Ruprecht-Karls-Universität Heidelberg

Fakultät für Mathematik und Informatik

Institut für Informatik

Masterarbeit

(Semi-)automatisierte Vergabe von aussagekräftigen Feature-Namen für User Stories in Issue-Tracking-Systemen

Name:	Markus Schindler
Matrikelnummer:	3409527
Betreuer:	Prof. Dr. Barbara Paech, M.Sc. Marcus Seiler
Datum der Abgabe:	18. Dezember 2018

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Sowohl inhaltlich als auch wörtlich entnommene Inhalte wurden als solche kenntlich gemacht. Die Arbeit ist in gleicher oder vergleichbarer Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Heidelberg, den 18. Dezember 2018

Abstract

Software features play an essential role in many aspects of software development. During release planning selected features determine the scope of the next release of a software. In software product line engineering entire software product families are built around key features. Analysing feature usage gains valuable insights into the patterns of user behaviour.

Despite their importance, software features are often not managed in a well structured manner. In many cases explicit feature information is thus not readily available but only implicitly contained in and scattered across many software artifacts. Often, this results in lacking traceability between features and their refining artifacts. It has been suggested that tagging software artifacts with feature labels helps to ease feature management and prevent fragmented feature knowledge. User stories are short natural language artifacts commonly used to describe the requirements of a software. Generally, user stories are managed in issue tracking systems.

This work proposes a semi-automatic model for assigning feature labels to user stories. The model achieves this goal by 1) extracting and accumulating feature-relevant information from user stories and 2) recommending suitable feature labels in accordance with previously assigned labels. Feature-relevant information is extracted by applying part-of-speech patterns on parse trees of user stories. Recommendation of feature labels is accomplished by the means of an artificial neural network. The evaluation based on two student project related user story datasets shows that the proposed model is capable of extracting meaningful feature-relevant information as well as recommending proper feature labels.

Zusammenfassung

Software-Features spielen eine essenzielle Rolle in vielen Aspekten der Software-Entwicklung. Die während des Release-Planning ausgewählten Features legen den Funktionsumfang der nächsten Version einer Software fest. Im Software-Product-Line-Engineering werden komplette Software-Produktfamilien um zentrale Features herum aufgebaut. Die Analyse der Verwendung von Features erlaubt wertvolle Einsichten beim Erkennen von Mustern innerhalb des Nutzerverhaltens.

Trotz ihrer Wichtigkeit werden Software-Features häufig nicht systematisch verwaltet. In vielen Fällen sind explizite Informationen über Features daher nicht leicht verfügbar. Oft sind diese nur implizit in den verschiedenen Artefakten der Software-Entwicklung enthalten und über diese verteilt. In der Literatur wurde angemerkt, dass die Vergabe von Feature-Labels für Artefakte helfen kann, die Verwaltung von Features zu erleichtern und der Fragmentierung von Feature-Wissen entgegen zu wirken. User Stories sind kurze, in natürlicher Sprache verfasste Artefakte, die häufig verwendet werden, um Anforderungen an eine Software festzuhalten. User Stories werden oft in Issue-Tracking-Systemen verwaltet.

In dieser Arbeit wird ein (semi-)automatisches Modell für die Vergabe von Feature-Labels für User Stories vorgestellt. Das Modell erreicht dies durch 1) die Extraktion und Bündelung feature-relevanter Informationen aus User Stories und 2) dem Vorschlagen von passenden Feature-Labels anhand bereits vergebener Labels. Feature-relevante Informationen werden durch das Anwenden von Part-Of-Speech-Mustern auf den Syntaxbäumen von User Stories extrahiert. Das Vorschlagen von Feature-Labels wird mithilfe eines künstlichen neuronalen Netzes erreicht. Die Evaluation anhand zweier Datensätze von User Stories aus studentischen Projekten zeigt, dass das vorgestellte Modell im Stande ist, aussagekräftige feature-relevante Informationen zu extrahieren und treffende Feature-Labels vorzuschlagen.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ACM Association for Computing Machinery.

idf Inverse-Document-Frequency.

IEEE Institute of Electrical and Electronics Engineers.

ITS Issue-Tracking-System.

KNN Künstliches neuronales Netz.

LDA Latent-Dirichlet-Allocation.

LSA Latent-Semantic-Analysis.

LSI Latent-Semantic-Indexing.

ML Machine-Learning.

NER Named-Entity-Recognition.

NLP Natural-Language-Processing.

OSS Open-Source-Software.

POS Part-Of-Speech.

SPLE Software-Product-Line-Engineering.

tf Term-Frequency.

tf-idf Term-Frequency-Inverse-Document-Frequency.

VVS Versionsverwaltungssystem.

1. Einleitung

In diesem Kapitel erfolgt eine Einführung in das Themengebiet der vorliegenden Arbeit. In Abschnitt 1.1 wird der Kontext und die Motivation der Arbeit vorgestellt. Die Problemstellung und Zielsetzung der Arbeit wird in Abschnitt 1.2 definiert. Schließlich wird in Abschnitt 1.3 der Aufbau der weiteren Arbeit erläutert.

1.1. Kontext und Motivation

Software-Features sind ein zentrales Konzept innerhalb vieler Gebiete der Software-Entwicklung. Im Release-Planning werden zu implementierende Software-Features priorisiert und legen somit sowohl den Inhalt, als auch den Umfang der nächsten Version einer Software fest [1]. Die Bestimmung von wesentlichen Software-Features ermöglicht im Software-Product-Line-Engineering (SPLE) die Wiederverwendung von Anforderungen innerhalb einer Produktfamilie [3]. Durch die Auswertung von Produktbeschreibungen und -kommentaren können oft genutzte Software-Features identifiziert und das Nutzerverhalten bzgl. der Software analysiert werden [13].

Trotz ihrer Relevanz werden Software-Features häufig nicht strukturiert erfasst, dokumentiert und verwaltet [27, 28]. So existiert innerhalb eines Projekts oft keine allgemeingültige und explizite Beschreibung von Software-Features. Vielmehr sind Feature-Beschreibungen nur implizit in den verschiedenen Software-Artefakten wie Anforderungen und Quellcode-Dateien enthalten [27, 28]. Dies erschwert insbesondere eine effiziente Nachverfolgbarkeit von Software-Features zu einzelnen Entwicklungsartefakten [28]. Die Vergabe von Labels in Form von Software-Features für Artefakte ist eine effektive Möglichkeit, um 1) die explizite Benennung von Features zu gewährleisten, 2) die Zuordnung von Features zu Artefakten zu ermöglichen und 3) der Fragmentierung von Features entgegen zu wirken [28].

Eine Fragmentierung von Features ist gegeben, wenn Informationen über Features über verschiedene Software-Artefakte wie spezifizierende Anforderungsdokumente und implementierende Quellcode-Dateien verteilt sind [28]. Dies resultiert häufig daraus, dass unterschiedliche Software-Werkzeuge wie Issue-Tracking-Systeme (ITS) und Versionsverwaltungssysteme (VVS) zur Dokumentation von Anforderungen bzw. zur Versionierung von Quellcode verwendet werden. Gerade bei

großen Projekten mit einer Vielzahl von unterschiedlichen Projektbeteiligten ist die Verwendung von ITS vorteilhaft. In Open-Source-Software (OSS)-Projekten arbeiten oft mehrere Entwickler an verschiedenen Standorten an einer gemeinsamen Codebasis und die Verwendung eines VVS ist daher häufig unerlässlich. Die Nutzung eines ITS wie Jira[1] ist, insbesondere im agilen Projektumfeld, zum Industriestandard geworden[2] und auch OSS-Projekte[3] verwenden ITS. Im Breich der Entwicklung von OSS stellt Git[4] die Basis vieler Code-Hosting-Plattformen[5,6] dar. ITS und VVS bieten oft die Möglichkeit, Labels für die jeweils verwalteten Software-Artefakte zu vergeben.

Jira unterstüzt die Verwaltung von Anforderungen in Form von User Stories. Gerade in der agilen Software-Entwicklung sind User Stories häufig verwendete Artefakte, um die Anforderungen einer Software zu beschreiben [33]. Sie sind oft kurz und in natürlicher Sprache verfasst. Zu ihrer Formulierung werden jedoch meist Satzschablonen eingesetzt, die eine grundlegende Struktur vorgeben [8]. User Stories haben, insbesondere durch die Verwendung von natürlicher Sprache und den nur schwachen Vorgaben für ihre Formulierung, einen informellen Charakter.

Eine Vergabe und konsistente Verwendung von aussagekräftigen Labels ist für Projektbeteiligte nicht immer unproblematisch. Labels stellen eine Abstraktion über den Inhalt eines Artefakts dar. Daher sind sie im Wesentlichen von der Interpretation des Inhalts eines Artefaktes anhängig. Insbesondere bei einer Vielzahl von Projektbeteiligten ist daher eine Vielzahl unterschiedlicher Labels zu erwarten. Auch bei einer ähnlichen Interpretation verhindert eine mögliche Nutzung von Synonymen oder morphologischer Varianten von Wörtern die Vergabe und konsistente Verwendung von Labels.

Weiterhin muss für eine Abstraktion über den Inhalts eines Artefakts zunächst dessen Semantik genau erfasst werden. Dies ist nicht nur bereits für Projektbeteiligte oft komplex, sondern insbesondere auch durch eine voll-automatische Methode schwer umzusetzen: Um die Semantik zu erfassen, muss zunächst der Inhalt analysiert werden. Dazu können Techniken und Verfahren des Natural-Language-Processing (NLP) eingesetzt werden. Häufig benötigen diese jedoch eine komplexe Vorverarbeitung des Artefakts. Modelle aus dem Bereich Machine-Learning (ML) können zur Klassifizierung der Artefakte anhand von Labels verwendet werden. Diese Modelle benötigen für das Training jedoch oft einen umfangreichen Datensatz von Artefakten mit bereits vergebenen Labels, um deren Semantik hinreichend auf den Inhalt der Artefakte abbilden zu können.

[1] `https://www.atlassian.com/software/jira`, abgerufen am 18. Dezember 2018
[2] `https://www.atlassian.com/customers`, abgerufen am 18. Dezember 2018
[3] `https://issues.apache.org/jira/secure/Dashboard.jspa`, abgerufen am 18. Dezember 2018
[4] `https://git-scm.com`, am 18. Dezember 2018
[5] `https://github.com/`, am 18. Dezember 2018
[6] `https://about.gitlab.com`, am 18. Dezember 2018

1.2. Problemstellung und Zielsetzung

In dieser Arbeit wird ein Modell entwickelt, dass eine (semi-)automatische Vergabe von Labels in Form von Feature-Namen für User Stories ermöglicht. Die Arbeit folgt der Annahme, dass eine voll-automatische Vergabe von Feature-Namen für User Stories nicht sinnvoll ist. Dies ist insbesondere der dazu nötigen Abstraktionsleistung und den damit verbundenen Schwierigkeiten geschuldet, die eine voll-automatische Methode schwer leisten bzw. bewältigen kann. Ein Beispiel ist folgende User Story, die aus den verwendeten Datensätzen stammt:

> In meiner Rolle als (System-)Nutzer, möchte ich die Räume auf dem (Gebäude-)Plan in verschiedenen Farben sehen, um schnell zwischen verschiedenen Raumtypen unterscheiden zu können.

