königreich“. Hier macht es Sinn, „Vereinigtes“ und „Königreich“ nicht als zwei unabhängige Wörter zu betrachten, sondern in Form eines Bigramms als eine Einheit zu behandeln. N-Grammen werden häufig Wahrscheinlichkeiten zugewiesen, da manche Wortkombinationen statistisch öfter auftreten als andere. Dies bildet die Basis für viele Anwendungen aus der natürlichen Sprachverarbeitung, wie zum Beispiel bei der Vorhersage des nächsten Wortes für eine Autovervollständigung.

2.4 Maschinelles Lernen

Beim maschinellen Lernen geht es allgemein darum, dass Programme automatisiert aus gegebenen Eingabedaten lernen [SB14]. Die Eingabedaten dienen als Training, aus denen das Programm „Wissen“ gewinnt. Mit Hilfe des Wissens kann das Programm wiederum bestimmte Aufgaben lösen. Maschinelles Lernen kommt vor allem dann zum Einsatz, wenn Wissen aus großen Datenmengen gewonnen werden soll oder eine explizite Programmierung zu schwer ist. Es ist zum Beispiel zu komplex, ein zufriedenstellendes Programm zur Spracherkennung ohne maschinelles Lernen zu programmieren, da Sprache viel zu variabel ist. Durch maschinelles Lernen werden dagegen Muster und Regelmäßigkeiten aus den Trainingsdaten automatisch erkannt. Außerdem sind Verfahren aus dem maschinellen Lernen adaptiv, das heißt sie können aus neuen Eingabedaten neues Wissen erlangen.

Die Techniken aus dem maschinellen Lernen können unter anderem anhand ihres Lernverfahrens kategorisiert werden: Überwachte Lernverfahren erfordern Trainingsdaten, die eine dazugehörige Musterlösung (gemäß der jeweiligen zu lösenden Aufgabe) besitzen. Unüberwachte Lernverfahren lernen dagegen mit Trainingsdaten ohne Musterlösung.

2.5 Künstliche Neuronale Netze

Künstliche neuronale Netze sind eine Klasse von Techniken des maschinellen Lernens, die parametrisierte mathematische Funktionen lernen können [Gol17]. Eingabedaten müssen numerisch repräsentiert werden, bevor sie in das Netz eingegeben werden. Da das neuronale Netz wie die Implementierung einer bestimmten Funktion wirken soll, muss dementsprechend die Ausgabe des Netzes wie die Ausgabe dieser Funktion aussehen. Hierzu wird das neuronale Netz durch Lernverfahren iterativ trainiert, dabei nähert sich die aktuelle Ausgabe in jeder Iteration der Sollausgabe an.

Ein Netz besteht dabei aus einer Menge von verbundenen künstlichen Neuronen. Abbildung 2.1 zeigt ein simples neuronales Netz (auch „Feedforward“-Netz genannt). Es nimmt zwei Eingabewerte, die an alle Neuronen in der versteckten Schicht in der Mitte weitergegeben werden. Diese verarbeiten die Werte und geben ihr Ergebnis an alle Neuronen in der nächsten Schicht weiter, wobei es mehrere versteckte Schichten geben kann. Die letzte Schicht nennt man Ausgabeschicht, welches das endgültige Ergebnis berechnet und ausgibt. In Abbildung 2.2 ist ein einzelnes Neuron dargestellt. Ein Neuron erhält die Eingaben x_i

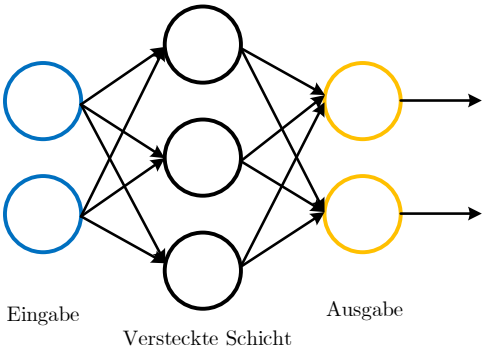


Abbildung 2.1: Ein einfaches künstliches neuronales Netz aus drei Schichten.

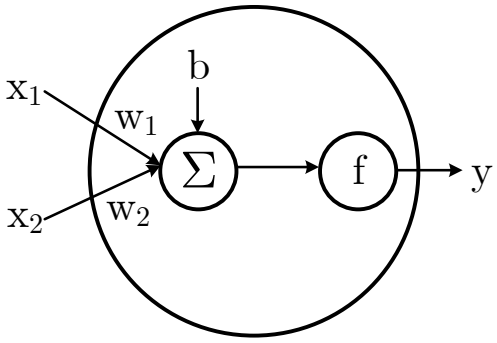


Abbildung 2.2: Aufbau eines Neurons.

aus der vorherigen Schicht, diese werden mit neuronspezifischen Gewichten w_i multipliziert und aufsummiert. Das Ergebnis davon wird in eine Aktivierungsfunktion f eingesetzt. Die Aktivierungsfunktion bildet die Eingabe auf einen bestimmten Wertebereich ab; gängige Bereiche sind $[0, 1]$ oder $[-1, 1]$. b wird Bias genannt und dient dazu, die Kurve der Aktivierungsfunktion zu verschieben. Das Ergebnis der Aktivierungsfunktion wird schließlich an alle Neuronen der nächsten Schicht weitergegeben. Die allgemeine Berechnung innerhalb eines Neurons mit n Eingabewerten ist in Gleichung 2.1 zusammengefasst.

$$y = f(b + w_1 \cdot x_1 + ... + w_n \cdot x_n) \tag{2.1}$$

Anzumerken gilt, dass das Neuron bis zur Summation der gewichteten Eingaben rein linear operiert. Erst durch die Aktivierungsfunktion f wird eine Nichtlinearität hinzugefügt, da für f zum Beispiel Sigmoidfunktionen oder der Tangens hyperbolicus verwendet wird.

Alle Neuronen in der versteckten Schicht und in der Ausgabe sind nach diesem Prinzip aufgebaut (Die Eingabeschicht leitet die Eingabe nur weiter), daher können die Berechnungen innerhalb einer Schicht wie folgt zusammengefasst werden: Die Gewichte w_i eines Neurons wird als einen Vektor aufgefasst und alle Gewichtsvektoren einer Schicht können zu einer Matrix W verschmolzen werden (Jede Spalte entspricht einem Gewichtsvektor). Wenn die Eingabewerte x_i und die Biaswerte b_i ebenfalls zu Vektoren zusammenfasst werden, können die Berechnungen innerhalb einer Schicht, wie in Gleichung 2.2 zu sehen, als eine Matrixmultiplikation mit einer Vektoraddition dargestellt werden.

$$y = f(Wx + b) \tag{2.2}$$

Da die Eingabe x einer Netzschicht die Ausgabe der vorigen Schicht entspricht, kann die Berechnung einer tieferen Schicht als verschachtelte Funktion ausgedrückt werden. Glei-

chung 2.3 zeigt dies beispielhaft an der Berechnung einer dritten Schicht (Eingabeschicht nicht mitgezählt, da diese nur weiterleitet). Die oberen Indizes geben an, aus welcher Schicht der entsprechende Term stammt.

$$y^3 = f^3(W^3 \cdot f^2(W^2 \cdot f^1(W^1 x + b^1) + b^2) + b^3)) \tag{2.3}$$

Künstliche neuronale Netze können trainiert werden; das heißt, dass die Gewichte der Neuronen durch ein Lernverfahren iterativ angepasst werden, sodass das Netz nach dem Training für bestimmte Eingabewerte bestimmte Ausgabewerte zurückgibt. Da beim neuronalen Netz ein überwachtes Training eingesetzt werden muss, müssen zu gegebenen Eingabewerten auch passende Ausgabewerte als Musterlösung vorhanden sein. Es existieren verschiedene Lernverfahren, jedoch ist das Grundprinzip gleich: Die Eingabewerte werden in das untrainierte Netz eingespeist und das Netz gibt eine zunächst falsche Ausgabe zurück. Eine Fehlerfunktion $L(y, \hat{y})$ (zum Beispiel die mittlere quadratische Abweichung) berechnet die Abweichung zwischen der aktuellen, falschen Ausgabe y und der Musterlösung $\hat{y}$. Durch Rückpropagierung (engl. backpropagation) und Gradientenabstieg werden alle Gewichte im Netz angepasst. Dafür wird für jedes Gewicht seinen Anteil am Fehler L berechnet; dieser Anteil entspricht der partiellen Ableitung von L nach dem jeweiligen Gewicht. y in $L(y, \hat{y})$ kann man durch eine verschachtelte Schreibweise ähnlich wie in Gleichung 2.3 ersetzen; dadurch hängt L nun von den Gewichten W ab und eine partielle Ableitung nach den Gewichten ist möglich. Die Ableitung der verschachtelten Gewichte (also Gewichte der tieferen Schichten) kann mit Hilfe der Kettenregel zu einer Multiplikation zwischen der Ableitung der inneren mit der äußeren Funktion umgeformt werden. Mit dem Wert ihrer partiellen Ableitungen werden die Gewichte der Neuronen angepasst, sodass die Trainingsdaten beim nächsten Durchlauf des Netzes näher an der Musterlösung liegen. Dies wird iterativ wiederholt, bis der Fehler zwischen Musterlösung und aktueller Ausgabe unter einen Schwellwert fällt.

2.5.1 Rekurrente neuronale Netze

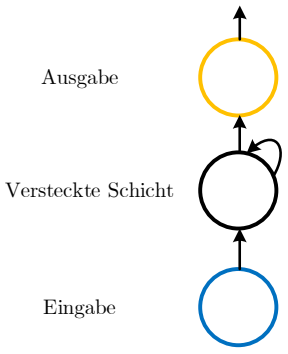


Abbildung 2.3: Ein rekurrentes neuronales Netz mit drei Schichten aus jeweils einem Neuron.

Rekurrente neuronale Netze (RNN) [Elm90] erweitern Feedforward-Netze um einen zeitlichen Aspekt, oft als „Gedächtnis“ referenziert; davon verspricht man sich eine bessere Modellierung, da zum Beispiel Sprache eine zeitliche Abfolge von Wörtern ist. Bei Feedforward-Netzen hängen alle Berechnungen nur von der aktuellen Eingabe ab. Rekurrente Neuronen können dagegen zusätzlich auf die berechneten Werte der vorherigen Iteration zugreifen und in die Berechnung mit der aktuellen Eingabe miteinbeziehen. Wie in Abbildung 2.3 zu sehen, wird das mit einer Schleife an den Neuronen veranschaulicht.

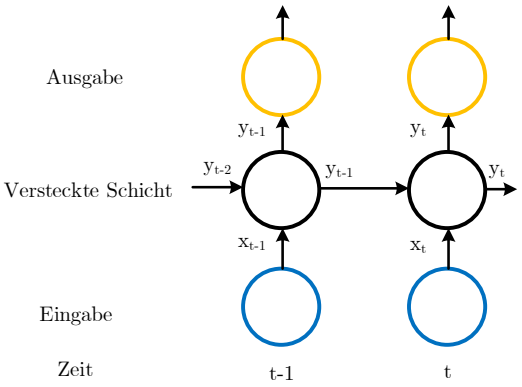


Abbildung 2.4: Hier wurde das versteckte Neuron aus Abbildung 2.3 ausgerollt, um die Informationsflüsse zwischen Zeitschritten t_i zu verdeutlichen.

Die Berechnungen innerhalb eines Neurons aus Gleichung 2.2 verändert sich mit dem Zeitindex t zu Gleichung 2.4.

$$y_t = f(Wx_t + b + Uy_{t-1}) \tag{2.4}$$

x_t ist die aktuelle Eingabe, die wie bereits bekannt mit den aktuellen Neurongewichten W multipliziert wird. Zusätzlich werden bei RNNs die berechneten Werte y_{t-1} aus der vorigen Iteration, gewichtet mit einer Matrix U, dazuaddiert. U wird in der Lernphase genauso wie W trainiert. Die Summe wird schließlich einer Aktivierungsfunktion f übergeben, um die aktuellen y_t-Werte zu berechnen.

In Abbildung 2.4 wurde die Schleife aus Abbildung 2.3 ausgerollt und eine Zeitachse hinzugefügt. Daran ist gut zu sehen, wie die Informationen aus der vergangenen Eingabe x_{t-2} indirekt über y_{t-1} zur Berechnung in y_t miteinfließen.

2.5.2 Neuronale Netze mit langem Kurzzeitgedächtnis

Neuronale Netze mit langem Kurzzeitgedächtnis (engl. long short-term memory, kurz LSTM)) [HS97] sind eine Verbesserung von RNNs. RNNs leiden bei langen Eingabefolgen unter dem Problem der verschwindenden Gradienten (engl. vanishing gradient problem): Bei RNNs wird die vergangene Berechnung in die aktuelle Berechnung miteinbezogen und danach eine Anpassungsänderung der Gewichte mit Rückpropagierung ermittelt. Die verschachtelten vergangenen Berechnungen werden bei der Rückpropagierung mit der Kettenregel zu einer Multiplikationskette von Gradienten umgewandelt. Ist die Eingabe also sehr lang, gibt es eine sehr tiefe Verschachtelung der vergangene Berechnungen - wenn die Gradienten kleiner als Eins sind, strebt die Anpassungsänderung wegen der langen Kette von Multiplikationen gegen Null; das heißt das Netz „lernt fasst nichts“.

LSTM-Netze wirken diesem Problem entgegen, indem sie bewerten, welche Informationen aus den vergangenen Berechnungen noch relevant für die aktuelle Berechnung sind und welche vergessen werden können. Dazu wird jeweils für die nächste Iteration $t+1$ nicht nur die aktuelle Ausgabe y_t weitergegeben, sondern zusätzlich ein Kontextvektor c_t. Der Kontextvektor entspricht einem Gedächtnis und durch dessen Modifikation wird entschieden, welche alten Informationen vergessen oder beibehalten werden sollen.

Abbildung 2.5 zeigt den Aufbau eines Neurons in einem LSTM-Netz. Das Neuron besitzt drei „Tore“ (Gates), die unterschiedliche Berechnungen mit den Eingaben durchführen.

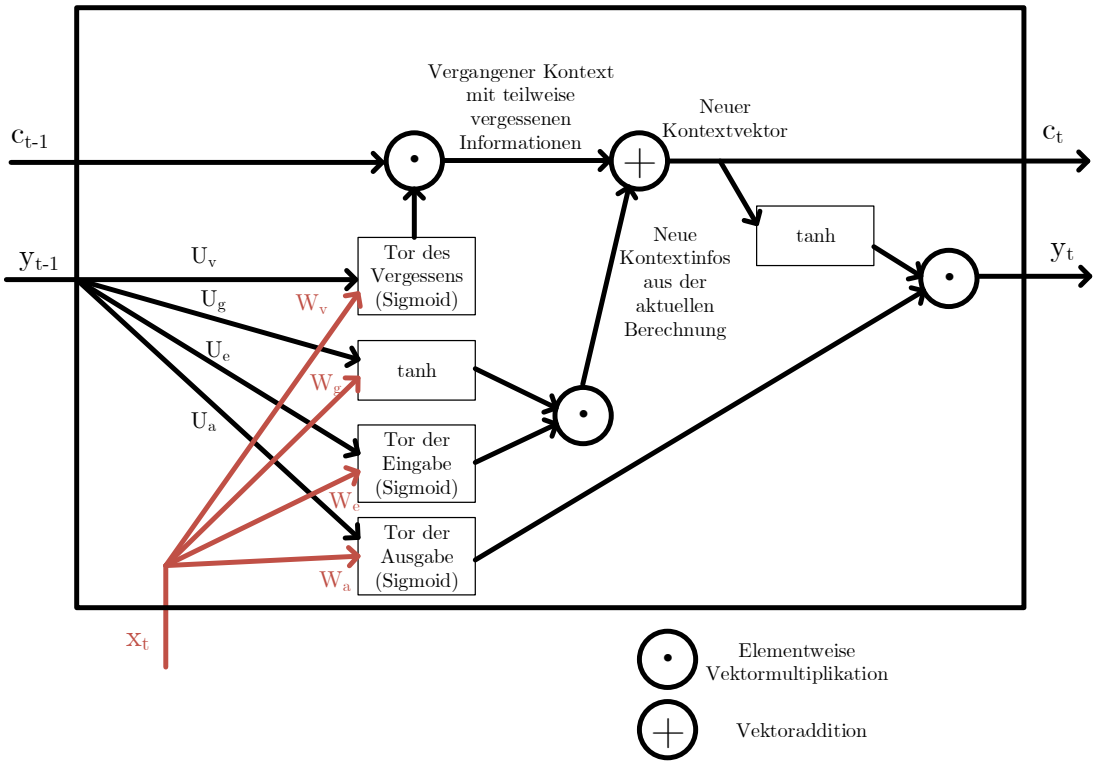


Abbildung 2.5: Informationsflüsse eines LSTM-Neurons.

Das Berechnungsschema ist jedoch gleich und ist in Gleichung 2.5 zu sehen. Der i-Index bezieht sich auf die Tore aus Abbildung 2.5.

$$sigmoid(W_i x_t + U_i y_{t-1}), i \in \{v, e, a\} \tag{2.5}$$

Es wird also eine Linearkombination aus der vergangenen Ausgabe y_{t-1} und der aktuellen Eingabe x_t mit torspezifischen Gewichten gebildet. Die Summe wird danach jeweils einer Sigmoidaktivierungsfunktion als Eingabe übergeben. Ein Sigmoid hat die Eigenschaft, dass die Eingabe auf Werte zwischen Null und Eins abgebildet werden, wobei die meisten Werte entweder sehr nah an Eins oder an Null sind. Dadurch sind die Ergebnisvektoren der Tore sehr ähnlich wie binäre Masken. Die Maske aus dem Tor des Vergessens wählt durch die elementweise Multiplikation mit dem vergangenen Kontextvektor aus, welche Elemente davon vergessen (mit annähernd Null multipliziert) und welche beibehalten (mit annähernd Eins multipliziert) werden sollen. Die linke tanh-Box in Abbildung 2.5 stellt die bekannte Verarbeitung der Eingabedaten wie in Feedforward-Neuronen mit tanh als Aktivierungsfunktion dar. Die Maske aus dem Tor der Eingabe wählt daraus diejenigen Informationen aus, die zum Kontextvektor (durch Vektoraddition) hinzugefügt werden sollen. Die Maske aus dem Tor der Ausgabe wählt schließlich aus dem neuen Kontextvektor (nachdem dieser durch die rechte tanh-Box auf einem Wertebereich von $[-1, 1]$ abgebildet worden ist) die Ausgabe y_t aus.

2.5.3 Kodierer-Dekodierer-Netze

Die bisher vorgestellten Netze nehmen zu einem Zeitpunkt eine Eingabe und berechnen daraus eine Ausgabe. In Kodierer-Dekodierer-Netzen (engl. encoder-decoder-network) [JM09] kann eine Sequenz von Eingaben gleichzeitig aufgenommen werden und daraus wird eine Sequenz von Ausgaben berechnet.

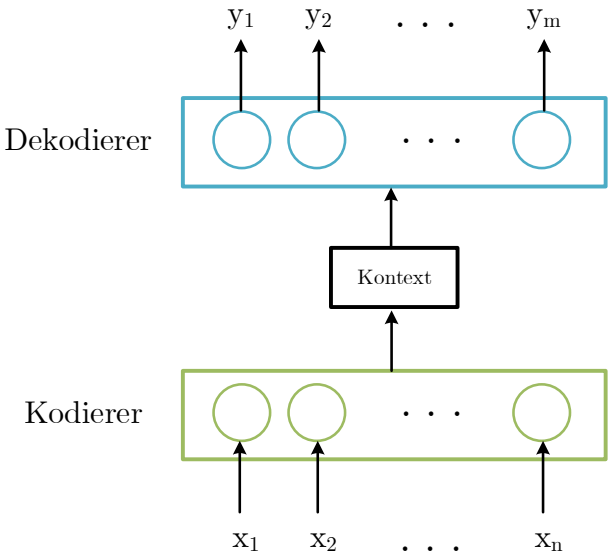


Abbildung 2.6: Aufbau eines Kodierer-Dekodierer-Netzes

In Abbildung 2.6 wird der schematische Aufbau aufgezeigt. Kodierer-Dekodierer-Netze sind zunächst abstraktes Netzmodelle; in der Implementierung kann eine Kodiererschicht zum Beispiel durch ein RNN oder eine LSTM-Schicht realisiert werden. Die Kodierer berechnen eine Zwischenrepräsentation (den Kontext), welcher an die Dekodierschicht weitergegeben wird. Der Dekodierer kann dabei nur auf den fertig berechneten Kontext zugreifen; etwaige Zwischenergebnisse zwischen den Kodiererneuronen sind für den Dekodierer nicht sichtbar. Ist der Kodierer zum Beispiel ein RNN, entspräche dieser Kontext (nur) der Ausgabe des RNNs im letzten Zeitschritt der Eingabe, die Zwischenergebnisse der vorherigen Zeitschritte sind nicht für den Dekodierer zugreifbar. Diese Separierung ermöglicht ein flexibles Austauschen von Kodierern und Dekodierern. Die Anzahl der Ausgabewerte muss außerdem nicht der Anzahl der Eingabewerte entsprechen.

2.5.4 Transformer-Netze

Transformer-Netze [VSP+17] sind Kodierer-Dekodierer-Netze mit einem mehrköpfigem Selbstbeachtungsmechanismus (engl. multi-headed self-attention mechanism). Die Verbesserung liegt darin, dass der Selbstbeachtungsmechanismus für die Eingabesequenz $x_1, ..x_n$ bewertet, „wie viel Beachtung“ ein x_i im Vergleich zu den anderen Eingaben erhalten soll. Je größer die Beachtung, desto mehr Einfluss hat das x_i auf die weitere Verarbeitung. Für die Berechnung der Beachtung werden zunächst für jede Eingabe x_i einen Anfrage-, Schlüssel- und Wertvektor (engl. query/key/value vector) berechnet, in dem sie, wie in Gleichung 2.6 zu sehen, mit Gewichtsmatrizen A, S, W multipliziert werden. Anfrage- und Schlüsselvektor haben dabei die gleiche Dimension d.

$$\begin{aligned} Anfragevektor : a_i &= A \cdot x_i \\ Schlüsselvektor : s_i &= S \cdot x_i \\ Wertvektor : w_i &= W \cdot x_i \end{aligned} \tag{2.6}$$

Anschließend werden „aus der Sicht“ von x_i alle Eingaben $x_j; j = 1, .., n$ (also inklusive x_i selber) dahingehend bewertet, wie viel Beachtung jede Eingabe erhalten soll. Dazu werden werden als Erstes Beachtungsgewichte α_i wie in Gleichung 2.7 berechnet.

$$\alpha_j^i = softmax\left(\frac{q_i s_j}{\sqrt{d}}\right); j = 1, .., n \tag{2.7}$$

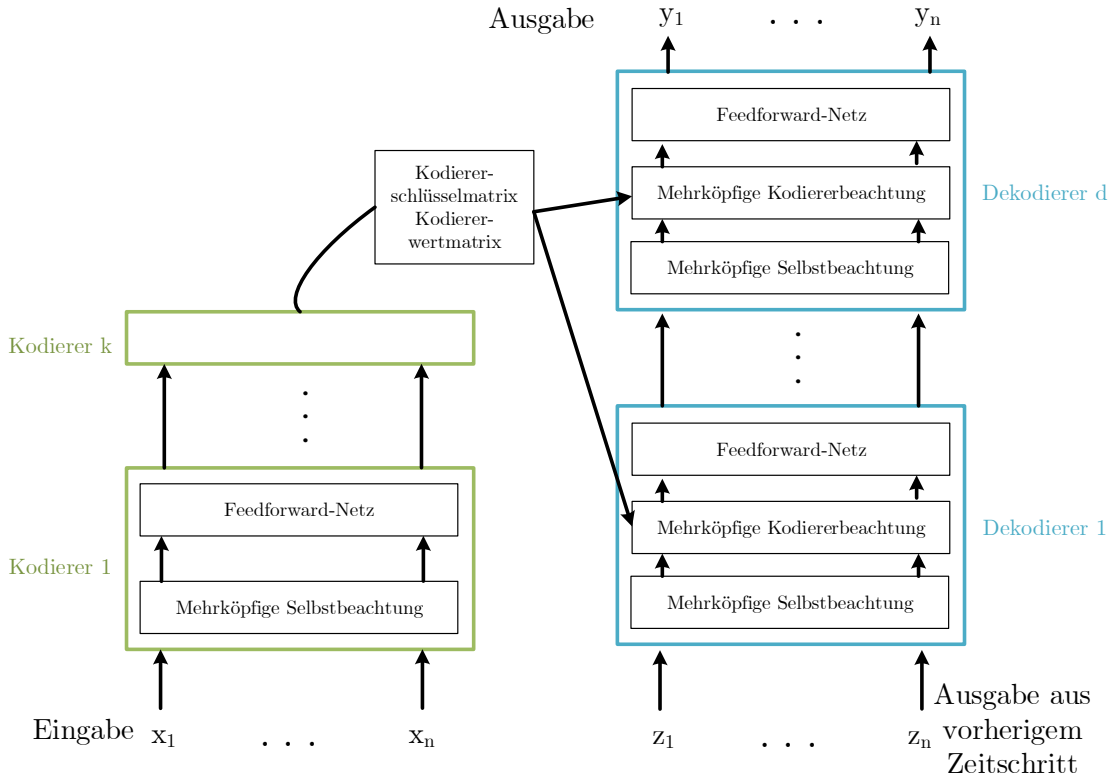


Abbildung 2.7: Aufbau eines Transformer-Netzes

Die Beachtungsgewichte sind aus der Sicht von Eingabe x_i, also wird hierfür konstant sein eigenes q_i als Anfragevektor verwendet. Dieser wird jeweils mit allen anderen Schlüsselgewichten s_j (inklusive s_i) skalarmultipliziert und anschließend durch einen Skalierungsfaktor $\sqrt{d}$ geteilt, um die Beachtungsgewichte α_j^i zu berechnen. Die Softmax-Funktion sorgt dafür, dass die Gewichte zwischen Null und Eins liegen und die Summe aller Gewichte Eins beträgt. Die Beachtung (welches ein Vektor ist) aus der Sicht von x_i wird nun durch die gewichtete Summe der Wertvektoren w_j aller Eingaben mit den korrespondierenden Beachtungsgewichten berechnet (siehe Gleichung 2.8).

$$Beachtung_i = \sum_{j=1}^{n} \alpha_j^i w_j \tag{2.8}$$

Für alle anderen Eingaben werden parallel aus der Sicht von $x_k, k \neq i$ jeweils $Beachtung_k$ mit eigenen Beachtungsgewichten α^k auf die gleiche Art und Weise berechnet. Die Beachtung wird trainiert, indem die Gewichtsmatrizen A, S, W im Training angepasst werden.

Bei einem mehrköpfigen Beachtungsmechanismus werden für jede Eingabe x_i gleich h verschiedene $Beachtung_i^m; m = 1, .., h$ berechnet, die jeweils ihre eigenen Gewichtsmatrizen $A^m, S^m, W^m; m = 1, .., h$ benutzen. Alle h verschiedene Beachtungsvektoren von x_i werden konkateniert und mit einer weiteren Matrix multipliziert, um den konkatenierten Vektor zurück auf die Größe eines einzelnen Beachtungsvektors zu skalieren. Die mehrfache Berechnung einer Beachtung pro Eingabe führt dazu, dass jedes Mal die größte Beachtung auf einer anderen Eingabe liegen kann. Dies kann zum Beispiel bei der Korreferenzauflösung von Pronomen hilfreich sein, um bei mehreren Pronomen nicht nur einen Kandidaten mit großer Beachtung zu haben.

In Abbildung 2.7 ist der prinzipielle Aufbau eines Transformer-Netzes dargestellt. Es gibt k Kodierer, die jeweils n Eingaben aufnehmen. Jeder Kodierer berechnet wie oben beschrieben die mehrköpfige Selbstbeachtung und schickt das Ergebnis davon durch ein

Feedforward-Netz. Der letzte Kodierer berechnet eine Kodiererwertmatrix und eine Kodiererschlüsselmatrix, die den „Kontext“ darstellen, welches an die Dekodierer weitergegeben wird. Die Dekodierer besitzen auch ein Modul mit mehrköpfiger Selbstbeachtung, der auf der Ausgabe der vorigen Dekodiererschicht bzw. der Ausgabe des vorherigen Zeitschritts berechnet wird. Die darauffolgende mehrköpfige Kodiererbeachtung führt ebenfalls die oben beschriebene Beachtungsberechnungen durch, jedoch wird dabei mit der Kodiererschlüsselmatrix und der Kodiererwertmatrix gearbeitet, um die Eingabeinformationen miteinzubeziehen.

2.6 Repräsentation von natürlicher Sprache

Bevor man Text für neuronale Netze oder Verfahren der Informationsrückgewinnung als Eingabe verwenden kann, müssen diese zuerst in eine numerische Form umgewandelt werden. Abschnitt 2.6.1, Abschnitt 2.6.2 und Abschnitt 2.6.3 erläutern Ansätze aus der Informationsrückgewinnung, deren Vektorrepräsentationen direkt mit dem Vorhandensein von Wörtern zusammenhängt. Abschnitt 2.6.4 listet eine Reihe von Verfahren auf, um kontinuierliche Wortvektoren zu generieren, die einige Defizite der IR-Ansätze beheben sollen.

Beispiel 2.4: 1-aus-N-Vektoren für einen Satz

Satz

Rosenkohl ist ein Wintergemüse.

1-aus-N-Vektor

$$\text{Wintergemüse} \rightarrow \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \begin{matrix} \text{Rosenkohl} \\ \text{Wintergemüse} \\ \text{ist} \\ \text{ein} \end{matrix}$$

2.6.1 1-aus-N-kodierte Vektoren

Einzelne Wörter eines Textes können mit einem 1-aus-N-kodiertem Vektor [JM09] dargestellt werden. Wie in Beispiel 2.4 zu sehen entspricht jeder Vektoreintrag einem Wort des Vokabulars. Um ein Wort zu repräsentieren wird die korrespondierende Zeile im Vektor auf Eins gesetzt, alle anderen Einträge sind Null. Die Dimension des Vektors entspricht also der Größe des Vokabulars.

2.6.2 Bag-of-Words

Während 1-aus-N-kodierte Vektoren einzelne Wörter repräsentieren, ist Bag-of-Words ein Ansatz zur Repräsentation von Texteinheiten (Dokumente, Sätze, Textteile,...) durch eine ungeordnete Menge von Wörtern [JM09]. Diese Menge von Wörtern wird wiederum durch einen Vektor ausgedrückt, wobei für jedes Wort die Anzahl seiner Vorkommen in der Texteinheit als Vektoreintrag festgehalten wird. In Beispiel 2.5 wird dies an zwei Sätzen demonstriert. Die Anzahl der Vektoreinträge wächst mit der Anzahl an Wörtern des Bag-of-Words.

Beispiel 2.5: Bag-of-Words für Sätze

Satz 1

Verschiedene Bücher befinden sich in verschiedenen Bereichen der Bibliothek.

Satz 2

Die Bibliothek befindet sich neben der Mensa.

Bag-of-Words

Satz 1	
verschieden	2
befinden	1
sich	1
Bereich	1
Mensa	0
Bibliothek	1

$\rightarrow \begin{pmatrix} 2 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \end{pmatrix}$

Satz 2	
verschieden	0
befinden	1
sich	1
Bereich	0
Mensa	1
Bibliothek	1

$\rightarrow \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}$

2.6.3 Vektorraummodelle

Vektorraummodelle [SWY75] werden vor allem in der Informationsrückgewinnung genutzt. Hierbei wird jedes Dokument zunächst auf einen Bag-of-Words-Vektor abgebildet. Danach kann eine zusätzliche Gewichtung der Wörter erfolgen: Als Gewicht wird in der Informationsrückgewinnung oft das tf-idf-Maß [DLMOP12] in Gleichung 2.9 eingesetzt.

$$tf - idf_{w,d} = \frac{\text{Anzahl Vorkommen von w in d}}{\text{Anzahl Wörter in d}} \cdot log(\frac{\text{Gesamtzahl Dokumente}}{\text{Anzahl Dokumente mit w}}) \quad (2.9)$$

w ist das Wort, für das das tf-idf-Gewicht berechnet wird und d ist das Dokument, aus dem das Wort stammt. Der linke Bruch ist umso größer, je öfter das Wort im Dokument auftaucht. Der rechte Bruch wird größer, je weniger Dokumente das Wort w beinhalten. Das tf-idf-Gewicht wird also am größten, wenn ein Wort in einem Dokument oft vorkommt, aber in anderen Dokumenten sehr selten. Die Intuition liegt darin, dass solche Wörter einen hohen Informationsgehalt besitzen, da sie besonders geeignet sind, die Dokumente zu diskriminieren. Kommt ein Wort in allen Dokumenten häufig vor, hilft das nicht bei der Unterscheidung. Nachdem die Vektoreinträge mit den Gewichten multipliziert wurden, wird paarweise die Ähnlichkeit zwischen dem Vektor des Anfragedokuments und den anderen Dokumentenvektoren mit Hilfe einer Distanzmetrik (zum Beispiel Kosinusähnlichkeit) berechnet und die ähnlichsten Dokumente als Antwort zurückgegeben. Da die Einträge der Vektoren, wie in Beispiel 2.5 zu sehen, Wörtern entsprechen, basiert die Ähnlichkeit auf das gemeinsame Auftreten von Wörtern im Dokument und in der Anfrage.

2.6.4 Worteinbettungen

1-aus-N-Vektoren und die Vektoren des IR-Vektorraummodells haben für jedes Wort im Vokabular eine eigene Dimension. Semantische oder grammatikalische Ähnlichkeiten zwischen Wörtern werden dadurch nicht kodiert. In Beispiel 2.4 ist die euklidische Distanz zwischen „Rosenkohl“ und „Wintergemüse“ genauso groß wie zwischen „Wintergemüse“

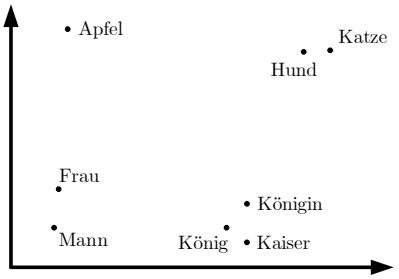


Abbildung 2.8: Schematische Veranschaulichung von beispielhaften Worteinbettungen in 2D.

und „ist“, obwohl die ersten beiden Wörter ähnlicher sind. Worteinbettungen repräsentieren Wörter durch reellwertige Vektoren und kodieren auch diese Ähnlichkeiten im Vektor. Dabei sollen die Einbettungen von ähnlichen Wörtern im Vektorraum nah beieinander liegen. Es gibt verschiedene Verfahren, um diese Vektoren für Wörter zu erzeugen, wobei die Vektoren für das gleiche Wort je nach Verfahren bezüglich ihrer Einträge und Dimension unterschiedlich aussehen können.

Eine Grundlagenarbeit wird von Bengio et al. in [BDVJ03] vorgestellt. Dort werden die Einbettungen durch Training eines neuronalen Netzes erzeugt. Die Trainingsaufgabe ist das Modellieren einer Funktion, die für alle Wörter des Vokabulars bestimmt, wie hoch ihre Auftrittswahrscheinlichkeit als nächstes Wort ist, wenn ein Kontext aus n Vorgängerwörtern gegeben ist. Das neuronale Netz besteht aus einer Eingabe- und Ausgabeschicht und zwei versteckte Schichten. Die erste versteckte Schicht bildet die Eingabewörter (also die n Vorgängerwörter) auf Worteinbettungen ab. Die Worteinbettungen entsprechen hier den Neurongewichten der ersten versteckten Schicht. Diese Schicht hat keine Aktivierungsfunktion und ist im Grunde nur eine Matrix, in der in jeder Zeile i die Worteinbettung für Wort w_i steht. Die Worteinbettungen der Vorgängerworte werden konkateniert und in dieser Form weitergegeben. Die zweite versteckte Schicht und die Ausgabeschicht repräsentieren die Funktion, die für jedes Wort des Vokabulars ihre Wahrscheinlichkeit als nächstes Wort berechnet. Die Worteinbettungen sind ein Parameter des neuronalen Netzes und werden im Training durch Gradientenabstieg nach und nach verbessert. Für die Worteinbettungen ist das Training folglich nur eine Pseudoaufgabe, da man die Vorhersage der Auftrittswahrscheinlichkeit nach dem Training nicht mehr benötigt.

Ein prominentes Beispiel aus [MYZ13] zur Ähnlichkeitskodierung ist die Möglichkeit, einfache Berechnung auf den Worteinbettungen ausführen zu können: Wird die Einbettung von „Mann“ von der Einbettung von „König“ subtrahiert und danach mit der Einbettung von „Frau“ addiert, ist das Resultat eine Einbettung bzw. ein Vektor, der am nächsten zur Einbettung von „Königin“ liegt. Aufgrund dieser Eigenschaft lassen sich Worteinbettungen besser verallgemeinern als zum Beispiel 1-aus-n-kodierte Wortvektoren; das heißt wiederum, dass Worteinbettungen auch gut mit unbekannten Wörtern funktionieren, die im Training nicht vorkamen, aber eine Ähnlichkeit zu anderen vorgekommenen Wörtern aufweisen. Worteinbettungen lassen sich darüber hinaus bezüglich der Vokabulargröße gut skalieren, da die Anzahl der Wortvektoreinträge von der Vokabulargröße unabhängig ist. Bei den Vektoren im IR-Vektorraummodell gibt es zum Beispiel für jedes Wort eine eigene Dimension, wodurch die Vektoren sehr groß und dünn besetzt sind. Einbettungen sind viel kleiner, dicht besetzt und dadurch ausdrucksstärker.

2.6.4.1 Word2Vec

Word2vec [MCCD13] ist eine Architektur und Implementierung zur Erzeugung von Worteinbettungen unter Verwendung eines künstlichen neuronalen Netzes. Das neuronales Netz von Word2Vec ist zu Beginn ähnlich wie Bengios Netz aufgebaut. Die erste versteckte Schicht, die Mikolov et al. als „Projektionsschicht“ umbenannt haben, kann genauso wie bei Bengio als eine aus den Neurongewichten zusammengesetzte Matrix, dessen Zeilen Worteinbettungen entsprechen, ansehen werden. Die Worteinbettungen aus dieser Schicht werden danach direkt zur die Ausgabeschicht weitergeleitet, es gibt also keine zweite versteckte Schicht. Wie Mikolov et al. anmerken, kann das Weglassen dieser Schicht zwar dazu führen, dass die Wörter nicht mehr so genau repräsentiert werden; dafür gibt es eine deutliche Effizienzsteigerung, da diese zweite versteckte Schicht unter den Schichten die größte Berechnungskomplexität besitzt.

Das Training besteht wieder aus der Vorhersage von Wörtern. Die Arbeit stellt hierfür zwei Modelle vor: Das Continuous Bag-of-Words-Modell (CBOW) und das Skip-Gram-Modell.

Beim CBOW-Modell wird der Kontext K des Wortes w in das neuronale Netz eingegeben mit der Aufgabe, w vorherzusagen. Der Kontext eines Wortes besteht aus insgesamt n seiner Vorgänger- und Nachfolgewörtern in einem Satz. Das Skip-Gram-Modell verläuft andersherum: Das Wort w wird als Eingabe genommen und damit sollen die Nachbarwörter in K vorhergesagt werden. Während dem Training haben sowohl w als auch die Wörter in K eine 1-aus-N-Kodierung (wobei n die Größe des gesamten Vokabulars ist). Diese werden anschließend in die Projektionsschicht eingegeben. Wegen der 1-aus-N-Kodierung entspricht die Verarbeitung in dieser Schicht das Auswählen einer Zeile aus der Matrix mit Neurongewichten. Dabei wird gerade die Zeile ausgewählt, die nach dem Training der Einbettung von w entspricht. Danach wird mit Hilfe dieser Auswahl, der als provisorischen Einbettung betrachtet werden kann, in der Ausgabeschicht die Wahrscheinlichkeiten für w bzw. K berechnet. Mit Hilfe einer Fehlerfunktion werden in beiden Fällen anschließend die Gewichte der Neuronen angepasst, um die Wahrscheinlichkeit für die Ausgabe w bzw. K zu maximieren. Nach der Trainingsphase wird der Vektor aus den Gewichten der Neuronen als Worteinbettung für w definiert.

Aus den Experimenten der Arbeit lässt sich außerdem sagen, dass das CBOW-Modell schneller ist, dafür ist das Skip-Gram-Modell bei seltenen Wörtern präziser. Das ist darauf zurückzuführen, dass beim CBOW-Modell aus dem gegebenen Kontext einen Durchschnittsvektor d aus den provisorischen Einbettungen berechnet wird, mit dem anschließend die Wahrscheinlichkeit $P(w|d)$ maximiert wird. Im Skip-Gram-Modell werden hingegen pro Eingabewort n Kontextwörter vorhergesagt, dabei werden die Wahrscheinlichkeiten $P(k|w), k \in K$ einzeln maximiert. Dadurch haben seltene (bzw. alle) Wörter beim Skip-Gram-Model mehr Trainingsdurchläufe als beim CBOW-Modell.

Im Vergleich zu Bengios Arbeit stellen Mikolov et al. mit Word2Vec eine effizientere Methode vor, die durch das Vereinfachen und Optimieren des neuronalen Netzes ein schnelleres Training für Worteinbettungen ermöglicht.

2.6.4.2 fastText

fastText [BGJM17] ist eine Erweiterung von Mikolovs Skip-Gram-Word2Vec (siehe Abschnitt 2.6.4.1). Hier werden Wörter nicht direkt einem Wortvektor zugewiesen, sondern zunächst in eine Menge von Buchstaben-N-Grammen zerlegt. Diese Menge enthält für jeden Buchstaben ihre Wortumgebung der Größe n sowie das ganze Wort. Für den Anfang, das Ende und das komplette Wort werden außerdem spitze Klammern $<, >$ als Kennzeichnung eingefügt. Zum Beispiel sieht diese Menge für das Wort „Fenster“ mit $n = 3$ wie in Beispiel 2.6.

Beispiel 2.6: Buchstaben-N-Gramme für „Fenster“ mit $n = 3$

$\{< Fe, Fen, ens, nst, ste, ter, er >, < Fenster >\}$

Durch die spitzen Klammern lässt sich zum Beispiel zwischen dem letzten „Fenster“-Trigramm $er >$ und dem Personalpronomen $< er >$ unterscheiden. Für jedes Element dieser Menge wird eine Einbettung berechnet. Die Summe dieser Einbettungen bildet schließlich die Einbettung für das Wort.

Durch diese Erweiterung kann fastText besser mit unbekannten Wörtern umgehen als Word2Vec, da unbekannte Wörter ebenfalls in N-Gramm-Mengen zerlegt werden und dabei die Wahrscheinlichkeit besteht, dass einige dieser N-Gramme bereits im Training gesehen wurden.

2.6.4.3 GloVe

GloVe (Global Vectors) von Pennington et al. [PSM14] ist ein weiteres Verfahren, um Worteinbettungen zu generieren. Techniken wie Word2Vec versuchen, den Kontext innerhalb eines Satzes vorherzusagen, das heißt das neuronale Netz realisiert eine Funktion, die die Worteinbettungen auf Wahrscheinlichkeiten für die Kontextwörter abbildet. Bei GloVe soll nicht nur der Kontext im Satz, sondern der komplette Korpus miteinbezogen werden (daher auch Global Vectors); aus diesem Grund wird in GloVe mit globalen Häufigkeiten über das gemeinsame Auftreten von Wörtern gearbeitet. Die Funktion, die in GloVe verwendet wird, soll, wie in Gleichung 2.10 zu sehen, drei Eingabeworteinbettungen auf den Quotienten der gemeinsamen Auftrittshäufigkeit der ersten beiden Wörter geteilt durch die gemeinsame Auftrittshäufigkeit der letzten beiden Wörter abbilden.

$$F(w_i, w_j, w_k) = \frac{\text{gemeinsame Auftrittshäufigkeit } w_k \text{ mit } w_i}{\text{gemeinsame Auftrittshäufigkeit } w_k \text{ mit } w_j} \tag{2.10}$$

Die gemeinsamen Auftrittshäufigkeiten werden bestimmt, indem im *gesamten* Korpus abgezählt wird, wie oft Wörter im Kontext anderer Wörter auftreten. Danach kann die Gleichung nach w_k umgeformt werden, um die Worteinbettung zu berechnen. Die Funktion F ist eine Kombination von Vektorsubtraktion und Skalarprodukt der Eingabeworteinbettungen, um die vektoriellen Worteinbettungen auf die skalaren Auftrittshäufigkeiten abbildbar zu machen.

2.6.4.4 ELMo

Word2Vec und fastText generieren für jedes Auftreten des Wortes „Bank“ immer die gleiche Einbettung, egal ob im Satz von einer Sitzbank oder einem Kreditinstitut die Rede ist. Mit ELMo (Embeddings from Language Models) [PNI+18] integrieren Peters et al. tiefergehende Kontextinformationen in die Worteinbettungen, um zum Beispiel Polysemie wie bei „Bank“ zu berücksichtigen. Diese Kontextsensitivität wird dadurch erreicht, dass für die Erzeugung der Einbettung der komplette Satz mit seinen Wörtern $(w_1, .., w_n)$ miteinbezogen wird. Das Vorgehen von ELMo ist in Abbildung 2.9 aufgezeigt. Es wird ein Netz mit L LSTM-Schichten eingesetzt, in das in jedem Zeitschritt t_i ein Wort w_i eingespeist wird, wobei das Wort als Menge seiner Buchstabeneinbettungen repräsentiert wird. Die initialen Buchstabeneinbettungen werden vorher durch andere Einbettungsverfahren getrennt berechnet und müssen nicht kontextsensitiv sein. Durch die rekurrenten Neuronen der versteckten LSTM-Schichten werden auch die Informationen des vorherigen Zeitschritt (das heißt die vorherigen Wörter $(w_1, .., w_{i-1})$ im Satz) bei jedem aktuellen

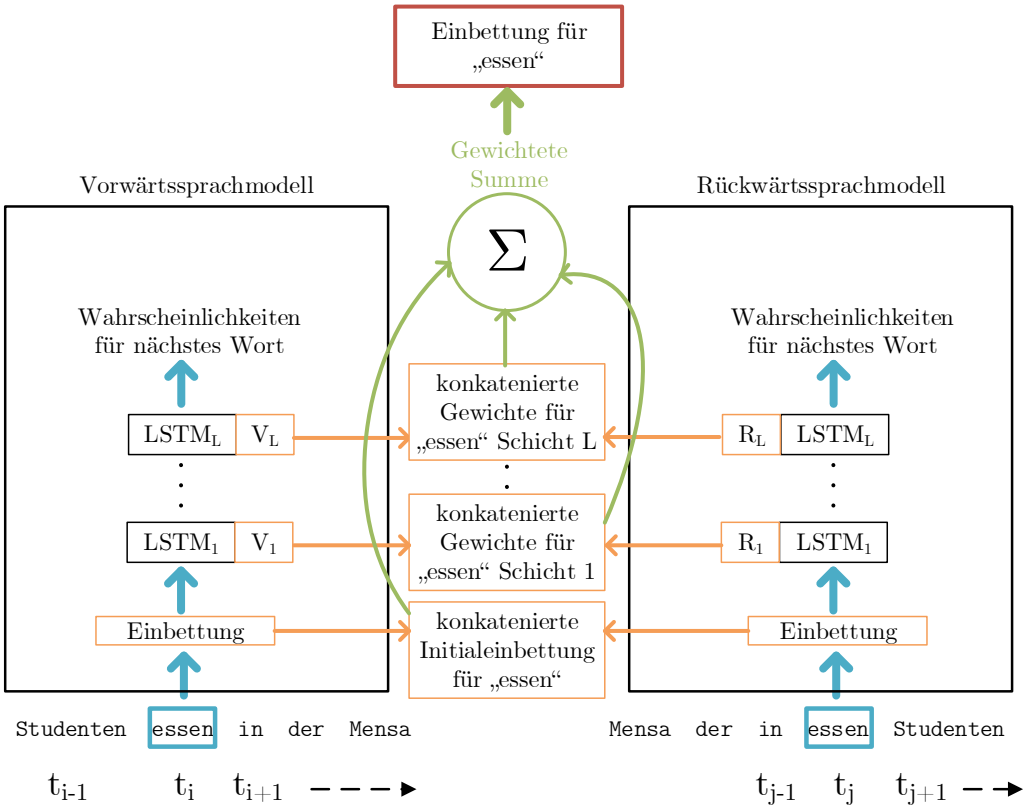


Abbildung 2.9: Ein Überblick über die Erzeugung einer Einbettung für das Wort „essen“. V_i und R_i sind die Gewichte der LSTM-Schichten.

Wort w_i mitberücksichtigt. Im Vorwärtsfall in Abbildung 2.9 fließen also beim Wort „essen“ auch Informationen vom Wort „Studenten“ mit ein. Die Trainingsaufgabe besteht darin, das wahrscheinlichste nächste Wort w_{i+1} vorherzusagen. Damit die restlichen Wörter $(w_{i+1}, ..w_n)$ des Satzes auch in die Berechnung der Worteinbettung für w_i miteinbezogen werden, wird der komplette Satz mit umgedrehter Reihenfolge der Wörter in ein zweites LSTM-Netz mit dem gleichen Vorhersagevorgehen wie oben beschrieben eingesetzt; in Abbildung 2.9 ist das auf der rechten Seite skizziert. Dieses Verfahren, bei der die Vorhersage einmal vorwärts und einmal rückwärts durchlaufen wird, wird als bidirektionales Sprachmodell bezeichnet. Nachdem beide Sprachmodelle trainiert sind, wird die Worteinbettung für w_i schließlich wie folgt berechnet (siehe auch den mittleren Teil in Abbildung 2.9): Für w_i werden die Gewichte der LSTM-Schichten des Vorwärtssprachmodells mit denen des Rückwärtssprachmodells schichtweise miteinander zu jeweils einem Vektor konkateniert. Das wird ebenfalls mit den initialen Buchstabeneinbettungen gemacht. Die gewichtete Summe aus diesen $L+1$ Vektoren bildet anschließend die Worteinbettung für w_i. Die Gewichte hängen davon ab, welche Aufgabe später mit den Worteinbettungen gelöst werden sollen.

Der Einsatz von mehreren LSTM-Schichten hilft dabei, die Worteinbettungen kontextsensitiver zu machen. Wie sich in der Evaluation gezeigt hat, werden in den niedrigeren Schichten vor allem Syntaxinformationen eingefangen, während in den Gewichten von höheren Schichten eher kontextabhängige Informationen wie Wortbedeutungen gespeichert werden.

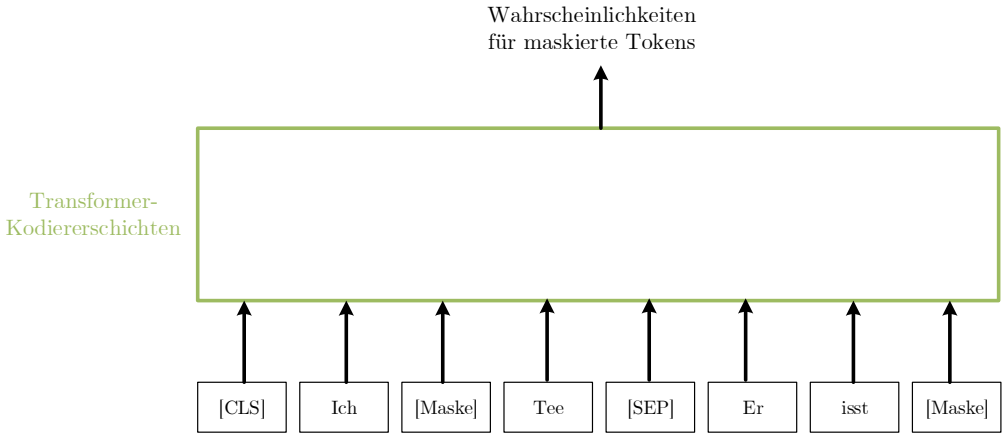


Abbildung 2.10: Schematischer Aufbau von Bert während der Vorhersage der maskierten Tokens.

2.6.4.5 BERT

Ähnlich wie ELMo versuchen Devlin et al. bei BERT (Bidirectional Encoder Representations from Transformers) [DCLT19] den bidirektionalen Kontext (alle Nachbarwörter links und rechts im Satz) mit in die Einbettung des aktuellen Worts einfließen zu lassen. Der Ansatz unterscheidet sich jedoch: Bei ELMo besteht die Bidirektionalität eigentlich aus zwei unabhängigen unidirektionalen Berechnungen mit dem (umgedrehten) Satz, welche hinterher mit einer gewichteten Summe kombiniert werden. Bei BERT wollten die Autoren eine „echte“ Bidirektionalität umsetzen: Dazu haben sie als Erstes die Trainingsaufgabe geändert - aus einem gegebenen Satz werden 15 Prozent der Wörter maskiert und die Aufgabe besteht nun darin, mit Hilfe der verbliebenen 85 Prozent der Wörter die maskierten Wörter vorherzusagen. Bei ELMo werden für eine Vorhersage entweder nur die Wörter links oder rechts des Satzes verwendet; bei BERT können die 85 Prozent der Wörter gleichzeitig links und rechts liegen. Devlin et al. führen an, dass so ein „echtes“ bidirektionales Modell mächtiger ist als Ansätze wie ELMo. BERT wurde außerdem mit einer zweiten Trainingsaufgabe trainiert, in der es den nächsten Satz voraussagen soll. Dazu werden zwei Sätze auf einmal, getrennt durch ein spezielles *[SEP]*-Token, eingegeben, wobei in 50% der Fälle der zweite Satz wirklich der nächste Satz ist; in den anderen Fällen wird ein zufälliger anderer Satz eingegeben. BERT versucht hierbei zu bestimmen, ob der zweite Satz ein echter nächster Satz ist oder nicht. Dadurch soll BERT auch Beziehungen zwischen Sätzen lernen.

Ein anderer Unterschied zu ELMo ist der Einsatz von Transformer-Netzen (siehe Abschnitt 2.5.4); allerdings wird in BERT nur die Kodiererhälfte verwendet, da man nur die Zwischenrepräsentationen (Einbettungen) haben möchte. Es können also mehrere Wörter - bzw. hier zwei Sätze - gleichzeitig in das Netz eingegeben werden, die teilweise maskiert sind. Als Anfangstoken steht des Weiteren immer ein spezielles *[CLS]*-Token. Die Eingaben werden auf initiale Einbettungen abgebildet und werden danach parallel durch mehrere Kodiererschichten hintereinander geschickt, bis in der Ausgabeschicht schließlich die Wahrscheinlichkeiten für die maskierten Wörter berechnet werden, dies ist in Abbildung 2.10 dargestellt. Durch den Beachtungsmechanismus werden die Nachbarwörter für die Verarbeitung eines Wortes gewichtet miteinbezogen. Die Worteinbettungen sind am Ende die Gewichte in der letzten Kodiererschicht. Die Gewichte für das *[CLS]*-Token entsprechen der Einbettung für den ganzen Satz.

In BERT werden eigentlich keine Wörter, sondern Subwörter eingegeben. Die Subwörter von „playing“ sind zum Beispiel „play“ und „##ing“, wobei „##“ den Suffix markiert. Das

reduziert die Vokabulargröße, da Suffixe bzw. Wortteile herausfaktorisiert werden und es erhöht auch die Verallgemeinerbarkeit auf im Training nicht gesehene Wörter. Es existieren also pro Subwort und pro Bedeutung (da kontextsensitiv) eine Einbettung, welches den Gewichten der letzten Kodiererschicht entspricht.

Des Weiteren wurde BERT so konzipiert, dass man es einfach durch Feinkalibrierung (engl. fine-tuning) für andere Aufgaben in der natürlichen Sprachverarbeitung einsetzen kann. BERT wird auf großen (dadurch sehr teuer bzw. zeitintensiv) Textkorpora vortrainiert, was nur einmal ausgeführt werden muss. Für die aufgabenspezifische Feinkalibrierung wird danach ein überwachtes Training mit entsprechenden Trainingsdaten als Eingabe durchgeführt, wobei die Gewichte aus dem Vortraining dadurch aufgabenspezifisch angepasst werden (billiger und schneller als das Vortraining).

2.6.4.6 Doc2Vec

Die bisherigen Einbettungsverfahren können nur pro Wort eine Einbettung erzeugen (Nur BERT kann auch Satzeinbettungen erzeugen). Doc2Vec [LM14] ist ein Verfahren, um Einbettungen für beliebig lange Textteile, sogenannte Paragrafeneinbettungen, zu generieren. In Abschnitt 2.6.3 wurden Vektoren des IR-Vektorraummodells vorgestellt, die ganze Dokumente repräsentieren können. Da diese Vektoren auf Bag-of-Words basieren, sind sie zum einen sehr groß und dünn besetzt, zum anderen gibt es keine Ordnung unter den Wörtern. So haben Sätze die gleiche Vektorrepräsentation, solange sie die gleichen Wörter beinhalten. Zum Beispiel werden „Gehen wir essen?“ und „Wir gehen essen.“ auf den gleichen Vektor abgebildet, obwohl die Sätze offensichtlich unterschiedliche Bedeutungen haben. Außerdem bleiben semantische Ähnlichkeiten unberücksichtigt.

Doc2Vec funktioniert sehr ähnlich wie Word2Vec (siehe Abschnitt 2.6.4.1). Hier gibt es auch zwei Trainingsmodelle: „Distributed Memory Model of Paragraph Vectors (PV-DM)“ funktioniert so wie das CBOW-Modell in Word2Vec - es wird eine Anzahl an aufeinanderfolgenden Wörtern aus dem Paragrafen in ein neuronales Netz eingegeben. Die Autoren haben hier mit acht Wörtern die besten Ergebnisse erzielt. Zusätzlich wird aber ein Paragraftoken eingespeist, der auch auf eine Einbettung abgebildet werden soll, das heißt die Neuronen haben jeweils ein weiteres Gewicht, das am Ende zusammen die Paragrafeneinbettung bildet. Die Trainingsaufgabe besteht darin, anhand der Eingaben das nächste Wort vorherzusagen. Der Paragrafentoken lernt dazu, wenn die anderen Wörter im Paragrafen vorhergesagt werden.

Das zweite Trainingsmodell ist „Distributed Bag of Words version of Paragraph Vector (PV-DBOW)“ und ist analog zum Word2Vecs Skip-Gram-Modell. Hier müssen Wörter aus einem zufälligen Kontextfenster nur mit dem Paragraftoken vorhergesagt werden, damit durch den Paragraftoken gelernt wird, welche Wörter im Paragrafen vorhanden sind.

2.7 Repräsentation von Quelltext

Genauso wie natürlichsprachiger Text muss Quelltext zunächst numerisch repräsentiert werden, bevor Verfahren aus dem machinellen Lernen darauf anwendet werden können. Analoge Verfahren mit dem IR-Vektorraummodell oder mit Worteinbettungen können hierbei verwendet werden. Diese werden in Abschnitt 2.7.1 bzw. Abschnitt 2.7.2 erläutert.

2.7.1 Bag-of-Words und IR-Vektorraummodelle mit Quelltext

Techniken wie Bag-of-Words (Abschnitt 2.6.2) oder IR-Vektorraummodelle (Abschnitt 2.6.3) sind auch auf Quelltext anwendbar. Dazu werden pro Quelltextausschnitt alle Bezeichner (Namen von Klassen, Variablen, etc.) als „natürlichsprachigen Text“ auffasst und darauf die Verfahren aus der Informationsrückgewinnung angewendet. Dies wurde zum Beispiel in [ACCL00] durchgeführt.

2.7.2 Quelltexteinbettungen

Die IR-Methoden mit Quelltext erben die gleichen Nachteile wie etwa die Proportionalität der Vektordimension mit der Größe des Vokabulars. Ein zusätzlicher Nachteil besteht darin, dass quelltextspezifische Merkmale ignoriert werden. Die folgenden Unterabschnitte erklären existierende Ansätze, die diesen Mängeln entgegenwirken sollen. Dazu werden Quelltexteinbettungen generiert, die analog wie Worteinbettungen Quelltextausschnitte mit einem reellwertigen Vektor repräsentieren. Quelltexteinbettungen sollen ferner die Ähnlichkeit zwischen ihren Quelltextausschnitten mitkodieren, wobei diese Ähnlichkeit auch durch quelltextspezifische Eigenschaften mitbestimmt wird.

2.7.2.1 Code2Vec und Code2Seq

Code2Vec [AZLY19] ist ein Verfahren, um Quelltextausschnitten, insbesondere Methoden, einen Quelltextvektor zuzuordnen.

Dazu wird ein neuronales Netz mit der Aufgabe trainiert, Methodennamen vorherzusagen. Als Eingabe wird dabei der abstrakte Syntaxbaum (AST) der Methode eingespeist. Für alle Blätter und Pfade ohne Blätter im Syntaxbaum werden Einbettungen trainiert. Die Berechnung des Quelltextvektors für die gesamte Methode läuft wie folgt ab: Alle Pfade zwischen Blättern im Syntaxbaum werden extrahiert und es wird für jeden Pfad ein Pfadvektor berechnet. Der Pfadvektor ist die Konkatenation der Einbettungen für das Startblatt, das Endblatt und dem Pfad, der Start- mit Endblatt verbindet. Der Pfadvektor wird durch eine weitere neuronale Netzschicht geleitet, welches den konkatenierten Pfadvektor in eine andere Zwischenrepräsentation, kombinierter Kontextvektor genannt, umwandelt. Diese Netzschicht soll lernen, wie man am besten die Werte der konkatenierten Vektoren zusammenfasst. Um nun einen Quelltextvektor für die Methode zu erhalten, werden die kombinierten Kontextvektoren gewichtet aufsummiert. Die Gewichte repräsentieren ähnlich wie bei Transformer (Abschnitt 2.5.4) die Beachtung, die jeder Pfad bekommt und werden so trainiert, dass Pfade, die die Kernfunktionalität der Methode beschreiben, mehr Beachtung erhalten (Der vorherzusagende Methodenname beschreibt im Grunde die Kernfunktionalität).

Code2vec ist programmiersprachenabhängig, da es auf dem abstrakten Syntaxbaum aufbaut.

Alon et al. haben 2019 mit Code2Seq [ABLY19] eine verbesserte Version von Code2Vec veröffentlicht. Die Grundidee, einen Quelltextausschnitt Menge seiner AST-Pfade zu repräsentieren, ist erhalten geblieben. Geändert hat sich die Verarbeitung der AST-Pfade: Während bei Code2Vec die Pfade aus der Konkatenation seiner Knoten berechnet wurde, werden die Knoten bei Code2Seq durch ein Kodierernetz (Abschnitt 2.5.3) verarbeitet. Dadurch lassen sich die Pfade besser repräsentieren, da die Vektoren durch ein neuronales Netz gelernt werden. Außerdem zerteilt Code2Seq Bezeichner mit Binnenmajuskelschreibweise und lernt aus den einzelnen Teilen durch ein Kodierernetz eine Einbettung für den ganzen Bezeichner. In Code2Vec wurde ein Bezeichner als Ganzes auf eine Einbettung abgebildet. Code2Seq ist aus diesem Grund toleranter gegenüber OOV-Bezeichner. Die Ausgabe in Code2Seq erfolgt durch ein Dekodierernetz, in der die Bestandteile des resultierenden Methodennamens einzeln vorhergesagt werden. Dabei wird wie in Code2Vec ein Beachtungsmechanismus eingesetzt, um die Pfadeinbettungsausgaben aus dem Kodierer zu gewichten. In Code2Vec wurde der Methodenname als Ganzes vorhergesagt. Dabei gab es eine feste Menge von möglichen Methodennamen. Auch hier ist Code2Seq flexibler: Code2Seq kann Kombinationen von Methodennamenbestandteilen ausgeben, während Code2Vec nur feste, ganze Methodennamen ausgeben kann. Durch die Anpassungen in Code2Seq konnten die Autoren in der Methodennamenvorhersage eine Verbesserung von über 20% F1-Maß erzielen.

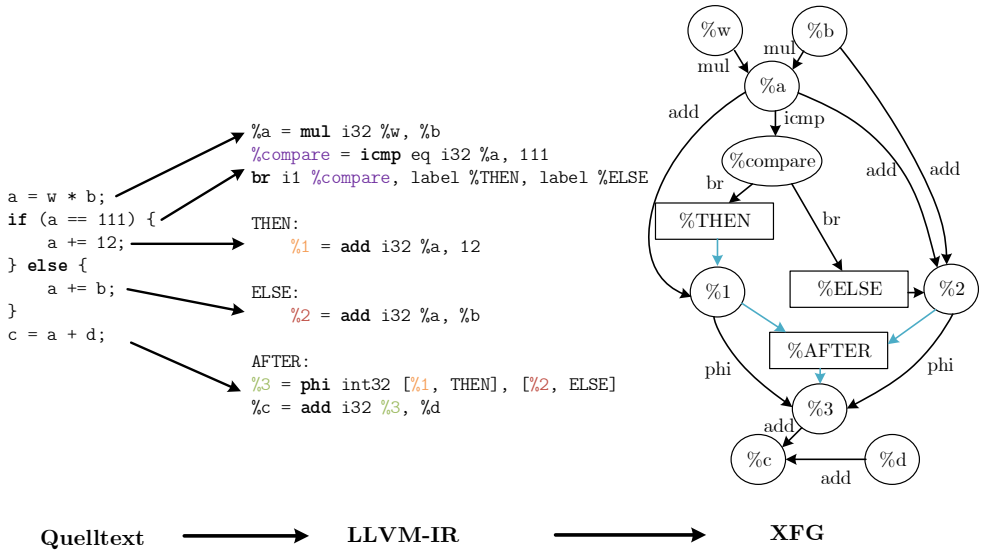


Abbildung 2.11: Beispiel einer Transformation von Quelltext über die LLVM-IR zum XFG. Alle Variablen seien vom Typ Integer. „i32“ ist der Integertyp in der LLVM-IR.

2.7.2.2 inst2vec

Ein programmiersprachenunabhängiger Ansatz zur Generierung von Quelltextvektoren ist inst2vec [BJH18] von Ben-Nun et al. Die Unabhängigkeit wird dadurch erreicht, dass der Quelltext zuerst in die LLVM-Zwischenrepräsentation (LLVM intermediate representation, LLVM-IR) transformiert wird. LLVM [LA04] besteht aus einer Kompiliererrahmenarchitektur, einer eigenen LLVM-Zwischenrepräsentation und aus Bibliotheken, um Maschinenquelltext zu erzeugen.

Die Idee hinter Worteinbettungen bestand darin, dass Wörter, die in ähnlichen Kontexten auftreten auch ähnliche Wortvektoren haben; wobei der Kontext die Nachbarwörter sind. Um dies auf Ausdrücke (engl. Statements) im Quelltext zu übertragen, haben Ben-Nun et al. zunächst definiert, was sie darunter verstehen: Der Kontext eines Ausdrucks sind die anderen Ausdrücke, deren Ausführung direkt voneinander abhängen. Das bedeutet konkret, dass Kontrollfluss- und Datenabhängigkeiten zwischen Ausdruck und Kontext bestehen. Die Darstellung in der LLVM-IR ist im Beispiel in Abbildung 2.11 zu sehen. Daraus können die Datenflussabhängigkeiten wie folgt extrahiert werden: Jede Instruktion nimmt Eingabeargumente entgegen, führt eine Operation durch und weist das Ergebnis einer Ausgabevariable zu. Dadurch bestehen Datenabhängigkeiten zwischen den Eingabeargumenten und der Ausgabe, also zum Beispiel in der ersten Zeile jeweils zwischen den Argumenten `w`/`b` und `a`. Die LLVM-IR unterliegt der „Static single assignment (SSA)“-Form. Das bedeutet, dass jeder Variable nur einmal etwas zugewiesen werden darf. In Abbildung 2.11 müssen dadurch neue Hilfsvariablen für `a` eingeführt werden (`%1` und `%2`). Für den Kontrollfluss gibt es Sprungoperanden und Labels, die Sprungursprung und -ziel markieren. Außerdem gibt es einen speziellen ϕ-Operand, der zum Einsatz kommt, wenn der Wert einer Variable davon abhängt, welche Verzweigung vorher traversiert wurde. Im vorliegenden Beispiel ist dies bei der Variable `a` der Fall. Der ϕ-Operand listet als Argumente alle möglichen Verzweigungen auf, die zum Wert von `a` führen können. Dadurch lassen sich Datenabhängigkeiten auch bei Sprüngen nachverfolgen.

Die Autoren definieren mit Hilfe der LLVM-IR einen „kontextuellen Flussgraphen (engl. ConteXtual Flow Graph, kurz XFG)“, welcher diese Daten- und Kontrollflussabhängigkeiten zusammenfasst. Alle Variablen und Labels werden zu Knoten. Konstanten werden

Beispiel 2.7: Überführung von Quelltext zu LLVM-IR zu Quelltexttripel

Quelltext

```
sum = 1 + 2;
```

LLVM-Zwischenrepräsentation

```
%sum = add i32 1, 2
```

Quelltexttripel

```
<add, "TypeOf", IntegerType>
<add, "Arg_1", CONST>
...
```

weggelassen. Kanten modellieren die Datenflussabhängigkeiten und haben den jeweiligen Operand als Kantenbeschriftung. Für die erste Zeile in Abbildung 2.11 gibt es also Knoten für `a`, `w` und `b`, wobei die letzteren beiden jeweils eine Kante nach `a` mit der Beschriftung `mul` haben. Zur Vervollständigung des Kontrollflusses gibt es außerdem unbeschriftete (im Beispiel blaue) Kanten zwischen Labels und der Variable der nächsten auszuführenden Instruktion.

Schließlich werden Einbettungen für die Ausdrücke mit dem Skip-Gram-Modell [MCCD13] von Mikolov et al. trainiert; dazu wird aus dem XFG ein Ausdrucksgraph erzeugt, in der jede LLVM-Instruktion einen Knoten besitzt. Kanten bestehen zwischen Instruktionen, wenn sie im XFG entsprechende Kanten besaßen. Beim Training wird eine Instruktion also als Eingabe genommen und es wird versucht, den Kontext (die Nachbarknoten) vorherzusagen. Die Gewichte der Neuronen entsprechen am Ende den Einbettungen.

2.7.2.3 IR2Vec

IR2Vec [SAJ+19] nutzt genau wie inst2vec die LLVM-Zwischenrepräsentation (LLVM-IR) als Basis, jedoch ist die Generierung der Quelltexteinbettungen eine andere. Es können in IR2Vec Quelltexteinbettungen auf verschiedenen Granularitätsebenen erzeugt werden - Einbettungen für Instruktionen, LLVM-Basisblöcke, Funktionen und Programme. Da Basisblöcke aus einer maximalen Menge an hintereinanderfolgenden Instruktionen ohne Sprünge bestehen, werden Basisblockeinbettungen durch Aggregation ihrer Instruktionseinbettungen berechnet. Analog wird dies bei Funktionen (bestehend aus Basisblöcken) und Programmen (bestehend aus Funktionen) durchgeführt.

Die Erzeugung der Instruktionseinbettungen läuft wie folgt ab: Der vorliegende Trainingsquelltext wird als Erstes in die LLVM-Zwischenrepräsentation überführt. Daraus werden Relationen in Form von Quelltexttripeln erstellt. Es gibt drei Arten von Relationen: `TypeOf` charakterisiert den Typ einer Instruktion, `Arg` die Argumente und `NextInst` die nächste Instruktion. Die Quelltexttripel haben danach die folgende Gestalt: `<Instruktionsname, Relationsart, Typ/Argument/Nächste Instruktion (Je nach Relationsart)>`.
Ein Minimalbeispiel ist in Beispiel 2.7 gezeigt. „i32“ ist der Integertyp in LLVM-IR. Außerdem werden Identifizierer und Konstanten in vordefinierte Labels wie `CONST`, `VAR`, etc. umbenannt, was die Quelltexttripel generischer macht. Aus diesen Quelltexttripel werden

Einbettungen trainiert, welches das Startwerteinbettungsvokabular (engl. seed embedding vocabulary) bilden. Mit Hilfe dieses Vokabulars werden Instruktionseinbettungen gebildet. Instruktionen bestehen aus mehreren Quelltexttripeln (wie auch in Beispiel 2.7 zusehen); das heißt Quelltexttripel repräsentieren Teilinstruktionen. Für die Berechnung der Instruktionseinbettung wird daher eine gewichtete Summe der korrespondierenden Einbettungen aus dem Startwerteinbettungsvokabular gebildet. Die Gewichte werden heuristisch bestimmt, zum Beispiel fällt der Typ einer Instruktion mehr ins Gewicht als ein Argument. Falls ein Argument in einer vorherigen Instruktion definiert wurde, wird die vorige Instruktionseinbettung zur aktuellen Argumenteinbettung dazu addiert. Dadurch werden Datenflussabhängigkeiten miteinbezogen. Falls der Wert eines Arguments davon abhängt, welche Verzweigung vorher genommen wurde, werden die Einbettungen aller möglichen Verzweigungen mit einem Gewicht zur aktuellen Argumentquelltexttripeleinbettung addiert. Dieses Gewicht entspricht der Wahrscheinlichkeit, dass entlang der Verzweigung traversiert wird, was mit einer statischen Analyse durch den Kompilierer berechnet wird. Damit sind auch Kontrollflussabhängigkeiten berücksichtigt.

Eine Basisblockeinbettung wird durch die ungewichtete Summe seiner lebenden (engl. live) Instruktionseinbettungen gebildet. Eine Instruktion ist lebend, wenn ihr Wert in einer späteren Instruktion eines anderen Basisblocks noch verwendet wird. Die Funktionseinbettung ist die ungewichtete Summe seiner Basisblockeinbettungen und die Programmeinbettung entspricht der ungewichteten Summe seiner Funktionseinbettungen.

Im Gegensatz zu inst2vec werden bei IR2Vec aus der LLVM-IR also keine XFGs, sondern Teilinstruktionen in Form von Quelltexttripeln extrahiert. Durch entsprechende Gewichte werden hier ebenfalls Kontroll- und Datenflussabhängigkeiten miteinbezogen. In einem Experiment der Autoren, in der sie die Aufgabe der Programmklassifikation einmal mit und einmal ohne Einbezug der Kontrollfluss- und Datenabhängigkeiten durchgeführt haben, hat sich herausgestellt, dass das Miteinbeziehen der Abhängigkeiten eine höhere Präzision ergibt. Durch das Training auf den generischen Quelltexttripeln sind die Einbettungen außerdem gut auf im Training ungesehene Instruktionen übertragbar; gleichzeitig benötigt IR2Vec weniger Trainingsdaten, da es auf Teilinstruktionsebene agiert, wohingegen inst2vec auf Instruktionsebene arbeitet.

2.8 Ähnlichkeitsmetriken für Einbettungen

Einbettungen kodieren die Ähnlichkeit ihrer repräsentierten Wörter durch die Distanz im Vektorraum. Umd die Distanz zu messen, gibt es unterschiedliche Optionen.

2.8.1 Euklidische Distanz

Die euklidische Distanz zweier Punkte u, v berechnet sich wie in Gleichung 2.11.

$$euklid(u, v) = ||u - v||_2 \tag{2.11}$$

Die euklidische Distanz entspricht der Länge der Strecke, die die beiden Punkte verbindet. Je kleiner die Distanz, desto ähnlicher sind sich die Punkte bzw. Einbettungen. Der Wertebereich beträgt $[0, \infty)$.

2.8.2 Kosinusähnlichkeit

Die Kosinusähnlichkeit bezieht den Winkel ϕ zwischen den Vektoren mit ein, wie man in Gleichung 2.12 sehen kann.

$$cossim(u, v) = \frac{u \cdot v}{||u||_2||v||_2} = \cos\phi \tag{2.12}$$

Die Vektoren werden außerdem durch ihre Länge normiert. Das führt dazu, dass die Länge der Vektoren im Gegensatz zu euklidischen Distanz keinen Einfluss auf die Ähnlichkeit hat. Bei Vektoren aus Vektorraummodellen wie in Abschnitt 2.6.3 wird dadurch die Ähnlichkeit nicht durch die Anzahl an Wortvorkommen beeinflusst. Der Wertebereich ist $[-1, 1]$. Bei Vektoren, die sich aus Auftrittswahrscheinlichkeiten berechnen und daher positiv sind, liegt der Wertebereich zwischen $[0, 1]$. Dabei ist 1 die maximale Ähnlichkeit, das heißt beide Vektoren sind identisch.

2.8.3 Word Mover's Distance

Die vorherigen Metriken messen die Ähnlichkeit zwischen zwei einzelnen Vektoren. Die Word Mover's Distance (WMD) [KSKW15] ist dagegen ein Ähnlichkeitsmaß zwischen Dokumenten bzw. allgemeiner zwischen Mengen von Vektoren.

Die Distanz entspricht dabei der minimalen kumulativen euklidischen Distanz, die nötig ist, um alle Vektoren aus der einem Menge auf die Vektoren der anderen Menge abzubilden. Da die WMD auf der euklidischen Distanz basiert, ist der Wertebereich $[0, \infty]$ und die Ähnlichkeit zweier Dokumente ist umso größer, je kleiner die Distanz ist.

3 Verwandte Arbeiten

In der Literatur wurden für die Rückverfolgbarkeit oft Techniken aus der Informationsrückgewinnung eingesetzt. Diese werden in Abschnitt 3.1 vorgestellt. Die Rückverfolgbarkeit zwischen Anforderungen und Quelltext mit Einbettungen wurde zwar bisher noch nicht untersucht, dafür gab es bereits Arbeiten zur Rückverfolgbarkeit zwischen anderen Artefakten. Dazu wird in Abschnitt 3.2 ein Überblick gegeben. Schließlich werden in Abschnitt 3.3 verwandte Arbeiten aufgeführt, die eine Abbildung zwischen unterschiedlichen Vektorräumen vornehmen. Dies muss durchgeführt werden, falls Anforderungs- und Quelltexteinbettungen in unterschiedlichen Vektorräumen abgebildet werden.

3.1 Rückverfolgbarkeit ohne Einbettungen

In der Vergangenheit wurde für die Rückverfolgbarkeit von Anforderungen oft die Informationsrückgewinnung (siehe Abschnitt 2.2) als praktikabler Ansatz angesehen und viele Optimierungen und Evaluationen vorgenommen [BRA14].

IR-Ansätze stützen sich auf die Grundlage, dass die zu verbindenden Artefakte textuell ähnlich beschrieben sind; je ähnlicher, desto größer die Wahrscheinlichkeit für eine Rückverfolgbarkeitsverbindung [DLMOP12]. Die Ähnlichkeit misst sich dabei größtenteils am Auftreten des gleichen Wortes in den Artefakten. Das allgemeine Vorgehen bei der Rückverfolgbarkeit mittels Informationsrückgewinnungstechniken wird von De Lucia et al. [DLMOP12] in vier Kernschritte unterteilt:

1. Dokumente (Artefakte) vorverarbeiten
2. Anwendung einer IR-Technik auf die Dokumente
3. Erzeugung einer nach Ähnlichkeit sortierten Kandidatenliste für Rückverfolgbarkeitsverbindungen
4. Analyse der Kandidatenliste, um tatsächliche Rückverfolgbarkeitsverbindungen zu identifizieren

In **Information retrieval models for recovering traceability links between code and documentation** [ACCL00] von Antoniol et al. wurden zum Beispiel IR-Vektorraummodelle für die Rückverfolgbarkeit zwischen Quelltext und Textdokumenten wie einer Bedienungsanleitung oder funktionalen Anforderungen eingesetzt. Aus den Textdokumenten werden nach einer Vorverarbeitung (alles in Kleinbuchstaben transformieren, Stoppwörter entfernen, Lemmatisierung) Dokumentvektoren wie in Abschnitt 2.6.3 generiert. Pro Quelltextklasse wird ebenfalls ein Dokumentenvektor erzeugt: Hierfür werden aus der Klasse

zunächst alle Bezeichner (Klassennamen, Attributnamen, etc.) extrahiert und in einzelne Wörter getrennt, falls der Bezeichner ein zusammengesetztes Wort ist. Die Bezeichner(-fragmente) werden anschließend als „natürlichsprachigen Text“ behandelt und durchlaufen die gleiche (Vor-)Verarbeitung wie die Wörter aus den Textdokumenten. Am Ende gibt es pro Quelltextklasse einen Dokumentenvektor. Die Vektoren der Quelltextklassen entsprechen den Anfragevektoren; diese werden paarweise mit den Textdokumentvektoren anhand der Kosinusähnlichkeit verglichen und daraus eine Liste mit möglichen Rückverfolgbarkeitsverbindungen (sortiert nach ihrer Ähnlichkeit) erstellt.
Die Evaluation wurde an zwei Fallstudien durchgeführt. In der ersten Fallstudien mussten 208 Quelltextklassen zu 88 Seiten einer Bedienungsanleitung zugeordnet werden. Wenn man jeweils nur den ersten Rückverfolgbarkeitskandidaten betrachtet, erreicht man eine Präzision von 25% und eine Ausbeute von 53%. Bezieht man jeweils die ersten 12 Kandidaten mit ein, beträgt die Präzision 4% und die Ausbeute 100%. Die zweite Fallstudie bestand aus 60 Quelltextklassen und 16 Anforderungen. Mit jeweils dem ersten Kandidaten wurden 48% Präzision und 50% Ausbeute erreicht. 100% Ausbeute mit 14% Präzision wurde bei Berücksichtigung der ersten sieben Kandidaten erzielt. Bei der ersten Fallstudie muss zum einen angemerkt werden, dass die Bedienungsanleitung mit Hilfe der Quelltextklassen generiert wurde. Dadurch waren auch viele Bezeichner in der Bedienungsanleitung enthalten, was die Zuordnung begünstigt hat. Andererseits hatten viele Quelltextklassen keine korrespondierende Bedienungsanleitung, was die Präzision wiederum negativ beeinflusste.