Die User Story beschreibt eine Filterfunktion für Räume auf einem Gebäudeplan. Der eigentliche Feature-Name *Filter* kommt in der User Story selbst jedoch nicht vor. Es ist also ein Abstraktionsschritt nötig, um ein Label in Form eines aussagekräftigen Feature-Namens zu definieren. Aus diesem Grund arbeitet das Modell semi-automatisch, d.h es erfordert die Interaktion bzw. Eingaben durch einen Nutzer. Das Modell wird jedoch mit der Absicht erstellt, dem Nutzer eine größtmögliche Unterstützung bei der Vergabe von Feature-Namen für User Stories zu geben. Weiterhin soll eine konsistente Verwendung von bereits vergebenen Feature-Namen erreicht werden. Um den aktuellen Stand der Forschung im Bereich der Vergabe von Labels für natürlichsprachliche Anforderungen festzustellen, wird eine Literaturrecherche durchgeführt. Das Modell wird prototypisch als Plugin für Jira implementiert und anhand der Metriken Precision, Recall und F1-Maß evaluiert.

1.3. Aufbau der Arbeit

Die für das Verständnis dieser Arbeit nötigen Grundlagen werden in Kapitel 2 eingeführt. In Kapitel 3 der Arbeit werden die Methodik und die Ergebnisse der durchgeführten Literaturrecherche beschrieben. Das in dieser Arbeit entwickelte Modell zur (semi-)automatischen Vergabe von Feature-Namen für User Stories wird in Kapitel 4 vorgestellt. In Kapitel 5 wird die prototypische Implementierung des entwickelten Modells beschrieben. Die Evaluation des entwickelten Modells erfolgt in Kapitel 6. Kapitel 7 fasst schließlich die Ergebnisse der vorliegenden Arbeit zusammen und gibt einen Ausblick auf weitere Arbeiten.

2. Grundlagen

In diesem Kapitel werden die für das Verständnis der vorliegenden Arbeit notwendigen Grundlagen beschrieben. In Abschnitt 2.1 werden User Stories und die zu ihrer Formulierung genutzten Satzschablonen beschrieben. Abschnitt 2.2 definiert die Begriffe Software-Feature und feature-relevante Information im Kontext dieser Arbeit. Grundlegende Techniken und weiterführende Verfahren aus dem Bereich des NLP, die für diese Arbeit wichtig sind, werden in Abschnitt 2.3 vorgestellt. Abschnitt 2.4 beschreibt künstliche neuronale Netze (KNN), ein Klassifikationsmodell aus dem Bereich des ML, das im Rahmen dieser Arbeit verwendet wird.

2.1. User Stories und Satzschablonen

User Stories sind informelle Anforderungsdokumente, die die Funktionalität einer Software aus Sicht der Projektbeteiligten, insbesondere der Nutzer, beschreiben. Sie sind oft in natürlicher Sprache und auf einem niedrigen Abstraktionsgrad geschrieben [33], d.h. sie geben spezifische und detailierte Informationen über konkrete Software-Funktionen wieder. User Stories können im Rahmen der Software-Entwicklung z.B. für das Requirements-Engineering oder im Release-Planning eingesetzt werden. Häufig werden User Stories mithilfe einer Satzschablone formuliert [33]. Eine oft verwendete Satzschablone ist das Connextra-Template [8]:

I as a (role), want (function), so that (business value).

Das Connextra-Template teilt eine User Story somit in drei Abschnitte: Der erste Abschnitt beschreibt eine Rolle oder Persona. In diesem Abschnitt wird also die Frage nach dem *Wer* beantwortet. Im zweiten Abschnitt wird eine Funktionalität, Aktivität oder Handlung beschrieben, die durch die Rolle oder Persona ausgeführt wird. Dieser Abschnitt beantwortet somit die Frage nach dem *Was*. Der dritte Abschnitt beschreibt das Ziel, die Absicht oder den Grund, warum die Funktionalität benötigt bzw. warum die Aktivität oder Handlung ausgeführt wird. Dieser Abschnitt beantwortet also die Frage nach dem *Warum*. Das Connextra-Template kann je nach Projektkontext angepasst werden. So wurden die User Stories aus den in dieser Arbeit genutzten Datensätzen mit der folgenden Satzschablone formuliert:

As a (role), i want to (function), in order to (business value).

Diese Satzschablone ist im Vergleich zum Connextra-Template leicht abgewandelt. Dennoch beschreibt sie dieselbe Aufteilung einer User Story in drei Abschnitte. Im Folgenden werden die Begriffe *Nutzer* (das Wer), *Aktivität* (das Was) und *Ziel* (das Warum) benutzt, um die jeweiligen Abschnitte der User Stories zu referenzieren. Abbildung 2.1 zeigt eine User Story aus den in dieser Arbeit verwendeten Datensätzen. Nach obiger Definition ist der Nutzer *technical user*, die Aktivität *use a function on all routes* und das Ziel *log all actions made via these endpoints*.

As a technical user, i want to use a function on all routes, in order to log all actions made via these endpoints.

Abbildung 2.1.: Beispiel für eine User Story aus den verwendeten Datensätzen.

2.2. Software-Features und feature-relevante Information

Innerhalb der Literatur existieren unterschiedliche Definitionen für den Begriff Software-Feature [27, 12, 7, 15, 5, 6]. Im Folgenden werden einige dieser Definitionen vorgestellt, um die Variabilität und Gewichtung einzelner Aspekte innerhalb der Definitionen zu zeigen. Danach wird die im Rahmen dieser Arbeit genutzte Definition der Begriffe Software-Feature und feature-relevante Information vorgestellt.

Das Standard-Glossar für Software-Terminologie [12] des Institute of Electrical and Electronics Engineers (IEEE) enthält die folgende Definition des Begriffs Software-Feature:

„A software characteristic specified or implied by requirements documentation (for example, functionality, performance, attributes, or design constraints).“

Nach dieser Definition ist ein Software-Feature eine Eigenschaft einer Software bzgl. bestimmter Anforderungen. Darunter fallen sowohl die eigentlichen funktionalen Anforderungen als auch Qualitätsanforderungen wie Leistung bzw. Effizienz. Classen et al. [7] haben in einer Übersicht weitere Definitionen aus der Literatur zusammengestellt, von denen einige im Folgenden vorgestellt werden: So definieren Kang et al. [15] ein Software-Feature als Abstraktion einer Funktionalität einer Software. Die Autoren betonen im Vergleich zu den noch folgenden Definitionen den Software-Lebenszyklus eines Features:

„Distinctively identifiable functional abstractions that must be implemented, tested, delivered, and maintained.“

Bosch et al. [5] definieren ein Software-Feature als das Verhalten einer Software bzgl. funktionaler und nicht-funktionaler Anforderungen. Ähnlich zur Definition des IEEE [12] wird ein Software-Feature dabei auf eine Menge von Anforderungen bezogen:

„A logical unit of behaviour specified by a set of functional and non-functional requirements.“

In Chen et al. [6] wird ein Software-Feature als die Eigenschaft eines Software-Produkts aus Nutzer- oder Kundensicht bezeichnet. Anders als die vorherigen Definitionen stellen die Autoren den Aspekt der Nutzer- bzw. Kundenorientierung eines Software-Features in den Vordergrund. Ähnlich zu den Definitionen des IEEE [12] und Bosch et al. [5] wird auch in dieser Definition ein Software-Feature in Bezug auf eine Menge von Anforderungen beschrieben:

„A product characteristic from user or customer views, which essentially consists of a cohesive set of individual requirements.“

Im Kontext dieser Arbeit wird die Definition eines Software-Features aus Quirchmayr et al. [27] übernommen, welche sich nach Aussage der Autoren u.A. auf die Defintion von Bosch et al. [5] bezieht:

„A feature describes an abstract unit of behaviour of a software system at a high level and bundles atomic information, which describe the unit of behaviour at a detail level. The latter is called atomic unit of feature-relevant information.“

Nach dieser Definition beschreibt ein Software-Feature ein bestimmtes Verhalten einer Software auf einem hohen Abstraktionsgrad, indem es eine Menge von feature-relevanten Informationen bündelt. Feature-relevante Informationen sind atomare Informationsbausteine, die ein bestimmtes (Teil-)Verhalten einer Software auf einem niedrigen Abstraktionsgrad beschreiben. Die Definition von Quirchmayr et al. [27] wurde gewählt, da sie die in dieser Arbeit behandelte Problemstellung treffend beschreibt: Wie in Abschnitt 2.1 bereits erläutert, beschreibt eine User Story Funktionalität einer Software auf einem niedrigen Abstraktionsgrad. Somit enthält sie feature-relevante Information. Ein zu vergebener Feature-Name für eine User Story beschreibt ein Software-Feature. Der zu vergebene Feature-Name bündelt daher feature-relevante Informationen aus User Stories und abstrahiert über die in den User Stories beschriebene Funktionalität. Im Rahmen dieser Arbeit werden Feature-Namen in Form von Labels für User Stories vergeben. Ein zu vergebenes Label repräsentiert damit ein Software-Feature. Feature-Labels erlauben somit eine Zuordnung von Software-Features zu einzelnen User Stories.

2.3. Techniken und Verfahren des NLP

Im Folgenden werden Techniken und Verfahren aus dem Bereich des NLP vorgestellt, die im Rahmen dieser Arbeit verwendet werden. Die Anwendung der einzelnen Techniken und Verfahren wird iterativ anhand der User Story aus dem Beispiel in Abbildung 2.1 verdeutlicht. Für die Beschreibung der Techniken und Verfahren werden die jeweiligen englischen Begriffe genutzt.

Sentence-Segmentation

Sentence-Segmentation umfasst das Identifizieren und Auftrennen einzelner Sätzen innerhalb von Dokumenten. Sätze können z.B. anhand bestimmter Satzzeichen wie Punkten oder Ausrufezeichen erkannt werden [14]. Aktuelle Verfahren für Sentence-Segmentation basieren dagegen auf ML-Modellen [14]. Es ist anzumerken, dass die User Story in Abbildung 2.1 bereits nur aus einem Satz besteht, weswegen eine andere User Story aus den verwendeten Datensätzen als Beispiel dient. Diese ist in Abbildung 2.2 einmal vor und einmal nach Durchführung des Sentence-Segmentation-Schrittes dargestellt. Es ist zu erkennen, dass die User Story anhand des Punktes am Satzende in zwei Teilsätze aufgetrennt wurde. Der Trennstrich - repräsentiert dabei die Satzgrenzen.

As a technical user with certain access level, i want to POST keywords of an organisation, in order to search and find organisations. The output should be JSON.

Abbildung 2.2a: User Story vor dem Sentence-Segmentation-Schritt.

As a technical user with certain access level, i want to POST keywords of an organisation, in order to search and find organisations. — The output should be JSON.

Abbildung 2.2b: User Story nach dem Sentence-Segmentation-Schritt.

Tokenization

Tokenization bezeichnet das Identifizieren und Auftrennen einzelner Tokens innerhalb eines Dokuments [18]. Ein Token bezeichnet eine konkrete Reihung von Schriftzeichen, die in ihrer Gesamtheit eine atomare Einheit darstellen [18]. Zum Beispiel stellen die einzelnen Wörter eines Dokuments Token dar. Je nach Anwendungsfall können unter einem Token aber auch die einzelnen Satz- und Sonderzeichnen verstanden werden. Abbildung 2.3 zeigt die User aus Abbildung 2.1 nachdem

sie in ihre einzelnen Tokens aufgetrennt wurde. Die Trennstriche repräsentieren dabei die Grenzen der einzelnen Tokens. Es ist anzumerken, dass auch der Punkt ein Token darstellt.

as — a — technical — user — , — i — want — to — use — a — function — on — all — routes — , — in — order — to — log — all — actions — made — via — these — endpoints — .

Abbildung 2.3.: User Story nach dem Tokenization-Schritt.

Stopword-Removal

Als Stoppwörter werden Wörter bezeichnet, die keinen Beitrag zur Erfassung des semantischen Inhalts eines Dokuments liefern oder sehr häufig in einem Dokument vorkommen. Stoppwörter sind oft unspezifisch und Erfüllen bestimmte grammatikalische Funktionen. Typische Arten von Stoppwörtern sind z.B. Artikel, Präpositionen und Konjunktionen. Aus diesen Gründen werden im NLP Stoppwörter häufig aus Dokumenten entfernt. Im Kontext von User Stories sind mögliche Stoppwörter z.B. die Wörter der Satzschablone (*as*, *an*, *a*, *i*, *want*, *in*, *order* und *to*). Diese kommen in jeder User Story der verwendeten Datensätze vor und haben keine Aussagekraft bzgl. der Semantik einer User Story. Es wird angenommen, dass in diesem Fall die Präpositionen bzw. Pronomen *on*, *all*, *via* und *these* weitere Stoppwörter sind. Diese kommen im Englischen ebenfalls häufig vor und helfen daher nicht beim Erfassen der Semantik der User Story. Dies gilt ebenso für die Satzzeichen wie den Punkt am Satzende und Kommata. Abbildung 2.4 zeigt die User Story aus dem Beispiel 2.1 nach dem Entfernen der Stoppwörter.

technical — user — use — function — routes — log — actions — made — endpoints

Abbildung 2.4.: User Story nach dem Stopword-Removal.

Stemming und Lemmatization

Stemming und Lemmatization bezeichnet das Rückführen von flektierten Wörtern auf ihrem Wortstamm (Stemming) bzw. ihre Wörterbuchform (Lemma) [18]. Die Flexion von Wörtern umfasst u.A. die Deklination von Substantiven und die Konjugation von Verben. So können z.B. im Fall von *(der) Inhalt* (Nominativ) und *(des) Inhalts* (Genitiv) die Substantive *Inhalt* und *Inhalts* auf den gemeinsamen Wortstamm *Inhalt* zurückgeführt werden. Im Fall von *(ich) gehe* (1. Person Singular) und *(du) gehst* (2. Person Singular) können die Verben *gehe* und *gehst* auf den gemeinsamen Wortstamm *geh* zurückgeführt werden.

Stemming und Lemmatization unterscheiden sich jedoch in ihrer Herangehensweise: Stemming bezeichnet ein eher informelles Verfahren, das mithilfe von heuristischen Regeln die Affixe von Wörtern abtrennt. Lemmatization dagegen beschreibt ein eher formelles Verfahren, das mithilfe von morphologischer Analyse und dem Referenzieren von Wörterbüchern arbeitet [18]. Dadurch ist es mithilfe von Lemmatization z.B. auch möglich unregelmäßige Steigerungsformen von Adjektiven wie *besser* und *am besten* auf ihre Wörterbuchform *gut* zurückzuführen, während dies mit Stemming nicht möglich ist. Ein Stemming-Verfahren würde in diesem Fall eher den Wortstamm *bes* identifizieren. Der Grund dafür ist, dass ein Lemmatization-Verfahren durch das Nachschlagen von *besser* in einem Wörterbuch die gemeinsame Wörterbuchform *gut* identifizieren kann. Ein Stemming-Verfahren kann dagegen nur auf dem Wort *besser* selbst arbeiten. Häufig verwendet wird im NLP-Bereich z.B. der Stemming-Algorithmus von Porter [26]. Im Falle der User Story aus dem Beispiel 2.1 können unter Einsatz eines Stemming-Verfahrens durch Abtrennen der Affixe die Wörter *user* und *use* auf den gemeinsamen Wortstamm *use* abgebildet werden. Die Wörter *technical* und *made* können auf ihre jeweiligen Wortstämme *technic* und *made* zurückgeführt werden. Die Wörter *routes*, *actions* und *endpoints* stellen Substantive im Plural dar und können daher auf die jeweiligen Singularformen *route*, *action* und *endpoint* zurückgeführt werden. Die resultierende User Story ist in Abbildung 2.5 dargestellt.

technic — use — use — function — route — log — action — made — endpoint

Abbildung 2.5.: User Story nach dem Stemming-Schritt.

Wird ein Lemmatization-Verfahren auf die User Story aus dem Beispiel 2.1 angewandt, sieht das Ergebnis in Abbildung 2.6 sehr ähnlich zu dem Ergebnis des Stemming-Verfahrens in Abbildung 2.5 aus. Ein Unterschied besteht jedoch darin, dass das Lemmatization-Verfahren durch das Referenzieren eines Wörterbuchs das Wort *made* auf seine Wörterbuchform *make* zurückführen kann.

technic — use — use — function — route — log — action — make — enpoint

Abbildung 2.6.: User Story nach dem Lemmatization-Schritt.

Termvektoren, tf-idf-Gewichtungsschema und Kosinus-Ähnlichkeit

Für viele Probleme innerhalb des NLP ist es sinnvoll, natürlichsprachliche Dokumente in reellwertige Vektoren zu überführen. So sind z.B. die Eingaben in viele Klassifikationsmodelle aus dem Bereich des ML reellwertige Vektoren. Ebenfalls ist die Kosinus-Ähnlichkeit, ein Ähnlichkeitsmaß, das zur Quantifizierung der semantischen Ähnlichkeit zweier natürlichsprachlicher Dokumente genutzt werden kann, anhand von zwei reellwertigen Vektoren a und b definiert:

$$cos\text{-}sim(a, b) = \frac{a \cdot b}{||a||\ ||b||} \tag{2.1}$$

Termvektoren sind eine Möglichkeit, natürlichsprachliche Dokumente als reellwertige Vektoren zu repräsentieren. Die Einträge eines Termvektors repräsentieren dabei die einzelnen Terme in einem Dokument. Der jeweilige Wert eines Eintrags gibt die semantische Wichtigkeit dieses Terms innerhalb des Dokuments an. Es ist wichtig, zwischen einem Term und einem Token zu unterscheiden. Wie bereits in Abschnitt 2.3 beschrieben sind Token die einzelnen Wörter oder Satzzeichen, wie sie in einem Dokument auftreten. Ein Term dagegen repräsentiert eine Menge von Token. Kommt z.B. das Token *System* häufig in einem Dokument vor, werden alle in dem Dokument vorkommenden Token durch den einzelnen Term *System* repräsentiert. Im Rahmen der Berechnung von Termvektoren ist es ebenfalls sinnvoll, semantisch ähnliche Token durch einen einzelnen Term zu repräsentieren. So können z.B. die beiden Token *System-Administrator* und *Admin* durch den einzelnen Term *Administrator* repräsentiert werden.

Zur Berechnung von Termvektoren kann das Term-Frequency-Inverse-Document-Frequency (tf-idf)-Gewichtungsschema genutzt werden. Es existieren eine Reihe von verschiedenen Definitionen für tf-idf-Gewichte [18]. Im Folgenden wird diese Definition vorgestellt: Der Wert $tf\text{-}idf_{t,d}$ des Terms t im Termvektor eines Dokuments d berechnet sich nach der Formel 2.2. Hierbei stellt $tf_{t,d}$ die Termfrequenz eines Terms t in einem Dokument d dar. Wie in Formel 2.3 dargestellt, gibt die Termfrequenz $tf_{t,d}$ also an, wie oft ein Term t innerhalb eines Dokuments d vorkommt. Je häufiger der Term vorkommt, desto höher ist damit auch sein Wert innerhalb des Termvektors und damit das Maß für seine semantische Wichtigkeit. Der zweite Teil der Formel idf_t bezieht sich auf die inverse Dokumentenfrequenz eines Terms t innerhalb einer Dokumentensammlung. Die inverse Dokumentenfrequenz wird nach der Formel 2.4 berechnet. Hierbei gibt die Dokumentenfrequenz d_t an, in wie vielen Dokumenten einer Dokumentensammlung der Term t vorkommt. Die Anzahl der Dokumente in der Sammlung wird durch N bezeichnet. An der Formel 2.4 kann man also erkennen, dass je höher die Dokumentenfrequenz d_t eines Terms t ist, desto geringer ist seine inverse Dokumentenfrequenz idf_t. Damit wird der Wert eines Terms t im Termvektors und somit das Maß für seine semantische Wichtigkeit umso geringer, je mehr Dokumente innerhalb einer Sammlung den Term t beinhalten.

$$tf\text{-}idf_{t,d} = tf_{t,d} \cdot idf_t \tag{2.2}$$

$$tf_{t,d} = |\{t_i \mid t_i \in d \wedge t_i = t\}| \tag{2.3}$$

$$idf_t = log_e\left(1 + \frac{N}{d_t}\right) \tag{2.4}$$

Im Folgenden wird die Berechnung von Termvektoren mit tf-idf-Gewichten und die Berechnung der Kosinus-Ähnlichkeit demonstriert. Dazu werden die mithilfe der bisher vorgestellten NLP-Techniken und -Verfahren vorverarbeiteten User Stories in Abbildung 2.7 als Beispiel genutzt. Die erste User Story entspricht dabei der User Story aus Abbildung 2.5. Die zweite User Story ist eine fiktive User Story, die ähnlich zu den User Stories aus den verwendeten Datensätzen ist.

technic — use — use — function — route — log — action — made — endpoint

Abbildung 2.7a: Vorverarbeitete User Story d_1.

admin — use — function — log — profile — access

Abbildung 2.7b: Vorverarbeitete User Story d_2.

In Tabelle 2.1 sind die jeweiligen Termfrequenzen $tf_{t,d}$ für die Terme der User Stories in Abbildung 2.7 dargestellt. Tabelle 2.2 zeigt die jeweiligen Dokumentenfrequenzen df_t sowie die resultierenden inversen Dokumentenfrequenzen idf_t unter der Annahme $N = 2$. In Tabelle 2.3 werden die resultierenden Termvektoren für die beiden User Stories mit dem tf-idf-Gewichtungsschema dargestellt. Anhand der Termvektoren der User Stories kann deren Kosinus-Ähnlichkeit nach Formel 2.1 berechnet werden. Diese ergibt sich zu $cos\text{-}sim(d_1, d_2) \approx 0.28$. Sind die Werte der einzelnen Einträge a_i, b_j der Vektoren a, b positiv, so besitzt die Kosinus-Ähnlichkeit nach Formel 2.1 einen Wertebereich von $[0, 1]$. Der errechnete Wert liegt somit am unteren Ende des Wertebereichs. Daher kann interpretiert werden, dass die User Stories eine geringe semantische Ähnlichkeit besitzen.

$tf_{t,d}$	technic	admin	use	function	route	log	action	made	endpoint	profile	access
d_1	1	0	2	1	1	1	1	1	1	0	0
d_2	0	1	1	1	0	1	0	0	0	1	1

Tabelle 2.1.: Termfrequenzen $tf_{t,d}$ der User Stories in Abbildung 2.7.

	technic	admin	use	function	route	log	action	made	endpoint	profile	access
df_t	1	1	2	2	1	2	1	1	1	1	1
idf_t	1.1	1.1	0.69	0.69	1.1	0.69	1.1	1.1	1.1	1.1	1.1

Tabelle 2.2.: Dokumentenfrequenzen df_t der User Stories in Abbildung 2.7.

$tf\text{-}idf_{t,d}$	technic	admin	use	function	route	log	action	made	endpoint	profile	access
d_1	1.1	0	1.38	0.69	1.1	0.69	1.1	1.1	1.1	0	0
d_2	0	1.1	0.69	0.69	0	0.69	0	0	0	1.1	1.1

Tabelle 2.3.: tf-idf-Gewichte der User Stories in Abbildung 2.7.