Eine Erweiterung des IR-Vektorraummodells wird in **Recovering documentation-to-source-code traceability links using latent semantic indexing** [MM03] vorgestellt. Mit Hilfe der latenten semantischen Indizierung (engl. latent semantic indexing, LSI) werden die Vektoren in einen Vektorraum mit niedrigerer Dimension transformiert, in der die resultierenden Vektoren keine Wörter, sondern Linearkombinationen von Wörtern als Dimensionen haben. Dadurch erhofft man sich einen besseren Umgang mit Synonymen als bei einfachen IR-Vektorraummodellen (in der überhaupt keine Synonyme berücksichtigt werden). Zusätzlich werden für die Quelltextklassen nicht nur Bezeichner, sondern auch Kommentare miteinbezogen.
Diese Erweiterung wurde auf den gleichen zwei Projekten wie bei Antoniol et al. evaluiert. In der ersten Fallstudie wurden 77% Präzision/60% Ausbeute mit einem Kandidaten und 12% Präzision/100% Ausbeute mit elf Kandidaten erreicht. Die Ergebnisse der zweiten Fallstudie betrugen 45% Präzision/46% Ausbeute mit einem Kandidaten und 16% Präzision/100% Ausbeute mit sechs Kandidaten. Die Resultate zeigen eine bessere Präzision mit LSI für die erste Fallstudie auf. Die Werte bei der zweiten Fallstudie sind dagegen fast identisch.

Im einfachen IR-Vektorraummodell besteht das Defizit, dass Synonyme als zwei unabhängige Wörter betrachtet werden, obwohl sie das Gleiche bedeuten. Mit Hilfe eines Thesaurus kann eine bessere Synonymunterstützung umgesetzt werden. Dies wurde in **Helping analysts trace requirements: an objective look** [HDSH04] durchgeführt; der Thesaurus wurde hier aus Wörtern des Glossars und anderer Anhänge der Anforderungsdokumente zusammengestellt. Das Vorgehen ist zunächst identisch wie beim Vektorraummodell von Antoniol et al. [ACCL00], nur bei der Berechnung der Kosinusähnlichkeit zwischen Dokumenten- und Anfragevektor werden zusätzliche Ähnlichkeitskoeffizienten zwischen ihren enthaltenen Wörtern berechnet und addiert. Die Ähnlichkeitskoeffizienten ordnen jedem Wortpaar aus dem Thesaurus einen Wert zwischen Null und Eins zu, der repräsentieren soll, wie ähnlich bedeutend die Wörter sind. Diese Koeffizienten werden in der Arbeit manuell durch einen Entwickler festgelegt. Dadurch ist diese Ähnlichkeitsmetrik stark subjektiv und je nach Größe des Thesaurus arbeitsintensiv. Weiterhin wurde das System so entworfen, dass eine interaktive Rückmeldung durch den Nutzer integriert ist. In der Kandidatenliste mar-

kiert der Nutzer die ersten N Einträge als wahr oder falsch Anhand der Markierungen wird schließlich der Anfragevektor angepasst.
Das Verfahren wurde in einer Rückverfolgbarkeit zwischen abstrakteren und implementierungsnäheren Anforderungen (beide in natürlicher Sprache) evaluiert, wodurch sich die Ergebnisse nicht direkt mit den obigen Arbeiten vergleichen lassen. Die Experimente beinhalten mehrere Iterationen, in jeder Iteration werden die Anfragevektoren durch Nutzerrückmeldung angepasst. Iteration Null entspricht dabei die Generierung einer Kandidatenliste ohne Nutzerrückmeldung, dies lässt sich also am ehesten mit den obigen Ansätzen vergleichen. Bei einer Kandidatenliste mit 100% Ausbeute erreicht das Thesaurusverfahren 12% Präzision. Im Vergleich dazu erzielt das einfache IR-Vektorraummodell, das auf dem gleichen Datensatz getestet wurde, 11% Präzision bei 100% Ausbeute. Durch Integration der Nutzerrückmeldung und weiterer Filterung der Kandidatenliste (zum Beispiel Filterung nach Schwellwert) konnte die Präzision bis auf 74% gesteigert werden.

Ein anderer, wahrscheinlichkeitsbasierter Ansatz wird in **A Traceability Technique for Specifications** [ANS08] präsentiert. Dort werden die Anforderungsdokumente und Quelltextklassen als Wahrscheinlichkeitsverteilungen modelliert (Verteilung der Auftrittswahrscheinlichkeit eines Wortes im Dokument/in der Klasse). Die Wörter einer Klasse sind dabei seine Bezeichner. Alle Dokumente und Klassen durchlaufen eine Vorverarbeitung (Transformation in Kleinbuchstaben, Stoppwortentfernung, Stemming, Trennung von Wortzusammensetzungen bei Bezeichner). Mit Hilfe der Jensen-Shannon-Ähnlichkeitsmetrik [CT06] wird die Distanz zwischen den Wahrscheinlichkeitsverteilungen der Dokumente gemessen. Die Arbeit beschreibt außerdem einige Varianten dieser Ähnlichkeitsmetrik, wo zum Beispiel zusätzlich die globale Auftrittshäufigkeit von Wörtern als Gewicht mitberücksichtigt wird.
Die Evaluation wurde auf zwei Datensätzen durchgeführt. Zum Vergleich wurden andere Techniken wie das einfache IR-Vektorraummodell auf den gleichen Datensätzen angewendet. Mit den fünf ersten Kandidaten erreicht die beste Jensen-Shannon-Variante auf dem ersten Datensatz 46% Präzision, das einfache IR-Vektorraummodell erzielt 43% Präzision. Mit 20 Kandidaten kommen Jensen-Shannon-Techniken auf bis zu 24% Präzision und 57% Ausbeute, während das einfache IR-Vektorraummodell 20% Präzision und 51% Ausbeute erreichen. Im zweite Datensatz sind die Anforderungen implementierungsnäher beschrieben. Daher sind alle Werte generell höher als beim ersten Datensatz. Das einfache IR-Vektorraummodell war dabei fast durchgehend die beste Technik. Die Präzision bei fünf Kandidaten betrug 80% (bis zu 70% bei Jensen-Shannon). Bei 20 Kandidaten sank die Präzision auf 50% (bis zu 49% bei Jensen-Shannon) mit 68% Ausbeute (bis zu 61% bei Jensen-Shannon). Zusammengefasst ist das Jensen-Shannon-Modell von den Ergebnissen her mit dem einfachen IR-Vektorraummodelle ähnlich.

In **Towards Feature-Aware Retrieval of Refinement Traces** [RMK13] wird das IR-Vektorraummodell um Artefaktbündel (engl. artifact cluster) erweitert. Wie im einfachen IR-Vektorraummodell werden alle Artefakte als Erstes auf Vektoren abgebildet. Zusätzlich werden aus den Artefakten der gleichen Abstraktionsebene jeweils ein Graph aufgebaut (zum Beispiel ein Anforderungsgraph und ein Quelltextgraph), in der die Artefakte die Knoten sind und gewichtete Kanten zwischen den Knoten existieren, wobei das Gewicht die Ähnlichkeit der verbundenen Artefakte repräsentiert. Das Gewicht entspricht der Kosinusähnlichkeit der verbundenen Artefakte. Die Knoten in beiden Graphen werden jeweils anhand eines Bündelungsverfahrens zu Bündeln gruppiert. Für die Verbindung von Artefakten werden nicht nur die Ähnlichkeiten ihrer Artefaktvektoren, sondern auch die Ähnlichkeit ihrer Bündel, zu denen sie gehören, berücksichtigt.
In der Evaluation wurde diese Methode mit dem einfachen Vektorraummodell und dem LSI-Vektorraummodell an drei Datensätzen verglichen; getestet wurde die Rückverfolgbarkeit nur zwischen natürlichsprachigen Artefakten. Beim ersten Datensatz erzielte LSI die

höchste durchschnittliche Präzision mit 30% (26% mit Bündel). Auf den anderen beiden Datensätzen lagen die Bündelmethode und das einfache IR-Vektorraummodell bezüglich der durchschnittlichen Präzision ungefähr gleichauf bei 43% bzw. 47% (LSI: In beiden Fällen weniger als 20%) Die unterschiedlichen Präzisionswerte zwischen den Datensätzen lässt sich durch die Unterschiedlichkeit der Datensätze erklären. Beispielsweise wurde in einem Datensatz zwischen Anwendungsfällen und natürlichsprachigen Testfällen abgebildet, während in einem anderen Anwendungsfälle zu abstrakten Anforderungen zugeordnet wurden.

Kuang et al. haben in ihrer Arbeit **Can Method Data Dependencies Support the Assessment of Traceability Between Requirements and Source Code?** [KMH+15] anhand von gegebenen Rückverfolgbarkeitsverbindungen weitere Kandidaten gefunden, indem sie Kontroll- und Datenflüsse zu anderen Methoden verfolgt haben. Die Präzision an korrekten Kandidaten lag bei 79-96% mit einer Ausbeute von 54-96%.

In allen Verfahren aus der Informationsrückgewinnung bis auf die von Kuang et al. werden die Quelltextklassen eigentlich als „Text“ bestehend aus ihren Bezeichnern und Kommentaren betrachtet - quelltextspezifische Merkmale wie Kontroll- und Datenflüsse werden außen vor gelassen. Außerdem basiert beim Vektorraummodell die Ähnlichkeit auf dem Vorkommen des gleichen Worts in beiden Vektoren - das führt dazu, dass die Qualität der Benamung von Klassen und Variablen einen starken Einfluss auf das Ergebnis hat.

Durch die niedrige Präzision haben die bisherigen IR-Techniken die Gemeinsamkeit, dass mindestens die Analyse der Kandidatenliste ein manuelles Eingreifen durch den Entwickler erfordert, um das praktische Einsetzen der Rückverfolgbarkeitsverbindungen zu ermöglichen. Dadurch sind die Techniken höchstens teilautomatisiert.

3.2 Rückverfolgbarkeit mit Einbettungen

Neben Rückverfolgbarkeitsansätzen durch Informationsrückgewinnung gibt es auch existierende Arbeiten, die Einbettungen verwenden.

Eine dieser Arbeiten ist **Semantically Enhanced Software Traceability Using Deep Learning Techniques** [GCC17] von Guo et al., in der eine Abbildung zwischen Anforderungs- und Entwurfsdokumenten (beide in natürlicher Sprache) erarbeitet wurde.

Wie in Abbildung 3.1 zu sehen, werden zunächst für alle Wörter in einem Anforderungs- bzw. Entwurfsdokument Worteinbettungen mit dem Skip-Gram-Modell von Word2Vec (siehe Abschnitt 2.6.4.1) berechnet; alle Worteinbettungen eines einzelnen Dokuments werden anschließend in ein RNN (siehe Abschnitt 2.5.1) eingegeben, welches eine Einbettung für das ganze Dokument berechnet. Danach werden die Richtungen der beiden Dokumentvektoren durch elementweise Multiplikation verglichen und ihre Distanz durch Subtraktion berechnet; beide berechneten Vektoren werden mit einer Feedforward-Schicht mit Sigmoidaktivierungsfunktion verarbeitet und verschmolzen. Mit dem verschmolzenen Vektor als Eingabe wird in der Ausgabeschicht schließlich eine Wahrscheinlichkeit berechnet, ob zwischen den beiden Dokumenten(-vektoren) eine Rückverfolgbarkeitsverbindung besteht oder nicht.
Das neuronale Netz wurde auf einen großen Industriedatensatz mit über 1500 echten Rückverfolgbarkeitsverbindungen angewendet, wobei die Hälfte zum Training bzw. zum Testen verwendet wurde. Zur Evaluierung wurde jeweils ein Quelldokument ausgewählt, das paarweise mit allen möglichen Zieldokumenten durch das Netz verarbeitet wird. Aus den jeweiligen Ergebnissen wird pro Quelldokument eine sortierte Liste an Zieldokumentkandidaten zusammengestellt, auf der wiederum Präzision und Ausbeute berechnet wird. Guo et al. haben auf ihren Datensatz zum Vergleich auch LSI und das einfache IR-Vektorraummodell angewendet und festgestellt, dass sie mit Einbettungen bei größerer Ausbeute eine höhere

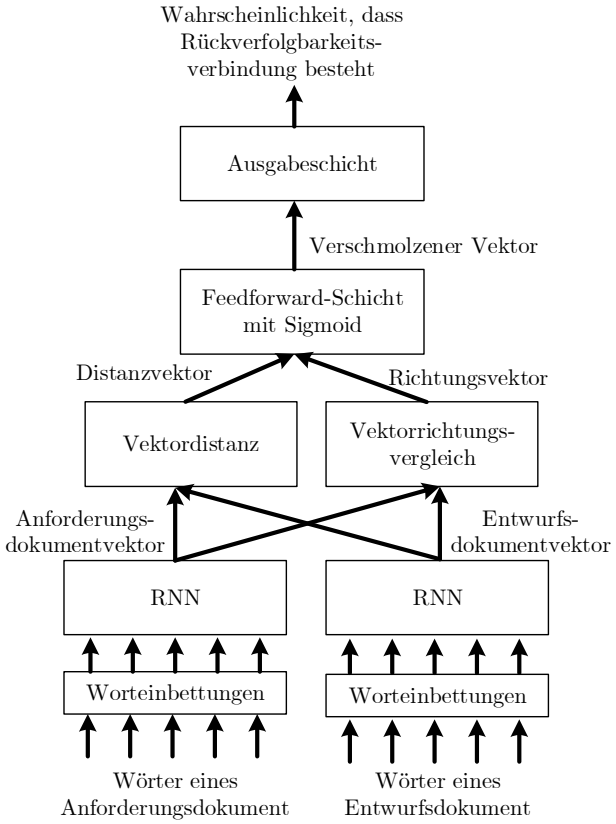


Abbildung 3.1: Aufbau der Netzarchitektur nach [GCC17].

Präzision erzielen (Bei 100% Ausbeute: 59,8% durchschnittliche Präzision mit Einbettungen, 42,3% bei Vektorraummodellen, 45,1% bei LSI).

Csuvik et al. haben mit **Source Code Level Word Embeddings in Aiding Semantic Test-to-Code Traceability** [CKV19] die Rückverfolgbarkeit zwischen Test- und Implementierungsklassen untersucht, indem sie Dokumenteinbettungen aus den Klassen erzeugt und verglichen haben. Dafür haben sie drei Varianten ausgearbeitet, wie die Dokumenteneinbettungen kreiert werden: Die erste Variante entfernt alle Syntaxsonderzeichen und fasst eine Klasse als einen Satz auf, aus dem eine Dokumenteneinbettung generiert wird. Die zweite Variante baut den abstrakten Syntaxbaum der Klasse auf und fasst alle Knotentypen darin als „Wörter“ auf, mit dem die Dokumenteneinbettung erzeugt wird. In der letzten Variante werden die Dokumenteneinbettungen nur aus den Bezeichnern kreiert. Die Wörter einer Test- oder Implementierungsklasse wurden mit Doc2Vec (siehe Abschnitt 2.6.4.6) verarbeitet, um die Dokumenteneinbettungen zu generieren. Die Ähnlichkeit wird mit einer Metrik bestimmt, die auf der Kosinusähnlichkeit basiert.
In der Evaluation wurden die drei Varianten miteinander verglichen, wobei die letzte Variante, die auf den Bezeichnern aufbaut, am präzisesten war. Im Vergleich zu einer Zuordnung durch LSI hat sich ergeben, dass das Verfahren mit Einbettungen besser war (Im Durchschnitt 30% Präzision mit LSI, 60% mit Einbettungen). Die Ausbeute wurde nicht evaluiert.

Die Arbeit **Mapping Bug Reports to Relevant Source Code Files Based on the Vector Space Model and Word Embedding** [LLS+19] von Liu et al. verbindet Fehlerberichte mit betreffenden Quelltextklassen, dabei werden Techniken aus der Informationsrückgewinnung mit Einbettungen kombiniert. Jeder Fehlerbericht und jede Klasse wird auf einen Vektor in einem IR-Vektorraummodell abgebildet (siehe Abschnitt 2.6.3). Für die Quelltextdateien werden dabei die Bezeichner als „Text“ erachtet. Für die Vektoren wird anschließend

paarweise die Kosinusähnlichkeit berechnet. Parallel dazu werden aus den Fehlerberichten und Klassen Einbettungen für alle enthaltenen Wörter mit Word2Vec oder GloVe (siehe Abschnitt 2.6.4.1 und Abschnitt 2.6.4.3) generiert (auch hier Bezeichner als Wörter einer Klasse). Eine Klasse und ein Fehlerbericht werden verglichen, indem ihre Worteinbettungen einzeln mit einer Ähnlichkeitsmetrik verglichen und in Abhängigkeit der Wortauftrittshäufigkeit aufsummiert werden. Die Ähnlichkeit aus dem IR-Vektorraummodell und die von den Einbettungen werden schließlich gewichtet addiert, um eine Gesamtähnlichkeit zu berechnen. Die Gewichte wurden empirisch aus Experimenten bestimmt. Der Ansatz wurde mit anderen Informationsrückgewinnungstechniken wie LSI verglichen. Daraus ergab sich, dass das Modell von Liu et al. präziser als Standardmethoden aus der Informationsrückgewinnung funktioniert: Bei Betrachtung der ersten fünf Kandidaten war die richtige Zuordnung je nach Datensatz in 50-70% der Fälle enthalten (LSI: 10-40%). Jedoch lag es mit einem erweiterten IR-Vektorraummodell (ohne Einbettungen), der vergangene Fehlerberichte berücksichtigt, ungefähr gleichauf (weniger als 5% Unterschied).

Zusammenfassend lässt sich sagen, dass im Bereich der Rückverfolgbarkeit mit Einbettungen Quelltextklassen bisher als Fließtext behandelt wurden. Quelltextspezifische Merkmale werden nicht berücksichtigt. Dennoch scheint die Nutzung solcher Bezeichnereinbettungen etwas bessere Ergebnisse zu erzielen als die reine Anwendung von Methoden der Informationsrückgewinnung.

3.3 Abbildung zwischen Vektorräumen

Wenn Worteinbettungen für Anforderungen und Quelltexteinbettungen wie die aus inst2vec oder IR2Vec (siehe Abschnitt 2.7.2) für Quelltext verwendet werden, werden die Artefakte auf zwei verschiedene Vektorräume abgebildet: Der Einbettungsraum der natürlichen Sprache und des Quelltextes. Dadurch können nicht mehr direkte Metriken wie die Kosinusähnlichkeit verwendet werden, um die Ähnlichkeit zwischen Anforderungs- und Quelltexteinbettungen zu messen. Vielmehr muss nun eine Abbildung zwischen den Vektorräumen gefunden werden. Im Bereich der Übersetzung von natürlicher Sprache gibt es dazu verwandte Arbeiten, in denen die Wörter von Quell- und Zielsprache auf Einbettungen (von verschiedenen Vektorräumen) abgebildet werden und für die Abbildung zwischen ihnen eine Transformationsmatrix durch maschinelles Lernen trainiert wird. Einbettungen aus der Quellsprache multipliziert mit der Transformationsmatrix sollen also die Einbettung der passenden Übersetzung aus der Zielsprache ergeben. Die Anforderung-zu-Quelltext-Rückverfolgbarkeit kann auch als eine Art Übersetzung zwischen der natürlichen Sprache und der Programmiersprache angesehen werden.

Für den Rest dieses Abschnitts sei X die Menge der Worteinbettungen der Quellsprache, Y die Menge der Worteinbettungen der Zielsprache und T die gesuchte Transformationsmatrix, die die passenden Übersetzungen $Tx = y$ mit $x \in X, y \in Y$ berechnen soll.

Aldarmaki et al. haben in **Unsupervised Word Mapping Using Structural Similarities in Monolingual Embedding** [AMD18] ein unüberwachtes Verfahren vorgestellt, mit dem sie zunächst eine Abbildungsfunktion M von Einbettungen der Quellsprache zu Einbettungen der Zielsprache trainieren. Die initialen Abbildungen dieser Funktion werden folgendermaßen bestimmt: Alle Worteinbettungen $x \in X$ und $y \in Y$ werden auf Spektraleinbettungen abgebildet. Die Spektraleinbettungen werden durch eine Eigenwertzerlegung der Adjazenzmatrix (enthält die Gaußsche Ähnlichkeit zu k nächste Nachbarn) der Worteinbettungen berechnet und sollen die lokalen Strukturen zwischen den Worteinbettungen im Vektorraum kodieren. Die Annahme besteht hierin, dass $x \in X$ und $y \in Y$ ähnliche Strukturen (Distanzen) zu ihren Nachbarn haben, wenn y die Übersetzung von x ist. In Abbildung 3.2 sieht man zum Beispiel, dass „Table“ und „Tisch“ im Raum der Spektraleinbettungen nah

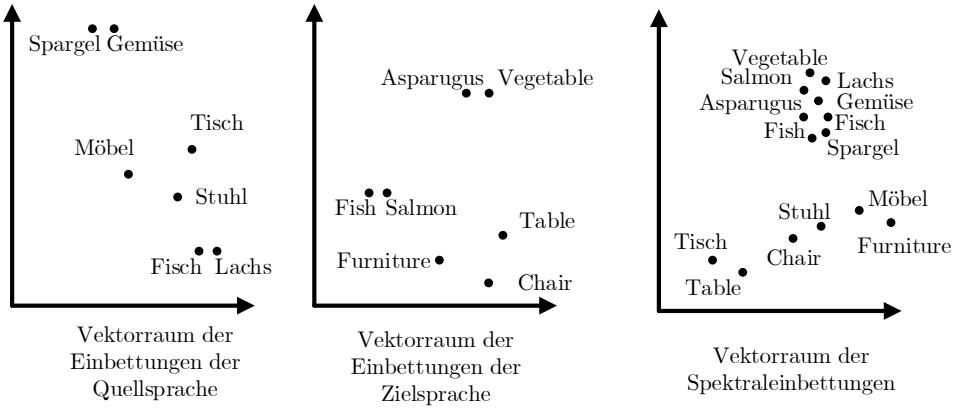


Abbildung 3.2: Schematische Veranschaulichung des Zusammenhangs von Worteinbettungen und Spektraleinbettungen nach [AMD18]. Die Idee liegt darin, dass Worteinbettungen mit ähnlichen Nachbarschaftsstrukturen im Spektralraum nah beieinander liegen.

beieinander liegen, da die jeweiligen Worteinbettungen ähnliche Distanzen zu ihren Nachbarn haben. Der nächste Nachbar aus der Zielsprache im Spektralraum wird als initiale Übersetzung in der Abbildungsfunktion M festgelegt. Die Spektraleinbettungen kodieren nur lokale Distanzen, nicht globale. Dadurch liegen „Fish“ und „Spargel“ im Spektralraum wegen ähnlicher Distanzen zu lokalen Nachbarn nah beieinander, obwohl die beiden Wörter global als Worteinbettungen weit weg voneinander liegen. Um die globalen Strukturen zu berücksichtigen, wird M durch einen iterativen Algorithmus optimiert. Dieser versucht, eine globale Kostenfunktion (Gleichung 3.1) zu minimieren.

$$Cost = \sum_{p,q}(EuklidDist(x_p, x_q) - EuklidDist(M(x_p), M(x_q)))^2; x_q, x_q \in X \tag{3.1}$$

Das heißt die Kosten sind am niedrigsten, wenn die euklidische Distanz zweier Einbettungen aus X und die ihrer Übersetzungen durch M möglichst gleich sind. Mit Hilfe eines neuronalen Netzes wird dann die Transformationsmatrix T trainiert, wobei M als „Musterlösung“ für die Übersetzungen fungiert.
In der Evaluation hat sich ergeben, dass die initialen Spektraleinbettungen meist keine validen Übersetzungen liefern, aber zumindest auf Wörter abbilden, die mit der richtigen Übersetzung semantisch ähnlich sind. Durch die iterative Optimierung mit der globalen Kostenfunktion hat sich auch die Präzision verbessert. In den Experimenten wurde auch ein überwachtes Training mit einem Wörterbuch sowie die Optimierung der globalen Kostenfunktion ohne initiale Spektraleinbettungen (stattdessen zufällige Initialisierung) durchgeführt. Bei Betrachtung der ersten 20 Kandidaten erzielte Aldarmakis Technik je nach Datensatz 70-90% Präzision, die überwachte Methode war bis zu 10% präziser. Die Variante mit der zufälligen Initialisierung war dagegen deutlich unpräziser (bei 10% auf allen Datensätzen) als die anderen beiden, das bedeutet, dass die Qualität der Startwerte einen großen Einfluss auf die Übersetzung hat. Die initialen Spektraleinbettungen wurden mit Hilfe von je 2000 Wörter aus der Quell- und Zielsprache gebildet.

Eine Möglichkeit, T ohne große Korpora zu lernen, wird von Artetxe et al. in **Learning bilingual word embeddings with (almost) no bilingual data** [ALA17] beschrieben. Dort wird mit einem kleinen Startwörterbuch mit nur 25 Übersetzungswortpaaren begonnen, das iterativ mit jedem Durchlauf neue Wörter integriert. Mit dem Startwörterbuch wird

durch Gleichung 3.2 die initiale Transformationsmatrix T berechnet.

$$\arg\min_T \sum_i \sum_j D_{ij} ||\hat{X}_i T - \hat{Y}_j||^2 \tag{3.2}$$

$\hat{X}$ und $\hat{Y}$ sind Matrizen, deren Zeilen i bzw. j aus den (je 25) Worteinbettungen aus X bzw. Y bestehen. D repräsentiert das Startwörterbuch und ist eine binäre Matrix, in der $D_{ij} = 1$ ist, falls $\hat{Y}_j$ die Übersetzung von $\hat{X}_i$ ist. Mit der Gleichung wird T also so berechnet, dass die euklidische Distanz zwischen der richtigen Übersetzung $\hat{Y}_j$ und der aktuellen Übersetzung $\hat{X}_i T$ minimiert wird. Durch die Faktorisierung mit D_{ij} werden nur die euklidischen Distanzen zwischen Übersetzungswortpaaren berücksichtigt (wenn $\hat{Y}_j$ keine Übersetzung von $\hat{X}_i$ ist, ist $D_{ij} = 0$). Nach der Bestimmung von T werden neue (und alte) Worteinbettungen aus $\hat{X}$ mit T multipliziert und der jeweilige nächste Nachbar in $\hat{Y}$ als Übersetzung festgelegt (das heißt das entsprechende D_{ij} wird auf Eins gesetzt). So erhält man ein neues Wörterbuch mit mehr Einträgen und kann wieder iterativ eine neue Transformationsmatrix T mit Gleichung 3.2 berechnen.
Für die Evaluierung wurde die Methode von Artetxe et al. mit anderen Arbeiten verglichen, die für die Arbeit mit größeren Wörterbüchern konzipiert wurden. Daher wurden alle Methoden sowohl mit 25 Wörtern als auch mit 5000 Wörtern getestet. Nicht überraschend liegt bei der 25-Wörter-Variante die Präzision bezüglich der Anzahl der richtigen Übersetzungen bei den anderen Methoden bei 0% -1%, während Artetxes Vorgehen 30% - 40% Präzision erzielen konnte. Beim Test mit 5000 Wörtern konnte Artetxes Methode jedoch ebenfalls ähnlich hohe Präzisionen (30% - 40%) wie die anderen Methoden erreichen (Unterschied liegt bei 1% - 2%). Aus der Evaluation hat sich weiterhin ergeben, dass man gute Übersetzungswortpaare für das Startwörterbuch wählen muss, da sonst die Optimierung in lokalen Optima hängen bleibt. Jedoch wurde nicht definiert, was hier „gut“ heißt. Wie die Autoren selber anmerken, wird ihre Methode beispielsweise durch das Neuberechnen des Wörterbuchs D (welches in jeder Iteration größer wird) sehr zeitintensiv. Daher ist hier eine effiziente Berechnung notwendig; hierfür wird in der Arbeit zum Beispiel Gleichung 3.2 nicht direkt berechnet, sondern durch Singulärwertzerlegung in eine einfachere Matrixmultiplikation umgeformt.

Die Arbeit von Conneau et al., **Word Translation Without Parallel Data** [CLR+18], geht noch einen Schritt weiter und benötigt überhaupt kein Startwörterbuch, stattdessen wird T mit einem gegnerischen Netzwerk (engl. adversarial network) trainiert. Dieses Netzwerk besteht aus zwei konkurrierenden neuronalen Netzen, die Generator und Diskriminator genannt werden. Der Generator hat das Ziel, ein $z = Tx, x \in X$ zu erzeugen, sodass z nicht von $y \in Y$ unterscheidbar ist. Der Diskriminator ist ein Klassifikator, der als Eingabe entweder ein z oder ein echtes y erhält und unterscheiden muss, ob die Eingabe nun echt ist oder nicht. Da bekannt ist, ob ein z oder y eingegeben wurde, können sich sowohl Generator als auch Diskriminator anhand einer Fehlerfunktion anpassen. Dadurch lernt der Generator eine Transformationsmatrix zu erzeugen, sodass z wie eine echte Übersetzung aussieht. Die gelernte Transformationsmatrix T wird danach in einem Verfeinerungsprozess verwendet, eine bessere Transformationsmatrix T^* zu lernen. Dazu wird T benutzt, um ein synthetisches Wörterbuch mit den häufigsten Wörtern zu bauen. Dieses Wörterbuch wird als Musterlösung verwendet, um eine bessere Transformationsmatrix T^* wie in Gleichung 3.3 überwacht zu lernen, wo die euklidische Distanz als Fehlermetrik verwendet wird.

$$T^* = \arg\min_T ||TX - Y||^2 \tag{3.3}$$

Mit T^* konnten die Autoren in den Experimenten bessere Präzisionen erzielen, als wenn sie nur T verwenden. Sie führen dies darauf zurück, dass das gegnerische Netzwerk versucht, alle Worteinbettungen aus beiden Vektorräumen unabhängig ihrer Auftrittshäufig-

keit aufeinander abzubilden; jedoch sind Worteinbettungen von seltenen Wörtern qualitativ „schlechter“, da sie weniger oft im Training gesehen wurden. Durch Weglassen der Abbildung zwischen „schlechten“ Einbettungen hat sich die Präzision verbessert.
Das gegnerische Netzwerk wurde anschließend mit einer Übersetzungsaufgabe evaluiert. Je nach Sprache ergaben sich (unter Nutzung von T^*) Präzisionen von bis zu 83%. Jedoch konnten diese hohe Präzisionen nur durch günstige Umstände erreicht werden, wie auch Artetxe et al. in [ALA18] beschreibt. Zum einen wurden diese Präzisionen auf Sprachen aus der gleichen Sprachfamilie erzielt, die inhärent ähnlichere Strukturen aufweisen. Zum anderen wurden die Worteinbettungen auf Wikipedia trainiert. Da Wikipediaartikel auch in unterschiedliche Sprachen strukturelle Gemeinsamkeiten aufweisen, konnte dadurch auch eine Präzisionssteigerung gewonnen werden. Conneau et al. haben Letzteres selber untersucht, mit dem Ergebnis, dass sie ungefähr zwei Prozent mehr Präzision erreichen, wenn die Einbettungen beider Sprachen auf dem Wikipediakorpus trainiert wurden.

In **Learning Unsupervised Word Mapping by Maximizing Mean Discrepancy** [YLW+18] von Yang et al. wird die Berechnung von T durch Minimierung der Maximalen Mittleren Diskrepanz (engl. Maximum Mean Discrepancy, kurz MMD) gelernt, welches ebenfalls ohne Startwörterbuch auskommt. Die MMD ist eine Metrik, die die Distanz zweier Verteilungen anhand der Distanz ihrer „Mittelpunkte“ angibt. Hier werden Y und $Tx, x \in X$ als Verteilungen angenommen und ihre Distanz in einem neuronalen Netz mit der MMD als Fehlerfunktion minimiert. Das Ganze wurde in der Evaluation auf sehr großen Korpora trainiert und erreicht Präzisionen zwischen 70% und 80%, je nach Sprachpaar.

Ein anderes Verfahren wird in **Gromov-Wasserstein Alignment of Word Embedding Spaces** [AJ18] von Alvarez at al. präsentiert. Hier werden X und Y ebenfalls als Verteilungen angenommen und die Abbildung zwischen ihnen als ein Problem des optimalen Transports formuliert, welches rechnerisch als Optimierungsproblem gelöst werden kann. Beim optimalen Transport geht es darum, eine Transportfunktion zu finden, die die Elemente einer Verteilung injektiv in die andere Verteilung transportiert (also abbildet), wobei die Kosten für den Transport minimiert werden soll. Die Transportkosten sind ein Distanzmaß der Vektoren zwischen den unterschiedlichen Vektorräumen; für die Anwendung auf den Worteinbettungsräumen wird hierfür die Gromov-Wasserstein-Distanz verwendet. Die Gromov-Wasserstein-Distanz ist eine Distanz von Distanzen - sie nimmt aus beiden Vektorräumen jeweils die Distanz zweier ähnlicher Vektoren und berechnet die Distanz zwischen diesen Distanzen. In der Evaluation hat sich gezeigt, dass diese Methode fast genauso präzise ist wie die Methode von Conneau et al., jedoch ist die Laufzeit kürzer, da kein neuronales Netz trainiert werden muss. Allerdings relativiert sich der Zeitaufwand bei großen Datenmengen, daher empfehlen die Autoren ihre Methode eher bei kleinen bis mittelgroßen Datensätzen.

Ein ähnlicher Anwendungsbereich von Abbildungen zwischen verschiedenen Vektorräumen findet sich in der Abbildung von gesprochener Sprache zu Text. Dies haben Chung et al. in **Unsupervised cross-modal alignment of speech and text embedding spaces** [CWTG18] erforscht. Worteinbettungen wurden mit fastText (siehe Abschnitt 2.6.4.2) trainiert und Spracheingaben mit Speech2Vec, welches ähnlich wie das Skip-Gram-Modell in Word2Vec (Abschnitt 2.6.4.1) trainiert wird, nur mit akustischen Merkmalen als Eingabe. Die Annahme ist hier, dass die Vektorräume der Sprach- und Worteinbettungen gewisse Ähnlichkeiten haben sollten, da sowohl Speech2Vec als auch fastText auf das Skip-Gram-Modell beruhen. Die Abbildung zwischen den Vektorräumen wird wie bei Conneau et al. [CLR+18] mit einem gegnerischen Netzwerk trainiert. Die daraus resultierende Transformationsmatrix T wird danach ebenfalls dafür verwendet, die verbesserte Transformationsmatrix T^* zu lernen.
In der Evaluation konnten mit dieser Methode Präzisionen von ungefähr 18% bis 25% erzielt werden. In einer weiteren Untersuchung dieser niedrigen Präzisionen haben Chung

et al. herausgefunden, dass das gesprochene Wort oft nicht auf das genaue textuelle Wort, aber stattdessen auf eines ihrer Synonyme bzw. semantisch ähnlichen Wörter (erhalten durch Kosinusähnlichkeit) abgebildet wurde. Deklariert man diese Synonyme als richtige Abbildungen, erhält man Präzisionen von 32% - 43% unter Betrachtung des Abbildungskandidaten mit der höchsten Wahrscheinlichkeit und 45% - 58%, wenn man jeweils die ersten fünf Wortkandidaten miteinbezieht.

4 Analyse und Entwurf

Mit der automatischen Anforderung-zu-Quelltextrückverfolgbarkeit sollen Verbindungen zwischen korrespondierenden Quelltextteilen und Anforderungen hergestellt werden. Anforderungen und Quelltext beschreiben eigentlich „das Gleiche“, nämlich die Funktionalitäten des zugrundeliegende System. Von daher korrespondieren diejenigen Quelltextteile und Anforderungen, die die gleiche Funktionalität beschreiben. Die Hauptschwierigkeit liegt hierbei in der Unterschiedlichkeit, wie die beiden Artefakte das System beschreiben. Anforderungen sind in natürlicher Sprache und Quelltext in einer Programmiersprache verfasst - dadurch sind sie bereits auf syntaktischer Ebene verschieden. Darüber hinaus befinden sich die beiden Artefakte auf unterschiedlichen Abstraktionsebenen: Quelltext ist auf der Implementierungsebene angesiedelt und enthält daher Implementierungsdetails, die in den abstrakteren Anforderungen gar nicht beschrieben sind. Folglich ist auch davon auszugehen, dass es keine bijektive Abbildung zwischen Quelltext und Anforderungen gibt. Eine Anforderung kann dadurch zum Beispiel mehrere korrespondierende Quelltextteile besitzen. Die Anzahl der Verbindungen wird auch durch die Granularität auf der Quelltextseite beeinflusst, das heißt ob man „Quelltextteil“ beispielsweise als Methode oder als Klasse versteht. Eine Klasse mit sechs Methoden hätte auf Methodenebene sechs Mal so viele Rückverfolgbarkeitsverbindungen wie auf Klassenebene.
Durch die Bezeichner (Klassennamen, Methodennamen, etc.) und Kommentare sind auch Elemente natürlicher Sprache im Quelltext enthalten. Dennoch unterscheiden sich diese Elemente von echtem natürlichsprachigen Text: Quelltextkommentare sind beispielsweise domänenspezifischer und weniger wortvielfältig [ED97] und Bezeichner beinhalten Wörter in Binnenmajuskelschreibweise.
Natürliche Sprache besitzt inhärent einige Eigenschaften, die die Zuordnung erschweren: Ein Wort kann mehrere Bedeutungen haben (Polysemie) und mehrere Wörter können auf die gleiche Bedeutung verweisen (Synonymie). Das heißt, dass Anforderung und Quelltextteil unterschiedliche Wörter verwenden können, um die gleiche Semantik zu beschreiben. Rückverfolgbarkeitstechniken, die auf bloßer Stichwortsuche und Vergleich von Zeichenketten basieren, haben dadurch große Präzisionseinbußen. Die Miteinbeziehung von Synonymie ergibt unter den semantischen Relationen dabei die verlässlichste Verbesserung der Rückverfolgbarkeit [MN15]. Welche Bedeutung eines Wortes gemeint ist kann nicht durch die isolierte Betrachtung des einzelnen Wortes bestimmt werden, daher reicht es nicht aus, Wörter nur als ihre bloße Zeichenkette zu repräsentieren - die Repräsentation des Wortes muss also den Kontext mitkodieren, da erst der Kontext die Informationen darüber liefert, welche Bedeutung gemeint sein könnte. Die Grundannahme ist hierbei, dass Wörter, die auf die gleiche Bedeutung verweisen, auch einen gleichen oder zumindest ähnlichen Kontext

besitzen. Wie in Abschnitt 2.6.4 erklärt, haben Worteinbettungen genau die Eigenschaft, dass Wörter mit ähnlichem Kontext auch auf ähnliche Vektoren (im Einbettungsvektorraum) abgebildet werden. Eine Anforderung-zu-Quelltextrückverfolgbarkeit basierend auf Anforderungs- und Quelltexteinbettungen verspricht folglich einen besseren Umgang mit Wortkontext und Wortbedeutungen und daher auch bessere Rückverfolgbarkeitsverbindungen als reine Stichwort- und Zeichenkettenverfahren.

4.1 Ziele

Das übergeordnete Ziel ist der Entwurf eines Verfahrens zur automatischen Anforderung-zu-Quelltext-Rückverfolgbarkeit durch Einsatz von Einbettungen. Dazu müssen folgende Schritte durchgeführt werden:

- **Entwurf von Anforderungseinbettungen**
  Es wird analysiert, welche Eigenschaften eine Anforderungseinbettung besitzen sollte und wie diese in Einbettungen integriert können. Dazu gehört die Betrachtung existierender Wort- und Dokumenteneinbettungsverfahren und die Analyse, wie und welche dieser Verfahren zur Generierung einer Anforderungseinbettungen verwendet werden können.
- **Entwurf von Quelltexteinbettungen**
  Wie bei den Anforderungen wird auf der Quelltextseite untersucht, welche Merkmale eine Quelltextrepräsentation haben sollte und wie man dies mit Einbettungen umsetzen kann. Insbesondere wird geprüft, ob quelltextspezifische Merkmale wie Kontroll- und Datenfluss dabei helfen können, die Rückverfolgbarkeit zu verbessern.
- **Erarbeitung eines Abbildungsverfahrens zwischen Quelltext- und Anforderungseinbettungen**
  Die Einbettungen von Anforderungen und Quelltextteilen sollen aufeinander abgebildet werden, sodass Rückverfolgbarkeitsverbindungen zwischen den Artefakten identifiziert werden können. In dieser Arbeit liegt der Fokus darauf, dass die Anforderungs- und Quelltexteinbettungen im gleichen Vektorraum liegen.

Die Analyse und der Entwurf von Anforderungseinbettungen wird in Abschnitt 4.2 erläutert. Analoges wird für die Quelltextseite in Abschnitt 4.3 diskutiert. Abschnitt 4.4 bespricht verschiedene Ansätze, um eine Abbildung zwischen Anforderungs- und Quelltexteinbettungen durchzuführen.

4.2 Analyse und Entwurf von Anforderungseinbettungen

Wie in der Einleitung zur Analyse bereits erklärt, soll die Rückverfolgbarkeit durch Einbettungen realisiert werden. Eine Anforderung ist in natürlicher Sprache verfasst und kann aus einem oder mehreren Sätzen bestehen. Wort- bzw. Satzeinbettungsverfahren bilden zwischen einzelnen Wörtern bzw. Sätzen ab. Damit lassen sich Anforderungen als Menge ihrer Wort- oder Satzeinbettungen repräsentieren. Man kann aber auch pro Anforderung eine einzige Einbettung berechnen, gegebenenfalls mit Hilfe von Wort- oder Satzeinbettungen, die zum Beispiel durch eine Funktion aggregiert werden. Es gibt also viele Möglichkeiten, wie Einbettungen für Anforderungen erzeugt werden können. Dazu werden zunächst die Eigenschaften von Einbettungen für Anforderungen diskutiert (Abschnitt 4.2.1). Verschiedene existierende Einbettungsverfahren werden darauffolgend verglichen (Abschnitt 4.2.2), bevor konkrete Verfahren für die Erzeugung von Anforderungseinbettungen vorgestellt werden (Abschnitt 4.2.3). In Abschnitt 4.2.4 werden Vorverarbeitungsschritte für die Anforderungsseite analysiert und in Abschnitt 4.2.5 befindet sich schließlich der Entwurf für Anforderungseinbettungen.

4.2.1 Eigenschaften von Anforderungseinbettungen

In diesem Abschnitt wird besprochen, welche Eigenschaften eine Anforderungseinbettung bzw. ihr Erzeugungsverfahren haben soll, um dadurch möglichst viele validen Rückverfolgbarkeitsverbindungen zu finden.

Repräsentation als kontinuierlicher Vektor
Die Rückverfolgbarkeit mit diskreten Vektoren wurde im Bereich der Informationsrückgewinnung bereits untersucht (siehe Abschnitt 3.1). Diskreten Vektoren wie zum Beispiel 1-aus-N-Vektoren haben alle den gleichen Abstand zueinander und kodieren deshalb keine semantische Ähnlichkeit durch die Distanz, wie es bei Einbettungen der Fall ist. Die Repräsentation als kontinuierlicher Vektor ist die wichtigste Eigenschaft von Einbettungen für die Anforderungsseite, da die Rückverfolgbarkeit mit Einbettungen erprobt werden soll und kontinuierliche Vektoren eine Ähnlichkeitskodierung ermöglichen.

Kodierung semantischer Ähnlichkeit
Die Ähnlichkeitseigenschaft ist ebenfalls sehr wichtig für die Anforderung-zu-Quelltextrückverfolgbarkeit, da zum einen die beiden Artefakte auf unterschiedlichen Abstraktionsebenen liegen und daher wahrscheinlich verschiedene Begriffe verwendet werden, um das Gleiche zu beschreiben. Zum anderen werden die Artefakte in der Praxis oft von unterschiedlichen Menschen erstellt, wodurch nochmals eine Begriffsvielfalt entsteht. Zu verbindende Quelltextteile und Anforderungen werden deshalb in den meisten Fällen nicht die exakt gleiche Formulierung oder Wortwahl besitzen, aber eine semantisch ähnliche, da sie das Gleiche beschreiben. Durch die Ähnlichkeitseigenschaft von Einbettungen soll dies ausgenutzt werden: Es ist hilfreich, wenn die Anforderungseinbettungen ebenfalls diese Ähnlichkeitsinformationen in sich tragen, damit die Artefakte besser einander zugeordnet werden können. Auf Wortebene sind sich zwei Wörter ähnlich, wenn sie zum Beispiel eine ähnliche Bedeutung haben. Für Anforderungseinbettungen muss dazu geklärt werden, was Ähnlichkeit zwischen Anforderungen bedeutet.

Beispiel 4.1: Semantische Ähnlichkeit von Anforderungen

(Anf. 1) In der Mensa können Studenten zu Mittag essen.
(Anf. 2) Die Cafeteria bietet in Vorlesungszeit ein Abendessen an.
(Anf. 3) Die KIT-Bibliothek kann mit der KIT-Karte betreten werden.

In Beispiel 4.1 sind drei Anforderungen aufgeführt. Ähnliche Elemente haben gewisse gemeinsame Eigenschaften. Mensa und Cafeteria sind im Gegensatz zur Bibliothek beides Gaststätten und in den ersten beiden Anforderungen geht es ums Essen. Intuitiv betrachtet sind die ersten beiden Anforderungen ähnlicher zueinander als zur dritten. Das liegt daran, dass Anforderung 1 und 2 ähnlichere Semantik (Mensa/Cafeteria vs. Bibliothek und Mittag/Abend essen vs. betreten) tragen. Die Ähnlichkeit gilt vor allem für Wortarten, die viel Semantik tragen wie etwa Nomen und Verben[CM98]. Andere Wortarten wie Artikel tragen weniger Semantik, da sie in sehr vielen Sätzen vorkommen. Sie erfüllen eher eine grammatikalische Funktion und nicht jede Sprache hat auch Artikel, deshalb sind sie weniger ausschlaggebend für die Ähnlichkeitsbetrachtung als zum Beispiel Nomen. Zwei Sätze, deren (semantikreiche) Wörter komplett verschiedene Semantik tragen, können auch auf Satzebene nicht plötzlich semantisch ähnlich werden - sie haben keine gemeinsame Eigenschaften. Konkret bedeutet dies, dass die Ähnlichkeit auf Satzebene von der Ähnlichkeit der Semantik auf der Wortebene abhängt. Im obigen Beispiel besteht jede Anforderung aus einem Satz, von daher sind hier die Ähnlichkeiten auf Satz- und Anforderungsebene äquivalent. Was ist, wenn eine Anforderung aus mehreren Sätzen besteht? Wenn sich zwei Anforderungen ähnlich sind, müssen sie teilweise gemeinsame Eigenschaften haben und da

eine Anforderung eine Menge von Sätzen ist, müssen die Sätze gemeinsame Eigenschaften besitzen. Die Ähnlichkeit auf Satzebene hängt wiederum von der Ähnlichkeit auf der Wortebene ab - das heißt, dass die Ähnlichkeit auf Anforderungsebene transitiv auf die von der Wortebene abhängt.

Polysemieunterstützung
Zur Berücksichtigung von Polysemie werden die verschiedenen Bedeutungen eines polysemen Wortes (bei kontextabhängigen Verfahren) auch auf verschiedene Einbettungen abgebildet. Auch Anforderungseinbettungen sollen Polysemie berücksichtigen. Eine Anforderung über eine Geldbank soll eine andere Einbettung besitzen als eine Anforderung über eine Sitzbank, da hier der semantische Inhalt komplett verschieden ist. Wird die Unterscheidung nicht gemacht, würde eine Einbettung von Geldbank zum Beispiel auch Ähnlichkeiten zu Wörtern wie Park oder Stuhl aufweisen, da diese Begriffe ähnlich zu „Sitzbank“ sind. Besser wären separate Einbettungen, damit die Bedeutungen getrennt werden und keine falschen Ähnlichkeitsbeziehungen entstehen. Polysemieunterstützung ist eine wichtige Eigenschaft, um solche Verfälschungen in der Ähnlichkeit zu reduzieren. Die Eigenschaft, dass es überhaupt eine Ähnlichkeitskodierung gibt, ist aber im Vergleich hierzu als noch wichtiger einzustufen. Die Polysemieeigenschaft auf der Anforderungsebene hängt genau wie bei der Ähnlichkeitseigenschaft wieder transitiv vom Umgang auf der Wortebene ab.

Behandlung von OOV-Wörtern
Bei trainierten Modellen mit neuronalen Netzen (und damit insbesondere Einbettungsverfahren) besteht allgemein das Problem, dass im Training nicht gesehene Daten später auch nicht verarbeitet werden können: Für unbekannte Eingaben wurden keine Einbettungen trainiert, daher werden sie einfach ignoriert oder sie bekommen einen Standardvektor wie dem Nullvektor als Einbettung zugewiesen. Ersteres führt offensichtlich zu Informationsverlust, aber auch Letzteres ist problematisch, da dadurch potentiell unbekannte, aber semantisch völlig verschiedene Eingaben auf die gleiche Standardeinbettung abgebildet werden können. Die OOV-Behandlung muss auch bei Anforderungseinbettungen bedacht werden, da insbesondere Anforderungen domänenspezifisch sind und damit Fachvokabular beinhalten, wohingegen viele existierende Worteinbettungsverfahren auf allgemeinen Textquellen trainiert wurden. Aus diesen Gründen ist eine Behandlungsmöglichkeit wünschenswert, die OOV-Wörter auf Einbettungen abbildet, wobei der Informationsgehalt dieser Einbettungen über die eines Standardvektors hinausgehen sollte. Die Einbettung sollte die Ähnlichkeitseigenschaft möglichst erhalten. Ob Polysemieunterstützung oder OOV-Behandlung wichtiger ist, kann nicht eindeutig festgelegt werden. Durch domänenspezifische Anforderungen werden sehr wahrscheinlich OOV-Wörter auftauchen, jedoch ist es genauso wahrscheinlich, dass bekannte Wörter aus dem allgemeinen Sprachgebrauch eine neue Bedeutung in der Domäne besitzen. Die beiden Eigenschaften sind gleich wichtig.

Bevorzugung von Nomen und Verben
In Anforderungen tragen Nomen und Verben mehr Informationen als andere Wortarten [CM98]. Wie oben erklärt, sind vor allem solche semantikreiche Wörter wichtig für die Ähnlichkeitseigenschaft. Von daher kann eine Bevorzugung dieser beiden Wortarten unter Umständen eine bessere Repräsentation erzeugen, da dadurch Wörter mit wenigen Informationen auch weniger ins Gewicht fallen. Allerdings muss beachtet werden, dass manche Wortarten wie Adjektive zwar weniger, aber auch nicht keine Informationen tragen. Präferiert man Nomen und Verben zu sehr, können unter Umständen zu viele Informationen der benachteiligten Wortarten verloren gehen. Es ist also schwer, die Gewichtungen der Wortarten so zu beziffern, sodass die Bevorzugung nicht kontraproduktiv wird. Diese Eigenschaft stellt daher nur eine optionale Optimierung dar und ist weniger wichtig als die anderen Eigenschaften.

Die bisher erläuterten Eigenschaften sind im Folgenden nochmal zusammengefasst und nach Wichtigkeit angeordnet. Die Reihenfolge von OOV-Behandlung und Polysemieunterstützung ist dabei wie oben erläutert miteinander vertauschbar.

Erwünschte Eigenschaften für Anforderungseinbettungen bzw. deren Verfahren

1. Repräsentation einer kompletten Anforderung als kontinuierlicher Vektor
2. Kodierung von semantischer Ähnlichkeit
3. Behandlungsmöglichkeit von OOV-Wörtern
4. Berücksichtigung von Polysemie
5. Bevorzugung von Nomen und Verben gegenüber andere Wortarten

4.2.2 Analyse existierender Worteinbettungsverfahren

Im vorherigen Abschnitt wurden Eigenschaften von Anforderungseinbettungen diskutiert. Diese hingen teilweise von den Eigenschaften auf der Wortebene ab. In diesem Abschnitt werden existierende Worteinbettungsverfahren analysiert und überprüft, inwieweit sie diese Eigenschaften erfüllen.

Die erste Arbeit zur Generierung von Einbettungen wurde von Bengio et al. vorgestellt (Abschnitt 2.6.4). Jedoch hat dieses Verfahren eine hohe Berechnungskomplexität, wodurch die Anwendung nicht praktikabel ist. Mikolov et al. haben mit Word2Vec (Abschnitt 2.6.4.1) eine vereinfachte und optimierte Variante von Bengios Verfahren erarbeitet, welches Worteinbettungen auch in der Praxis effizient berechnen kann. Jedes Wort besitzt ihre eigenen korrespondierenden Neurongewichte, daher repräsentieren die Einbettungen einzelne, ganze Wörter. Außerdem sind diese Einbettungen kontextunabhängig, was bedeutet, dass die verschiedenen Bedeutungen von polysemen Wörtern dieselbe Einbettung besitzen. Das liegt daran, dass das neuronale Netz Wörter in Form von 1-aus-N-kodierte Vektoren entgegennimmt. Die verschiedenen Bedeutungen von polysemen Wörtern haben den gleichen 1-aus-N-kodierten Vektor und folglich auch die gleiche Einbettung. Ein Vorteil dieser Kontextunabhängigkeit ist die direkte Verwendbarkeit der berechneten Worteinbettungen - das Trainingsmodell wird nach der Trainingsphase nicht mehr benötigt. Eine wichtige Eigenschaft von Worteinbettungen ist die Berücksichtigung von Ähnlichkeit zwischen Wörtern - doch was ist diese Ähnlichkeit bei Word2Vec-Vektoren? Einbettungen von ähnlichen Wörtern liegen im Vektorraum nah beieinander, das heißt die Vektoreinträge bzw. Neurongewichte sind sich numerisch ähnlich. Im Training werden Auftrittswahrscheinlichkeiten von Wörtern und ihrem Kontext vorhergesagt und mit dessen Hilfe die Neurongewichte angepasst. Der Kontext ist dabei ein lokales symmetrisches Fenster um ein Wort in einem Satz, wobei Positionsinformationen ignoriert werden. Daraus folgt, dass bei Word2Vec die Ähnlichkeit zwischen zwei Wörtern zunimmt, je mehr und je öfter die identischen Kontextwörter im Training gesehen wurden.
Ein Problem bei Word2Vec ist die Behandlung von OOV-Wörtern. Mit Word2Vec ist es nicht möglich, Einbettungen für solche Wörter zu generieren, da dafür keine korrespondierenden Neurongewichte trainiert wurden.

fastText (Abschnitt 2.6.4.2) erweitert Word2Vec, um mit OOV-Wörtern umgehen zu können. Wie in Abschnitt 2.6.4.2 beschrieben wird für jedes Wort die Summe der Einbettungen ihrer Buchstaben-N-Gramme berechnet. Bei OOV-Wörtern besteht daher die Chance, dass ihre Buchstaben-N-Gramme im Training vorkamen und dadurch die Berechnung einer Worteinbettung möglich ist. Jedoch existiert hier das Risiko, dass ein OOV-Wort aus Buchstaben-N-Grammen zusammengesetzt wird, die semantisch nichts mit dem OOV-Wort gemeinsam haben. Ist zum Beispiel „heiter“ ein OOV-Wort, könnte es aus unter anderem aus dem N-Gramm „heit“ zusammengebaut werden. „heit“ ist jedoch gleichzeitig

auch eine häufige Endung von Nomen wie „Krankheit“. Nicht alle Wörter mit „heit“-Endung haben einen semantischen Zusammenhang mit „heiter“, von daher verfälscht es die Einbettung von „heiter“, wenn es aus „heit“ zusammengesetzt wird.
Da fastText auf Word2Vec basiert, werden einige andere Word2Vec-Eigenschaften geerbt: Die fastText-Worteinbettungen sind kontextunabhängig, weil die verschiedenen Bedeutungen eines polysemen Wortes die gleichen Buchstaben-N-Gramme besitzen. Das Training verläuft bei fastText wie beim Skip-Gram-Modell in Word2Vec, also sind sich zwei Wörter ähnlich, wenn sie möglichst viele identische Buchstaben-N-Gramme in ihrem Kontext haben. „Viele identische Buchstaben-N-Gramme als Kontext“ korreliert dabei natürlich mit „viele identische Wörter als Kontext“. Durch Experimente hat sich außerdem gezeigt, dass fastText bei seltenen Wörtern und bei syntaktischen Aufgaben mit morphologisch vielfältigen Sprachen besser als Word2Vec funktioniert. Das ist auf die N-Gramm-Zerlegung zurückzuführen, da die N-Gramme öfter im Training vorkommen als das komplette Wort.

GloVe (Abschnitt 2.6.4.3) generiert für jedes Wort kontextunabhängige Worteinbettungen wie bei Word2Vec, allerdings unterscheidet sich die Ähnlichkeit zwischen Wörtern: Bei Word2Vec wird die Ähnlichkeit durch das Auftreten von identischen Kontextwörtern beeinflusst: Im symmetrischen Fenster um dem Wort müssen viele gleiche Kontextwörter auftreten, damit sich zwei Wörter ähnlich sind. Bei GloVe werden alle Auftrittshäufigkeiten zweier Wörter global auf einmal gezählt. Für die Auftrittswahrscheinlichkeiten, aus denen später die Worteinbettungen berechnet werden, wird die gemeinsame Auftrittshäufigkeit zweier Wörter zusätzlich ins Verhältnis zu Auftrittshäufigkeiten mit anderen Probewörtern gesetzt. Bei GloVe sind sich zwei Worteinbettungen ähnlich, wenn sie viele gemeinsame Kontextwörter besitzen (so wie bei Word2Vec) und darüber hinaus gleich selten oder häufig zusammen mit den anderen Probewörtern auftreten. Mit OOV-Wörtern kann GloVe genauso wie Word2Vec nicht umgehen, da diese Wörter im Training eine Auftrittswahrscheinlichkeit von Null haben und somit keine Repräsentation besitzen.

Die bisherigen Verfahren berücksichtigen keine polysemen Wörter. Der Grund liegt darin, dass die unterschiedlichen Bedeutungen eines polysemen Wortes im Training auf die gleiche Startrepräsentation (zum Beispiel als 1-aus-N-kodierter Vektor) abgebildet wurde. Bei ELMo (Abschnitt 2.6.4.4) wird die Kontextabhängigkeit dadurch erreicht, dass beim Training einer Worteinbettung der gesamte Satz als Eingabe für das neuronalen Netz entgegengenommen wird. Dabei unterscheiden sich zum einen je nach Bedeutung die anderen Wörter im Satz, zum anderen kann auch die Position der Wörter anders sein. Beispielsweise unterscheidet sich die Position von „überlegen“ je nachdem, ob das Verb oder das Adjektiv gemeint ist. Die Position wird in ELMo durch Einsatz eines mehrschichtigen LSTM-Netz berücksichtigt. Die verschiedenen Bedeutungen eines polysemen Wortes treten also mit jeweils anderen Kontextwörtern und Positionen im Satz auf. Diese Unterschiede werden bei ELMo berücksichtigt, da die Wörter nicht isoliert, sondern zusammen mit ihrem Satz als Eingabe entgegengenommen werden. Daher werden auch unterschiedliche Worteinbettungen generiert. Die kontextabhängigen Informationen werden dabei vor allem in den höheren Netzschichten eingefangen.
OOV-Wörter werden in ELMo unterstützt, da die Wörter als Summe ihrer Buchstabeneinbettungen repräsentiert werden.
Bezüglich der Ähnlichkeit liegen zwei Worteinbettungen im Vektorraum nah beieinander, wenn ihre zugehörigen Wörter möglichst viele Sätze mit möglichst gleichen Kontextwörtern besitzen, wobei die Positionen des Wortes und der Kontextwörter ebenfalls möglichst gleich sein sollten.

Genau wie bei ELMo wird bei BERT [DCLT19] der komplette Satz zur Berechnung einer Worteinbettung herangezogen. Da für die Verarbeitung kein rekurrentes Netz, sondern ein Transformer-Netz eingesetzt wird, werden Positionsinformationen in einem Vorverarbeitungsschritt gesondert hinzugefügt. Somit sind die Worteinbettungen von BERT analog

Tabelle 4.1: Zusammenfassung der Eigenschaften der existierenden Worteinbettungsverfahren

Verfahren	Repräsentation	Kontext-abhängig	OOV	Ähnlichkeit
Word2Vec	Wörter	Nein	Nein	Gleiche Wörter im lokalen Kontext
FastText	Buchstaben-N-Gramme	Nein	Ja	Gleiche Buchstaben-N-Gramme im lokalen Kontext
GloVe	Wörter	Nein	Nein	Gleiche Wörter im globalen Kontext
ELMo	Buchstaben-N-Gramme	Ja	Ja	Gleiche Buchstaben-N-Gramme im Satz inkl. Position
BERT	Subwörter	Ja	Ja	Gleiche Wörter im Satz inkl. Position und gleiche nächste Sätze

zu ELMo kontextabhängig und die Ähnlichkeit beruht auf dem Vorkommen der gleichen Wörter im Satz an der gleichen Position. Jedoch unterscheidet sich das Einbettungsverfahren: Bei ELMo wird das nächste Wort im Satz und parallel dazu das nächste Wort im umgedrehten Satz vorhergesagt. Es werden also zwei Modelle mit separaten Neurongewichten trainiert, die danach durch eine gewichtete Summe kombiniert werden. BERT maskiert dagegen eine Prozentzahl an Wörtern im Satz heraus, die vorhergesagt werden müssen. Diese maskierten Wörter können beliebig links oder rechts vom gegebenen Wort liegen. Laut den BERT-Autoren ist BERT durch diese echte Bidirektionalität mächtiger als ELMo, wo zwei verschiedene nicht-bidirektionale Modelle kombiniert werden. Weiterhin wurde BERT auch mit der Vorhersage des nächsten Satzes trainiert, dadurch steigt auch die Ähnlichkeit zweier Worteinbettungen, wenn sie die gleichen nächsten Sätze besitzen. Da BERT außerdem Einbettungen für Subwörter trainiert, werden OOV-Wörter unterstützt, sofern deren Subwörter im Training vorkamen. Bei ELMo wurde die OOV-Unterstützung mit Buchstaben-N-Grammen umgesetzt, wo das Problem bestand, dass OOV-Wörter eventuell aus semantisch nicht verwandten Buchstaben-N-Grammen zusammengesetzt werden können. Bei Bert wird dagegen ein Wort zum Beispiel in Wortstamm und Konjugationsendung aufgeteilt, wodurch bei OOV-Verben versucht wird, das Verb aus bekannten Wortstämmen und Konjugationsendungen zusammenzusetzen. Die Subwörter behalten also im Vergleich zu Buchstaben-N-Grammen teilweise semantische Informationen bei, was ein Vorteil für BERT darstellt.

Tabelle 4.1 fasst die bisherige Analyse der verschiedenen Worteinbettungsverfahren zusammen. In Anbetracht der gewünschten Eigenschaften von Anforderungseinbettungen erfüllen ELMo und BERT die meisten Eigenschaften, wobei BERT aufgrund der echten bidirektionalen Vorhersage und der Subwort-OOV-Behandlung ELMo überlegen ist. BERT und ELMo sollten laut ihren Autoren für bessere Ergebnisse mit Daten des konkreten Anwendungsfalls feinkalibriert werden, was wiederum Trainingsaufwand bedeutet. Daher bietet sich fastText als Alternative an, welches nicht zwingend feinkalibriert werden muss. Zwar ist fastText nicht mehr kontextabhängig, aber unterstützt im Vergleich zu Word2Vec und GloVe OOV-Wörter. Das ist insbesondere für etwaige domänenspezifische Fachbegriffe wichtig. Aus diesen Gründen sind BERT und fastText geeignete Kandidaten, die für ein Verfahren für Anforderungseinbettungen genutzt werden können.

4.2.3 Verfahren zur Erzeugung von Anforderungseinbettungen

Nachdem existierende Worteinbettungsverfahren analysiert wurden, wird im Folgenden eruiert, wie Einbettungen für ganze Anforderungen im Rahmen der Anforderung-zu-Quelltextrückverfolgbarkeit erzeugt werden können und wie man die existierenden Verfahren dafür nutzen kann.

4.2.3.1 Anforderungseinbettungen aus existierenden Verfahren

Mit einigen existierenden Einbettungsverfahren lassen sich direkt aus dem Text einer Anforderung eine Anforderungseinbettung erzeugen.

BERT (Abschnitt 2.6.4.5) kann Sätze entgegennehmen und daraus eine Satzeinbettung berechnen. Theoretisch ist es auch möglich, eine ganze Anforderung zu übergeben und die resultierende „Satz“-einbettung als Anforderungseinbettung zu definieren. Statt einem Satz wird hier also eine Anforderung als Kontext für die Einbettungserzeugung miteinbezogen. Durch BERT werden dadurch auch OOV und Polysemie behandelt. Ein Problem bei BERT ist die Eingabelängenbeschränkung: Zu lange Anforderungen müssen aus Performanzgründen hinten abgeschnitten werden. Die Qualität der Einbettungen hängen folglich davon ab, ob die Anforderungen die Eingabelängenbeschränkung überschreiten. Ein weiteres Problem liegt darin, dass BERT nicht dafür trainiert wurde, mehrere zusammenhängende Sätze einzubetten. Daher kann a priori nicht bestimmt werden wie gut dieses Vorgehen in der Praxis funktioniert.

Doc2Vec (Abschnitt 2.6.4.6) kann aus einem Text beliebiger Länge eine Dokumenten- bzw. Paragrafeneinbettung berechnen. Es wird jeweils ein Wortkontext aus dem Text und ein Paragrafentoken zur Vorhersage verwendet. Je nach Trainingsvariante muss der Wortkontext oder das nächste Wort vorhergesagt werden. Der Paragrafentoken lernt durch die Vorhersage, welche Wörter im Paragrafen vorhanden sind. Das heißt diese Token funktioniert wie ein weiteres Wort, dessen Wortkontext am Ende aus allen Wörtern des Dokuments besteht. Wenn man dieses Verfahren unverändert auf Anforderungen anwendet, entspricht die Einbettung des Paragrafentokens am Ende der Anforderungseinbettung. Allerdings sind die Eigenschaften der Anforderungseinbettungen direkt abhängig von Doc2Vec. Genau wie Word2Vec ist Doc2Vec zum Beispiel nicht kontextsensitiv und OOV-Wörter werden nicht unterstützt. Es ist aber denkbar, Doc2Vec diesbezüglich nach Vorbild anderer Worteinbettungsverfahren anzupassen: Wörter könnten zum Beispiel wie in ELMo in Buchstaben-N-Grammen zerlegt und durch zwei mehrschichtige LSTM-Netze gelernt werden, um OOV-Unterstützung und Kontextsensitivität zu erreichen. Es existiert außerdem kein vortrainiertes Doc2Vec-Modell, daher muss ein eigenes Modell trainiert werden. Dies wird im nächsten Abschnitt diskutiert.

4.2.3.2 Anforderungseinbettungen trainieren

Um Anforderungseinbettungen zu erzeugen, kann eine neuronales Netz dafür genutzt und trainiert werden.

Mit Doc2Vec (Abschnitt 2.6.4.6) kann ein Modell trainiert werden, welches aus einer ganzen Anforderung eine Anforderungseinbettung erzeugt. Das Trainieren eines eigenen Modells mit Anforderungstexten führt dazu, dass die Anforderungseinbettungen lernen, welche Wörter in den Anforderungen enthalten sind - also inklusive aller domänenspezifischen Wörter. Dadurch repräsentieren die resultierenden Einbettungen die Anforderungen besser als bei einem allgemeinen Modell, aber gleichzeitig sinkt die Übertragbarkeit des trainierten Modells auf andere Projekte, die diese domänenspezifischen Wörter nicht besitzen. Statt dem Training auf Softwareprojektdaten ist auch ein Training auf allgemeinen Datensätzen wie Wikipedia möglich. Hier besteht jedoch aufgrund der Polysemie von Wörtern

die Gefahr, dass in der jeweiligen Domäne Wörter eine andere Bedeutung als im allgemeinen Sprachgebrauch besitzen. Selbst wenn die Wörter kontextsensitiv gelernt werden, würde es eine Präferenz für die allgemeine Bedeutung geben, da diese im allgemeinen Trainingskorpus viel öfter enthalten ist. Als weitere Option können allgemeine und domänenspezifischen Daten gemischt werden oder ein existierendes allgemeines Modell kann mit domänenspezifischen Daten nachtrainiert werden. Hier ist jedoch das Mischungsverhältnis ausschlaggebend: Gibt es zum Beispiel viel weniger domänenspezifische Daten als allgemeine, erzielt das Nachtrainieren womöglich fast keine Wirkung. Es ist a priori nicht bestimmbar wie ein gutes Mischverhältnis zwischen den Datensätzen aussieht; dies müsste also empirisch ermittelt werden.
Beim Trainieren eines eigenen Doc2Vec-Modells wird die Ähnlichkeitseigenschaft erfüllt: In Abschnitt 4.2.1 wurde hergeleitet, dass die Ähnlichkeit auf der Anforderungsebene von der Ähnlichkeit auf der Wortebene abhängt. Beim Doc2Vec-Vorgehen werden die einzelnen Wörter einer Anforderung vorhergesagt. Deshalb resultieren ähnliche Anforderungen auch in ähnliche Einbettungen, da sie ähnliche Wörter beinhalten. Die OOV-Behandlung und die Polysemieeigenschaft sind nicht erfüllt; dafür müsste zum Beispiel eine Erweiterung wie bei ELMo vorgenommen werden, was in Abschnitt 4.2.3.1 diskutiert wurde. Bezüglich Gewichtungen wie etwa nach Wortart lässt sich das obige Vorgehen nur indirekt beeinflussen, indem zum Beispiel bestimmte Wörter in der Vorverarbeitung herausgefiltert werden, sodass die übrigen Wörter mehr Gewicht tragen. Dadurch ist die Gewichtung nicht feingranular konfigurierbar, da Wörter entweder ganz oder gar nicht enthalten sind - es gibt keine Abstufung dazwischen.

Doc2Vec sagt Wörter in einem Text voraus. Eine Variationsmöglichkeit ist die Vorhersage von Sätzen statt Wörtern mit Hilfe von separat erzeugten Satzeinbettungen. Die Trainingsaufgabe besteht darin, mit einem gegebenen Satz(-Einbettung) den nächsten Satz(-Einbettung) in der Anforderung vorherzusagen. Dabei lernen die Neuronen des Paragrafentoken, welche Sätze eine Anforderung besitzt. Durch dieses Training haben Anforderungen mit ähnlichen Sätzen auch eine ähnliche Einbettung. Jedoch muss für bessere Ergebnisse bei diesem Verfahren eine Anforderung aus möglichst viele Sätzen bestehen, da die Anzahl der Vorhersagen von der Anzahl an nächsten Sätzen abhängt. Dies entspricht jedoch nicht den allgemeinen Empfehlungen bezüglich des Stils von Anforderungen: Gute Anforderungen sollten eher kurz formuliert sein [AS02], da dadurch die Verständlichkeit steigt.

Als andere Option ist die Trainingsaufgabe von Sätzen bei BERT: Es werden paarweise zwei Sätze als Eingabe verwendet und es muss klassifiziert werden, ob der zweite Satz ein nächster Satz des ersten ist oder nicht. Allerdings setzt die BERT-Variante mindestens zwei Sätze pro Anforderung voraus und beim Training werden viel mehr negative als positive Satzpaare beobachtet. Um dies auszugleichen, sollten nur so viele negative wie positive Satzpaare verwendet werden, wodurch aber die Trainingsdatenmenge erheblich schrumpft. Die Anforderungen müssen also auch hier aus vielen Sätzen bestehen, um viele positive Satzpaare erstellen zu können.

Alternativ können bei der Vorhersage durch BERT oder Doc2Vec wie oben beschrieben alle Sätze von Anforderungen des gleichen Projekts verwendet werden, anstatt nur die anderen Sätze der gleichen Anforderung zu nutzen. Anforderungssätze aus anderen Projekten fungieren dabei als Negativbeispiele. Durch diese Variante wird das Problem der Satzknappheit gelöst. Jedoch entstehen auf diese Weise auch neue Probleme: Zum einen können die resultierenden Anforderungseinbettungen sehr projektspezifisch werden. Dadurch sinkt die Übertragbarkeit auf andere Projekte. Zum anderen wird beim Training für jede Anforderung alle anderen Anforderungssätze des Projekts nacheinander als Kontext betrachtet. Das hat zur Folge, dass die Anforderungseinbettungen auch die Informationen aus den anderen Anforderungen kodieren. Dies wirkt sich jedoch nachteilig auf die Rückverfolgbarkeit aus, da die Anforderungen bei der Zuordnung schlechter voneinander

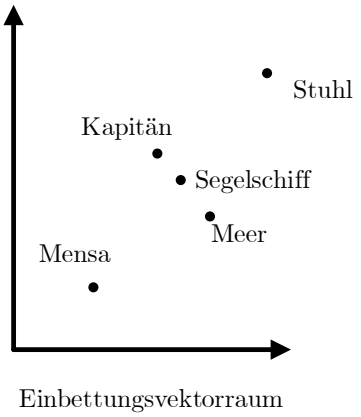


Abbildung 4.1: Fünf beispielhafte Einbettungen als Punkte im Vektorraum.

zu unterscheiden sind.

Da beim Training von Anforderungseinbettungen wie oben beschrieben andere Einbettungsverfahren zugrunde gelegt werden, hängt die Behandlung von Polysemie und OOV-Wörter vom jeweiligen Verfahren ab. Die Eigenschaft, dass Nomen und Verben bevorzugt werden, kann teilweise durch eine Vorfilterung realisiert werden. Zusammenfassend ist die Umsetzung eines eigenen Anforderungseinbettungsmodell mit Satzeinbettungen wie oben beschrieben schwierig, da die Anforderungen meist nicht genügend Sätze besitzen. Im Vergleich zur direkten Anwendung des ursprünglichen Doc2Vecs, welches auf Wörtern arbeitet, ist der Mehrwert also eher gering.

4.2.3.3 Satz- und Worteinbettungen mit Aggregation

Anforderungseinbettungen müssen nicht unbedingt trainiert werden - Anforderungswörter können auch mit einem existierenden Verfahren auf Einbettungen abgebildet und danach aggregiert werden, wobei diese Aggregation nicht wie bei den trainierten Anforderungseinbettungen durch ein neuronales Netz mit zusätzlichem Token durchgeführt werden muss. Stattdessen ist es auch möglich, einfach den Durchschnitt der Worteinbettungen als Anforderungseinbettung zu definieren. Damit ist diese Methode performanter als ein eigenes Anforderungseinbettungsmodell, da nichts trainiert werden muss. Jedoch gehen auf der anderen Seite durch die Durchschnittsbildung auch Informationen verloren. Hat man zum Beispiel einen Satz über „Stuhl“ und „Mensa“ und der Einbettungsvektorraum sieht wie in Abbildung 4.1 aus, würde man bei einer Durchschnittsbildung ungefähr bei „Segelschiff“ als resultierenden Vektor landen, was mit den ursprünglichen Begriffen überhaupt nichts mehr zu tun hat. Darüber hinaus liegt der Durchschnitt über „Meer“ und „Kapitän“ ebenfalls bei „Segelschiff“ und direkt neben dem Durchschnitt von „Stuhl“ und „Mensa“, wodurch eine inkorrekte semantische Ähnlichkeit der beiden Durchschnittsvektoren entsteht. Bei dieser Methode hat die Vorverarbeitung großen Einfluss, da dadurch direkt bestimmt wird, zwischen welchen Wörtern der Durchschnitt gebildet wird. Insbesondere die Stoppwortentfernung ist notwendig, um eine Verfälschung durch Rauschen zu minimieren. Trotz des Informationsverlustes ist die Durchschnittsbildung eine valide Vorgehensweise: In vergangenen Experimenten stellte sich die Durchschnittsbildung als gute Grundlinie heraus [CKL+18].
Die Durchschnittsbildung kann auch auf Satzebene einer Anforderung durchgeführt werden. Das heißt es werden Sätze auf Satzeinbettungen abgebildet und anschließend deren Durchschnitt als Anforderungseinbettung definiert. Wie bei den Worteinbettungen besteht die Gefahr, dass der Durchschnitt bei Vektoren liegt, die mit den ursprünglichen Sätzen

nichts zu tun haben. Hier schließt sich die Frage an, ob es einen Unterschied macht, ob der Durchschnitt über Wörter oder Sätze gebildet wird.

Beispiel 4.2: Zwei Anforderungen mit gleichen Wörtern

Anforderung 1

Der Mitarbeiter aktiviert das System. Der Kunde muss es bestätigen.

Anforderung 2

Der Kunde aktiviert das System. Der Mitarbeiter muss es bestätigen.

In Beispiel 4.2 sind dazu zwei Anforderungen aufgeführt. Der Durchschnitt über Wörter ignoriert Satzgrenzen. Nutzt man ein kontextunabhängiges Worteinbettungsverfahren, wo Wörter isoliert betrachtet werden, hat dies zur Folge, dass die beiden Anforderungen auf den gleichen Durchschnittsvektor abgebildet werden, da sie die gleichen Wörter beinhalten, obwohl die Anforderungen verschiedene Bedeutungen haben. Bei Satzeinbettungen wird der ganze Satz betrachtet, daher haben die beiden Anforderungen auch verschiedene Satzeinbettungen und damit auch verschiedene Durchschnittsvektoren. In diesem Fall sind Satzeinbettungen also überlegen. Bei kontextabhängigen Worteinbettungen wird ebenfalls der komplette Satz betrachtet, um für die Wörter kontextabhängige Einbettungen zu berechnen. Dennoch sind sie nicht identisch mit Satzeinbettungen: Satzeinbettungen werden trainiert, indem die Wörter im nächsten Satz vorhergesagt werden oder indem bei gegebenen Satzkandidaten klassifiziert wird, ob ein nächster Satz vorliegt. Satzeinbettungen beziehen folglich Informationen über die benachbarten Sätze mit ein, während kontextabhängige Worteinbettungen nur die Informationen des Satzes berücksichtigen, zu dem das Wort gehört. Dadurch ist auch der Durchschnitt nicht identisch.

Eine weitere Möglichkeit ist das zweistufige Aggregieren: Pro Satz wird keine Satzeinbettung, sondern der Durchschnitt der Worteinbettungen berechnet. Über alle Satzdurchschnittsvektoren wird nochmals der Durchschnitt gebildet, um einen aggregierten Anforderungsvektor zu erhalten. Die doppelte Durchschnittsbildung hat jedoch den entscheidenden Nachteil, dass der resultierende Vektor noch weiter abseits von den ursprünglichen Bedeutungen liegen kann als es bei der einfachen Durchschnittsbildung der Fall ist. Wenn zum Beispiel die Satzdurchschnittsvektoren bereits stark von den Wortbedeutungen abweichen und über diese „falschen“ Satzdurchschnittsvektoren wieder ein Durchschnitt gebildet wird, entfernt sich der resultierende Vektor höchstwahrscheinlich noch weiter.

Beim Durchschnitt werden die Einbettungen ungewichtet aggregiert. Das hat zur Folge, dass relevante und weniger relevante Wörter gleich behandelt werden. Durch eine Gewichtung könnten bestimmte Wörter oder Sätze jedoch mit größerem Einfluss auf die Summe eingerechnet werden. Damit ist die gewünschte Eigenschaft, Nomen und Verben zu bevorzugen, umsetzbar. Es gibt auch andere Gewichtungsmöglichkeiten als nach ihrer Wortart, wie etwa das tf-idf-Maß (Abschnitt 2.6.3). Damit werden Wörter bevorzugt, die in wenigen Dokumenten besonders oft auftauchen. Solche Wörter diskriminieren die wenigen Dokumente von anderen Dokumenten und besitzen daher einen hohen Informationsgehalt. Das tf-idf-Maß besitzt jedoch auch Nachteile: Synonyme werden als eigenständige Wörter behandelt und die verschiedenen Bedeutungen eines polysemen Wortes werden

nicht getrennt. Außerdem ist es nur eine heuristische Annahme, dass häufige Wörter, die in wenigen Dokumenten vorkommen, auch einen hohen Informationsgehalt besitzen. Die Häufigkeit eines Wortes muss nicht immer mit ihrem Informationsgehalt korrelieren.
Allgemein bei gewichtete Summen besteht bezüglich der Ähnlichkeitseigenschaft das gleiche Problem wie bei der Durchschnittsbildung, falls die Gewichte schlecht gewählt sind: Der resultierende Vektor kann sehr abseits von den ursprünglichen Bedeutungen liegen. Die Wahl der Gewichte ist bei dieser Aggregation entscheidend.