Named-Entity-Recognition

Mit Named-Entity-Recognition (NER) wird das Identifizieren von Textbestandteilen bezeichnet, die eine Entität im Sinne eines Eigennamens beschreiben. Die identifizierten Textbestandteile werden häufig mit einem Label versehen, das eine Klassifizierung der Entitäten erlaubt [14]. Beispiele für solche Labels sind u.A. *PER*, *ORG* und *LOC* [14]. Diese Label beschreiben jeweils Eigennamen für Personen, Organisationen und Orte. In dem Satz *Heidelberg ist eine Stadt in Baden-Württemberg*, würden die Token *Heidelberg* und *Baden-Württemberg* mit dem Label *LOC* versehen. Es ist anzumerken, dass in der User Story aus dem Beispiel 2.1 keine Eigennamen vorkommen. Aus diesem Grund wird das NER-Verfahren anhand der fiktiven User Story in Abbildung 2.8 demonstriert. In diesem Fall wurde *Germany* als ein Ort identifiziert und mit dem Label *LOC* versehen.

As a user, i want to view the result of the search for a city name on the map of Germany [LOC], in order to locate a searched city.

Abbildung 2.8.: User Story mit vergebenen NER-Tags.

Part-Of-Speech-Tagging

Part-Of-Speech (POS)-Tagging bezeichnet das Vergeben von POS-Tags für Token in einem Dokument [14]. POS-Tags beschreiben Wortarten und grammatikalische Klassen von Token [14]. Eine im Bereich des NLP häufig verwendete Datenbank, die Definitionen von POS-Tags enthält, ist die Penn-Treebank [20]. Beispiele für Labels aus der Penn-Treebank sind *JJ* und *NN*, welche jeweils ein Adjektiv bzw. ein Nomen repräsentieren. Abbildung 2.9 zeigt die User Story aus Beispiel 2.1 nach Anwendung des POS-Tagging. Zu erkennen ist, dass z.B. *technical* ein Verb und *user* ein Nomen ist. Für die Bedeutung der übrigen Labels sei auf den Anhang B.1 verwiesen.

As [IN] a [DT] technical [JJ] user [NN] , [,] i [PRP] want [VPB] to [TO] use [VB] a [DT] function [NN] on [IN] all [DT] routes [NNS] , [,] in [IN] order [NN] to [TO] log [VB] all [DT] actions [NNS] made [VBD] via [IN] these [DT] endpoints [NNS] . [.]

Abbildung 2.9.: User Story mit vergebenen POS-Tags.

Syntaktisches Parsen

Syntaktisches Parsen bezeichnet das Identifizieren eines Satzes und seiner syntaktischen Struktur innerhalb eines Dokuments [14]. Die Ausgabe eines syntaktischen Parsers ist häufig ein Syntaxbaum, welcher die grammatikalische Struktur je eines Satzes aus dem Eingabedokument darstellt. Abbildung 2.10 zeigt den geparsten Syntaxbaum der User Story in Abbildung 2.1. Die Knoten und Blätter des Baumes enthalten Labels, die der Parser den jeweiligen Satzbestandteilen bzw. Token zugeordnet hat. Die Blätter enthalten wiederum einen Verweis auf das eigentliche Token, das sie repräsentieren. Die vergebenen Labels sind der Penn-Treebank [20, 29] entnommen. Neben den im vorherigen Abschnitt beschriebenen Definitionen für Labels, die eine Wortart repräsentieren, enthält die Penn-Treebank auch weiterführende Labels, die eine syntaktische Einheit beschreiben. Beispiele für solche Labels sind z.B. *PP* und *VP*, welche jeweils eine Präpositionalphrase bzw. eine Verbalphrase definieren. Für eine Übersicht der Labels für syntaktische Einheiten sei auf den Anhang B.2 verwiesen.

Die Labels der Blätter des Baumes entsprechen immer einem POS-Tag, das die Wortart des zugehörigen Tokens beschreibt. So ist z.B. an den rot umrandeten Blättern des Baumes in 2.10 zu erkennen, dass das Blatt mit dem Label *JJ* und dem zugehörigen Token *technical* ein Adjektiv repräsentiert. Analog stellt das Blatt mit dem Label *NN* und dem zugehörigen Token *user* ein Nomen dar. Die Labels der Knoten entsprechen immer der syntaktischen Einheit, die der zugehörige Satzbestandteil darstellt. So ist z.B. an den rot gestrichelten Knoten zu erkennen, dass der Knoten mit dem Label *PP* und dem zugehörigen Satzbestandteil *on all routes* eine Präpositionalphrase repräsentiert. Analog stellt der Knoten mit dem Label *VP* und dem zugehörigen Satzbestandteil *log all actions made via these endpoints* eine Verbalphrase dar.

Der Unterschied von syntaktischem Parsen und POS-Tagging ist, dass beim syntaktischen Parsen eine tiefgehende hierarchische Darstellung der syntaktischen Struktur eines Satzes eines Dokuments erzeugt wird, während beim POS-Tagging den einzelnen Token des Dokuments eine Wortart bzw. grammatikalische Klasse zugeordnet wird.

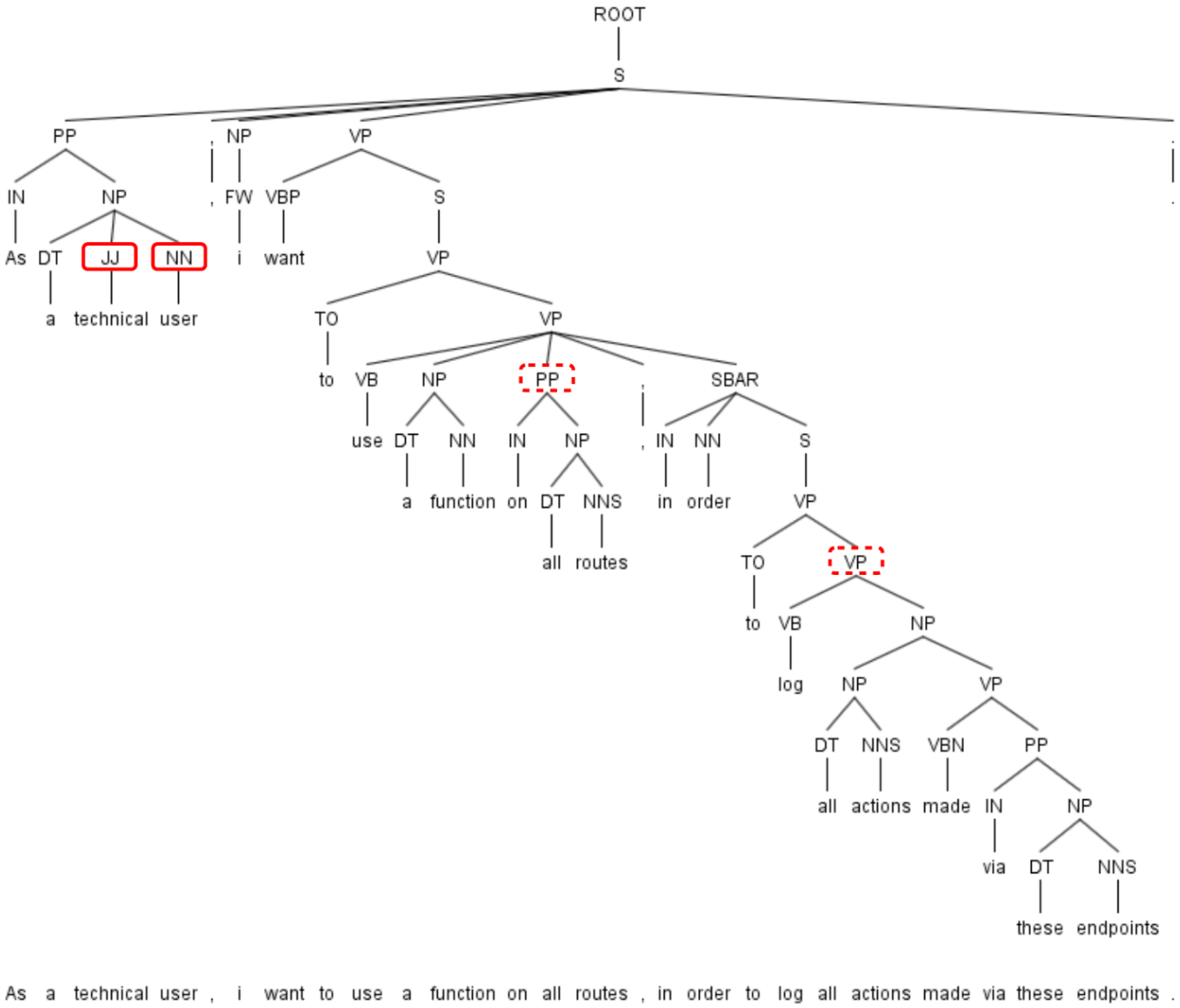

Abbildung 2.10.: Syntaxbaum einer User Story.

2.4. Künstliche neuronale Netze

Im folgenden Abschnitt werden Klassifikationsprobleme und die zur ihrer Lösung eingesetzten KNN aus dem Bereich des ML beschrieben. Ein Klassifikationsproblem kann auf die folgende Art und Weise definiert werden: Gegeben ist eine Menge von Eingabedaten X und eine Menge von Klassen Y. Gesucht ist dann eine Funktion $f : X \to Y$. Die gesuchte Funktion ordnet also einer Eingabe $x_i \in X$ eine Klasse $y_j \in Y$ zu. Die Eingabe x_i ist typischerweise als reellwertiger Vektor $x_i = (x_1, x_2, ..., x_n)$ definiert. Ein Beispiel für ein Klassifikationsproblem ist das Folgende: Gegeben sind zwei Eingabedaten $x_1 = (1.85, 80)$, $x_2 = (1.65, 60)$ und eine Menge von Klassen $Y = \{♂, ♀\}$. Für die gesuchte Funktion muss dann $f(x_1) = ♂$ und $f(x_2) = ♀$ gelten.

Ein KNN ist ein ML-Modell, das zur Lösung eines Klassifikationsproblems verwendet werden kann. Es gibt verschiedene Architekturen für KNN, die sich in ihrem Aufbau unterscheiden. Im Folgenden soll das klassische Feed-Forward-Netz beschrieben werden. Ein KNN des Feed-Forward-Typs besteht im Allgemeinen aus einer Menge von Knoten, die in Schichten angeordnet sind. Die Anzahl der Knoten einer Schicht ist grundsätzlich beliebig. Typischweise gibt es mindestens eine Eingabe- und Ausgabeschicht von Knoten. Dazwischen können je nach Aufbau des Netzes zusätzliche Schichten (versteckte Schichten) liegen. Jeder Knoten einer Schicht ist mit jedem Knoten der darauf folgenden Schicht durch gewichtete Kanten verbunden [4]. Abbildung 2.11 zeigt ein KNN des Feed-Forward-Typs mit einer Eingabeschicht von zwei Knoten, einer Ausgabeschicht mit einem Knoten und einer versteckten Schicht mit zwei Knoten. Die Eingabeschicht besteht aus den Knoten x_i. Die Knoten z_j bilden die Knoten der versteckten Schicht. Die Ausgabeschicht besteht aus den Knoten y_k. Die Gewichte der Kanten zwischen den Knoten der Eingabeschicht und jenen der versteckten Schicht sind mit $w_{i,j}$ bezeichnet. Die Gewichte der Kanten zwischen den Knoten der versteckten Schicht und jenen der Ausgabeschicht $v_{j,k}$.