Bei der Nutzung von Satzeinbettungen können die Satzeinbettungen gewichtet werden, sodass Sätze mit mehr Informationen bevorzugt werden. Bei Anforderungsdokumenten, in denen zum Beispiel der erste Satz eine Art eine Ein-Satz-Beschreibung ähnlich wie bei Wikipedia darstellt, könnte eine höhere Gewichtung des ersten Satzes potentiell hilfreich sein, da dieser Satz wahrscheinlich den höchsten Informationsgehalt besitzt. Ob Satzgewichtungen sinnvoll sind, hängt folglich von der Beschaffenheit der jeweiligen Anforderungsdokumente ab. Beispielsweise können Anwendungsfälle, die in textueller Form beschrieben sind, eine Überschrift besitzen [BD09]. Die Überschrift fasst in der Regel den Kerninhalt der Anwendungsbeschreibung zusammen und sollte damit höher gewichtet werden als die anderen Sätze. Anforderungen können jedoch auch ohne Überschrift oder Vergleichbares formuliert sein und aus mehreren Sätzen bestehen, die von der äußerlichen Form nicht zu unterscheiden sind. In diesem Fall kann nicht so einfach wie bei einer Überschrift abgeleitet werden, welche Teile der Anforderung wahrscheinlich einen höheren Informationsgehalt besitzen.

Eine weitere Aggregationsoption ist die Konkatenation, das heißt pro Anforderung bildet die Konkatenation der enthaltenen Wort- oder Satzeinbettungen die Anforderungseinbettung. Der Vorteil hierbei ist, dass zunächst keine Informationen wie bei der Durchschnittsbildung verloren gehen. Ein großer Nachteil dieses Vorgehens ist die Länge des resultierenden Vektors: Sie hängt direkt von der Anzahl Wörter bzw. Sätze in der Anforderung ab und wird sich folglich je nach Anforderung unterscheiden. Für spätere Berechnungen mit den Vektoren wie für die Kosinusähnlichkeit müssen die Vektoren jedoch die gleiche Dimension haben. Um die Vektoren auf die gleiche Länge zu bringen, können zu lange Vektoren abgeschnitten und zu kurze Vektoren mit Standardwert aufgefüllt werden. Je nach Anwendungsfall kann dies nachteilig sein: Falls Ähnlichkeitsmetriken verwendet werden, dies sich aus der Lage der Einbettungen zueinander im Vektorraum berechnen, würde das Auffüllen das Ergebnis verfälschen. Zur Eingabe in ein Trainingsverfahren können solch Vektoren allerdings genutzt werden, um neue Einbettungen zu lernen.

Mit der Auswahl einer repräsentativen Einbettung unter den Satz- bzw. Worteinbettungen kann auch eine Anforderungseinbettung definiert werden. Die gewählte Satz- oder Worteinbettung sollte den Kerninhalt der Anforderung, ähnlich wie eine Überschrift, beinhalten. In der Praxis ist jedoch nicht trivial, den Kernsatz oder das -wort zu identifizieren. Außerdem muss es nicht unbedingt einen Kernsatz oder ein -wort geben, da die Semantik der Kernaussage nicht immer in einem Wort oder Satz ausgedrückt werden kann. Die Durchschnittsbildung ist hier überlegen: Beim Durchschnitt können mehrere bzw. alle Wörter/Sätze berücksichtigt werden und es muss kein Kernsatz oder -wort existieren.

Da die obigen Aggregationsverfahren existierende Satz- und Worteinbettungsverfahren nutzen, hängen die Eigenschaften über OOV- und Polysemieunterstützung direkt vom verwendeten Verfahren ab. Weiterhin besteht eine Abhängigkeit zu den Trainingsdaten, mit denen das existierende Verfahren trainiert wurde. Diese sind meist auf allgemeinen Datensätzen trainiert und können bei Wörtern nicht korrekt abbilden, die in spezifischen Domänen eine andere Bedeutung haben als im allgemeinen Sprachgebrauch. Wie bei Doc2Vec-Training in Abschnitt 4.2.3.2 ausgeführt, ist es möglich, die vortrainierten Modelle durch neue domänenspezifische Modelle zu ersetzen oder mit domänenspezifischen Daten nach-

zutrainieren. Dabei ergeben sich die gleichen Gefahren, die in Abschnitt 4.2.3.2 erklärt wurden: Das neue Modell könnte zu domänenspezifisch werden oder es gibt zu wenige Trainingsdaten, um einen Einfluss zu bewirken. Die Eigenschaft des kontinuierlichen Vektors wird durch die Aggregation nicht verändert. Die Ähnlichkeitseigenschaft ist bei allen Aggregationsverfahren beeinträchtigt, da es zu Informationsverlust kommt.

4.2.3.4 Bag-of-Embeddings

Die einzelnen Satz- oder Worteinbettungen einer Anforderung können auch ohne Aggregation ähnlich wie ein Bag-Of-Words (Abschnitt 2.6.2) als „Bag-of-Embeddings“ der Anforderung zugeordnet werden. Hierbei gibt es folglich keine einzige resultierende Anforderungseinbettung wie bisher in den obigen Verfahren. Die OOV- und Polysemieeigenschaft hängt auch hier vom verwendeten Einbettungsverfahren ab. Eine Gewichtung der Sätze oder Wörter und die Frage, ob man überhaupt mit einer Menge von Einbettungen statt einem Vektor pro Anforderung umgehen kann, hängt vom später verwendeten Abbildungsverfahren zwischen Anforderungen und Quelltext ab. Der Vorteil bei dieser Repräsentation liegt vor allem darin, dass keine Informationen verloren gehen.

4.2.4 Vorverarbeitung

Bevor Wörter und Sätze auf Einbettungen abgebildet werden, werden sie durch eine Vorverarbeitung aufbereitet. Dies dient dazu, dass zum Beispiel das Rauschen verringert wird. In Abschnitt 2.3 wurden verschiedene Vorverarbeitungsschritte aufgelistet, aber nicht jede Vorverarbeitung ist auch immer sinnvoll.

Die Stoppwortentfernung filtert häufig auftretende Wörter mit wenig Informationsgehalt aus dem Text heraus. Bei Nutzung kontextunabhängiger Worteinbettungsverfahren wie fastText ist das sinnvoll, da Stoppwörter nur einen niedrigen Informationsgehalt haben. Einbettungen solcher Wörter haben im Vektorraum eine Vielzahl von gleich oder ähnlich entfernten Nachbarn, denn Stoppwörter tauchen in vielen Kontexten mit vielen verschiedenen Wörtern auf. Die Weiterverarbeitung von Stoppworteinbettungen würde also nur für mehr Rauschen sorgen. Ein weiterer positiver Nebeneffekt, wenn Stoppwörter entfernt werden, ist die bessere Performanz, da weniger Wörter weiterverarbeitet werden müssen. Bei kontextabhängigen Verfahren wie ELMo und BERT wird der Kontext inklusive Stoppwörter verwendet, um Einbettungen zu berechnen. Die Stoppwörter verrauschen hier die Daten nicht, vielmehr ist der Kontext notwendig, um die richtige Bedeutung eines Wortes zu identifizieren: Durch das mehrschichtige LSTM-Netz bzw. den Beachtungsmechanismus lernen die neuronalen Netze, welche Nachbarwörter viel Information enthalten und welche nicht. Aus diesem Grund ist es hier nicht nötig, die Wörter mit wenig Informationsgehalt vorher zu entfernen. Bei den angesprochenen Aggregationsverfahren aus Abschnitt 4.2.3.3 ist die Stoppwortentfernung wiederum nützlich, um weniger relevante Wörter vor der Aggregation herauszufiltern.

Die Transformation in Kleinbuchstaben sorgt dafür, dass das gleiche Wort, egal ob groß- oder kleingeschrieben, gleich behandelt wird (auf die gleiche Einbettung abgebildet wird). Im Englischen beschränkt sich die Großschreibung größtenteils auf Wörter am Satzanfang und Eigennamen. Wörter verändern nicht ihre Bedeutung, wenn sie am Satzanfang stehen, folglich ist es unsinnig, für die Großschreibung eine separate Einbettung zu haben. Eigennamen haben meistens keine andere Bedeutung, wenn sie klein- statt großgeschrieben werden - also macht es keinen Unterschied, ob die Einbettung auf dem groß- oder kleingeschriebenen Wort berechnet wird. Aus diesen Gründen ist die Transformation in Kleinbuchstaben ein sinnvoller Vorverarbeitungsschritt - außer es wird ein existierendes Einbettungsverfahren verwendet, welches auf Text ohne Kleinbuchstabentransformation trainiert wurde.

Ob Lemmatisierung im Kontext von Worteinbettungen sinnvoll ist, hängt von der Frage ab, ob verschiedene Formen des gleichen Wortes verschiedene Worteinbettungen besitzen sollen bzw. ob die Informationen über die verschiedenen Wortformen nützlich genug sind. Durch die Kodierung der semantischen Ähnlichkeit liegen die verschiedenen Wortformen im Einbettungsraum wahrscheinlich sowieso nah beieinander. Es gibt Worteinbettungsverfahren, die eine eigene Lemmatisierung vornehmen. Die Subworteinteilung von BERT trennt zum Beispiel bereits Wortstamm und Konjugationsendung und auf dieser eigenen Subwortaufteilung wird anschließend auch trainiert. In diesem Fall ist also eine gesonderte Lemmatisierung nicht nötig. Bei Verfahren mit Buchstaben-N-Grammen haben beispielsweise verschiedene Konjugationsformen teilweise gleiche Buchstaben-N-Gramme (da gleicher Wortstamm) und dadurch auch ähnliche Worteinbettungen. Auch hier werden verschiedene Wortformen intern im Worteinbettungsverfahren behandelt.
In einer Arbeit von Kutuzov und Kuzmenko [KK19] wurde eine Wortbedeutungsdisambiguierung einmal mit und einmal ohne Lemmatisierung mit Hilfe von ELMo-Einbettungen für Englisch und Russisch durchgeführt. Dabei stellte sich heraus, dass die Lemmatisierung für Englisch nicht unbedingt nötig ist, aber für das morphologisch vielfältigere Russisch trotz der internen Buchstaben-N-Gramm-Aufteilung durch ELMo durchaus eine kleine Verbesserung darstellt. Ein Vorteil bei der Lemmatisierung ist außerdem, dass die verschiedenen Wortformen auf die Grundform abgebildet werden, womit das Wort in der Grundform öfter im Korpus auftaucht. Das ist nützlich wenn man ein eigenes Modell trainieren will, aber nur kleine Korpora zu Verfügung hat. Die Lemmatisierung sollte außerdem durchgeführt werden, wenn man ein Modell verwendet, welches auf lemmatisierten Daten trainiert wurde. Ansonsten ist dieser Schritt optional.

Binnenmajuskelschreibweisen treten in natürlicher Sprache nicht auf. Die Auflösung solcher Schreibweisen ist für die Anforderungsseite nur nötig, falls zum Beispiel Klassennamen in Binnenmajuskelschreibweise referenziert werden. Dies kann unter anderem bei Dokumenten auftreten, die aus Quelltext generiert wurden oder die implementierungsnahe Anforderungen beschreiben.

Ein Nichtbuchstabenfilter sollte eingesetzt werden, um alle Zeichen außer Buchstaben herauszufiltern. Im Falle von BERT können Satzzeichen erhalten bleiben, da diese eingebettet und als Kontext berücksichtigt werden. Es macht keinen Sinn, für Zahlen und Sonderzeichen Einbettungen zu berechnen, da diese bei der Zuordnung zu Quelltext nicht hilfreich sind. Zeichen wie „#“ oder „(“ tragen nicht genug Semantik und würden nur Rauschen verursachen. Zahlen kommen in Quelltext sehr oft bei primitiven Zahlentypen vor. Die Zahlen in Anforderungen und die im Quelltext können also meistens nicht korrekt aufeinander abgebildet werden.

Ein Wortlängenfilter kann zusätzlich verwendet werden, um Wörter mit wenigen Buchstaben zu entfernen, die wahrscheinlich wenig semantischen Inhalt besitzen oder Abkürzungen darstellen. Für BERT sollte der Wortlängenfilter nicht benutzt werden, da BERT, wie bei der Stoppwortentfernung erklärt, auch kurze Wörter mit wenig Semantik als Kontext benötigt. Für fastText kann es jedoch verwendet werden, um die Stoppwortentfernung zu ergänzen bzw. Abkürzungen zu entfernen, da diese hier nicht aufgelöst werden.

4.2.5 Entwurf von Anforderungseinbettungen

Das Ziel ist die Erzeugung von Anforderungseinbettungen, die für die Rückverfolgbarkeit zwischen Anforderungen und Quelltext verwendet werden sollen. In Abschnitt 4.2.1 wurden Eigenschaften diskutiert, die die Anforderungseinbettungen idealerweise besitzen sollten. Dabei war die Repräsentation als kontinuierlichen Vektor die Wichtigste. Dies ist bei allen im vorigen Abschnitt vorgestellten Verfahren erfüllt. Bei den anderen Eigenschaften treten Unterschiede zwischen den Verfahren auf.

Die vorgestellten Anforderungseinbettungsverfahren lassen sich in drei Kategorien aufteilen: Existierende, trainierte und aggregierte Anforderungseinbettungsverfahren. Ersteres erzeugen direkt eine ganze Anforderungseinbettung. Diskutiert wurden BERT und Doc2Vec, wobei BERT ursprünglich nicht für diesen Zweck trainiert wurde und für Doc2Vec kein offizielles vortrainiertes Modell existiert. Bei der zweiten Kategorie werden die Einbettungen durch ein neuronales Netz trainiert und bei der Letzten werden Einbettungen durch vorhandene Einbettungsverfahren erzeugt und ohne neuronales Netz zu Anforderungseinbettungen aggregiert. Wie in Abschnitt 4.2.3.2 erklärt, gibt es beim Trainieren eines eigenen Modells das Problem, dass dieses entweder zu allgemein oder zu domänenspezifisch werden kann bzw. ein gutes Mischungsverhältnis a priori unklar ist. Da es kein vortrainiertes Doc2Vec-Modell gibt, gilt dies auch für die Nutzung von Doc2Vec. Hierfür müssen die Gewichte also empirisch bestimmt werden.

Wie in Abschnitt 4.2.3.3 ausgeführt, ist die Berechnung einer Aggregation wie der Durchschnittsberechnung performanter, da kein eigenes Modell trainiert werden muss. Trotz des Informationsverlustes bildet die Durchschnittsbildung eine valide Grundlinie. Eine Gewichtung kann man bei der Aggregation durch die gewichtete Summe direkt miteinfließen lassen und durch Nutzung entsprechender Einbettungsverfahren werden OOV und Polysemie unterstützt.
Unter den Aggregationsmöglichkeiten waren die Durchschnittsbildung und die gewichtete Summe der Konkatenation, dem repräsentativen Vektor und der zweistufigen Aggregation überlegen. Daher werden diese beiden Möglichkeiten bevorzugt verwendet.

Ob existierende, trainierte oder aggregierte Anforderungseinbettungsverfahren besser sind, ist aus der Analyse nicht abschließend bestimmbar. Dies muss also empirisch evaluiert werden. Als existierendes Verfahren soll BERT verwendet werden, als trainiertes Verfahren Doc2Vec und bei den Aggregationen sollen der Durchschnitt und die gewichtete Summe umgesetzt werden.

Aus der Analyse der existierenden Worteinbettungsverfahren in Abschnitt 4.2.2 hat sich ergeben, dass fastText und BERT geeignete Kandidaten sind, die für Anforderungseinbettungsverfahren genutzt werden können.

Bezüglich der Vorverarbeitung wird bei fastText die Transformation in Kleinbuchstaben, der Nichtbuchstabenfilter und die Stoppwortentfernung eingesetzt. Abkürzungen werden nicht aufgelöst, daher soll auch ein Wortlängenfilter verwendet werden. Die Lemmatisierung ist in diesem Fall optional. Falls in den Anforderungen Binnenmajuskelschreibweisen auftreten, werden diese aufgeteilt. Die Schritte sind in Aufzählung 4.1 zusammengefasst.

In Zusammenhang mit BERT wird, wie in Aufzählung 4.2 zu sehen, die Binnenmajuskelschreibweise aufgetrennt, da der BERT-Portionierer dies nicht erkennen kann. Der Nichtbuchstabenfilter filtert bei BERT alle Zeichen außer Buchstaben und Satzzeichen heraus,

1. Nichtbuchstabenfilter
2. Binnenmajuskelschreibweise auflösen
3. Transformation in Kleinbuchstaben
4. Lemmatisierung
5. Stoppwortentfernung
6. Wortlängenfilter

Aufzählung 4.1: Vorverarbeitungsschritte mit fastText für Anforderungen

1. Nichtbuchstabenfilter
2. Binnenmajuskelschreibweise auflösen

Aufzählung 4.2: Vorverarbeitungsschritte mit BERT für Anforderungen

um Rauschen zu verringern. Die anderen Vorverarbeitungsschritte können durch BERT gehandhabt werden oder sind für BERT kontraproduktiv.

Die Nutzung von Bag-of-Embeddings hängt von der Abbildung zu Quelltexteinbettungen ab und wird in Abschnitt 4.4.4 diskutiert.

4.3 Analyse und Entwurf von Quelltexteinbettungen

Für die Quelltextseite der Anforderung-zu-Quelltextrückverfolgbarkeit sollen ebenfalls Einbettungen zum Einsatz kommen. Wie in IR-Verfahren kann Quelltext als Fließtext aufgefasst werden, jedoch können die verschiedenartigen Elemente wie Kommentare oder Bezeichner auch differenziert berücksichtigt werden. Es gibt viele Möglichkeiten, wie und welche Elemente kombiniert werden, um Einbettungen zu erzeugen. In diesem Abschnitt werden zuerst allgemein die erwünschten Eigenschaften von Quelltexteinbettungen mit Blick auf die Anforderung-zu-Quelltext-Rückverfolgbarkeit diskutiert (Abschnitt 4.3.1). Anschließend werden die existierenden Quelltexteinbettungsverfahren beurteilt (siehe Abschnitt 4.3.2) und eigene Verfahren erarbeitet (Abschnitt 4.3.6). In Abschnitt 4.3.4 werden Vorverarbeitungsschritte für Quelltext diskutiert und in Abschnitt 4.3.6 wird der Entwurf für Quelltexteinbettungen vorgestellt.

4.3.1 Eigenschaften von Quelltexteinbettungen

Es wird als Erstes eruiert, welche Eigenschaften eine Quelltexteinbettung im Rahmen der Anforderung-zu-Quelltext-Rückverfolgbarkeit im Idealfall besitzen soll.

Repräsentation als kontinuierlicher Vektor
Quelltexteinbettungen sollen kontinuierliche Vektoren sein, um die Rückverfolgbarkeit mit Einbettungen zu untersuchen zu können. Dadurch ist eine Ähnlichkeitskodierung durch die Distanz im Einbettungsvektorraum möglich. Wie für die Anforderungsseite in Abschnitt 4.2.1 bereits dargelegt, ist dies aus diesem Grund die wichtigste Eigenschaft.

Kodierung der Ähnlichkeit
Ein großer Vorteil von Einbettungen besteht in der Kodierung von Ähnlichkeit zwischen den repräsentierten Elementen durch die Lage der Vektoren im Einbettungsraum. Hier muss zunächst geklärt werden, was unter Ähnlichkeit zwischen Quelltextelementen zu verstehen ist. Für Anforderungen wurde in Abschnitt 4.2.1 erklärt, dass ihre Ähnlichkeit von der Semantik abhängt. Das Ziel der Rückverfolgbarkeit ist die Zuordnung von Anforderungen zu Quelltextelementen. Insofern ist es sinnvoll, den Grad der Ähnlichkeit zwischen Quelltextelementen vom Grad der Ähnlichkeit ihrer korrespondierenden Anforderungen abhängig zu machen.
Es gibt auch die Möglichkeit, die Ähnlichkeit durch quelltextspezifische und syntaktische Merkmale zu definieren. „getName()“ und „getID()“ sind sich dadurch ähnlich, dass sie beide Getter-Funktionen sind. Oder zwei Quelltextelemente sind sich ähnlich, weil sie die gleichen syntaktischen Konstrukte besitzen: Schleifen, bedingte Sprünge, etc. Solche Ähnlichkeitsdefinitionen sind jedoch in Anbetracht der Anforderungen-zu-Quelltextrückverfolgbarkeit nicht hilfreich. Wenn zwei Methoden zu verschiedenen Anforderung gehören, ist es kontraproduktiv, wenn ihre Ähnlichkeit steigt, nur weil sie beide Getter-Funktionalitäten umsetzen oder eine while-Schleife besitzen. Quelltexteigenschaften, die wie die obigen zu implementierungsnah sind, sind bezüglich der Rückverfolgbarkeit weniger hilfreich.
Es kann jedoch sein, dass „abstraktere“ Quelltexteigenschaften als Indiz für eine Ähnlichkeit der Anforderungen verwendet werden können. Dazu gehören Aufrufbeziehungen zwischen Methoden. Eine Anforderung kann auch durch mehrere Methoden umgesetzt werden, die sich gegenseitig aufrufen und jeweils einen Teilaspekt der Anforderung umsetzen. In diesem Fall bestehen also Aufrufbeziehungen zwischen Methoden, die der gleichen

Anforderung zugeordnet sind. Diese Überlegungen decken sich auch mit den Ergebnissen der Studie von Burgstaller et al. [BE10]. Dort wurden die Aufrufbeziehungen zwischen Methoden untersucht und es wurde festgestellt, dass Methoden, die zur gleichen Anforderung gehören, mit größerer Wahrscheinlichkeit eine Aufrufbeziehung zueinander haben als mit irgendeiner zufälligen Methode.
Auch Methoden, die nicht zur gleichen, aber zu ähnlichen Anforderungen gehören, können eine Aufrufbeziehung zueinander besitzen. Es ist zum Beispiel denkbar, dass Methoden einer „Ausleihen"-Anforderung einer Bibliothek Methoden einer Anforderung über die Kontoverwaltung aufruft, da das Ausleihen ein Konto erfordert. Solche Aufrufbeziehungen zwischen Methoden können infolgedessen Hinweise auf eine gemeinsame oder ähnliche Anforderung geben.
Außer Aufrufbeziehungen existieren auch Datenflussabhängigkeiten. Zwischen Methoden liegt dies vor, wenn sie Objekte beim Methodenaufruf übergeben oder wenn sie auf den gleichen globalen Variablen arbeiten. Beim Ersteren liegt eine Aufrufbeziehung vor und wurde soeben bereits analysiert. Der zweite Fall kann es aber genauso ein Indiz für die Verbindung zur gleichen oder einer ähnlichen Anforderung sein. Es ist nicht unwahrscheinlich, dass Methoden der gleichen oder einer ähnlichen Anforderung auch auf den gleichen Daten arbeiten. Die Methoden der „Ausleihen"-Anforderung und der „Kontoverwaltung"-Anforderung können auf das gleiche Kontoobjekt zugreifen. Kuang et al. [KMH+15] haben hierzu festgestellt, dass Datenabhängigkeiten genauso wichtig wie Aufrufbeziehungen sind und sich sogar mit ihnen in einer komplementären Art und Weise ergänzen.
Vererbungsbeziehungen sind eine weitere Quelle für mögliche Ähnlichkeitsimplikationen. Unterklassen erben Funktionalitäten ihrer Oberklassen, dadurch ist es realistisch, dass Ober- und Unterklasse zur gleichen oder einer ähnlichen Anforderung gehören.
Zusammengefasst sollte die Ähnlichkeit zwischen Quelltexteinbettungen im Idealfall von der Ähnlichkeit ihrer Anforderungen abhängen. Quelltextspezifische Merkmale können dabei helfen, solche Ähnlichkeiten festzustellen. Zu implementierungsnahe Eigenschaften sind dagegen für die Rückverfolgbarkeit nicht hilfreich.

Einbezug natürlichsprachiger Elemente des Quelltextes
Wie bei der Ähnlichkeitseigenschaft erklärt, unterstützen zu implementierungsnahe Quelltexteigenschaften die Rückverfolgbarkeit nicht, etwas abstraktere Eigenschaften wie Aufrufe zwischen Methoden dagegen schon. Nur mit ausschließlich solchen abstrakteren Quelltexteigenschaften lässt sich jedoch keine Anforderung-zu-Quelltextrückverfolgbarkeit durchführen. Mit Hilfe von Aufrufbeziehungen können zwar Gruppen von Methoden identifiziert werden, die wahrscheinlich zu der gleichen Anforderung gehören, aber daraus ist nicht ableitbar, welche Anforderung das ist. Mit quelltextspezifischen Eigenschaften alleine können keine Verbindungen zu Anforderungen hergestellt werden, da die quelltextspezifischen Eigenschaften nichts über die Semantik von Anforderungen aussagen. Diese Semantik kann allerdings von den natürlichsprachigen Elementen im Quelltext, also Kommentare und Bezeichner, widergespiegelt werden. Eine Anforderung über eine Mensa kann mit einer Klasse namens „Mensa" implementiert werden und eine Beschreibung über die Essensausgaben als Kommentar besitzen. Aus diesem Grund müssen bei der Erzeugung von Quelltexteinbettungen natürlichsprachige Elemente aus dem Quelltext miteinbezogen werden. Dabei soll die Semantik der Bezeichner und Kommentare in der Quelltexteinbettung kodiert werden, damit diese auf die Semantik der Anforderungseinbettungen abgebildet werden können. Diese Eigenschaft ist wichtiger als die Ähnlichkeitseigenschaft, da sonst keine Chance besteht, Verbindungen zu Anforderungen herzustellen.

OOV-Behandlung
Wie bei den Worteinbettungen kann es bei den Quelltexteinbettungen auch zu OOV-Fällen kommen, wenn die Klasse oder Methode bzw. deren Bestandteile nicht im Training vorkamen. Entwickler können Bezeichnern willkürlich wählen und Implementierungen sind

domänenspezifisch. Dies fördert das Auftreten von OOV-Bezeichnern. Folglich ist auch eine Behandlung solcher Fälle wünschenswert.

Die diskutierten Eigenschaften sind im Folgenden zusammengefasst:

Erwünschte Eigenschaften für Quelltexteinbettungen bzw. deren Verfahren

1. Repräsentation von Klassen oder Methoden als kontinuierlichen Vektor
2. Einbezug natürlichsprachiger Elemente des Quelltextes
3. Ähnlichkeit zwischen Methoden/Klassen, die ähnliche Anforderungen besitzen
4. Behandlung von OOV-Fällen

4.3.2 Analyse existierender Quelltexteinbettungsverfahren

Es existieren bereits Verfahren, um Quelltexteinbettungen zu erzeugen (Abschnitt 2.7.2). Im Folgenden wird analysiert, ob diese die im vorigen Abschnitt beleuchteten Eigenschaften erfüllen.

Ein existierendes Verfahren ist Code2Vec (Abschnitt 2.7.2.1). Damit lassen sich Quelltextausschnitte mit einem Vektor repräsentieren. Was genau unter „Quelltextausschnitt“ verstanden wird, wurde in der Arbeit nicht genauer erläutert. Jedoch wird in der Trainingsphase versucht, für Methodenrümpfe ein Label bzw. einen Methodennamen vorherzusagen, welches die Funktionalität des gegebenen Quelltextausschnittes beschreibt. Daher ist anzunehmen, dass zumindest das vortrainierte Modell am besten mit Methoden funktioniert. Code2Vec basiert auf einer gewichteten Summe von Pfaden im abstrakten Syntaxbaum (AST), daher sind sich zwei Quelltexteinbettungen ähnlich, wenn sie ähnliche AST-Pfade mit möglichst gleichen Gewichten haben. Da die Reihenfolge der Pfade ungeordnet ist, spielt es hierbei keine Rolle, ob zum Beispiel eine Schleife am Anfang oder am Ende einer Methode steht.
OOV-Pfade, -AST-Blätter und -Labels sowie unbekannte Programmiersprachen werden inhärent nicht unterstützt: Zum einen ist Code2Vec programmiersprachenabhängig, da mit dem AST gearbeitet wird. Zum anderen wird jedem AST-Pfad in Code2Vec als Ganzes eine interne Pfadeinbettung zugewiesen. Das heißt, dass zwei Pfade auch zwei komplett unterschiedliche Einbettungen besitzen, sobald sie sich nur in einem Knoten unterscheiden (zum Beispiel statt Double ein Integer). Folglich können OOV-Pfade nicht abgebildet werden, falls sie unbekannte Pfade besitzen bzw. sich nur um einen Knoten von bekannten Pfaden unterscheiden. Ähnliches gilt für AST-Blätter, die oft Bezeichner repräsentieren. Diese werden ebenfalls als eine Einheit aufgefasst (keine Trennung wenn Bezeichner aus mehreren Wörtern besteht) und auf interne AST-Blatt-Einbettungen abgebildet. Unbekannte Bezeichner haben daher keine AST-Blatt-Einbettungen und können nicht repräsentiert werden. Die Vorhersage der Methodennamen ist eine Klassifikation, welcher Methodenname aus dem Training am wahrscheinlichsten ist; dabei werden die Methodennamen als Ganzes betrachtet und nicht zerteilt. Daraus folgt, dass keine unbekannten Methodennamen vorhergesagt werden können. Code2Vec kompensiert das OOV-Problem teilweise, indem auf einem sehr großen Korpus trainiert und dadurch die tatsächliche Auftrittswahrscheinlichkeit von OOV-Fällen reduziert wird. Dennoch können sehr domänenspezifische Namen nicht vorhergesagt werden. Jedoch enthält das Code2Vec-Modell viele generische Methodennamen wie „findIndex()“. Für einen sehr seltenen Namen wie „findPrecalculatedCatImageIndexByRank()“ besteht die Chance, dass der ähnliche „findIndex()“-Name vorhergesagt wird, falls ihre AST-Pfade ähnlich sind.
Daten- und Kontrollflussabhängigkeiten innerhalb des Quelltextausschnittes bzw. Methode werden implizit berücksichtigt. Eine Datenabhängigkeit besteht zum Beispiel, wenn am Anfang einer Methode eine Variable initialisiert wird und diese später verwendet wird. Der AST ist ein zusammenhängender Baum, also existiert auf jeden Fall ein Pfad von

der Initialisierung zur Verwendung, welches implizit die Datenabhängigkeit repräsentiert. Gleiches Argument gilt für den Kontrollfluss, wo zum Beispiel die verschiedenen Sprungziele einer If-Abfrage auch mit einem Pfad im AST verbunden sind. Jedoch muss in der Praxis beachtet werden, dass Code2Vec aus Performanzgründen einen Parameter für die maximale Pfadlänge besitzt. Das heißt, dass Daten- oder Kontrollflussabhängigkeiten bei Pfaden, die die maximale Länge überschreiten, nicht mehr berücksichtigt werden.
Die von Code2Vec erzeugten Einbettungen erfüllen nicht die gewünschte Ähnlichkeitseigenschaft für Quelltexteinbettungen. Die Code2Vec-Einbettungen fangen die Pfade und die dazugehörigen Daten- und Kontrollflussabhängigkeiten innerhalb einer Methode ein. Diese Informationen sind zu implementierungsnah, um zuverlässige Aussagen über die Ähnlichkeit der dazugehörigen Anforderungen zu machen. Ähnliche Anforderungen haben nicht unbedingt ähnliche AST-Pfade in der Implementierung. Die Verbesserung von Code2Vec, Code2Seq (Abschnitt 2.7.2.1), basiert ebenfalls auf Pfaden innerhalb einer Methode. Daher ändert sich die Ähnlichkeitseigenschaft hier nicht. Allerdings werden bei Code2Seq die Bezeichner entlang der Binnenmajuskelschreibweise getrennt. Dadurch können OOV-Bezeichner verarbeitet werden, solange ihre Bestandteile im Training gesehen wurde, was eine Verbesserung gegenüber Code2Vec darstellt.

Bei inst2vec (Abschnitt 2.7.2.2) wird der Quelltext als Vorverarbeitung in die LLVM-Zwischenrepräsentation überführt. Dabei wird für jede LLVM-Instruktion anhand des Ausdrucksgraphen eine Einbettung trainiert. Das Training besteht dabei aus der Vorhersage der Nachbarknoten im Ausdrucksgraphen und zwei Knoten sind im Ausdrucksgraphen verbunden, wenn sie im kontextuellen Flussgraphen eine entsprechende Daten- oder Kontrollflusskante besaßen. Daher sind sich hier zwei Einbettungen ähnlich, wenn sie ähnliche Daten- und Kontrollflussabhängigkeiten zu ähnlichen Instruktionen besitzen. Durch den kontextuellen Flussgraphen werden hier die Daten- und Kontrollflussabhängigkeiten zwischen Instruktionen explizit berücksichtigt. OOV-Instruktionen können nicht abgebildet werden, da diese im Training nicht gesehen wurden und daher keine Einbettung besitzen. Nur ungesehene Bezeichner und Konstanten (mit bekannten Instruktionen) werden toleriert, da diese beim Training im kontextuellen Flussgraphen durch Platzhalter ersetzt werden. Durch die Transformation in die LLVM-IR ist inst2vec außerdem programmiersprachenunabhängig. Bezüglich der gewünschten Ähnlichkeitseigenschaft ist inst2vec zu implementierungsnah. Die Daten- und Kontrollflussabhängigkeiten werden auf Instruktionsebene miteinbezogen. Das Vorhandensein einer ähnlichen Schleife impliziert zum Beispiel Ähnlichkeit auf Instruktionsebene. Solche Gemeinsamkeiten sind, wie in Abschnitt 4.3.1 erklärt, zu detailliert, als dass sie verlässliche Rückschlüsse auf ähnliche Anforderungen zulassen. Außerdem werden natürlichsprachige Elemente nicht berücksichtigt.

Das Problem mit OOV-Instruktionen wird in IR2Vec (Abschnitt 2.7.2.3) gelöst, indem LLVM-Instruktionen in Teilinstruktionen in Form von Quelltexttripel zerlegt werden. Aus unbekannten Instruktionen können bekannte Quelltexttripel gewonnen und dadurch auf Einbettungen abgebildet werden. Dies wurde in der Arbeit auch in einem Experiment bestätigt, in der sowohl Code2Vec und inst2vec OOV-Fälle hatten, während bei IR2Vec kein einziger aufgetreten ist. Genau wie bei inst2vec werden auch hier alle Bezeichner und Konstanten durch Platzhalter ersetzt und durch die LLVM-Transformation ist IR2Vec programmiersprachenunabhängig. Wie in Abschnitt 2.7.2.3 dargelegt, kann IR2Vec für LLVM-Instruktionen, LLVM-Basisblöcke, Funktionen und Programme Einbettungen generieren. Daten- und Kontrollflussabhängigkeiten werden auf Instruktionsebene explizit berücksichtigt. Die Abhängigkeiten auf Basisblock-, Funktions- und Klassenebene werden implizit miteinbezogen, da auf Instruktionsebene die Einbettung der aufgerufenen Funktion gewichtet mitsummiert wird. Durch die verschachtelte Summenbildung über die Instruktionseinbettungen zur Berechnung der grobgranularen Einbettungen gehen jedoch Informationen verloren: Eine Instruktionseinbettung berechnet sich aus der gewichteten Summe

Tabelle 4.2: Zusammenfassung der Eigenschaften der Quelltexteinbettungsverfahren

Verfahren	Repräsentation	Daten-/Kontrollfluss	OOV	Ähnlichkeit
Code2Vec	Methoden	implizit auf Ausdrucksebene	Nein	Ähnlich gewichtete AST-Pfade
Code2Seq	Methoden	implizit auf Ausdrucksebene	Ja	Ähnlich gewichtete AST-Pfade
inst2vec	LLVM-Instruktionen	explizit auf Instruktionsebene	Nein	Gleiche DF-/KF-Abhängigkeiten zu anderen Instruktionen
IR2Vec	LLVM-Instruktionn, Basisblöcke, Funktionen, Programme	explizit auf Instruktions- / Basisblockebene, implizit auf Funktions-/ Programmebene	Ja	Gleiche DF-/KF-Abhängigkeiten zu anderen Instruktionen

seiner Quelltexttripel, wovon es drei Typen gibt: Typ, Argument und nächste Instruktion. Zwei Instruktionseinbettungen sind sich also ähnlich, wenn wie möglichst die gleichen Typen, Argumente ($\widehat{=}$ Daten-/Kontrollflussabhängigkeiten) und nächste Instruktionen ($\widehat{=}$ Kontrollflussabhängigkeiten) besitzen. Die Einbettungen der anderen Programmelemente sind sich dementsprechend ähnlich, wenn sie sich aus ähnlichen Instruktionseinbettungen summieren. Zwei Methoden, die beide Getter-Funktionalitäten umsetzen, haben die gleichen Instruktionen, wodurch die gewichtete Summe über die Instruktionseinbettungen auch gleich sind. Bezeichner werden durch Platzhalter ersetzt und ignoriert. Obwohl die beiden Getter-Methoden die gleiche Einbettung auf Funktionsebene besitzen, müssen sie nicht unbedingt zur gleichen Anforderung gehören. Daraus folgt, dass auch bei IR2Vec die Ähnlichkeitseigenschaft zu implementierungsnah umgesetzt ist.

Tabelle 4.2 fasst die Eigenschaften der Quelltexteinbettungsverfahren zusammen. Bezüglich der gewünschten Eigenschaften von Quelltexteinbettungen erfüllt kein Verfahren die gewünschte Ähnlichkeitseigenschaft, da vorrangig die Kontroll- und Datenflüsse innerhalb einer Methode kodiert werden. Bei inst2Vec und IR2Vec werden darüber hinaus die natürlichsprachigen Elemente, also die Bezeichner, durch Platzhalter ersetzt. Das macht die Rückverfolgbarkeit mit Anforderungen unmöglich. Hier besteht also noch Verbesserungspotential.

4.3.3 Verfahren zur Erzeugung von Quelltexteinbettungen

Im letzten Abschnitt wurde festgestellt, dass die bloße Anwendung eines existierenden Quelltexteinbettungsverfahrens natürlichsprachige Elemente nicht ausreichend berücksichtigt. Im Folgenden werden hierzu neue Verfahren diskutiert.

4.3.3.1 Repräsentationsmöglichkeiten

In der Anforderung-zu-Quelltextrückverfolgbarkeit gibt es mehrere Optionen, zu welchen Quelltextelementen eine Rückverfolgbarkeitsverbindung hergestellt werden soll, das heißt auf welcher Granularitätsebene die repräsentierten Quelltextelementen liegen. Ein Programm ist eine Zusammenstellung von verschachtelten Quelltextelementen. Grundsätzlich

ist also eine Zuordnung von Anforderungen zu Instruktionen, Ausdrücken, Methoden, Klassen oder Paketen denkbar, aber nicht alles ist für die Rückverfolgbarkeit gleichermaßen geeignet.
Instruktionen liegen im Vergleich zu den anderen Elementen auf der niedrigsten Abstraktionsebene. Sie beschreiben grundlegende mathematische und logische Operationen oder Sprünge. Solche Operationen und Sprünge lassen sich schwer den Anforderungen zuordnen, da der Unterschied ihrer Abstraktionsebenen zu groß ist. Grundlegende Operationen und Sprünge sind Implementierungsdetails, die nicht in den Anforderungen beschrieben werden.
Ausdrücke liegen bezüglich der Abstraktionsebene zwischen Instruktionen und Methoden. Auf dieser Ebene werden zum Beispiel Schleifen iteriert, Bedingungen abgefragt, Hilfsvariablen deklariert oder auf Datenstrukturen zugegriffen. Die Funktionalität, die durch einen einzelnen Ausdruck umgesetzt wird, ist zu feingranular für die Zuordnung zu Anforderungen. Es gibt zum Beispiel verschiedene Schleifenarten - eine Anforderung gibt jedoch keine Hinweise darauf, welche Schleife verwendet werden soll. Wie in Abschnitt 4.3.1 diskutiert, können Methodenaufrufe, welche auch Ausdrücke sind, bezüglich der Ähnlichkeitseigenschaft hilfreich auf sein. Für die Rückverfolgbarkeit ist es jedoch nicht sinnvoll, Anforderungen nur mit Methodenaufrufsausdrücken zu verbinden. Anforderungen werden nicht ausschließlich durch Methodenaufrufsausdrücke implementiert. Zuverlässige Rückschlüsse zur Anforderung sind anhand eines einzelnen Ausdrucks also nicht möglich, jedoch können Ausdrücke wie etwa Methodenaufrufe bereits Hinweise auf die Anforderungen geben. Daraus folgt, dass die Betrachtung mehrerer Ausdrücke zusammen einen größeren Erfolg der Rückverfolgbarkeit verspricht.
Methoden sind Mengen zusammengehöriger Ausdrücke. Sie fassen die Ausdrücke zusammen und setzen komplexere Funktionalitäten um. Methoden können dadurch als Ganzes einen abstrakteren „Sinn“ umsetzen, der die Semantik einer Anforderung zu kodieren kann. Die Abstraktionsebene der Methoden ist ausreichend, um eine Zuordnung zu Anforderungen zu ermöglichen.
Auch Klassen können zu Anforderungen zugeordnet werden. Sie bzw. ihre Instanzen bilden die Objekte, die im umzusetzenden System interagieren und orientieren sich an den Anforderungen. Eine Anforderung über Bankkonten kann beispielsweise durch eine Klasse „Konto“ modelliert werden. Die Rückverfolgbarkeit mit Klassen ist folglich auch eine valide Option.
Pakete enthalten meistens ein Vielzahl von Klassen. Da Anforderungen bereits durch eine einzige Klasse implementiert werden können, kommt es vor, dass Pakete viele zugeordnete

„Mit ausreichend viel Geld können die Waren an der Kasse gekauft werden.“

```
public class Kasse {

    public Ware kaufen(double geld, Ware w) {
        if geld >= w.getPreis() {
            return w;
        else {
            return null;
        }
    }
}
```

```
...
%compare = fcmp ge double %a, %b
br i1 %compare, label %THEN, label %ELSE
...
```

Abbildung 4.2: Eine Anforderung und eine mögliche Implementierung inklusive eines Ausschnittes der LLVM-Zwischensprache.

Anforderungen besitzen. Für eine einzelne Anforderung kann dadurch die Zuordnung erschwert werden, da die Informationen in einem Paket durch die Implementierungen anderer Anforderungen verrauscht werden. Außerdem muss bedacht werden, dass der praktische Nutzen bei Paketen sinken kann: Eine Anforderung-zu-Paket-Verbindung bedeutet nicht unbedingt, dass die Anforderungen mit allen Klassen im Paket in Beziehung steht. Falls die Klasse identifiziert werden soll, muss folglich zusätzlicher Aufwand betrieben werden. Aus diesen Gründen sind Pakete zu grobgranular für die Rückverfolgbarkeit.
Um die verschiedenen Granularitätsebenen zu illustrieren, ist in Abbildung 4.2 eine Beispielanforderung aufgeführt. Daran lässt sich erkennen, dass einzelne Instruktionen wie zum Beispiel diejenigen aus der LLVM-Zwischensprache zu implementierungsnah für eine Rückverfolgbarkeit ist. In der ersten Beispielinstruktion lässt sich nur herauslesen, dass eine Fließkommazahl verglichen wird. Anhand dieser einzelnen Information lässt sich nicht die zugehörige Anforderung ermitteln. Auf der Ausdrucksebene wird im Beispiel ebenfalls ein Vergleich durchgeführt und anhand des Ergebnisses ein Sprung vorgenommen. Durch die Bezeichner kann zwar abgeleitet werden, dass hier Geld mit einem Preis verglichen wird, doch diese Information alleine genügt noch nicht, um eine zuverlässige Verbindung zur Anforderung auf der rechten Seite herzustellen - die Anforderung beschreibt einen Kaufvorgang, der über den Preisvergleich hinausgeht. Andere Ausdrücke wie „return null“ lassen überhaupt keine Rückschlüsse auf die Anforderung zu. Erst durch Betrachtung auf der Methodenebene wird die Zuordnung deutlich. In der Methodensignatur ermöglichen die Bezeichner „kaufen“, „Ware“ und „Geld“ eine zuverlässigere Zuordnung. Auf der Klassenebene ist der Klassenname „Kasse“ ebenfalls hilfreich für die Abbildung auf die Anforderung. Eine Abbildung auf der Klassenebene ist bereits möglich, indem die Zuordnung von der Abbildung der enthaltenen Methoden abgeleitet wird.

Zusammenfassend sind Methoden und Klassen also geeignete Kandidaten, um sie einer Anforderung zuzuordnen. Die anderen besprochenen Quelltextelemente sind zu grob- oder feingranular und versprechen daher weniger zuverlässige Abbildungen. Außerdem sind die Klassen- und Methodenebenen auch die in der Literatur verwendeten Abstraktionsebenen für die Anforderung-zu-Quelltextrückverfolgbarkeit.

4.3.3.2 Methodeneinbettungen

Wie im vorigen Abschnitt erläutert, sollen Quelltexteinbettungen Klassen oder Methoden repräsentieren. Im diesem Abschnitt werden zunächst die Möglichkeiten für Methodeneinbettungen besprochen. Eine Methode hat eine Signatur, einen Rumpf und einen Kommentar. Im Folgenden wird analysiert, wie aus diesen Elementen Einbettungen erzeugt und für die Rückverfolgbarkeit genutzt werden können.

Methodensignatur
Methodensignaturen bestehen aus einem Namen, einem Rückgabetypen und Parameter. Der Methodenname ist ein natürlichsprachige Bezeichner und beschreibt normalerweise die Funktionalität der Methode. In Quelltextausschnitt 4.1 macht der Methodenname zum Beispiel deutlich, dass eine Suchfunktionalität umgesetzt wird. Im Quelltext tragen Bezeichner (und Kommentare) die Semantik, anhand dessen das Quelltextelement einer Anforderung zugeordnet werden kann. Daher sind Methodennamen ein guter Anhaltspunkt für die Abbildung. Der Methodennamen ist aber auch nicht der einzige Bezeichner in der Signatur: Parameternamen sind ebenfalls Bezeichner. Der Rückgabetyp und die Parametertypen können als Bezeichner aufgefasst werden. Diese Elemente tragen also ebenfalls Semantik in sich und können weitere Hinweise für die Zuordnung liefern. Im Beispiel wird erst durch den Rückgabetyp „Buch“ und die Parameternamen „titel“ vermittelt, womit und wonach gesucht wird. Der Methodenname „suchen“ ist sehr generisch und ohne Parameter und Rückgabetyp weniger zuverlässig bei der korrekten Zuordnung, da es potentiell mehrere Anforderungen über verschiedene Suchfunktionen geben kann. Die Parameter und

Quelltextausschnitt 4.1: Methode über eine Buchsuche.

```
public class Student  {
  public Buch suchen(String titel) {
    List<Buch> ergebnis = new ArrayList<Buch>();
    for b in this.auswahl:
      if b.match(titel) {
        ergebnis.add(b);
      }
    return ergebnis;
  }
}
```

Quelltextausschnitt 4.2: Alternative Implementierung der Methode über eine Buchsuche.

```
public class Student  {
  public Buch suchen(int id) {
    String buchTitel = idTitelMap.get(id);
    List<Buch> ergebnis = new ArrayList<Buch>();
    for b in this.auswahl:
      if b.match(buchTitel) {
        ergebnis.add(b);
      }
    return ergebnis;
  }
}
```

der Rückgabetyp verbessern jedoch nicht immer die Zuordnung. Beispielsweise sind primitive Typen sehr implementierungsnah. Ob ein Parameter ein „int“ oder „double“ ist, wird nicht in der Anforderung beschrieben, daher sind diese Typen anders als der Rückgabetyp „Buch“ nicht hilfreich. Auch ist vorstellbar, dass der Titel wie in Quelltextausschnitt 4.2 durch eine ID repräsentiert wird, die erst in einem zweiten Schritt auf den Titel abgebildet wird. Die Methode würde dadurch einen viel generischeren Parameter „id“ statt „titel“ besitzen, der weniger ähnliche Semantik mit der Anforderung trägt. Nichtsdestotrotz besteht die Möglichkeit, dass sowohl Methodenname als auch Parameter und Rückgabetypen die Semantik beinhalten können, die die Semantik ihrer Anforderungen widerspiegelt und somit eine Abbildung zur korrekten Anforderung ermöglicht.
Allgemein besteht bei Bezeichnern das Problem, dass sie frei durch den Entwickler gewählt werden können. Es kann also vorkommen, dass Bezeichner ausgesucht werden, die weniger semantisch ähnlich mit den Anforderungen sind, weil sie zum Beispiel wie „suchen“ sehr allgemein sind. Dadurch wird die Nützlichkeit der Bezeichner maßgeblich durch die Wahl des Entwicklers bestimmt. Komplett ohne natürlichsprachige Elemente wie Bezeichner ist die Rückverfolgbarkeit mit Anforderungen wiederum nicht möglich, wie bereits in Abschnitt 4.3.1 erläutert wurde.

Für die Erzeugung von Einbettungen aus der Methodensignatur sind verschiedene Verfahren denkbar. Die offensichtliche Option ist die Abbildung der Bezeichner mit Hilfe eines existierende Worteinbettungsverfahren. Jedoch muss beachtet werden, dass Bezeichnern oft zusammengesetzte Wörter sind. Da Worteinbettungsverfahren nur einzelne Wörter einbetten, muss entweder ein neues neuronales Netz für Bezeichner trainiert werden oder die Teilbezeichner müssen einzeln eingebettet und anschließend aggregiert werden. Diese beiden Vorgehensweisen sind im Grunde die gleichen Optionen, die bereits für Anforderungen in Abschnitt 4.2.3 besprochen wurden. Die Aggregationsmöglichkeiten, die

für die Anforderungsseite in Abschnitt 4.2.3.3 erörtert wurden - Durchschnitt, gewichtete Summe, Konkatenation oder repräsentativer Vektor - können auch für Bezeichner eingesetzt werden. Die Teilwörter eines Bezeichners bilden jedoch meistens keinen Satz, wie es in Anforderungen der Fall ist. Parameter- und Klassennamen sind oft zusammengesetzte Nomen, ggf. mit Adjektiv, das heißt Nominalphrasen. Methodennamen sind meistens Verben, ggf. mit einem Nomen, die zusammen eine Verbalphrase bilden. Die Phrasen können allerdings unvollständig sein, da Wortarten wie Präpositionen oft in den Bezeichnern ausgelassen werden. Bezüglich der Aggregationsmöglichkeiten ist dieser Umstand jedoch weniger signifikant: Wie in Abschnitt 4.2.1 diskutiert, tragen vor allem Verben und Nomen mehr Semantik als andere Wortarten. Das Fehlen von Präpositionen, Artikeln, etc. in den Bezeichnern verursacht somit keinen großen Informationsverlust. Des Weiteren ist es möglich, aus Methodensignaturen künstlich einen vollständigen Satz zu erzeugen, indem die Nominalphrase des Klassennamens mit der Verbalphrase des Methodennamens kombiniert wird. Auch semantisch betrachtet macht diese Kombination Sinn: In der Objektorientierung sind die Instanzen der Klasse die Objekte, die später die Operationen, also die Methoden, durchführen. Dadurch kann die (Instanz der) Klasse als semantisch korrektes Subjekt angesehen werden. Das Objekt des Satzes kann in den Parametern und Rückgabetypen enthalten sein. Die Nominalphrasen dieser Elemente können also ebenfalls zum künstlichen Satz hinzugefügt werden. Durch Kombination dieser Elemente kann auf diese Weise ein vollständiger Satz mit Subjekt, Prädikat und Objekt gebildet werden. Für die Methode in Quelltextausschnitt 4.1 könnte dieser Satz „Student suchen Buch [mit] Titel [und] Autor.“ lauten. Wie oben schon erwähnt, enthalten „mit“ und „und“ weniger Semantik als die anderen Wörter, sodass das Fehlen dieser Wörter bei der Aggregation keinen großen Einfluss hat. Die falsche Konjugation von „suchen“ ist ebenfalls kein Problem, da die verschiedenen Konjugationsformen eines Wortes im Einbettungsraum aufgrund der Ähnlichkeitseigenschaft sowieso nah beieinander liegen. Wenn in der Vorverarbeitung eine Lemmatisierung eingesetzt wird, ist die richtige Konjugation sowieso überflüssig. Dennoch muss so ein künstlich erzeugter Satz nicht unbedingt semantisch korrekt sein. Zum Beispiel ist es auch denkbar, die „suchen“-Methode in einer Klasse „Bibliothek“ zu platzieren und den suchenden Studenten als Parameter zu übergeben. Dann wäre die Bibliothek das Subjekt, das den Studenten als Objekt sucht - was semantisch natürlich nicht richtig ist. Bezüglich der Aggregationsmöglichkeiten ohne neuronales Netz ist jedoch dieser Umstand weniger problematisch, da die Reihenfolge der zu aggregierenden Wörter sowieso ignoriert wird. Es ist wichtiger, dass die Bezeichner semantische Ähnlichkeiten zu ihren Anforderungen aufweisen.

Mit den Bezeichnern in der Methodensignatur kann also ein Durchschnitt, eine gewichtete Summe, eine Konkatenation oder ein repräsentativer Vektor gebildet werden. Die Vor- und Nachteile der Optionen bleiben dabei erhalten: Beim Durchschnitt und der gewichteten Summe besteht das Risiko, dass der resultierende Vektor sehr abseits von den ursprünglichen Vektoren liegen kann.

Die Konkatenation erzeugt Vektoren unterschiedlicher Länge, die ggf. durch Standardwerte ohne Semantik aufgefüllt werden müssen. Der repräsentative Vektor kann unter Umständen zu viel Semantik auslassen, die in den anderen Vektoren kodiert waren.

Bezüglich der Aggregation durch eine gewichtete Summe kann die Frage gestellt werden, ob Teile der Methodensignatur stärker gewichtet werden sollten als andere. Eine erste Vermutung ist, dass der Methodenname am wichtigsten ist, da der Name zum einen die Funktion der Methode beschreibt bzw. zusammenfasst und es zum auch Methoden ohne Rückgabe oder Parameter gibt. Außerdem können Rückgabetyp und Parameter nicht hilfreiche primitive Typen besitzen. Aus dem Beispiel in Quelltextausschnitt 4.1 wurde oben jedoch die Schlussfolgerung gezogen, dass der Rückgabetyp „Buch“ und der Parameter „titel“ zur richtigen Zuordnung zur Anforderung beitragen. Der Methodennamen „suchen“ ist sehr generisch und ist daher weniger informativ. Den Methodennamen höher zu gewichten

kann also nur als Heuristik für die Gewichtsfestlegung dienen. Die Heuristik kann auch für die Bestimmung eines Repräsentanten als Aggregationsmöglichkeit verwendet werden (Methodennamen als Repräsentant auswählen).
Die Methodensigantur kann des Weiteren zweistufig aggregiert werden: Einmal über jeweils die Teilbezeichner und danach nochmals über die resultierenden Bezeichnereinbettungen. Wie in Abschnitt 4.2.3.3 erläutert, birgt das mehrstufige Aggregieren ein erhöhtes Risiko, dass der resultierende Vektor sehr abseits von den ursprünglichen Bedeutungen liegt. Daher ist es potentiell sinnvoller, nur einmal über alle Teilbezeichner in der Signatur zu aggregieren.
Bezüglich der Aggregation mit einem neuronalen Netz kann ein Dokumenteneinbettungsverfahren eingesetzt werden. Wie oben beschrieben können aus Methodensignaturen Sätze gebaut werden. Das restliche Vorgehen würde analog wie auf der Anforderungsseite ablaufen (Abschnitt 4.2.3.1). Allerdings muss nicht unbedingt ein Dokumenteneinbettungsverfahren eingesetzt werden, da aus der Methodensignatur nur ein einziger Satz entsteht. Daher kann auch ein Verfahren für Satzeinbettungen verwendet werden. Bei den küsntlichen Sätzen besteht dennoch der Nachteil, dass die Sätze nicht vollständig sind. Im obigen Beispielsatz fehlen die Wörter „mit“ und „und“. Für die Aggregation ohne neuronales Netz ist dieser Umstand kein Problem, da die Wörter wenig Semantik beitragen und eventuell sogar durch eine Stoppwortentfernung sowieso herausgefiltert werden. Bei Satzeinbettungsverfahren wie BERT werden solche Wörter jedoch als Kontext trotzdem genutzt, um die richtige Bedeutung eines Wortes zu bestimmen. Das Fehlen dieser Wörter erschwert hier also die korrekte Abbildung.

Einbettungen aus Methodenbezeichnern beziehen offensichtlich natürlichsprachige Elemente mit ein, somit ist diese Quelltexteinbettungseigenschaft erfüllt. Die Ähnlichkeitseigenschaft fordert, dass Quelltextelemente, die zu ähnlichen Anforderungen gehören, auch ähnliche Quelltexteinbettungen besitzen sollen. Das ist hier erfüllt. Die Quelltexteinbettungen wie oben beschrieben leiten sich aus den Bezeichnern ab. Die Rückverfolgbarkeit stützt sich darauf, dass die zu verbindenden Quelltextteile und Anforderungen eine ähnliche Semantik in ihren natürlichsprachigen Elementen aufweisen. Zwei Quelltextelemente, die zu ähnlichen Anforderungen gehören, müssen folglich transitiv betrachtet auch ähnliche natürlichsprachige Elemente wie Bezeichner besitzen. Die Behandlung von OOV-Fällen hängt von der Behandlung von unbekannten Wörtern in den Bezeichnern ab, was wiederum vom jeweiligen Worteinbettungsverfahren bestimmt wird.

Die Quelltexteinbettungen aus Methodensignaturen sind in Anbetracht der erfüllten geforderten Eigenschaften also ein geeignetes Verfahren für die Quelltextseite.

Methodenrumpf
Eine Methode besitzt außer in der Signatur auch natürlichsprachigen Elemente im Rumpf. Im Methodenrumpf sind Bezeichner lokale Variablen, Attribute, auf die zugegriffen werden oder Methodenaufrufe. Methodenaufrufe werden in Abschnitt 4.3.3.4 und Attribute in Abschnitt 4.3.3.3 genauer beleuchtet. Lokale Variablen speichern Daten, die innerhalb der Methode verwendet werden. Sie sind oftmals Hilfsvariablen wie beispielsweise Schleifeniterationsvariablen. In Quelltextausschnitt 4.1 besteht die Schleifeniterationsvariable sogar nur aus einem Buchstaben. Bezeichner solcher lokalen Variablen würden die Rückverfolgbarkeit nicht unterstützen, da sie Implementierungsdetails darstellen. Es ist jedoch denkbar, dass lokale Variablen auch hilfreicher benannt sein können, das heißt Semantik tragen können, die ähnlich zur verbundenen Anforderung ist. In der alternativen Implementierung in Quelltextausschnitt 4.2 wird erst durch die lokale Variable „buchTitel“ vermittelt, dass die Suche anhand des Buchtitels durchgeführt wird. Diese Information würde der richtigen Zuordnung helfen. Methodenrümpfe sind implementierungsnäher als Methodensignaturen, dadurch tragen die enthaltenen Bezeichner tendenziell weniger hilfreiche Semantik für die Rückverfolgbarkeit als Signaturbezeichner, aber die Chance auf hilfreiche Bezeichner be-

steht trotzdem.
Über die Bezeichner im Rumpf kann ein Durchschnitt, eine gewichtete Summe, ein repräsentativer Vektor und eine Konkatenation gebildet werden. Für die gewichtete Summe und den repräsentativen Vektor ist allgemein nicht eindeutig bestimmbar, welche lokale Variablen semantisch ähnlichere Informationen zu den Anforderungen tragen und damit bevorzugt werden sollen. Es können höchsten Variablen mit wenigen Buchstaben in der Vorverarbeitung herausgefiltert werden, da diese zum Beispiel als Schleifeniterationsvariablen genutzt werden. Ansonsten besitzen die Aggregationsverfahren die bereits in Abschnitt 4.2.3.3 besprochenen Vor- und Nachteile.
Satz- oder Dokumenteneinbettungsverfahren werden auf ganzen Sätzen trainiert. Diese Verfahren sind hier weniger geeignet, da der Methodenrumpf keine Sätze bildet.

Methodenkommentare
Methodenkommentare beschreiben in natürlicher Sprache, welche Funktionalitäten eine Methode implementiert. Dadurch können sie semantische Informationen beinhalten, die für die Rückverfolgbarkeit hilfreich sind. Kommentare haben im Gegensatz zu Bezeichnern noch den Vorteil, dass sie in ganzen Sätzen formuliert sein können. Damit sind sie Anforderungen von der Form her ähnlicher als Bezeichner, was eine bessere Rückverfolgbarkeit verspricht. Die Beschreibung in den Kommentaren ist abstrakter als die Implementierung, falls nicht jedes Implementierungsdetail dokumentiert wird. Das kommt der Rückverfolgbarkeit zu gute, da Anforderungen zumeist abstrakter als die Implementierung formuliert sind. Der Kommentar kann aber auch Implementierungsdetails dokumentieren, wie etwa das Verhalten bei Ausnahmen (engl. Exceptions). Wird für die Methode auch der Rumpf miteinbezogen, ist in diesem Fall der Kommentar zwar nicht hilfreich, aber schadet auf der anderen Seite zumindest auch nicht, da die Informationen nur dupliziert wurden. Außerdem könne Ausnahmebeschreibungen in der Vorverarbeitung herausgefiltert werden, falls sie im Kommentar durch vordefinierte Syntaxwörter gekennzeichnet werden. In der Regel wird im Methodenkommentar aber nicht zu sehr in die Implementierungsdetails wie lokale Variablen oder Schleifen eingegangen, sondern eher die gesamte Funktion der Methode beschrieben. Dadurch beinhalten sie Informationen, die semantisch ähnlich zu den verbundenen Anforderungen sein können, die hilfreich für die Rückverfolgbarkeit sind.

Kommentare können aus mehreren Sätzen in natürlicher Sprache bestehen. Insofern sind sie von der Form her mit Anforderungen gleich und es stehen die gleichen Optionen wie bei Anforderungseinbettungen (Abschnitt 4.2.3) zur Verfügung, um aus einem Kommentar eine Einbettung zu erzeugen: Nutzung von Dokumenteneinbettungsverfahren oder Aggregation durch zum Beispiel Durchschnittsbildung. Die diskutierten Vor- und Nachteile aus dem Anforderungsanalysekapitel bleiben dabei erhalten. Für die Vorverarbeitung sollte jedoch bei Kommentaren zusätzlich eine Trennung der Binnenmajuskelschreibweise durchgeführt werden, da Kommentare im Gegensatz zu Anforderungen oft Bezeichner referenzieren.
Kommentare stellen nur zusätzliche Dokumentation für die Implementierung dar, folglich sind sie oft nicht für jede Methode vorhanden. Aus diesem Grund ist die ausschließliche Verwendung von Kommentareinbettungen in der Praxis nicht ausreichend, um alle implementierten Anforderungen abzudecken. Allerdings können Kommentare, falls vorhanden, als zusätzliche Informationsquelle ergänzend zu zum Beispiel Methodensignaturen genutzt werden. Um Methodeneinbettungen mit ihren Kommentareinbettungen zu aggregieren, können die Möglichkeiten in Abschnitt 4.2.3.3 angewendet werden. Bei der gewichteten Summe stellt sich die Frage, wie Kommentar und Methode zueinander gewichtet werden sollen. Für die Anforderung-zu-Quelltextrückverfolgbarkeit ist es hilfreich, wenn dasjenige Element stärker gewichtet wird, welches semantisch ähnlichere Informationen beinhaltet wie die Anforderungen. In der Regel ist der Methodenkommentar abstrakter als der Methodenrumpf, da nicht jedes Implementierungsdetail der Methode erwähnt wird. Im Vergleich zur Signatur muss der Kommentar jedoch nicht abstrakter sein. Beide Elemente beschrei-

ben die Funktionalität der Methode. Allerdings ähneln sich Kommentare bezüglich ihrer Form den Anforderungen mehr als Bezeichner. Tendenziell sollte also der Kommentar ein höheres Gewicht besitzen. Falls ein Repräsentant zwischen Methode und Kommentar ausgewählt werden muss, würde die Wahl folglich auf den Kommentar fallen.
Das bisher erläuterte Vorgehen berechnet Einbettungen für Methoden und Kommentare, indem ihre Wörter bzw. Sätze auf Wort- bzw. Satzeinbettungen abgebildet und anschließend aggregiert werden. Danach wird nochmals über die resultierenden Methoden- und Kommentareinbettungen aggregiert. Bei der Durchschnittsbildung und der gewichteten Summe bestand das Problem, dass der resultierende Vektor sehr abseits von den ursprünglichen Vektoren liegen kann. Durch mehrstufige Aggregieren steigt, wie in Abschnitt 4.2.3.3 erklärt, dieses Risiko: Der Durchschnitt von abseits liegenden Vektoren liegt wahrscheinlich noch weiter abseits. Es ist als Alternative auch möglich, direkt alle Wörter von den Methoden und Kommentaren auf Worteinbettungen abzubilden und danach nur einmal zu aggregieren. Satzeinbettungen können hier nur für Kommentare und Methodensignaturen eingesetzt werden. Für Methodenrümpfe ist das nicht möglich, hier muss ein anderes Verfahren genutzt werden.

Sichtbarkeit
Methoden besitzen auch eine Sichtbarkeit. Die Sichtbarkeit ist zwar kein natürlichsprachiges Element, aber daraus können Schlussfolgerungen auf den Grad der Implementierungsnähe gezogen werden: Private Methoden können nur innerhalb einer Klasse genutzt werden und beinhalten oftmals Hilfsfunktionalitäten. Diese gehören meistens zu den Implementierungsdetails, die nicht in den Anforderungen beschrieben werden. Das Herausfiltern von privaten Methoden kann also dazu führen, das Rauschen zu verringern. Natürlich ist es trotzdem denkbar, dass private Methoden zumindest teilweise Funktionalitäten umsetzen, die in Anforderungen beschrieben werden. Hierzu kann folgende Heuristik helfen: Die privaten Methoden können von einer öffentlichen oder einer anderen privaten Methoden aufgerufen werden. Wenn bei der öffentlichen Methode der Methodenrumpf miteinbezogen wird, wird auch die Signatur der privaten Methode durch den Aufruf in der öffentlichen Methode berücksichtigt. Der Rumpf der privaten Methode wird ignoriert, da hier die Wahrscheinlichkeit nochmals größer als bei der Signatur ist, dass sie zu implementierungsnah ist. Für die private Methode wird keine eigene Methodeneinbettung generiert, sie wird nur indirekt durch die öffentliche Methode inkludiert. Wenn die private Methode von einer anderen privaten Methode aufgerufen wird, steigt die Wahrscheinlichkeit, dass sie tatsächlich nur implementierungsnahe Details umsetzt, da sie potentiell die Hilfsmethode einer Hilfsmethode ist. In diesem Fall wird die private Methode komplett ausgelassen. Auf diese Weise können Informationen aus den privaten Methoden mit verringertem Risiko auf Implementierungsdetails integriert werden.

Alles in allem sind Methodensignaturen und -kommentare die hervorstehenden Kandidaten zur Bildung einer Methodeneinbettung. Beide Elemente beschreiben und fassen die Funktionalitäten einer Methode zusammen. Methodenrümpfe sind implementierungsnäher als Signaturen und Kommentare und haben ein höheres Risiko, irrelevante Informationen für die Rückverfolgbarkeit zu enthalten. Ob sie trotzdem hilfreich sind, muss empirisch evaluiert werden. Private Methoden haben ebenfalls ein höheres Risiko, nur Implementierungsdetails umzusetzen. Deswegen sollten sie heuristisch nur indirekt über die Aufrufe in Methodenrümpfen von öffentlichen Methoden inkludiert werden.

4.3.3.3 Klasseneinbettungen

Im letzten Abschnitt wurden Möglichkeiten beleuchtet, um pro Methode eine Einbettung zu generieren. Wie in Abschnitt 4.3.3.1 diskutiert, sind Quelltexteinbettungen auf Klassenebene ebenfalls für die Anforderung-zu-Quelltextrückverfolgbarkeit geeignet.

Für die Klasseneinbettungen kann man die Klasse wie in der Informationsrückgewinnung als Fließtext auffassen und daraus eine Einbettung erzeugen. Dies wird im nächsten Abschnitt analysiert. Die Klassenelemente (Klassenname, Methoden, Attribute, etc.) können jedoch auch separat betrachtet und eingebettet werden. Diese werden danach einzeln beleuchtet.

Klasse als Text
Eine Klasse kann als natürlichsprachiger Text angesehen werden. Von einer Klasse werden dazu alle Syntaxwörter der Programmiersprache entfernt, sodass nur noch die natürlichsprachigen Bezeichner und Kommentare übrig bleiben. Dadurch werden auch alle Bezeichner von Methoden inkludiert. Anschließend werden die Wörter mit einem existierenden Worteinbettungsverfahren abgebildet und danach aggregiert (siehe Abschnitt 4.2.3.3). Bei der Durchschnittsaggregation sind alle Wörter gleich gewichtet. In einer Klasse sind jedoch nicht alle Wörter bezüglich der Rückverfolgbarkeit gleich relevant. Klassennamen sind beispielsweise wichtiger als Namen von lokalen Variablen, da Letztere implementierungsnäher und damit höchstwahrscheinlich bei der Verbindung zu Anforderungen weniger hilfreich sind.
Bei der Aggregation mit einer gewichteten Summe kann der Klassennamen bevorzugt werden. Wie weiter oben bereits analysiert, sind Methodensignaturen abstrakter als ihr Methodenrumpf, daher sollten Erstere ein höheres Gewicht besitzen. Nicht alle Methoden sind gleich wichtig bezüglich der Rückverfolgbarkeit. Es kann abstraktere, aber auch implementierungsnähere Methoden ohne korrespondierende Anforderung geben. In Programmiersprachen mit Vererbung gibt es als abstrakt gekennzeichnete Methoden ohne Implementierung. Dies kann als ein Indiz dafür dienen, dass solche Methoden wahrscheinlich keine Implementierungsdetails darstellen und eine Rückverfolgbarkeitsverbindung besitzen. Daher sollten diese Methoden tendenziell ein höheres Gewicht haben. Eine private Sichtbarkeit ist ein Hinweis darauf, dass die Methode wahrscheinlich Implementierungsdetails enthält. Solche Methoden sollten also niedriger gewichtet werden. Für die restlichen Methoden ist es nicht zuverlässig bestimmbar, ob sie eine Rückverfolgbarkeitsverbindung haben. Daher ist die obige Vorgehensweise nur heuristisch zu betrachten.
Für die Aggregation durch Auswahl eines Repräsentanten können die gleichen Überlegungen wie bei der Gewichtung vorgenommen werden, um Elemente nach ihrer Nützlichkeit für die Rückverfolgbarkeit zu beurteilen. Trotzdem besitzt dieses Aggregationsverfahren das bekannte Problem, dass der Repräsentant potentiell nicht alle relevanten Informationen in der Klasse kodieren kann.
In der Variante mit der Nutzung eines existierenden Dokumenteneinbettungsverfahrens stößt man auf das Problem, dass sich die Menge an Bezeichnern in der Klasse nicht aus ganzen, vollständigen Sätzen zusammensetzt. Dadurch wird die Abbildung verfälscht, da die Modelle auf ganzen Sätzen trainiert wurden.

Die Eigenschaft der Berücksichtigung von Bezeichnern ist bei den obigen Verfahren erfüllt. Bezüglich der Ähnlichkeitseigenschaft lässt sich Folgendes feststellen: Die Einbettungen werden im Kern aus den Bezeichnern generiert. Daraus folgt, dass die Quelltexteinbettungen ähnlich sind, falls ihre Bezeichner ähnlich sind. Die Rückverfolgbarkeit stützt sich darauf, dass Anforderungen und ihre korrespondierenden Quelltextelemente semantisch ähnlich sind. Dadurch sind ähnliche Quelltexteinbettungen auch mit ähnlichen Anforderungen verbunden. Die Ähnlichkeitseigenschaft ist also erfüllt, jedoch kann sie stark verrauscht sein, da nicht alle Bezeichner semantisch wichtige Informationen bezüglich der Rückverfolgbarkeit tragen. Die OOV-Behandlung hängt auch hier vom zugrundeliegenden Worteinbettungsverfahren ab.

Aggregation der Methodeneinbettungen
Statt der Auffassung als Fließtext können die Methoden in der Klasse getrennt betrachtet werden: Es werden zunächst Methodeneinbettungen berechnet, die danach zu einer

einzigen Klasseneinbettung aggregiert werden. Auch hier werden offensichtlich Methoden inkludiert. Die Methodeneinbettungen können mit den Möglichkeiten in Abschnitt 4.3.3.2 erzeugt werden. Aggregationsmöglichkeiten wurden bereits im Anforderungskapitel in Abschnitt 4.2.3.3 besprochen. Die Vor- und Nachteile bei der Durchschnittsbildung und der Konkatenation bleiben hier erhalten. Bei der gewichteten Summe können Methoden anhand ihrer Sichtbarkeit und Abstraktheit gewichtet werden. Öffentliche Methoden enthalten mit größerer Wahrscheinlichkeit nützlichere Informationen für die Rückverfolgbarkeit als private. Abstrakte Methoden haben eine kleinere Wahrscheinlichkeit, Implementierungsdetail zu sein. Diese beiden Methodenarten sollten also höher gewichtet werden. Falls für die Auswahl eines Repräsentanten nur eine einzige Methode ausgewählt werden soll, ist die Entscheidung bei mehreren öffentlichen oder abstrakten Methoden nicht eindeutig. Es kann a priori nicht weiter differenziert werden, welche Methode für die Rückverfolgbarkeit nützlicher ist.

Klasseneinbettungen wie oben beschrieben beziehen natürlichsprachige Bezeichner mit ein und erfüllen die Ähnlichkeitseigenschaft, da das Verfahren, ähnlich wie bei den Methodeneinbettungen erklärt, im Kern auf Bezeichnern arbeitet. Bei der Aggregierung gehen Informationen verloren (siehe Abschnitt 4.2.3.3) - hinzukommt, dass bei der obigen Methode mehrfach aggregiert wird. Dadurch erhöht sich erneut das Risiko, dass der resultierende Klassenvektor abseits von den ursprünglichen Wortbedeutungen liegt. Die Ähnlichkeitseigenschaft bei solchen Klasseneinbettungen ist also in der Praxis wahrscheinlich verrauscht. Wie gut die Rückverfolgbarkeit damit funktioniert, muss empirisch evaluiert werden. Die OOV-Eigenschaft hängt vom Worteinbettungsverfahren ab.
Für die Methodeneinbettungen wurde diskutiert, dass Methodenkommentareinbettungen mit Methodensignatureinbettungen aggregiert werden können, um die Kommentare miteinzubeziehen. Wenn Kommentare für Kommentareinbettungen, Kommentareinbettungen mit Methodeneinbettungen und anschließend nochmal pro Klasse alle Methodeneinbettungen aggregiert werden, wird insgesamt drei Mal aggregiert. Es ist auch möglich, die Kommentareinbettungen nicht zuerst mit den Methodeneinbettungen zu aggregieren, sondern direkt bei der Aggregation für die Klasseneinbettung miteinzubeziehen, damit über die Kommentare nur zwei Mal aggregiert wird und das Risiko verringert wird, dass die resultierenden Vektoren abseits liegen.

Klassenname
Ein Klassenname ist ein natürlichsprachiger Bezeichner und kann somit durch ein existierendes Einbettungsverfahren, analog wie bei den Methodennamen, auf eine Einbettung abgebildet werden. Der Klassenname ist tendenziell abstrakter als die anderen Bezeichner wie etwa Methodennamen und beschreibt in der Regel die gesamte Funktionalität der Klasse. Dadurch ist die Wahrscheinlichkeit höher, dass der Klassenname eine ähnlichere Semantik wie die dazugehörigen Anforderungen trägt. Der Klassenname ist wahrscheinlich der wichtigste Bezeichner in der Klasse. Jedoch ist die alleinige Betrachtung des Klassennamens nicht genügend, um alle Anforderungen zu Klassen zuzuordnen. Die beschriebenen Funktionalitäten aus den Anforderungen werden werden durch Methoden implementiert. Deswegen ist es sinnvoll, die Methoden auch bei Klassennameneinbettungen miteinzubeziehen.
Für die Kombination von Klassennamen und Methoden stehen verschiedene Optionen zur Verfügung. Klassennamen können, wie in Abschnitt 4.3.3.2 diskutiert, zu den Methodeneinbettungen hinzugefügt werden, um einen Satz mit Subjekt-Prädikat-Objekt zu bilden, welcher näher an normalen natürlichsprachigen Sätzen liegt. Klassennamen können aber auch separat auf eine Einbettung abgebildet werden uns erst bei der Aggregation der Methoden eingesetzt werden. Dadurch wird dem Klassennamen beim Durchschnitt automatisch mehr Gewicht verliehen, als wenn es bei den Methodeneinbettungen einfließen würde. Außerdem kann die Gewichtung zwischen Klassennamen und Methoden bei einer

gewichteten Summe auf diese Weise einfach durchgeführt werden. Die Priorisierung des Klassennamens kann vorteilhaft sein, da er abstrakter als Methoden ist.

Der Klassennahme beinhaltet also hilfreiche Informationen für die Rückverfolgbarkeit und sollte genutzt werden.

Klassenkommentare
Klassenkommentare beschreiben die Funktionen einer Klasse auf abstrakterer Ebene als die Implementierung. Genau wie Methodenkommentare dienen sie also als Indikator für die Funktionen der umgesetzten Anforderungen und helfen dadurch bei der Rückverfolgbarkeit. Des Weiteren sind Klassenkommentare natürlichsprachige Elemente mit ganzen Sätzen und somit den Anforderungen ähnlicher als Quelltext. Die alleine Betrachtung des Klassenkommentars ohne Methoden ist nicht sinnvoll, da ansonsten die Anforderungen, die in Methoden implementiert sind, außen vor gelassen werden. Außerdem besitzt nicht unbedingt jede Klasse einen Klassenkommentar.

Um Klassenkommentare zu integrieren, können die gleichen Vorgehensweise verwendet werden, die bereits für Methodenkommentare in Abschnitt 4.3.3.2 diskutiert wurden: Die berechneten Klassenkommentareinbettungen können zusammen mit den Methodeneinbettungen durch einen Durchschnitt oder einer gewichteten Summe aggregiert werden, um die Klasseneinbettung zu erzeugen. Der Kommentar sollte bei einer Gewichtung ein höheres Gewicht tragen, da es tendenziell die Funktion der Klasse abstrakter beschreibt - hier besteht das gleiche Verhältnis von Methodenkommentar zu Methode. Die anderen Eigenschaften, die von den Methodeneinbettungen abgeleitet werden, bleiben von der Inklusion von Klassenkommentaren unberührt.

Attribute
Attribute besitzen einen Typ, einen Bezeichner, und eventuell eine Initialisierung oder einen Kommentar. Im Folgenden wird analysiert, wie und ob Attribute bei der Rückverfolgbarkeit helfen können. Eine Anforderung kann theoretisch nur mit Klassen, die ausschließlich aus Methoden bestehen und keine Attribute besitzen, implementiert werden. Andersherum ist das nicht möglich - Anforderungen werden also primär durch Methoden, nicht durch Attribute implementiert. Daher ist die Betrachtung von Attributen nur in Kombination mit Methoden sinnvoll. Hieran schließt sich die Frage an, ob Attribute zusätzliche und nützliche Informationen für die Rückverfolgbarkeit bieten, als wenn nur Methodeneinbettungen verwendet werden.
Auf der einen Seite können Attributbezeichner hilfreich sein: Eine „Bankkonto“-Klasse kann ein Attribut namens „Inhaber“ besitzen, was semantisch aussagekräftig und nützlich für die Rückverfolgbarkeit sein kann. Auf der anderen Seite haben Attribute oftmals primitive Typen und Datenstrukturen als Attributtypen. Diese Typen sind sehr implementierungsnah und erhöhen damit die Wahrscheinlichkeit, dass das Attribut auch ein Implementierungsdetail darstellt. In diesem Fall würden Attribute also nur Rauschen verursachen. Bei anderen Typen besteht jedoch die Chance auf semantikreicheren Informationen, die die Abbildung auf die Anforderungen verbessert.
Weiterhin speichern Attribute im Grunde nur Daten. Die Daten bzw. das Attribut muss also in mindestens einer Methode benutzt werden, ansonsten wäre die Definition des Attributs sinnlos gewesen. Falls die Methodenrümpfe in den Methodeneinbettungen miteinberechnet werden, müssten Attribute also theoretisch nicht mehr zusätzlich berücksichtigt werden. Hier wird die Frage aufgeworfen, ob die indirekte Inklusion durch Methodenrümpfe besser als die getrennte Variante mit expliziten Attributeinbettungen ist. Für die indirekte Variante spricht, dass die Attributinformationen nur dort berücksichtigt werden, wo sie auch verwendet werden. Bei der expliziten Variante kann es zum Beispiel vorkommen, dass ein Attribut in nur einer von zehn Methoden benutzt wird. Durch anschließende Aggregation über alle Attribut- und Methodeneinbettungen kann das Attribut zu viel Gewicht

bekommen und die Klasseneinbettung zusätzlich verrauschen. Jedoch ist auch anzumerken, dass die Attributinformationen bei der indirekten Variante durch andere Methodenrumpfelemente verrauscht werden. Dieser Punkt würde also für die explizite Variante sprechen. Es gibt Vor- und Nachteile, ob Attributinformationen gesondert bzw. überhaupt miteinbezogen werden sollen. Die Nützlichkeit muss in diesem Fall in der Praxis evaluiert werden. Attributkommentare ähneln sich den Anforderungen von der Form her mehr als die Attributbezeichner. Deswegen ist zu erwarten, dass Attributkommentare besser für die Rückverfolgbarkeit funktionieren als die Attribute. Da die Attribute Implementierungsdetails sein können, kann der Attributkommentar theoretisch ebenfalls nur Implementierungsdetails beschreiben. Auch hier muss der Nutzen also evaluiert werden.

Konstruktoren
Ein Konstruktor initialisiert die Attribute der Klasse. Aus diesem Grund überlappen sich viele Informationen zwischen Attributen und Konstruktoren. Jedoch besitzen Konstruktoren noch andere Informationen durch die Signatur und ggf. einen Kommentar. Der Name des Konstruktors ist meistens ein vordefinierter Bezeichner (zum Beispiel in Python) oder der Name ist identisch mit dem Klassennamen (zum Beispiel in Java). Der erste Fall trägt keine nützliche Semantik für die Rückverfolgbarkeit und der zweite Fall ist redundant, falls der Klassenname bereits miteinbezogen wird. Die Parameter des Konstruktors können andere Typen und Namen haben als die zu initialisierenden Attribute. Wenn das Attribut „Student“ heißt, aber der Konstruktor die Parameter „Name“ und „Fachrichtung“ dafür entgegennimmt, tragen die Parameter weitere semantische Informationen, die hilfreich für die Zuordnung zu Anforderungen sein können. Für den Kommentar gilt die gleiche Argumentation wie für Methodenkommentare (Abschnitt 4.3.3.2). Sie sind durch ihre Form und ihr Abstraktionsniveau der Implementierung überlegen und dienen durch die abstrakte Beschreibung des Konstruktors als Indikator für die Zuordnung zu den in Anforderungen beschriebenen Funktionalitäten.
Im Konstruktorrumpf werden die Attribute initialisiert. Dieser enthält dadurch zum Großteil auch die gleichen Informationen wie die Attribute. Jedoch kann es auch vorkommen, dass komplexere Initialisierungen vorgenommen werden, in der weitere benötigte Objekte erstellt oder Berechnungen durchgeführt werden. Diese könnten potentiell auch nützliche Informationen für die Rückverfolgbarkeit liefern. Da Attribute bereits das Risiko beherbergen, irrelevante Implementierungsdetails zu sein, ist bei solchen Objekten und Berechnungen, die für die Attributinitialisierung durchgeführt werden, die Gefahr nochmals größer, irrelevant für die Rückverfolgbarkeit zu sein.

Die Informationen aus dem Konstruktorrumpf sind also wahrscheinlich nicht hilfreich oder redundant, falls Attribute betrachtet werden. Der Kommentar und die Parameter können dagegen helfen.

Superklassifizierer
Eine Klasse kann Superklassifizierer in Form von Oberklassen oder implementierten Schnittstellen besitzen.
Ober- und Unterklassen haben durch die Vererbung gemeinsame Eigenschaften, daher ist es wahrscheinlich, dass beide Klassen mit der gleichen oder einer ähnlichen Anforderungen in Verbindung stehen. Durch Betrachtung der Oberklassen können also potentiell weitere Elemente gefunden werden, die semantisch ähnlich zur korrespondierenden Anforderung sind. Oberklassen, die aus externen Bibliotheken stammen, sind jedoch nicht nützlich, da sie höchstwahrscheinlich nicht in den Anforderungen beschrieben sind. Nur die Integration der Informationen aus nicht externen Oberklassen kann also bei der Rückverfolgbarkeit helfen. Die Oberklasse besitzt einen Klassennamen, Attribute und Methoden. Die Klassennamen können mit den Optionen in Abschnitt 4.2.3.3 aggregiert werden. Für die gewichtete Summe muss diskutiert werden, ob die Oberklasse oder die Unterklasse mehr Einfluss haben soll. Oberklassen sind allgemein abstrakter als Unterklassen. Ist die Oberklasse zu

generisch, ist das für die Rückverfolgbarkeit nicht hilfreich: Beispielsweise kann die Klasse „TextFileReader“ eine Oberklasse namens „FileReader“ besitzen, welcher sehr generisch ist. Auf der anderen Seite kann die Unterklasse auch zu implementierungsspezifisch benannt sein, was ebenfalls nicht die Rückverfolgbarkeit unterstützen würde. Pauschal kann folglich keine Bevorzugung zwischen den Namen der Ober- und Unterklasse vorgenommen werden. Aus dem gleichen Grund ist die Entscheidung eines Repräsentanten zwischen Unter- und Oberklasse als Aggregation schwierig.
Eine Oberklasse hat weiterhin Attribute und Methoden. Im Folgenden werden nur solche Attribute und Methoden betrachtet, die nicht privat sind, denn nur diese sind in den Unterklassen sichtbar. Wie im Abschnitt über Attribute bereits erläutert, können Attribute zu implementierungsspezifisch sein oder auch indirekt über Methodenrümpfe inkludiert werden. Die Nützlichkeit steht hier nicht eindeutig fest. Dies gilt auch für die Attribute der Oberklasse. Falls die Attribute miteinbezogen werden sollen, können sie wie die Attribute der Unterklasse auf Attributeinbettungen abgebildet und danach aggregiert werden. Die Oberklassenmethoden können auf Methodeneinbettungen abgebildet und zusammen mit den Unterklassenmethodeneinbettungen aggregiert werden. Auch hier stellt sich die Frage nach der Gewichtung. Wie bei den Klassennamen können die Oberklassenmethoden oder -attribute in der Theorie zu generisch sein, während die Unterklassenmethoden und -attribute zu implementierungsspezifisch sein können. Es kann also keine klare Favorisierung konstatiert werden - dies muss empirisch ermittelt werden.
Schnittstellen haben einen Namen und abstrakte Methoden. Für die Namen gilt die gleiche Analyse, die bei den Oberklassen vorgenommen wurde. Die Schnittstellenmethoden müssen im Gegensatz bei Oberklassen nicht berücksichtigt werden - sie werden in der jeweiligen Unterklasse implementiert. Die Signatur verändert sich durch die Implementierung nicht und ist daher durch Miteinbeziehung der konkreten Methode in der Unterklasse bereits inkludiert.

Zusammenfassend kann die Ähnlichkeitseigenschaft durch die Superklassifizierer verbessert werden, alle anderen Eigenschaften bleiben unverändert.

Innere Klassen
Klassen können verschachtelte, innere Klassen besitzen. Aus der Verschachtelung lassen sich zwei Schlussfolgerungen ziehen: Die innere Klasse und die äußere Klasse sind gekoppelt. Das erhöht die Wahrscheinlichkeit, dass beide Klassen zur gleichen oder ähnlichen Anforderung gehören bzw. die Integration der Informationen aus der inneren Klasse in die äußere Klasseneinbettung kann potentiell die semantische Ähnlichkeit zur Anforderung steigern. Die zweite Schlussfolgerung ist die Abstraktionsebene: Die verschachtelte innere Klasse ist mit größerer Wahrscheinlichkeit ein Implementierungsdetail, da sie zusammen mit der äußeren Klasse platziert wurde und keine eigenständige (äußere) Klasse bildet. Innere Klassen können eine private Sichtbarkeit besitzen - dies würde darauf hindeuten, dass die sie tatsächlich ein Implementierungsdetail sind. Dies ist jedoch nur ein Indiz - es kann theoretisch auch eine innere Klasse geben, die ein Implementierungsdetail ist und nicht privat ist. Als Kompromiss ist es möglich, bei inneren Klasse nur den Klassennamen und die Methodensignaturen zu berücksichtigen. Die Methodenrümpfe und Attribute sind tendenziell implementierungsnäher - durch das Auslassen dieser Elemente verringert sich das Risiko auf die Inklusion von Implementierungsdetails. Außerdem dienen viele innere Klassen nur zur Datenhaltung, die als Methoden nur Getter und Setter haben. Die Methodenrümpfe sind in diesem Fall sowieso nicht hilfreich.

Um die inneren Klassen abzubilden, können die bereits besprochenen Klasseneinbettungsverfahren für (äußere) Klassen angewendet werden. Um die innere mit der äußeren Klasseneinbettung zu kombinieren, sind die üblichen Aggregationsmöglichkeiten (Abschnitt 4.2.3.3) anwendbar. Die gewichtete Summe ist hierbei dem Durchschnitt vorzuziehen, da beim Ersteren die äußere Klasseneinbettung höher gewichtet werden kann, um die Gefahr auf

Implementierungsdetails durch die innere Klasse zu reduzieren.

Die Einbeziehung von inneren Klassen kann also die Rückverfolgbarkeit verbessern, jedoch eine geeignete Gewichtung zwischen inneren und äußeren Klassen ermittelt werden, um das potentielle Rauschen durch die inneren Klassen zu vermindern.

4.3.3.4 Aufrufabhängigkeiten miteinbeziehen

Bei den Quelltexteinbettungseigenschaften in Abschnitt 4.3.1 wurde festgestellt, dass Aufrufbeziehungen Hinweise darauf geben können, dass Quelltextelemente zur gleichen oder zu ähnlichen Anforderungen gehören. Jedoch wurde auch aufgezeigt, dass Aufrufbeziehungen alleine nicht für die Rückverfolgbarkeit genügen. In den vorigen Abschnitten wurden Klassen- und Methodeneinbettungen vorgestellt. Diese können mit Aufrufabhängigkeiten kombiniert werden.

Methodeneinbettungen mit Aufrufabhängigkeiten
Aufrufabhängigkeiten können als Aufrufgraph veranschaulicht werden, in der die Methoden die Knoten darstellen und die Kanten die Aufrufbeziehungen. Eine Methode besitzt in diesem Graph zwei Arten von Nachbarknoten: Zum einen Methoden, die es selbst aufruft (Aufgerufenennachbar) und zum anderen andere Methoden, von denen die aktuelle Methode aufgerufen wird (Aufrufernachbar). Es stellt sich die Frage, ob beide Arten von Nachbarn als Abhängigkeit miteinbezogen werden sollen. Methoden mit ausschließlich Implementierungsdetails sollten nicht berücksichtigt werden, da sie nicht in Anforderungen beschrieben sind. Die Aufgerufenennachbarn sind potenziell implementierungsnäher als die aktuelle Methode, daher kann es das Rauschen verringern, wenn nur die Aufrufernachbarn verwendet werden. Dies muss jedoch nicht immer zutreffen: Es kann auch vorkommen, dass zum Beispiel eine Anforderung durch zwei Methoden umgesetzt wird, die jeweils die Hälfte der Anforderung implementieren, wobei die erste die zweite Methode aufruft. Betrachtet man bei der ersten Methode nur Aufrufernachbarn, würde man die Informationen aus der zweiten Methode außen vor lassen.
Statt einer lokalen Betrachtung der Nachbarn kann auch die globale Position bzw. Tiefe der Nachbarn im gesamten Aufrufgraphen verwendet werden. Der Aufrufgraph wird dafür als Baum aufgefasst und die Wurzel ist der Eintrittspunkt des Programms. Auch hier kann zunächst angenommen werden, dass Methoden mit großer Tiefe wahrscheinlicher Implementierungsdetails umsetzen, da sie eine lange Aufrufkette zur Wurzel besitzen. Das Herausfiltern von Methoden mit einem Schwellwert der Tiefe könnte folglich das Rauschen durch zu implementierungsnahe Methoden verringern. Jedoch ist auch vorstellbar, dass zum Beispiel die Wurzelmethode bereits eine Hilfsmethode aufruft, die nur Implementierungsdetails beinhaltet. Diese Hilfsmethode hätte eine Tiefe von Eins und würde nicht herausgefiltert werden. Die Auswahl der Nachbarn bzw. Knoten im Graph lässt sich also nicht eindeutig anhand der Tiefe oder der Richtung der Aufrufsbeziehung vorfiltern.
Außer den unmittelbaren Nachbarn können transitiv auch die Nachbarn der Nachbarn, und von ihnen wiederum ihre Nachbarn, usw. miteinbezogen werden. Dadurch können potentiell weitere Methoden gefunden werden, die zur gleichen oder einer ähnlichen Anforderung der aktuellen Methode gehören. Gleichzeitig steigt aber auch das Risiko, dass Methoden miteinbezogen werden, die nicht mehr zur gleichen Anforderung gehören. Auch hier lässt sich a priori nicht angeben, wie groß diese transitive Tiefe sein sollte.

Die Informationen der Nachbarmethoden können auf verschiedene Art und Weisen integriert werden. Die einfachste Methode ist Anwendung einer Aggregationsmethode auf die Vereinigung aller Bezeichner der aktuellen und der Nachbarmethoden. Eine Durchschnittsbildung würde dazu führen, dass die Nachbarmethoden mit mehr Gewicht in die Einbettung einfließen, falls es mehr als eine Nachbarmethode gibt. Wegen der Gefahr, dass nicht alle Nachbarn zu ähnlichen Anforderungen der aktuellen Methode gehören müssen, sollte

Letzteres höher gewichtet werden. Die aktuelle Methode hat, vor allem durch Einbezug des Klassennamens und des Methodenkommentars, die größte Chance, mit der richtigen Anforderung verbunden zu werden. Aufrufbeziehungen können dagegen theoretisch zu einer beliebige anderen Methode in einer beliebigen Klasse bestehen. Die Einbettung der aktuellen Methode und die der Nachbarmethoden sollten folglich separat berechnet und anschließend gewichtet aufsummiert werden, anstatt einen Durchschnitt zu bilden. Die Auswahl eines Repräsentanten ist in diesem Fall nicht sinnvoll, da dadurch wahrscheinlich die aktuelle Methodeneinbettung ausgewählt und die Nachbarmethodeneinbettungen ignoriert werden.

Durch den Einbezug der Nachbarmethoden wird potentiell die Wahrscheinlichkeit bei nicht hilfreich benannten Methoden erhöht, zur richtigen Anforderung zugeordnet zu werden. Es kann zum Beispiel vorkommen, dass die aktuelle Methode einen sehr generischen Namen besitzt. Falls Nachbarn, die zur gleichen Anforderung gehören, einen semantikreicheren Namen tragen, vergrößert sich die Wahrscheinlichkeit, dass die aktuelle Methode trotzdem richtig verbunden wird. Die anderen Eigenschaften der Methodeneinbettungen werden durch die Aufrufabhängigkeiten nicht verändert.

Klasseneinbettungen mit Aufrufabhängigkeiten
Die Aufrufabhängigkeiten können auch auf Klassenebene mit Klasseneinbettungen verrechnet werden. Dabei besteht eine Klassenaufrufabhängigkeit, falls mindestens eine ihrer Methoden eine entsprechende Methodenaufrufabhängigkeit besitzt. Das Vorgehen und die Analyse über die Nachbarknoten und die Tiefe im Aufrufgraphen sind analog wie bei den Methoden aus dem vorigen Abschnitt: Die Aggregation ist bevorzugt eine gewichtete Summe, wobei die aktuelle Klasse ein höheres Gewicht hat als die Nachbarn. Über Aufrufer- und Aufgerufenennachbarn sowie die Tiefe im Aufrufgraphen lassen sich keine allgemeingültigen Filterkriterien ableiten.
Es gibt dennoch Unterschiede, ob Aufrufabhängigkeiten auf Klassen- oder Methodenebene inkludiert werden. Beim Methodenaufrufgraphen kann eine Methode eine andere Methode aus derselben Klasse als Nachbar haben. Wenn zwei Methoden zur selben Klasse gehören, erhöht sich die Wahrscheinlichkeit, dass sie zur gleichen oder zu einer ähnlichen Anforderung gehören. Auf diese Weise können sie durch eine Aufrufbeziehung Informationen austauschen. Auf der Klassenebene würde diese Beziehung nicht existieren, da die Knoten Klassen sind und dies folglich eine Aufrufbeziehung zu sich selber wäre. Es würden durch die Schleife keine neuen Informationen gewonnen werden. Daher sind im Klassenaufrufgraphen alle Nachbarn garantiert andere Klassen. Das hat wiederum zur Folge, dass man im Vergleich zum Methodenaufrufgraphen mit größerer Wahrscheinlichkeit Nachbarn besitzt, die nichts Hilfreiches zur Verbindung mit der korrekten Anforderung beitragen. Um dies auszugleichen, muss das Gewicht der aktuellen Klasse tendenziell höher gewählt werden als beim Methodenaufrufgraphen. Auch wird die Tiefe bezüglich der transitiven Berücksichtigung der Nachbarn niedriger ausfallen als bei den Methoden.

Insgesamt kann die Miteinbeziehung der Aufrufabhängigkeiten die Rückverfolgbarkeit unterstützen. Dabei verspricht die gewichtete Summe die beste Aggregation der Aufrufnachbarn.

4.3.3.5 Datenflussabhängigkeiten miteinbeziehen

Außer Kontrollflussabhängigkeiten wie dem Methodenaufruf kann eine Methode auch Datenflussabhängigkeiten besitzen. Eine Datenflussabhängigkeit besteht zu einer anderen Methode, wenn sie auf dem gleichen Objekt arbeiten. Wenn Methoden auf dem gleichen Objekt operieren, steigt die Chance, dass sie zur gleichen oder einer ähnlichen Anforderung gehören. Eine Anforderung über das Überweisen von Geld kann beispielsweise durch zwei Methoden über das Abbuchen und das Einzahlen implementiert werden, die auf dem gleichen Geldobjekt arbeiten. Durch Integration der Informationen der anderen Methode kann

die semantische Ähnlichkeit zur Anforderung steigen. Kuang et al. [KMH+15] sind in ihrer Arbeit zum Ergebnis gekommen, dass sich die Informationen von Kontroll- und Datenfluss komplementär ergänzen. Aus diesem Grund können solche Abhängigkeiten nützlich für die Rückverfolgbarkeit sein.
Für die Datenflussabhängigkeit sind zwei Fälle denkbar: Es gibt einen Methodenaufruf zwischen den Methoden und das Objekt wird als Argument übergeben oder die Methoden rufen sich nicht auf, stattdessen greifen sie auf das gleiche Attribut zu. Im ersten Fall ist geht die Datenflussabhängigkeit mit einem Methodenaufruf einher. Durch Einbezug des Methodenaufrufs wie im vorigen Abschnitt wird also gleichzeitig die Datenflussabhängigkeit berücksichtigt.
Im zweiten Fall wird durch die Datenflussabhängigkeit eine andere Methode gefunden, dessen Informationen in die aktuelle Methode integriert werden sollen. Das ist im Grunde die gleiche Situation, die im vorigen Abschnitt bei den Aufrufabhängigkeiten vorliegt. Dort wurden ebenfalls andere Methoden identifiziert, deren Informationen in die aktuelle eingegliedert werden sollen. Daher können hier die gleichen Aggregationsmöglichkeiten angewendet werden, die für Aufrufabhängigkeiten bereits beleuchtet wurden (Durchschnitt, gewichtete Summe, ...). Die Vor- und Nachteile bleiben erhalten. Anzumerken gilt, dass es bei Aufrufabhängigkeiten die Unterscheidung zwischen dem Aufrufer und dem Aufgerufenen gibt. Wenn zwei Methoden durch Zugriff auf das gleiche Attribut in Abhängigkeit stehen, gibt es so eine Ordnung jedoch nicht. Die Frage, ob nur Aufrufer, nur Aufgerufene oder beide miteinbezogen werden sollen, entfällt bei den Datenflussabhängigkeiten.

Für Datenflussabhängigkeiten auf der Klassenebene können wieder zwei Fälle auftreten: Die Daten können als Parameter eines Methodenaufrufs zwischen den Klassen übergeben werden - dies muss nicht gesondert betrachtet werden, wenn die Aufrufabhängigkeiten bereits inkludiert werden. Die Daten können auch über ein Attribut übertragen werden. Dafür muss eine Methode auf ein öffentliches Attribut einer anderen Klasse zugreifen. Wegen dem Kapselungsprinzip sind die meisten Attribute jedoch privat. Deshalb tritt dieser Fall selten auf. Ansonsten kann auch hier wie bei den Aufrufabhängigkeiten auf der Klassenebene vorgegangen werden.

Durch die Betrachtung der Datenflussabhängigkeiten soll genau wie bei den Aufrufabhängigkeiten die Ähnlichkeitseigenschaft verbessert werden, da Informationen aus anderen Methoden integriert werden, die potentiell zur gleichen oder einer ähnlichen Anforderung zugeordnet sind. Alle anderen Eigenschaften der Quelltexteinbettungen bleiben unberührt.

Somit sind Datenflussabhängigkeiten eine weitere Möglichkeit, die Rückverfolgbarkeit zu verbessern.

4.3.3.6 Bag-of-Embeddings

Bei einer Aggregierung von Methodeneinbettungen (bzw. allgemeiner: Einbettungen von Klassenelementen) wie etwa der Durchschnittsbildung besteht der Nachteil, dass der Durchschnitt an einem Punkt im Vektorraum liegt, der wenig mit den ursprünglichen Methodeneinbettungen zu tun hat. Anstatt die Methodeneinbettungen zu aggregieren kann einer Klasse alle Methodeneinbettungen als Menge zugeordnet werden. Dadurch gehen keine Informationen durch eine Aggregation verloren. Wie bei den Bag-of-Embeddings für die Anforderungen hängt hier die Nutzungsmöglichkeit vom später verwendeten Abbildungsverfahren ab.

4.3.4 Vorverarbeitung

Für den Quelltext kann eine Vorverarbeitung eingesetzt werden, um das Rauschen zu verringern und um den Quelltext für die Einbettung aufzubereiten. Im Folgenden werden die Vorverarbeitungsschritte aus Abschnitt 2.3 in Bezug auf Quelltext diskutiert.

Bei der Stoppwortentfernung sollen Wörter entfernt werden, die vergleichsweise wenig Informationsgehalt in sich tragen und daher die Einbettungen eher verrauschen. Für die Quelltextseite muss beachtet werden, dass sowohl natürlichsprachige als auch quelltextspezifische Stoppwörter herausgefiltert werden können. Natürlichsprachige Stoppwörter können sich in den Bezeichnern und Kommentaren wiederfinden. Für den Quelltext können hier noch weitere natürlichsprachige Stoppwörter hinzugefügt werden: Wörter wie „get“ oder „index“ kommen oft in Bezeichnern vor und verlieren daher an Aussagekraft. Solche Wörter können auch in Anforderungen auftauchen, wie etwa in einer über einen Aktienindex. Das Herausfiltern von „index“ scheint hier auf den ersten Blick die Zuordnung zu erschweren. Falls jedoch der Quelltext auch Methoden mit dem Namen „getIndex“ hat, die nichts mit der Aktienindexanforderung zu tun haben, würde das Beibehalten des Wortes womöglich falsche Verbindungen zu diesen „getIndex“-Methoden herbeiführen. Das Entfernen dieser Wörter hat also Vor- und Nachteile. Der Nutzen muss in der Praxis getestet werden. Zu den quelltextspezifischen Stoppwörtern gehören Syntaxwörter der Programmiersprachen. Diese kodieren keine Semantik, die Ähnlichkeiten zu den Anforderungen aufweisen. Weiterhin wäre es möglich, häufig auftretende, grundlegende Typen als Stoppwörter herauszufiltern. Dazu gehören die primitiven Typen oder Typen wie „List“. Diese Typen sind sehr implementierungsnah und lassen sich deshalb in der Regel nicht in Anforderungen wiederfinden. Für Typen wie „List“, die theoretisch doch in Anforderungen vorkommen können, gilt die gleiche Argumentation wie oben für „index“: Das Beibehalten kann falsche Verbindungen erzeugen, während das Herausfiltern zu fehlenden Verbindungen führen kann.

Wie in der Stoppwortentfernung für die Anforderungsseite diskutiert (Abschnitt 4.2.4), ist eine Stoppwortentfernung nicht immer sinnvoll - das gilt auch für die Quelltextseite: Wenn kontextabhängige Verfahren wie ELMo oder BERT eingesetzt werden, sollten Stoppwörter beibehalten werden, da diese als Kontext dazu beitragen, Wörter auf korrekte Einbettungen abzubilden. Dies gilt jedoch nur für die natürlichsprachigen Stoppwörter. Die quelltextspezifischen Stoppwörter tragen keine Semantik der Anforderungen in sich und tauchen auch nicht bei BERT oder ELMo als Kontext auf und sollten daher entfernt werden (außer man trainiert ein eigenes Modell, das diese Wörter nicht entfernt).

Die Transformation in Kleinbuchstaben sollte für Quelltext eingesetzt werden. Die Begründung ist gleich wie auf der Anforderungsseite: Wörter ändern im Englischen nicht ihre Bedeutung, wenn sie klein geschrieben werden, daher sind zwei verschiedene Einbettungen je nach Groß- oder Kleinschreibung redundant.

Auch für die Lemmatisierung kann die gleiche Analyse wie auf der Anforderungsseite für die Quelltextseite angewendet werden, da Bezeichner und Kommentare natürlichsprachige Elemente sind. Die Lemmatisierung ist also für Englisch optional, da aufgrund der Ähnlichkeitseigenschaft der Einbettungen die verschiedenen Wortformen im Einbettungsraum sowieso nah beieinander liegen und Lemmatisierer auch Fehler einführen können.

Die Trennung der Binnenmajuskelschreibweise wird hier nötig sein. Für Bezeichner, die aus mehreren Wörtern bestehen, ist es üblich, die Wörter mit der Binnenmajuskelschreibweise aneinander zu reihen. Existierende Worteinbettungsverfahren sind für natürlichsprachigen Text ausgelegt und können nicht selbständig Binnenmajuskelschreibweisen behandeln. Daher muss dieser Verarbeitungsschritt vorher separat durchgeführt werden. Trennt man solche Wörter nicht, fassen existierende Worteinbettungsverfahren das zusammengesetztes Wort fälschlicherweise als ein einziges Wort auf.

Der Nichtbuchstabenfilter ist hier nötig, um alle programmiersyntaxbedingten Zeichen zu entfernen, da diese keine Semantik für die Rückverfolgbarkeit tragen. Zahlen treten im Quelltext vor allem bei den sehr implementierungsnahen primitiven Typen auf. Das Einbetten von Zahlen würde nur Rauschen verursachen, daher sollten sie auch herausgefiltert

werden. Satzzeichen, die in Kommentaren auftreten können, sollten für fastText, jedoch nicht für BERT entfernt werden, da Letzteres damit umgehen kann.

Ein Wortlängenfilter sollte eingesetzt werden, um kurze Abkürzungen oder nicht aussagekräftige Bezeichner mit wenigen Buchstaben zu entfernen. Dies gilt jedoch nur für fastText. Für BERT sollte kein Wortlängenfilter verwendet werden, da dadurch auch Präpositionen oder Artikel, die BERT als Kontext verwendet, entfernt werden.

Grundsätzlich gilt für die Vorverarbeitung, dass möglichst die gleichen Vorverarbeitungsschritte für die Anforderungs- und Quelltextseite verwendet werden sollten. Sind sie zu unterschiedlich, kann es die Abbildung zwischen den resultierenden Anforderungs- und Quelltexteinbettungen erschweren. Wenn zum Beispiel nur für Anforderungen Stoppwörter entfernt werden, wird der gleiche Satz auf beiden Seiten theoretisch unterschiedlich abgebildet, was natürlich kontraproduktiv für die Rückverfolgbarkeit ist.

4.3.5 Worteinbettungsverfahren für Quelltexteinbettungen

In den vorherigen Abschnitten wurden einige Verfahren diskutiert, die existierende Worteinbettungsverfahren nutzen, um Bezeichner oder Kommentare abzubilden.
In Abschnitt 4.2.2 wurde festgestellt, dass von den existierenden Verfahren BERT und fastText genutzt werden sollen. Bei BERT ist es im Gegensatz zu fastText erforderlich, ganze Sätze als Eingabe zu übergeben. Diese sind im Quelltext nur in den Kommentaren vorhanden. Zusätzlich können aus Methodensignaturen künstliche Sätze wie in Abschnitt 4.3.3.2 erzeugt werden, welcher jedoch nicht immer semantisch korrekt sein muss. Alle anderen Quelltextelemente können nicht in Satzform gebracht werden. Aus diesen Gründen wird BERT für Quelltext voraussichtlich weniger gut funktionieren als für natürlichsprachigen Text.

Von fastText und BERT sind vortrainierte Modelle verfügbar, die direkt verwendet werden können. Diese Modelle wurden jedoch auf allgemeinen Textquellen trainiert. Es ist also möglich, die Modelle mit Quelltextdaten nachzutrainieren oder ein neues Modell nur mit Quelltextdaten zu trainieren. Wie in Abschnitt 4.2.3.2 diskutiert, hat die Nutzung von nicht allgemeinen Daten Vor- und Nachteile: Quelltext kann dadurch potentiell genauer repräsentiert werden, aber es kann auch zu spezifisch werden, sodass die Übertragbarkeit auf andere (Software-)Projekte sinkt. Efstathiou et al. haben in [ES19] neue fastText-Modelle für unterschiedliche Programmiersprachen nur auf Quelltext von Softwareprojekten trainiert. Darunter sind ca. 3000 Java-Projekte. In der Evaluation haben sie drei Java-Bibliotheken auf Worteinbettungen abgebildet und zwischen ihnen paarweise die Word-Mover's Distance (Abschnitt 2.8.3) berechnet. Zwei der Bibliotheken waren Logging-Bibliotheken und zwischen ihnen bestand eine kleinere Distanz als jeweils zur dritten Bibliothek aus einem anderen Bereich. Die Autoren zogen die Schlussfolgerung, dass die trainierten Einbettungen die Information der Bibliotheken so kodieren, sodass Ähnlichkeitsaussagen aufgestellt werden können, selbst wenn das Modell auf einer Vielzahl von verschiedenen Projekten trainiert wurde. Jedoch haben die Autoren die Evaluation nicht mit den vortrainierten Modellen von fastText zum Vergleich durchgeführt. Aus der Arbeit kann also nicht gefolgert werden, dass Quelltextmodelle besser als allgemeine sind.
Des Weiteren ist ein rein auf Quelltext trainiertes Modell in Bezug auf die Anforderung-zu-Quelltextrückverfolgbarkeit nicht unbedingt hilfreicher. Für Anforderungseinbettungen wurden Verfahren vorgestellt, die ebenfalls intern ein existierendes Worteinbettungsverfahren verwenden. Wenn auf beiden Seiten der Rückverfolgbarkeit existierende Worteinbettungsverfahren eingesetzt werden, sollte es idealerweise das gleiche Worteinbettungsverfahren sein, damit die resultierenden Anforderungs- und Quelltexteinbettungen möglichst auf den gleichen Vektorraum abgebildet werden. Dadurch ist es möglich, Ähnlichkeitsmetriken wie die Kosinusähnlichkeit direkt anzuwenden. Andernfalls muss zusätzlich eine Abbildung zwischen verschiedenen Vektorräumen durchgeführt werden. Falls für die

Anforderungs- und Quelltextseite ein auf Quelltext trainiertes Modell benutzt wird, wird die Anforderungsseite potentiell schlechter repräsentiert als bei einem allgemeinen Modell, da Quelltext nicht nur vollständige ganze Sätze wie Anforderungen enthält. Dieses Risiko ist jedoch andersherum auch vorhanden: Wird ein rein auf ganzen Sätzen trainiertes Modell eingesetzt, kann der Quelltext eventuell nicht so gut repräsentiert werden. Es muss aber bedacht werden, dass Quelltext ganze Sätze in Form von Kommentaren enthält, die von einem Quelltextmodell nicht unbedingt besser repräsentiert werden als von einem allgemeinen. Dies spricht dafür, das Modell im Zweifelsfall mit ganzen Sätzen zu trainieren. Es ist auch denkbar, das Modell aus gemischten Daten, also ganze Sätze und Quelltext, zu trainieren. Dadurch besteht die Möglichkeit, dass sowohl Quelltext und Anforderungen besser repräsentiert werden. Allerdings wird dadurch die Anforderungsseite theoretisch weniger gut repräsentiert als bei einem allgemeinen Modell mit nur ganzen Sätzen, das gilt auch für die Quelltextseite mit einem Quelltextmodell. Dadurch kann es passieren, dass im Endeffekt beide Artefakte durch das Mischen schlechter repräsentiert werden. Darüber hinaus ist das Mischungsverhältnis zwischen Quelltext und natürlichsprachigem Text ein wichtiger Parameter, da das Verhältnis den Einfluss der Artefakte festlegt. Dieser muss jedoch empirisch bestimmt werden.

Als existierende Worteinbettungsverfahren können also für die Quelltextseite auch fastText und BERT eingesetzt werden. Eine Verbesserung der Abbildung durch Einsatz von quelltextspezifischen Modellen kann nicht definitiv konstatiert werden und hängt von Parametern wie dem Mischungsverhältnis ab.

4.3.6 Entwurf von Quelltexteinbettungen

Für die Rückverfolgbarkeit sollen Einbettungen für die Quelltextseite erzeugt werden. Dazu wurden in den vorigen Unterkapiteln viele unterschiedliche Möglichkeiten vorgestellt.

Bei der Diskussion über die Granularität der repräsentierten Quelltextelemente in Abschnitt 4.3.3.1 haben sich Methoden und Klasse als geeignete Kandidaten herauskristallisiert. Ob nun Anforderungen mit Methoden oder Klassen verbunden werden sollen, hängt grundsätzlich vom jeweiligen Anwendungsfall und vom verwendeten Abbildungsverfahren ab. In dieser Arbeit soll die Rückverfolgbarkeit durch Einbettungen mit anderen Rückverfolgbarkeitstechniken anhand von Evaluationsprojekten verglichen werden. Diese Projekte besitzen Anforderung-zu-Klasse-Rückverfolgbarkeitsverbindungen, daher müssen Klasseneinbettungen verwendet werden. Es ist jedoch auch möglich, die Zuordnung der Klasse aus ihren Methodeneinbettungen abzuleiten. Dies wird in Abschnitt 4.4.2 ausführlich diskutiert.

Aus der Analyse der existierenden Quelltexteinbettungsverfahren in Abschnitt 4.3.2 ist hervorgegangen, dass keines für die Rückverfolgbarkeit geeignet ist. Die für die Abbildung wichtigen natürlichsprachigen Elemente im Quelltext werden nicht berücksichtigt oder die Ähnlichkeit beruht zu sehr auf den zu feingranularen Kontroll- und Datenflüsse innerhalb einer Methode. In Abschnitt 4.3.3 wurden darauffolgend eigene Verfahren präsentiert, die natürlichsprachige Elemente miteinbeziehen. Für eine Klasse gibt es grundsätzlich zwei Möglichkeiten: Die Klasse als Fließtext auffassen oder die Klassenelemente separat betrachten und aggregieren. Letzteres ist vorzuziehen, da nicht alle Elemente in einer Klasse gleichermaßen relevant für die Rückverfolgbarkeit sind und die Möglichkeit bestehen sollte, diese Elemente zu gewichten bzw. auszulassen. Die Fließtextvariante sollte dennoch als Vergleich durchgeführt werden, da bisherige Arbeiten so vorgegangen sind. So kann die Wirkung von Einbettungen direkt verglichen werden.
In Abschnitt 4.3.3.2 und Abschnitt 4.3.3.3 wurden die einzelnen Bestandteile von Methoden bzw. Klassen auf ihre Nützlichkeit für die Rückverfolgbarkeit analysiert. Diese sind in Tabelle 4.3 zusammengefasst. Mehr Pluszeichen bedeuten tendenziell größere Nützlichkeit

Tabelle 4.3: Überblick über die zu kombinierenden Quelltextelemente

Quelltextelement	Tendenz der Nützlichkeit
Klassenname	++
Klassenkommentar	++
Methodensignatur	++
Methodenkommentar	++
Methodenrumpf	0
Attribut	0
Attributkommentar	0
Konstruktor	0
Superklassifizierer	+
Innere Klassifizierer	0

für die Rückverfolgbarkeit. Es hat sich ergeben, dass Klassennamen, Methodensignaturen, Klassen- und Methodenkommentare wichtige Informationen für die Zuordnung zu Anforderungen liefern. Konstruktoren, Attribute, Attributkommentare, Methodenrümpfe und innere Klassen können bei der Rückverfolgbarkeit helfen, jedoch ist hier das Risiko für unwichtige Implementierungsdetails höher als bei Signaturen oder Klassen- und Methodenkommentaren. Superklassifizierer sind darüber hinaus eine weitere Möglichkeit, die die Rückverfolgbarkeit verbessern kann. Sie haben nicht die Gefahr, Implementierungsdetails zu beinhalten (außer die Unterklasse besteht selbst nur aus Implementierungsdetails). Die Superklassifizierer können jedoch anderen Anforderungen zugeordnet sein als die Unterklasse. Daher wurde die Nützlichkeit der Superklassifizierer nur mit einem Plus festgelegt.

Mit diesen Elementarten sollen verschiedene Kombinationsmöglichkeiten umgesetzt werden. Da Methoden den Großteil der Funktionalität aus den Anforderungen implementieren, sollten Methodensignaturen in jeder Kombination enthalten sein. Klassennamen beschreiben die ganze Klasse und sollte auch in jeder Kombination enthalten sein. Zum Vergleich soll hier auch eine Variante nur mit Methodensignaturen und ohne Klassenname implementiert werden. Alle anderen Elementarten in müssen nicht immer in jeder Klasse vorhanden sein, daher sind diese Elementarten nur in Kombination mit der Methodensignatur und dem Klassennamen sinnvoll. Die Kombinationen setzen sich also jeweils aus der Methodensignatur, dem Klassennamen und einem der restlichen Elementarten aus Tabelle 4.3 zusammen. Aus den drei Elementarten wird jeweils eine separate Aggregation ihrer Wörter durchgeführt, die resultierenden Vektoren werden nochmals aggregiert und als Klasseneinbettung definiert (zweistufige Aggregation). Durch Vergleich mit der Variante bestehend aus Methodensignatur und Klassenname soll die Nützlichkeit des dritten Klassenelement beurteilt werden. Danach sollen die Elemente, die sich als hilfreich herausgestellt werden, nochmals kombiniert werden. Weiterhin soll eine Variante alle Klassenelemente aus Tabelle 4.3 zweistufig aggregieren und mit den anderen Kombinationen, die nur eine Auswahl treffen, verglichen werden. Für den Methodenkommentar wurde diskutiert, ob dieser zunächst mit der Methodensignatur oder separat aggregiert werden soll. Hier sollen beide Möglichkeiten umgesetzt und verglichen werden. Aufruf- und Datenflussabhängigkeiten sind zwei weitere Möglichkeiten, die die Rückverfolgbarkeit verbessern können. Methodensignaturen sollen mit ihren Aufruf- bzw. Datenflussnachbarmethodensignaturen aggregiert werden und als Methodeneinbettung definiert werden. Diese Variante soll mit der Variante, die ausschließlich aus Methodensignaturen besteht, verglichen werden.
Die Sichtbarkeit von Methoden wurde ebenfalls in Abschnitt 4.3.3.2 angesprochen. Dort wurde der Kompromiss gefunden, private Methodensignaturen nur indirekt über ihre Auf-

1. Klasse als Fließtext (einstufige Aggregation)
2. Methodensignaturen
3. Methodensignaturen + Klassenname
4. Methodensignaturen + Klassenname + Klassenkommentare
5. Methodensignaturen + Klassenname + Methodenkommentare zu Methode
6. Methodensignaturen + Klassenname + Methodenkommentare separat
7. Methodensignaturen + Klassenname + Methodenrümpfe
8. Methodensignaturen + Klassenname + Konstruktoren
9. Methodensignaturen + Klassenname + Attribute
10. Methodensignaturen + Klassenname + Attributkommentare
11. Methodensignaturen + Klassenname + Superklassifizierer
12. Methodensignaturen + Klassenname + innere Klassifizierer
13. Alle Klassenelemente (zweistufige Aggregation)
14. Methodensignaturen + Aufrufabhängigkeiten
15. Methodensignaturen + Datenflussabhängigkeiten

Aufzählung 4.3: Alle umzusetzende Kombinationen für Quelltexteinbettungen

1. Nichtbuchstabenfilter
2. Binnenmajuskelschreibweise auflösen
3. Transformation in Kleinbuchstaben
4. Lemmatisierung
5. Stoppwortentfernung (inkl. quelltextspezifische)
6. Wortlängenfilter

Aufzählung 4.4: Vorverarbeitungsschritte mit fastText für Quelltext

rufe in anderen Methoden miteinzubeziehen. Dies wird durch die Kombinationsmöglichkeit mit Methodenrümpfen abgedeckt. Es sollen also nur öffentliche Methoden berücksichtigt werden. Damit soll die Gefahr verringert werden, durch private Methoden Implementierungsdetails zu inkludieren.
In Aufzählung 4.3 sind alle besprochenen Kombinationen aufgeführt.

Für die Aggregation der Klassenelemente sind wie bei der Anforderungsseite der Durchschnitt und die gewichtete Summe der Konkatenation und dem repräsentativen Vektor vorzuziehen. Die Konkatenation erzeugt Vektoren unterschiedlicher Länge, die ausgeglichen werden müssten. Für die Auswahl eines Repräsentanten gehen potentiell relevante Informationen aus den nicht ausgewählten Elementen verloren. Durch eine gewichtete Summe könnten diese Informationen feingranularer kombiniert werden, ohne dass Elemente komplett ausgelassen werden. Die genauen Gewichte der Elemente müssen jedoch empirisch bestimmt werden.

Bezüglich der Worteinbettungsverfahren werden die gleichen Verfahren eingesetzt wie für die Anforderungsseite: BERT bzw. fastText. Dadurch sollen die Wörter auf beiden Seiten in den gleichen Vektorraum abgebildet werden, wodurch direkte Ähnlichkeitsmetriken wie die Kosinusähnlichkeit eingesetzt werden können. Da BERT Sätze entgegennimmt, schränkt sich hier die Nutzbarkeit auf Kommentare und Methodensignaturen (künstliche Satzbildung) ein. Das Trainieren von quelltextspezifischen Modellen für BERT oder fastText könnte, wie in Abschnitt 4.3.5 erklärt, eine verbesserte Repräsentation der Quelltextelemente erzielen, aber gleichzeitig die Repräsentation für die Anforderungen verschlechtern. Es soll untersucht werden, ob so ein Modell hilfreich ist.

Die Vorverarbeitungsanalyse für Quelltext in Abschnitt 4.3.4 hat ergeben, dass eine Trennung der Binnenmajuskelschreibweise, ein Nichtbuchstabenfilter, der Wortlängenfilter und

1. Nichtbuchstabenfilter
2. Binnenmajuskelschreibweise auflösen
3. quelltextspezifische Stoppwortentfernung

Aufzählung 4.5: Vorverarbeitungsschritte mit BERT für Quelltext

die Transformation in Kleinbuchstaben notwendig sein wird. Die Lemmatisierung ist optional. Quelltextspezifische Stoppwörter (Syntaxwörter) sollen in jedem Fall entfernt werden. Die natürlichsprachige Stoppwortentfernung sollte nur bei fastText, aber nicht bei BERT verwendet werden, da Letzteres diese als Kontext benötigt. Die Vorverarbeitungsschritte sind in Aufzählung 4.4 und Aufzählung 4.5 zusammengefasst. Außer der quelltextspezifischen Stoppwortentfernung sind das also die gleichen Vorverarbeitungsschritte wie für die Anforderungsseite. Das ist vorteilhaft, um den gleiche Satz im Quelltext und in den Anforderungen möglichst gleich abzubilden.

Die Verwendung von Bag-Of-Embeddings wie in Abschnitt 4.3.3.6 wird durch die Abbildung zwischen der Anforderungs- und Quelltextseite bestimmt und wird in Abschnitt 4.4.4 besprochen.

4.4 Abbildung zwischen Anforderungs- und Quelltexteinbettungen

Bisher wurden Verfahren zur Erzeugung von Einbettungen für Anforderungen und Quelltext vorgestellt. Nun sollen die Einbettungen aufeinander abgebildet werden, sodass am Ende Rückverfolgbarkeitsverbindungen identifiziert werden können. Die Abbildung kann auf verschiedene Art und Weisen durchgeführt werden. Da die Evaluationsprojekte Musterlösungen für Rückverfolgbarkeitsverbindungen zwischen Anforderungen und Klassen besitzen, müssen aus der Abbildung auch solche Verbindungen erzeugt werden können. Dazu können Klasseneinbettungen aus Abschnitt 4.3.3.3 für die Quelltextseite verwendet werden. Diese Möglichkeiten werden in Abschnitt 4.4.1 beleuchtet. Es ist jedoch auch denkbar, die Verbindungen zu Klassen aus zum Beispiel Methodeneinbettungen abzuleiten, wie etwa durch einen Mehrheitsentscheid. Die Abbildung durch Elemente einer Klasse wird in Abschnitt 4.4.2 analysiert. Die obigen Abbildungen erfolgen mit Vektoren, die im gleichen Vektorraum liegen, da sie intern das gleiche Worteinbettungsverfahren einsetzen. Der Fall, dass die Vektorräume der Anforderungs- und Quelltexteinbettungen verschieden sind, wird in Abschnitt 4.4.3 diskutiert. Schließlich wird in Abschnitt 4.4.4 der Entwurf der Abbildung vorgestellt.

4.4.1 Direkte Abbildung auf der Klassenebene

In diesem Unterabschnitt wird analysiert, wie Anforderungs- und Klasseneinbettungen aufeinander abgebildet werden können, um Rückverfolgbarkeitsverbindungen herzustellen. Die Klasseneinbettungen können dabei durch die verschiedenen (Kombinations-)Möglichkeiten in Abschnitt 4.3.3.3 erzeugt werden: Auffassung als Fließtext, Aggregation der Methoden und Klassennamen, etc. Die Erzeugung der Anforderungseinbettungen können nach den Optionen in Abschnitt 4.2.3 (Aggregation der Worteinbettungen, Dokumenteneinbettungsverfahren, etc.) durchgeführt werden. Es muss dabei darauf geachtet werden, dass das gleiche zugrundeliegende Einbettungsverfahren verwendet wird. Wenn zum Beispiel die Anforderungen mit fastText abgebildet werden, sollten für die Quelltextseite nicht BERT, sondern auch fastText verwendet werden. Durch den Einsatz des gleichen zugrundeliegenden Einbettungsverfahrens liegen Quelltext- und Anforderungseinbettungen im gleichen Vektorraum.
Die Motivation für den Einsatz von Einbettungen für die Rückverfolgbarkeit ist die Ähnlichkeitseigenschaft der Einbettungen. Wenn eine Klasse eine Anforderung implementiert,

sollen ihre Einbettungen zueinander semantisch ähnlicher sein als zu anderen Einbettungen. Daher ist es für die Bestimmung der Rückverfolgbarkeitsverbindungen naheliegend, die Ähnlichkeiten der erzeugten Quelltext- und Anforderungseinbettungen zu berechnen und alle Verbindungen ab einem gewissen Ähnlichkeitswert als korrekt zu definieren. Die Wahl dieses Schwellwerts ist entscheidend dafür, wie gut die Abbildung funktioniert. Ist er zu klein gewählt, sind womöglich zu viele falsche Verbindungen im Ergebnis des Verfahrens enthalten. Falls er zu groß gesetzt wird, können dahingegen richtige Verbindungen unbeabsichtigt herausgefiltert werden.
Für die Berechnung der Ähnlichkeit kommen die in Abschnitt 2.8 vorgestellten Ähnlichkeitsmetriken in Frage. Diese werden im Folgenden nacheinander diskutiert.

4.4.1.1 Euklidische Distanz

Die euklidische Distanz (Abschnitt 2.8.1) kann als Ähnlichkeitsmaß zwischen der Klassen- und Anforderungseinbettung eingesetzt werden. Da der Wertebereich der euklidischen Distanz unbegrenzt ist, müssen alle Einbettungen entweder vorher normiert werden oder es die Distanzen müssen hinterher auf einen festen Wertebereich abgebildet werden, damit die Schwellwertfilterung zwischen verschiedenen Abbildungen vergleichbarer wird.

4.4.1.2 Kosinusähnlichkeit

Die Kosinusähnlichkeit (Abschnitt 2.8.2) ist ein Ähnlichkeitsmaß, das den Winkel zwischen den Vektoren betrachtet. Dadurch wird die Ähnlichkeit zum einen nicht von den Vektorbeträgen beeinflusst, zum anderen ist der Wertebereich abgeschlossen, was eine zusätzliche Normalisierung für die Schwellwertfilterung wie bei der euklidischen Distanz obsolet macht. Daher ist die Kosinusähnlichkeit der euklidischen Distanz überlegen. Des Weiteren wurden in vielen bisherigen Arbeiten, in denen Ähnlichkeiten zwischen Worteinbettungen berechnet wurden, ebenfalls die Kosinusähnlichkeit verwendet (zum Beispiel [MCCD13], [BGJM17], [PNZY18]).

4.4.1.3 Word Mover's Distance

Die Ähnlichkeit zwischen einer Klasse und einer Anforderung kann auch durch die Word Mover's Distance (Abschnitt 2.8.3) bestimmt werden. Die Klasse wird dabei als Menge ihrer Wörter aufgefasst und es wird die kleinste Distanz berechnet, die nötig ist, um alle Quelltextwörter auf Anforderungswörter abzubilden. Welche Wörter in der Klasse betrachtet werden, kann variiert werden.

Der Unterschied zu den obigen Ähnlichkeitsmetriken ist der Folgende: bei den oberen Verfahren müssen die Wörter einer Klasse bzw. Anforderung zunächst zu einem Vektor aggregiert werden, wie etwa durch eine Durchschnittsbildung. Danach kann die euklidische Distanz oder die Kosinusähnlichkeit auf die beiden Durchschnittsvektoren angewendet werden. Durch die Aggregation gehen jedoch Informationen verloren, die bei der Ähnlichkeitsberechnung nicht mehr beachtet werden können. Die WMD betrachtet alle Paare von Worteinbettungen, um die minimalen Distanzen zu ermitteln. Daher müssen die einzelnen Worteinbettungen der Klasse als Bag-of-Embeddings (Abschnitt 4.3.3.6) vorliegen. Hier werden für die Ähnlichkeitsberechnung also alle Informationen beibehalten. Damit verspricht die WMD ein bessere Ähnlichkeitsberechnung als die oberen Vorgehensweisen mit Kosinusähnlichkeit oder euklidische Distanz.

Da die WMD intern die euklidische Distanz verwendet, müssen die Distanzen hier für die anschließende Schwellwertfilterung normiert werden.

4.4.1.4 Ähnlichkeit mit BERT

BERT (Abschnitt 2.6.4.5) besitzt einen Klassifizierer, der als Eingabe zwei Mengen von Wörtern (eigentlich Sätze) entgegennehmen kann. Hiermit ist es theoretisch möglich, BERT eine Klasse und eine Anforderung zu übergeben und die Wahrscheinlichkeit berechnen zu lassen, ob zwischen ihnen eine Rückverfolgbarkeitsbeziehung besteht. Diese Wahrscheinlichkeit könnte als Ähnlichkeit fungieren. Praktisch betrachtet besitzt BERT zum einen eine Zeichenbegrenzung von 500 Zeichen. Eine ganze Klasse und eine ganze Anforderung zu übergeben würde das Limit überschreiten. Zum anderen ist BERT ursprünglich nicht für Quelltext, sondern für natürlichsprachige Texte mit ganzen Sätzen konzipiert worden. Daher ist nicht vorhersehbar, wie gut BERT mit fremden Textformaten funktioniert. Diese Gründe sprechen also gegen den Einsatz von BERT so wie es oben beschrieben wird.

Statt einer ganzen Klasse können auch einzelne Klassenelemente oder Anforderungssätze übergeben werden. Die Möglichkeiten mit BERT für (Quell-)Textelemente werden in Abschnitt 4.4.2.3 diskutiert.

4.4.2 Abbildung mit Hilfe der Elementebene

Im letzten Abschnitt wurden Ähnlichkeiten zwischen der ganzen Klasse und der Anforderung berechnet. Es können aber auch Ähnlichkeiten zwischen den Elementen der Artefakte (zum Beispiel Anforderungssätze und Methodensignaturen) berechnet werden, aus denen im zweiten Schritt Rückverfolgbarkeitsverbindungen auf der Klasse-zu-Anforderungsebene abgeleitet werden.

In Abschnitt 4.4.2.1 und Abschnitt 4.4.2.2 werden zwei Verfahren erklärt, um aus der Ähnlichkeit auf der Elementebene Rückverfolgbarkeitsverbindungen auf der Klassenebene zu gewinnen. Die möglichen Ähnlichkeitsmetriken für die Elementebene werden danach in Abschnitt 4.4.2.3 besprochen.

4.4.2.1 Mehrheitsentscheid

Zur Ableitung von Rückverfolgbarkeitsverbindungen auf der Klassenebene kann über die Elemente einer Klasse ein Mehrheitsentscheid durchgeführt werden. Die Klassenelemente stimmen für Anforderungen ab und zur Anforderung mit den meisten Stimmen wird eine Rückverfolgbarkeitsverbindung für die Klasse hergestellt. Es muss hierfür bestimmt werden, für welche Anforderungen ein Klassenelement abstimmt. Naheliegend ist es, die Anforderungen und Klassenelemente auf Einbettungen abzubilden und danach ihre Ähnlichkeiten zueinander zu berechnen. Die „ähnlicheren“ Anforderungen können dann als Stimmen des Klassenelements definiert werden. Was „ähnlicher“ ist, kann zum Beispiel durch eine Schwellwertfilterung bestimmt werden, das heißt nur die Anforderungen, die eine größere Ähnlichkeit als der Schwellwert aufweisen, werden als Stimme gezählt. Hier kann auch variiert werden:
Statt dem Schwellwert kann auch pro Klassenelement die Anforderung mit der maximalen Ähnlichkeit als Stimme festgelegt werden. Dadurch wird die Wahrscheinlichkeit erhöht, dass nur für Anforderungen abgestimmt werden, zu denen tatsächlich eine korrekte Verbindung besteht. Auf der Kehrseite kann eine Stimme pro Klassenelement zu restriktiv sein, da ein Klassenelement auch zu mehreren Anforderungen gehören kann. Die Präzision ist also theoretisch höher als beim Schwellwert, aber die Ausbeute wahrscheinlich viel niedriger. Eine weitere Alternative wäre, pro Klassenelement die Top-N Anforderungen mit den größten Ähnlichkeiten als Stimmen zu werten. Ein Problem hierbei ist, dass N von der Anzahl an Anforderungen abhängt und nicht jedes Projekt gleich viele Anforderungen besitzt. N prozentual festzulegen schafft hier Abhilfe. Ein Vorteil gegenüber dem Schwellwert liegt darin, dass bei Top N jedes Klassenelement N Stimmen besitzt. Beim Schwellwert

kann es Klassenelemente ohne Stimmen geben, falls all ihre Ähnlichkeiten zu Anforderungen unter dem Schwellwert liegen. Durch Top-N werden keine Klassenelemente außen vor gelassen, wodurch sich die Ausbeute erhöhen kann. Jedoch besteht dadurch das Risiko, dass Klassenelemente ohne korrekte Verbindungen abstimmen dürfen, was der Rückverfolgbarkeit schaden würde. Dass Klassenelemente ohne Stimmen möglich sind, kann also auch ein Vorteil der Schwellwertvariante sein. Tendenziell ist für die Stimmenbestimmung folglich der Schwellwert zu favorisieren.

Nachdem die Stimmen pro Klassenelement feststehen kann der eigentliche Mehrheitsentscheid durchgeführt werden. Die „Gewinner“ der Abstimmung sind die Anforderungen, zu denen eine Verbindung mit der aktuellen Klasse hergestellt wird. Eine Möglichkeit ist es, pro Klasse die Anforderung mit der größten Stimmanzahl als Rückverfolgbarkeitsverbindung der Klasse festzulegen. Hier kann es aber zu restriktiv sein, wenn pro Klasse nur eine Rückverfolgbarkeitsverbindung festgelegt wird. Deswegen sollten zumindest alle Anforderungen, die die meiste Stimmanzahl bekommen haben, als Verbindungskandidaten angesehen werden (es kann mehrere Anforderungen geben, die gleich viel Stimmen haben). Zusätzlich ist es denkbar, die Top-N Anforderungen mit den meisten Stimmen als Klasse-zu-Anforderungsverbindungen zu bestimmen. Dadurch erhöht sich natürlich wiederum das Risiko, dass falsche Rückverfolgbarkeitsverbindungen hergestellt werden.

Bei der direkten Abbildung auf der Klassenebene in Abschnitt 4.4.1 wurde am Ende ein weiterer Schwellwert eingesetzt, um etwaige falsche Verbindungen mit zu niedrigen Ähnlichkeitswerten zu entfernen. Dies kann hier auch durchgeführt werden. Jedoch muss hier beachtet werden, dass bei Mehrheitsentscheid die Ähnlichkeiten zwischen Klassenelementen und Anforderungen berechnet wurden. Ähnlichkeiten der nun vorliegenden Klasse-zu-Anforderungsverbindungen existieren noch nicht und müssten für die Schwellwertfilterung bestimmt werden. Sie können zum Beispiel berechnet werden, indem jeweils die Ähnlichkeiten der Klassenelemente zur gleichen Anforderung aggregiert werden.
Eine Menge von Ähnlichkeitswerten muss hier also zu einem einzelnen reduziert werden. Dafür stehen zwei Optionen zur Verfügung: Durchschnitt oder Maximum. Durch das Maximum werden niedrigere Ähnlichkeiten ignoriert - die Anforderungen, die die Abstimmung gewinnen, erhalten dadurch größere Zuversicht und werden weniger wahrscheinlich durch den letzten Schwellwert herausgefiltert. Beim Durchschnitt ist es andersherum: Falls sich die Ähnlichkeiten der einzelnen Stimmen zu sehr unterscheiden, wird ein Mittelwert gefunden, der niedriger als das Maximum ist. Hier ist die Wahrscheinlichkeit größer, dass die Verbindung zur Anforderung durch den nachfolgenden Schwellwertfilter entfernt werden kann. Welche Reduktion besser ist, kann nicht allgemein beantwortet werden. Da die Ähnlichkeiten beim Mehrheitsentscheid bereits bei der Stimmenbestimmung zum Beispiel durch einen Schwellwert vorgefiltert werden, sollten die verbliebenen Ähnlichkeiten sowieso alle relativ hoch sein. Ob nun der hohe Durchschnitt oder das sehr hohe Maximum ausgewählt wird, sollte somit keinen beträchtlichen Unterschied machen.

Für die Anforderungsseite wurde bisher aus der ganzen Anforderung eine Einbettung erzeugt und die Klassenelement-zu-Anforderungsähnlichkeiten berechnet. Eine Alternative besteht darin, auch für die Anforderungsseite die Einbettungen auf der Elementebene - also Anforderungssatzeinbettungen - zu verwenden. Hierdurch müssen am oben diskutierten Vorgehen jedoch einige Anpassungen vorgenommen werden. Für die Stimmenbestimmung des Mehrheitsentscheid wurde bisher die Ähnlichkeit zur ganzen Anforderung verwendet, um zu entscheiden, ob das Klassenelement für die Anforderung abstimmt. Nun liegen jedoch Ähnlichkeiten zu den einzelnen Anforderungssätzen vor. Eine naheliegende Option ist die Reduktion der Satzähnlichkeiten zu einer einzigen Ähnlichkeit für die ganze Anforderung. Dazu gibt es die gleichen Möglichkeiten wie bei der Reduktion, die weiter oben für Klassenelemente beschrieben wurde: Es wird ein Durchschnitt der Satzähnlichkeiten berechnet oder aus den Satzähnlichkeiten wird die maximale Ähnlichkeit als Repräsen-

tant ausgewählt. Beide Methoden haben ihre Vor- und Nachteile: Wenn eine Quelltextelementeinbettung zu einem von vier Sätzen eine hohe Ähnlichkeit erzielt und zu den anderen eine sehr niedrige, wird durch das Maximum eine insgesamt hohe Ähnlichkeit zur gesamten Anforderung festgelegt, obwohl die Mehrheit der Sätze keine große Ähnlichkeit aufweist und damit die Wahrscheinlichkeit für eine tatsächliche Rückverfolgbarkeitsverbindung sinkt. Bei einem Durchschnitt wäre die Ähnlichkeit niedriger und kann mit größerer Wahrscheinlichkeit bei einer späteren Schwellwertfilterung entfernt werden. Jedoch kann es auch sein, dass eine hohe Ähnlichkeit zu drei Sätzen besteht und eine niedrige zur vierten. In diesem Fall steigt die Wahrscheinlichkeit, dass der vierte Satz weniger relevant ist und nur Rauschen darstellt. Durch das Maximum würde das Rauschen richtigerweise herausgefiltert werden, während die Durchschnittsbildung durch den verrauschten vierten Satz beeinträchtigt wird. Um den Durchschnitt zu verbessern, kann vorher eine weitere Schwellwertfilterung durchgeführt werden, um Verbindungen mit zu niedrigen Ähnlichkeiten auszusortieren. Dadurch könnte auch bei der Durchschnittsbildung der verrauschte vierte Satz herausgefiltert werden. Beim Maximum hat die zusätzliche Schwellwertfilterung keinen Effekt, außer die maximale Ähnlichkeit liegt unter dem Schwellwert. In diesem Fall besteht höchstwahrscheinlich keine Verbindung und die Anforderung wird korrekterweise aussortiert. Es ist hier nicht eindeutig, ob das Maximum oder der Durchschnitt überlegen ist. Dies muss in der Praxis getestet werden.

Die Ähnlichkeiten können also zwischen Klassenelementen und Anforderungssätzen oder ganzen Anforderungen berechnet werden. Es stellt sich die Frage, welche Variante besser ist. Bei den Klassenelementen gibt es diejenigen, die aus wenigen Bezeichnern bestehen (und aus denen eventuell ein künstlicher Satz gebaut werden kann) und die Kommentare, die aus mehreren Sätzen zusammengesetzt sein können. Auf dem ersten Blick sind sich Kommentare und ganzen Anforderungen sowie Bezeichner und Anforderungssätzen von der Form her ähnlicher. Jedoch bedeutet dies nicht unbedingt, dass dadurch zum Beispiel Kommentare mit ganzen Anforderungen bessere Ergebnisse erzielen als mit einzelnen Anforderungssätzen. Die Informationen des Kommentars können auch nur in einem Satz der Anforderung enthalten sein, wodurch die Anforderungssatzvariante eine höhere Ähnlichkeit erzielt, während die Ähnlichkeit zur ganzen Anforderung durch die anderen Sätze verrauscht wird. Bei den Bezeichnern, die den Anforderungssätzen von der Form her mehr ähneln, kann es dagegen sein, dass die Bezeichnerinformationen nicht in einem einzigen Satz, sondern über mehrere Anforderungssätze verteilt enthalten sind. Dann kann die Ähnlichkeitsberechnung mit der ganzen Anforderung größere Werte erzeugen.
Jedoch ist mit der Anforderungssatzvariante eine zusätzliche Schwellwertfilterung möglich. Auf diese Weise kann das Rauschen durch potentiell weniger relevante Sätze reduziert werden, was die Genauigkeit der Ähnlichkeit zur Anforderung verbessert. Das ist bei ganzen Anforderungseinbettungen nicht möglich. Aus diesem Grund tendiert die Auswahl in Richtung Anforderungssatzvariante.

Der Mehrheitsentscheid hat gegenüber der Abbildung mit Klasseneinbettungen den Vorteil, dass unwichtige Klassenelemente bzw. Anforderungssätze einer Klasse weniger Einfluss auf die Rückverfolgbarkeitsverbindung haben. Zum Beispiel kann eine Klasse vier Methoden besitzen, wovon eines eher Implementierungsdetail darstellt und keine Anforderung implementiert. Bei der Durchschnittsbildung für die Klasseneinbettung würde diese für die Rückverfolgbarkeit nutzlose Methode dennoch miteinberechnet. Beim Mehrheitsentscheid besteht die Möglichkeit, dass die anderen drei Methoden, die die korrekte Anforderung implementieren, die nutzlose Methode überstimmen, sodass sie gar keinen Einfluss mehr besitzt. Jedoch muss bedacht werden, dass der Mehrheitsentscheid mehr Parameter durch die Schwellwerte, die Reduktionsfunktionen und der Stimmenbestimmung besitzt. Es muss zusätzlicher Aufwand betrieben werden, diese zu bestimmen, da bei einer schlechten Parameterkonfiguration der Mehrheitsentscheid schlechter funktionieren kann als die

Abbildung auf der Klassenebene. Zum Beispiel kann der Mehrheitsentscheid zu restriktiv sein, falls pro Klasse zu wenige Verbindungen zugelassen werden. Insgesamt bietet der Mehrheitsentscheid jedoch das Potential, eine Verbesserung gegenüber den Klasseneinbettungen darzustellen.

4.4.2.2 Mehrheitsentscheid mit Aufrufabhängigkeiten

Im vorigen Abschnitt konnten die Elemente einer Klasse, die in Abschnitt 4.3.3.3 vorgestellt wurden, in der Mehrheitsentscheidung für die Klasse teilnehmen. In Abschnitt 4.3.3.4 und Abschnitt 4.3.3.5 wurden mit Aufruf- bzw. Datenflussabhängigkeiten Nachbarmethoden gefunden, die semantische Ähnlichkeiten zur verbundenen Anforderung beinhalten können. Diese Nachbarmethoden können also dabei helfen, korrekte Rückverfolgbarkeitsverbindungen zu erstellen. Daher kann es sinnvoll sein, den Mehrheitsentscheid mit diesen Informationen zu erweitern. Eine Art der Integration von diesen Informationen wurde in Abschnitt 4.3.3.4 bzw. Abschnitt 4.3.3.5 vorgestellt. Dort wurden für die aktuelle Methode und die Nachbarn zunächst separate Methodeneinbettungen berechnet. Danach wurden sie gewichtet aufsummiert und als Methodeneinbettung der aktuellen Methode definiert. Diese Methoden können dann den Mehrheitsentscheid, wie im vorigen Abschnitt beschrieben, durchlaufen.

Es ist jedoch auch möglich, statt Methodeneinbettungen der Nachbarn direkt ihre Ähnlichkeiten zu den Anforderungen zu berechnen und gewichtet aufzusummieren. Das bedeutet, dass bei der Stimmenbestimmung der aktuelle Methode die Nachbarmethoden miteinbezogen werden: Für die aktuelle Methode kann die Ähnlichkeit zu einer Anforderung A_i mit den Ähnlichkeiten der Nachbarmethoden zur Anforderung A_i gewichtet aufsummiert werden. Die resultierende Summe wird als neue Ähnlichkeit der aktuellen Methode zur Anforderung A_i definiert und es wird damit fortgefahren. Alles andere im Mehrheitsentscheid bleibt unverändert.
So wie in Abschnitt 4.3.3.4 ist auch hier die gewichtete Summe dem Durchschnitt vorzuziehen, damit die aktuelle Methode höher gewichtet werden kann. Die Nachbarmethoden können auch zu anderen Anforderungen verbunden sein als die aktuelle Methode. Die Informationen der aktuellen Methode sind daher nützlicher und wichtiger, um die korrekte Anforderung zu finden, als diejenigen aus Nachbarmethoden, die potentiell in komplett andere Klassen liegen können. Aus dem gleichen Grund ist das Maximum der Ähnlichkeit unter den (Nachbar-)Methoden auch nicht sinnvoll: Eine einzelne Nachbarmethode kann eine sehr hohe Ähnlichkeit zu einer Anforderung haben - das muss jedoch nicht bedeuten, dass die aktuelle Methode ebenfalls mit dieser Anforderung in Verbindung steht. Vor allem wenn die eigene Ähnlichkeit der aktuellen Methode zu dieser Anforderung im Kontrast sehr niedrig ist, besteht die Wahrscheinlichkeit, dass die aktuelle und die Nachbarmethode zu verschiedenen, weniger ähnlichen Anforderungen gehören. Die bevorzugte Aggregationsmethode bleibt also die gewichtete Summe.
Für die Integration der Nachbarn können also entweder die Nachbareinbettungen oder die Nachbarähnlichkeiten gewichtet aufsummiert werden. Hier drängt sich die Frage auf, welche Methodik besser funktioniert. Angenommen, es existiert eine Nachbarmethode, die nicht zur gleichen Anforderung A_i wie die aktuelle Methode gehört. Bei der gewichteten Summe der Einbettungen werden diese Informationen trotzdem in die aktuelle Methodeneinbettung integriert - was in diesem Fall für Rauschen sorgen würde. Bei den gewichteten Ähnlichkeiten werden die Ähnlichkeiten der Nachbarmethoden getrennt berechnet. Hier ist es möglich, die Ähnlichkeiten er Nachbarmethode zu A_i durch einen Schwellwert vorzufiltern. Damit würde diese Ähnlichkeit nicht in die gewichtete Summe der Ähnlichkeiten miteingerechnet - wodurch also Rauschen vermindert wurde. Aus diesem Grund würde die gewichtete Summe über Ähnlichkeiten favorisiert werden.

Außer der gewichteten Summe der Nachbarähnlichkeiten kann auch ein Mehrheitsentscheid

als weitere Option durchgeführt werden. Das heißt bei der Stimmenbestimmung stimmen die aktuelle Methode und die Nachbarmethoden mit ihren Ähnlichkeiten in einem ersten Mehrheitsentscheid ab, für welche Anforderung(en) die aktuelle Methode im zweiten Mehrheitsentscheid abstimmen soll. Der zweite Mehrheitsentscheid ist der reguläre Mehrheitsentscheid der Klasse, welcher bisher in den obigen Abschnitten beschrieben wurde. Für die Bestimmung der Stimme(n) im ersten Mehrheitsentscheid bestehen die gleichen Optionen, die im vorigen Abschnitt 4.4.2.1 für den regulären Mehrheitsentscheid bereits erläutert wurden: Es kann die Anforderung mit der maximalen Ähnlichkeit als Stimme festgelegt werden. Alternativen sind: Die Top N Anforderungen mit den größten Ähnlichkeiten oder eine Filterung durch einen Schwellwert. Auch hier gilt, dass zu wenige Stimmen möglicherweise dazu führen, dass nicht alle verbundenen Anforderungen abgedeckt werden. Zu viele Stimmen verrauschen das Ergebnis dagegen durch Stimmabgabe für nicht korrekte Anforderungen. Weiterhin sollten die Stimmen für die aktuelle Methode vervielfacht werden, damit sie im Vergleich zu den Nachbarn mehr Gewicht hat.
Ob der doppelte Mehrheitsentscheid besser funktioniert als die gewichtete Ähnlichkeitssumme wird davon abhängen, wie viele Stimmen zugelassen werden. Prinzipiell ist es beim doppelten Mehrheitsentscheid möglich, dass Nachbarmethoden, die nicht zur gleichen Anforderung wie die aktuelle Methode gehören, überstimmt werden und dadurch ausgelassen werden. Das würde das Rauschen vermindern. Bei der gewichteten Ähnlichkeitssumme können Ähnlichkeiten von Nachbarmethoden aber auch durch einen Schwellwertfilter entfernt werden. Daher kann a priori kein Favorit zwischen Ähnlichkeitssumme und doppelter Mehrheitsentscheid bestimmt werden.

Noch eine Variante, die die Nachbarmethoden miteinbezieht, kann wie folgt aussehen: Für die aktuelle Methode werden ganz normal die Anforderungen bestimmt, für die abgestimmt werden soll (ohne Nachbarmethoden). Danach werden die Stimmen anhand der Ähnlichkeiten der Nachbarmethoden bewertet. Haben die Nachbarmethode ebenfalls eine hohe Ähnlichkeit, wird die Ähnlichkeit der Stimme erhöht, andernfalls wird sie erniedrigt. Ein Vorteil hierbei ist, dass nur die Stimmen der aktuellen Methode bewertet werden. Es können keine neuen Stimmen durch die Nachbarmethoden, die potentiell zu anderen Anforderungen gehören, dazukommen. Dies ist gleichzeitig auch ein Nachteil, falls Nachbarmethoden doch zur gleichen Anforderung gehören. Außerdem muss hier ein weiterer Satz an Parametern ermittelt werden, das heißt um wie viel die Ähnlichkeit bei der Bewertung erhöht oder erniedrigt wird und ab welchem Schwellwert die Ähnlichkeit der Nachbarmethoden als Bestärkung oder Abschwächung zählt.
Im Vergleich zur Ähnlichkeitssumme ist die Bewertungsmethode restriktiver: Es wird stets angenommen, dass die Nachbarmethoden zu keinen neuen, korrekten Anforderungen verbunden sind, da sie keine neuen Stimmen erzeugen können. Bei der Ähnlichkeitssumme ist das jedoch grundsätzlich möglich und Rauschen durch Nachbarmethoden, die zu anderen Anforderung gehören, kann mit der Gewichtung zwischen aktueller Methode und Nachbarmethoden sowie einer vorher durchgeführten Schwellwertfilterung vermindert werden. Daher ist in diesem Fall die Ähnlichkeitssumme überlegen.

4.4.2.3 Ähnlichkeit auf Elementebene

Beim Mehrheitsentscheid wird eine Ähnlichkeitsberechnung zwischen Einbettungen von Klassen- und Anforderungselementen zugrunde gelegt. Im Folgenden werden dazu verschiedene Optionen diskutiert.

Euklidische Distanz und Kosinusähnlichkeit
Die Euklidische Distanz (Abschnitt 2.8.1) und Kosinusähnlichkeit (Abschnitt 2.8.2) können zwischen Klassen- und Anforderungssatzeinbettung angewendet werden, um die Ähnlichkeit zu bestimmen. Wie in Abschnitt 4.4.1.2 diskutiert, ist dabei die Kosinusähnlichkeit der

euklidischen Distanz vorzuziehen, da Letzteres unabhängig vom Betrag der Einbettungen ist und auch in vergangenen Arbeiten eingesetzt wurde.

Word Mover's Distance
Die WMD (Abschnitt 2.8.3) kann ebenfalls auf der Elementebene verwendet werden, beispielsweise zwischen Methodensignaturen und Anforderungssätzen. Die Wörter des Klassenelements müssen dazu einzeln auf Worteinbettungen als Bag-of-Embeddings (siehe Abschnitt 4.3.3.6) abgebildet werden.

In Abschnitt 4.4.1.3 wurde der Vorteil angeführt, dass die WMD der einfachen Kosinusähnlichkeit überlegen ist, da alle Abbildungsmöglichkeiten zwischen den Wörtern der Anforderungssätze- und Klassenelemente betrachtet werden. Dies gilt auf der Elementebene natürlich auch.

Ähnlichkeit durch BERT
BERT (Abschnitt 2.6.4.5) ist in der Lage, zwei Sätze entgegenzunehmen und sie zu klassifizieren. Es ist also denkbar, ein neues Modell zu trainieren, dass jeweils ein Klassenelement und einen Anforderungssatz entgegennimmt und klassifiziert, ob zwischen den beiden Elementen eine Rückverfolgbarkeitsverbindung besteht. BERT gibt dabei eine Wahrscheinlichkeit für das Bestehen einer Verbindung aus. Diese Wahrscheinlichkeit kann wiederum in den obigen Mehrheitsentscheidverfahren als Ähnlichkeit verwendet werden. Da Klassenelemente und Anforderungssätze viel kürzer als ganze Klassen oder Anforderungen sind, tritt das Zeichenlimitproblem, das in Abschnitt 4.4.1.4 beschrieben wurde, weniger wahrscheinlich auf. Das andere Problem, dass BERT nicht für den Umgang von Quelltext konzipiert wurde und daher nicht absehbar ist, wie gut BERT damit funktioniert, verbleibt. Dies kann jedoch abgemildert werden, indem Kommentarsätze als Klassenelemente eingesetzt werden. Außerdem besteht auch die Möglichkeit, aus Methodensignaturen künstliche Sätze zu bilden, wie es in Abschnitt 4.3.3.2 beschrieben wurde. Nicht jede Methode hat einen Kommentar, daher ist es sinnvoll, die Kommentarsatzvariante in Kombination mit der Methodensignaturvariante zu verwenden. Bei der Kombination kann zwischen der Kommentar- und Methodensignaturwahrscheinlichkeit aggregiert werden, wie etwa durch den Durchschnitt, durch eine gewichtete Summe oder durch einen Repräsentanten. Da Kommentarsätze im Gegensatz zu den künstlichen Signatursätzen echte natürliche Sätze sind, sollten diese stärker gewichtet werden, anstatt einen ungewichteten Durchschnitt zu bilden. Es ist auch möglich, die Kommentarwahrscheinlichkeit als Repräsentanten auszuwählen und nur auf die Signaturwahrscheinlichkeit zurückzugreifen, falls es keinen Kommentar zur Methode gibt. Dies würde den Kommentar noch mehr bevorzugen. Ob trotzdem eine Gewichtung sinnvoller ist und wie groß die Gewichte sein sollten, muss empirisch ermittelt werden.

4.4.3 Abbildung zwischen unterschiedlichen Vektorräumen

Die in den vorherigen Abschnitten beschriebenen Abbildungsverfahren nutzen Ähnlichkeitsmetriken, die nur funktionieren, wenn die beteiligten Vektoren im selben Vektorraum liegen. Das wird dadurch erreicht, dass für Quelltext und Anforderungen das gleiche zugrundeliegende Wort- oder Satzeinbettungsverfahren eingesetzt wird. Da beide Artefakte verschieden sind, muss nicht unbedingt das gleiche Einbettungsverfahren auch für beide Artefakte gleich ideal sein. Beispielsweise kann für Anforderungen BERT und für Quelltextbezeichner fastText verwendet werden. Da bei Anforderungen Sätze vorhanden sind, kann durch BERT die genauere Repräsentation durch kontextsensitive Einbettungen ausgenutzt werden. Falls jedoch unterschiedliche Einbettungsverfahren für beide Seiten verwendet werden, befinden sich die Einbettungen nicht mehr im gleichen Vektorraum. Dann muss eine Abbildung zwischen verschiedenen Vektorräumen konstruiert werden. Dazu sind in Abschnitt 3.3 verwandte Ansätze vorgestellt worden, die hier potentiell auch benutzt werden können. Nachdem die Vektoren beider Artefakte im gleichen Vektorraum liegen,

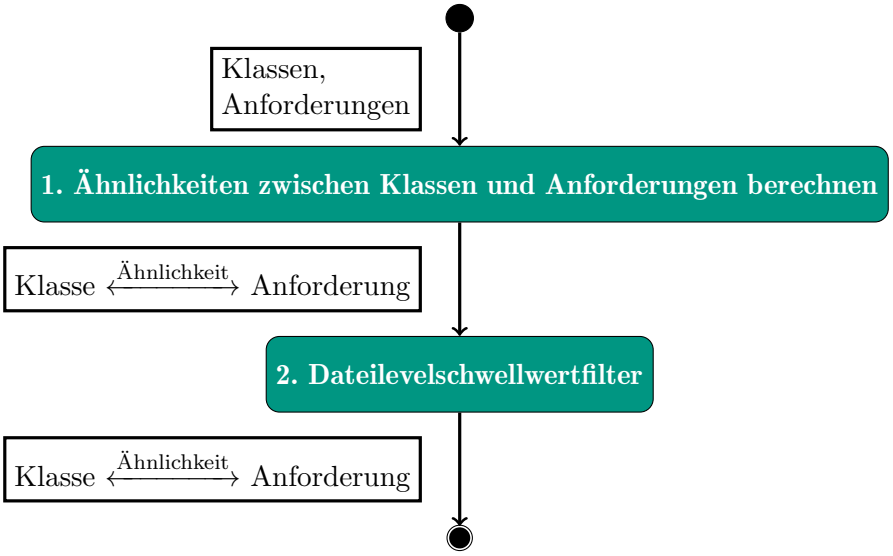


Abbildung 4.3: Aktivitätsdiagramm zum Ablauf der Abbildung auf der Klassenebene

können die in den vorherigen Abschnitten beschriebenen Verfahren wie etwa der Mehrheitsentscheid eingesetzt werden, um die Rückverfolgbarkeitsverbindungen zu ermitteln. Durch den zusätzlichen Abbildungsschritt können jedoch neue Fehler in das Verfahren eingeführt werden, die potentiell die Vorteile wieder zunichtemachen.

4.4.4 Entwurf der Abbildung zwischen Anforderungs- und Quelltexteinbettungen

Mit Hilfe eines Abbildungsverfahrens zwischen Anforderungs- und Quelltexteinbettungen sollen Rückverfolgbarkeitsverbindungen zwischen korrespondierenden Anforderungen und Klassen gefunden werden.

Hierfür wurden Abbildungen zwischen Klasseneinbettungen und Anforderungseinbettungen (Abschnitt 4.4.1) und Abbildungen, die sich aus der Elementebene ableiten (Abschnitt 4.4.2) beleuchtet. Aus der Analyse ist hervorgegangen, dass der Mehrheitsentscheid unwichtige Klassenelemente bei der Abstimmung überstimmen und aussortieren kann. Dies würde gegen die Abbildung auf der Klassenebene sprechen. Jedoch hängt der Mehrheitsentscheid auch von einer guten Parameterkonfiguration ab. Daher sollen beide Varianten umgesetzt und miteinander verglichen werden.

Der Ablauf der Abbildung auf der Klassenebene ist in Abbildung 4.3 aufgezeigt. In Schritt Eins werden alle Klassen und Anforderungen eingebettet und zwischen ihnen paarweise die Ähnlichkeiten berechnet. Anschließend werden die Verbindungen anhand der Ähnlichkeiten durch einen Schwellwert, hier Dateilevelschwellwert genannt, gefiltert, um wahrscheinlich falsche Verbindungen, die eine niedrige Ähnlichkeit besitzen, auszusortieren.