Der Bezug zum Klassifikationsproblem ist der Folgende: Die Eingabeschicht nimmt einen reellwertigen Vektor x auf: Jeder Knoten der Eingabeschicht repräsentiert dabei einen Wert x_i des Eingabevektors x. Die Eingabe eines Vektors x produziert eine Ausgabe an den Knoten y_k der Ausgabeschicht. Die Werte der Knoten in der Ausgabeschicht ergeben sich dabei zu $y_k = \sum_j v_{j,k} \cdot h(z_j)$, wobei die Werte der versteckten Schicht nach $z_j = \sum_i w_{i,j} \cdot h(x_i)$ berechnet werden. Der Wert eines jeden Knotens (außer derer in der Eingabeschicht) ist somit die gewichtete Summe aus den Werten all seiner Vorgängerknoten. Der eigentliche Ausgabewert eines Knotens y_k kann dann als eine Klasse codiert werden. So kann z.B. definiert werden, dass ein Ausgabewert $y_k < 0$ die Klasse ♂ repräsentiert und ein Ausgabewert $y_k > 0$ die Klasse ♀. In den oben genannten Berechnungsvorschriften ist die Funktion $h(x)$ auffällig. Diese Funktion wird als Aktivierungsfunktion bezeichnet und transformiert den Wert eines Knotens, bevor dieser in den Berechnungen der

nachfolgende Schichten verwendet wird. Von der Wahl einer passenden Aktivierungsfunktion wird die Klassifikationsleistung eines KNN erheblich beeinflusst, weshalb eine Reihe unterschiedlicher Aktivierungsfunktionen wie die Softmax-, RELU- und Sigmoid-Funktion existieren [24].

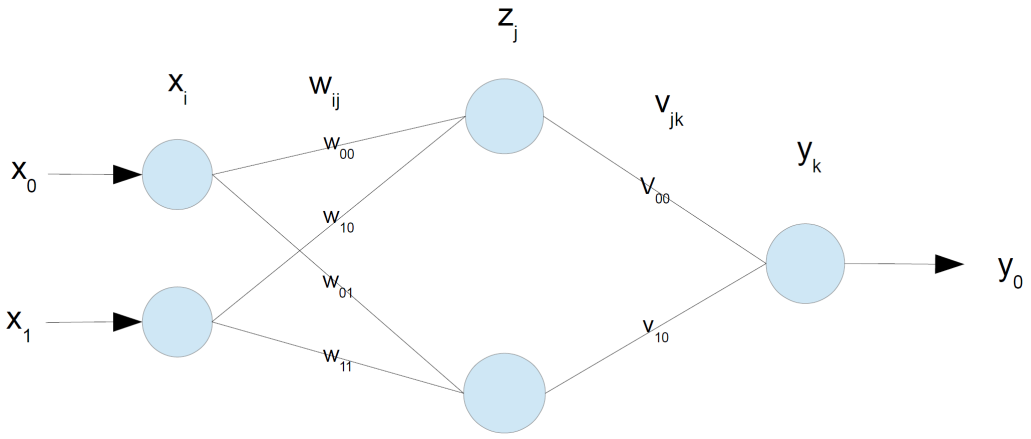

Abbildung 2.11.: Darstellung eines künstlichen neuronalen Netzes.

Gegeben ist die Menge von Eingabedaten X und Klassen Y mit einer bekannten Zuordnung aller Eingaben zu ihren jeweiligen Klassen $f(x) = y$. Da Eingabewerte und die jeweiligen Ausgabewerte bekannt sind, können die Gewichte des KNN so angepasst werden, dass bei jeder Eingabe der gewünschte Ausgabewert, der die jeweilige Klase codiert, möglichst erreicht wird. Dieser Prozess wird als Training eines KNN bezeichnet. Die Menge der Eingabedaten X, Klassen Y und die jeweiligen Zuordnungen $f(x) = y$ werden als Trainingsdatensatz bezeichnet. Für die Anpassung der Gewichte gibt es eine Reihe von mathematischen Verfahren, die im Kontext dieser Arbeit nicht weiter erklärt werden. Oft verwendet werden jedoch numerische Optimierungsmethoden wie die Gradient-Descent-Methode [4]. Das Ziel des Trainings ist es, die inhärenten Muster innerhalb der Daten des Trainingsdatensatzes durch die Werte der Kantengewichte abzubilden. Somit kann das KNN nach dem Training genutzt werden, um unbekannten Eingaben anhand ihres berechneten Ausgabewertes eine Klasse zuzuordnen.

3. Literaturrecherche

Die Literaturrecherche wurde durchgeführt, um den allgemeinen Stand der Forschung im Bereich der Vergabe von Labels für natürlichsprachliche Anforderungen festzustellen. Für die Entwicklung eines Ansatzes zur Lösung der Problemstellung dieser Arbeit ist es erforderlich, eine Übersicht bestehender Ansätze aus der Literatur zu haben, um deren Anwendbarkeit und Nutzen im Kontext dieser Arbeit beurteilen zu können. Es ist zu erwarten, dass manche Ansätze in diesem Punkt zweckdienlicher sind als andere. Weiterhin ist es erforderlich, geeignete Techniken und Methoden zu identifizieren, die den entwickelten Ansatz realisieren können: Werden z.B. ML-Modelle verwendet, müssen eventuell erst umfangreiche Datensätze vorhanden sein, um die Modelle zu trainieren. Der in dieser Arbeit entwickelte Ansatz ist aber auf einer beliebigen Menge von User Stories anwendbar. Diese Menge kann unter Umständen nur wenige User Stories umfassen. In diesem Fall sind z.B. Ansätze, die auf der Extraktion von Termen basieren, geeigneter als ML-Modelle. Zudem sind User Stories oft informell und kurz gehalten. Weiterhin werden sie häufig mithilfe von Satzschablonen formuliert. Somit kann es sein, dass bestehende Ansätze und Techniken nicht für die Nutzung mit User Stories adaptiert werden können. Notwendigerweise muss ein zu entwickelnder Ansatz die besonderen Eigenschaften von User Stories berücksichtigen. Aus diesem Kontext ergibt sich die Forschungsfrage 3.1, welche im Zuge der Literaturrecherche zu beantworten ist:

FF: Welche Ansätze, insbesondere welche Techniken, zur semi-automatisierten Vergabe von Labels für eine Menge von Anforderungen existieren? (3.1)

Das Kapitel zur Literaturrecherche ist wie folgt aufgebaut: In Abschnitt 3.1 wird zunächst die Methodik der Literaturrecherche beschrieben. Im Anschluss erfolgt in Abschnitt 3.2 die Vorstellung der Suchtreffer. Diese werden in Abschnitt 3.3 anhand formaler Kriterien verglichen. In Abschnitt 3.4 erfolgt ein Vergleich bzgl. der Forschungsfrage. In Abschnitt 3.5 werden einige Besonderheiten der Ansätze und Techniken der Suchtreffer diskutiert. Abschließend werden in Abschnitt 3.6 die Erkenntnisse aus der Literaturrecherche vorgestellt.

3.1. Methodik der Literaturrecherche

In Abbildung 3.1 ist der generelle Ablauf der Literaturrecherche dargestellt. Zunächst wurden unter Berücksichtigung des Kontextes dieser Arbeit und der Zielsetzung der Literaturrecherche die zu beantwortende Forschungsfrage 3.1 definiert. Die Recherche lief dann in zwei Phasen ab, der informellen Vorsuche und der formellen Suche. Die informelle Vorsuche diente dabei als Vorbereitung der formellen Suche. Die Ergebnisse der informellen Vorsuche sind nicht dokumentiert, jedoch baut die formelle Suche auf diesen auf. Die formelle Suche lieferte als Ergebnis die in dieser Arbeit genutzten Literaturquellen und ist in Anhang A dokumentiert.

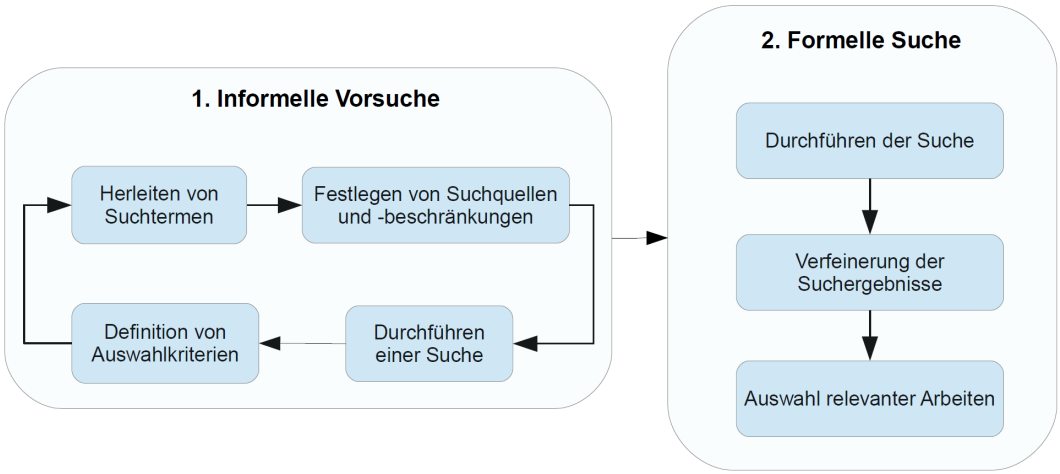

Abbildung 3.1.: Ablauf der Literaturrecherche.

Während der Vorsuche wurden die folgenden Schritte iterativ durchgeführt: Herleiten von Suchtermen, Festlegen von Suchquellen und -beschränkungen und Definition von Auswahlkriterien für Suchtreffer nach Sichtung der jeweiligen Suchergebnisse. Im Verlauf der Vorsuche wurden so die gebräuchlichen Begriffe des Themengebiets und mögliche Suchtreffer identifiziert. Mithilfe dieser konnte eingeschätzt werden, welche Suchbegriffe geeignet sind, um relevante Suchergebnisse bzw. Suchtreffer zu liefern. Während der Vorsuche wurden ebenfalls verschiedene Suchquellen und -beschränkungen berücksichtigt, um deren Eignung für das Gewinnen von Suchtreffern zu beurteilen. Die während der Vorsuche gesichteten Suchergebnisse erlaubten Rückschlüsse auf notwendige Auswahlkriterien für mögliche Suchtreffer. Nach Abschluss der Vorsuche wurden die hergeleiteten Suchterme, festgelegten Suchquellen und -beschränkungen und definierten Auswahlkriterien genutzt, um die formelle Suche durchzuführen. Die aus der formellen Suche gewonnen Suchergebnisse wurden in einem weiteren Schritt durch Anwendung der Suchbeschränkungen verfeinert. Im letzten Schritt erfolgte die Sichtung der Suchergebnisse. Anhand der definierten Auswahlkriterien wurden

in einem manuellen Auswahlverfahren die Suchtreffer bestimmt. Die einzelnen Schritte beider Phasen und deren Ergebnisse werden in den folgenden Abschnitten detailliert beschrieben.

3.1.1. Herleiten von Suchtermen

Für die Herleitung der Suchterme wurde die Zielsetzung dieser Arbeit und die Forschungsfrage 3.1 als Grundlage genutzt. Es ergeben sich zwei Aspekte unter denen die Suchterme gebildet wurden. Der erste Aspekt ist die Suche nach Ansätzen zur Vergabe von Labels für natürlichsprachliche Anforderungen. Der zweite Aspekt ist die Suche nach Ansätzen zur Extraktion von Software-Features aus natürlichsprachlichen Anforderungen, da die zu vergebenen Labels im Kontext dieser Arbeit aussagekräftige Software-Features darstellen. Insgesamt wurden die vier Suchterme 3.2, 3.3, 3.4 und 3.5 gebildet, um die genannten Aspekte abzudecken. Die Aufteilung in vier separate Suchterme wurde aufgrund technischer Einschränkungen mancher Suchquellen (z.B. die Begrenzung der Suchterme auf 15 Begriffe in der Suchfunktion des IEEE) vorgenommen. Außerdem konnte so die Menge der Suchergebnisse pro Suchterm eingeschränkt werden, was eine manuelle Sichtung der Ergebnisse vereinfachte.