Für den Mehrheitsentscheid wurde eine Variante mit ganzen Anforderungen und eine mit Anforderungssätzen vorgestellt. Letzteres gilt wegen des zusätzlichen Schwellwertfilters für Anforderungssätze als Favorit, ansonsten laufen beide Verfahren jedoch sehr ähnlich ab. Es sollen beiden Varianten umgesetzt werden, um die tatsächliche Verbesserung der Anforderungssatzvariante zu beurteilen.

In Abbildung 4.4 ist der Ablauf der Variante mit ganzen Anforderungen abgebildet. Im ersten Schritt werden zuerst paarweise die Ähnlichkeiten zwischen Klassenelement und Anforderung (bzw. Klassenelement- und Anforderungseinbettung) berechnet. Jedes Klassenelement hat nun eine Verbindung zu jeder Anforderung. Für die Stimmenbestimmung

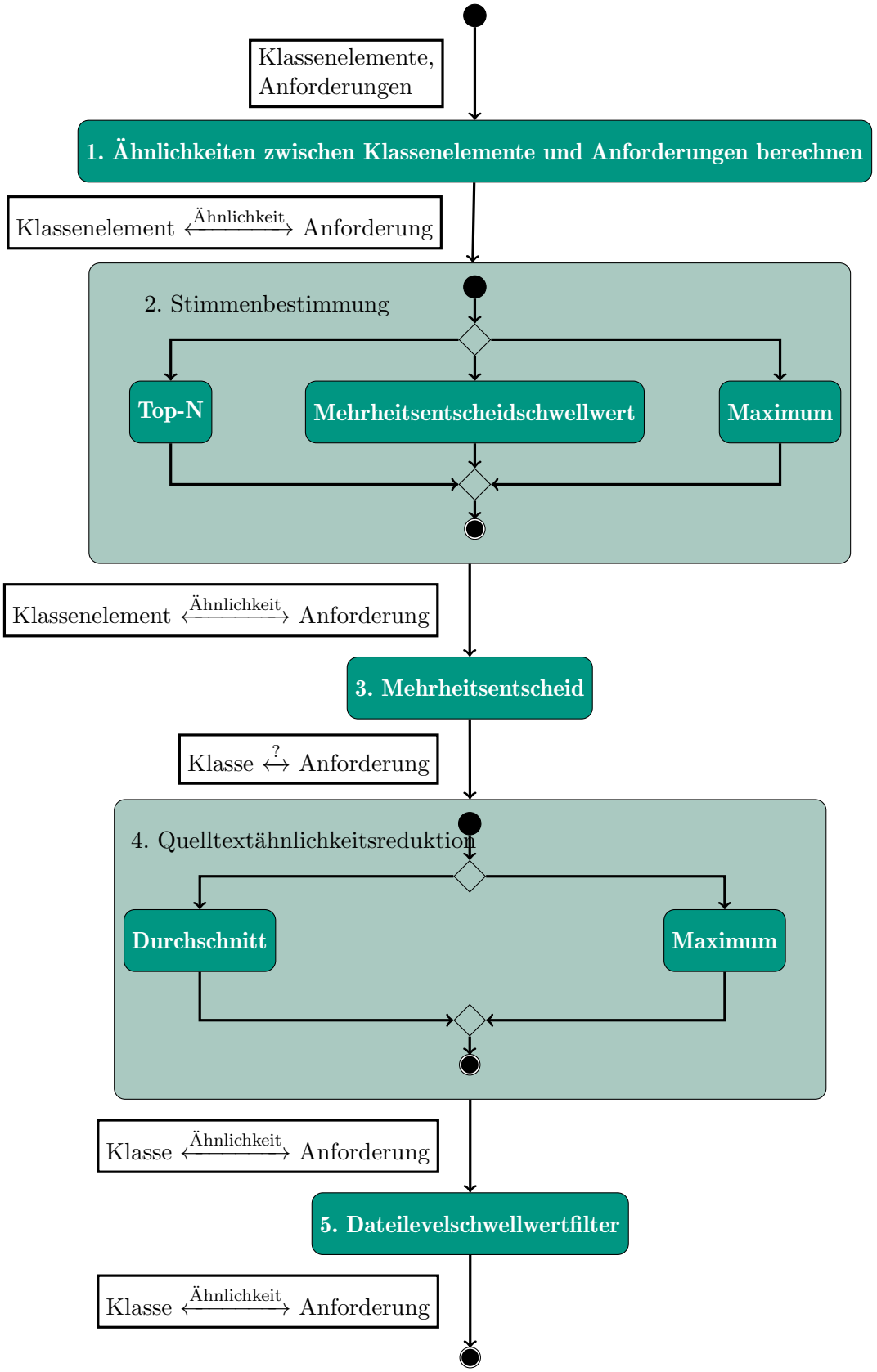


Abbildung 4.4: Aktivitätsdiagramm zum Ablauf des Mehrheitsentscheids

gibt es wie in der Analyse drei Möglichkeiten: Das Maximum, ein Schwellwert (in Abbildung 4.4 Mehrheitsentscheidschwellwert genannt) oder Top-N. Das Maximum ist potentiell zu restriktiv, da es nur eine Stimme pro Klassenelement zulässt. Hier sollen also die anderen beiden Vorgehen umgesetzt werden. In Schritt vier erfolgt der eigentliche Mehrheitsentscheid, wo die Anforderungen mit der maximalen Stimmanzahl als vorläufige Verbindung festgelegt werden. Danach werden Klasse-zu-Anforderungsähnlichkeiten bestimmt, hier gibt es die Auswahl zwischen Durchschnitt und Maximum. Da die Analyse hier keinen eindeutigen Favoriten aufzeigen konnte, sollen beide Möglichkeiten umgesetzt werden. Als Letztes werden die vorläufigen Verbindungen nochmals mit einem Ähnlichkeitsschwellwert (in Abbildung 4.4 Dateilevelschwellwert genannt) gefiltert, um etwaige falsche Verbindungen mit niedriger Ähnlichkeit zu entfernen. Alle übrigen Verbindungen werden als Rückverfolgbarkeitsverbindungen festgelegt.

Der Mehrheitsentscheid mit Anforderungssätzen ist in Abbildung 4.5 dargestellt. Hier werden als Erstes die Ähnlichkeiten zwischen Klassenelementen und Anforderungssätzen berechnet. Anschließend erfolgt die erste Schwellwertfilterung, hier als Elementschwellwertfilter bezeichnet, um Anforderungssätze mit niedrigen Ähnlichkeiten auszusortieren. Im nächsten Schritt werden die Ähnlichkeiten der Anforderungssätze auf eine Ähnlichkeit für die ganze Anforderung reduziert. Genau wie bei der Quelltextähnlichkeitsreduktion in Abbildung 4.4 gibt es hier zwei Optionen, wobei keine Option eindeutig überlegen ist. Daher sollen beide Möglichkeiten umgesetzt werden. Nach dem dritten Schritt verläuft der Mehrheitsentscheid identisch wie in Abbildung 4.4. Die Wahlmöglichkeiten bei der Stimmenbestimmung und bei der Quelltextähnlichkeitsreduktion existieren hier auch, aber wurden aus Platzgründen in der Abbildung weggelassen.

Für den Mehrheitsentscheid wurde auch die Erweiterung mit Aufruf- und Datenflussbeziehungen (Abschnitt 4.4.2.2) analysiert. Das Miteinbeziehen von Nachbarmethoden kann mehr semantische Ähnlichkeit zur verbundenen Anforderung erzeugen, aber auch zusätzliches Rauschen verursachen. Hier sollten also beide Varianten getestet und miteinander verglichen werden. Bezüglich der Art und Weise, wie die Nachbarmethoden berücksichtigt werden, konnte in der Analyse kein Favorit zwischen doppelter Mehrheitsentscheid und Ähnlichkeitssumme festgelegt werden. Daher sollen hier beide Möglichkeiten umgesetzt werden. Zum Vergleich soll auch die Variante mit der Vektorsumme implementiert werden.

Als Klassenelemente, mit denen im Mehrheitsentscheid die Ähnlichkeiten berechnet werden, sollen alle Elemente, die zur Aggregation für eine Klasseneinbettung in Abschnitt 4.3.3.3 vorgestellt bzw. in Tabelle 4.3 zusammengefasst sind, eingesetzt werden. Das bedeutet es können beispielsweise Methoden oder Kommentare einer Klasse abstimmen. Diese sollen hier auch umgesetzt und mit der Klasseneinbettungsvariante verglichen werden. Wie im Entwurf für die Quelltextseite erklärt, sollten Methodensignaturen und Klassennamen in jedem Fall dabei sein, da Methoden den Großteil der Funktionalität implementieren bzw. beschreiben. Diese bilden jeweils eine Kombination mit einem der anderen Elemente aus Tabelle 4.3.

Bezüglich der verschiedenen Ähnlichkeitsmetriken (Abschnitt 4.4.2.3, Abschnitt 4.4.1) ist aus der Analyse hervorgegangen, dass die Kosinusähnlichkeit der euklidischen Distanz überlegen ist. Deswegen wird von diesen beiden das Erstere verwendet.
Die Word Mover's Distance berechnet die Ähnlichkeit im Gegensatz zur Kosinusähnlichkeit auf Mengen von Vektoren. Zwar kann zum Beispiel die Kosinusähnlichkeit auf den Durchschnitt von einer Vektormenge angewendet werden, aber durch die Aggregation gehen Informationen verloren. Daher soll auch die WMD in Kombination mit dem Mehrheitsentscheid und der Abbildung auf der Klassenebene getestet werden. Für die Repräsentation der Artefakte bzw. Klassen- und Anforderungselemente müssen dabei Bag-Of-Embeddings

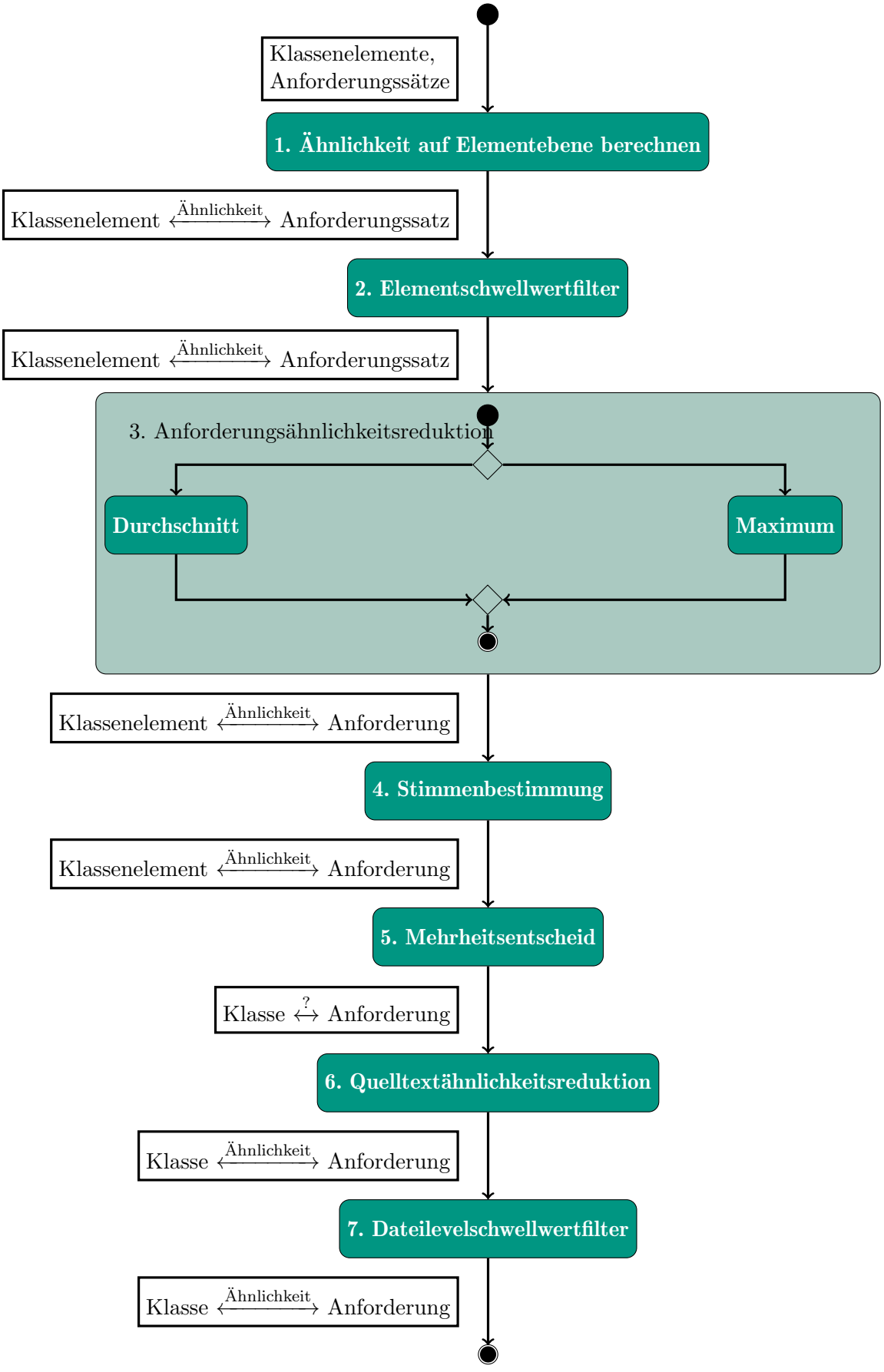


Abbildung 4.5: Aktivitätsdiagramm zum Ablauf des Mehrheitsentscheids, falls die Ähnlichkeiten auf der Elementebene berechnet werden

(Abschnitt 4.2.3.4, Abschnitt 4.3.3.6) verwendet werden. Hier muss auch entschieden werden, welche Wörter einer Anforderung bzw. Klasse für den Bag-Of-Embeddings ausgewählt werden. Dazu sollen alle Möglichkeiten aus Tabelle 4.3 für die Quelltextseite untersucht werden. Die Anforderungsseite werden Anforderungen oder Anforderungssätze durch die Menge ihrer Worteinbettungen repräsentiert.
Bei der Ähnlichkeit durch BERT ist unklar, wie BERT mit Quelltexte umgehen kann. Durch Kommentarsätze oder künstliche Methodensätze kann eine Annäherung zur natürlichen Sprache vorgenommen werden. Die BERT-Ähnlichkeiten sollen also auch untersucht werden. Da ganze Anforderungen und Klassen wahrscheinlich die Zeichenbegrenzung für BERT überschreiten, soll hier nur die Abbildung auf der Elementebene durch den Mehrheitsentscheid berücksichtigt werden.

Die Abbildung zwischen verschiedenen Vektorräumen (Abschnitt 4.4.3) ist nicht Fokus dieser Arbeit und wird daher nicht umgesetzt.

Die umzusetzenden Abbildungsverfahren sind in Aufzählung 4.6 zusammengefasst.

Die eingesetzten Vorverarbeitungsschritte wurden in Abschnitt 4.2.5 und Abschnitt 4.3.6 bereits vorgestellt.

1. Abbildung auf der Klassenebene (Kosinusähnlichkeit)
2. Abbildung durch den Mehrheitsentscheid (Kosinusähnlichkeit)
3. Abbildung durch den Mehrheitsentscheid (Kosinusähnlichkeit) mit Aufruf-/Datenflussabhängigkeiten
4. Abbildung durch den Mehrheitsentscheid (Ähnlichkeit mit BERT)
5. Abbildung durch den Mehrheitsentscheid (WMD)

Aufzählung 4.6: Umzusetzende Abbildungsverfahren

5 Implementierung

In diesen Kapitel wird die Implementierung der Entwürfe (siehe Abschnitt 4.2.5, Abschnitt 4.3.6 und Abschnitt 4.4.4) vorgestellt. Abschnitt 5.1 beschreibt die Anbindung von fastText und BERT, da diese als zugrundeliegende Worteinbettungsverfahren verwendet werden. Die Umsetzung der Vorverarbeitung wird in Abschnitt 5.2 erläutert. Abschnitt 5.3 und Abschnitt 5.4 befassen sich mit der Implementierung von Anforderungs- bzw. Quelltexteinbettungen. Schließlich wird in Abschnitt 5.5 die Umsetzung der Abbildungsverfahren erklärt.

5.1 Worteinbettungsverfahren

Bei allem Einbettungsverfahren werden intern BERT oder fastText verwendet, um Wörter abzubilden. `WordEmbeddingCreator` bildet die Oberklasse der Worteinbettungsverfahren, die eine Methode zur Berechnung der Einbettung aus einem Wort und eine Methode zur Berechnung der Word Mover's Distance aus zwei Listen von Wörtern vorgibt.

`FastTextEmbeddingCreator` lädt fastText-Modell im `.bin`-Format mit Hilfe der Bibliothek gensim[1]. Es wurde gensim verwendet, da die Bibliothek auch eine vorimplementierte WMD-Funktion zur Verfügung stellt. Jedoch kann die Funktion nur direkt auf zwei Listen von Wörtern angewendet werden, zwischen denen die WMD berechnet werden soll. Eine Vorberechnung der Einbettungen ist hiermit folglich nicht möglich, was sich auch auf die Implementierungen der Abbildungsverfahren mit WMD auswirkt (siehe Abschnitt 5.5). fastText wurde hier mit dem englischen Standardmodell der fastText-Autoren [BGJM17] verwendet. Die genauen Modelle sind in Tabelle 5.1 benannt.

Die Klasse `FastTextAlignedEngItalEmbeddingCreator` implementiert die Funktionalität, mit zwei verschiedenen Modellen, die im gleichen Vektorraum ausgerichtet sind, umzugehen. Zum einen lädt diese Klasse Modelle aus `.vec`-Dateien, zum anderen musste hier die WMD-Funktion derart angepasst werden, um mit zwei Wortlisten umzugehen, die in verschiedenen Sprachen formuliert sein können. Diese Implementierung wird für die Abbildung von SMOS, welches englische und italienische Quelltextteile beinhaltet, benötigt. Die verwendeten fastText-Modelle für Englisch und Italienisch gingen aus der Arbeit von Conneau et al. [CLR+18] hervor. Sie sind im gleichen Vektorraum ausgerichtet, das heißt die Einbettungen eines Wortes und seiner Übersetzung liegen im Vektorraum nah beieinander. Zur Erzeugung dieser Modelle wurde ein Wörterbuch verwendet, um eine

[1] `https://pypi.org/project/gensim/`

Tabelle 5.1: Namen und Quellen der verwendeten Modelle für BERT und fastText

Modell	Dateiname(n)	Quelle
Standard-fastText-Modell (engl.)	cc.en.300.bin	[BGJM17]
Standard-BERT-Modell (engl.)	bert-base-uncased	[DCLT19]
Ausgerichtete Modelle (engl./ital.)	wiki.multi.en.vec / wiki.multi.it.vec	[CLR+18]
Nachtrainierte Modelle	avg3.vec / avg4.vec	[Tel20]

Transformationsmatrix zu trainieren, die die Einbettungen der verschiedenen Sprachen in den gleichen Vektorraum abbildet. Die ursprünglichen fastText-Autoren bieten ebenfalls ausgerichtete Modelle an, jedoch hat sich durch Versuche gezeigt, dass die Modelle von Conneau et al. etwas besser funktionieren.

Es wurde ebenfalls die Verwendung eines mit Anforderungsdaten nachtrainierten Anforderungsmodells umgesetzt. Das Anforderungsmodell stammt aus der Arbeit von Telge [Tel20]. Dort wurden aus Anforderungsbeschreibungen von 65 verschiedenen Projekten ein Korpus gebildet, welcher anschließend mit dem Skip-Gram-Vorgehen (siehe Abschnitt 2.6.4.2) zum Training eines fastText-Modells verwendet wurde. Danach wurde dieses Modell mit dem Standard-fastText-Modell kombiniert, indem die Vektoren normalisiert und gewichtet addiert wurden. Das Modell besteht aus zwei `.vec`-Dateien, wie auch in Tabelle 5.1 zu sehen ist. avg3.vec enthält die Einbettungen von Wörtern, während avg4.vec Einbettungen für Buchstaben-N-Gramme beinhaltet. Die Anbindung dieses speziellen Modells wurde in der Klasse `FineTunedFastTextEmbeddingCreator` implementiert.

Die Einbettungsvektoren von fastText werden außerdem in allen obigen Varianten auf die Länge Eins normiert, da dies für die Nutzung der WMD-Bibliotheksfunktion empfohlen wurde.

Für BERT wurde ein vortrainiertes BERT-Modell der Autoren [DCLT19] eingesetzt. Die Implementierung befindet sich in der Klasse `BertSentenceEmbeddingCreator`. Hier werden Satzeinbettungen aus Sätzen erzeugt, die der Einbettung des `[CLS]`-Tokens entspricht. Die Nutzung von BERT als Klassifikator bzw. Ähnlichkeitsmetrik wird in Abschnitt 5.5 erklärt.

5.2 Vorverarbeitung

Bevor Vorverarbeitungsschritte durchgeführt werden können, muss eine Portionierung (engl. Tokenizing) erfolgen, um einen Text in seine Wörter aufzutrennen. Für natürlichsprachigen Text wurde die Bibliothek `nltk`[2] verwendet, die bereits eingebaute Portionierer für Wörter und Sätze in mehreren Sprachen, darunter Englisch und Italienisch, anbietet. Die Implementierung befindet sich in der Datei `Tokenizer.py`. `nltk` ist eine häufig genutzte Bibliothek für die natürliche Sprachverarbeitung in Python und wurde auch hier verwendet.

Für Java-Quelltext geschieht die Portionierung in den Klassen in der Datei `CodeeASTTokenizer.py`. Zusätzlich wird der Syntaxbaum von Quelltext extrahiert. `javalang`[3] wurde als Zerteiler für Java eingesetzt. Diese Bibliothek ist in der Lage, aus Java-Quelltext einen abstrakten Syntaxbaum aufzubauen. Da das Evaluationsprojekt LibEST in C programmiert ist, wird auch ein C-Zerteiler namens `pycparser`[4] verwendet. Die Wahl viel auf diese

[2] `https://www.nltk.org/`
[3] `https://github.com/c2nes/javalang`
[4] `https://github.com/eliben/pycparser`

Tabelle 5.2: Implementierungen der Vorverarbeitungsschritte

Vorverarbeitungsschritt	Klasse
Trennung der Binnenmajuskelschreibweise	`CamelCaseSplitter`
Lemmatisierung	`Lemmatizer`
Transformation in Kleinbuchstaben	LowerCaseTransformer
Worttrennung	`Separator`
Stoppwortentfernung	`StopWordRemover`
Javastoppwortentfernung	`JavaCodeStopWordRemover`
URL-Entfernung	`UrlRemover`
Nichtbuchstabenentfernung	`NonLetterFilter`
JavaDoc-Filter	`JavaDocFilter`
Entfernung doppelter Leerzeichen	`DuplicateWhiteSpaceFilter`
Satzpunkt setzen	`AddFullStop`
Wortlängenfilter	`WordLengthFilter`

beiden Bibliotheken, da andere Zerteiler oft eine grafische Repräsentation des abstrakten Syntaxbaums erzeugen, die für diesen Anwendungsfalls nicht benötigt wird, oder nicht mit Python kompatibel sind. Für die Nutzung von `pycparser` müssen alle verwendete Typen in den Klassen deklariert sein, das gilt auch für Standardtypen. Fehlende Typen sollen nach der Empfehlung des Autors durch Stummel ersetzt werden. Dies muss auch hier durchgeführt werden, da in den LibEST-Dateien die meisten Importe fehlen. Die Stummel befinden sich im Ordner `fake_libc_include`. Ein Nachteil von `pycparser` besteht darin, dass Kommentare ignoriert werden. Aus diesem Grund wird zusätzlich die Bibliothek `comment-parser`[5] verwendet, welche die Kommentare extrahiert.

Die portionierten bzw. zerteilten Artefakte werden anschließend in eine Zwischenrepräsentation mit der Oberklasse `FileRepresentation` transformiert. Die abstrakten Syntaxbäume von Quelltext werden in eine `CodeFileRepresentation` umgewandelt. `JavaLangUtil.py`, `PycparserUtil.py` und `CommentParserUtil.py` enthalten Funktionalitäten, um eine `CodeFileRepresentation` aufzubauen.

Für die Anforderungen wird die Repräsentation durch den `Tokenizer` erzeugt. Es gibt verschiedene implementierte Repräsentationsmöglichkeiten. Je nachdem, ob zum Beispiel in ganzen Sätzen oder in einzelnen Wörter portioniert werden soll, wird eine andere (Unterklasse der) `FileRepresentation` erstellt. Pro Anforderungs- und Quelltextdatei wird also eine `FileRepresentation` aufgebaut, die die für die Einbettungen benötigten Wörter und Informationen speichert. Auf der `FileRepresentation` wird auch die Vorverarbeitung durchgeführt.

Nachdem die Wörter oder Sätze portioniert sind, können die Vorverarbeitungsschritte aus Abschnitt 4.2.4 bzw. Abschnitt 4.3.4 angewendet werden. Alle Vorverarbeitungsschritte sind in der Datei `Preprocessor.py` zu finden. Die Vorverarbeitung ist als Fließband aufgebaut und nimmt eine beliebige Anzahl an `PreprocessingStep`-Objekte entgegen. Die `PreprocessingStep`-Klassen sind in Tabelle 5.2 aufgelistet. Einige neue Vorverarbeitungsschritte, die nicht in Abschnitt 2.3 vorgestellt wurden, sind aus implementierungs- und projektspezifischen Gründen hinzugekommen.

Die Worttrennung trennt Wörter, an denen die Portionierer scheitern. Dazu gehören zum Beispiel zwei Wörter, die mit einem Slash konkateniert sind. Die URL-Entfernung filtert URLs heraus, die in manchen Artefkaten vorhanden sind. URLs liefern in der Regel keine

[5] `https://pypi.org/project/comment-parser/`

nützliche Semantik für die Rückverfolgbarkeit. Der JavaDoc-Filter entfernt JavaDoc-Tags wie „@param“. Bei manchen Vorverarbeitungsschritten werden einzelne Wörter entfernt, sodass mehrere Leerzeichen hintereinander stehen können. Zur besseren Darstellung kann der `DuplicateWhiteSpaceFilter` verwendet werden, um die zusätzlichen Leerzeichen zu entfernen. `AddFullStop` setzt am Satzende einen Punkt, falls noch kein Satzzeichen vorhanden ist. Bei BERT werden ganze Sätze inklusive Satzzeichen eingebettet.

Für die Lemmatisierung wurde die `nltk`-Bibliothek verwendet. Auch die Stoppwortentfernung greift auf die englischen und italienischen Stoppwortlisten von `nltk` zurück. Quelltextstoppwörter wurden manuell aus Quelltextsyntaxwörter und implementierungsnahe Typen zusammengestellt. Der Wortlängenfilter filtert alle Wörter mit zwei oder weniger Buchstaben heraus.

5.3 Anforderungseinbettungen

Einbettungen werden durch `EmbeddingCreator` bzw. ihre Unterklassen erzeugt. Sie werden mit den zu verwendenden Vorverarbeitungsschritten und dem Worteinbettungsverfahren parametrisiert. In der Einschubmethode `create_embeddings` wird pro Artefakt die korrespondierende `FileRepresentation` entgegengenommen. In den Unterklassen wird in dieser Methode definiert, wie daraus eine oder mehrere Einbettungen erzeugt werden soll. Anschließend wird ein `Embedding`-Objekt, welches den eigentlichen Einbettungsvektor kapselt, als Ergebnis zurückgegeben, falls Einbettungen erzeugt wurden.

Anforderungseinbettungen werden konkret durch Klassen in der Datei `RequirementEmbeddingCreator.py` erzeugt. `AverageWordEmbeddingCreator` bildet über alle Wörter einer Anforderung bzw. eines `FileRepresentation`-Objekts einen flachen Durchschnitt, der als Anforderungseinbettung definiert wird. `AverageSentenceEmbeddingCreator` erzeugt die Anforderungseinbettung durch eine zweistufige Aggregation, indem pro Satz über alle Einbettungen ihrer Wörter und anschließend nochmals über alle Satzeinbettungen ein Durchschnitt gebildet wird. `AverageBERTSentenceEmbeddingCreator` erzeugt Satzeinbettungen durch BERT, die danach für die Anforderungseinbettung mit dem Durchschnitt aggregiert werden. Die erzeugten Anforderungseinbettungen sind Instanzen der Klasse `RequirementEmbedding` in der Datei `Embedding.py`.

Die im Entwurf (Abschnitt 4.2.5) beschriebenen Verfahren mit Doc2Vec und die Aggregation durch eine gewichtete Summe wurden nicht implementiert. Der Einsatz eines softwarespezifischen Modells wurde durch die Nutzung eines nachtrainierten fastText-Modells wie in Abschnitt 5.1 beschrieben umgesetzt.

5.4 Quelltexteinbettungen

Quelltexteinbettungen werden durch die Klasse `CodeEmbedding` in der Datei `Embedding.py` repräsentiert. Pro Klasse wird ein `CodeEmbedding`-Objekt erzeugt. Diese Objekte können einen einzelnen Vektor besitzen, falls sie eine Klasseneinbettung repräsentieren. Wenn einzelne Klassenelementeinbettungen (zum Beispiel für den Mehrheitsentscheid) benötigt werden, können sie eine Liste solcher Einbettungen besitzen.

Unterklassen von `CodeEmbeddingCreator` implementieren in der Einschubmethode `create_embeddings` das Vorgehen, um aus einer übergebenen `FileRepresentation` (bzw. in diesem Fall `CodeFileRepresentation`) pro Klasse ein `CodeEmbedding`-Objekt zu erzeugen. `IdentifierEmbeddingCreator` ist ein `CodeEmbeddingCreator`. Diese Klasse und seine Unterklassen implementieren die verschiedenen Kombinationsmöglichkeiten der Klassenelemente aus dem Entwurf (Abschnitt 4.3.6). Als Aggregationsmethode wurde ausschließlich der Durchschnitt umgesetzt.

Für Quelltexteinbettungen mit Aufrufabhängigkeiten muss ein `MethodCallGraphEmbeddingCreator` verwendet werden, der `MethodCallGraphEmbedding`-Instanzen erzeugt. Da bei den Aufrufabhängigkeiten auf andere Methoden zugegriffen werden, müssen die Vektoren der Methoden unterscheidbar und identifizierbar sein. Dies wird durch `MethodCallGraphEmbedding` ermöglicht. In einem `CodeEmbedding`-Objekt sind die Klassenelementvektoren nicht unterscheidbar. Zur Extraktion von Aufrufabhängigkeiten in Java wurde `java-callgraph`[6] verwendet. `java-callgraph` ist ein Programm, dass den Aufrufgraphen für ein Java-Programm erzeugen und in einer Textdatei ablegen kann. Das Einlesen der Textdatei für die Einbettungserzeugung wird durch die Funktionen in der Datei `CallGraphUtil.py` bewerkstelligt.

Es wurden ausschließlich die Aufrufbeziehungen zwischen Java-Methoden berücksichtigt bzw. implementiert. Datenflussabhängigkeiten, Aufrufbeziehungen für C und zwischen Klassen wurden nicht umgesetzt.

5.5 Abbildungsverfahren

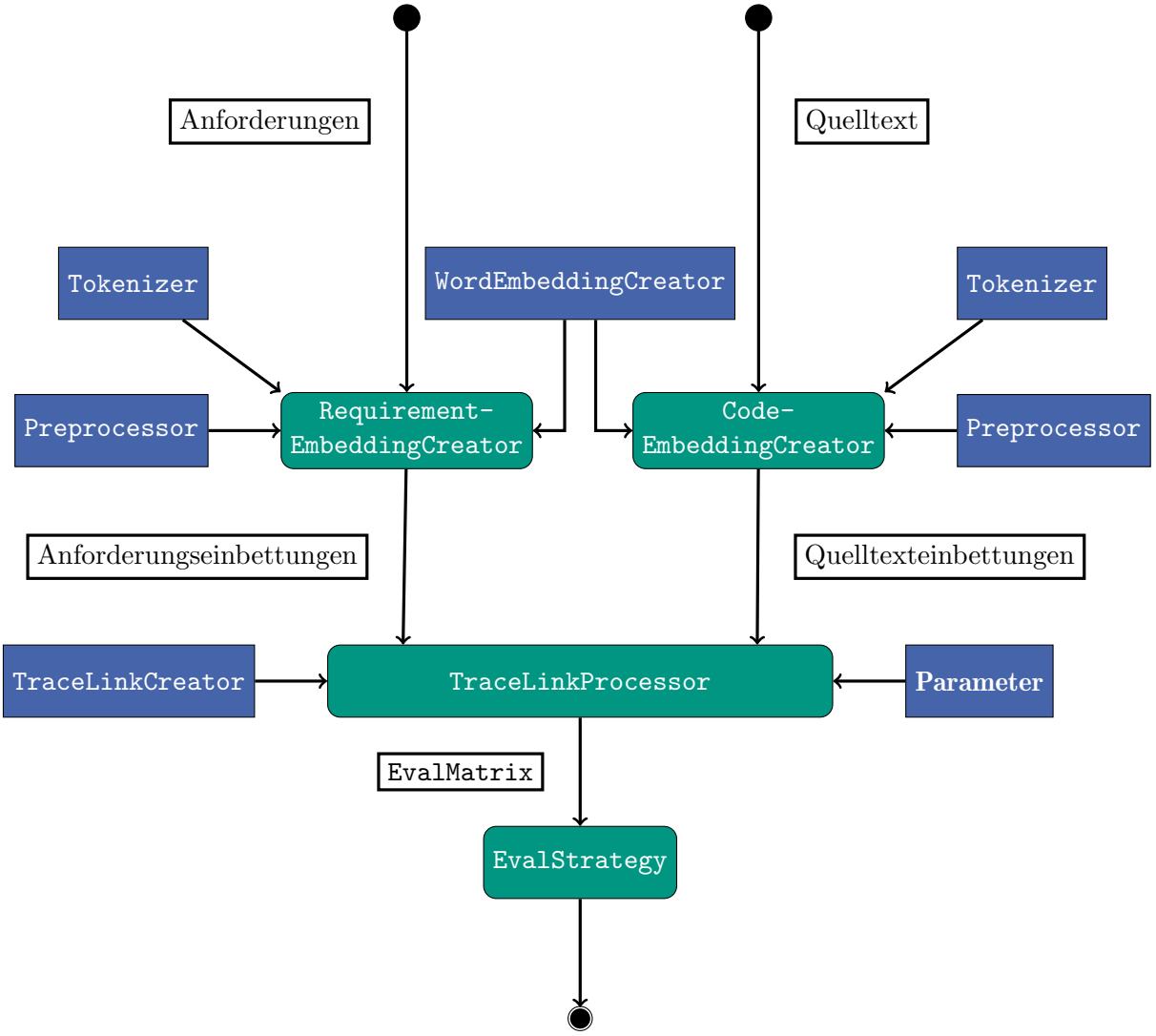


Abbildung 5.1: Ablauf der Einbettungserzeugung und der Abbildung

Nach der Erzeugung der Anforderungs- und Quelltexteinbettungen folgt die Generierung von Rückverfolgbarkeitsverbindungen. In Abbildung 5.1 ist der komplette Ablauf darge-

[6] `https://github.com/gousiosg/java-callgraph`

Tabelle 5.3: Anforderungs- und Klassenelementpaare für den BERT-Klassifikator

Anforderungselement	Klassenelement
Anforderungssatz	Methodenkommentarsatz
Anforderungssatz	Methodenbezeichner
Anforderungssatz	Methodenbezeichner mit Klassenname als Präfix
Ganze Anforderung	Ganzer Methodenkommentar

stellt. Ein `TraceLinkProcessor` nimmt die erzeugten Einbettungen entgegen, führt die Abbildung durch, vergleicht die Verbindungen mit der Musterlösung und speichert für alle Schwellwertkombinationen die resultierenden Ausbeute/Präzisionswertepaare in eine `EvalMatrix` ab.

Die `EvalMatrix` wird an eine `EvalStrategy` weitergereicht. Die Ausbeute/Präzisionswertepaare aus der `EvalMatrix` werden hier genutzt, um zu Beispiel F1-Werte oder die durchschnittliche Präzision zu berechnen.

Zu den Parametern, die ein `TraceLinkProcessor` benötigt, gehören unter anderem die Schwellwerte, nach denen die Ähnlichkeiten gefiltert werden sollen und die Funktionen für die Quelltext- bzw. Anforderungsähnlichkeitsreduktion. `TraceLinkCreator`-Objekte legen fest, wie die Einbettungen miteinander verglichen und wie Rückverfolgbarkeitsverbindungen hergestellt werden. Der `FileLevelCosSimTraceLinkCreator` vergleicht zum Beispiel Anforderungs- und Klasseneinbettungen mit der Kosinusähnlichkeit. Ein `MajorityTraceLinkCreator` führt einen Mehrheitsentscheid mit einem Mehrheitsentscheidschwellwert durch, während der `TopNTraceLinkCreator` die Stimmenbestimmung mit Top-N durchführt. Diese `TraceLinkCreator`-Klassen können sowohl mit Anforderungssätzen als auch mit ganzen Anforderungen umgehen. Damit wird der Mehrheitsentscheid aus dem Entwurf (Abschnitt 4.4.4) umgesetzt. Als Ähnlichkeitsreduktionsfunktionen für Quelltext und Anforderungen sind beide diskutierten Varianten (Durchschnitt und Maximum) implementiert. Die Stimmenbestimmung mit dem Maximum wurde jedoch nicht umgesetzt, sondern nur Top-N und der Mehrheitsentscheidschwellwert wie oben beschrieben.

Der Entwurf sieht außerdem die Berücksichtigung von Aufrufbeziehungen vor. Die Unterklassen von `CallGraphTLC` implementieren verschiedene Varianten, Aufrufsnachbarmethoden in den Mehrheitsentscheid miteinzubeziehen, wie etwa durch eine Ähnlichkeitssumme (`WeightedSimilaritySumCallMajorityTLC`), durch einen doppelten Mehrheitsentscheid (`CallGraphDoubleMajorityTLC`) oder durch eine Vektorsumme (`WeightedVectorSumCallMajorityTLC`).

Die Durchführung vieler Abbildungs- und Einbettungsverfahren ist zeitaufwändig. Deswegen müssen bei den meisten `TraceLinkProcessors` Einbettungen oder Ähnlichkeitspaare zwischen Einbettungen vorberechnet und als `.json`-Datei abgelegt werden. Die `TraceLinkProcessors` bieten dafür Methoden zur Vorberechnung und zum Laden dieser Dateien an.

Gemäß dem Entwurf in Abschnitt 4.4.4 sollen außer der Kosinusähnlichkeit auch die WMD und Ähnlichkeiten durch BERT implementiert werden.

Für die Ähnlichkeit mit BERT wird ein BERT-Klassifikator trainiert, welcher jeweils ein Anforderungs- und ein Quelltextelement entgegennimmt und eine Wahrscheinlichkeit berechnet, dass zwischen den beiden Elementen eine Rückverfolgbarkeitsverbindung besteht. In Tabelle 5.3 sind die Elementpaare aufgelistet. Zur Umsetzung wurde die Bibliothek

Tabelle 5.4: Trainingskonfiguration für BERT

Basismodell	Lernrate	Epochen	Batchgröße
bert-base-uncased	2e-05	4	32

`transformers`[7] verwendet, die auf `pytorch`[8] aufbaut. `pytorch` ist eine Bibliothek für die Domäne des maschinellen Lernens. Im Vergleich zu anderen Bibliotheken in diesem Bereich hält `pytorch` die Balance zwischen abstrakten und implementierungsnahen Schnittstellen und wird hier daher eingesetzt.
Für das Training werden jeweils 80% der Musterlösungsverbindungen von den Evaluationsprojekten Dronology, eTour und iTrust zufällig ausgewählt. Zwischen allen ausgewählten Klassen(-elementen) und Anforderungen werden (für jedes Projekt separat) künstliche Verbindungen hinzugefügt und als invalide Verbindung gekennzeichnet. Von den Klassen und Anforderungen, die keine Musterlösungsverbindung besitzen, werden ebenfalls 80% zufällig ausgesucht und der Trainingsmenge hinzugefügt, wo sie invalide Verbindungen bilden. Die Trainingsverbindungen werden durch die Funktionen in `DataExtractor.py` erstellt und als `.csv`-Dateien gespeichert. Um das Gleichgewicht zwischen der Anzahl an validen und invaliden Verbindungen herzustellen (es gibt viel mehr invalide), werden während dem Training nur so viele invalide Verbindungen zufällig ausgewählt, wie es valide Verbindungen gibt. Die Trainingsparameter sind in Tabelle 5.4 aufgelistet. Es wurde die empfohlene Parameterkonfiguration der BERT-Autoren [DCLT19] verwendet.
Die Implementierung des Trainings findet sich im Google-Colab-Notebook mit dem Namen `BERT_Training.ipynb`. Für die anschließende Nutzung von den nachtrainierten BERT-Modellen wurden die Klassen `BertPredictor` und `BertSelfTrainedTraceLinkProcessor` implementiert. Ersteres nutzt das BERT-Modell, um eine Wahrscheinlichkeit bzw. Ähnlichkeit zwischen einem Anforderungs- und Klassenelement zu berechnen. Letzteres setzt den Mehrheitsentscheid mit dem `BertPredictor` um. Hier muss beachtet werden, dass eigentlich keine Einbettungen erzeugt werden, daher benötigt der `BertSelfTrainedTraceLinkProcessor` keinen `EmbeddingCreator`. Außerdem wurden die 20% Testdaten wie oben beschrieben vor dem Training bereits bestimmt und liegen nicht mehr als Quelltext- oder Textdateiformat vor, sonder als `.csv`-Dateien.
Des Weiteren können Elementpaare aus Tabelle 5.3 kombiniert werden. Die Optionen sind: Anforderungssatz↔Methodenkommentarsatz mit Anforderungssatz↔Methodenbezeichner oder mit Anforderungssatz↔Methodenbezeichner(mit Klassenamenpräfix). `BertCombinedSelfTrainedTLP` ist der `TraceLinkProcessor`, der Datensätze aus zwei Quellen laden kann. Die Unterklassen von `CombinationTLC` Implementierungen verschiedene Kombinationsvorgehen. `SeparateMajorityCombinationTLC` führt für jede Quelle einen separaten Mehrheitsentscheid durch, um die zu verbindenden Anforderungen für die Klasse zu bestimmen. `UnionMajorityCombinationTLC` führt einen einzigen Mehrheitsentscheid durch, in der die Klassenelemente aus beiden Quellen abstimmen. `ReturnAllMajorityCombinationTLC` führt keinen Mehrheitsentscheid durch und definiert alle (vorgefilterten) Stimmen aus beiden Quellen als Verbindung. WMD ist eine weitere Ähnlichkeitsmetrik, die (ausschließlich für fastText) umgesetzt wurde. Wie in Abschnitt 5.1 bereits erwähnt, wird hierfür die WMD-Bibliotheksfunktion von `gensim` verwendet. Jedoch nimmt diese Funktion nicht zwei Listen an Einbettungen, sondern zwei Wortlisten entgegen. Dies hat zwei Konsequenzen zu Folge: Erstens müssen hier Anforderungen und Quelltext nicht wie im Entwurf beschrieben (Abschnitt 4.4.4) als Bag-Of-Embeddings, sondern als Bag-Of-Words repräsentiert werden. Zweitens ist es dadurch nicht mehr möglich, Einbettungen von An-

[7] `https://huggingface.co/transformers/`
[8] `https://pytorch.org/`

forderungen und Quelltext separat durch die `EmbeddingCreators` wie in Abbildung 5.1 zu berechnen. Zur Nutzung der Architektur wurden die Klassen `MockEmbeddingCreator` und `MockEmbedding` angelegt. Der `MockEmbeddingCreator` verhält sich äußerlich wie ein `EmbeddingCreator` - er liest die Anforderungen und Quelltexte ein und wendet die Vorverarbeitung auf sie an. Als Ergebnis erzeugt es Instanzen von `MockEmbedding`, die keine echten Einbettungsvektoren, sondern nur die vorverarbeiteten Daten in Form einer `FileRepresentation` enthalten. Welche Klassen- oder Anforderungselemente genutzt oder kombiniert werden, wird hier nicht durch die `EmbeddingCreators`, sondern durch den `TraceLinkProcessor` (bzw. in den Unterklassen von `FastTextWMDTLP`) bestimmt. Da alle WMD-`TraceLinkProcessors` auf vorberechneten Daten arbeiten, findet die Auswahl und Kombination der Elemente in den `precalculate_tracelinks`-Methoden statt. In Tabelle 5.5 sind die WMD-`TraceLinkProcessors` mit den Elementen, zwischen denen die WMD berechnet wird, aufgelistet.

Tabelle 5.5: Implementierte Abbildungsverfahren mit WMD

Klasse	Anforderungselement	Quelltextelement
`FastTextIdentifierWMDTLP`	Anforderung	Klasse Menge der Klassennamen und Methodensignaturbezeichner
`FastTextCommentWMDTLP`	Anforderung	Klasse als Menge der Kommentarwörter
`FastTextCommentLevelWMDTLP`	Anforderung	Kommentar
`FastTextFileLevel-IdentifierWMDTLP`	Anforderung	Methode als Signaturbezeichner mit Klassenname
`FastTextSentenceLevel-IdentifierWMDTLP`	Anforderungssatz	Methode als Signaturbezeichner mit Klassenname
`FastTextComment-IdentifierWMDTLP`	Anforderung	Methode als Signaturbezeichner mit Klassenname und Kommentarwörter
`FastTextComment-IdentifierSentenceWMDTLP`	Anforderungssatz	Methode als Signaturbezeichner mit Klassenname und Kommentarwörter

6 Evaluation

Die Evaluation soll überprüfen, wie gut die verschiedenen implementierten Abbildungsverfahren, Anforderungs- und Quelltexteinbettungen im Vergleich zueinander und zu anderen Arbeiten funktionieren. Es wurden dafür verschiedene Projekte ausgesucht, an denen eine Evaluation der Rückverfolgbarkeit zwischen Anforderungen und Quelltext möglich ist. Diese werden in Abschnitt 6.1 beschrieben. Die Metriken, anhand derer evaluiert wird, sind in Abschnitt 6.2 aufgeführt. Der Vergleich der verschiedenen Abbildungs- und Einbettungsoptionen findet in Abschnitt 6.3 statt. Die projektspezifischen und projektübergreifenden besten Verfahren werden in Abschnitt 6.4 evaluiert. Schließlich wird in Abschnitt 6.5 der Vergleich zu den Ergebnissen anderer Rückverfolgbarkeitsarbeiten gezogen.

6.1 Datensätze und Modelle

In dieser Arbeit soll die Rückverfolgbarkeit zwischen Anforderungen und Quelltext evaluiert werden, daher müssen die Testdatensätze Anforderung und dazu passende Quelltextdateien besitzen. Außerdem wird ein Goldstandard benötigt, der die richtigen Rückverfolgbarkeitsverbindungen des Testprojekts auflistet. Dies ist notwendig, damit die durch Einbettungen erzeugten Verbindungen auf ihre Korrektheit überprüft werden können. Es existieren Projekte, die bereits einen Goldstandard besitzen. Alternativ kann auch ein eigener Goldstandard erstellt werden. Da in dieser Arbeit auch der Vergleich zu anderen Arbeiten gezogen werden soll, werden Projekte mit existierenden Goldstandards bevorzugt.

Anhand dieser beiden Kriterien wurden zunächst die folgenden fünf Projekte ausgewählt: eTour, iTrust [HSP18], Dronology [CVB18], SMOS [GOPL11] und LibEST [MPB+20]. Die SMOS-, eTour- und iTrust-Datensätze sind auch auf der CoEST-Website[1] verfügbar. CoEST ist eine Gemeinschaft, die die Forschung zur Rückverfolgbarkeit fördert.

Die Eigenschaften der Datensätze sind in Tabelle 6.1 und Tabelle 6.2 zusammengefasst. Die zweite und dritte Spalte der ersten Tabelle zählen die Anzahl der Anforderungs- und Quelltextdateien. Die Verb.-Spalte stellt die Anzahl an Rückverfolgbarkeitsverbindungen gemäß des Goldstandards dar. Die zwei nachfolgenden Prozent-Spalten errechnen sich wie folgt: Es werden alle Anforderungsdateien mit allen Quelltextdateien verbunden und nichts herausgefiltert. Das ist also die maximale Anzahl an Verbindungen, die theoretisch möglich sind. Indem die Anzahl an Goldstandardlinks durch die Gesamtzahl geteilt wird, erhält

[1] `http://sarec.nd.edu/coest/datasets.html`

Tabelle 6.1: Evaluationsdatensätze für die Rückverfolgbarkeit

Projekt	Anf.	Quell.	Verb.	Naive Präz.	Naive F1	A. in MuLö	Q. in MuLö
eTour	58	113	362	5,5%	10,4%	100%	83,2%
iTrust	131	226	286	1%	2%	80,2%	38,5%
SMOS	67	98	1044	15,9%	27,4%	100%	68,4%
Dronology	99	43	60	1,4%	2,8%	24,2%	25,6%
LibEST	52	14	204	28%	43,8%	78,6%	90%

Tabelle 6.2: Sprache und Typ der Evaluationsdatensätze

Projekt	A.sprache	Q.sprache	Progr.sprache	Typ
eTour	Ital/Eng	Ital/Eng	Java	UC↔Klasse
iTrust	Eng	Eng	Java	UC↔Klasse
SMOS	Ital	Ital/Eng	Java	UC↔Klasse
Dronology	Eng	Eng	Java	Anf↔Klasse
LibEST	Eng	Eng	C	Anf↔.c/.h

man eine Präzision bei 100% Ausbeute und kann daraus wiederum den F1-Wert berechnen (siehe auch Abschnitt 6.2). Die letzten beiden Spalten geben an, wie viel Prozent der Anforderungen bzw. Quelltextdateien mindestens einmal im Goldstandard auftauchen, das heißt wie viele Dateien mindestens eine Rückverfolgbarkeitsverbindung besitzen.
Die zweite Tabelle listet die Anforderungs-, Quelltext- und Programmiersprache der Projekte auf. Die letzte Spalte zeigt an, mit welchen Dateien die Anforderungen im Goldstandard verbunden sind. Im Falle von eTour, iTrust und SMOS sind es eigentlich keine Anforderungen, aber Anforderungsfälle.

Der eTour-Datensatz ist fehlerhaft und unvollständig. Er enthielt ursprünglich 116 Quelltextklassen. Davon sind zwei Testklassen dabei, die „TestoNewsRendererTest“ und „NewsTableModelTest“ heißen. Da im Datensatz die Klassen „TestoNewsRenderer“ und „NewsTableModel“, welche beide fehlen, referenziert werden, ist anzunehmen, dass die beiden Testklassen irrtümlicherweise anstelle der richtigen Klassen in den Datensatz eingefügt wurden. Außerdem ist eine andere Quelltextklasse syntaktisch nicht korrekt und auch nicht reparierbar. Diese drei Klassen wurden daher für die Evaluation entfernt. Dadurch verbleiben noch 113 Quelltextdateien. Der Goldstandard enthält eigentlich 385 Rückverfolgbarkeitsverbindungen. Jedoch referenziert er auch Klassen, die nicht im Datensatz enthalten sind. Diese Rückverfolgbarkeitsverbindungen sind unmöglich zu finden und wurden daher aus dem Goldstandard entfernt.
Weiterhin sind die Artefakte in eTour teilweise auf Englisch, teilweise auf Italienisch formuliert. Die Anforderungen haben die Form von Anwendungsfällen und besitzen Titel auf Italienisch, während der Rest auf Englisch geschrieben ist. Der Quelltext besitzt englische Kommentare, aber italienische Bezeichner. Für die Evaluation wurden die italienischen Bestandteile händisch übersetzt, um ein komplett englisches Modell anwenden zu können. Dabei wurde ein Englisch-Italienisch-Wörterbuch verwendet, um die einzelnen Wörter zu übersetzen. Gleiche italienische Wörter werden dabei mit den gleichen englischen Wörter übersetzt, falls sie an mehreren Stellen auftreten.

Im iTrust-Datensatz sind außer Quelltextdateien auch JSP-Dateien (JavaServer Pages) enthalten, zu denen es auch einen Goldstandard gibt. Diese Dateien können zwar teilweise

Java-Quelltext beinhalten, jedoch sind diese Quelltextausschnitte nicht in Klassen oder Methoden gegliedert. Außerdem sind diese Dateien oft mit anderen Formaten wie HTML gemischt. Aus diesen Gründen wurden sie für die Anforderung-zu-Quelltextrückverfolgbarkeit nicht herangezogen. Im Quelltextordner von iTrust war eine außerdem eine Nicht-Java-Datei enthalten; diese wurde entfernt.

SMOS ist komplett italienisch bis auf die Bezeichner im Quelltext - diese sind Englisch. Eine Übersetzung wurde hier nicht vorgenommen. Außerdem wurden zwei Nicht-Quelltextdateien entfernt, die fälschlicherweise im Datensatz enthalten waren. Damit verbleiben 98 von 100 Quelltextdateien.

Der Dronology-Datensatz ist unvollständig. Die vorhandenen Anforderungen und Quelltextdateien passen nur teilweise mit der Musterlösung zusammen. Wie man in Tabelle 6.1 sehen kann, sind nur jeweils ca. ein Viertel der Anforderungen und Quelltextdateien in der Musterlösung enthalten. Des Weiteren enthielt die Musterlösung ursprünglich 874 Rückverfolgbarkeitsverbindungen. Davon sind nur 60 verblieben, da die anderen Verbindungen 209 fehlende Klassen referenzierten.

LibEST ist in der Programmiersprache C geschrieben und damit in Kontrast zu den anderen Projekten nicht objektorientiert. Die `.c`- und `.h`-Dateien enthalten Methoden bzw. Methodensignaturen und Kommentare, die trotzdem für die Erzeugung von Einbettungen genutzt werden können. Es existieren gemäß der Musterlösung Verbindungen zu `.c`- und `.h`-Dateien, wobei die Verbindung zu einer `.c`-Datei nicht unbedingt eine Verbindung zur passenden `.h`-Datei impliziert.

Die Modelle, die für fastText und BERT verwendet wurden, sind in Abschnitt 5.1 aufgeführt.

6.2 Evaluationsmetriken

Evaluationsmetriken beurteilen Verfahren und machen sie so vergleichbar mit anderen Techniken. In dieser Arbeit wurden verschiedene Methoden vorgestellt, die eine Anforderung-zu-Quelltextrückverfolgbarkeit umsetzen sollen. Bei allen Verfahren werden vorläufige Verbindungen zwischen den Artefakten hergestellt, die durch einen oder mehrere Schwellwert(e) gefiltert werden. Am Ende bleibt eine Liste an Rückverfolgbarkeitsverbindungen zwischen Anforderungen und Klassen übrig. Auf dieser Ergebnisliste werden mit Hilfe des Goldstandards die folgenden Metriken berechnet. Als ein richtig positives Ergebnis (engl. true positive, im Folgenden kurz TP) gilt eine Verbindung, die sowohl in dieser Ergebnisliste, als auch im Goldstandard zu finden ist. Ein fälschlich positives Ergebnis (engl. false positive, im Folgenden kurz FP) ist dagegen eine Verbindung, die in der Ergebnisliste vorhanden ist, aber keine valide Verbindung gemäß dem Goldstandard darstellt. Das sind die Rückverfolgbarkeitsverbindungen, die das Verfahren fälschlicherweise herstellt. Die komplette Ergebnisliste setzt sich also aus der Vereinigung der TPs und FPs zusammen. Fälschlich negative Ergebnisse (engl. false negatives, kurz FMs) sind Verbindungen, die im Goldstandard aufgelistet sind, aber nicht durch das Verfahren gefunden werden, das heißt nicht in der Ergebnisliste enthalten sind. Die gesamte Menge der Goldstandardverbindungen ist demnach die Vereinigung der TPs und FNs. Richtig negative Ergebnisse (engl. true negatives, kurz TNs) sind falsche Verbindungen, die nicht im Goldstandard stehen und korrekterweise auch nicht in der Ergebnisliste zu finden sind.

Ausbeute
Die Ausbeute [JM09] gibt an, wie viel Prozent der im Goldstandard aufgelisteten Verbindungen in der Ergebnisliste vorhanden sind. Wie in Gleichung 6.1 zu sehen, werden dafür

die TPs durch die Anzahl an Musterlösungsverbindungen geteilt. Diese besteht, wie oben erklärt, aus TPs und FNs.

$$Ausbeute = \frac{TP}{TP + FN} \tag{6.1}$$

Präzision
Die Präzision [JM09] gibt an, wie viel Prozent der Verbindungen in der Ergebnisliste korrekt sind, das heißt auch im Goldstandard vorhanden sind. Hierzu wird die Anzahl der TPs durch die Größe der Ergebnisliste geteilt, wie auch in Gleichung 6.2 dargestellt ist. Die Ergebnisliste setzt sich aus TPs und FPs zusammen.

$$\text{Präzision} = \frac{TP}{TP + FP} \tag{6.2}$$

F_1-Maß
Der F_1-Wert [JM09] ist eine weitere gängige Kennzahl, um die Ergebnisse zu beurteilen. Es stellt das harmonische Mittel zwischen der Präzision und der Ausbeute dar und wird wie in Gleichung 6.3 kalkuliert. Nur mit Ausbeute und Präzision kann es schwierig sein, Verfahren miteinander zu vergleichen, wenn zum Beispiel ein Verfahren höhere Präzision, während ein anderes eine höhere Ausbeute besitzt. Durch den F_1-Wert werden beide Metriken zu einer einzigen kombiniert, sodass einfacher beurteilt werden kann, welches Verfahren besser funktioniert.

$$F_1 = 2 \cdot \frac{\text{Ausbeute} \cdot \text{Präzision}}{\text{Ausbeute} + \text{Präzision}} \tag{6.3}$$

Durchschnittliche Präzision

Für die Ergebnisliste werden die Rückverfolgbarkeitsverbindungen durch mindestens einen Schwellwert gefiltert. Die Schwellwerte können variiert werden, was zu unterschiedlichen Präzisionen, Ausbeuten und F1-Werten führt. Die oberen drei Metriken berechnen also jeweils die Werte an einem festen Schwellwert. Die durchschnittliche Präzision fasst dabei die Präzision über variierte Schwellwerte zusammen. Jedoch berechnet sich diese Metrik nicht wie ein üblicher Durchschnitt („Alles aufsummieren und durch die Anzahl teilen“). Die durchschnittliche Präzision ist bei einer kontinuierlichen Ausbeute-Präzisionskurve das Integral, das heißt als Flächeninhalt unter der Kurve. Beim diskreten Fall, der hier durch die Auswertung an diskreten Schwellwerten vorliegt, wird die durchschnittlichen Präzision wie in Gleichung 6.4 kalkuliert [BR99].

$$\text{Durchschnittliche Präzision} = \sum_n (\text{Ausbeute}_n - \text{Ausbeute}_{n-1}) \cdot \text{Präzision}_n \tag{6.4}$$

n ist dabei die Nummer des dazugehörigen Schwellwerts. Der Startwert für die erste Ausbeute ist Null. Es sollten möglichst viele Ausbeute-Präzisionspaare ausgewertet werden, um sich dem Integral bestmöglich anzunähern.

Die durchschnittliche Präzision wird in einer Vergleichsarbeit verwendet, daher wird sie auch in dieser Arbeit berechnet.

Bei allen vier Metriken liegt der Wertebereich zwischen Null und Eins, wobei Eins der bestmögliche Wert ist.

6.3 Evaluation der Einbettungs- und Abbildungsverfahren

In diesem Abschnitt werden die verschiedenen Verfahren verglichen, die im Entwurf vorgestellt wurden. Dazu wird in Abschnitt 6.3.1 das Vorgehen bei der nachfolgenden Evaluation erklärt. Anschließend werden die Abbildungsverfahren inklusive verschiedener Einbettungsmöglichkeiten für Anforderungen und Quelltexte nacheinander evaluiert. Aus dem Entwurf ist hervorgegangen, dass die Abbildung grundsätzlich auf verschiedenen Ebenen stattfinden kann. Die Abbildung auf der Klassenebene wird in Abschnitt 6.3.2 ausgewertet. Dort werden auch verschiedene Optionen verglichen, die Klassen- und Anforderungseinbettungen zu erzeugen. Abbildungen wie der Mehrheitsentscheid, die sich aus der Elementebene ableiten, werden in Abschnitt 6.3.3 evaluiert. Schließlich gibt es noch Variationen in der Wahl der Ähnlichkeitsmetrik, was in Abschnitt 6.3.4 geprüft wird.

6.3.1 Evaluationsmethodik

Das Ziel in diesem Abschnitt ist der Vergleich der verschiedenen Möglichkeiten, Anforderungs- und Quelltexteinbettungen zu erzeugen und sie aufeinander abzubilden. Hierbei hängt es auch vom Abbildungsverfahren ab, welche Einbettungsverfahren in Frage kommen. Die folgenden Abschnitte betrachten daher nacheinander die unterschiedlichen Abbildungsverfahren und prüfen jeweils verschiedene Optionen für die Einbettungen, die dabei verwendet werden können.

Es wird hierbei nicht auf allen fünf vorgestellten Projekten evaluiert. Wie bereits angesprochen ist Dronology unvollständig - der Großteil der in der Musterlösung referenzierten Artefakte ist nicht vorhanden.
LibEST ist in C programmiert - hier kann der Entwurf dieser Arbeit, der sich auf objektorientierten Programmen stützt, folglich nicht vollständig ausgeschöpft werden.
SMOS ist zweisprachig. Da die Abbildung von multilingualen Einbettungen nicht Fokus dieser Arbeit ist, wird SMOS nur für den Vergleich mit verwandten Arbeiten, die auch SMOS verwendet haben, benutzt.
eTour wurde komplett ins Englische übersetzt und bietet eine hohe Abdeckung der Artefakte in der Musterlösung. Die Evaluation wird aus diesen Gründen primär auf dem eTour-Datensatz durchgeführt, um die verschiedenen Einbettungs- und Abbildungsverfahren zu vergleichen.
Diesbezüglich wurde auch auf iTrust evaluiert. Die Auswertung davon befindet sich im Anhang in Abschnitt A.1.

6.3.2 Abbildung auf der Klassenebene

Für die Abbildung auf der Klassenebene werden pro Klasse und Anforderung jeweils eine einzige Einbettung erzeugt, die danach mit der Kosinusähnlichkeit paarweise verglichen werden. Die Kosinusähnlichkeit ist die Metrik, die in den meisten vergangenen Arbeiten aus der Literatur zur Rückverfolgbarkeit auch verwendet wurde. Andere Ähnlichkeitsmetriken werden in Abschnitt 6.3.4 behandelt. Die Paare werden durch einen Ähnlichkeitsschwellwert gefiltert und alle übrigen Paare werden als Rückverfolgbarkeitsverbindungen angenommen und mit dem Goldstandard evaluiert. Zur Erzeugung der Anforderungs- bzw. Klasseneinbettung wurden verschiedene Möglichkeiten evaluiert. Die Zusammensetzung der Klasseneinbettung wird in Abschnitt 6.3.2.1 diskutiert und die der Anforderungseinbettung in Abschnitt 6.3.2.2.

6.3.2.1 Zusammensetzung der Klasseneinbettungen

In diesem Abschnitt werden verschiedene Möglichkeiten verglichen, eine Klasseneinbettung zu erzeugen. Im Folgenden werden Anforderungseinbettungen immer auf die gleiche Weise (aus dem Durchschnitt ihrer Worteinbettungen) erzeugt. Das zugrundeliegende

Worteinbettungsverfahren für Anforderungen und Quelltext ist fastText mit dem vortrainierten Standardmodell der fastText-Autoren (siehe Implementierung Abschnitt 5.1). Die Abbildung mit dem BERT-Standardmodell wurde ebenfalls untersucht, jedoch liefern sie schlechtere Ergebnisse als mit fastText. Das liegt daran, dass BERT ganze Sätze entgegennimmt und Quelltext außer in den Kommentaren keine natürlichen ganzen Sätze aufweist, was die Einbettung durch BERT verzerrt. Daher wird im Folgenden nur mit fastText gearbeitet. Jedoch kann BERT als Ähnlichkeitsmetrik genutzt werden, was in Abschnitt 6.3.4.1 evaluiert wird.
Die Vorverarbeitung der Artefakte verläuft wie in Aufzählung 4.1 bzw. Aufzählung 4.4. Aus allen Klassen wird jeweils ein Durchschnitt ihrer Elemente erzeugt. Wie im Entwurf Abschnitt 4.3.6 erklärt, hat sich der Durchschnitt für Einbettungen in der Vergangenheit als valide Möglichkeit herausgestellt. Andere Aggregationsverfahren wurden nicht umgesetzt bzw. evaluiert. Anschließend werden alle Klassen- und Anforderungseinbettungen paarweise verbunden und die Kosinusähnlichkeiten zwischen ihnen berechnet. Die Verbindungen werden mit einem Schwellwert gefiltert. Alle übrigen Verbindungen stellen somit das Ergebnis des Verfahrens dar und können mit dem Goldstandard ausgewertet werden. Dadurch kann die Ausbeute und Präzision bezüglich des Schwellwerts berechnet werden. Die Auswahl der Elemente, die zum Durchschnitt für die Klasseneinbettung verwendet werden, kann variieren, wie im Entwurf beschrieben wurde (siehe Tabelle 4.3 und Aufzählung 4.3). Es soll hier evaluiert werden, welche Klassenelemente wie gut zur Rückverfolgbarkeit beitragen. Eine grundlegende Möglichkeit ist der Durchschnitt über alle Methodensignaturen, wobei jede Methodensignatur ein Durchschnitt ihres Namens, ihrer Parameter (Typ und Name), ihres Rückgabetyps und des Klassennamens ist. Wie in Abschnitt 4.3.6 diskutiert, werden ausschließlich öffentliche Methoden berücksichtigt.
Eine weitere Option ist es, den Klassennamen nochmal zusätzlich zu berücksichtigen, das heißt der Durchschnitt bildet sich aus dem Klassennamen und allen Methodensignaturen. Die Methodensignaturen werden dabei wie oben beschrieben gebildet.
Der Großteil der Funktionalität ist in Methoden implementiert, daher ist aus dem Entwurfskapitel hervorgegangen, dass Methoden stets in der Kombination enthalten sein sollten. Um die Signifikanz der Methoden zu überprüfen, wurde eine Kombination nur aus dem Klassennamen und dem Klassenkommentar gebildet (also ohne Methoden).
Außerdem wurde die naive Möglichkeit betrachtet, den flachen Durchschnitt über alle Bezeichner in der Klasse als Klasseneinbettung zu definieren.
Statt dem flachen Durchschnitt können die Elemente auch zweistufig aggregiert werden: Für Methodensignaturen, -rümpfe, -kommentare, Attribute, Attributkommentare, Klassennamen, Klassenkommentare, Superklassifizierernamen und innere Klassifizierer können aus ihren Wörtern jeweils ein erster Durchschnitt gebildet werden, die anschließend nochmal mit einem Durchschnitt zur Klasseneinbettung transformiert werden.

Für diese fünf oben beschriebenen Varianten wurde die Rückverfolgbarkeit wie zu Beginn des Abschnitts beschrieben durchgeführt. Durch Variation des Schwellwertes, hier als Dateilevelschwellwert (D-SW) bezeichnet, konnten viele Ausbeute-Präzisionswertepaare erzeugt und zu einer Kurve geformt werden, die für die fünf Varianten in Abbildung 6.1 zu sehen sind. Die Schwellwerte wurden von Null bis Eins in 0,01er Schritten variiert. Die Punkte der Kurven entsprechen jeweils Schwellwerten, an denen die Ausbeute und Präzision berechnet wurden. Je weiter links der Punkt liegt, desto größer ist der Schwellwert: Durch einen großen Schwellwert werden viele Verbindungen herausgefiltert, was zu einer niedrigen Ausbeute führt. Außerdem befindet sich in der rechten oberen Ecke des Diagramms der Punkt mit 100% Ausbeute und Präzision. Je höher und weiter rechts also ein Punkt bzw. eine Kurve liegt, desto besser. Zur Beurteilung der Kurven eignen sich weiterhin die Werte an den höheren Schwellwerten mehr als die niedrigeren. Bei niedrigen Schwellwerten können noch viele falsche Verbindungen in der Ergebnisliste vorhanden sein. Erst bei hohen Schwellwerten zeigt sich, ob die falschen Verbindungen herausgefiltert und

die richtigen beibehalten werden, da im Idealfall nur die richtigen Verbindungen eine hohe Ähnlichkeit besitzen.

In Tabelle 6.3 sind die besten F1-Werte der Verfahren aufgelistet. Der F1-Wert wurde mit den Werten in den Ausbeute- und Präzisionsspalten berechnet. Die D-SW-Spalte gibt den Dateilevelschwellwert an, an dem die Ausbeute und Präzision gemessen wurde. Die letzte Spalte zeigt die durchschnittliche Präzision der Varianten. Durch Variation des D-SWs werden Ausbeute-Präzisionspaare erzeugt, mit dessen Hilfe die durchschnittliche Präzision gemäß der Formel in Gleichung 6.4 berechnet wird. Auch hier beträgt die Variation von Null bis Eins in 0,01er Abständen. Die durchschnittliche Präzision entspricht dem Flächeninhalt unter der jeweiligen Kurve in Abbildung 6.1.

In dieser Auswertung ist zu sehen, dass der Durchschnitt zwischen Klassennamen und Methodensignaturen bessere Ergebnisse liefert als wenn nur die Methodensignaturen berücksichtigt werden. Sowohl der F1-Wert als auch die durchschnittliche Präzision sind höher. Der Klassenname trägt also wie vermutet Semantik in sich, die wichtig für die Rückver-

Tabelle 6.3: Beste F1-Werte für Klasseneinbettungsvarianten mit Methodensignaturen, Klassennamen oder allen Elementen

Variante	Bestes F1	Ausbeute	Präz.	D-SW	D. Präz.
Methodensignatur (`SIG`)	27,8%	29,6%	26,2%	0,69	18,6%
Klassenname, Meth.sig. (`SIG+KL`)	**31,9%**	30,7%	33,2%	0,69	23,5%
Klassenname, Klassenkommentar	18,8%	29%	13,9%	0,66	12,5%
Alle Bezeichner in Klasse	22,0%	40,6%	15,1%	0,84	16,0%
Alle Klassenelem. zweistufig	29,8%	28,7%	31%	0,78	**27,2%**

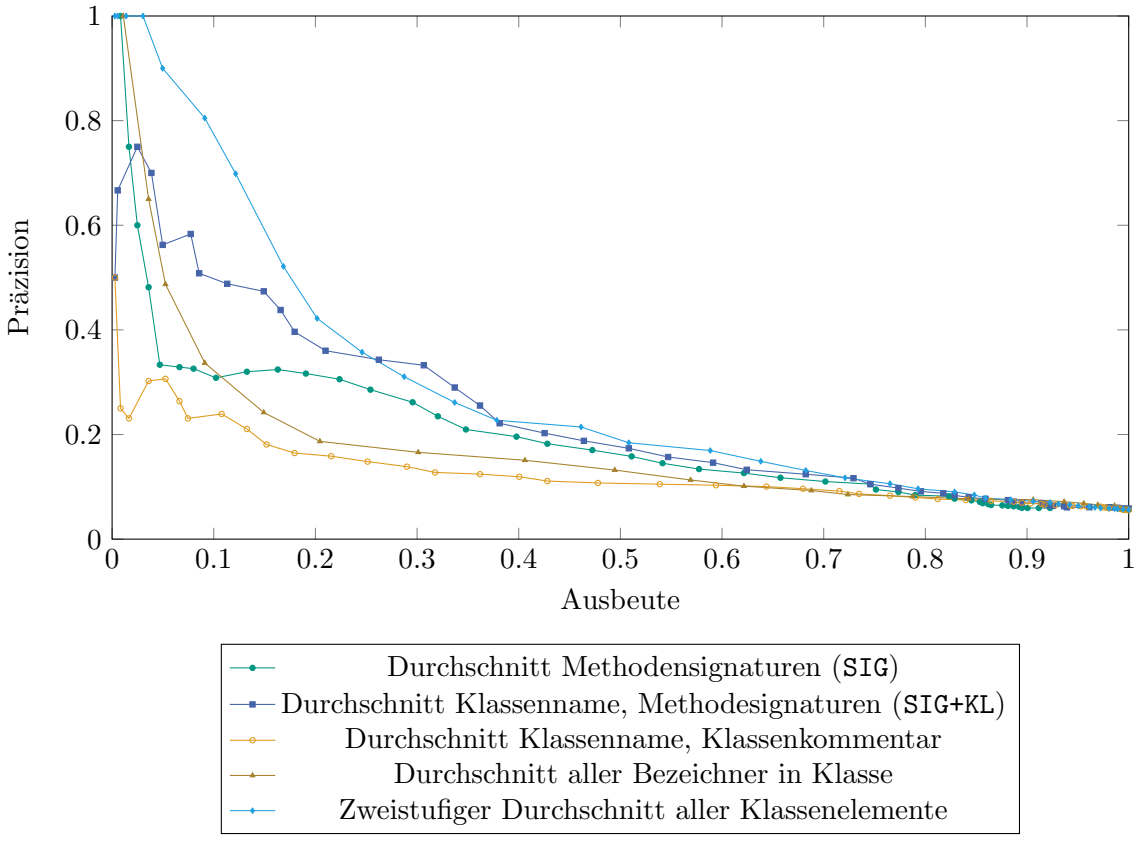


Abbildung 6.1: eTour: Ausbeute-Präzisionskurven für Klasseneinbettungensvarianten mit Methodensignaturen, Klassennamen oder allen Elementen

folgbarkeit ist. In der Methodensignaturvariante wird der Klassenname eigentlich bereits miteingerechnet, jedoch verliert es an Gewicht, wenn er mit den anderen Methodensignaturwörtern aggregiert wird, die danach nochmals zur Klasseneinbettung aggregiert werden. Durch den zusätzlichen Einbezug des Klassennamens bei der zweiten Aggregation hat er mehr Gewicht bekommen, was sich positiv auf die Rückverfolgbarkeit ausgewirkt hat. Dies alles deckt sich mit der Analyse, dass der Klassenname tendenziell der wichtigste Bezeichner in der Klasse ist, da er die komplette Klasse beschreibt und in der Regel abstrakter als die anderen Bezeichner ist.
Der Durchschnitt zwischen Klassenname und -kommentar ohne Methoden schneidet wie erwartet schlechter ab als die Varianten mit Methoden. Methoden implementieren zu viel Funktionalität, als dass sie ausgelassen werden können.
Der Durchschnitt über alle Bezeichner einer Klasse ist ebenfalls in Abbildung 6.1 eingetragen. Es ist zu sehen, dass dieser deutlich schlechter als die Variante mit Methodensignatur und Klassenname ist . Bei einem flachen Durchschnitt verlieren wichtige Elemente wie der Klassenname an Einfluss. Elemente, die oft verrauscht sind (wie zum Beispiel der Bezeichner einer Schleifeniterationsvariable) werden hier nicht herausgefiltert und fließen mit dem gleichen Gewicht wie ein Klassenname in den Durchschnitt mit ein. Das führt dazu, dass das Endergebnis auch verrauscht wird.
Die zweistufige Aggregation aller Elemente hat im Vergleich hierzu eine deutliche Verbesserung verursacht. Bei höheren Schwellwerten sind die Ausbeute-Präzisionswerte sogar besser als bei der Methodensignaturvariante mit Klassenname. Dies lässt sich auch an der durchschnittlichen Präzision beobachten, die fast 4% größer ist. Der beste F1-Wert ist allerdings etwas schlechter. Zwischen der zweistufigen Aggregation aller Elemente und der Methodensignatur mit Klassenname kann also nicht ein eindeutig überlegeneres Verfahren festgelegt werden. Die Verbesserung gegenüber dem flachen Durchschnitt kann dadurch erklärt werden, dass Elemente wie der Klassenname hier implizit mit einem höheren Gewicht in die Klasseneinbettung miteingerechnet werden: Beim flachen Durchschnitt wird durch die Anzahl aller Wörter in der Klasse normiert, hier wird durch aber die Anzahl der Klassenelemente geteilt. Letzteres ist viel kleiner, dadurch erhält der Klassennamenvektor vergleichsweise höheres Gewicht. Eine Evaluation mit iTrust führt zu ähnlichen Schlussfolgerungen (siehe Abbildung A.1). Dort besitzt die zweistufige Aggregation den besten F1-Wert, während die Kombination aus Methodensignatur und Klassennamen die bessere durchschnittliche Präzision erzielt. Auch hier kann kein Favorit zwischen den beiden Verfahren bestimmt werden.

Bei manchen Kurven fällt auf, dass sie bei niedriger Ausbeute (das heißt hohen Schwellwerten) einen Knick haben. Im Idealfall steigt die Präzision, je niedriger die Ausbeute ist, das heißt es werden mehr falsche als richtige Verbindungen aussortiert. Ein Knick bedeutet, dass hier durch einen größeren Schwellwert richtige Verbindungen herausgefiltert wurden, während falsche Verbindungen verblieben sind. Dies entsteht dadurch, dass die falschen Verbindungen eine größere Kosinusähnlichkeit aufweisen als die richtigen. Daraus kann die Schlussfolgerung gezogen werden, dass die Verfahren bei hohen Ähnlichkeitswerten noch nicht sehr gut zwischen richtigen und falschen Verbindungen unterscheiden können und daher noch nicht zuverlässig genug funktionieren.
Von den bisherigen fünf Varianten erzielten diejenigen mit der Methodensignatur und dem Klassennamen sowie die mit der zweistufigen Durchschnittsbildung die besten Werte. Für den Rest dieses Abschnitts wird Ersteres als `SIG+KL` referenziert und als Grundlinie angenommen, da sie das Minimum an Klassenelementen berücksichtigt, die relativ sicher nützlich für die Rückverfolgbarkeit sind. `SIG+KL` wird im Folgenden um weitere Klassenelemente erweitert und es wird geprüft, ob dadurch eine Verbesserung oder Verschlechterung der Ergebnisse erzielt wird.

Methoden besitzen nicht nur eine Signatur, sondern auch einen Rumpf und ggf. einen

Tabelle 6.4: Beste F1-Werte für die Varianten des Methodendurchschnitts

Variante	Bestes F1	Ausb.	Präz.	D-SW	D. Präz.
`SIG+KL`	31,9%	30,7%	33,2%	0,69	23,5%
Mit Methodenkomm. (`SIG+KL+KO`)	**33,4%**	30,4%	37%	0,74	**28,8%**
Mit Methodenrumpf	21,1%	45,6%	13,7%	0,69	17,7%
Mit Methodenkomm. und -rumpf	29,9%	27,1%	33,3%	0,77	24,1%
Alle Klassenelem. zweistufig	29,8%	28,7%	31%	0,78	27,2%

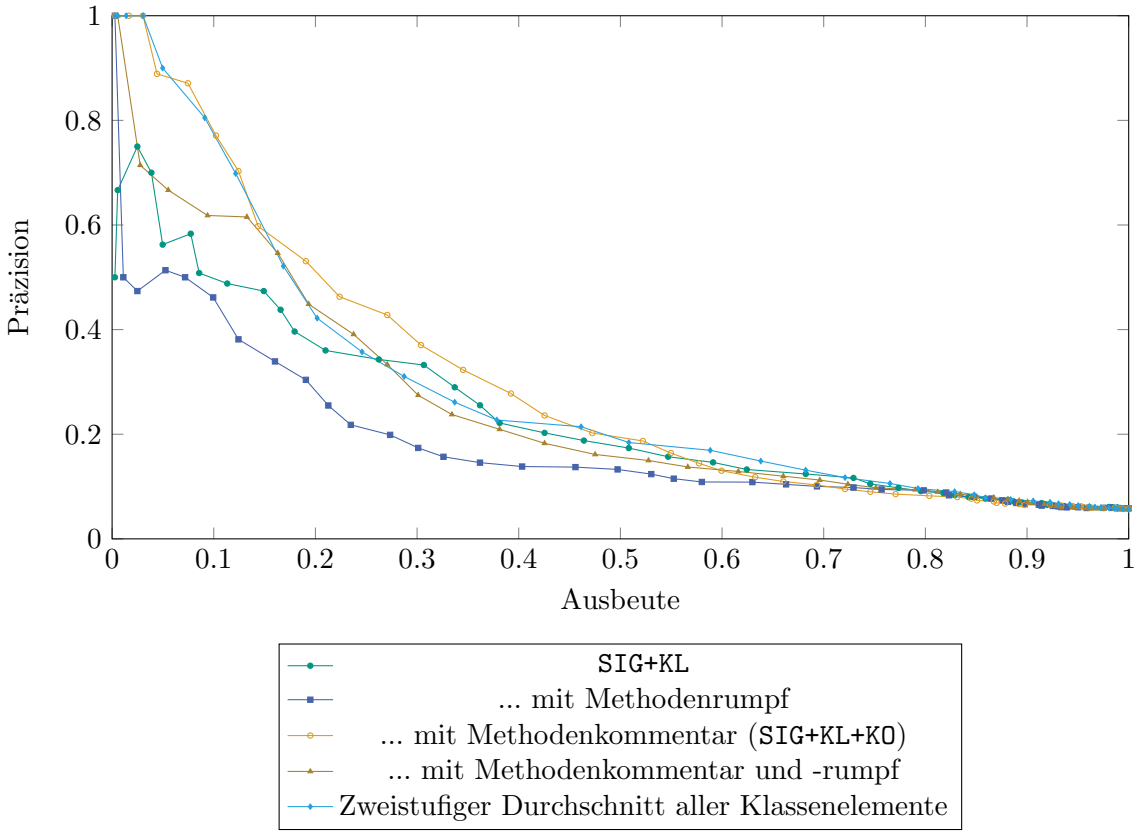


Abbildung 6.2: eTour: Varianten des Methodendurchschnitts

Kommentar. Diese können auch zum Methodendurchschnitt hinzugerechnet werden: Es wird ein Durchschnitt über die Signatur- und Klassennamenwörter plus Kommentarwörter bzw. Rumpfbezeichnerwörter gebildet. Für den Durchschnitt sind die Wörter ungeordnet. In Abbildung 6.2 und Tabelle 6.4 sind die Ergebnisse hiervon aufgezeigt. Zum Vergleich sind hier `SIG+KL` und die zweistufige Durchschnittsbildung, die bereits in Abbildung 6.1 zu sehen sind, ebenfalls eingezeichnet.
Dem Diagramm und den F1-Werten sind zu entnehmen, dass die Miteinbeziehung des Kommentars zum Methodendurchschnitt zu einer Verbesserung führt. Auch ist dadurch die durchschnittliche Präzision gestiegen. Wie in der Analyse angenommen ist der Kommentar hilfreich für die Rückverfolgbarkeit, da er sowohl von der Form als auch vom Abstraktionsniveau ähnlicher zu den Anforderungen ist als Bezeichner. Außerdem wird durch diese Ergebnisse bestätigt, dass nicht alle Klassenelemente gleich hilfreich für die Rückverfolgbarkeit sind. Anstatt alle Klassenelemente zu berücksichtigen, kann eine richtige Auswahl an Klassenelementen bessere Ergebnisse liefern.
Die Inklusion der Bezeichner im Methodenrumpf führt zum Beispiel zu einer Verschlechterung. Dies schlägt sich sowohl im F1-Wert als auch in der Ausbeute-Präzisionskurve und in der durchschnittlichen Präzision nieder.