Während der informellen Vorsuche wurde festgestellt, dass in der Literatur der Begriff *tag recommendation* für die (semi-)automatische Vergabe von Labels für natürlichsprachliche Dokumente gebräuchlich ist. Im Kontext dieser Arbeit ist die Vergabe von Labels für natürlichsprachliche Anforderungen relevant. Anforderungen werden häufig in einem Software-Repository verwaltet. Um dies in den Suchtermen abzubilden wurden die Begriffe *requirement* und *software repository* verwendet. Innerhalb der informellen Vorsuche wurden Suchanfragen mithilfe der genannten Begriffe durchgeführt. Durch Sichtung der Suchergebnisse wurden die folgenden Erkenntnisse gewonnen: Die Verwendung des Begriffs *requirement* führte häufig zu mehreren tausend Suchergebnissen, sodass eine Sichtung der Suchergebnisse nicht praktikabel erschien. Zudem wurde festgestellt, dass viele Suchergebnisse aus nicht relevanten Forschungsgebieten wie medizinischer Bildverarbeitung oder verschiedenen Ingenieursdisziplinen stammten. Zur Einschränkung der Suchergebnisse wurde daher in den Suchtermen nachfolgender Iterationen der Begriff *software requirement* an Stelle von *requirement* verwendet. Zudem wurden die Begriffe *software requirement* und *software repository* in Anführungszeichen gesetzt, um die Suchergebnisse weiter zu präzisieren. Die Verwendung von Anführungszeichen um einen Suchbegriff bedeutet in den genutzten Suchquellen, dass exakt dieser Begriff innerhalb der Suchergebnisse vorkommen muss. Dies schließt z.B. Suchergebnisse aus, in denen nur *requirement* oder *repository* vorkommt. Um eine maximale Abdeckung der Suchterme zu erreichen, wurden in den informellen Suchanfragen auch Synonyme der einzelnen Begriffe

berücksichtigt. So wurde der Begriff *label* als aussagekräftiges Synonym für *tag* bzgl. der Qualität der Suchergebnisse identifiziert. Dies gilt ebenso für den Begriff *assignment* als Synonym zu *recommendation*. Dagegen führte die Berücksichtigung verwandter Begriffe wie *issue tracking system* oder *bug tracker* anstelle von *software repository* zu einer erheblichen Einschränkung der Suchergebnisse. Diese Beobachtung gilt ebenso für den anstelle von *software requirement* berücksichtigten Suchbegriff *user story*.

Im Kontext dieser Erkenntnisse wurde der erste Suchterm 3.2 für die Verwendung in der formellen Suche definiert, um den Aspekt der Vergabe von Labels für natürlichsprachliche Anforderungen in Software-Repositories abzudecken. Mithilfe dieses Suchterms lässt sich z.B. der Begriff *tag recommendation* sowie die möglichen Synonyme *tag assignment* oder *label assignment* bilden. Der Wildcard-Operator * wurde genutzt, um Formulierungen mit Gerundien wie *recommending tags* oder *assigning labels* zu berücksichtigen. Die Bedeutung des Wildcard-Operators * ist, dass dieser in Suchergebnissen durch eine Reihe beliebiger Zeichen ersetzen werden kann.

$$\begin{aligned}&\text{(tag OR label) AND (recommend* OR assign*)}\\&\text{AND (”software requirement”OR ”software repository”)}\end{aligned} \tag{3.2}$$

Während der Sichtung der Suchergebnisse der informellen Vorsuche wurde weiterhin festgestellt, dass in einzelnen Arbeiten auch die Begriffe *tagging*, *labeling* und *annotation* für die Vergabe von Labels für natürlichsprachliche Dokumente gebraucht werden. Zudem nutzten Arbeiten oft die Technik des (Dokumenten-) Clustering, um natürlichsprachliche Dokumente in Cluster semantisch ähnlicher Dokumente einzuteilen. Für eine aussagekräftige Benennung dieser Cluster mithilfe von Labels wurden verschiedene (semi-)automatische Ansätze beschrieben. Daher wurden weitere informelle Suchanfragen mithilfe der Begriffe *tagging*, *labeling*, *annotation* und *clustering* in Verbindung mit *software requirement* und *software repository* durchgeführt. Es zeigte sich jedoch, dass die Begriffe *tagging* und *labeling* nur wenig Einfluss auf die Menge und Qualität der jeweiligen Suchergebnisse hatte. Basierend auf diesen Beobachtungen wurde der zweite Suchterm 3.3 für die formelle Suche definiert.

$$\begin{aligned}&\text{(clustering OR annotation)}\\&\text{AND (”software requirement”OR ”software repository”)}\end{aligned} \tag{3.3}$$

Durch Lesen der während der informellen Vorsuche gefundenen Arbeiten, konnten (neben Clustering) eine Reihe weiterer Techniken für die Vergabe von Labels für natürlichsprachliche Dokumente identifiziert werden. Die beschriebenen Techniken sind *topic modeling*, *concept mining* und *keyphrase extraction*. Auffällig dabei war,

dass die Gerundien (*modeling*, *mining* und *extraction*) synonym verwendet wurden z.B. in *concept extraction* oder *keyphrase mining*. Außerdem wurde festgestellt, dass die Begriffe *keyphrase mining* und *keyword mining* häufig dieselbe Technik beschreiben. Um Forschungsarbeiten zu finden, die die o.g. Techniken konkret auf natürlichsprachliche Anforderungen anwenden, wurde für die formelle Suche der dritte Suchterm 3.4 definiert. Die Wildcard-Operatoren berücksichtigen sowohl die Begriffe *keyphrase* und *keyword*, als auch *extraction* und die Gerundiumsform *extracting*.

$$\text{(key* OR concept OR topic) AND (modeling OR mining OR extract*) AND "software requirement"} \tag{3.4}$$

Die bisher genannten Suchterme decken den Aspekt der Suche nach Ansätzen (Terme 3.2, 3.3) bzw. konkreten Techniken (Term 3.4) zur Vergabe von Labels für natürlichsprachliche Anforderungen in Software-Repositories ab. Noch zu berücksichtigen ist der Aspekt, dass die zu vergebenen Labels für Anforderungen Software-Features beschreiben sollen. Daher sind auch Ansätzen zur Extraktion von Software-Features aus natürlichsprachlichen Anforderungen von Interesse. Die extrahierten Software-Features können als Labels für eine Menge von Anforderungen genutzt werden. Zu diesem Zweck wurden während der informellen Vorsuche ebenfalls Suchanfragen mit den Begriffen *feature extraction* und *feature mining* in Verbindung mit *software requirement* durchgeführt. Während der Sichtung der Suchergebnisse stellte sich heraus, dass die Begriffe *feature extraction* und *feature mining* in der Literatur gebräuchlich sind. Ebenso konnten keine weiteren bzgl. der Suche nützlichen Synonyme für diese Terme identifiziert werden. Für die formelle Suche wurde daher der vierte Suchterm 3.5 definiert.

$$\text{(feature AND (extraction OR mining)) AND "software requirement"} \tag{3.5}$$

3.1.2. Festlegen von Suchquellen und –beschränkungen

Zu Beginn der Vorsuche wurden auf Empfehlung des betreuenden Lehrstuhls die wissenschaftlichen Datenbanken des IEEE und der Association for Computing Machinery (ACM) als Suchquellen genutzt. Weiterhin wurden die Datenbanken der Verlage Elsevier (ScienceDirect), Wiley und Springer berücksichtigt. Die nach Abschluss der Vorsuche festgelegten Suchbeschränkungen für die formelle Suche lassen sich in zwei Kategorien einordnen: Formale Suchbeschränkungen und verfeinernde Suchbeschränkungen. Formale Suchbeschränkungen wurden prinzipiell bei jeder Suchanfrage der formellen Suche angewandt. Dagegen wurden verfeinernde

Suchbeschränkungen nur im Rahmen der formellen Suche genutzt, falls die Menge der Suchergebnissen für eine Suchanfrage zu groß für die manuelle Auswahl relevanter Literatur war. In die Kategorien der formalen Suchbeschränkungen fallen die folgenden Beschränkungen: Die Menge der Suchergebnisse wurde auf Arbeiten aus den letzten 10 Jahren beschränkt, da im Fachgebiet Informatik Forschungsergebnisse aus älteren Arbeiten oft schnell überholt sind. Zudem beruhen aktuelle Arbeiten oft auf den Ergebnissen älterer Arbeiten, sofern diese noch für den jeweiligen Themenbereich relevant sind. Weiterhin wurden ausschließlich Arbeiten in englischer Sprache berücksichtigt, da der überwiegende Teil relevanter Literatur im Fachbereich Informatik in englischer Sprache verfasst wird.

In die Kategorie der verfeinernden Suchbeschränkungen fallen die folgenden Beschränkungen: Die Suche mit den Suchtermen wurde nur im Titel, Abstract und in den Keywords einer Arbeit durchgeführt. Im Gegensatz zu Treffern im gesamten Text einer Arbeit haben Treffer im Titel, Abstract und den Keywords die größere Aussagekraft bzgl. des Inhalts der Arbeit. Eine weitere verfeinernde Beschränkung ist diese: Es wurde nur Literatur aus dem Fachbereich Informatik berücksichtigt, um Suchergebnisse aus nicht relevanten Forschungsgebieten auszuschließen. Die letzte der verfeinernden Suchbeschränkungen ist die folgende: Die Menge der Suchergebnisse wurde auf wissenschaftliche Artikel beschränkt, um z.B. Buchkapitel nicht in den Suchergebnissen anzuzeigen.

3.1.3. Definition von Auswahlkriterien

In diesem Abschnitt werden die Auswahlkriterien für Suchtreffer beschrieben. Die Kriterien wurden während der informellen Vorsuche iterativ durch Sichtung der Suchergebnisse definiert. Es ist jedoch anzumerken, dass sie erst im Rahmen der formellen Suche bei der Auswahl von Suchtreffern angewendet wurden. Die folgenden Kriterien sind als Ausschlusskriterien zu verstehen, d.h. ein Suchergebnis wurde während der formellen Suche verworfen, sobald eines der genannten Kriterien zutraf.

1. Die Arbeit beschreibt keine Methode zur Vergabe von aussagekräftigen Labels oder die Methode bzw. Teilschritte lassen sich nicht für die Vergabe von Labels adaptieren.

2. Die Arbeit beschreibt keine Methode zur Extraktion von Software-Features oder die Methode bzw. Teilschritte lassen sich nicht für die Extraktion von Software-Features adaptieren.

3. Die Methode arbeitet nicht auf natürlichsprachlichen Anforderungen oder natürlichsprachlichen Dokumenten, die Software-Features beschreiben.

4. Die Methode beschreibt keine (semi-)automatische Methode.

5. Die Methode wird nicht in hinreichender Tiefe, d.h. in mindestens 4 Seiten reinem Text, beschrieben.

6. Die Arbeit beschreibt eine Methode, die von den Autoren bereits in einem nachfolgenden Artikel aktualisiert wurde.

Zu Beginn von Abschnitt 3.1.1 wurden bereits die beiden Aspekte beschrieben, die der Herleitung der Suchterme zugrunde liegen: Der erste Aspekt ist die direkte Vergabe von Labels für natürlichsprachliche Anforderungen. Der zweite Aspekt ist die Extraktion von Software-Features aus natürlichsprachlichen Anforderungen und die anschließende Verwendung der extrahierten Features als Labels für die Anforderungen. Offensichtlich sind Suchergebnisse nicht relevant, wenn sie keine Methode zum Erreichen dieser Ziele beschreiben. Während der informellen Vorsuche wurden jedoch auch Arbeiten identifiziert, deren Methodenbeschreibung nicht direkt der Zielsetzung einer der beiden Aspekte entsprach. Einer solche Arbeit kann relevant sein, wenn sich die beschriebene Methode bzw. einzelne Teilschritte bzgl. der Zielsetzung adaptieren lassen. Zum Beispiel muss eine Arbeit, deren Ziel die automatische Erstellung eines Domänenmodells aus natürlichsprachlichen Anforderungen ist, zunächst eine Methode für die Extraktion der Domänenkonzepte aus den Anforderungen beschreiben. Auch eine Arbeit, deren Zielsetzung das Vorschlagen relevanter Software-Features zur Wiederverwendung im SPLE ist, muss zunächst eine Methode zur Extraktion der Software-Features aus den Anforderungen haben. Aus diesen Beobachtungen leiten sich die Auswahlkriterien 1 und 2 ab. Diese wurden bewusst weit gefasst, um die o.g. Fälle einzuschließen. Während der informellen Vorsuche wurde festgestellt, dass eine engere Definition der Kriterien bezogen auf Arbeiten, deren Zielsetzung exakt dem Kontext dieser Arbeit entspricht, für viele Suchanfragen keine Suchtreffer geliefert hätte.

Ergänzend zu den Auswahlkriterien 1 und 2, schließt Kriterium 3 solche Arbeiten aus, deren Methode keine natürlichsprachlichen Anforderungen nutzt. Diese Methoden müssen sich nicht unbedingt auch für die Verwendung mit natürlichsprachlichen Anforderungen eignen. So ist z.B. eine Arbeit auszuschließen, die sich mit der Extraktion von Software-Features aus Quellcode-Dateien beschäftigt, da für Quellcode-Dateien diesbezüglich andere Voraussetzungen als für natürlichsprachliche Anforderungen gelten. Anzumerken ist, dass Auswahlkriterium 3, neben natürlichsprachlichen Anforderungen, auch Dokumente, die Software-Features beschreiben wie z.B. Benutzerhandbücher oder Software-Produktbeschreibungen einschließt. Vergleichbar mit den Auswahlkriterien 1 und 2 hätte eine engere Definition von Kriterium 3, d.h. die Beschränkung auf natürlichsprachliche Anforderungen, für viele Suchanfragen der informellen Vorsuche keine Suchtreffer geliefert.

Im Kontext dieser Arbeit soll ein (semi-)automatischer Ansatz zur Vergabe von Feature-Labels für User Stories entwickelt werden. Daher sind Arbeiten zu verwerfen, die einen rein manuellen Ansatz zur Vergabe von Labels beschreiben z.B. ein Vorgehen für ein Team von Entwicklern oder ein Software-Werkzeug zur manuellen Annotation von Anforderungen. Dies spiegelt sich in Auswahlkriterium 4 wider. Für die Entwicklung eines eigenen Ansatzes im Kontext dieser Arbeit ist es nötig, Ansätze aus verwandten Arbeiten hinsichtlich ihrer Eignung für den eigenen Ansatz analysieren zu können. Dafür muss der beschriebene Ansatz in hinreichender Tiefe beschrieben sein. Eine Arbeit wurde als hinreichend tief beschrieben beurteilt, wenn diese mindestens vier Seiten reinen Text enthielt. Während der informellen Vorsuche befanden sich unter den Suchergebnissen häufig Arbeiten, deren Methode nur grob beschrieben wurde und nicht im Detail nachvollziehbar war. Um diese Arbeiten zu verwerfen, wurde Ausschlusskriterium 5 definiert. Unter den Suchergebnisse der informellen Vorsuche befanden sich oft aufeinander aufbauende Arbeiten, in denen dieselben Autoren ihre Methode schrittweise aktualisierten. Um in diesen Fällen nur die jeweils aktuellste Arbeit zu berücksichtigen, wurde Ausschlusskriterium 6 definiert.

3.1.4. Durchführen der formalen Suche

Nach Abschluss der informellen Suche standen die einzelnen Suchterme, Suchquellen und -beschränkungen und Auswahlkriterien für Suchtreffer fest. Darauf basierend wurde die formale Suche durchgeführt. Dazu wurde für jede Kombination aus Suchterm und Suchquelle eine Suchanfrage gestellt. Dabei wurden die Besonderheiten der Suchquellen berücksichtigt. So mussten die Suchterme für die Suchquelle ACM in ein spezielles Format umgeformt werden. Außerdem unterstützte die Suchquelle ScienceDirect keine Wildcard-Operatoren bei der Suche in den Feldern Titel, Abstract und Keywords. Im Zuge der letzten durchgeführten Suchanfragen wurde eine Häufung von Duplikaten bei Suchergebnissen und Suchtreffern festgestellt. Aus diesem Grund wurden die in der informellen Vorsuche hergeleiteten Suchterme, festgelegten Suchquellen und -beschränkungen und definierten Auswahlkriterien als geeignet betrachtet, um eine maximale Abdeckung an Suchergebnissen und relevanten Arbeiten zu erzielen.

3.1.5. Verfeinerung der Suchergebnisse

Eine Verfeinerung der Suchergebnisse im Rahmen der formellen Suche wurde vorgenommen, wenn die Menge der Suchergebnisse einer Suchanfrage sehr gering bzw. umfangreich war. Ziel der Verfeinerung der Suchergebnisse war die Menge der Ergebnisse soweit einzugrenzen, dass die manuelle Auswahl relevanter Artikel durchführbar ist, gleichzeitig aber keine relevanten Artikel auszuschließen. Generell

wurde eine Menge von wenigen hundert Suchergebnissen pro Suchanfrage angestrebt. Eine sehr geringe Menge an Suchergebnissen pro Suchanfrage wurde als eine niedrige zweistellige Zahl betrachtet. In diesem Fall wurden die Anführungszeichen um einzelne Suchbegriffe in den Suchtermen weggelassen. Die Anführungszeichen bedeuten bei den berücksichtigten Suchquellen, dass der Begriff innerhalb der Anführungszeichen exakt in einem Suchergebnis vorkommen muss. Das Weglassen der Anführungszeichen führt somit zu einer Erhöhung der Suchergebnisse. Eine sehr umfangreiche Menge an Suchergebnissen wurde als eine Zahl im vierstelligen Bereich betrachtet. In diesem Fall war die manuelle Auswahl relevanter Literatur nicht praktikabel, sodass die verfeinernden Suchbeschränkungen aus dem Abschnitt 3.1.2 angewandt wurden. Eine Kombination beider Maßnahmen, d.h. Erweitern der Suchergebnisse durch Weglassen der Anführungszeichen und anschließende Beschränkung der Suchergebnisse durch Anwenden verfeinernder Suchanfragen, wurde ebenfalls genutzt.

3.1.6. Auswahl relevanter Arbeiten

Die (verfeinerten) Suchergebnisse für eine Suchanfrage wurden als Kandidaten für eine relevante Arbeit betrachtet. Um aus der Menge der Kandidaten die tatsächlich relevanten Arbeiten zu identifizieren, wurde das folgende Auswahlverfahren für jeden Kandidaten durchgeführt: Zunächst wurde der Titel der jeweiligen Arbeit gelesen. Häufig traf bereits mindestens eines der in 3.1.3 genannten Ausschlusskriterien zu und die entsprechende Arbeit konnte verworfen werden. War dies nicht der Fall, wurde der Abstract der Arbeit gelesen und bzgl. der Ausschlusskriterien geprüft. Bei Zutreffen eines der Kriterien wurde die entsprechende Arbeit jetzt verworfen. War dies erneut nicht der Fall, wurde die Arbeit komplett gelesen und eine finale Beurteilung anhand der Ausschlusskriterien vorgenommen. Wie in den vorherigen Schritten wurde die Arbeit bei Zutreffen eines der Ausschlusskriterien jetzt verworfen. War dies nicht der Fall wurde der Artikel als relevant betrachtet und als Quelle für diese Arbeit verwendet.

3.2. Vorstellung der Suchtreffer

In diesem Abschnitt werden die durch die Literaturrecherche identifizierten Suchtreffer zusammengefasst. Im Folgenden werden die eigentlichen Suchtreffer auch als verwandte Arbeiten bezeichnet. Die Zusammenfassung dient dem besseren Verständnis des Vergleichs der Arbeiten in den Abschnitten 3.3 und 3.4.

Arora et al. [2] beschreiben eine Methode zur automatisierten Konstruktion eines Glossars aus natürlichsprachlichen Anforderungen. Die Autoren nutzen Heuristi-

ken basierend auf den POS-Tags von Termen, um relevante Glossarbegriffe zu extrahieren. Dabei werden semantisch ähnliche Terme durch ein Ähnlichkeitsmaß identifiziert. Dieses berücksichtigt sowohl die syntaktische Ähnlichkeit von Termen mithilfe der Levenshtein-Distanz als auch ihre semantische Ähnlichkeit basierend auf den Relationen der Terme in WordNet [22]. WordNet ist eine externe Wissensquelle und verwaltet Konzepte durch die Gruppierung von Wörtern und ihren Synonymen in sogenannten *Synsets*. Zwischen *Snysets* werden außerdem verschiedene semantische Relationen wie Is-A- und Is-Part-Of-Assoziationen gespeichert.

In [10] stellen Hindle et al. eine Methode zur Analyse der Evolution einer Software vor. Die Autoren nutzen Topic Modeling bzw. Latent-Dirichlet-Allocation (LDA), um latente Themen in den Commit-Nachrichten von Entwicklern zu identifizieren. Den latenten Themen werden Labels in Form von Qualitätsanforderungen zugewiesen, was die Identifikation von Schwerpunkten während der Software-Wartung erlaubt. Die Autoren evaluieren zwei Ansätze für die Vergabe der Labels: 1) Eine manuell erstellte, externe Taxonomie definiert Labels und zugehörige, beschreibende Terme. Die Labels werden durch Matching der Terme aus der Taxonomie und jenen der latenten Themen zugewiesen. 2) Es wird ein ML-Modell auf den manuell mit Labels versehen latenten Themen trainiert.

Johann et al. [13] entwickeln eine Methode zum Matching von App-Features in Produktbeschreibungen und Nutzerkommentaren, um Entwicklern Hinweise auf häufig genutzte Features zu geben. Die Autoren nutzen Heuristiken basierend auf den POS-Tags von Termen zur Extraktion der Features. Das Matching erfolgt mithilfe der Kosinus-Ähnlichkeit.

Quirchmayr et al. [27] stellen eine Methode zur Extraktion feature-relevanter Informationen aus Benutzerhandbüchern vor. Die Autoren nutzen dazu Heuristiken basierend auf den POS-Tags von Termen in Syntaxbäumen. Es werden eine Reihe syntaktischer Muster vorgeschlagen, um durch den Parser entstandene Fehler in den Syntaxbäumen auszubessern. Die extrahierten feature-relevanten Informationen werden in Cluster aufgeteilt. Durch einen Domänenexperten kann den Clustern dann Labels in Form von Features zugewiesen werden.

In einem Übersichtsartikel stellen Bakar et al. [3] verschiedene Methoden aus der Literatur zur Extraktion von Features aus natürlichsprachlichen Anforderungen vor. Der Kontext ist die Identifikation von Features zur Wiederverwendung im Rahmen des SPLE. Die vorgestellten Methoden verwenden u.A. Heuristiken basierend auf den POS-Tags von Termen, Ähnlichkeitsmaße wie die Kosinus-Ähnlichkeit und semantische Kennzahlen wie tf-idf-Gewichte, Topic Modeling bzw. Latent-Semantic-Analysis (LSA) und externe Wissensquellen wie WordNet.