In der Analyse wurde das Risiko besprochen, dass der Methodenrumpf implementierungsnäher als die Signatur oder der Kommentar ist. Dadurch kann es also die Rückverfolgbarkeit verrauschen, was in diesem Fall eingetreten ist.
Die Kombination von Methodenkommentar und -rumpf ist schlechter als die Methodenkommentarvariante. Wie bereits bei der Methodenrumpfvariante beobachtet, ist der Rumpf nicht hilfreich für die Rückverfolgbarkeit. Durch die Methodenkommentare konnte die Inklusion des Methodenrumpfes „ausgeglichen“ werden, sodass die durchschnittliche Präzision und der F1-Wert wieder näher den ursprünglichen Werten von `SIG+KL` liegen.
Die Methodenkommentarvariante ist bisher das beste Einbettungsverfahren und wird nun als `SIG+KL+KO` referenziert.
Auch bei iTrust ist `SIG+KL+KO` die bisher beste Variante (siehe Abbildung A.2).

Der Methodenkommentar wurde oben zum Durchschnitt der Methode hinzugerechnet. Da der Kommentar, wie in der Analyse erklärt, potentiell mehr hilfreiche Semantik tragen kann als die Signatur, kann die separate Inklusion des Methodenkommentars zu einer Verbesserung führen. Dadurch erhalten die Kommentare mehr Gewicht. Beim Vergleich zwischen `SIG` und `SIG+KL` hat dies eine Verbesserung bewirkt. Pro Methode wird also ein Durchschnitt über die Signatur wie in `SIG+KL` gebildet. Außerdem wird pro Kommentar ein Durchschnitt über alle Wörter berechnet. Anschließend wird für die Klasseneinbettung

Tabelle 6.5: Beste F1-Werte für den Vergleich der separaten Inklusion des Methodenkommentars

Variante	Bestes F1	Ausbeute	Präz.	D-SW	D. Präz
`SIG+KL`	31,9%	30,7%	33,2%	0,69	23,5%
`SIG+KL+KO`	**33,4%**	30,4%	37%	0,74	**28,8%**
Methodenkommentar separat	33,3%	40,0%	28,6%	0,73	26,1%

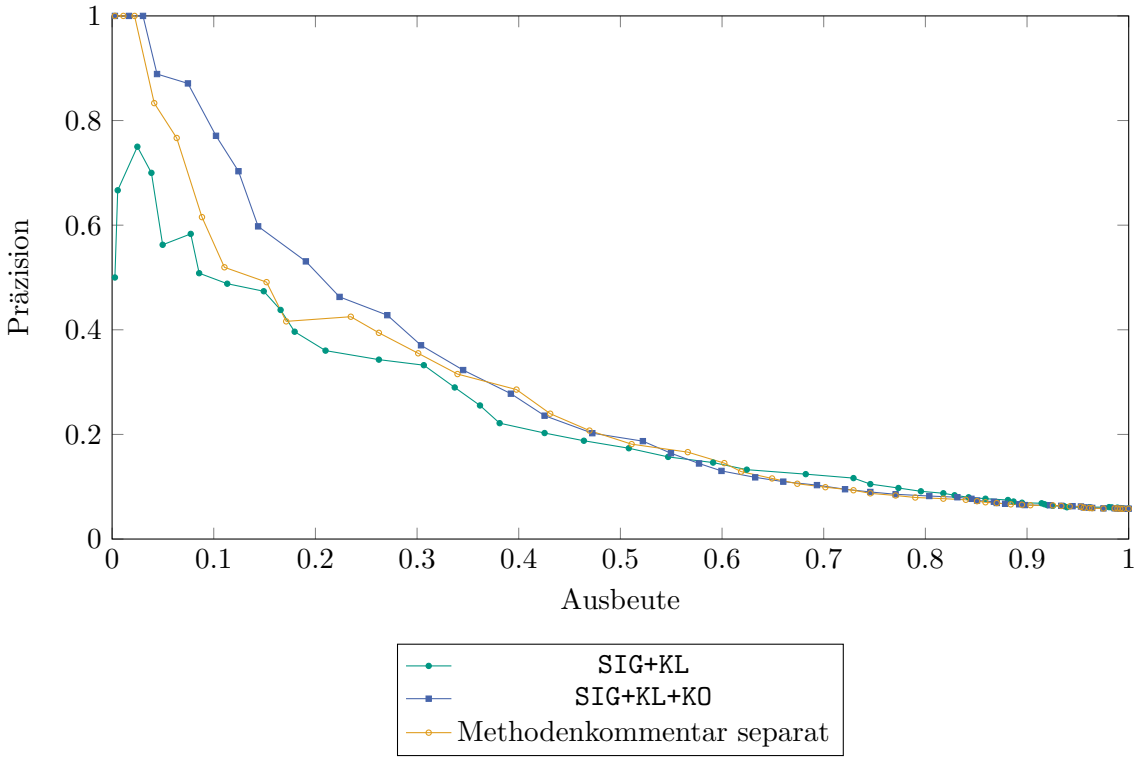


Abbildung 6.3: eTour: Ausbeute-Präzisionskurven über zwei Varianten, den Methodenkommentar miteinzubeziehen

nochmal der Durchschnitt zwischen allen Methoden- und Kommentardurchschnitte kalkuliert. Die Ergebnisse dieses Vorgehens sind in Abbildung 6.3 und Tabelle 6.5 eingetragen. Die separate Behandlung der Kommentare zeigt keine Verbesserung gegenüber der Variante, in der der Kommentar zum Methodendurchschnitt hinzugerechnet wird. Der beste F1-Wert der separaten Vorgehensweise ist um 0,1% kleiner als bei der anderen Variante. Dieser Unterschied ist jedoch zu klein, um definitive Aussagen treffen zu können. Für kleine Schwellwerte (das heißt große Ausbeuten) sind die beiden Ausbeute-Präzisionskurven fast gleich, für größere Schwellwerte ist die separate Behandlung schlechter. Das zeigt sich auch in der durchschnittlichen Präzision, die bei `SIG+KL+KO` um zwei Prozent höher liegt. Für iTrust (siehe Abbildung A.3) ist die separate Behandlung bezüglich des besten F1-Wertes und der durchschnittlichen Präzision auch schlechter. Jedoch beträgt die Verschlechterung hier weniger als ein Prozent. Aus den vorliegenden Ergebnissen kann also nur gefolgert werden, dass die beiden Inklusionsvarianten des Methodenkommentars sehr ähnlich sind. Für die weitere Evaluation wird es keinen großen Unterschied machen, mit welcher Kommentarvariante fortgefahren wird.

Im Folgenden werden zusätzlich Attribute, Superklassifizierer und Klassenkommentare miteinbezogen. Aus diesen Elementen wird jeweils eine separate Einbettung aus dem Durchschnitt ihrer Wörter berechnet, die danach jeweils mit den Methodeneinbettungen (bestehend aus Methodensignatur und Klassenname) und Klassennameneinbettungen mit einem weiteren Durchschnitt aggregiert werden. Bei den Superklassifizierern wurde nur der Klassenname des Superklassifizierers berücksichtigt. Die Evaluationsergebnisse sind in Abbildung 6.4 und Tabelle 6.6 dargestellt. Zum Vergleich wurden hier ebenfalls `SIG+KL` und `SIG+KL+KO` eingezeichnet. Aus der Abbildung lässt sich herauslesen, dass sich die Kurve von `SIG+KL+KO` bei größere Schwellwerten von den anderen Kurven distanziert. Der beste F1-Wert und die höchste durchschnittliche Präzision sind ebenfalls bei `SIG+KL+KO` zu finden. Die Klassenkommentarvariante besitzt den zweitbesten F1-Wert, der um ein Prozent schlechter ist als bei `SIG+KL+KO`. Gegenüber der `SIG+KL` lässt sich jedoch durch den Klassenkommentar bezüglich des besten F1-Wertes und der durchschnittlichen Präzision und eine kleine Verbesserung erzielen. Auch bei iTrust (siehe Abbildung A.4) ist durch den Klassenkommentar eine Verbesserung des besten F1-Wertes und der durchschnittlichen Präzision zu beobachten. Aus diesen Gründen sind Klassenkommentare in diesem Fall als hilfreich einzustufen.
Das Miteinbeziehen von Superklassifizierern verschlechtert den besten F1-Wert im Vergleich zu `SIG+KL` um ein Prozent und die durchschnittlichen Präzision um 0,2 Prozent. Dies spricht auf den ersten Blick für eine negative Wirkung durch die Superklassifizierer. Bei Betrachtung des Diagramms liegen ihre beiden Kurven aber relativ nah beieinander. Manchmal liegt die Kurve von `SIG+KL`, manchmal die vom Superklassifizierer höher. Der bessere F1-Wert von `SIG+KL` bei eTour könnte also ein Ausreißer sein, vor allem da sich die durchschnittliche Präzision fast nicht ändert. Des Weiteren erzielt die Evaluation mit iTrust eine Steigerung des besten F1-Wertes um 0,4% und eine Erhöhung der durchschnittlichen Präzision um 0,7% (siehe Abbildung A.4). Die Wertänderungen sind insgesamt zu gering und zu uneindeutig, um allgemein von einer Verbesserung oder Verschlechterung zu sprechen. Es lässt sich nur feststellen, dass die Superklassifizierer nur eine kleine Änderung bewirken. Dies kann mehrere Ursachen haben: Es kann daran liegen, dass die Superklassifiziereinbettung im Verhältnis zu den Methodensignatureinbettungen beim Durchschnitt zu wenig Gewicht hat, wenn eine Klasse zum Beispiel sehr viele Methoden besitzt.
Jedoch müsste dies auch bei Klassennameneinbettungen zutreffen - wie in Abbildung 6.1 zu sehen, haben diese trotzdem eine größere Änderung bewirkt. Ein anderer Faktor ist, dass nicht jeder Klassifizierer auch einen Superklassifizierer besitzt - bei Klassen ohne Superklassifizierer führt es also zu keiner Änderung. Schließlich kann der Grund auch darin bestehen, dass die Superklassifizierer nur wenig neue semantische Informationen beherbergen können, wenn beispielsweise Klassenname und Superklassifizierername fast gleich

Tabelle 6.6: Beste F1-Werte für die Varianten mit Superklassifizierer, Klassenkommentare und Attribute

Variante	Bestes F1	Ausbeute	Präz.	D-SW	D. Präz
`SIG+KL`	31,9%	30,7%	33,2%	0,69	23,5%
`SIG+KL+KO`	**33,4%**	30,4%	37%	0,74	**28,8%**
Mit Klassenkommentar	32,3%	38,1%	28,0%	0,7	23,7%
Mit Superklassifizierer	30,9%	35,4%	27,5%	0,68	23,3%
Mit Attribut	24,8%	38,1%	18,3%	0,69	15,8%
Mit Attribut und Attributkommentar	24,9%	39,2%	18,2%	0,69	16,3%

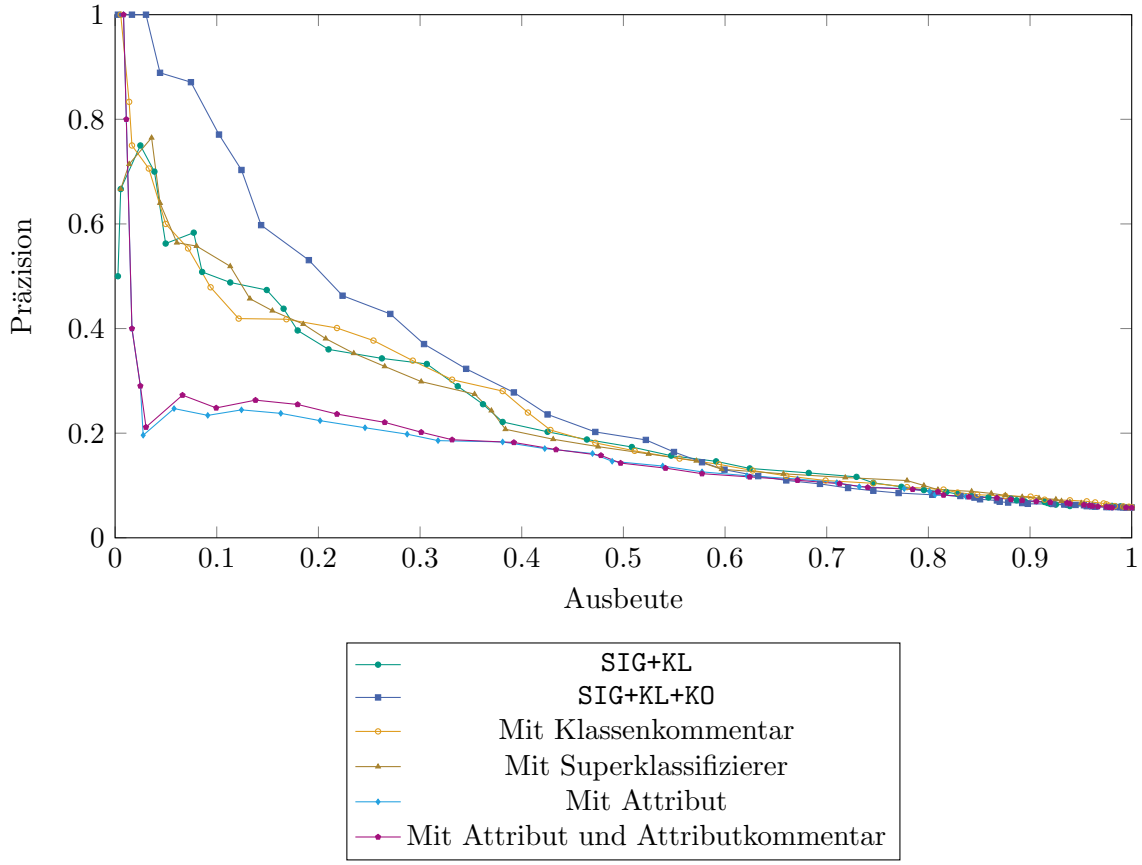


Abbildung 6.4: eTour: Ausbeute-Präzisionskurven der Varianten mit Superklassifizierer, Attribute und Klassenkommentare

sind.
Superklassifizierer können das Ergebnis je nach Projekt verbessern oder verschlechtern - insgesamt ist der Einfluss auf das Ergebnis jedoch sehr klein.

Als Letztes sind in Abbildung 6.4 und Tabelle 6.6 die Daten für Attribute zu sehen. Es wurde eine Attributeinbettung aus dem Typ, dem Namen und der Initialisierung via Durchschnittsbildung berechnet. Das wurde einmal mit und einmal ohne Attributkommentare durchgeführt. Diese Klasseneinbettungen sind vor allem bei den höheren Schwellwerten deutlich schlechter als die anderen Varianten. Wie in der Analyse schon angeführt, besteht bei Attributen das Risiko, dass sie zu implementierungsnah sind, da sie sich zum Beispiel oft aus primitiven Typen oder Datenstrukturen zusammensetzen und daher weniger hilfreich für die Rückverfolgbarkeit sind. Dieses Evaluationsergebnis unterstützt diese Vermutung. Eine Auswertung mit iTrust führt zum gleichen Ergebnis (siehe Abbildung A.4).

Bis jetzt haben sich Methoden- und Klassenkommentare als hilfreiche Erweiterung der simplen SIG+KL-Variante herausgestellt. Diese beiden Kommentararten können auch kombiniert werden. Die Methodenkommentare werden zum Methodendurchschnitt miteinberechnet. Für Klassenkommentare wird eine separate Einbettung aus dem Durchschnitt ihrer Wörter kalkuliert. Die Klasseneinbettung ist schließlich der Durchschnitt über die Klassennameneinbettung, die Klassenkommentareinbettung und die Methodendurchschnitte.

Die Auswertung hiervon ist in Tabelle 6.7 bzw. Abbildung 6.5 zu finden. Durch Kombination beider Kommentararten (ab jetzt als SIG+KL+KO+KKO referenziert) konnte hier nochmals eine kleine Verbesserung im F1-Wert auf 34,1% bewirkt werden. Die Ausbeute-Präzisionskurve verläuft sehr ähnlich zur Kurve von SIG+KL+KO. Bei mittleren Schwellwerten ist SIG+KL+KO+KKO ein wenig besser, bei höheren ist SIG+KL+KO überlegen. Die durchschnittlichen Präzision ist bei SIG+KL+KO+KKO um 0,6% schlechter. Im Falle von iTrust lässt sich mit SIG+KL+KO+KKO sowohl im F1-Wert, als auch in der durchschnittlichen Präzision eine Verbesserung gegenüber SIG+KL+KO erreichen (siehe Abbildung A.5).

Von den betrachteten Kombinationen für die Klasseneinbettung in diesem Abschnitt erzielt SIG+KL+KO+KKO mit leichter Tendenz die besten Ergebnisse, SIG+KL+KO ist jedoch ähnlich gut. Superklassifizierer ändern das Ergebnis nur gering. Attribute und Methodenrümpfe

Tabelle 6.7: Beste F1-Werte für die Kombinationen der Kommentare

Variante	Bestes F1	Ausbeute	Präz.	D-SW	D. Präz.
SIG+KL	31,9%	30,7%	33,2%	0,69	23,5%
SIG+KL+KO	33,4%	30,4%	37%	0,74	**28,8%**
Mit Klassenkommentar	32,3%	38,1%	28,0%	0,7	23,7%
SIG+KL+KO+KKO	**34,1%**	40,9%	39,4%	0,74	28,2%

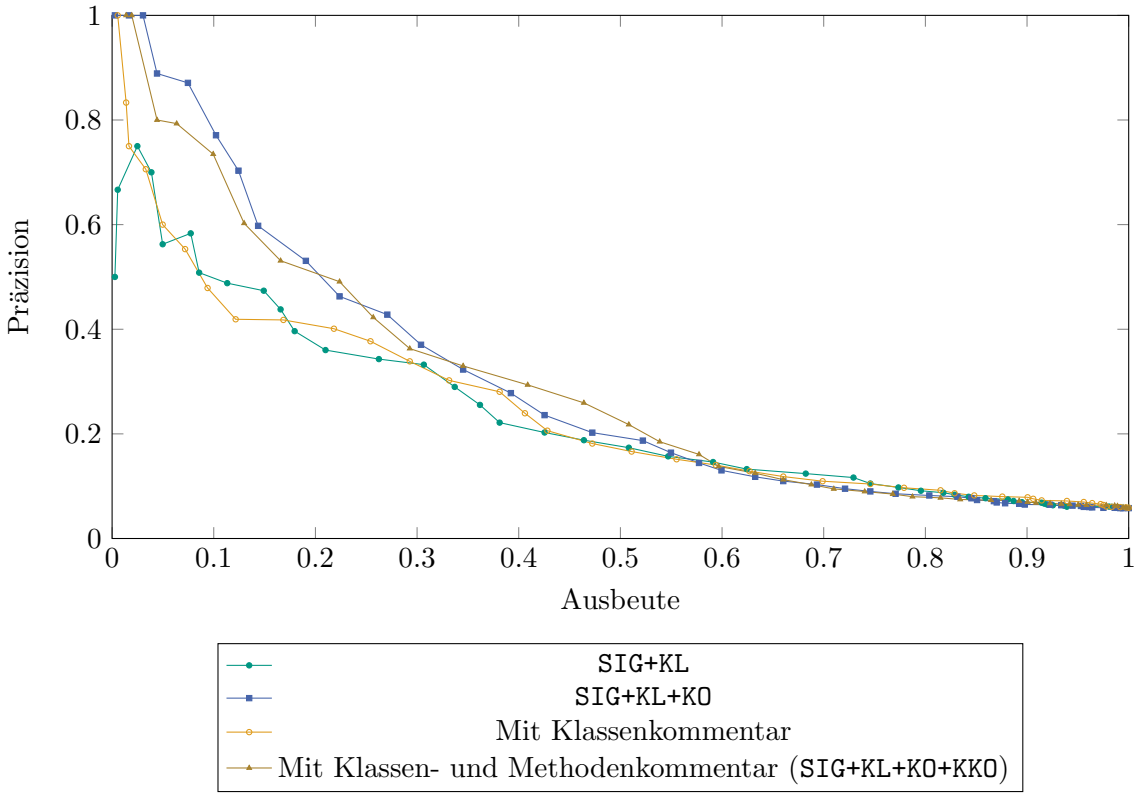


Abbildung 6.5: eTour: Vergleich der Kombination von Klassen- und Methodenkommentaren zu den bisherigen Varianten

enthalten tendenziell zu viel Implementierungsdetail und führen zu keiner Verbesserung. Außer den bisher vorgestellten wurden noch weitere Kombinationen untersucht (zum Beispiel Methodenrumpf separat von der Methodensignatur miteinbeziehen). Diese erzielen aber keine besseren Ergebnisse und werden an dieser Stelle ausgelassen.

6.3.2.2 Zusammensetzung der Anforderungseinbettungen

Für die Anforderungsseite wurden die Einbettungen als flacher Durchschnitt ihrer Worteinbettungen berechnet. Dies wurde im vorherigen Abschnitt bereits durchgeführt. Bei den Klasseneinbettungen hat sich ergeben, dass die zweistufige Aggregation besser ist als die flache. Deshalb wurde auch für die Anforderungen überprüft, ob so eine Aggregation hilft. Für jeden Satz der Anforderung wird also ein Durchschnitt ihrer Wörter berechnet. Die Anforderungseinbettung wird danach als Durchschnitt der Satzdurchschnitte definiert. Für die zweistufigen Anforderungseinbettungen wurde die Rückverfolgbarkeit mit `SIG+KL` und `SIG+KL+KO+KKO` durchgeführt und mit den Ergebnissen aus den vorigen Abschnitten verglichen, die mit dem flachen Durchschnitt der Anforderungswörter ausgeführt wurden. Die Auswertung findet sich in Abbildung 6.6 und Tabelle 6.8.

Der einfache Durchschnitt ist dabei der zweistufigen Aggregation überlegen. Die F1-Werte

Tabelle 6.8: Beste F1-Werte für die Varianten der Anforderungseinbettung

Variante	Bestes F1	Ausbeute	Präz.	D-SW	D. Präz.
`SIG+KL`	31,9%	30,7%	33,2%	0,69	23,5%
`SIG+KL+KO+KKO`	**34,1%**	40,9%	39,4%	0,74	**28,2%**
`SIG+KL`. (2x Aggr.)	27,1%	33,4%	22,8%	0,65	21,9%
`SIG+KL+KO+KKO` (2x Aggr.)	30,4%	34,3%	27,3%	0,73	25,5%

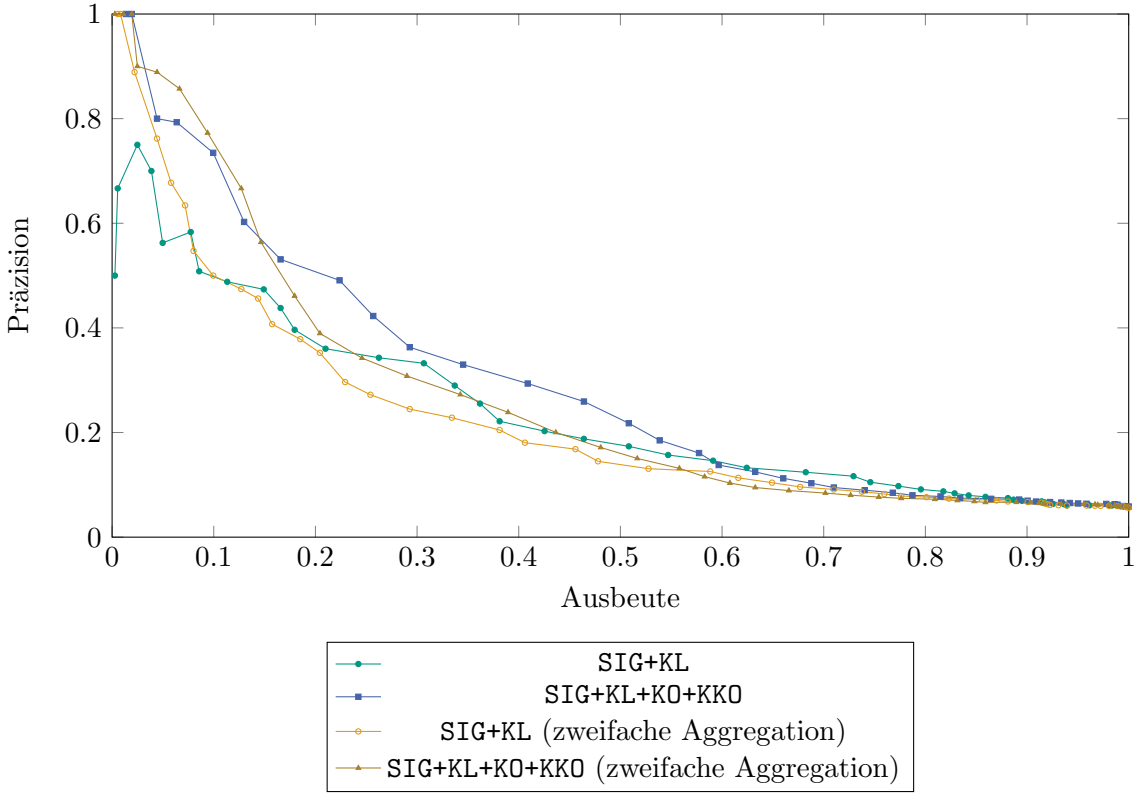


Abbildung 6.6: eTour: Vergleich der Anforderungseinbettungen mit einfacher und zweifacher Aggregation

sind bei der zweistufigen Aggregation schlechter. Sowohl bei `SIG+KL+KO+KKO` als auch bei `SIG+KL` sinkt der F1-Wert um ca. vier Prozent. Auch die durchschnittliche Präzision nimmt um mehrere Prozent ab. Dies ist auch an den Ausbeute-Präzisionskurven zu beobachten: Sie zeigen größere Bereiche, in der die einstufige Aggregation besser abschneidet.

Die zweistufige Aggregation beherbergt ein erhöhtes Risiko, dass der resultierende Vektor abseits von den ursprünglichen Vektoren liegen kann, was schließlich zu verzerrten Repräsentationen und schlechterer Rückverfolgbarkeit führen kann. Aus einer Evaluation mit iTrust (siehe Abbildung A.6) lassen sich die gleichen Schlussfolgerungen ziehen.

Für die Anforderungsseite wird im Folgenden die einstufige Aggregation bevorzugt.

6.3.2.3 Nutzung eines softwarespezifischen Modells

In den vorigen Abschnitten wurden die Kombinationsmöglichkeiten mit fastText durchgeführt. Es wurde das vortrainierte Standardmodell der fastText-Autoren verwendet. In diesem Abschnitt wird ein mit Anforderungsdaten trainiertes fastText-Modell evaluiert und mit dem Standardmodell verglichen.

Dieses Anforderungsmodell wird in Abschnitt 5.1 genauer beschrieben.

Tabelle 6.9: Vergleich der besten F1-Werte für Klasseneinbettungen mit dem Standard- und Anforderungsmodell

Variante	Bestes F1	Ausbeute	Präzision	D-SW	D. Präz.
`SIG+KL`	31,9%	30,7%	33,2%	0,69	23,5%
`SIG+KL+KO+KKO`	**34,1%**	40,9%	39,4%	0,74	**28,2%**
(A) `SIG+KL`	28,9%	35,1%	24,6%	0,83	20,8%
(A) `SIG+KL+KO+KKO`	30,1%	31,2%	29,1%	0,88	24,1%

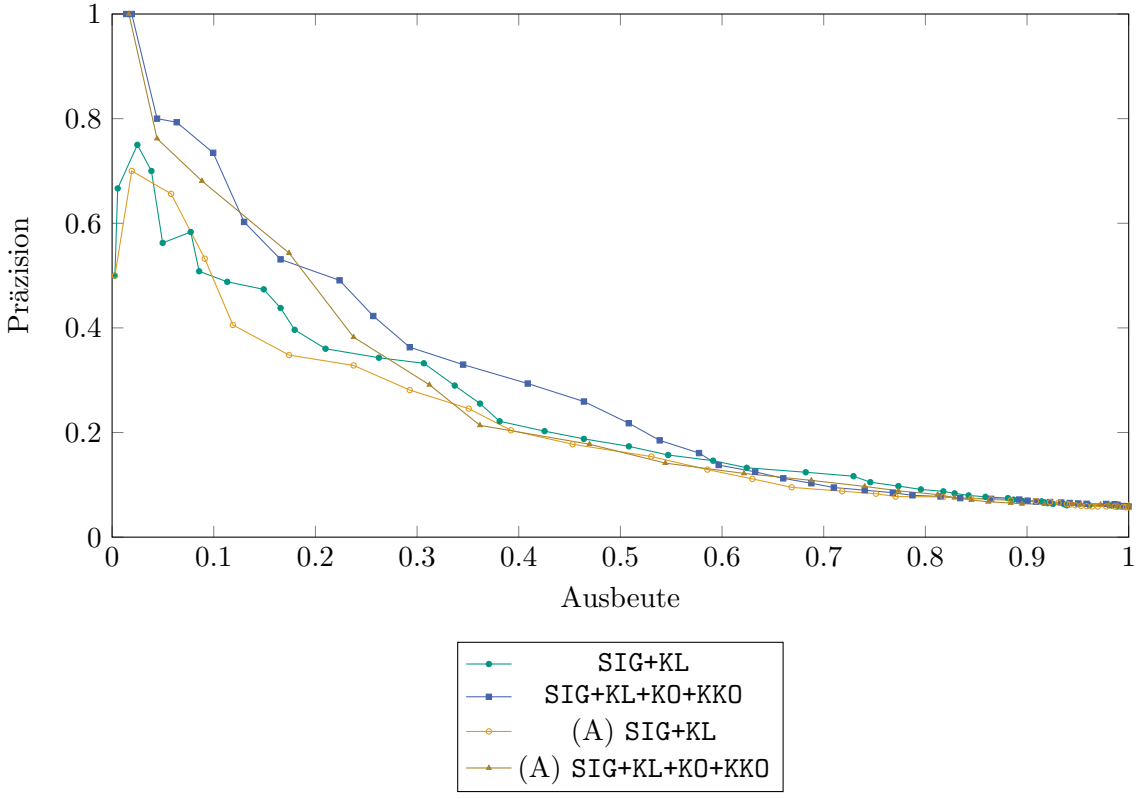


Abbildung 6.7: eTour: Vergleich der Rückverfolgbarkeit mit dem Standard- und Anforderungsmodell für fastText

Zur Evaluation des angepassten Modells wurde die Rückverfolgbarkeit mit Klasseneinbettungen wie in Abschnitt 6.3.2.1 durchgeführt. Für die Anforderungsseite wurde pro Anforderung ein Durchschnittsvektor aus den einzelnen Worteinbettungen generiert. In Abbildung 6.7 und Tabelle 6.9 sind die Ergebnisse für zwei Zusammensetzungen der Klasseneinbettungen zu sehen. Ein (A) kennzeichnet, dass das angepasste Anforderungsmodell verwendet wurde. Ist keine gesonderte Kennzeichnung vorhanden, wurde das Standardmodell wie in Abschnitt 6.3.2.1 eingesetzt. In beiden Fällen sind sowohl die durchschnittliche Präzisionen als auch die besten F1-Werte schlechter mit dem Anforderungsmodell. Auch grafisch liegen die Kurven tiefer als die des Standardmodells. Gleiches wird bei iTrust beobachtet (siehe Abbildung A.7). In der Analyse wurde das Risiko geäußert, dass ein Anforderungsmodell zu spezifisch bezüglich der verwendeten Trainingsprojekte werden kann. Dies kann dazu führen, dass das angepasste Modell schlechter funktioniert als ein generisches Modell.

Für die weitere Evaluation wird daher auf angepasste Modelle verzichtet und auf allgemeine Standardmodelle zurückgegriffen.

6.3.3 Abbildung auf der Elementebene

Neben einer Abbildung mit Hilfe von Klassen- und Anforderungseinbettungen wurde im Entwurf auch die Abbildung auf der Elementebene vorgestellt. Dazu gehört der Mehrheitsentscheid, der in Abschnitt 6.3.3.1 evaluiert wird. Die Erweiterung mit Aufrufabhängigkeiten wird danach in Abschnitt 6.3.3.2 betrachtet.

6.3.3.1 Mehrheitsentscheid

Eine Klasseneinbettung kann aus verschiedenen Klassenelementen (Methoden, Attribute, Kommentare) zusammengesetzt werden - dies wurde im vorherigen Abschnitt evaluiert. Der Mehrheitsentscheid bildet im Gegensatz dazu keine einzige Klasseneinbettung pro Klasse, sondern verbindet die Klassenelemente separat mit Anforderungen, um daraus Rückverfolgbarkeitsverbindungen abzuleiten. Der Ablauf des Mehrheitsentscheids verläuft wie im Entwurf Abschnitt 4.4.4 erklärt und in Abbildung 4.4 dargestellt. Für die Stimmenbestimmung wurde ein Mehrheitsentscheidschwellwert verwendet. Die Quelltextähnlichkeitsreduktion benutzt das Maximum der Ähnlichkeiten als Funktion. Beim eigentlichen Mehrheitsentscheid in Schritt drei werden die Anforderung(en) mit der maximalen Stimmanzahl als Verbindung festgelegt.

Zur Evaluierung des Mehrheitsentscheids werden die gleichen Kombinationen von Klassenelementen wie bei den Klasseneinbettungen verwendet, um den Mehrheitsentscheid durchzuführen. Auf diese Weise kann verglichen werden, ob der Mehrheitsentscheid eine Verbesserung erzielt. Da die zweistufige Aggregation bei den Anforderungseinbettungen in Abschnitt 6.3.2.2 keine Verbesserung erzielt hat, wird hier die Anforderung ausschließlich aus einem flachen Durchschnitt der Worteinbettungen berechnet. Wie im vorigen Abschnitt wird fastText mit dem vortrainierten Standardmodell verwendet und es wird die Kosinusähnlichkeit als Metrik benutzt.

Für den Rest des Abschnitts werden Kombinationen, die mit dem Mehrheitsentscheid durchgeführt werden, mit einem „(M)“ gekennzeichnet. Kombinationen ohne Kennzeichnung benutzen Klasseneinbettungen, wie sie in Abschnitt 6.3.2.1 berechnet wurden. Zur Erzeugung der Ausbeute-Präzisionskurven wurde der Schwellwert in 0,01er Intervallen variiert, um Ausbeute-Präzisionswertepaare zu erhalten. Bei mehreren Schwellwerten wie hier beim Mehrheitsentscheid wird der letzte Schwellwert, also der Dateilevelschwellwert, variiert, während die anderen Schwellwerte konstant gehalten werden. Es werden also für verschiedene Mehrheitsentscheidschwellwerte unterschiedliche Kurven erzeugt. Davon wird

Tabelle 6.10: Beste F1-Werte für Methodensignatur- und Methodenkommentarvarianten

Variante	Bestes F1	Ausbeute	Präz.	D-SW	M-SW	D. Präz.
`SIG+KL`	31,9%	30,7%	33,2%	0,69	-	23,5%
`SIG+KL+KO`	33,4%	30,4%	37%	0,74	-	**28,8%**
(M) `SIG+KL`	**35,9%**	30,1%	44,4%	0.69	0,49	22,9%
(M) `SIG+KL+KO`	34,4%	32,6%	35,4%	0,76	0.45	25,8%

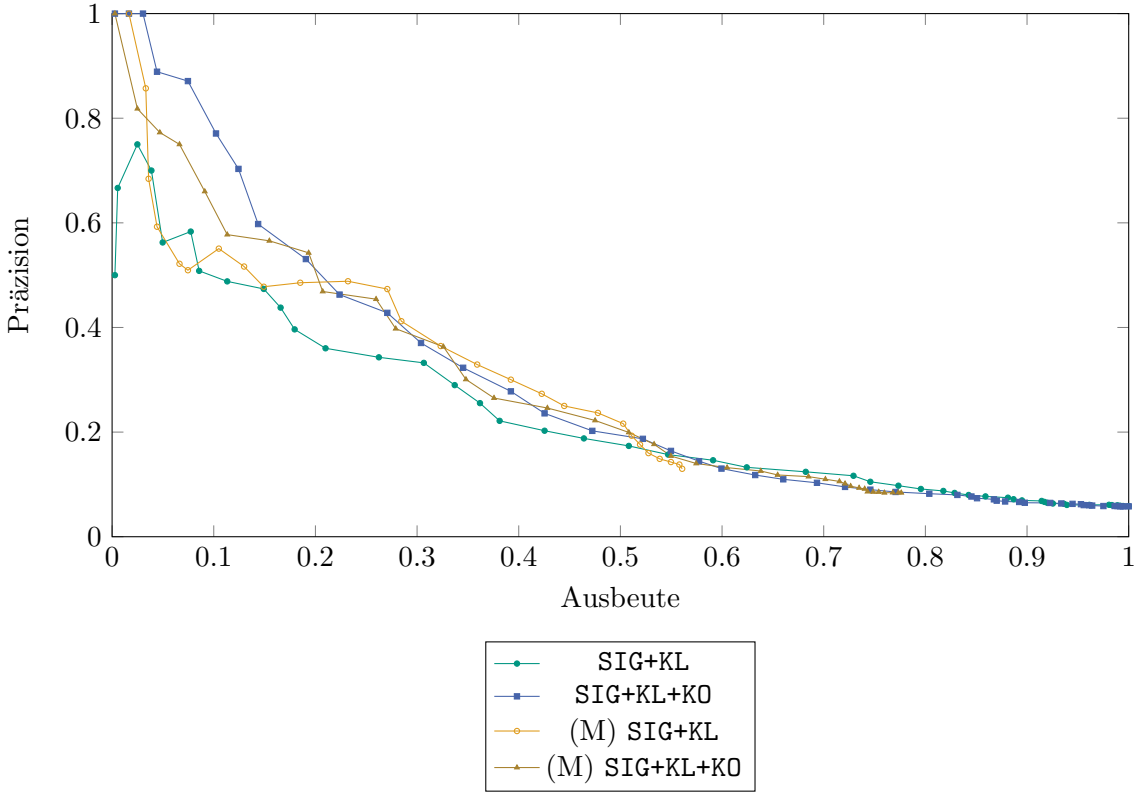


Abbildung 6.8: eTour: Vergleich zwischen Mehrheitsentscheid und Klasseneinbettung bezüglich der Methodensignatur- und Methodenkommentarvarianten

im Folgenden diejenige mit dem besten F1-Wert im Diagramm dargestellt. Aus den dazugehörigen Tabellen lässt sich der verwendete Mehrheitsentscheidschwellwert herauslesen: „M-SW“ bezeichnet den Mehrheitsentscheidschwellwert und „D-SW“ ist der Dateilevelschwellwert.

In Tabelle 6.10 und Abbildung 6.8 werden `SIG+KL` und die Variante mit Methodenkommentaren, `SIG+KL+KO`, verglichen. Im Diagramm ist zu sehen, dass die Ausbeute-Präzisionskurven des Mehrheitsentscheids teilweise über denen der Klasseneinbettungen liegen. Bei `SIG+KL` ist die Kurve des Mehrheitsentscheid über größere Bereiche deutlich besser als die Klasseneinbettungsvariante. (M) `SIG+KL+KO` verbessert sich gegenüber `SIG+KL+KO` weniger stark in den Ausbeute-Präzisionswertepaaren und ist bei höheren Schwellwerten sogar schlechter. Die durchschnittlichen Präzisionen der Mehrheitsentscheidvarianten sind in beiden Fällen niedriger. Bei genauerer Betrachtung ist zu sehen, dass ihre Kurven bei circa 58% und 78% und Ausbeute enden. Das liegt spricht dafür, dass der Mehrheitsentscheid zu restriktiv ist, da nie alle korrekten Verbindungen gefunden werden. Dies kann kann mehrere Ursachen haben. Es kann daran liegen, dass die das Zulassen der Anforderung(en) mit der maximalen Stimmanzahl zu wenige Verbindungen ermöglicht. Für die vorliegende Auswertung des Mehrheitsentscheids wurde auch ein Mehrheitsentscheidschwellwert von

0,49 bzw. 0,45 verwendet. Wenn korrekte Stimmen eine zu niedrige Ähnlichkeit besitzen, werden sie durch diese Schwellwerte entfernt. Der Mehrheitsentscheid funktioniert hier also noch nicht zuverlässig genug.

Die durchschnittlichen Präzisionen zwischen den Klasseneinbettungen und dem Mehrheitsentscheid können jedoch nicht direkt miteinander verglichen werden. Beim Ersteren wurde zur Berechnung der durchschnittlichen Präzision über alle Schwellwerte variiert (es gibt nur einen Schwellwert), während beim Letzteren nur der Dateilevelschwellwert verwendet wurde, um Ausbeute-Präzisionswertepaare zu erzeugen. Sie können aber dafür verwendet werden, zwischen Mehrheitsentscheidverfahren bzw. zwischen Klasseneinbettungsverfahren zu vergleichen.

Bezüglich der besten F1-Werte sind die Mehrheitsentscheidvarianten höher als die Varianten mit Klasseneinbettungen. Auffällig ist, dass beim Mehrheitsentscheid (M) `SIG+KL` einen höheren F1-Wert erzielt als mit zusätzlicher Berücksichtigung der Methodenkommentare. (M) `SIG+KL+KO` hat dafür bei höheren Schwellwerten bessere Ausbeute-Präzisionswerte und

Tabelle 6.11: Beste F1-Werte für den Vergleich des Mehrheitsentscheids bezüglich verschiedener Kommentarvarianten

Variante	Bestes F1	Ausbeute	Präz.	D-SW	M-SW	D. Präz.
`SIG+KL+KO+KKO`	34,1%	40,9%	39,4%	0,74	-	**28,2%**
Mit Klassenkommentar	32,3%	38,1%	28,0%	0,7	-	23,7%
(M) `SIG+KL+KO+KKO`	31,7%	26,2%	40,1%	0.78	0,49	25,1%
(M) Mit Klassenkomm.	31,7%	38,7%	26,8%	0,71	0.52	19,3%
(M) `SIG+KL+KO`	**34,4%**	32,6%	36,3%	0,76	0.45	25,8%

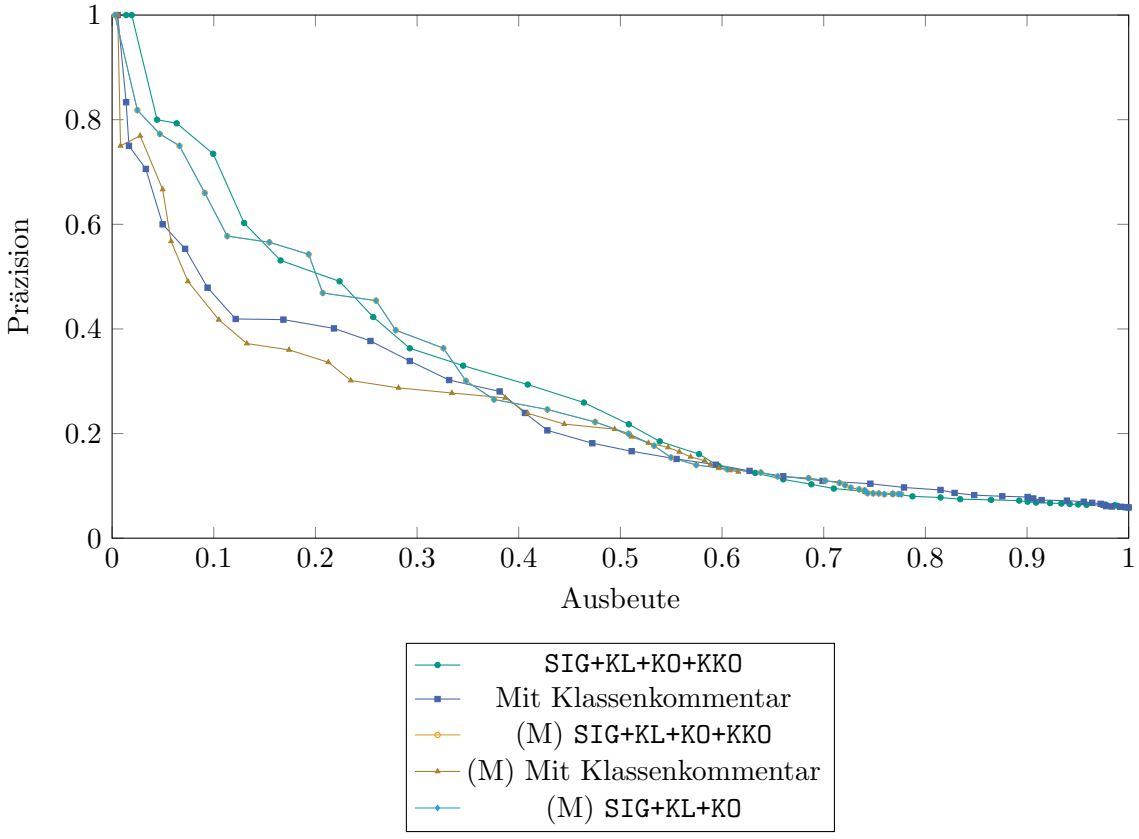


Abbildung 6.9: eTour: Vergleich zwischen Mehrheitsentscheid und Klasseneinbettung bezüglich verschiedener Kommentarvarianten

auch eine höhere durchschnittliche Präzision. Der beste F1-Wert von (M) `SIG+KL` könnte also ein Ausreißer sein. Für iTrust ist die korrespondierende Evaluation in Abbildung A.8 zu sehen. Dort sind ebenfalls die Mehrheitsentscheidvarianten besser, aber in diesem Fall erzielt (M) `SIG+KL+KO` einen deutlich höheren F1-Wert als (M) `SIG+KL`.

Bei den obigen beiden Zusammensetzungen ist also insgesamt der Mehrheitsentscheid, was die besten F1-Werte und die Betrachtung der Kurvenverläufe betrifft, leicht besser als die Klasseneinbettungen. Jedoch wurden auch die Mängel bezüglich der Ausbeute, die nie 100% erreicht, angesprochen.

In Tabelle 6.11 und Abbildung 6.9 sind zwei Kombinationen zu sehen, die im Mehrheitsentscheid schlechter abschneiden als bei der Klasseneinbettung. Beide Kombinationen beinhalten den Klassenkommentar. Eine Evaluation mit iTrust (siehe Abbildung A.9) zeigt, dass sich die Klassenkommentarvariante im Mehrheitsentscheid auch verschlechtert und (M) `SIG+KL+KO+KKO` sich nicht verändert. Das schlechtere Abschneiden ist überraschend, da die Klassenkommentare bei den Klasseneinbettungen hilfreich für die Rückverfolgbar-

Tabelle 6.12: Beste F1-Werte für den Vergleich mit dem Mehrheitsentscheid bezüglich der Varianten mit Methodenrumpf und Superklassifizierer

Variante	Bestes F1	Ausbeute	Präz.	D-SW	M-SW	D. Präz.
Mit Methodenrumpf	21,1%	45,6%	13,7%	0,69	-	17,7%
Mit Superklassifizierer	30,9%	35,4%	27,5%	0,68	-	23,3%
(M) Mit Methodenrumpf	24,9%	36,7%	18,9%	0.73	0,52	12,7%
(M) Mit Superklassifizierer	**36,2%**	30,1%	45,4%	0,71	0.49	24,7%
(M) `SIG+KL+KO`	34,4%	32,6%	36,3%	0,76	0.45	**25,8%**

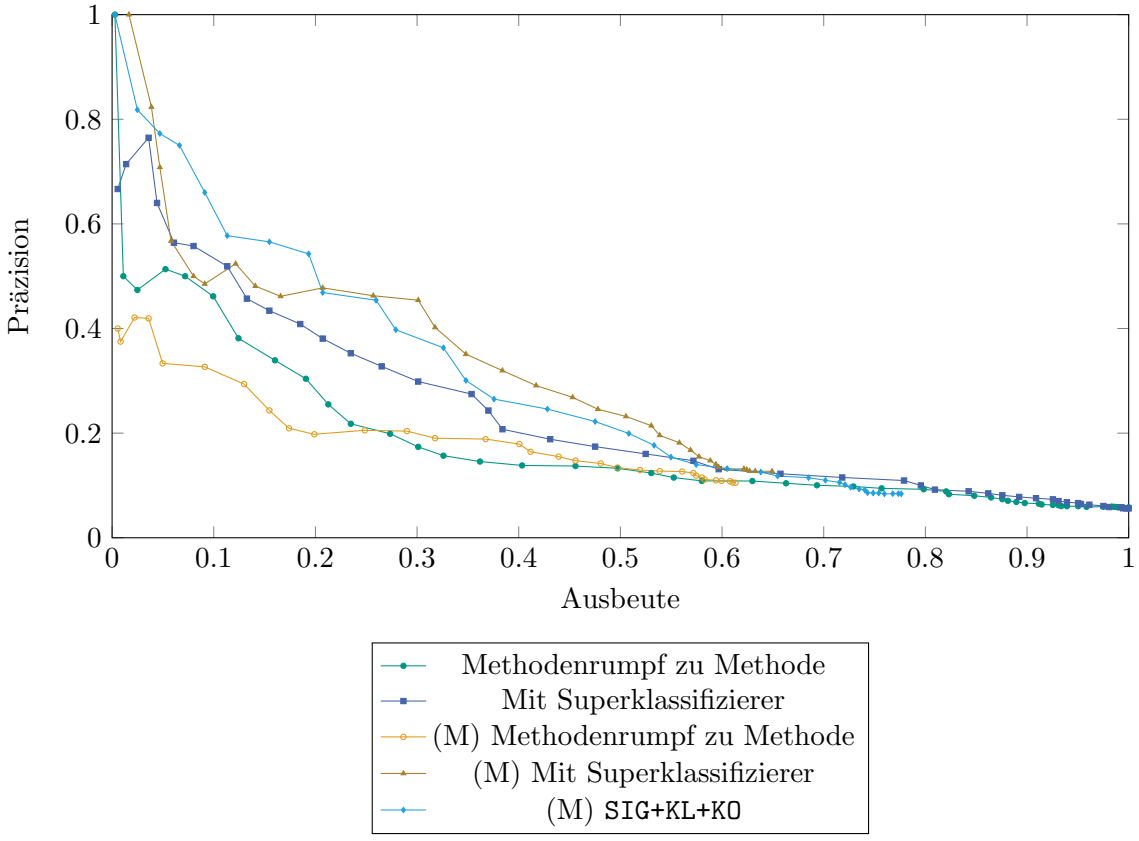


Abbildung 6.10: eTour: Vergleich zwischen Mehrheitsentscheid und Klasseneinbettung bezüglich der Varianten mit Methodenrumpf und Superklassifizierer

keit waren. Es muss jedoch nicht alleine am Klassenkommentar liegen, dass die Resultate niedriger sind. Der Mehrheitsentscheid kann auch selbst die Ursache sein. Wie bereits angesprochen, ist er potentiell zu restriktiv, wenn pro Klasse nur die Anforderung(en) mit der maximalen Stimmanzahl als Rückverfolgbarkeitsverbindungen zugelassen werden, wie es hier getan wurde. Es kann zum Beispiel vorkommen, dass die Methoden einer Klasse für drei richtige Anforderungen abstimmen und diese drei Anforderungen gleich viele Stimmen besitzen. Wenn nun der Klassenkommentar hinzukommt und für eine der drei richtigen Anforderungen abstimmt, werden die zwei anderen eigentlich korrekten Anforderungen überstimmt und ausgelassen, was zu einem schlechterem Ergebnis führt, als wenn man den Kommentar nicht hinzugenommen hätte. Bei den Klasseneinbettungen würden solche Informationen trotzdem durch den Durchschnitt miteinbezogen werden. In dieser Auswertung zeigt sich, dass der Mehrheitsentscheid nicht eindeutig besser als die Klasseneinbettungen ist.

In Abbildung 6.10 und Tabelle 6.12 sind zwei weitere Kombinationsmöglichkeiten eingetragen. Bei den Methodenrümpfen ist die Kurve des Mehrheitsentscheids bei höheren Schwellwerten niedriger, besitzt aber einen besseren F1-Wert. Die Kurve der Superklassifiziererkombination hebt sich beim Mehrheitsentscheid zwischendurch sichtbar von der Klasseneinbettungsvariante ab. Sein F1-Wert übertrifft den Wert der Klasseneinbettungsvariante um ca. sechs Prozent und den von (M) `SIG+KL+KO` um fast zwei Prozent. Die durchschnittliche Präzision ist dafür niedriger. Bei iTrust (siehe Abbildung A.10) ist der Methodenrumpf im Mehrheitsentscheid um 0,3% im F1-Wert schlechter. Dieser Unterschied ist zu klein, um definitive Aussagen zu treffen. Der Superklassifizierer ist jedoch auch besser im Mehrheitsentscheid, allerdings übertrifft er nicht den F1-Wert von (M) `SIG+KL+KO`. Der gute F1-Wert der Superklassifizierer bei eTour kann also ein Ausreißer sein.

In Anbetracht des besten F1-Wertes ist der Mehrheitsentscheid den Klasseneinbettungen überlegen. Jedoch wurden bei einigen Verfahren auch schlechtere Ergebnisse erzielt. Das alleinige Zulassen der Anforderungen, die die maximale Stimmanzahl besitzen, kann zu restriktiv sein, was eine Verschlechterung zur Folge haben kann. Eine Verbesserungsmöglichkeit könnte darin bestehen, zusätzlich die Anforderungen mit den zweitmeisten Stimmen (bzw. allgemeiner n-tmeisten Stimmen) zuzulassen. Auch ist es möglich, nicht die absolute Stimmanzahlen direkt zu betrachten, sondern alle Anforderungen nach ihrer Stimmanzahl absteigend zu sortieren und die ersten n Anforderungen als Verbindung festzulegen.

6.3.3.2 Mehrheitsentscheid mit Aufrufabhängigkeiten

Durch Aufrufabhängigkeiten können weitere Methoden gefunden werden, die eventuell zur gleichen oder einer ähnlichen Anforderung wie die aktuelle Methode gehören.

Um die Auswirkungen der Aufrufabhängigkeiten zu evaluieren, werden hier die Ergebnisse mit den Auswertungen aus Abschnitt 6.3.2.1 und Abschnitt 6.3.3.1 gegenüber gestellt. Dort wurde die Rückverfolgbarkeit mit Quelltexteinbettungen ohne Aufrufabhängigkeiten durchgeführt. Außer den Quelltexteinbettungen bleibt der Rest des Verfahrens gleich: Es werden auch hier Anforderungseinbettungen als Durchschnitt der Worteinbettungen berechnet. Das zugrundeliegende Worteinbettungsverfahren bleibt fastText mit dem generischen Standardmodell und die Ähnlichkeitsmetrik ist die Kosinusähnlichkeit.

Aufrufabhängigkeiten können auf der Klassen- oder Methodenebene inkludiert werden. Da bisher der Mehrheitsentscheid auf der Methodenebene besser funktioniert hat, werden die Aufrufabhängigkeiten für die Evaluation auch auf dieser Ebene miteinbezogen. Das Vorgehen des Mehrheitsentscheid ist ansonsten identisch wie in Abschnitt 6.3.3.1. Die Stimmenbestimmung erfolgt durch den Mehrheitsentscheidschwellwert und die Quelltextähnlichkeitsreduktionsfunktion ist hier ebenfalls das Maximum.

Im Entwurf wurden verschiedene Möglichkeiten genannt, die Aufrufabhängigkeiten zu berücksichtigen. In einer Möglichkeit wird pro Methode eine gewichtete Ähnlichkeitssumme (ÄS) berechnet. Zunächst werden Methodeneinbettungen als Durchschnitt ihrer Methodensignatur- und Klassennamenwörter berechnet. Damit werden Ähnlichkeiten zu allen Anforderungseinbettungen berechnet und mit dem Elementschwellwert (E-SW) gefiltert. Pro Methode wird danach (für jede Anforderung getrennt) eine gewichtete Summe über die Ähnlichkeiten der Methode und der direkten Nachbarn kalkuliert. Diese Ähnlichkeiten werden mit einem Mehrheitsentscheidschwellwert (M-SW) gefiltert und die übrigen Verbindungen zu den Anforderungen werden als Stimmen für den Mehrheitsentscheid der Klasse verwendet. Die Anforderungen mit der größten Stimmanzahl werden als vorläufige Verbindung für die Klasse festgelegt. Am Ende werden alle vorläufigen Verbindungen durch den Dateilevelschwellwert (D-SW) gefiltert. Die verbliebenen Verbindungen werden mit der Musterlösung ausgewertet.

Die Ergebnisse hiervon sind in Tabelle 6.13 und Abbildung 6.11 zu finden.
In Abschnitt 4.3.3.4 wurde erklärt, dass es Aufrufer- und Aufgerufenennachbarn gibt. Die

Tabelle 6.13: Beste F1-Werte für den Mehrheitsentscheid mit (ÄS) `SIG` in verschiedenen Varianten

Variante	Bestes F1	Ausb.	Präz.	D-SW	M-SW	E-SW	MG	D. Präz.
`SIG`	33,4%	40,1%	28,7%	0,67	-	-	-	- 25%
(M) `SIG`	32,7%	30,7%	35%	0,71	0,6	-	-	17,4%
(ÄS) `SIG`↕	33,7%	41,7%	28,3%	0,65	0,6	0	0,67	16,7%
(ÄS) `SIG`↑	**34,8%**	34,5%	35,1%	0,71	0,6	0,59	0,64	**18,2%**
(ÄS) `SIG`↓	33,5%	37%	30,5%	0,68	0,6	0	0,8	16,7%

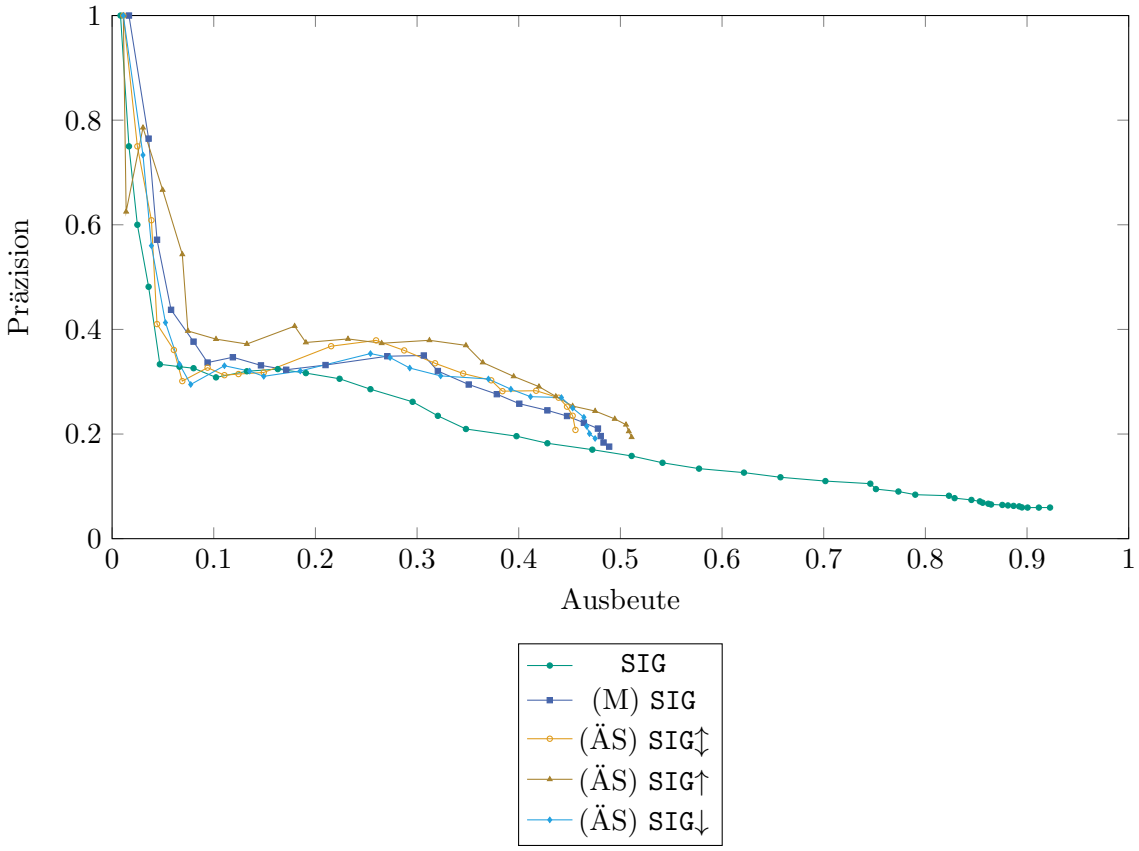


Abbildung 6.11: eTour: Vergleich der Varianten mit (ÄS) `SIG`

Pfeile geben an, welche Aufrufnachbarn miteinbezogen wurden. Ein ↑ zeigt an, dass nur Aufrufernachbarn (Nachbarmethoden, die die aktuelle Methode aufrufen) berücksichtigt wurden. ↓ und ↕ signalisieren dementsprechend, dass nur Aufgerufenennachbarn bzw. beide Nachbararten zur Berechnung hinzugezogen wurden. Die MG-Spalte in Tabelle 6.13 listet das Methodengewicht auf. Ein MG von $0,8$ bedeutet, dass die Ähnlichkeit der aktuellen Methode mit $0,8$ gewichtet wird, während die Nachbarmethoden mit insgesamt $1 - 0,8 = 0,2$ Gewicht in die Summe miteinfließen. Bei einem MG von $0,66$ ist die aktuelle Methode also doppelt so hoch gewichtet wie die Nachbarn. Es wurden verschiedene Schwellwerte und Methodengewichte ausprobiert. Die Tabelle und das Diagramm listen jeweils die Konfiguration mit dem besten F1-Wert auf.

In der Tabelle 6.13 ist zu sehen, dass die Miteinbeziehung der Aufrufbeziehungen bis zu zwei Prozent bessere F1-Werte bewirken. (ÄS) `SIG`↑ hat dabei unter den ÄS-Varianten den höchsten F1-Wert. Dies kann daran liegen, dass durch Weglassen der Aufgerufenennachbarn, die eine größere Wahrscheinlichkeit für Implementierungsdetails haben, Rauschen verringert wird. Bei Betrachtung von Abbildung 6.11 ist ebenfalls festzustellen, dass die Kurve von (ÄS) `SIG`↑ bei den meisten Schwellwerten immer über den anderen Kurven liegt. Dies manifestiert sich auch in der durchschnittlichen Präzision, die größer als bei

Tabelle 6.14: Beste F1-Werte für den Mehrheitsentscheid mit (VS) `SIG` im Vergleich zu den bisherigen Varianten

Variante	Bestes F1	Ausb.	Präz.	D-SW	M-SW	E-SW	MG	D. Präz.
`SIG`	27,8%	29,6%	26,2%	0,69	-	-	-	18,6%
(M) `SIG`	32,7%	30,7%	35%	0,71	0,6	-	-	17,4%
(ÄS) `SIG`↑	34,8%	34,5%	35,1%	0,71	0,6	0,59	0,64	18,2%
(VS) `SIG`↑	**36%**	34,8%	37,3%	0,71	0,6	-	0,78	**20%**

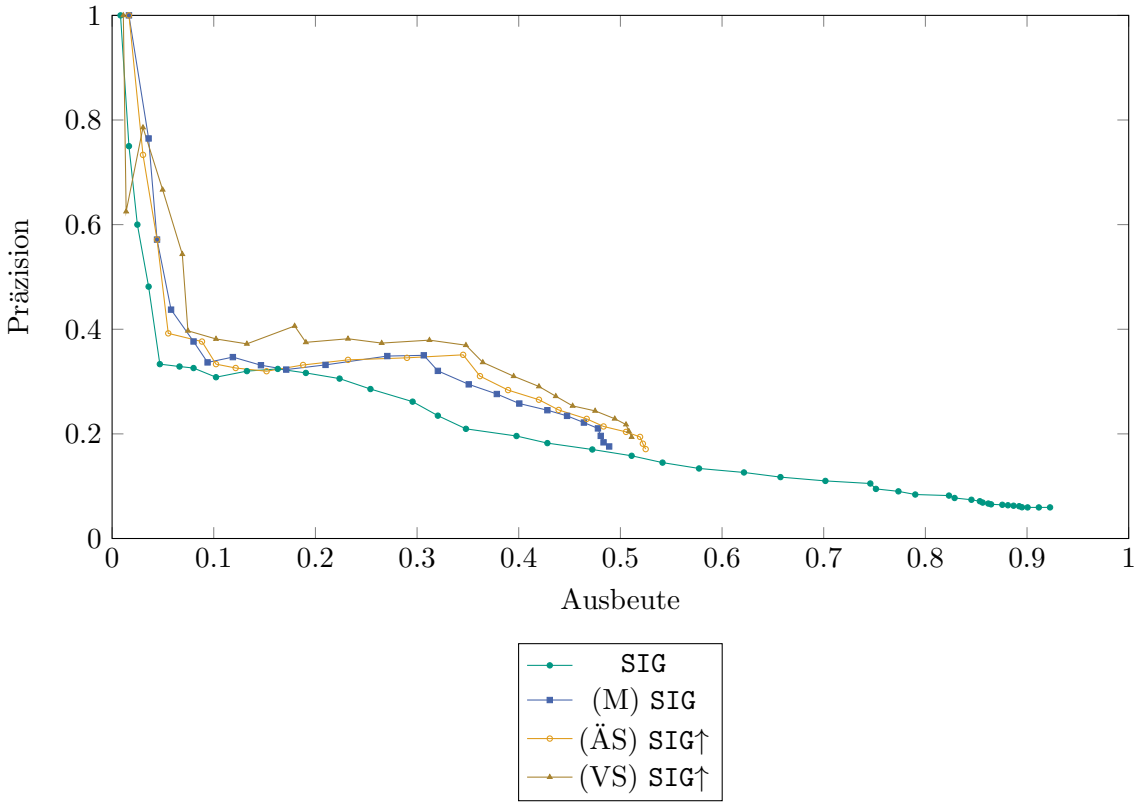


Abbildung 6.12: eTour: Vergleich der bisherigen Varianten mit (VS) `SIG`

den anderen ÄS-Varianten ist. Die anderen beiden ÄS-Varianten oszillieren dagegen um die Kurve von (M) `SIG`. Im obigen Vergleich kann folglich (ÄS) `SIG`↑ als bestes Verfahren angenommen werden. Bei iTrust (siehe Abbildung A.12) liegen die F1-Werte um ca. ein Prozent niedriger als beim Mehrheitsentscheid ohne Aufrufabhängigkeiten. Bei Betrachtung der Kurven oszilliert vor allem (ÄS) `SIG`↑ um die Kurve von (M) `SIG`. Die Differenzen sind hier zu gering, um eindeutige Aussagen zu treffen.
Es wurde ebenfalls die Möglichkeit untersucht, die Nachbarmethoden ungewichtet zu inkludieren. Dies hat jedoch schlechtere Ergebnisse bewirkt. Das bestätigt die Vermutung aus der Analyse, dass die aktuelle Methode für die Rückverfolgbarkeit am nützlichsten ist.

Eine andere Option für die Integration von Aufrufbeziehungen ist die Summe der Nachbarvektoren anstatt der Nachbarähnlichkeiten. Pro Methode wird eine Einbettung wie für `SIG` berechnet. Dieser Vektor wird mit allen Nachbarvektoren gewichtet aufsummiert und als neuer Vektor für die aktuelle Methode definiert. Mit diesen Vektoren wird schließlich ein regulärer Mehrheitsentscheid durchgeführt. Der Elementschwellwert wird hierbei nicht eingesetzt, da die Ähnlichkeit zwischen Methodeneinbettungen und Anforderungen (und nicht Anforderungssätzen) berechnet wird.

Abbildung 6.12 und Tabelle 6.14 zeigen die Auswertung hiervon im Vergleich zu den bisherigen Ergebnissen. Die Ausbeute-Präzisionskurve von (VS) `SIG`↑ liegt zu großen Teilen

Tabelle 6.15: Beste F1-Werte für (VS) `SIG` mit Kommentaren

Variante	Bestes F1	Ausb.	Präz.	D-SW	M-SW	MG	D. Präz
(VS) `SIG`↑	39,8%	45%	35,7%	0,7	0.49	0,85	28,1%
(VS) `SIG+KO`↕	**39,0%**	40,9%	37,3%	0,8	0,64	0,65	**26,5%**
(VS) `SIG+KO`↑	38,4%	36,7%	40,3%	0,8	0,61	0,66	25,3%
(VS) `SIG+KO`↓	35,3%	34,5%	36,1%	0,8	0,62	0,65	23,5%

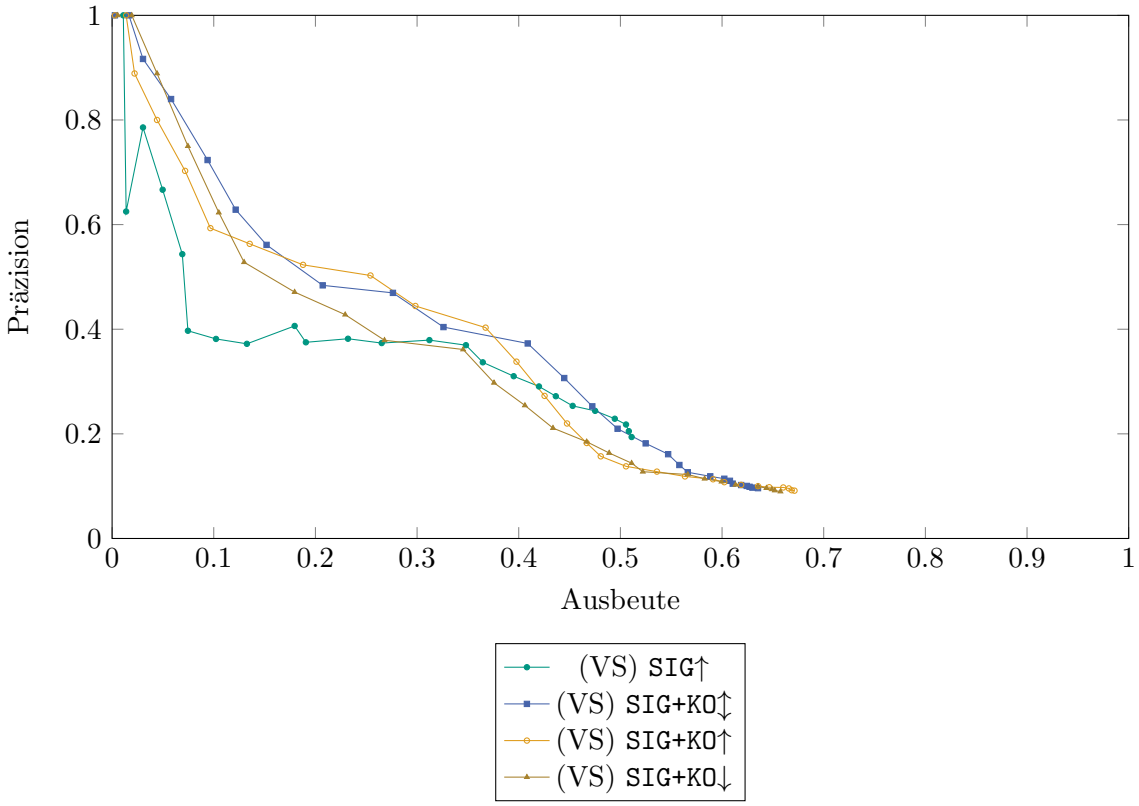


Abbildung 6.13: eTour: Vergleich der Varianten von (VS) `SIG` mit Kommentaren

etwas höher als die von (ÄS) `SIG`. Dies wird durch die größere durchschnittliche Präzision bestätigt. Auch der beste F1-Wert ist um 1,2% gestiegen. Dieses Ergebnis ist unerwartet, da aus der Analyse hervorging, dass die Ähnlichkeitssumme wegen der Nutzung eines Elementschwellwerts der Vektorsumme überlegen ist. Durch den zusätzlichen Schwellwert können Nachbarn mit kleinen Ähnlichkeitswerten aussortiert werden, womit das Rauschen verringert wird. Praktisch kann es aber sein, dass das Verfahren an sich nicht zuverlässig genug funktioniert und dass Methoden zu falschen Anforderungen eine hohe Ähnlichkeit aufweisen können. Wenn die Nachbarvektoren richtigerweise nur eine niedrige Ähnlichkeit zu der falschen Anforderung besitzen, kann die hohe Ähnlichkeit beim Vektordurchschnitt nach unten korrigiert werden. Bei der Ähnlichkeitssumme würden die niedrigen Ähnlichkeiten herausgefiltert werden, sodass die falsche hohe Ähnlichkeit verbleibt.
Für iTrust (Abbildung A.13) ist ebenfalls (VS) `SIG`↑ die bisher beste Variante.

Bei der Evaluation von Klasseneinbettungen haben sich Methodenkommentare als sehr hilfreich herausgestellt. In Tabelle 6.15 und Abbildung 6.13 sind Ergebnisse zu sehen, die (VS) `SIG` mit Methodenkommentaren erweitern. Die Methodenkommentarwörter werden dabei ungeordnet zusammen mit den Methodensignaturwörtern und dem Klassennamen auf Worteinbettungen abgebildet und anschließend mit einem Durchschnitt zu einer Methodeneinbettung aggregiert.

Bei den Ausbeute-Präzisionskurven lässt sich eine deutliche Verbesserung in den oberen Schwellwerten beobachten. Die durchschnittliche Präzision steigt auf bis zu 26,5% und der beste F1-Wert auf bis zu 39%. Nützlichkeit von Methodenkommentaren für die Rückverfolgbarkeit wird hier also nochmals bestätigt. Bezüglich der Nachbarmethoden erzielt dieses Mal die Variante, in der sowohl Aufrufer- als auch Aufgerufenennachbarn berücksichtigt werden, den besten F1-Wert. Bei Betrachtung des Diagramms liegen die Kurven von (VS) `SIG+KO`↕ und (VS) `SIG+KO`↑ relativ nah beieinander. Abschnittsweise liegt manchmal die eine, manchmal die andere Kurve höher. Im Falle von iTrust (Abbildung A.14) hat `SIG+KO`↑ den höchsten F1-Wert. Beim hohen F1-Wert für eTour kann es sich also auch um einen Ausreißer handeln.

Die Variante mit dem doppelten Mehrheitsentscheid aus dem Entwurf (Abschnitt 4.3.6) wurde ebenfalls evaluiert. Sie liefert jedoch deutlich schlechtere Ergebnisse. Beim einfachen Mehrheitsentscheid wurde bereits angeführt, dass er zu restriktiv sein kann. Dieser Effekt wird beim doppelten Mehrheitsentscheid nochmals verstärkt.

Insgesamt lässt sich feststellen, dass die Miteinbeziehung von Aufrufabhängigkeiten in Form einer Vektor- oder Ähnlichkeitssumme in der Tat eine Verbesserung der Rückverfolgbarkeit darstellt. Bezüglich der Nachbararten sollten tendenziell Aufrufernachbarn oder beide Nachbararten inkludiert werden.

6.3.4 Ähnlichkeitsmetriken

In der bisherigen Evaluation wurde ausschließlich die Kosinusähnlichkeit als Metrik eingesetzt. Im Entwurf wurden jedoch noch andere Ähnlichkeitsmaße präsentiert.
Abschnitt 6.3.4.1 beschäftigt sich mit der Ähnlichkeit, die durch ein BERT-Modell gewonnen werden kann und in Abschnitt 6.3.4.2 wird die Word Mover's Distance verwendet.

6.3.4.1 Abbildung mit BERT

BERT ist in der Lage, zwei Sätze entgegen zu nehmen und eine Wahrscheinlichkeit zu berechnen, ob der zweite Satz ein nachfolgender Satz des ersten ist, das heißt BERT kann eine Klassifikation von Sätzen vornehmen. Dies kann genutzt werden, einen Klassifikator für Anforderungen und Quelltext zu trainieren: Es wird jeweils ein Anforderungssatz und einen Methodenname als Eingabe genommen und vorhergesagt, ob zwischen ihnen eine

Tabelle 6.16: Beste F1-Werte für den Mehrheitsentscheid mit Ähnlichkeiten durch BERT

Variante	Bestes F1	Ausb.	Präz.	D-SW	M-SW	E-SW	D. Präz.
(M) `SIG`	32,7%	30,7%	35%	0,71	0,6	-	17,4%
(B) `SIG`	**41,9%**	87,5%	27,5%	0,65	0	0	23,3%
(B) Nur Meth.komm.	27,7%	55,6%	18,4%	0,67	0,47	0	9,0%

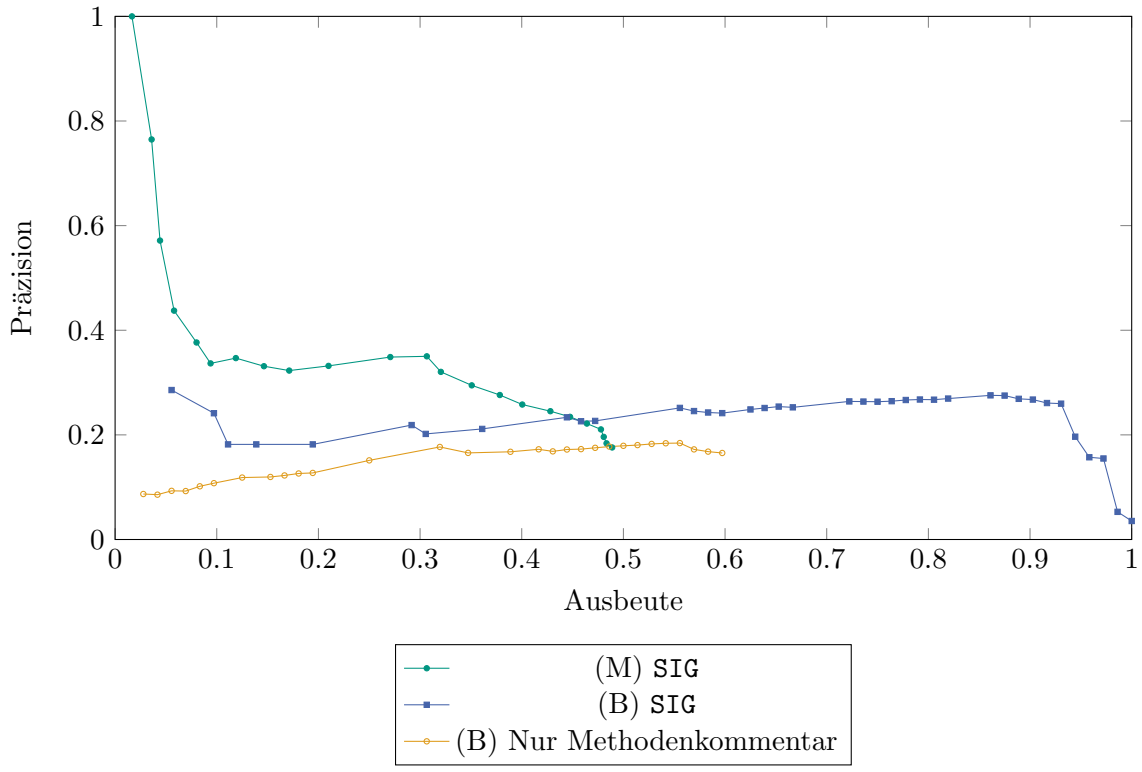


Abbildung 6.14: eTour: Ausbeute-Präzisionskurven des Mehrheitsentscheid mit Ähnlichkeiten durch BERT

Rückverfolgbarkeitsverbindung besteht. Dem Methodennamen kann zusätzlich der Klassenname vorangestellt werden, um einen künstlichen Satz wie in Abschnitt 4.3.3.2 beschrieben zu erzeugen. Aus Dronology, eTour und iTrust werden 80% der Daten zum Training ausgewählt. Das genaue Vorgehen und die Trainingsparameter werden in Abschnitt 5.5 erklärt.

Das daraus resultierende Modell wurde anschließend verwendet, um die Rückverfolgbarkeit auf den verbliebenen 20% Daten zu evaluieren. Pro Methodenname (inklusive Klassenname als Präfix) und Anforderungssatz wird eine Ähnlichkeit durch das BERT-Modell berechnet. Anschließend wird eine Mehrheitsentscheidung wie in Abbildung 4.5 durchgeführt. Als Ähnlichkeitsreduktionsfunktion wurde für die Anforderungen der Durchschnitt und für den Quelltext das Maximum verwendet. Die Stimmenbestimmung wird durch einen Mehrheitsentscheidschwellwert festgelegt. Pro Klassen werden Verbindungen zu den Anforderung(en) mit der größten Stimmanzahl aufgebaut.

In Tabelle 6.16 und Abbildung 6.14 sind die Resultate zu sehen. Zum Vergleich wurde auch die Mehrheitsentscheidvariante mit Kosinusähnlichkeit aus Abschnitt 6.3.3.1 eingezeichnet. Auf dem ersten Blick sieht der F1-Wert von $41,9\%$ relativ gut aus. Bei Begutachtung der Ausbeute, an dem sich der beste F1-Wert berechnet, ist jedoch festzustellen, dass hierfür Mehrheits- und Elementschwellwerte von Null angesetzt wurden. Das bedeutet, dass Schwellwerte größer als Null schlechtere F1-Werte bewirken. Daraus lässt sich folgern, dass

viele korrekte Verbindungen niedrige Ähnlichkeiten besitzen, die bereits bei einer kleinen Erhöhung der Schwellwerte entfernt werden und das Ergebnis verschlechtern. Dies wird durch die Ausbeute-Präzisionskurve bestätigt: (B) `SIG` liegt nur bei niedrigeren Schwellwerten über der Kurve von (M) `SIG`. Die höchste Präzision von (B) `SIG` wird bei hoher Ausbeute (= niedriger Schwellwert) erreicht. Danach sinkt die Präzision, je größer der Schwellwert wird. Das bedeutet, dass mehr richtige als falsche Verbindungen aussortiert werden, was dadurch verursacht wird, dass richtige Verbindungen kleinere Ähnlichkeiten als falsche besitzen.

Die niedrige Ähnlichkeit durch BERT kann dadurch entstehen, dass die künstlichen Methodensätze keine echten natürlichsprachigen Sätze sind. In Abbildung 6.14 ist eine weitere Kurve zu sehen, in der BERT nicht mit künstlichen Sätzen aus dem Methodennamen, sondern Kommentarsätzen trainiert wurde. Der Rest des Trainingsverfahrens ist identisch. Zwar sind Kommentarsätze im Gegensatz zu den Methodennamen natürlichsprachige Sätze, jedoch besitzt nicht jede Methode einen Kommentar. Das schlechtere Ergebnis der Kommentarvariante kann also dadurch erklärt werden, dass nicht alle Implementierungen der Anforderung abgedeckt wurden. Jedoch ist auch bei dieser Kurve zu sehen, dass die Präzision bei niedrigerer Ausbeute ebenfalls sinkt - sie verläuft sehr ähnlich zu der Kurve von (B) `SIG`. Auch hier haben also korrekte Verbindungen tendenziell niedrigere Ähnlichkeiten als falsche. Dies ist ein Indiz dafür, dass diese Ähnlichkeitsmetrik nicht gut funktioniert, da eigentlich korrekte Verbindungen die größeren Ähnlichkeiten besitzen sollten.

Die anderen BERT-Varianten, die in der Implementierung in Tabelle 5.3 aufgeführt sind, erzielen keine besseren Ergebnisse. Auch die Kombination mehrerer Quellen bewirkt keinen positiven Einfluss. Dies kann daran liegen, dass die Kombinationsmethoden zu restriktiv sind oder zu viele verrauschte Elemente zulassen.

Möglicherweise funktionieren die BERT-Varianten besser, wenn höhere Ähnlichkeiten grundsätzlich aussortiert werden und die niedrigen Ähnlichkeiten auf $[0, 1]$ abgebildet werden. In der jetzigen Form kann die BERT-Klassifikation jedoch nicht eindeutig als bessere Ähnlichkeitsmetrik als die Kosinusähnlichkeit konstatiert werden. Außerdem benötigt diese Abbildung das Vorhandensein einer Musterlösung, um den BERT-Klassifikator zu trainieren. Das ist in der Praxis meist nicht gegeben.

6.3.4.2 Abbildung mit Word Mover's Distance

Die Word Mover's Distance (Abschnitt 2.8.3) ist eine weitere Metrik, die anstatt der Kosinusähnlichkeit verwendet werden kann. Die WMD ist eine Distanz zwischen Mengen von Wörtern bzw. Einbettungen, daher müssen Anforderungen und Quelltexte als Bag-of-Embeddings repräsentiert werden (siehe Abschnitt 4.2.3.4 bzw. Abschnitt 4.3.3.6). Dafür gibt es bei beiden Artefakten mehrere Optionen. Für die Anforderungsseite kann eine komplette Anforderung (`A`) oder jeder Satz getrennt (`AS`) als Bag-of-Embedding dargestellt werden. Für den Quelltext werden im Folgenden nur Methodensignaturen und -kommentare betrachtet, da diese sich bisher als sehr hilfreich für die Rückverfolgbarkeit herausgestellt haben. Es können folglich Methodensignaturwörter (inklusive Klassenname) mit oder ohne Kommentarwörter als Bag-Of-Embeddings repräsentiert werden. Ersteres wird mit `MS`, Letzteres mit `MK` symbolisiert.

Aus den obigen vier Mengen können vier mögliche Kombination gebildet werden. Es wurden jeweils zwischen Klassenelementen (`MS` oder `MK`) und Anforderungselementen (`A` oder `AS`) die WMD berechnet. Bei den Schwellwerten ist zu beachten, dass die Ähnlichkeit größer ist, je kleiner die WMD ist. Das bedeutet ein Schwellwert von Eins filtert in diesem Fall nichts heraus und ein Schwellwert von Null alles. Damit wird anschließend ein Mehrheitsentscheid durchgeführt. Der Ablauf mit Anforderungssätzen entspricht dem Vorgehen

in Abbildung 4.5 und die Abbildung mit ganzen Anforderungen ist in Abbildung 4.4 illustriert. Bei der Stimmenbestimmung wird ein Mehrheitsentscheidschwellwert eingesetzt. Als Verbindungen werden die Anforderung(en) mit der maximalen Stimmanzahl festgelegt. Bezüglich der Reduktionsfunktionen wurden das Minimum und der Durchschnitt überprüft. In der nachfolgenden Auswertung wird jeweils die Konfiguration gezeigt, die den besten F1-Wert erzeugt. Die Kombinationen mit ganzen Anforderungen (`A`) nutzen das Minimum als Quelltextähnlichkeitsreduktionsfunktion. Die anderen beiden Varianten mit `AS` nutzen den Durchschnitt als Anforderungsreduktionsfunktion. `MK+AS` verwendet für die Quelltextähnlichkeitsreduktion das Minimum und `MS+AS` den Durchschnitt. Da die WMD auf der euklidischen Distanz basiert, muss hier auf $[0, 1]$ normiert werden, damit die Schwellwertfilterung unverändert eingesetzt werden kann. Dies wird einmal vor dem ersten Schwellwertfilter (je nach Verfahren Element- oder Mehrheitsentscheidschwellwertfilter) durchgeführt. Außerdem hat sich durch Versuche ergeben, dass eine weitere Normierung vor dem Dateilevelschwellwertfilter hilfreich ist, daher wird dies hier auch genutzt.

Tabelle 6.17: Beste F1-Werte für den Mehrheitsentscheid mit WMD-Kombinationen

Variante	Bestes F1	Ausb.	Präz.	D-SW	M-SW	E-SW	MG	D. Präz.
`MS+AS`	**51,8%**	49,4%	54,4%	0,58	0,64	0,65	-	**41,9%**
`MK+AS`	46,9%	48,1%	45,8%	0,59	0,63	0,65	-	39,1%
`MS+A`	47,3%	52,2%	43,2%	0,67	0,61	-	-	36,2%
`MK+A`	44,1%	38,4%	51,7%	0,58	0,59	-	-	32,3%
(VS) `SIG+KO↕`	39,0%	40,9%	37,3%	0,8	0,64	-	0,65	26,5%
(M) `SIG+KL+KO`	34,4%	32,6%	36,3%	0,76	0.45	-	-	25,8%

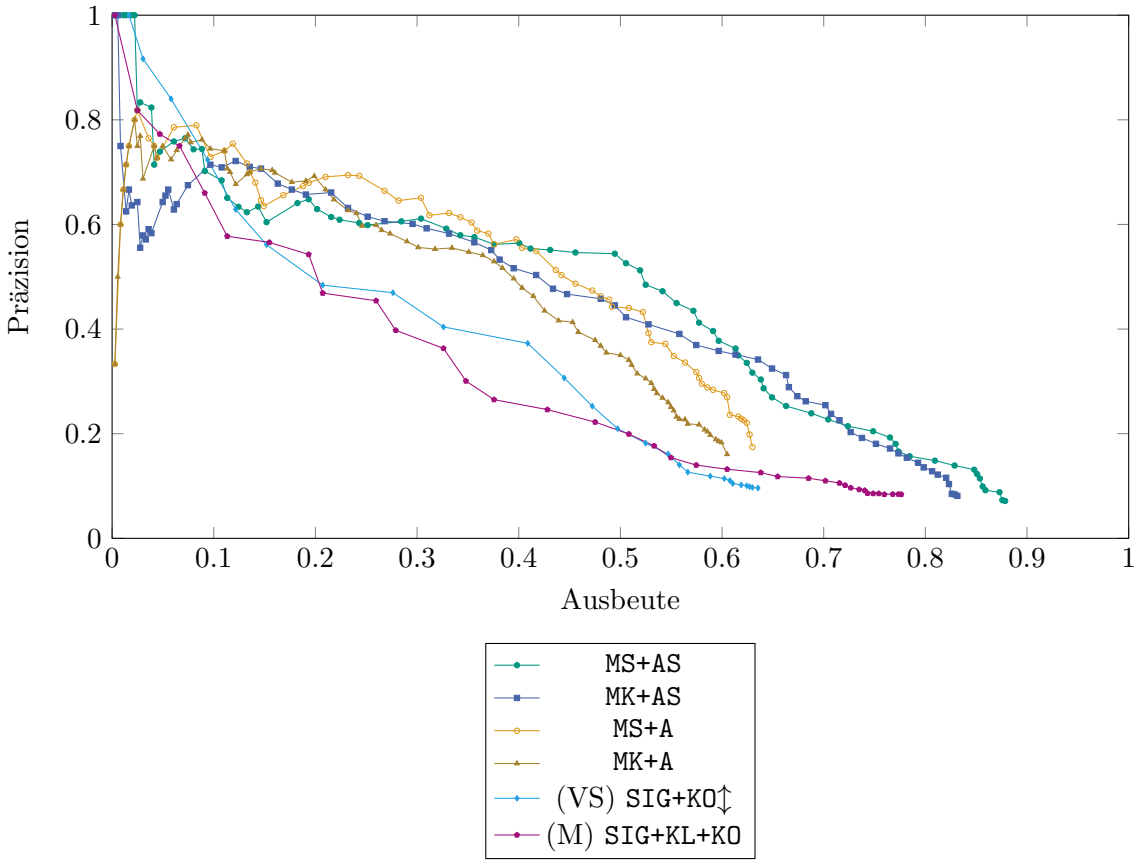


Abbildung 6.15: eTour: Ausbeute-Präzisionskurven des Mehrheitsentscheid mit verschiedenen WMD-Kombinationen

In Tabelle 6.17 und Abbildung 6.15 sind die Evaluationsergebnisse dazu aufgezeigt. Es ist zu beobachten, dass `MS+AS` bezüglich des F1-Wertes am besten funktioniert. WMD hat gegenüber der Kosinusähnlichkeit den Vorteil, dass alle Ähnlichkeiten der Wortpaare zwischen Methodenwörtern und Anforderungswörtern betrachtet werden. Bei der bisherigen Vorgehensweise wurde ein Durchschnittsvektor gebildet, der mit der Kosinusähnlichkeit verglichen wird. Durch den Durchschnitt gehen Informationen verloren, was die Ähnlichkeit negativ beeinträchtigt.

An den obigen Ergebnissen ist zu sehen, dass die Varianten ohne Methodenkommentare erstaunlicherweise etwas besser funktionieren, obwohl bei den Verfahren aus den vorigen Abschnitten damit immer eine Verbesserung erzielt werden konnte. Ein Erklärungsansatz bestünde darin, dass der Informationsverlust durch die Durchschnittsbildung zufällig Rauschen aus den Kommentaren reduziert hat, während diese bei der WMD miteinbezogen wurden.

Insgesamt ist jedoch festzustellen, alle F1-Bestwerte der vier WMD-Varianten deutlich über die der besten Variante mit Kosinusähnlichkeit liegt. Auch die Kurven der WMD-Verfahren verlaufen zum größeren Teil über der Kurve von (VS) `SIG+KO↕`. Außerdem sind

Tabelle 6.18: Beste F1-Werte für den WMD-Mehrheitsentscheid mit Mehrheitentscheidschwellwert und Top-N

Variante	Bestes F1	Ausbeute	Präz.	D-SW	M-SW	E-SW	D. Präz.
`MS+AS`	**51,8%**	49,4%	54,4%	0,58	0,64	0,65	**41,9%**
`MS+A`	47,3%	52,2%	43,2%	0,67	0,61	-	36,2%
(T) `MS+AS`	50,5%	47,0%	54,7%	0,59	0,35	0,64	35,6%
(T) `MS+A`	49,4%	45%	54,7%	0,51	0,2	0,65	38,5%

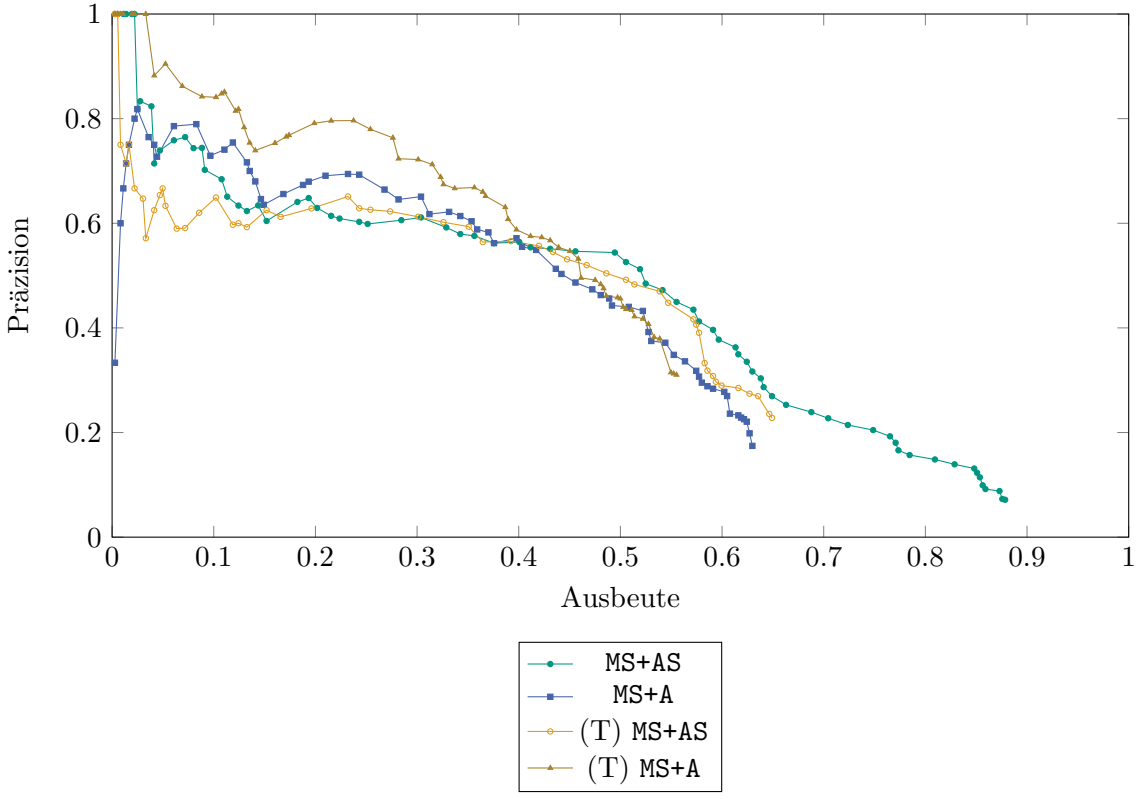


Abbildung 6.16: eTour: Ausbeute-Präzisionskurven des WMD-Mehrheitsentscheid mit Mehrheitentscheidschwellwert und Top-N

die durchschnittliche Präzisionen von `MS+A` und `MS+AS` höher als bei den Varianten mit Kosinusähnlichkeiten.

Bei iTrust (siehe Abbildung A.15) erzielen die obigen WMD-Variante schlechtere F1-Werte als (M) `SIG+KL+KO`. Dies kann jedoch auch am iTrust-Projekt selbst liegen, was in Abschnitt 6.4.3 genauer erläutert wird. Unter den Varianten sind allerdings diejenigen mit Methodenkommentaren besser.

Für die Stimmenbestimmung ist Top-N eine Alternative zum Mehrheitsentscheidschwellwert. Diese Variante wurde für `MS+AS` und `MS+A` untersucht. Alle anderen Parameter des Mehrheitsentscheids bleiben unverändert. Die einzige Ausnahme bildet der Elementschwellwert, der für Top-N zusätzlich verwendet werden kann, um Ähnlichkeiten zu *ganzen* Anforderungen zu filtern, bevor Top-N angewendet wird. Bei der Mehrheitsentscheidvariante ist der Elementschwellwert nicht nötig, da der Mehrheitsentscheidschwellwert bereits diese Filterung vornimmt.

In Tabelle 6.18 und Abbildung 6.16 sind die Ergebnisse dargestellt. Die Top-N-Varianten werden mit einem (T) gekennzeichnet. Die Werte in der M-SW-Spalte für die Top-N-Verfahren sind die prozentuale Anzahl an Anforderungen, die pro Klassenelement als Stimme festgelegt werden. Das N entspricht bei einem Wert von 0,35 also circa ein Drittel der Gesamtzahl an Anforderungen von eTour (so viele Stimmen hat also jedes Klassenelement)
In der Auswertung ist zu beobachten, dass Top-N sowohl besser als auch schlechter als der Mehrheitsentscheidschwellwert funktionieren kann. Dies schlägt sich in den F1-Werten und in den durchschnittlichen Präzisionen nieder. Gleiches gilt für eine Evaluation mit iTrust (siehe Abbildung A.16).
Bei der Evaluation in Abschnitt 6.3.3.1 wurde der Punkt angesprochen, dass der Mehrheitsentscheid zu restriktiv sein könnte. Durch Top-N erhält jedes Klassenelement immer (bis zu) N Stimmen, während beim Mehrheitsentscheidschwellwert Klassenelemente ohne Stimmen existieren können, falls die Ähnlichkeiten alle unter dem Schwellwert liegen. Dadurch stimmen mehr Klassenelemente ab, was die Abdeckung der Rückverfolgbarkeitsverbindungen verbessern kann. Jedoch können dadurch auch Klassenelemente Stimmen bekommen, die keine korrespondierende Anforderungen haben und korrekterweise keine Stimmen besitzen sollten. In diesem Fall würde Top-N schlechter funktionieren. Allgemein kann also kein eindeutiger Favorit zwischen Top-N oder Mehrheitsentscheidschwellwert für die Stimmenbestimmung festgelegt werden.

In der Implementierung (Abschnitt 5.5) bzw. in Tabelle 5.5 sind außer den obigen WMD-Verfahren noch einige weitere aufgeführt. Für eTour haben diese Verfahren schlechtere Ergebnisse erzielt, da sie auf der Klassenebene arbeiten oder nur Kommentare ohne Methodensignaturen berücksichtigen. Dies wird hier nicht weiter ausgeführt.

6.4 Vergleich zwischen Projekten

In diesem Abschnitt werden die besten Verfahren pro Projekt in Abschnitt 6.4.1, Abschnitt 6.4.2, Abschnitt 6.4.3, Abschnitt 6.4.4 und Abschnitt 6.4.5 vorgestellt. In den einzelnen Abschnitten werden auch die Schwierigkeiten und Eigenschaften der Projekte diskutiert. Anschließend wird in Abschnitt 6.4.6 ausgewertet, welche Verfahren projektübergreifend am besten funktionieren.

6.4.1 eTour

eTour wurde bereits in Abschnitt 6.3 ausführlich evaluiert. Das Verfahren mit dem besten F1-Wert ist `MS+AS` mit der WMD als Ähnlichkeitsmetrik und erreicht einen F1-Wert von 51,8%. Im Vergleich zur naiven Vorgehensweise aus Tabelle 6.1 stellt dies eine absolute Verbesserung von ca. 40% dar.

Quelltextausschnitt 6.1: Beispielkommentare in eTour.

```
/**
* Interface for the tag handler by the Operator Agency
*/
public interface ITagAgencyOperatorManager extends ITagCommonManager

/**
* Class that implements the interface SearchPreference
*/
public class DBSearchPreference implements IDBSearchPreference

/**
* Modify a convention in the database
*
* @Param data pConvention Convention of the Convention to be updated
* @Return True if there 'was a modified false otherwise
*/
public boolean modifyConvention(BeanConvention pConvention)
```

Die Anforderungen von eTour sind eigentlich Anforderungsfälle, die eine Überschrift besitzen. Diese Überschrift ist sehr hilfreich für die Rückverfolgbarkeit, da sie sich oft mit den Bezeichnern der zu verbindenden Klassenelemente überschneiden. Beispielsweise besitzt ein Anwendungsfall namens „ViewConventionHistory“ Verbindungen zu den Klassen „DBConvention“, „IDBConvention“ und „BeanConvention“. Durch das Vorkommen des gleichen Wortes steigt die Ähnlichkeit der berechneten Anforderungs- und Quelltexteinbettungen und erleichtert die Rückverfolgbarkeit. Dies ist vermutlich auch eine Ursache dafür, dass sich `SIG+KL` gegenüber `SIG` bei den Klasseneinbettungen in Abschnitt 6.3.2.1 stark verbessert hat.

Kommentare von eTour sind oft weniger hilfreich oder nicht vorhanden. in Quelltextausschnitt 6.1 sind einige Beispiele aufgeführt. Diese Kommentare stellen inhaltlich zum Großteil nur eine Wiederholung der Bezeichner dar und liefern semantisch keine neue Informationen. Möglicherweise hat dies auch dazu beigetragen, warum bei den WMD-Varianten in Abschnitt 6.3.4.2 die Verfahren mit Kommentaren nicht besser waren. Nützlicher für die Rückverfolgbarkeit wäre eine abstraktere Beschreibung der Funktionalität. Dennoch sind Kommentare durch ihre natürlichsprachige Satzform ähnlicher zu Anforderungen und können dadurch die Rückverfolgbarkeit verbessern, selbst wenn sie keine hilfreiche semantischen Informationen beinhalten.

6.4.2 LibEST

In Abbildung 6.17 und Tabelle 6.19 sind die Werte für das Verfahren mit dem besten F1-Wert für LibEST zu sehen. Es handelt sich um ein Verfahren mit Mehrheitsentscheid, in der zwischen Anforderungssätzen und Klassenelementen bestehend aus Methodensignatur und -kommentar die WMD berechnet wird. Für die Stimmenbestimmung wurde ein Mehrheitsentscheidschwellwert verwendet, die Ähnlichkeitsreduktionsfunktion für Anforderungen ist das Minimum und für Quelltext der Durchschnitt. Wie bei eTour ist an der Ausbeute-Präzisionskurve ein Knick bei hohen Schwellwerten zu sehen. Das bedeutet, dass auch hier das Verfahren bei großen Schwellwerten nicht zuverlässig genug funktioniert. Der F1-Wert von 60,1% liegt viel höher als bei anderen Projekten. Dazu muss beachtet werden, dass LibEST wie in Abschnitt 6.1 ausgeführt, bereits einen hohen naiven F1-Wert von 43,8% besitzt. Die Ursache liegt in der großen Anzahl an korrekten Verbindungen. 28% aller möglichen Verbindungen sind valide.

Tabelle 6.19: Genaue Werte für das Verfahren mit dem besten F1 Wert für LibEST

Variante	Bestes F1	Ausbeute	Präz.	D-SW	M-SW	E-SW	D. Präz.
MK+AS	60,1%	75%	50,1%	0,51	0,66	1	48,4%

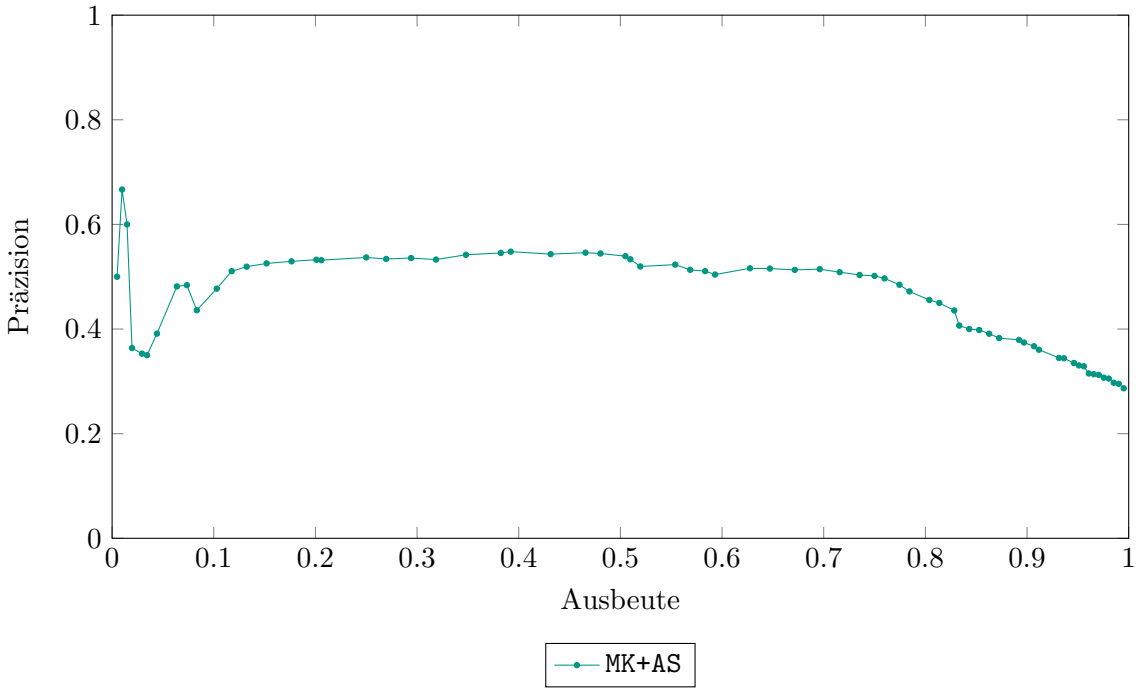


Abbildung 6.17: LibEST: Ausbeute-Präzisionskurven für das Verfahren mit dem besten F1-Wert

6.4.3 iTrust

Das Verfahren mit dem besten F1-Wert für iTrust ist ein Mehrheitsentscheid mit Kosinusähnlichkeit. Anforderungseinbettungen werden aus dem Durchschnitt ihrer Wortvektoren berechnet. Auf der Quelltextseite findet ein Mehrheitsentscheid zwischen Klassenname, Klassenkommentar, Methodenkommentar, Methodensignatur und Methodenrumpf (SIG+KL+KO+KKO+R) statt. Methodenkommentar und -rumpf werden also separat von den Methodensignaturen behandelt, das heißt sie werden nicht mit den Methodensignaturen zusammen aggregiert. Die genauen Evaluationswerte des Verfahrens sind in Abbildung 6.18 und Tabelle 6.20 eingetragen. Zum Vergleich sind auch die Werte für SIG+KL+KO+KKO+R mit Klasseneinbettungen und das WMD-Verfahren mit dem besten F1-Wert für iTrust zu sehen. Es ist überraschend, dass ein Mehrheitsentscheid mit Kosinusähnlichkeit bezüglich des F1-Wertes besser als ein WMD-Verfahren funktioniert. Auch die durchschnittliche Präzision ist bei der Kosinusähnlichkeit besser. Jedoch muss beachtet werden, dass bei MK+A ausschließlich Methodensignaturen und -kommentare berücksichtigt wurden. Aus diesem Grund sind MK+A und (M) SIG+KL+KO+KKO+R nicht direkt vergleichbar. Möglicherweise erzielt ein WMD-Verfahren, das die gleichen Elemente wie SIG+KL+KO+KKO+R inkludiert, noch bessere Ergebnisse.

Allgemein ist bei iTrust festzustellen, dass die F1-Werte in einem niedrigeren Bereich liegen als bei anderen Projekten. Eine Ursache ist die geringe Abdeckung der Artefakte in der Musterlösung. Bei der obigen Auswertung von (M) SIG+KL+KO+KKO+R wurden insgesamt 250 Verbindungen gefunden, von denen nur 70 richtige Verbindungen sind. Daraus lässt sich schließen, dass das Verfahren fälschlich wahre Verbindungen noch nicht gut genug erkennen bzw. aussortieren kann.

Tabelle 6.20: Genaue Werte für das Verfahren mit dem besten F1 Wert für iTrust

Variante	Bestes F1	Ausbeute	Präz.	D-SW	M-SW	D. Präz.
`SIG+KL+KO+KKO+R`	21,8%	25,5%	19,0%	0,82	-	13,3%
(M)`SIG+KL+KO+KKO+R`	**26,1%**	24,5%	28%	0,82	0,32	**14,2%**
`MK+A`	24,2%	26,2%	23,4%	0,52	0,56	13,1%

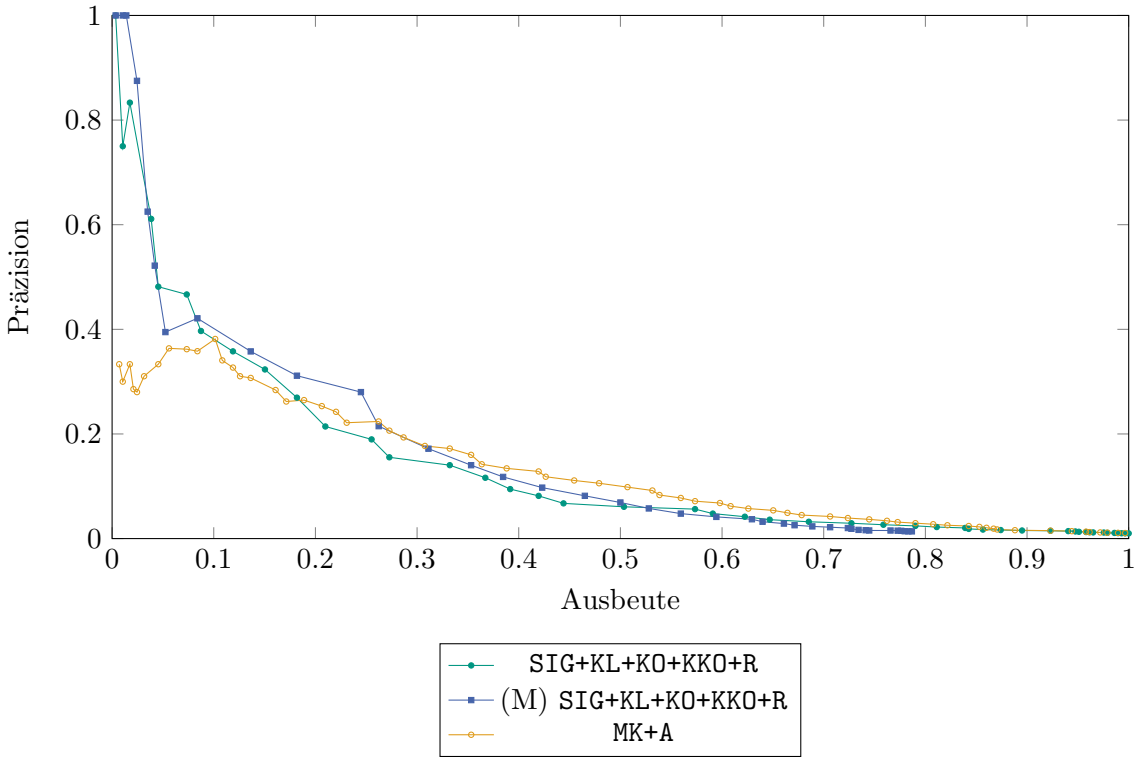


Abbildung 6.18: iTrust: Ausbeute-Präzisionskurven für das Verfahren mit dem besten F1-Wert

Eine weitere Schwierigkeit bei iTrust liegt darin, dass 21 Klassen von der Klasse `BeanValidator` erben und 28 Klassen die Schnittstelle `BeanLoader` implementieren. Diese Klassen besitzen bis auf einen generischen Typparameter die gleichen Methodensignaturen. Dadurch sind sie schwer voneinander zu differenzieren, obwohl sie nicht alle zur den gleichen Anforderungen gehören. Der größte Unterschied in diesen Klassen ist der Klassenname und die Methodenrümpfe weißen auch Differenzen auf. Dies trägt wahrscheinlich auch dazu bei, dass die WMD-Verfahren bei iTrust schlechter abschneiden als bei (M) `SIG+KL+KO+KKO+R`, da beim Letzteren Methodenrümpfe und Klassennamen separat berücksichtigt werden.

Wie in Abschnitt 6.1 gezeigt, beträgt der naive F1-Wert nur 2% für iTrust, also hat sich der F1-Wert damit um 24,1% verbessert, womit zumindest die absolute Verbesserung höher ist als bei LibEST.

6.4.4 Dronology

Bei Dronology passen Musterlösung und Artefakte nicht genau zusammen. Die validen Verbindungen des Goldstandards referenzieren zum Großteil Artefakte, die nicht verfügbar sind. Gemäß der Dronology-Website wurden der Datensatz zu den Musterlösungen noch nicht veröffentlicht. Bei den vorhandenen Artefakten handelt es sich vermutlich um eine frühere Version von Dronology.

Tabelle 6.21: Genaue Werte für SIG+KL mit Dronology

Variante	Bestes F1	Ausbeute	Präz.	D-SW	D. Präz.
SIG+KL	13,7%	11,7%	16,7%	0,81	6,4%

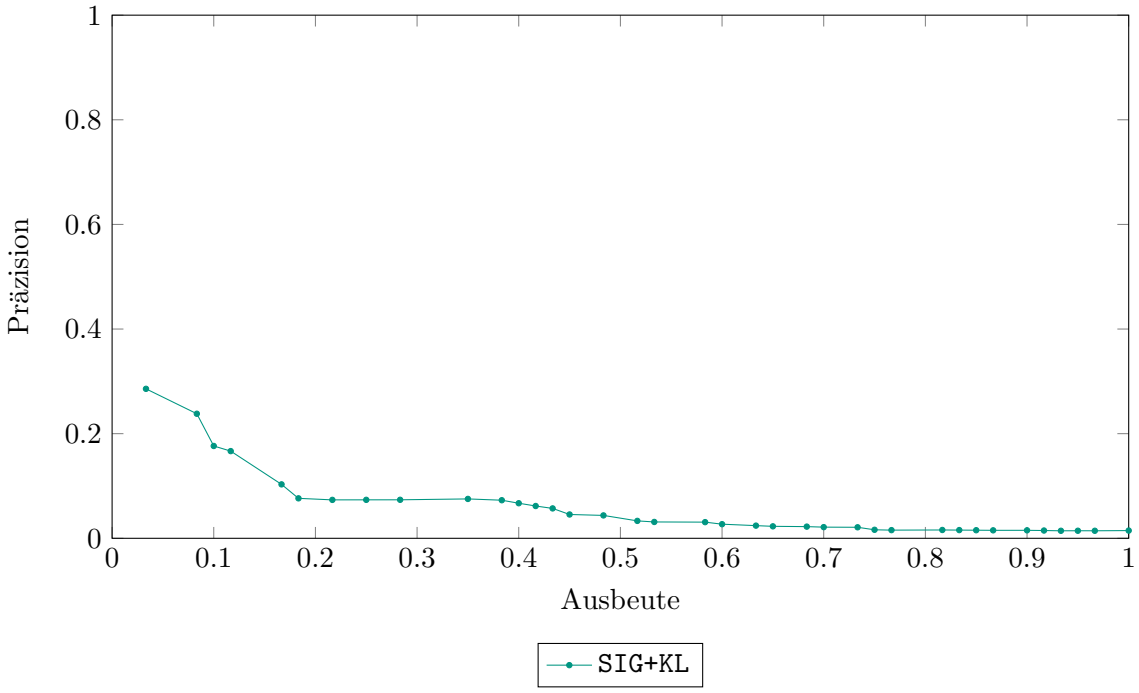


Abbildung 6.19: Dronology: Ausbeute-Präzisionskurven für SIG+KL

Tabelle 6.23: Kosinusähnlichkeit zwischen Übersetzungswortpaaren

Englisch	Italienisch	Kosinusähnlichkeit
cat	gatto	0,56
food	piatto	0,4
green	verde	0,71
sun	sole	0,6
beautiful	bello	0,49
search	ricercare	0,47
drink	bere	0,7

Die Unvollständigkeit macht sich auch in der Rückverfolgbarkeit bemerkbar. In Tabelle 6.21 und Abbildung 6.19 ist beispielhaft die Auswertung von SIG+KL als Klasseneinbettungen mit Kosinusähnlichkeit zu sehen. Die Werte liegen deutlich unter denen der anderen Projekte.

Für die weitere Evaluation wird Dronology außen vor gelassen und der Fokus auf die anderen Projekte gelegt.

6.4.5 SMOS

Die Verfahren mit dem besten F1-Wert sind für SMOS in Abbildung 6.20 und Tabelle 6.22 aufgelistet. (M) SIG+KL+KO ist der Mehrheitsentscheid mit Kosinusähnlichkeit, der analog wie für eTour in Abschnitt 6.3.3.1 durchgeführt wurde. (W) SIG+KL ist ein WMD-Verfahren, welches auf der Klassenebene abbildet. Eine Anforderung wird als Menge seiner

Tabelle 6.22: Genaue Werte für die besten Verfahren mit SMOS

Variante	Bestes F1	Ausbeute	Präz.	D-SW	M-SW	D. Präz.
(M) `SIG+KL+KO`	**32,1%**	52,7%	23%	0,68	0,38	22,7%
(W) `SIG+KL`	31,6%	45%	24,3%	0,53	-	**26,7%**

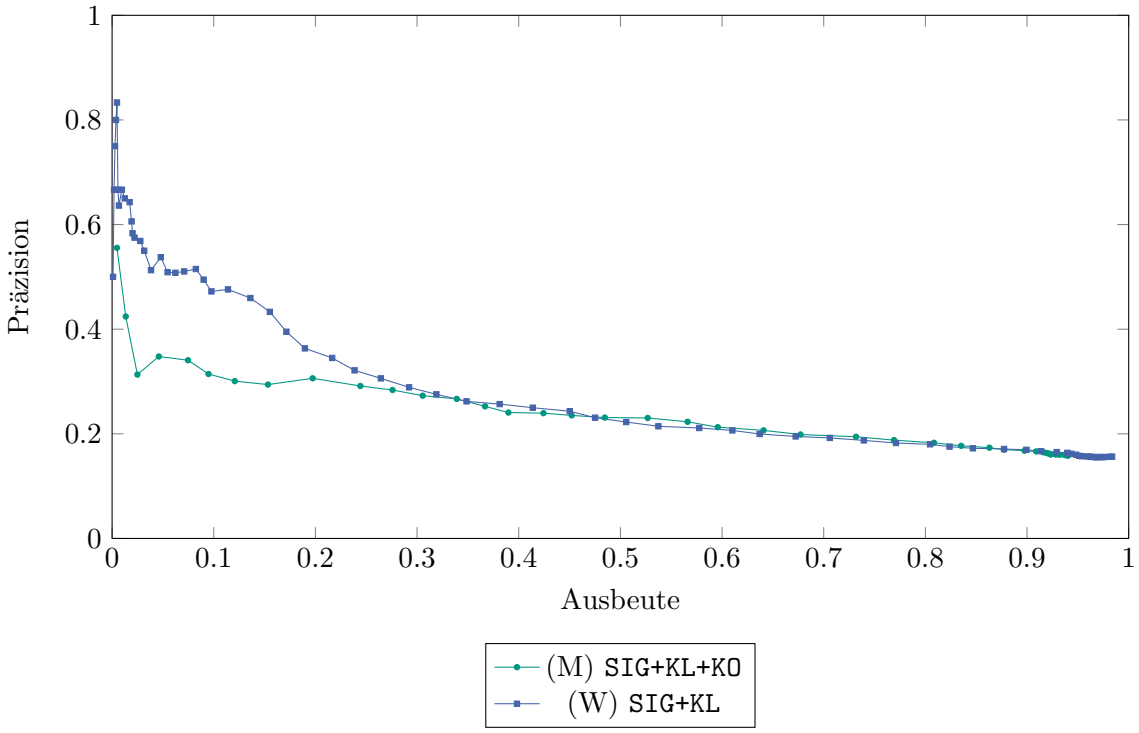


Abbildung 6.20: SMOS: Ausbeute-Präzisionskurven für die beiden Verfahren mit dem höchsten F1-Wert für SMOS

Wörter, eine Klasse als Menge seiner Methodenbezeichner und Klassennamen repräsentiert. Der F1-Wert der Variante mit Kosinusähnlichkeit ist um 0,5% höher. Bei Betrachtung der Ausbeute-/Präzisionskurven ist jedoch zu sehen, dass das WMD-Vorgehen meistens bessere Ergebnisse erzielt. An der Stelle mit Ausbeute = 52,7%, wo der F1-Wert berechnet wurde, liegt die Kurve von (M) `SIG+KL+KO` kurzzeitig höher. Es handelt sich hier also um einen Ausreißer.

Im Vergleich zur naiven Vorgehensweise, die bereits einen F1-Wert von 27,4% erreicht, liegt der F1-Wert von 32,1% nur um ca. 5% höher, was viel niedriger als bei iTrust, eTour oder LibEST ist. Dazu tragen mehrere Ursachen bei. SMOS besitzt 60 Klassen, die alle von der Oberklasse `HttpServlet` erben und jeweils die gleichen beiden Methoden überschreiben. Genau wie bei iTrust erschwert dies die Differenzierung zwischen diesen Klassen, da die Methodensignaturen gleich sind. Darüber hinaus ist SMOS zweisprachig. Deswegen wurden zwar zwei Sprachmodelle verwendet, die im gleichen Vektorraummodell ausgerichtet sind, jedoch ist die Ausrichtung bezüglich der Kosinusähnlichkeit nicht perfekt. Probeweise wurden zwischen einigen englischen Wörtern und ihre Übersetzung (bzw. zwischen ihren Einbettungen) die Kosinusähnlichkeit berechnet. Die Ergebnisse davon sind in Tabelle 6.23 aufgelistet. Bei den Probewörtern schwanken die Kosinusähnlichkeiten zwischen ca. 0,4 bis 0,7. Eigentlich sollten die Ähnlichkeit zwischen den Wörtern und ihren Übersetzungen in der Nähe von Eins liegen, da dies der Optimalfall darstellt. Dass die Ähnlichkeiten bereits zwischen korrekten Übersetzungswortpaaren nicht hoch ist, erschwert zusätzlich die Rückverfolgbarkeit von zum Beispiel englischen Bezeichnern und italienischen Anforderungen.

Aus den einzelnen Evaluationen der Projekte hat sich bisher ergeben, dass der beste F1 für eTour bei 51,8%, für iTrust bei 26,1% und für LibEST bei 60,1% liegt. eTour hat hilfreiche Überschriften in den Anforderungsfällen, die der Rückverfolgbarkeit helfen. Bei iTrust verursacht das Erben und das Implementieren der gleichen Klassen und Schnittstellen gleiche Methodensignaturen in den Unterklassen, was die Rückverfolgbarkeit erschwert. Die hohen Werte bei LibEST werden auch durch die hohe Anzahl an Musterlösungsverbindungen begünstigt. Dronology wird wegen der Unvollständigkeit nicht genauer evaluiert. SMOS wird außer für den Vergleich mit anderen Arbeiten in Abschnitt 6.5 ebenfalls nicht weiter evaluiert, da die Abbildung von mehrsprachigen Artefakten nicht Fokus dieser Arbeit ist.

6.4.6 Projektübergreifender Vergleich

In den bisherigen Evaluationen wurden die Testprojekte separat ausgewertet und nach den besten Verfahren gesucht. In diesem Abschnitt wird ein Verfahren ermittelt, das projektübergreifend am besten funktioniert. Als Projekte wurden dabei eTour, iTrust und LibEST miteinbezogen.

Zur Bestimmung des besten projektübergreifenden Verfahrens wurden die verschiedenen vorgestellten Verfahren auf alle drei Projekte angewendet, wobei alle Schwellwerte zwischen Null und Eins aus Performanzgründen in 0,05er statt 0,01er Schritten variiert werden. Bei drei verschachtelten Schwellwerten entstehen dabei 8000 mögliche Schwellwertkombinationen, an denen für alle drei Projekte jeweils der F1-Wert berechnet wird. Unter diesen F1-Werten werden für die drei Projekte separat der höchste F1-Wert ermittelt, welche anschließend gemittelt werden. Das Verfahren mit dem höchsten F1-Durchschnitt wird als bestes projektübergreifendes Verfahren festgelegt. Das Ergebnis hiervon ist in Tabelle 6.25

Tabelle 6.24: Zusammensetzung der besten Verfahren gemessen am F1-Wert

Verfahren	F1	D-SW	M-SW	E-SW	SB	AR	QR
`MS+AS`	**42,5%**				Top	Dur	Min
eTour	51,4%	0,65	0,4	0,55			
iTrust	20,3%	0,5	0,25	0,85			
LibEST	54,6%	0,65	1	0,8			
`MK+AS`	42,3%				Top	Dur	Min
eTour	49%	0,65	0,25	0,65			
iTrust	20%	0,55	0.4	0,75			
LibEST	57,6%	0,6	1	0,85			
`MK+A`	42,2%				Top	-	Min
eTour	47,6%	0,65	0,15	0,55			
iTrust	23,4%	0,7	1	0,5			
LibEST	55,6%	0,7	1	0,65			

Tabelle 6.25: Zusammensetzung der drei besten Konfiguration gemessen am F1-Wert

Verf.	D. F1	eTour	iTrust	LibEST	D-SW	M-SW	E-SW	SB	AR	QR
`MK+AS`	**39,4%**	39,4%	22,9%	55,9%	0,6	0,6	0,6	SW	Min	Min
`MK+AS`	**39,4%**	39,4%	22,9%	55,9%	0,6	1	0,6	Top	Min	Min
`MS+AS`	39,1%	41,5%	18,3%	57,3%	0,65	0,95	0,6	Top	Min	Dur

dargestellt. MS+AS hat sich als bestes projektübergreifende Verfahren herausgestellt. Es erreicht einen F1-Durchschnitt von 42,5%, was in der ersten Zeile zu sehen ist. Die einzelnen F1-Werte der Projekte, aus denen sich der Durchschnitt berechnet, sind ebenfalls in der Tabelle in den drei darunter liegenden Zeilen aufgelistet. Außerdem sind pro Projekt die Schwellwerte angegeben, an denen der jeweilige F1-Wert berechnet wurde. Die SB-Spalte zeigt die verwendete Stimmenbestimmung an, also Mehrheitsentscheidschwellwert (SW) oder Top-N. Im Falle von Top-N bezeichnet der Wert in der M-SW-Spalte den prozentualen Anteil an der Gesamtzahl von Anforderungen, aus dem sich die absolute Anzahl der Stimmen pro Klassenelement berechnet. AR und QR stellen die Ähnlichkeitsreduktionsfunktionen dar, die für die Anforderungs- bzw. Quelltextseite verwendet wurden. Dur ist der Durchschnitt, Min ist das Minimum. Pro Verfahren bleiben die Stimmenbestimmung und die Reduktionsfunktionen für die drei Projekte gleich.

Bei den Verfahren in Tabelle 6.25 handelt es sich um Verfahren mit WMD als Ähnlichkeitsmetrik. Das war zu erwarten, da eTour und LibEST in ihrer separaten Evaluation bereits mit WMD-Verfahren die besten Ergebnisse erzielt haben. Es ist überraschend, dass MS+AS besser abschneidet als MK+AS, obwohl sich Kommentare eigentlich als hilfreich herausgestellt haben. Jedoch wurde in Abschnitt 6.4.1 festgestellt, dass eTour viele nicht hilfreiche Kommentare besitzt. Bei Begutachtung der separaten F1-Werte ist daher auch zu sehen, dass eTour bei MS+AS mit einem besseren Wert zum Durchschnitt miteingerechnet wurde als bei MK+AS. Bei LibEST ist es umgekehrt, da das beste LibEST-Verfahren Kommentare miteinbezieht. Allgemein unterscheiden sich die durchschnittlichen F1-Werte in Tabelle 6.25 nur um 0,2 - 0,3%. Das ist zu wenig, um eindeutig ein überlegeneres Verfahren unter diesen drei zu bestimmen. Welches Verfahren am besten ist, hängt von der Nützlichkeit der Klassenelemente ab.

Auffällig sind die Werte in der M-SW-Spalte für LibEst und iTrust: Sie beträgt in einigen Konfigurationen Eins, das heißt es werden alle verfügbaren Stimmen pro Klassenelement zugelassen. Dass hierdurch bzw. durch Top-N-Verfahren die besten F1-Werte erreicht werden, spricht nochmals dafür, dass der Mehrheitsentscheid mit einem Mehrheitsentscheidschwellwert möglicherweise zu restriktiv ist, wie bereits in Abschnitt 6.3.3.1 diskutiert wurde.

Neben dem besten Verfahren wurde auch die beste projektübergreifende Konfiguration bestimmt, das heißt ein Verfahren mit einer Schwellwertkombination, die auf allen drei Projekten am besten funktioniert. Dazu werden wie oben alle Schwellwerte in 0,05er Schritten abgetastet. Pro Schwellwertkombination eines Verfahrens wird für alle drei Projekte der F1-Wert bestimmt und daraus ein schwellwertspezifischer Durchschnitt gebildet. Das Verfahren mit dem maximalen schwellwertspezifischen F1-Durchschnitt, also die beste projektübergreifende Konfiguration, ist in Tabelle 6.24 gezeigt. Es handelt sich wieder um ein WMD-Verfahren. Zwei Konfigurationen von MK+AS erreichen den besten durchschnittlichen F1-Wert von 39,4%. Bei Betrachtung der projektspezifischen F1-Werte und der Schwellwerte ist festzustellen, dass sie fast gleich sind. Der einzige Unterschied liegt in der Stimmenbestimmung und im M-SW-Wert, der dadurch eine jeweils andere Bedeutung besitzt. Die gleiche Auswertung ist darauf zurückzuführen, dass in beiden Fällen ein Elementschwellwert von 0,6 und die gleichen Reduktionsfunktionen eingesetzt wurden. Dadurch stehen bei der Stimmenbestimmung die gleichen Kandidaten zur Auswahl. Der Mehrheitsentscheidschwellwert beträgt 0,6, also werden hier alle Kandidaten als Stimmen zugelassen. Allerdings werden auch bei Top-N 100% der Kandidaten zugelassen. Daher verläuft die Evaluation in beiden Fällen gleich.

Wie beim besten Verfahren liegen die F1-Werte von MS+AS und MK+AS nah beieinander. Auch hier kann also keine eindeutig bessere Konfiguration festgelegt werden.

6.5 Vergleich mit anderen Arbeiten

In diesem Abschnitt wird ein Vergleich zur bestehenden Arbeit von Moran et al. [MPB+20] durchgeführt. Dort wurde die Anforderung-zu-Quelltextrückverfolgbarkeit mit Techniken aus der Informationsrückgewinnung umgesetzt. Die Arbeit stützt sich auf die Grundannahme, dass die Existenz einer Rückverfolgbarkeitsverbindung wahrscheinlichkeitsbasiert ist. Die Begründung liegt darin, dass ein Softwareentwicklungsprozess durch zufällige Faktoren randomisiert wird. Zu diesen Faktoren gehören zum Beispiel Schreibstil der Autoren, die schwer vorhersagbar sind und daher als zufällig angesehen werden können. Für die Rückverfolgbarkeit nutzen die Autoren mehrere verschiedene Vorgehen aus der Informationsrückgewinnung, wie etwa dem Vektorraummodell (Abschnitt 2.6.3), mit deren Hilfe jeweils eine Ähnlichkeit zwischen Anforderungs- und Quelltextpaaren berechnet werden können. Die Ähnlichkeitswerte werden dazu genutzt, die Parameter einer Wahrscheinlichkeitsverteilung zu bestimmen, welche die Wahrscheinlichkeit für die Existenz einer Rückverfolgbarkeitsverbindung zwischen zwei Artefakten angibt. Zur Rückverfolgbarkeit werden hier also mehrere Ähnlichkeitsmetriken aus der IR kombiniert genutzt.

Die Autoren haben darüber hinaus weitere Informationen wie Entwicklerfeedback in ihr Wahrscheinlichkeitsmodell integriert. Für den Vergleich mit Einbettungen sind jedoch nur die IR-Techniken interessant. In der Arbeit von Moran et al. wurde auch eine Evaluation durchgeführt, die nur die angesprochenen kombinierten IR-Verfahren nutzt und als „Stage 1“ bezeichnet wird. Außerdem haben sie die verwendeten IR-Techniken einzeln auf die Testprojekte angewendet, um es mit ihrem kombinierten und probabilistischen Ansatz zu vergleichen. Im Folgenden werden diese und die Stage-1-Ergebnisse mit den Resultaten der Einbettungsverfahren verglichen.

In Tabelle 6.26 sind die Ergebnisse aus der Arbeit von Moran et al. (links) und die Auswertung der Einbettungsverfahren (rechts) dazu eingetragen. Die Prozentwerte sind die besten durchschnittliche Präzisionen, die auf dem jeweiligen Projekt erzielt wurden. „Best Base“ gibt die beste durchschnittliche Präzision an, die mit einem einzelnen IR-Verfahren erreicht wurde. „Med. Base“ zeigt den Median der durchschnittlichen Präzisionen der Verfahren, die eine einzelne IR-Technik anwenden. In „Stage 1“ finden sich die durchschnittlichen Präzisionen durch die kombinierte Vorgehensweise von Moran et al. Die „EB“-Spalte listet die beste durchschnittliche Präzision mit Einbettungen auf; das dazugehörige Verfahren und die Schwellwerte sind anschließend gezeigt. Bei mehreren Schwellwerten wurde bei der Berechnung der durchschnittlichen Präzision nur der letzte Schwellwert (D-SW) variiert. Für die Einbettungsseite basieren die Verfahren bei allen Projekten außer iTrust auf der WMD. Bei LibEST und eTour sind es Mehrheitsentscheidverfahren, die den Mehrheitsentscheidschwellwert nutzen. `MK+AS` nutzt das Minimum und `MS+AS` den Durchschnitt für beide Reduktionsfunktionen. `SIG+KL+KO+KKO` und (W) `SIG+KL` sind Verfahren auf der Klassenebene und besitzen daher keine Einträge in den letzten beiden Spalten. (W) `SIG+KL` berechnet die WMD zwischen ganzen Anforderungen und Klassen, wobei Klassen als Menge ihrer Klassen- und Methodennamen repräsentiert werden.

An den Werten in Tabelle 6.26 ist zu sehen, dass meistens ein IR-Verfahren das beste Ergebnis erzielt. Hier muss jedoch beachtet werden, dass die IR-Verfahren optimal auf den jeweiligen Testprojekten mit Hilfe der Musterlösung konfiguriert wurden. Dieser Optimalfall wird in der Praxis selten auftreten, da meistens keine Musterlösung vorhanden ist. Das kombinierte Verfahren von Moran et al. und die Einbettungsverfahren benötigen keine Musterlösung und können direkt angewendet werden, das heißt es wird verglichen, wie nahe die Ergebnisse dem Optimalfall kommen. Bezüglich iTrust und SMOS liegen die Ergebnisse 2-3% niedriger als der Optimalfall, aber trotzdem über den Median der IR-Techniken. Bei eTour übertrifft ein Einbettungsverfahren die anderen Techniken. Nur bei LibEST liegt das Resultat unter dem Median der IR-Verfahren.

Tabelle 6.26: Vergleich der durchschnittlichen Präzisionen aus [MPB+20] mit Einbettungsverfahren

Projekt	Best Base	Med. Base	Stage 1	EB	Verfahren	M-SW	E-SW
LibEST	**69%**	55%	63%	51%	`MK+AS`	0,66	1
eTour	40%	30%	38%	**42%**	`MS+AS`	0,64	0,65
SMOS	**29%**	25%	25%	26,7%	(W) `SIG+KL`	-	-
iTrust	**17%**	13%	**17%**	15,4%	`SIG+KL+KO+KKO`	-	-

Hierzu müssen einige Anmerkungen gemacht werden. Die durchschnittliche Präzision entspricht dem Flächeninhalt unter der Kurve, die aus Ausbeute-Präzisionswertepaaren besteht. Sowohl bei den IR-Verfahren als auch bei Moran et al. existiert nur ein einzelner Schwellwert, der zur Berechnung der Ausbeute-Präzisionswertepaaren bzw. der durchschnittlichen Präzision variiert wird. Die durchschnittliche Präzisionen, die in Tabelle 6.26 für eTour und LibEST berechnet wurden, stammen von Verfahren mit drei Schwellwerten, wobei nur der Letzte variiert wurde. Dadurch kommt es vor, dass die Ausbeute-Präzisionskurven nie eine Ausbeute von 100% erreichen, da durch die ersten Schwellwerte bereits Verbindungen herausgefiltert wurden. Dies ist zum Beispiel an den Kurven des Mehrheitsentscheids in Abschnitt 6.3.3.1 zu beobachten. Aus diesem Grund sind die durchschnittlichen Präzisionen niedriger.
Es kann argumentiert werden, dass ein Vergleich trotz der unterschiedlichen Berechnungen valide ist, da für die praktische Anwendung des Einbettungsverfahrens aus Komplexitätsgründen nur der letzte Schwellwert als konfigurierbar festgelegt wird, während die anderen Schwellwerte konstant bleiben. In diesem Fall würde die durchschnittliche Präzision, bei der nur der letzte Schwellwert variiert wird, korrekterweise die praktische Leistung des Verfahrens bemessen.

Für die theoretische Leistung des Verfahren muss über alle Schwellwerte variiert werden, um alle Schwellwertkonfigurationen zu erhalten. Daher wird eine weitere Evaluation durchgeführt, in der alle Schwellwerte variiert werden. Für die Berechnung der durchschnittlichen Präzision wurden die Schwellwerte bisher in 0,01er Schritten zwischen Null und Eins geändert. Die Komplexität zur Berechnung der durchschnittlichen Präzision für ein Verfahren mit drei Schwellwerten beträgt also $100 \cdot 100 \cdot 100 \cdot$ Anzahl Artefaktverbindungen. Aus Performanz- und Zeitgründen konnte die Evaluation in dieser Form bei Verfahren mit mehreren Schwellwerten nicht durchgeführt werden. Eine Option für die Komplexitätssenkung besteht in der Vergrößerung der Schrittgröße auf 0,05. Hier besteht jedoch der Nachteil, dass Schwellwertkonfigurationen mit den besten F1-Werten (welche wahrscheinlich bessere Präzisionswerte besitzen) übersprungen werden. Beispielsweise wurde der beste F1-Wert von LibEST in bei einem M-SW von 0,66 gemessen. Aus diesem Grund wurde eine heuristische Vorgehensweise gewählt: Der letzte Schwellwert wird in 0,01er Schritten variiert. Die anderen Schwellwerte werden nur an der Stelle, an der der beste F1-Wert aufgetreten ist sowie an der Stelle Null ausgewertet.

Tabelle 6.27: Vergleich der durchschnittlichen Präzisionen mit unterschiedlichen Schrittgrößen

Projekt	Verfahren	L	0,05	H	0,01
LibEST	`MK+A`	35,2%	52,8%	54,8%	56,3%
eTour	`MS+AS`	37,2%	41,8%	43,8%	-
iTrust	(M) `SIG+KL+KO+KKO+R`	14,2%	10,0%	14,7%	-
SMOS	(M) `SIG+KL+KO`	22,7%	21,2%	23,2%	-

Beispiel 6.1: Heuristische Zusammensetzung von Schwellwerten für die durchschnittliche Präzision

Für ein Verfahren wird mit einem Projekt der beste F1-Wert an der Stelle `D-SW = 0,5 / M-SW = 0,66 / E-SW = 0,34` erreicht.
Die Schwellwertkombinationen, die für die Berechnung der durchschnittlichen Präzisionen verwendet werden, setzen sich aus den Folgenden zusammen:

```
D-SW = [0, 0.01, ... 1]
M-SW = [0, 0.66]
E-SW = [0, 0.34]
```

Beispiel 6.1 demonstriert die resultierenden Schwellwerte an einem Beispiel. Die Präzisionswerte der besten Konfiguration werden für die durchschnittliche Präzision berücksichtigt, da die Kombination `D-SW = 0,5 / M-SW = 0,66 / E-SW = 0,34` enthalten ist. Es kann jedoch sein, dass bei dieser Kombination nie 100% Ausbeute erreicht wird, wodurch die Präzisionswerte niedriger sind. Daher ist auch die Kombination `D-SW = 0 / M-SW = 0 / E-SW = 0` enthalten, an dem keine Rückverfolgbarkeitsverbindungen herausgefiltert und daher meistens 100% Ausbeute erreicht wird.

Mit dieser Methode konnten für eTour und LibEST noch bessere durchschnittliche Präzisionen gefunden werden, die in Tabelle 6.27 zu sehen sind. Die L-Spalte zeigt die durchschnittliche Präzision, wenn nur der letzte Schwellwert an der Stelle mit dem besten F1-Wert variiert wird. In der nächsten Spalte wurden alle Schwellwerte in 0,05er Schritten geändert. Die Ergebnisse mit der Heuristik sind in der H-Spalte zu finden. Die letzte Spalte zeigt die durchschnittliche Präzision, wenn alle Schwellwerte in 0,01er Schritten variiert werden. Dies wurde zum Vergleich probeweise für das vorliegende Verfahren mit LibEST durchgeführt. Das Verfahren nutzt nur zwei Schwellwerte und LibEST besitzt die wenigsten Artefakte unter den Testprojekten, was die Berechnung vereinfacht hat. An den Werten in der Tabelle ist zu beobachten, dass die Heuristik etwas besser abschneidet als die groben 0,05er Schritte oder die Auswertung nur am letzten Schwellwert. Mit einer durchschnittlichen Präzision von 56,3% liegt LibEST nun auch über dem Median der IR-Techniken. Für iTrust und SMOS wurden keine besseren Werte durch die Heuristik ermittelt als die Werte aus Tabelle 6.26 (welche allerdings auf Verfahren basieren, die nur einen einzigen Schwellwert besitzen). Dennoch sind in Tabelle 6.27 beispielhaft die Auswertung für zwei Mehrheitsentscheidverfahren mit Kosinusähnlichkeit für iTrust und SMOS dargestellt. An diesen Werten ist zu sehen, dass die 0,05er Schritte zu grobgranular sind und daher schlechtere Resultate liefern als die Auswertung am besten F1-Wert. Die heuristische Vorgehensweise erzielt aber auch hier die besten durchschnittlichen Präzisionen.

Die F1-Werte der Verfahren aus [MPB+20] standen nicht zu Verfügung, daher wurden

Tabelle 6.28: Zusammensetzung der Evaluationsdatensätze, die in dieser Arbeit verwendet wurden (links), im Vergleich zu [MPB+20] (rechts).

Projekt	Anf.	Quell.	Verb.	Anf.	Quell.	Verb.
eTour	58	113	362	58	116	308
iTrust	131	226	286	131	367	399
SMOS	67	98	1044	67	100	1044
LibEST	52	14	204	59	11	204

die Ausbeute-Präzisionskurven der besten Verfahren grafisch in Abbildung 6.21, Abbildung 6.22, Abbildung 6.23, Abbildung 6.24, Abbildung 6.25 und Abbildung 6.26 gegenübergestellt. Die Einbettungsverfahren sind die gleichen, die bereits in Abschnitt 6.4 vorgestellt wurden.

In den Abbildungen aus der Arbeit von Moran et al. sind NUTS, MAP und VI die Verfahren, mit denen mehrere IR-Techniken kombiniert werden. Die grauen gestrichelten Kurven entsprechen dem besten bzw. schlechtesten einzelnen IR-Verfahren bezüglich der durchschnittlichen Präzision und die graue durchgezogene Linie gehört zum Verfahren, der den Median der durchschnittlichen Präzision darstellt.
Die Kurven mit Einbettungsverfahren verlaufen bei eTour und iTrust gleich zur besten IR-Kurve oder liegen leicht darüber. Nur bei LibEST verläuft die Kurve für die Einbettungen bei höheren Schwellwerten niedriger. Bereits bei der durchschnittlichen Präzision in Tabelle 6.26 schnitt LibEST im Vergleich zu den IR-Techniken deutlich schlechter ab als bei den anderen Projekten. Eine Ursache kann darin liegen, dass in dieser Arbeit für WMD-Verfahren nur Methodensignaturen und - Kommentare berücksichtigt wurden, die Inklusion von Methodenrümpfen wurde nicht implementiert. In LibEST existieren viele Methoden mit großen Methodenrümpfen. Es könnte zu einer Verbesserung führen, wenn diese auch miteinbezogen werden.

Im Folgenden werden einige Punkte zur Validität des Vergleichs zwischen den Arbeiten diskutiert. Die durchschnittliche Präzision ist eigentlich keine optimale Metrik, um zwischen Arbeiten zu vergleichen. Zum einen besteht die Problematik beim Vergleich zwischen Arbeiten mit unterschiedlich vielen Schwellwerten, die bereits angesprochen wurde. Zum anderen hängt die durchschnittliche Präzision von der Anzahl der Abtastungen ab. Wie in Tabelle 6.27 zu sehen ist, können sich die Werte stark ändern, je nachdem, ob nur der letzte Schwellwert oder alle Schwellwerte in 0,05er Schritten oder in 0,01er Schritten variiert werden. Das liegt daran, dass die Abtastung an diskreten Werten nur eine Annäherung an das Integral der Kurve ist. Moran et al. haben nicht angegeben, welche Schrittgröße verwendet wurde. Der Vergleich kann also verfälscht sein, falls in einer Arbeit viel fein- oder grobgranularer vorgegangen wurde.

Des Weiteren unterscheiden sich die genauen Zusammensetzungen der Evaluationsdatensätze, wie in Tabelle 6.28 gezeigt ist. Auf der linken Seite ist die Zusammensetzung dieser Arbeit und auf der rechten die von Moran et al. In Abschnitt 6.1 wurde erklärt, dass einige Quelltextartefakte bei eTour und SMOS entfernt wurden, da sie entweder fehlerhaften Quelltext oder überhaupt keinen Quelltext enthalten. Für IR-Techniken wird Quelltext als Fließtext aufgefasst, daher ist ein intakter Quelltext nicht notwendig. Obwohl dadurch für die Arbeit von Moran et al. mehr Artefakte für eTour zur Verfügung standen, beinhaltet die Musterlösung weniger Verbindungen. Warum dort Verbindungen fehlen, wird nicht erklärt. Bei iTrust haben Moran et al. viel mehr Quelltextartefakte und Musterlösungsverbindungen. Das liegt daran, dass sie durch die IR-Techniken auch JavaServer Pages verarbeiten können, die im iTrust-Datensatz enthalten sind. Für Einbettungsverfahren

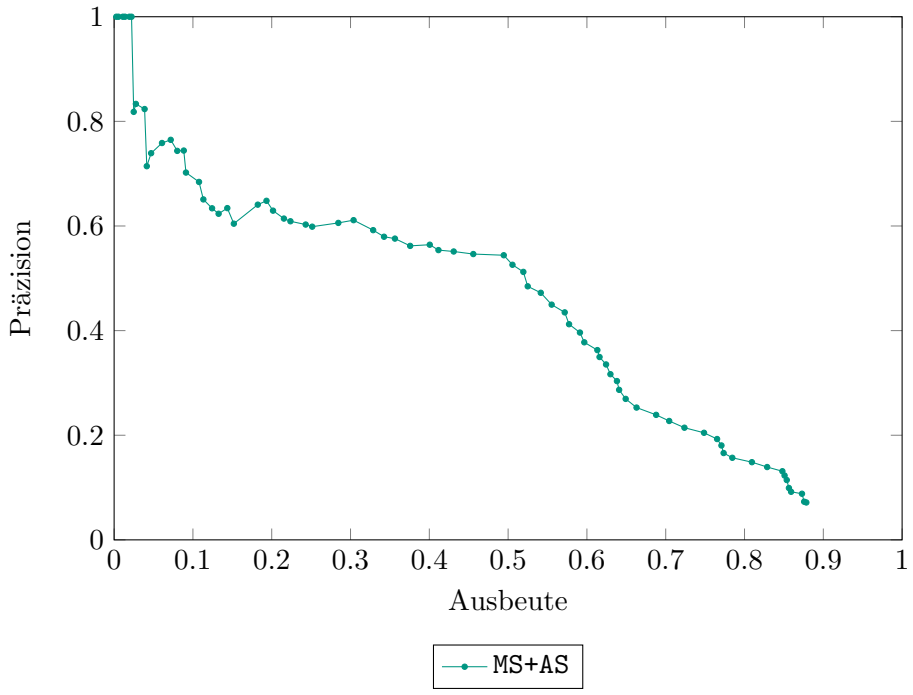


Abbildung 6.21: eTour: Verfahren mit dem besten F1-Wert

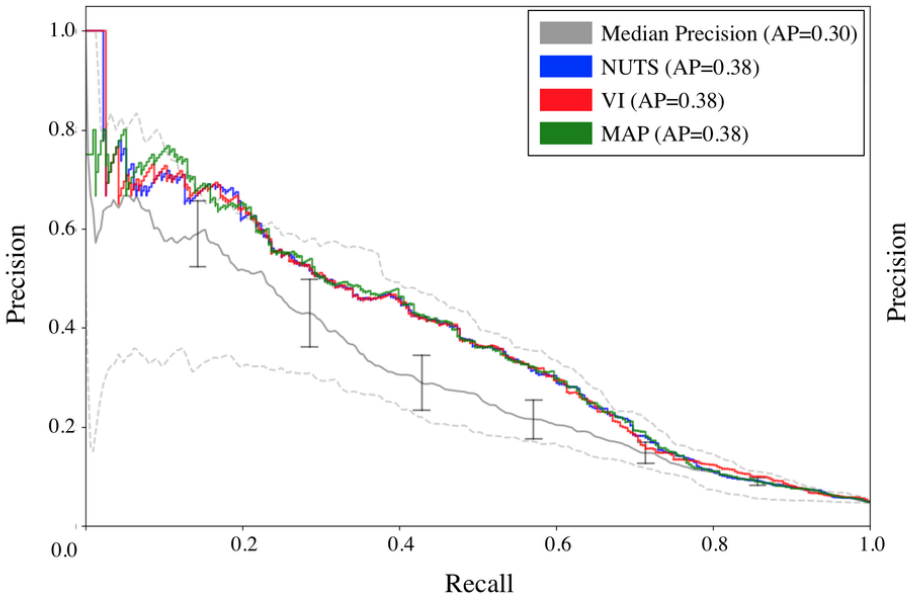


Abbildung 6.22: Ergebnisse für eTour aus [MPB+20]

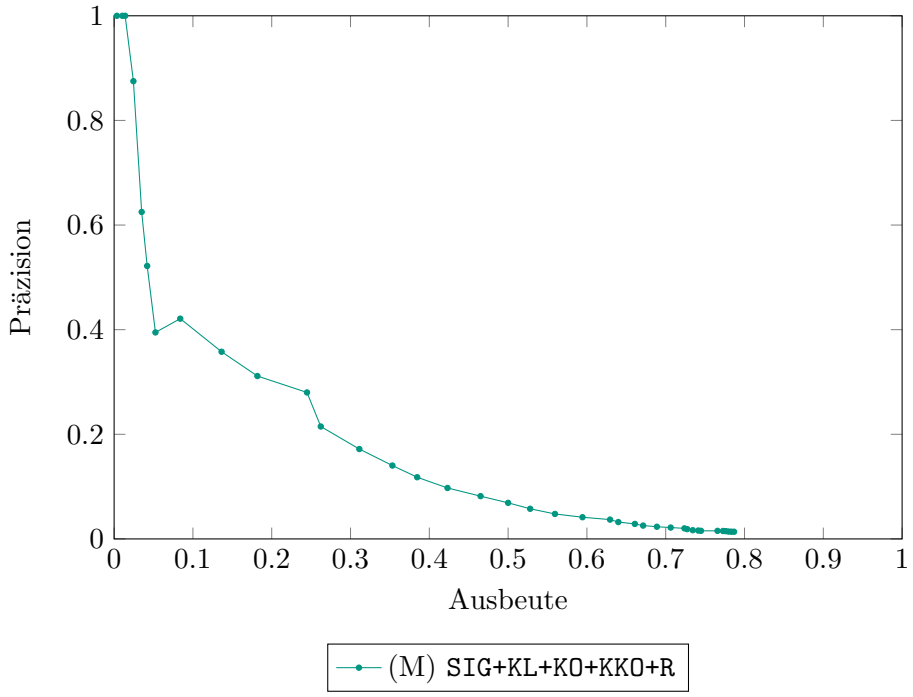
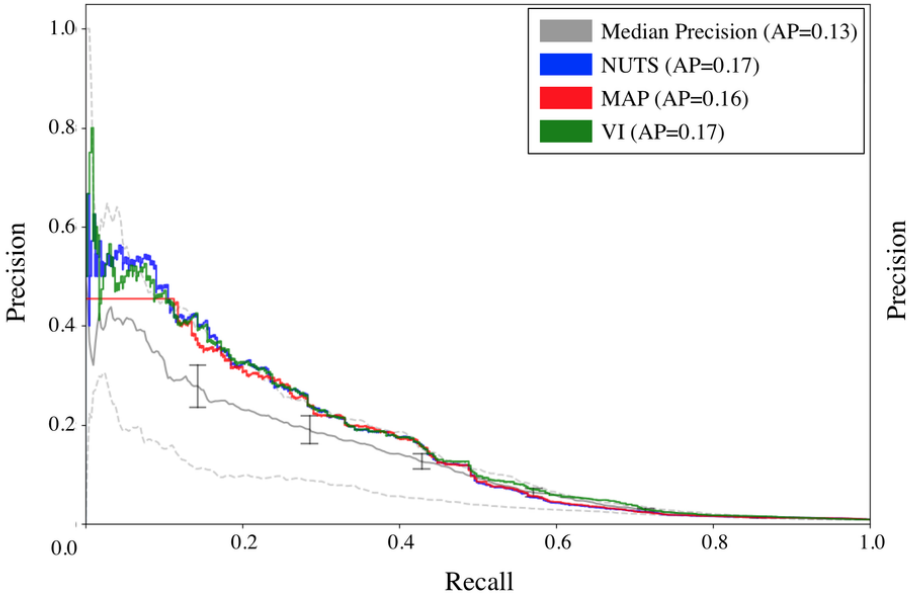


Abbildung 6.23: iTrust: Verfahren mit dem besten F1-Wert



P/R Curve for iTrust: Rq → Src

Abbildung 6.24: Ergebnisse für iTrust aus [MPB+20]

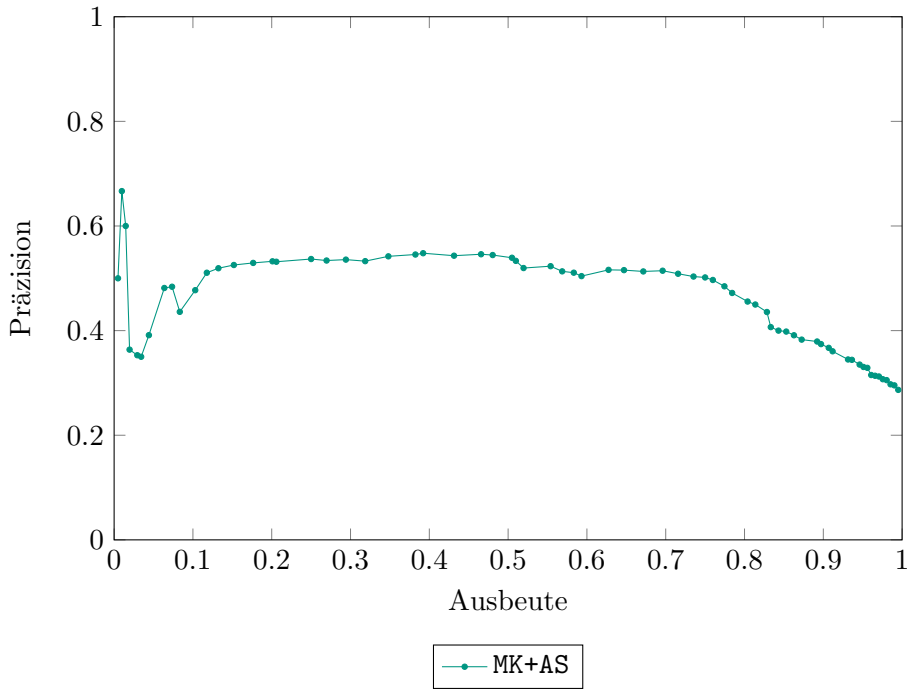


Abbildung 6.25: LibEST: Verfahren mit dem besten F1-Wert

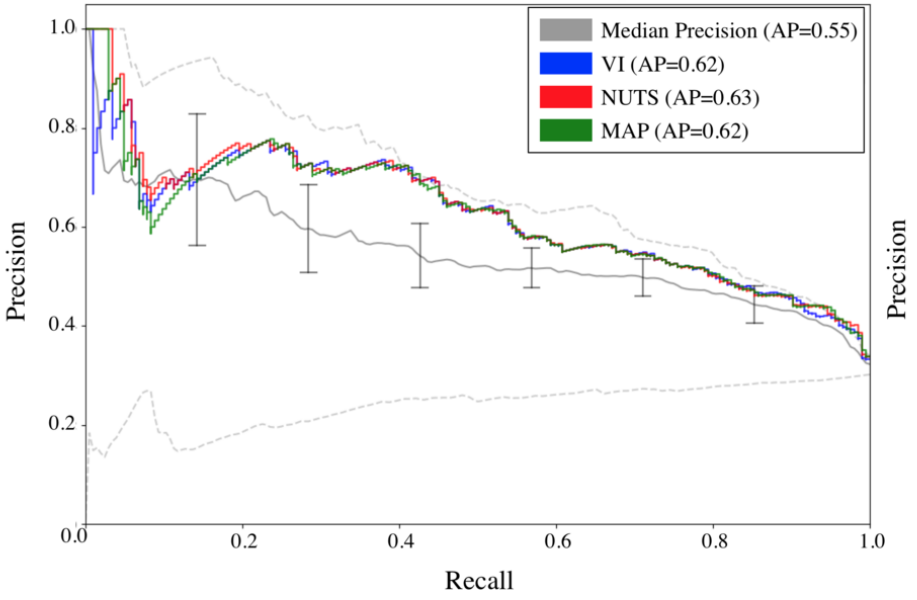


Abbildung 6.26: Ergebnisse für LibEST aus [MPB+20]

sind diese Dateien wie in Abschnitt 6.1 erläutert nicht geeignet und wurden weggelassen. Als Letztes fehlen bei Moran et al. bei LibEST drei Quelltextdateien, während für diese Arbeit sieben Anforderungsartefakte absent sind. Der LibEST-Datensatz wurde für diese Arbeit aus dem verknüpften Depot von Moran et al. bezogen und es wurden alle vorhandenen Artefakte verwendet. Warum die Zahlen in der Arbeit von Moran et al. abweichen, wird nicht erklärt.

Insgesamt übertreffen die besten durchschnittlichen Präzisionen dieser Arbeit die Werte von den Median-IR-Techniken. Außer bei LibEST weichen die Werte von den besten IR-Werten um 2-3% ab. Jedoch wurden auch bezüglich der Datensatzzusammenstellung Unterschiede festgestellt, sodass vor allem bei iTrust die Ergebnisse nicht unbedingt vergleichbar sind.

7 Zusammenfassung und Ausblick

In dieser Arbeit sollte die Rückverfolgbarkeit zwischen Anforderungen und Quelltext mit Hilfe von Einbettungen umgesetzt werden. Dazu mussten Verfahren zur Einbettung von Quelltext und Anforderungen sowie zur Abbildung zwischen ihnen bzw. ihrer Einbettungen entworfen werden. Existierende Einbettungsverfahren können einzelne Wörter oder Sätze einbetten. Diese wurden genutzt, um die Wörter bzw. Sätze in Anforderungen oder in den Bezeichnern und Kommentaren im Quelltext als Einbettungen zu repräsentieren. Pro Artefakt werden anschließend die Wort- oder Satzeinbettungen aggregiert, um Anforderungs- und Quelltexteinbettungen zu kreieren. Hierbei kann variiert werden, welche Anforderungswörter oder Quelltextelemente für die Aggregation miteinbezogen werden. Wie auch in der Evaluation bestätigt wurde, ist insbesondere für die Quelltextseite wichtig, eine richtige Auswahl an Elementen zu treffen, da nicht alle Quelltextelemente nützliche Informationen für die Rückverfolgbarkeit beinhalten. Als besonders hilfreiche Klassenelemente haben sich Klassennamen, Methodensignaturen und -kommentare herausgestellt. Weiterhin wurden auch Aufrufbeziehungen zwischen Methoden betrachtet. Aus der Evaluation geht hervor, dass auch sie die Rückverfolgbarkeit unterstützen.
Für die Abbildung zwischen Anforderungs- und Quelltexteinbettungen wurde Varianten auf der Klassenebene und Varianten durch die Ableitung aus der Elementebene mit einem Mehrheitsentscheid gegenübergestellt. Hierbei hat sich ergeben, dass der Mehrheitsentscheid bezüglich des F1-Wertes bei den meisten Kombinationen von Klassenelementen besser funktioniert. Durch den Mehrheitsentscheid können für die Rückverfolgbarkeit unwichtige Elemente überstimmt und ausgelassen werden. Weiterhin wurden ebenfalls verschiedene Ähnlichkeitsmetriken untersucht. Die Word Mover's Distance funktioniert dabei bei den meisten Projekten am besten. Dies ist dadurch zu erklären, dass bei der WMD keine Informationen durch eine Aggregation verloren gehen und die minimale Distanz zwischen zwei Wortmengen bzw. Artefakten bestimmt wird.
In der Evaluation erreichen die besten Verfahren mit LibEST einen F1-Wert von 60,1%, mit iTrust einen Wert von 26,1%, und mit eTour einen Wert von 51,8%.
Die beste Konfiguration über alle drei Projekte hinweg wurde ebenfalls ermittelt und erreicht einen durchschnittlichen F1-Wert von 39,4%. Beim Vergleich mit den Werten aus der Arbeit von Moran et al. [MPB+20] liegt die durchschnittlichen Präzision bei allen Verfahren über dem Median der optimal konfigurierten IR-Verfahren. Bei eTour und SMOS wurden bessere Werte gemessen als die kombinierte IR-Variante von Moran et al. Jedoch wurden auch Mängel an der durchschnittlichen Präzision und an der Zusammensetzung der Datensätze angesprochen, die die Ergebnisse verfälschen können.

In zukünftigen Arbeiten können noch weitere Kombinationsmöglichkeiten der Klassen- bzw. Anforderungselemente und Ähnlichkeitsmetriken untersucht werden. In dieser Arbeit wurde eher in der Breite nach Elementen und Techniken gesucht, die hilfreich für die Rückverfolgbarkeit sind. Es wurden nicht alle Kombinationen der betrachteten Techniken ausgewertet. Beispielsweise wurden für die WMD-Verfahren nur Methodensignaturen und -kommentare betrachtet. Hier können noch weitere Elemente hinzugenommen werden. Auch wäre es möglich, WMD-Verfahren mit Aufrufabhängigkeiten zu erweitern, da sich diese auch als hilfreich herausgestellt haben. Darüber hinaus wurden in der Analyse einige Techniken angesprochen, die nicht umgesetzt wurden. Dazu gehören Datenflussabhängigkeiten oder die Einbettung mit einem Dokumenteneinbettungsverfahren. Außerdem wurde zur Aggregation in dieser Arbeit fast immer der Durchschnitt benutzt. Hier können noch andere Optionen wie die gewichtete Summe untersucht werden, um zum Beispiel Wörter nach ihrer Wortart stärker oder schwächer in die Summe miteinfließen zu lassen.
Bezüglich des Mehrheitsentscheids kann außerdem genauer untersucht werden, wie die Parameter optimal konfiguriert werden sollten, damit das Verfahren nicht zu restriktiv ist, ohne zu viel Rauschen zuzulassen.
In der vorliegenden Arbeit wurden die Artefakte auf Einbettungen abgebildet, die im gleichen Vektorraum liegen, damit direkt Ähnlichkeitsmetriken angewendet werden können. Für eine zukünftige Arbeit besteht auch die Möglichkeit, für die Artefakte verschiedene Einbettungsverfahren einzusetzen, die möglicherweise für das jeweilige Artefakt bessere Repräsentationen erzeugen. Dadurch liegen die Einbettungen jedoch in unterschiedlichen Vektorräumen. Hier muss also eine zusätzliche Abbildung zwischen Vektorräumen entworfen werden, der den Vorteil der besseren Repräsentation nicht zunichtemacht.

Literaturverzeichnis

[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 Seite 21).

[ACCL00] Antoniol ; Canfora ; Casazza ; Lucia, De: Information Retrieval Models for Recovering Traceability Links between Code and Documentation. In: *Proceedings 2000 International Conference on Software Maintenance*, 2000. – ISSN 1063–6773, S. 40–49 (zitiert auf den Seiten 20, 27 und 28).

[AJ18] Alvarez-Melis, David ; Jaakkola, Tommi: Gromov-Wasserstein Alignment of Word Embedding Spaces. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium : Association for Computational Linguistics, Oktober 2018, S. 1881–1890 (zitiert auf Seite 35).

[ALA17] Artetxe, Mikel ; Labaka, Gorka ; Agirre, Eneko: Learning Bilingual Word Embeddings with (Almost) No Bilingual Data. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada : Association for Computational Linguistics, Juli 2017, S. 451–462 (zitiert auf Seite 33).

[ALA18] Artetxe, Mikel ; Labaka, Gorka ; Agirre, Eneko: A Robust Self-Learning Method for Fully Unsupervised Cross-Lingual Mappings of Word Embeddings. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia : Association for Computational Linguistics, Juli 2018, S. 789–798 (zitiert auf Seite 35).

[AMD18] Aldarmaki, Hanan ; Mohan, Mahesh ; Diab, Mona: Unsupervised Word Mapping Using Structural Similarities in Monolingual Embeddings. In: *Transactions of the Association for Computational Linguistics* 6 (2018), August, S. 185–196. `http://dx.doi.org/10.1162/tacl_a_00014`. – DOI 10.1162/tacl_a_00014 (zitiert auf den Seiten xiii, 32 und 33).

[ANS08] Abadi, Aharon ; Nisenson, Mordechai ; Simionovici, Yahalomit: A Traceability Technique for Specifications. In: *2008 16th IEEE International Conference on Program Comprehension*, 2008. – ISSN 1092–8138, S. 103–112 (zitiert auf Seite 29).

[AS02] Alexander, Ian F. ; Stevens, Richard: *Writing Better Requirements*. Pearson Education, 2002. – ISBN 0–321–13163–0 (zitiert auf Seite 45).

[AZLY19] Alon, Uri ; Zilberstein, Meital ; Levy, Omer ; Yahav, Eran: Code2Vec: Learning Distributed Representations of Code. In: *Proc. ACM Program. Lang.* 3 (2019), Januar, Nr. POPL, S. 40:1–40:29. `http://dx.doi.org/10.1145/3290353`. – DOI 10.1145/3290353. – ISSN 2475–1421 (zitiert auf Seite 21).

[BD09] Bruegge, Bernd ; Dutoit, Allen H.: *Object-Oriented Software Engineering Using UML, Patterns, and Java*. 3rd. USA : Prentice Hall Press, 2009. – ISBN 978–0–13–606125–0 (zitiert auf Seite 48).

[BDVJ03] Bengio, Yoshua ; Ducharme, Réjean ; Vincent, Pascal ; Janvin, Christian: A Neural Probabilistic Language Model. In: *J. Mach. Learn. Res.* 3 (2003), März, S. 1137–1155. – ISSN 1532–4435 (zitiert auf Seite 15).

[BE10] Burgstaller, Benedikt ; Egyed, Alexander: Understanding Where Requirements Are Implemented. In: *2010 IEEE International Conference on Software Maintenance*, 2010. – ISSN 1063–6773, S. 1–5 (zitiert auf Seite 53).

[BGJM17] Bojanowski, Piotr ; Grave, Edouard ; Joulin, Armand ; Mikolov, Tomas: Enriching Word Vectors with Subword Information. In: *Transactions of the Association for Computational Linguistics* 5 (2017), S. 135–146. http://dx.doi.org/10.1162/tacl_a_00051. – DOI 10.1162/tacl_a_00051 (zitiert auf den Seiten 16, 78, 91 und 92).

[BJH18] Ben-Nun, Tal ; Jakobovits, Alice S. ; Hoefler, Torsten: Neural Code Comprehension: A Learnable Representation of Code Semantics. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. Montréal, Canada : Curran Associates Inc., Dezember 2018 (NIPS'18), S. 3589–3601 (zitiert auf Seite 22).

[BR99] Baeza-Yates, Ricardo A. ; Ribeiro-Neto, Berthier: *Modern Information Retrieval*. USA : Addison-Wesley Longman Publishing Co., Inc., 1999. – ISBN 978–0–201–39829–8 (zitiert auf Seite 102).

[BRA14] Borg, Markus ; Runeson, Per ; Ardö, Anders: Recovering from a Decade: A Systematic Mapping of Information Retrieval Approaches to Software Traceability. In: *Empirical Software Engineering* 19 (2014), Dezember, Nr. 6, S. 1565–1616. http://dx.doi.org/10.1007/s10664-013-9255-y. – DOI 10.1007/s10664–013–9255–y. – ISSN 1573–7616 (zitiert auf den Seiten 4 und 27).

[CKL+18] Conneau, Alexis ; Kruszewski, German ; Lample, Guillaume ; Barrault, Loïc ; Baroni, Marco: What You Can Cram into a Single Vector: Probing Sentence Embeddings for Linguistic Properties. In: *arXiv:1805.01070 [cs]* (2018), Juli (zitiert auf Seite 46).

[CKV19] Csuvik, Viktor ; Kicsi, András ; Vidács, László: Source Code Level Word Embeddings in Aiding Semantic Test-to-Code Traceability. In: *2019 IEEE/ACM 10th International Symposium on Software and Systems Traceability (SST)*, 2019. – ISSN 2157–2186, S. 29–36 (zitiert auf Seite 31).

[CLR+18] Conneau, Alexis ; Lample, Guillaume ; Ranzato, Marc'Aurelio ; Denoyer, Ludovic ; Jégou, Hervé: Word Translation Without Parallel Data. In: *arXiv:1710.04087 [cs]* (2018), Januar (zitiert auf den Seiten 34, 35, 91 und 92).

[CM98] Chowdhury, Abdur ; McCabe, M. C.: Improving Information Retrieval Systems Using Part of Speech Tagging, 1998 (zitiert auf den Seiten 39 und 40).

[CT06] Cover, Thomas M. ; Thomas, Joy A.: *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. USA : Wiley-Interscience, 2006. – ISBN 978–0–471–24195–9 (zitiert auf Seite 29).

[CVB18] Cleland-Huang, Jane ; Vierhauser, Michael ; Bayley, Sean: Dronology: An Incubator for Cyber-Physical Systems Research. In: *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*. Gothenburg, Sweden : Association for Computing Machinery, Mai 2018 (ICSE-NIER '18). – ISBN 978–1–4503–5662–6, S. 109–112 (zitiert auf Seite 99).

[CWTG18] Chung, Yu-An ; Weng, Wei-Hung ; Tong, Schrasing ; Glass, James: Unsupervised Cross-Modal Alignment of Speech and Text Embedding Spaces. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. Montréal, Canada : Curran Associates Inc., Dezember 2018 (NIPS'18), S. 7365–7375 (zitiert auf Seite 35).

[DCLT19] Devlin, Jacob ; Chang, Ming-Wei ; Lee, Kenton ; Toutanova, Kristina: BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding. In: *arXiv:1810.04805 [cs]* (2019), Mai (zitiert auf den Seiten 19, 42, 92 und 97).

[DLMOP12] De Lucia, Andrea ; Marcus, Andrian ; Oliveto, Rocco ; Poshyvanyk, Denys: Information Retrieval Methods for Automated Traceability Recovery. Version: 2012. `http://dx.doi.org/10.1007/978-1-4471-2239-5_4`. In: Cleland-Huang, Jane (Hrsg.) ; Gotel, Orlena (Hrsg.) ; Zisman, Andrea (Hrsg.): *Software and Systems Traceability*. London : Springer, 2012. – DOI 10.1007/978–1–4471–2239–5_4. – ISBN 978–1–4471–2239–5, S. 71–98 (zitiert auf den Seiten 14 und 27).

[DP98] DömgesRalf ; PohlKlaus: Adapting Traceability Environments to Project-Specific Needs. In: *Communications of the ACM* (1998), Dezember (zitiert auf Seite 1).

[ED97] Etzkorn, L.H. ; Davis, C.G.: Automatically Identifying Reusable OO Legacy Code. In: *Computer* 30 (1997), Oktober, Nr. 10, S. 66–71. `http://dx.doi.org/10.1109/2.625311`. – DOI 10.1109/2.625311. – ISSN 1558–0814 (zitiert auf Seite 37).

[Elm90] Elman, Jeffrey L.: Finding Structure in Time. In: *Cognitive Science* 14 (1990), Nr. 2, S. 179–211. `http://dx.doi.org/10.1207/s15516709cog1402_1`. – DOI 10.1207/s15516709cog1402_1. – ISSN 1551–6709 (zitiert auf Seite 8).

[ES19] Efstathiou, Vasiliki ; Spinellis, Diomidis: Semantic Source Code Models Using Identifier Embeddings. In: *Proceedings of the 16th International Conference on Mining Software Repositories*. Piscataway, NJ, USA : IEEE Press, 2019 (MSR '19), S. 29–33 (zitiert auf Seite 73).

[GCC17] Guo, Jin ; Cheng, Jinghui ; Cleland-Huang, Jane: Semantically Enhanced Software Traceability Using Deep Learning Techniques. In: *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, 2017. – ISSN 1558–1225, S. 3–14 (zitiert auf den Seiten xiii, 30 und 31).

[Gol17] Goldberg, Yoav: *Neural Network Methods in Natural Language Processing*. Morgan & Claypool Publishers, 2017 (Synthesis Lectures on Human Language Technologies Vol. 37). – ISBN 978–1–62705–298–6 (zitiert auf Seite 6).

[GOPL11] Gethers, Malcom ; Oliveto, Rocco ; Poshyvanyk, Denys ; Lucia, Andrea D.: On Integrating Orthogonal Information Retrieval Methods to Improve Traceability Recovery. In: *2011 27th IEEE International Conference on*

Software Maintenance (ICSM), 2011. – ISSN 1063–6773, S. 133–142 (zitiert auf Seite 99).

[HDSH04] Hayes, J.H. ; Dekhtyar, A. ; Sundaram, S.K. ; Howard, S.: Helping Analysts Trace Requirements: An Objective Look. In: *Proceedings. 12th IEEE International Requirements Engineering Conference, 2004.*, 2004. – ISSN 1090–705X, S. 249–259 (zitiert auf Seite 28).

[HS97] Hochreiter, Sepp ; Schmidhuber, Jürgen: Long Short-Term Memory. In: *Neural Computation* 9 (1997), November, Nr. 8, S. 1735–1780. http://dx.doi.org/10.1162/neco.1997.9.8.1735. – DOI 10.1162/neco.1997.9.8.1735. – ISSN 0899–7667 (zitiert auf Seite 9).

[HSP18] Heckman, Sarah ; Stolee, Kathryn T. ; Parnin, Christopher: 10+ Years of Teaching Software Engineering with Itrust: The Good, the Bad, and the Ugly. In: *Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training*. Gothenburg, Sweden : Association for Computing Machinery, Mai 2018 (ICSE-SEET '18). – ISBN 978–1–4503–5660–2, S. 1–4 (zitiert auf Seite 99).

[JM09] Jurafsky, Daniel ; Martin, James H.: *Speech and Language Processing (2nd Edition)*. USA : Prentice-Hall, Inc., 2009. – ISBN 978–0–13–187321–6 (zitiert auf den Seiten 4, 6, 10, 13, 101 und 102).

[KK19] Kutuzov, Andrey ; Kuzmenko, Elizaveta: To Lemmatize or Not to Lemmatize: How Word Normalisation Affects ELMo Performance in Word Sense Disambiguation. In: *Proceedings of the First NLPL Workshop on Deep Learning for Natural Language Processing*. Turku, Finland : Linköping University Electronic Press, September 2019, S. 22–28 (zitiert auf Seite 50).

[KMH+15] Kuang, Hongyu ; Mäder, Patrick ; Hu, Hao ; Ghabi, Achraf ; Huang, LiGuo ; Lü, Jian ; Egyed, Alexander: Can Method Data Dependencies Support the Assessment of Traceability Between Requirements and Source Code? In: *J. Softw. Evol. Process* 27 (2015), November, Nr. 11, S. 838–866. http://dx.doi.org/10.1002/smr.1736. – DOI 10.1002/smr.1736. – ISSN 2047–7473 (zitiert auf den Seiten 30, 53 und 71).

[KSKW15] Kusner, Matt J. ; Sun, Yu ; Kolkin, Nicholas I. ; Weinberger, Kilian Q.: From Word Embeddings To Document Distances. In: *ICML*, 2015 (zitiert auf Seite 25).

[LA04] Lattner, C. ; Adve, V.: LLVM: A Compilation Framework for Lifelong Program Analysis Transformation. In: *International Symposium on Code Generation and Optimization, 2004. CGO 2004.*, 2004. – ISSN null, S. 75–86 (zitiert auf Seite 22).

[LLS+19] Liu, Guangliang ; Lu, Yang ; Shi, Ke ; Chang, Jingfei ; Wei, Xing: Mapping Bug Reports to Relevant Source Code Files Based on the Vector Space Model and Word Embedding. In: *IEEE Access* 7 (2019), S. 78870–78881. http://dx.doi.org/10.1109/ACCESS.2019.2922686. – DOI 10.1109/ACCESS.2019.2922686. – ISSN 2169–3536 (zitiert auf Seite 31).

[LM14] Le, Quoc ; Mikolov, Tomas: Distributed Representations of Sentences and Documents. In: *International Conference on Machine Learning*, 2014, S. 1188–1196 (zitiert auf Seite 20).

[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 den Seiten 16, 23 und 78).

[MG12] Mäder, Patrick ; Gotel, Orlena: Towards Automated Traceability Maintenance. In: *Journal of Systems and Software* 85 (2012), Oktober, Nr. 10, S. 2205–2227. http://dx.doi.org/10.1016/j.jss.2011.10.023. – DOI 10.1016/j.jss.2011.10.023. – ISSN 0164–1212 (zitiert auf Seite 3).

[MM03] Marcus, Andrian ; Maletic, Jonathan I.: Recovering Documentation-to-Source-Code Traceability Links Using Latent Semantic Indexing. In: *Proceedings of the 25th International Conference on Software Engineering*. Portland, Oregon : IEEE Computer Society, Mai 2003 (ICSE '03). – ISBN 978–0–7695–1877–0, S. 125–135 (zitiert auf Seite 28).

[MN15] Mahmoud, Anas ; Niu, Nan: On the Role of Semantics in Automated Requirements Tracing. In: *Requirements Engineering* 20 (2015), September, Nr. 3, S. 281–300. http://dx.doi.org/10.1007/s00766-013-0199-y. – DOI 10.1007/s00766–013–0199–y. – ISSN 1432–010X (zitiert auf Seite 37).

[MPB+20] Moran, Kevin ; Palacio, David N. ; Bernal-Cárdenas, Carlos ; McCrystal, Daniel ; Poshyvanyk, Denys ; Shenefiel, Chris ; Johnson, Jeff: Improving the Effectiveness of Traceability Link Recovery Using Hierarchical Bayesian Networks. In: *arXiv:2005.09046 [cs]* (2020), Mai. http://dx.doi.org/10.1145/3377811.3380418. – DOI 10.1145/3377811.3380418 (zitiert auf den Seiten xiv, xviii, 99, 135, 136, 137, 138, 139, 140, 141 und 143).

[MRS08] Manning, Christopher D. ; Raghavan, Prabhakar ; Schütze, Hinrich: *Introduction to Information Retrieval*. Cambridge University Press, 2008. http://dx.doi.org/10.1017/CBO9780511809071. http://dx.doi.org/10.1017/CBO9780511809071 (zitiert auf Seite 4).

[MYZ13] Mikolov, Tomas ; Yih, Wen-tau ; Zweig, Geoffrey: Linguistic Regularities in Continuous Space Word Representations. In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Atlanta, Georgia : Association for Computational Linguistics, Juni 2013, S. 746–751 (zitiert auf Seite 15).

[NdS13] Nair, Sunil ; de la Vara, Jose L. ; Sen, Sagar: A Review of Traceability Research at the Requirements Engineering Conferencere@21. In: *2013 21st IEEE International Requirements Engineering Conference (RE)*, 2013. – ISSN 2332–6441, S. 222–229 (zitiert auf Seite 3).

[PNI+18] Peters, Matthew E. ; Neumann, Mark ; Iyyer, Mohit ; Gardner, Matt ; Clark, Christopher ; Lee, Kenton ; Zettlemoyer, Luke: Deep Contextualized Word Representations. In: *arXiv:1802.05365 [cs]* (2018), März (zitiert auf Seite 17).

[PNZY18] Peters, Matthew E. ; Neumann, Mark ; Zettlemoyer, Luke ; Yih, Wen-tau: Dissecting Contextual Word Embeddings: Architecture and Representation. In: *arXiv:1808.08949 [cs]* (2018), September (zitiert auf Seite 78).

[PSM14] Pennington, Jeffrey ; Socher, Richard ; Manning, Christopher: Glove: Global Vectors for Word Representation. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar : Association for Computational Linguistics, Oktober 2014, S. 1532–1543 (zitiert auf Seite 17).

[RMK13] Rempel, P. ; Mäder, P. ; Kuschke, T.: Towards Feature-Aware Retrieval of Refinement Traces. In: *2013 7th International Workshop on Traceability in*

Emerging Forms of Software Engineering (TEFSE), 2013, S. 100–104 (zitiert auf Seite 29).

[SAJ+19] S, Venkata K. ; Aggarwal, Rohit ; Jain, Shalini ; Desarkar, Maunendra S. ; Upadrasta, Ramakrishna ; Srikant, Y. N.: IR2Vec: A Flow Analysis Based Scalable Infrastructure for Program Encodings. In: *arXiv:1909.06228 [cs]* (2019), September (zitiert auf Seite 23).

[SB14] Shalev-Shwartz, Shai ; Ben-David, Shai: *Understanding Machine Learning: From Theory to Algorithms*. USA : Cambridge University Press, 2014. – ISBN 978–1–107–05713–5 (zitiert auf Seite 6).

[SWY75] Salton, G. ; Wong, A. ; Yang, C. S.: A Vector Space Model for Automatic Indexing. In: *Communications of the ACM* 18 (1975), November, Nr. 11, S. 613–620. http://dx.doi.org/10.1145/361219.361220. – DOI 10.1145/361219.361220. – ISSN 0001–0782 (zitiert auf Seite 14).

[Tel20] Telge, Tobias: *Worteinbettungen Für Die Anforderungsdomäne*, Karlsruher Institut für Technologie (KIT) – IPD Tichy, Bachelor's Thesis, März 2020 (zitiert auf Seite 92).

[VSP+17] Vaswani, Ashish ; Shazeer, Noam ; Parmar, Niki ; Uszkoreit, Jakob ; Jones, Llion ; Gomez, Aidan N. ; Kaiser, Lukasz ; Polosukhin, Illia: Attention Is All You Need, 2017 (zitiert auf Seite 11).

[YLW+18] Yang, Pengcheng ; Luo, Fuli ; Wu, Shuangzhi ; Xu, Jingjing ; Zhang, Dongdong ; Sun, Xu: Learning Unsupervised Word Mapping by Maximizing Mean Discrepancy. In: *arXiv:1811.00275 [cs]* (2018), November (zitiert auf Seite 35).

Anhang

A Weitere Evaluationsergebnisse

A.1 Klasseneinbettungen

Im Folgenden sind die Evaluationsergebnisse für iTrust, die analog für eTour in Abschnitt 6.3.2.1 durchgeführt wurden. Das Verfahren ist gleich: Die Anforderungen wird also jeweils als Durchschnitt ihrer Wörter repräsentiert und für die Klasseneinbettungen werden verschiedene Kombinationen getestet. Es wird die Kosinusähnlichkeit verwendet.

Tabelle A.1: Beste F1-Werte für Klasseneinbettungsvarianten mit Methodensignaturen, Klassennamen oder allen Elementen

Variante	Bestes F1	Ausbeute	Präz.	D-SW	D. Präz.
`SIG`	18,8%	18,2%	19,4%	0,77	11,9%
`SIG+KL`	19,7%	19.2%	20,2%	0,76	**12,6%**
Klassenname, Klassenkommentar	13,3%	11,5%	15,8%	0,75	6,2%
Alle Bezeichner in Klasse	11,8%	12,6%	11%	0,84	4,6%
Alle Klassenelem. zweistufig	**20,5%**	19,9%	21%	0,79	11,9%

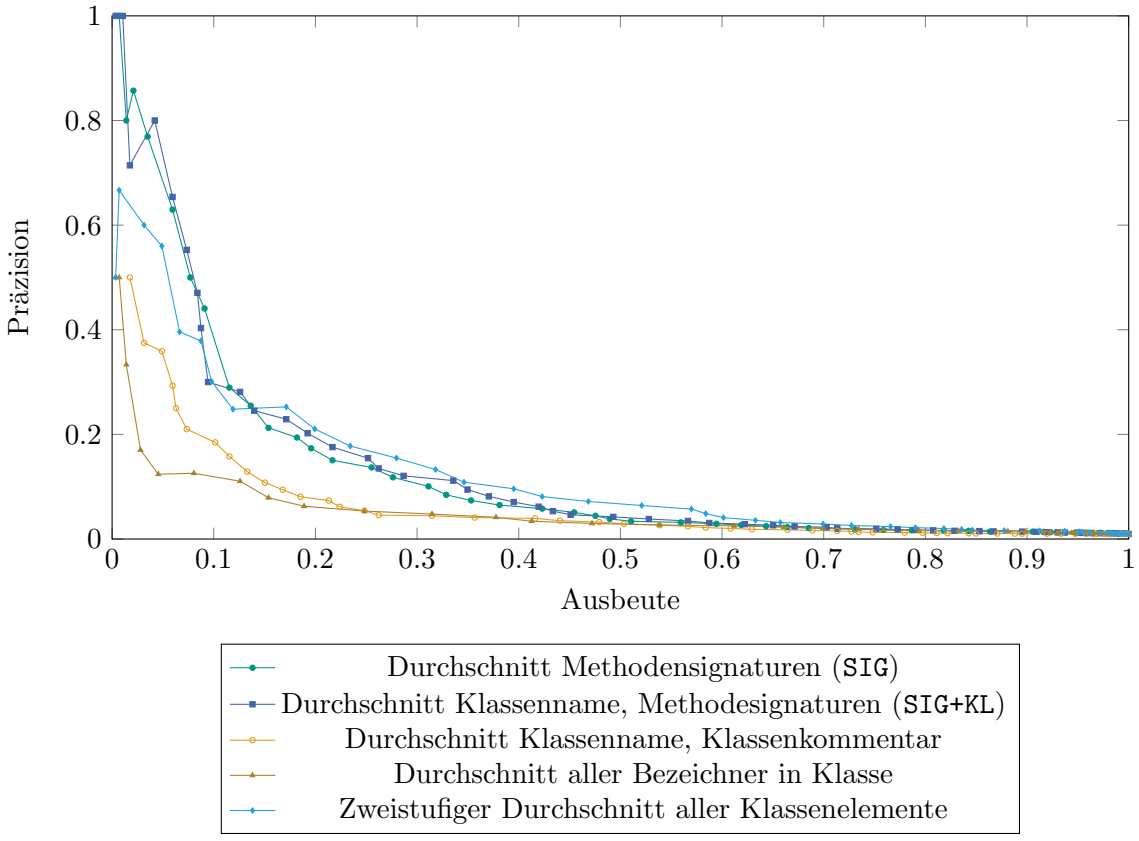


Abbildung A.1: iTrust: Ausbeute-Präzisionskurven für Klasseneinbettungensvarianten mit Methodensignaturen, Klassennamen oder allen Elementen

Tabelle A.2: Beste F1-Werte für die Varianten des Methodendurchschnitts

Variante	Bestes F1	Ausbeute	Präz.	D-SW	D. Präz.
`SIG+KL`	19,7%	19.2%	20,2%	0,76	12,6%
`SIG+KL+KO`	**21,5%**	19,9%	23,3%	0,79	**14,7%**
Mit Methodenrumpf	19,7%	18,2%	21,5%	0,79	11,1%
Mit Methodenkommentar und -rumpf	19,3%	15,4%	25,9%	0,81	11,4%
Alle Klassenelem. zweistufig	20,5%	19,9%	21%	0,79	11,9%

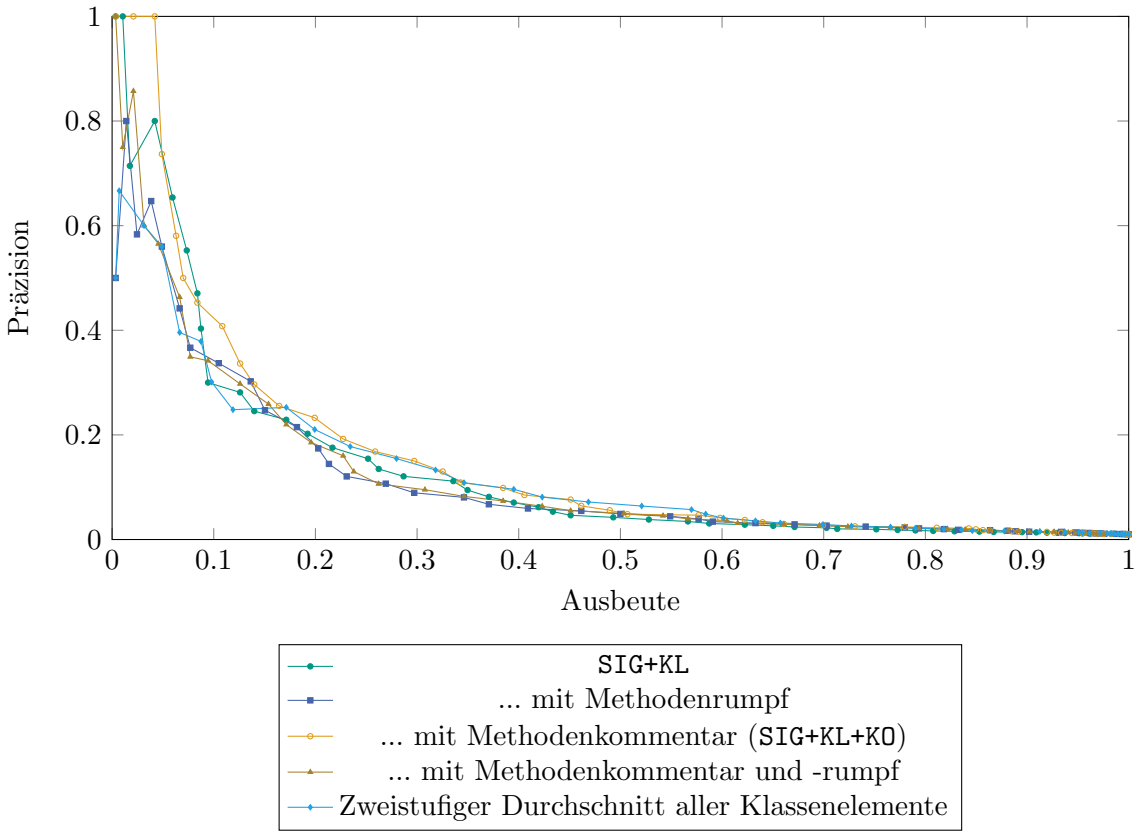


Abbildung A.2: iTrust: Varianten des Methodendurchschnitts

Tabelle A.3: Beste F1-Werte für den Vergleich der separaten Inklusion des MEthodenkommentars

Variante	Bestes F1	Ausbeute	Präz.	D-SW	D. Präz
`SIG+KL`	19,7%	19.2%	20,2%	0,76	12,6%
`SIG+KL+KO`	**21,5%**	19,9%	23,3%	0,79	**14,7%**
Methodenkommentar separat	20,9%	16,8%	27,6%	0,81	14,2%

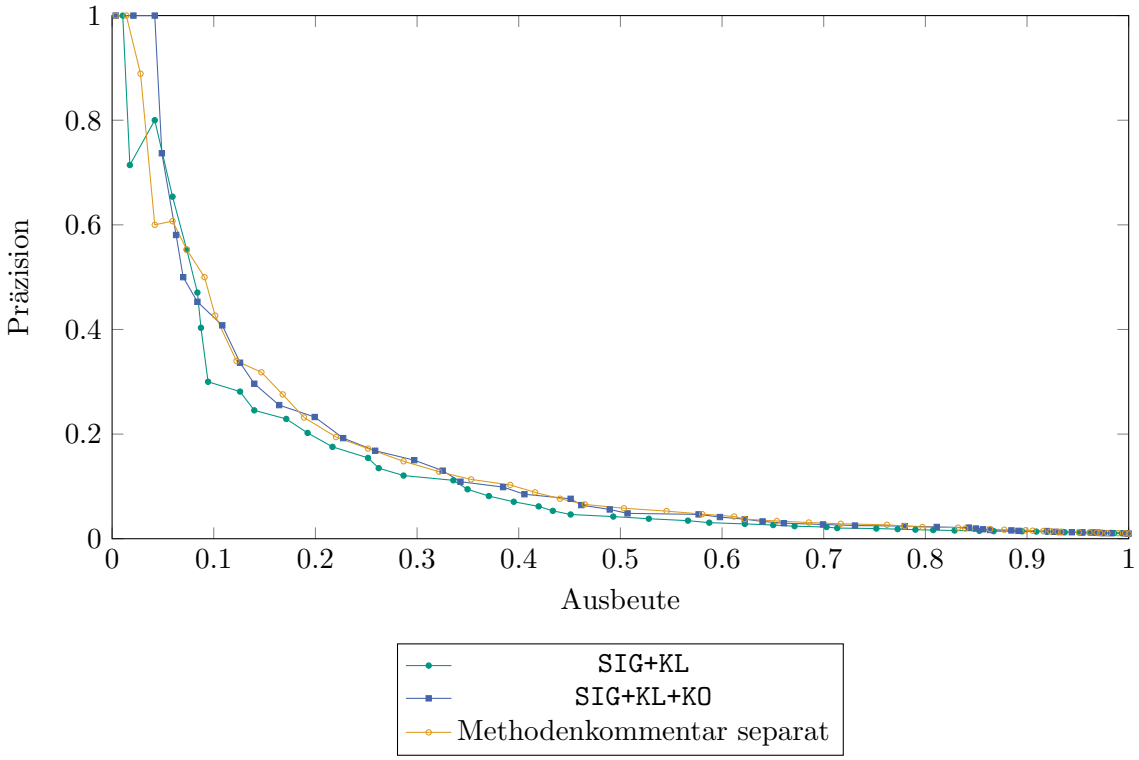


Abbildung A.3: iTrust: Ausbeute-Präzisionskurven über zwei Varianten, den Methodenkommentar miteinzubeziehen

Tabelle A.4: Beste F1-Werte für die Varianten mit Klassenkommentare, Superklassifizierer und Attribute

Variante	Bestes F1	Ausbeute	Präz.	D-SW	D. Präz
`SIG+KL`	19,7%	19.2%	20,2%	0,76	12,6%
`SIG+KL+KO`	**21,5%**	19,9%	23,3%	0,79	**14,7%**
Mit Klassenkommentar	20,3%	24,1%	17,5%	0,76	13,6%
Mit Superklassifizierer	20,1%	25,2%	16,8%	0,74	13,3%
Mit Attribut	14,5%	19,9%	11,4%	0,73	7,8%
Mit Attribut und Attributkommentar	14,4%	17,5%	12,3%	0,74	7,6%

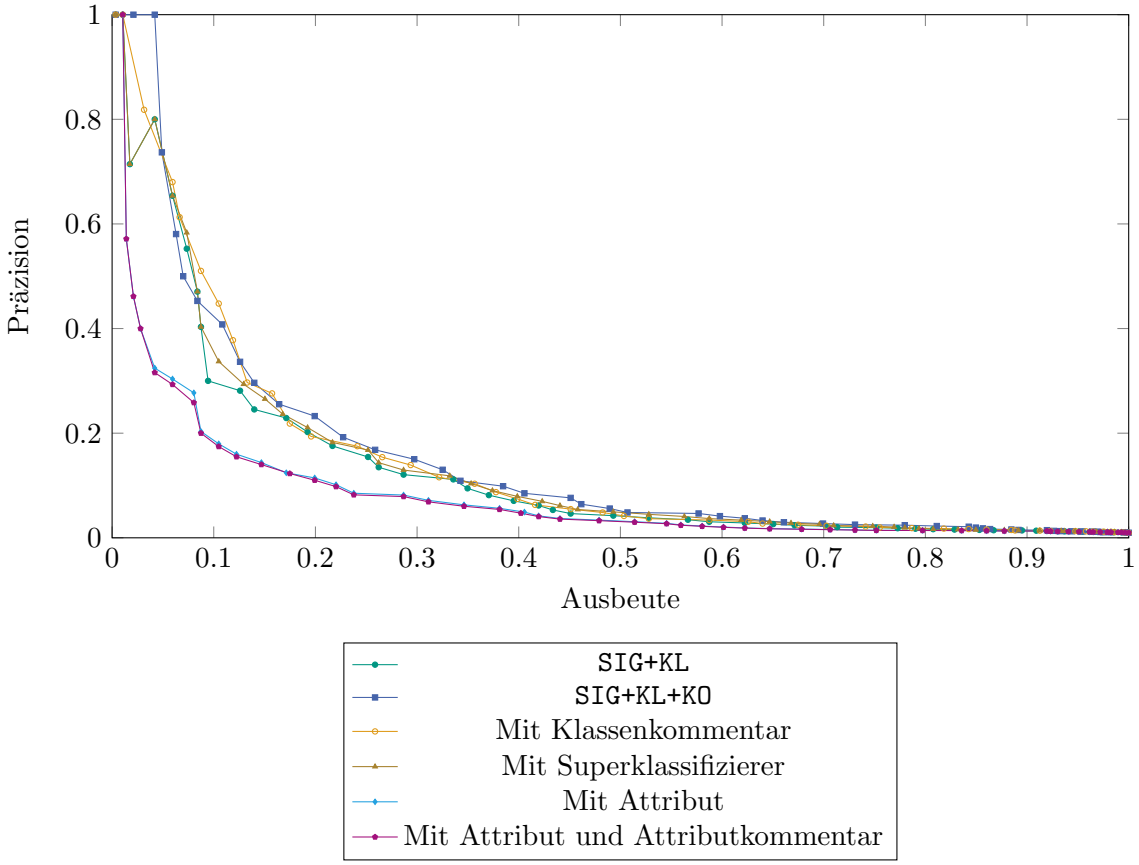


Abbildung A.4: iTrust: Ausbeute-Präzisionskurven der Varianten mit Superklassifizierer, Attribute und Klassenkommentare

Tabelle A.5: Beste F1-Werte für die Kombinationen der Kommentare

Variante	Bestes F1	Ausbeute	Präz.	D-SW	D. Präz.
`SIG+KL`	19,7%	19.2%	20,2%	0,76	12,6%
`SIG+KL+KO`	21,5%	19,9%	23,3%	0,79	14,7%
Mit Klassenkommentar	20,3%	24,1%	17,5%	0,76	13,6%
`SIG+KL+KO+KKO`	**23,3%**	24,1%	22,5%	0,79	**15,4%**

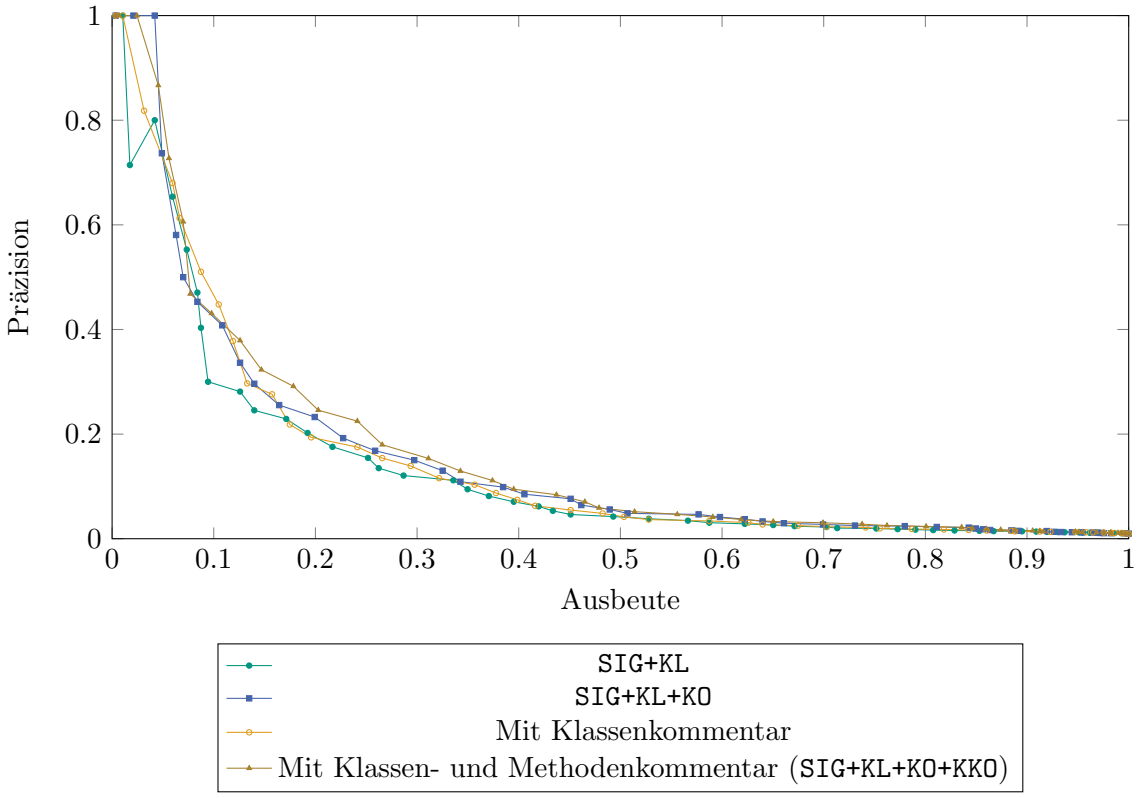


Abbildung A.5: iTrust: Vergleich der Kombination von Klassen- und Methodenkommentaren zu den bisherigen Varianten

Tabelle A.6: Beste F1-Werte für die Varianten der Anforderungseinbettung

Variante	Bestes F1	Ausbeute	Präz.	D-SW	D. Präz.
SIG+KL	19,7%	19.2%	20,2%	0,76	12,6%
SIG+KL+KO+KKO	**23,3%**	24,1%	22,5%	0,79	**15,4%**
SIG+KL. (2x Aggr.)	18,6%	22,4%	15,9%	0,74	12,0%
SIG+KL+KO+KKO (2x Aggr.)	22,4%	22,7%	21,1%	0,79	14,1%

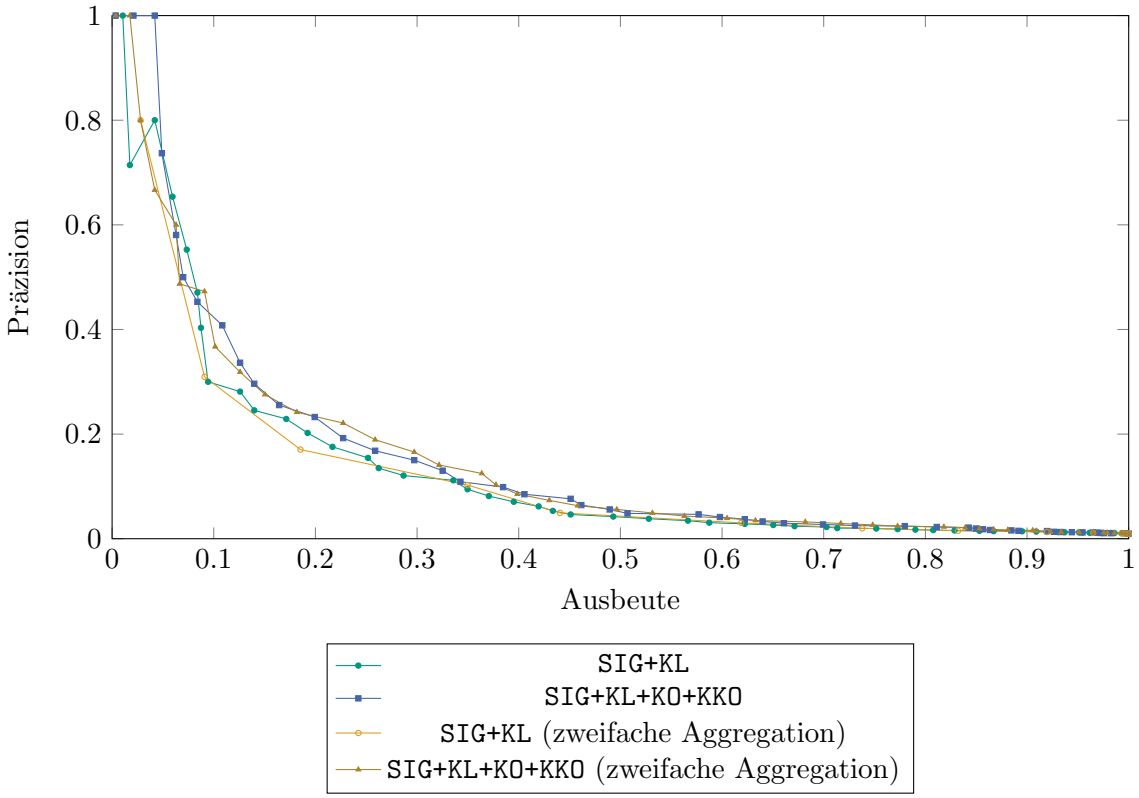


Abbildung A.6: iTrust: Vergleich der Anforderungseinbettungen mit einfacher und zweifacher Aggregation

Tabelle A.7: Beste F1-Werte für Kombinationen mit dem Standard- und dem Anforderungsmodell für fastText

Variante	Bestes F1	Ausbeute	Präzision	D-SW	D. Präz.
SIG+KL	19,7%	19.2%	20,2%	0,76	12,6%
SIG+KL+KO+KKO	**23,3%**	24,1%	22,5%	0,79	**15,4%**
(A) SIG+KL	18,8%	15,4%	24,3%	0,89	10,2%
(A) SIG+KL+KO+KKO	21,2%	22,7%	19,9%	0,9	12,7%

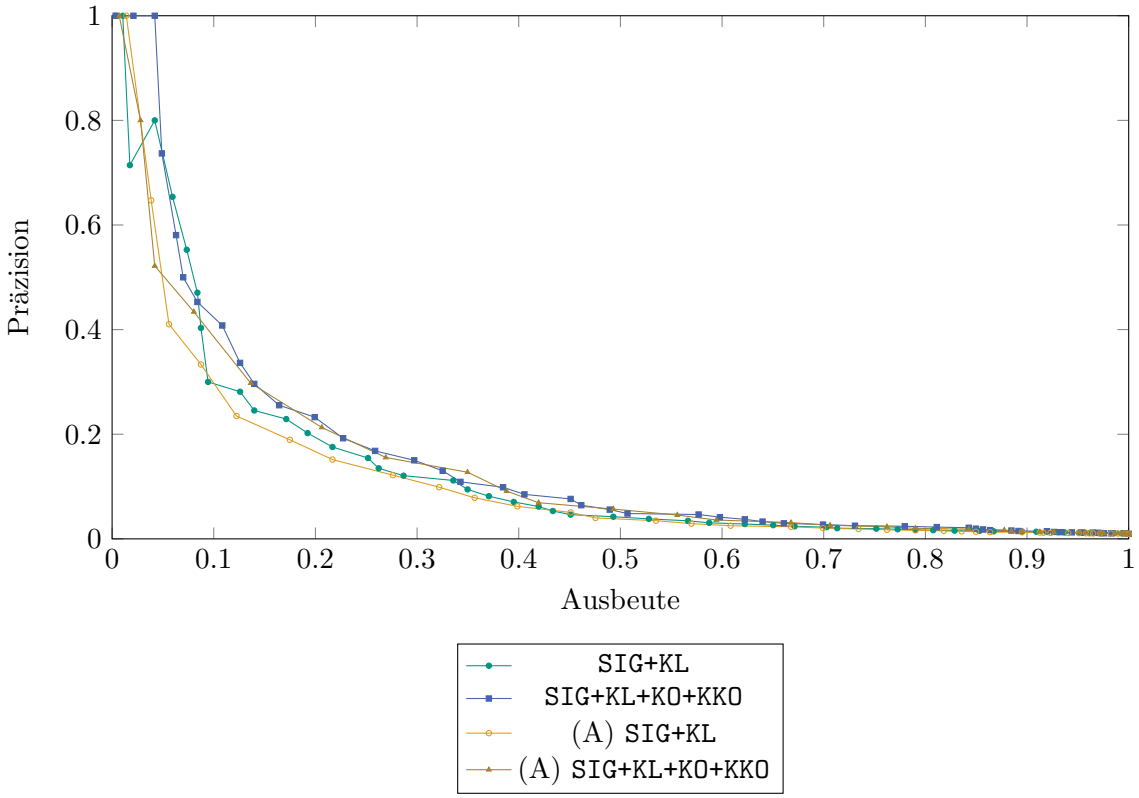


Abbildung A.7: iTrust: Vergleich der Rückverfolgbarkeit mit dem Standard- und dem Anforderungsmodell für fastText

A.2 Mehrheitsentscheid

In diesen Abschnitt befinden sich die Evaluationsergebnisse des Mehrheitsentscheid mit iTrust. Das Vorgehen ist identisch wie in Abschnitt 6.3.3.1.

Tabelle A.8: Beste F1-Werte für Methodensignatur- und Methodenkommentarvarianten des Mehrheitsentscheids

Variante	Bestes F1	Ausbeute	Präz.	D-SW	M-SW	D. Präz.
`SIG+KL`	19,7%	19.2%	20,2%	0,76	-	12,6%
`SIG+KL+KO`	21,5%	19,9%	23,3%	0,79	-	**14,7%**
(M) `SIG+KL`	20,4%	24,5%	17,9%	0,77	0,29	11,6%
(M) `SIG+KL+KO`	**25,2%**	26,2%	24,2%	0,79	0,59	13,8%

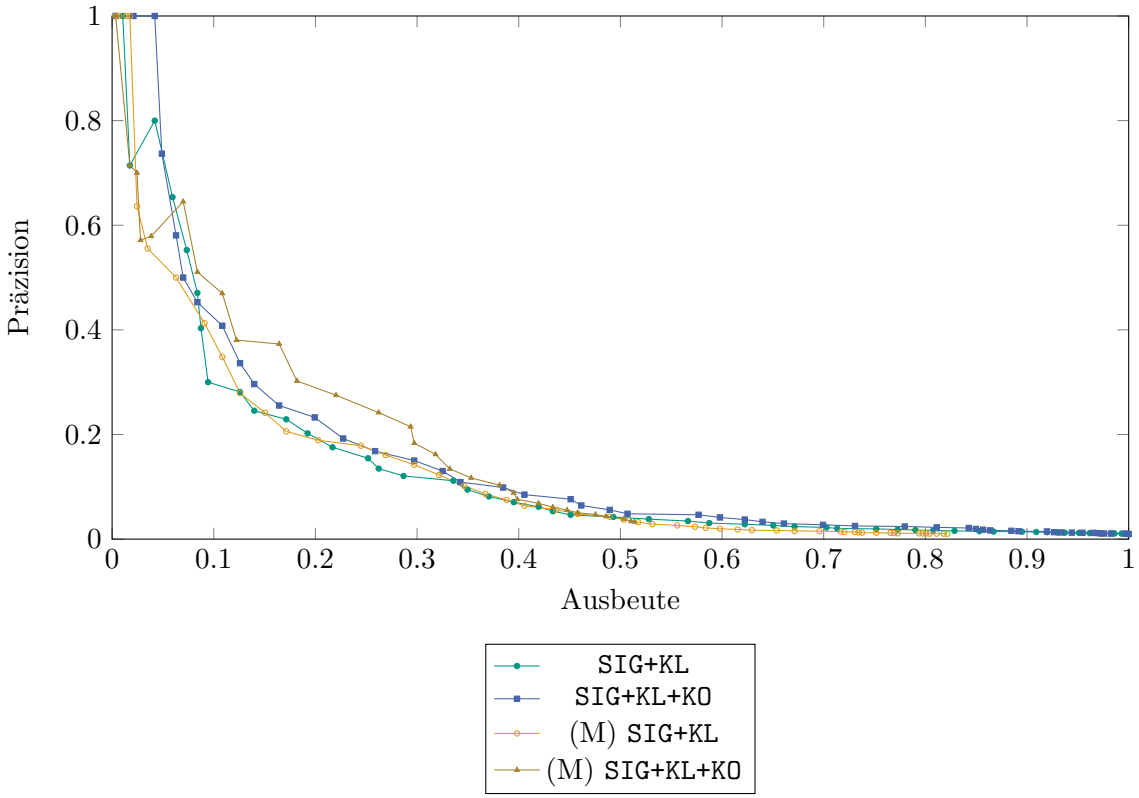


Abbildung A.8: iTrust: Vergleich zwischen Mehrheitsentscheid und Klasseneinbettung bezüglich der Varianten mit Methodensignatur oder -kommentar

Tabelle A.9: Beste F1-Werte für den Vergleich des Mehrheitsentscheids mit Kommentarvarianten

Variante	Bestes F1	Ausbeute	Präz.	D-SW	M-SW	D. Präz.
`SIG+KL+KO+KKO`	23,3%	24,1%	22,5%	0,79	-	**15,4%**
Mit Klassenkommentar	20,3%	24,1%	17,5%	0,76	-	13,6%
(M) `SIG+KL+KO+KKO`	23,3%	18,2%	32,5%	0,82	0,59	13,6%
(M) Mit Klassenkomm.	18,7%	12,9%	33,6%	0,82	0,54	10,5%
(M) `SIG+KL+KO`	**25,2%**	26,2%	24,2%	0,79	0,59	13,8%

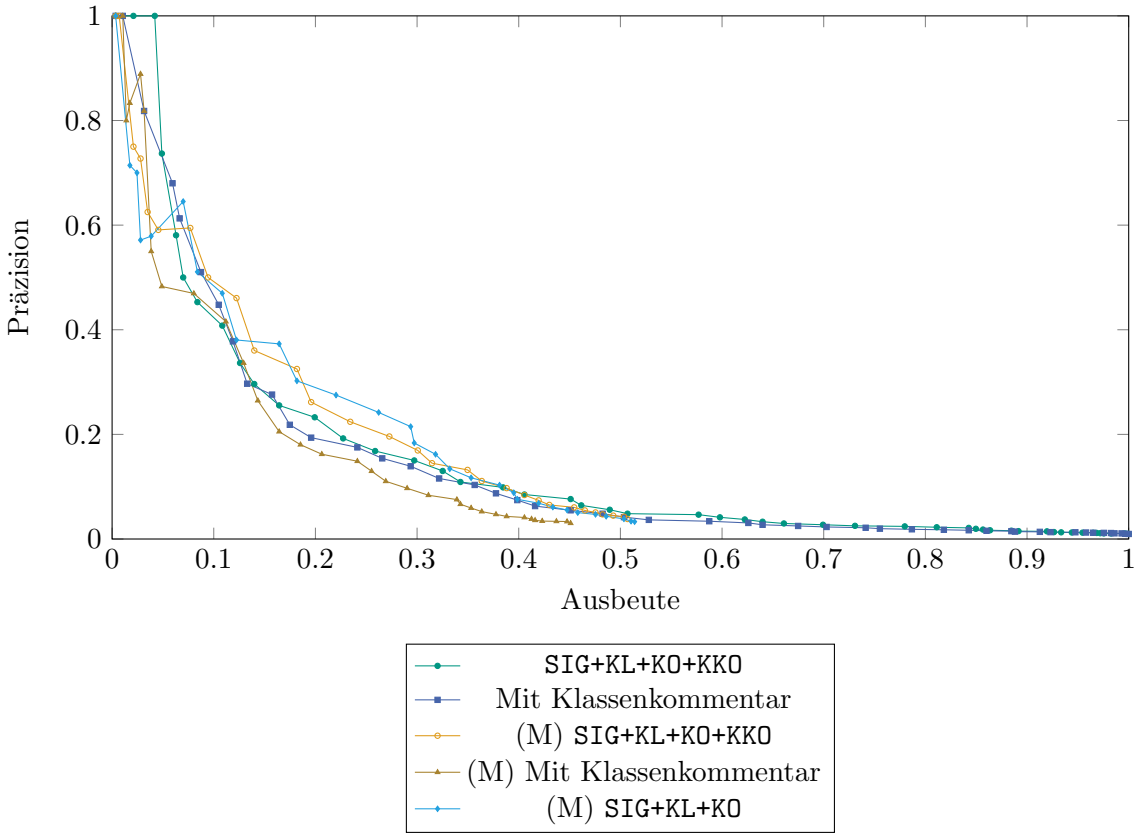


Abbildung A.9: iTrust: Vergleich der Kommentarvarianten zwischen Mehrheitsentscheid und Klasseneinbettung

Tabelle A.10: Beste F1-Werte der Varianten mit Superklassifizierer und Methodenrümpfe für den Vergleich mit dem Mehrheitsentscheid

Variante	Bestes F1	Ausbeute	Präz.	D-SW	M-SW	D. Präz.
Mit Methodenrumpf	19,7%	18,2%	21,5%	0,79	-	11,1%
Mit Superklassifizierer	20,1%	25,2%	16,8%	0,74	-	13,3%
(M) Mit Methodenrumpf	19,4%	18,2%	20,7%	0,79	0,61	8,8%
(M) Mit Superklassifizierer	21,2%	24,5%	18,7%	0,77	0,3	12,2%
(M) `SIG+KL+KO`	**25,2%**	26,2%	24,2%	0,79	0,59	**13,8%**

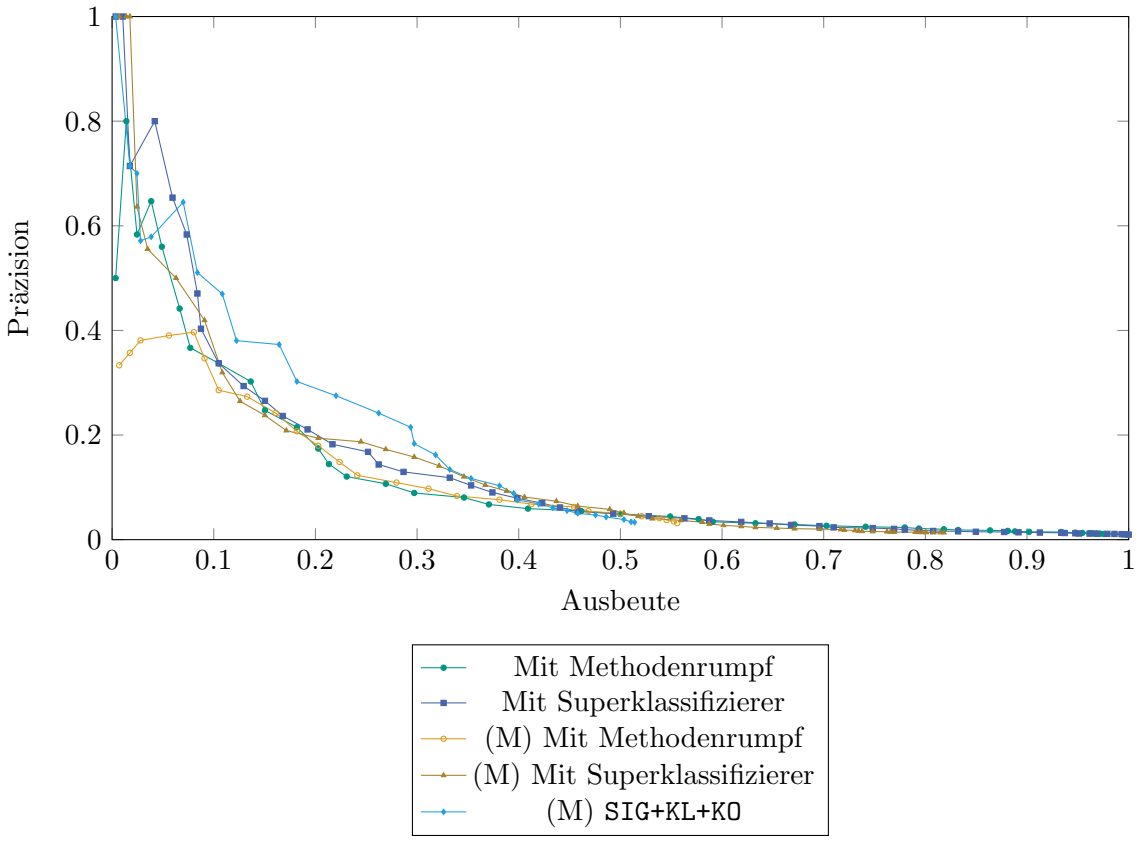


Abbildung A.10: iTrust: Vergleich der Varianten mit Superklassifizierer und Methodenrümpfe zwischen Mehrheitsentscheid und Klasseneinbettung

A.3 Mehrheitsentscheid mit Aufrufabhängigkeiten

Die folgenden Daten zeigen die gleiche Auswertung wie in Abschnitt 6.3.3.2 für iTrust.

Abbildung A.11: Beste F1-Werte für den Mehrheitsentscheid mit (ÄS) `SIG` in verschiedenen Varianten

Variante	Bestes F1	Ausb.	Präz.	D-SW	M-SW	E-SW	MG	D. Präz.
`SIG`	18,8%	18,2%	19,4%	0,77	-	-	-	11,9%
(M) `SIG`	21,0%	24,5%	18,3%	0,77	0,45	-	-	11,3%
(ÄS) `SIG`↕	20,1%	20,6%	19,5%	0,73	0,69	0	0,8	7,1%
(ÄS) `SIG`↑	19,6%	22,4%	17,4%	0,77	0,45	0	0,66	**12,0%**
(ÄS) `SIG`↓	**20,6%**	18,9%	22,7%	0,75	0,7	0,5	0,81	7,4%

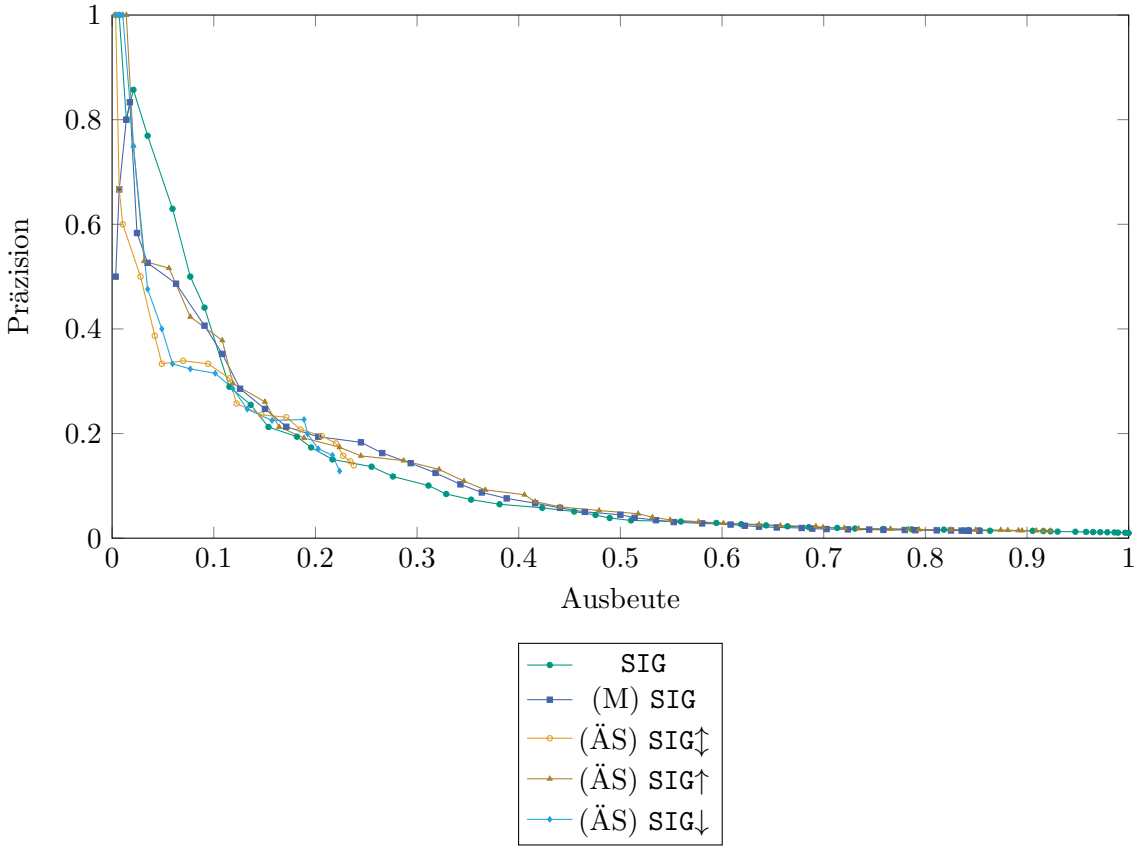


Abbildung A.12: iTrust: Vergleich der Varianten mit (ÄS) `SIG`

Tabelle A.11: Beste F1-Werte für den Mehrheitsentscheid mit (VS) `SIG` und den bisherigen Varianten

Variante	Bestes F1	Ausb.	Präz.	D-SW	M-SW	E-SW	MG	D. Präz.
`SIG`	18,8%	18,2%	19,4%	0,77	-	-	-	11,9%
(M) `SIG`	21,0%	24,5%	18,3%	0,77	0,45	-	-	11,3%
(ÄS) `SIG`↑	19,6%	22,4%	17,4%	0,77	0,45	0	0,66	**12,0%**
(VS) `SIG`↑	**22,3%**	19,2%	26,4%	0,78	0,72	-	0,66	10,4%

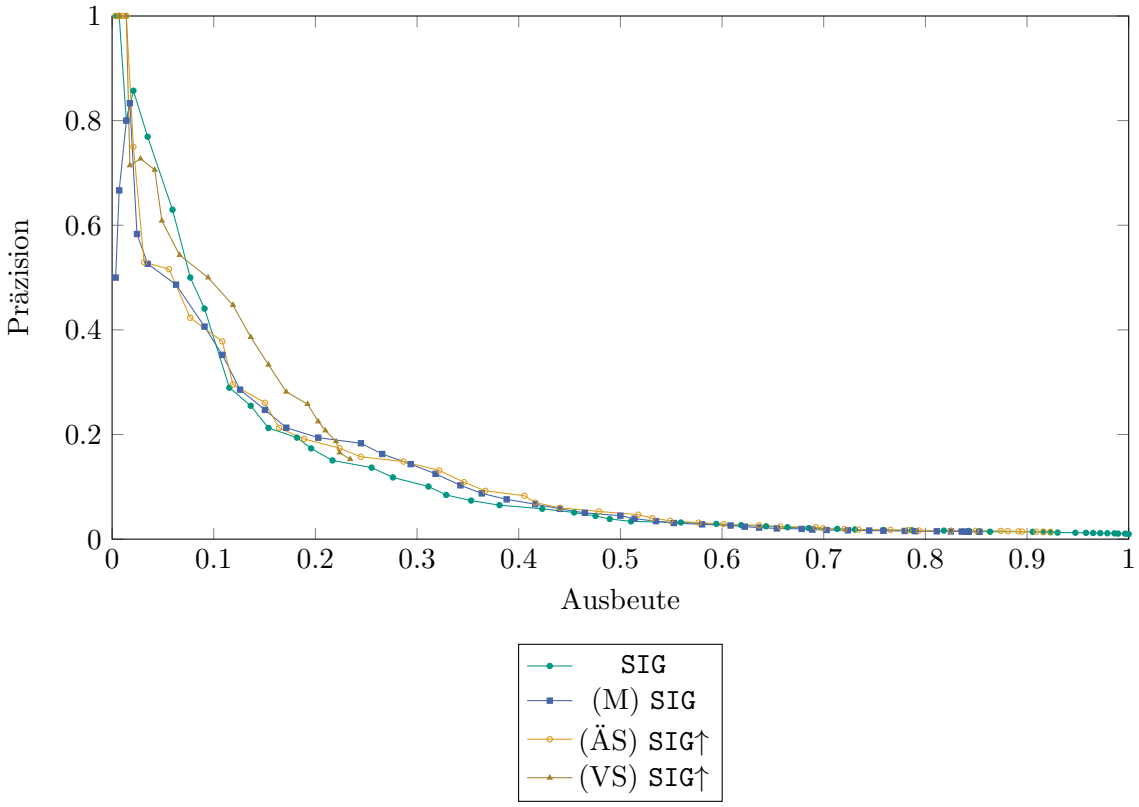


Abbildung A.13: iTrust: Vergleich der bsiherigen Varianten mit (VS) `SIG`

Tabelle A.12: Beste F1-Werte für (VS) `SIG` mit Kommentaren

Variante	Bestes F1	Ausb.	Präz.	D-SW	M-SW	MG	D. Präz
(VS) `SIG`↑	22,3%	19,2%	26,4%	0,78	0,72	0,66	10,4%
(VS) `SIG+KO`↕	20,4%	25,5%	17%	0,8	0,59	ungewichtet	**12,6%**
(VS) `SIG+KO`↑	**24%**	22%	26,4%	0,82	0,69	0,68	12,5%

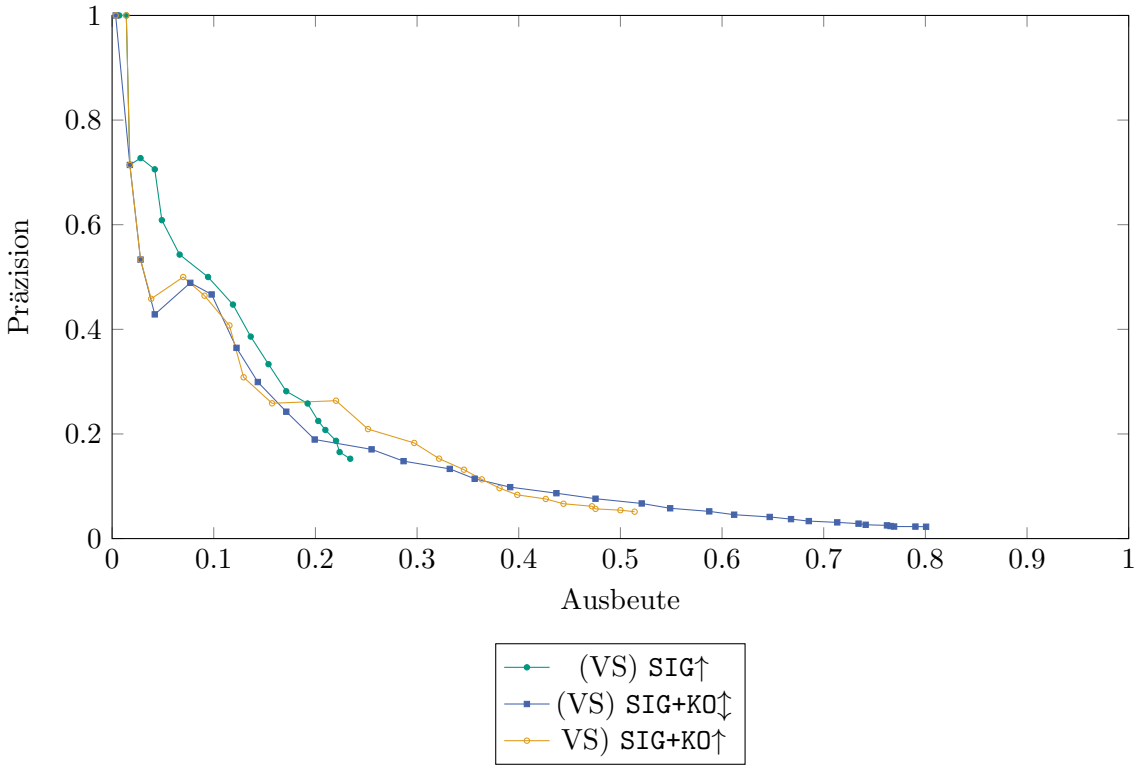


Abbildung A.14: iTrust: Vergleich der Varianten von (VS) `SIG` mit Kommentaren

A.4 Mehrheitsentscheid mit WMD

Die folgenden Daten zeigen die gleiche Auswertung wie in Abschnitt 6.3.4.2 für iTrust.

Tabelle A.13: Beste F1-Werte für den Mehrheitsentscheid mit WMD-Kombinationen

Variante	Bestes F1	Ausb.	Präz.	D-SW	M-SW	E-SW	MG	D. Präz.
`MS+AS`	21,3%	24,1%	19%	0,6	0,6	1	-	10,4%
`MK+AS`	23,4%	34,3%	17,8%	0,61	0,59	0,6	-	10,2%
`MS+A`	19,8%	19,2%	20,4%	0,54	0,64	-	-	9,7%
`MK+A`	24,2%	26,2%	22,4%	0,52	0,59	-	-	13,1%
(VS) `SIG+KO↑`	24%	22%	26,4%	0,82	0,69	-	0,68	12,5%
(M) `SIG+KL+KO`	**25,2%**	26,2%	24,2%	0,79	0,59	-	-	**13,8%**

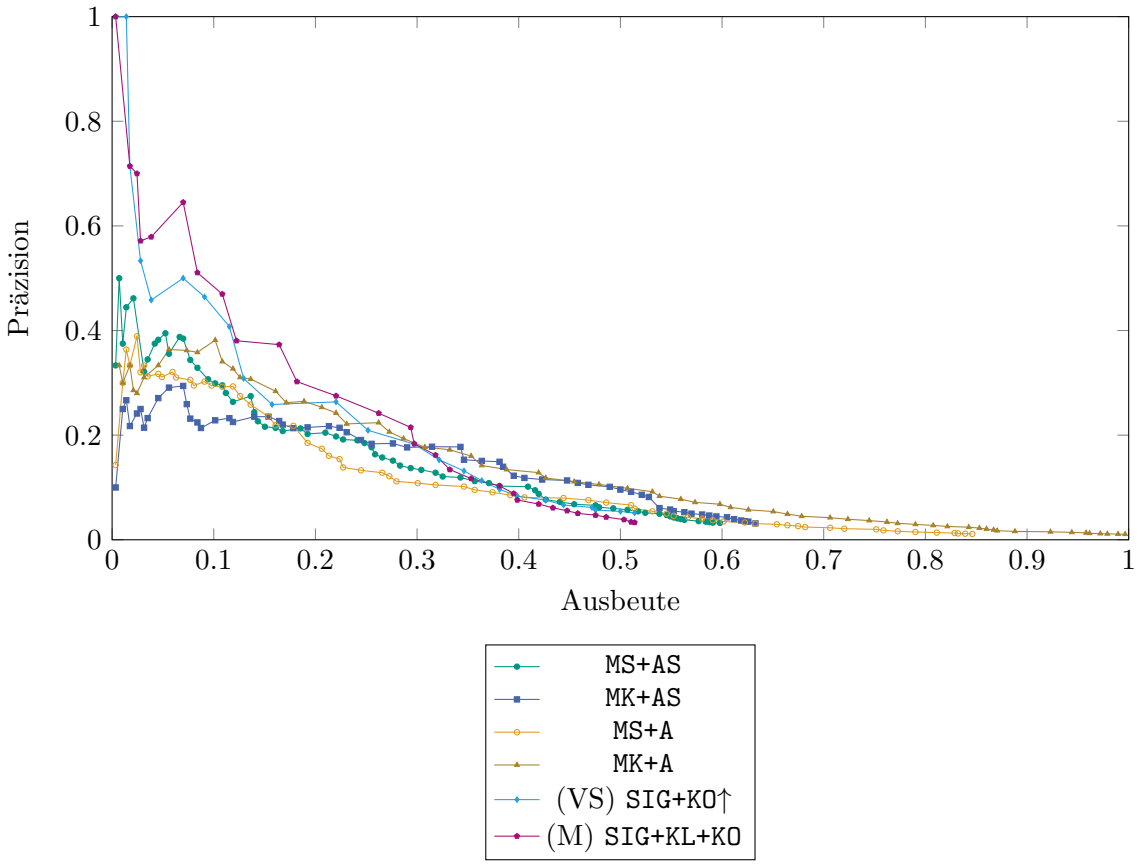


Abbildung A.15: iTrust: Ausbeute-Präzisionskurven des Mehrheitsentscheid mit verschiedenen WMD-Kombinationen. Reduktionsfunktionen:
`MS+AS`: Durchschnitt (Quell.), Minimum (Anf.)
`MK+AS`: Minimum (Quell.), Minimum (Anf.)
`MS+A`: Minimum (Quell.)
`MK+A`: Minimum (Quell.)

Tabelle A.14: Beste F1-Werte für den WMD-Mehrheitsentscheid mit mit Mehrheitentscheidschwellwert und Top-N

Variante	Bestes F1	Ausbeute	Präz.	D-SW	M-SW	E-SW	D. Präz.
`MS+AS`	**21,3%**	24,1%	19%	0,6	0,6	1	10,4%
`MS+A`	19,8%	19,2%	20,4%	0,54	0,64	-	9,7%
(T) `MS+AS`	21,1%	24,8%	18,4%	0,52	0,97	1	**11,3%**
(T) `MS+A`	20,0%	18,2%	22,1%	0,5	0,75	0,64	10,7%

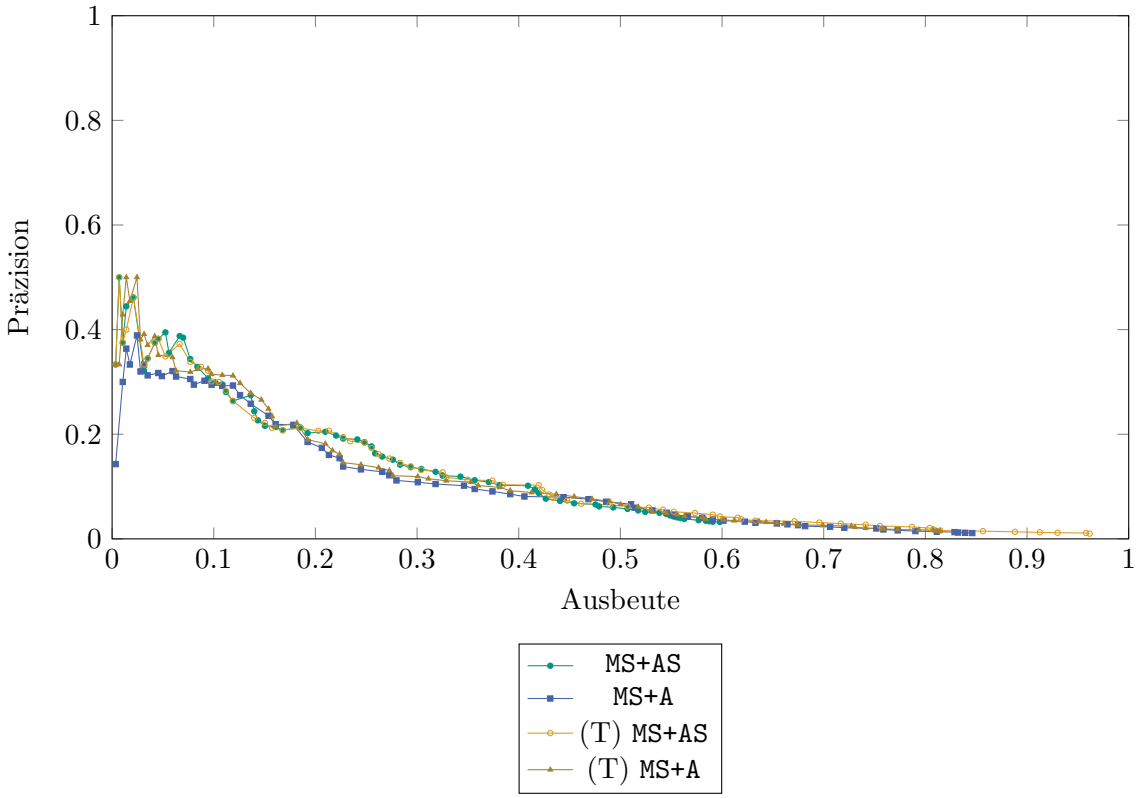


Abbildung A.16: iTrust: Ausbeute-Präzisionskurven des WMD-Mehrheitsentscheid mit Mehrheitentscheidschwellwert und Top-N. Reduktionsfunktionen:
`MS+AS`: Durchschnitt (Quell.), Minimum (Anf.)
`MS+A`: Minimum (Quell.)
(T) `MS+AS`: Durchschnitt (Quell.), Minimum (Anf.)
(T) `MS+A`: Minimum (Quell.)