Liu et al. [16] stellen eine Methode zur Extraktion von App-Features und Domänenwissen aus Produktbeschreibungen vor. Features und Feature-Relationen werden mithilfe von Heuristiken basierend auf den POS-Tags von Termen in Syntaxbäumen

extrahiert. Die Autoren nutzen Topic Modeling (eine modifizierte Version von LDA), um latente Themen innerhalb der extrahierten Features zu identifizieren. Aus latenten Themen, extrahierten Features und deren Relationen wird ein formales Modell der Domäne generiert.

In [31] entwickeln Wang et al. eine Methode zur Vergabe von Labels für Produktbeschreibungen von OSS. Beschreibungen aus einem Trainingsdatensatz werden zunächst manuell mit Labels versehen. Jedem Term einer Beschreibung kann so ein Label in einer Term-Label-Matrix zugewiesen werden. Durch Topic Modeling (Labeled-LDA) kann anhand der Term-Label-Matrix einer unbekannten Beschreibungen ein Label vorgeschlagen werden.

Eine Reihe von Arbeiten [11, 30, 32] betrachten die Vergabe von Labels für Nutzerfragen auf Software-Informationsseiten. Hong et al. [11] erstellen zunächst einen Trainingsdatensatz mit manuell gelabelten Fragen. Für eine unbekannte Frage werden durch Topic Modeling (LDA) die semantisch ähnlichsten bereits mit einem Label versehenen Fragen bestimmt. Aus diesen wird mithilfe eines Ranking-Verfahrens das zutreffendste Label für die unbekannte Frage identifiziert.

Die Methode von Wang et al. [30] macht die folgenden Annahmen: 1) Ein Nutzerfrage ist eine Wahrscheinlichkeitsverteilung über eine Menge von Labels. 2) Ein Label ist eine Wahrscheinlichkeitsverteilung über Terme. Mithilfe von Topic Modeling (Labeled-LDA) und statistischer Inferenz wird für beide Annahmen jeweils ein Modell anhand eines Trainingsdatensatzes trainiert. Beide Modelle schlagen einer unbekannten Nutzerfrage die jeweils wahrscheinlichsten Labels vor. Die Vorschläge beider Modelle werden gewichtet und basierend auf einem Relevanz-Maß wird ein Ranking gebildet. Das Label mit dem höchsten Relevanz-Maß wird dann vorgeschlagen.

Wang et al. [32] stellen ein Modell aus mehreren Komponenten vor: 1) Ein anhand eines Trainingsdatensatzes trainiertes Multi-Label-Klassifikationsmodell, das einer unbekannten Nutzerfrage eine Menge von Labels zuordnen kann. 2) Ein Ähnlichkeitsmaß, das einer Frage die Labels einer ähnlichen, bereits gelabelten Frage zuweist. 3) Ein Ranking-Verfahren, das Relationen von Labels zu Termen einer Frage berücksichtigt. Die Vorschläge aus allen drei Komponenten werden gewichtet und der unbekannten Frage wird das Label mit dem höchsten resultierenden Relevanz-Maß vorgeschlagen.

In [21] beschreiben Ménard et al. eine Methode zur Extraktion von Domänenkonzepten aus Geschäftsprozessdokumenten. Die Konzepte werden mithilfe von Heuristiken basierend auf den POS-Tags von Termen extrahiert. Die Autoren nutzen verschiedene Filter- und Validierungsschritte, wie das Nachschlagen extrahierter Terme in Wörterbüchern und die Berücksichtigung domänenspezifischer Stoppwörter, um die Qualität der extrahierten Konzepte zu erhöhen.

Lucassen et al. [17] stellen eine Methode zur automatisierten Erstellung eines Domänenmodells aus Anforderungen vor. Die Autoren nutzen Heuristiken basierend auf den POS-Tags von Termen, um die Domänenkonzepte aus User Stories zu extrahieren.

Niu et al. [25] evaluieren verschiedene semantische Kennzahlen, um Cluster von natürlichsprachlichen Anforderungen automatisch mit Labels zu versehen. Für die Labels werden dabei Terme aus den Anforderungen genutzt. Der Term mit dem höchsten Wert bzgl. einer semantischen Kennzahl wird jeweils als Label definiert. Die Autoren berücksichtigen u.A. Kennzahlen basierend auf cluster-internen Termen (*tf*), cluster-externen Termen (*idf*) und sämtlichen Termen der Cluster (*tf-idf*).

In [23] entwickeln Misra et al. ein mehrstufiges Modell, um natürlichsprachliche Anforderungen in Cluster aufzuteilen und diesen Labels zuzuweisen. Kandidatenterme für Labels werden mithilfe von Heuristiken basierend auf den POS-Tags von Termen extrahiert. Anhand der extrahierten Terme werden durch Topic Modeling bzw. Latent-Semantic-Indexing (LSI) latente Themen innerhalb der Anforderungen identifiziert. Der Grad semantischer Abhängigkeit von Termen zu anderen Termen und von Termen zu Themen wird durch einen gewichteten Graphen modelliert. Dabei stellen die Knoten des Graphen Terme bzw. Themen dar. Die gewichteten Kanten beschreiben den Grad semantischer Abhängigkeit. Mithilfe des graphentheoretischen Konzepts der Zentralität eines Knotens werden semantisch bedeutende Terme identifiziert und als Labels verwendet. Anforderungen werden nun basierend auf der Ähnlichkeit ihrer extrahierten Terme mit den identifizierten Labels um die Labels herum in Cluster aufgeteilt.

3.3. Vergleich der Suchtreffer anhand formaler Kriterien

In diesem Abschnitt werden die verwandten Arbeiten anhand formaler Kriterien verglichen. Der Vergleich anhand formaler Kriterien erlaubt eine Einordnung der Arbeiten unabhängig von dem genutzten Ansatz bzw. der verwendeten Technik. Die Kriterien wurden während der Literaturrecherche anhand der wesentlichen Unterschiede der verwandten Arbeiten definiert. Der Vergleich der Arbeiten ist ebenfalls in Tabelle 3.1 dargestellt. Die verwandten Arbeiten stammen aus dem Forschungsgebiet Software-Entwicklung, betreffen jedoch unterschiedliche Forschungsbereiche. Diese Bereiche sind Requirements-Engineering [25, 23], Domänenanalyse [2, 16, 21, 17], Feature-Modeling [13, 27, 16], Software-Wartung [10] und SPLE [3]. Die anderen Arbeiten beschreiben Methoden zur Verschlagwortung von Software-Artefakten [31, 11, 30, 32].

Arbeit	[2]	[10]	[13]	[27]	[3]	[16]	[31]	[11]	[30]	[32]	[21]	[17]	[25]	[23]
Forschungs-bereich														
Requirements-Engineering	-	-	-	-	-	-	-	-	-	-	-	-	✓	✓
Domänen-analyse	✓	-	-	-	-	✓	-	-	-	-	✓	✓	-	-
Feature-Modeling	-	-	✓	✓	-	✓	-	-	-	-	-	-	-	-
Verschlag-wortung von Artefakten	-	-	-	-	-	-	✓	✓	✓	✓	-	-	-	-
Sonstige	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-
Methode														
Labels durch Termextraktion	✓	-	✓	✓	✓	✓	-	-	-	-	✓	✓	-	-
direkte Vergabe von Labels	-	✓	-	-	-	-	✓	✓	✓	✓	-	-	✓	✓
Eingabe-artefakt														
Anforderungen	✓	-	-	-	✓	-	-	-	-	-	-	✓	✓	✓
Nutzerfragen und -Kommentare	-	-	✓	-	✓	-	-	✓	✓	✓	-	-	-	-
Produkt-beschreibungen	-	-	✓	-	✓	✓	✓	-	-	-	-	-	-	-
Sonstige	-	✓	-	✓	✓	-	-	-	-	-	✓	-	-	-
Ergebnis														
Terme (Features)	-	-	✓	-	✓	✓	-	-	-	-	-	-	-	-
Terme (Domänen-konzepte)	-	-	-	-	-	✓	-	-	-	-	✓	✓	-	-
Terme (Sonstiges)	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-
Labels für An-forderungen	-	-	-	-	-	-	-	-	-	-	-	-	✓	✓
Labels für Nutzerfragen und -Kommentare	-	-	-	-	-	-	-	✓	✓	✓	-	-	-	-
Labels für Sonstiges	-	✓	-	-	-	-	✓	-	-	-	-	-	-	-

Tabelle 3.1.: Vergleich der verwandten Arbeiten anhand formaler Kriterien.

Die verwandten Arbeiten lassen sich außerdem nach der Art ihres Beitrag für diese Arbeit unterscheiden: Die Methoden von [2, 13, 27, 3, 16, 21, 17] beschreiben die Extraktion semantisch wichtiger Terme aus natürlichsprachlichen Software-Artefakten. Im Kontext dieser Arbeit können die extrahierten Terme als Labels für die jeweiligen Software-Artefakte verwendet werden. Dagegen werden in [10, 31, 11, 30, 32, 25, 23] Methoden für die direkte Vergabe von Labels für natürlichsprachliche Software-Artefakte beschrieben. Ein wesentlicher Unterschied der verwandten Arbeiten liegt in den Software-Artefakten, mit denen die vorgestellten Methoden arbeiten. Die Mehrzahl der Methoden nutzt natürlichsprachliche Anforderungen als Eingabe [2, 3, 17, 25, 23]. Häufig verwendet wurden ebenso Nutzerfragen und -kommentare von Software-Informationsseiten [13, 3, 11, 30, 32] und Software-Produktbeschreibungen [13, 3, 16, 31]. Die übrigen identifizierten Eingabeartefakte wurden jeweils nur von wenigen der vorgestellten Methoden berücksichtigt: [27, 3] nutzen Benutzerhandbücher, während [21] Dokumente über Geschäftsprozesse verwenden. Commit-Nachrichten aus VVS dienen als Eingabe für die Methode von [10]. Die verwandten Arbeiten unterscheiden sich außerdem hinsichtlich der Ergebnisse der jeweils beschriebenen Methoden. Die erste Kategorie umfasst als Ergebnis mit Labels versehene Software-Artefakte [10, 31, 11, 30, 32, 25, 23]. Die mit Labels versehenen Software-Artefakte der jeweiligen Methoden sind natürlichsprachliche Anforderungen [25, 23], Nutzerfragen von Software-Informationsseiten [11, 30, 32], Software-Produktbeschreibungen [31] und Commit-Nachrichten [10]. Die zweite Kategorie beinhaltet aus natürlichsprachlichen Software-Artefakten extrahierte Terme, welche dann als Labels verwendet werden können. Die extrahierten Terme haben je nach Methode eine unterschiedliche Bedeutung. Extrahierte Terme sind entweder Feature-Beschreibungen [13, 3, 16], Domänenkonzepte [16, 21, 17], feature-relevante Informationen [27] oder Glossarbegriffe [2].

3.4. Vergleich der Suchtreffer anhand der Forschungsfrage

Im folgenden Abschnitt werden die verwandten Arbeiten in Bezug zur Forschungsfrage 3.1 verglichen. Der Vergleich ist außerdem in Tabelle 3.2 dargestellt. In Bezug auf die Forschungsfrage wurden vier wesentliche Ansätze für die Vergabe von Labels für natürlichsprachliche Anforderungen identifiziert: Der linguistische Ansatz [2, 13, 27, 3, 16, 21, 17, 23] beruht auf den morphologischen, syntaktischen und semantischen Eigenschaften von Termen. Der linguistische Ansatz nutzt Konzepte und Techniken aus dem Bereich des NLP wie POS-Tagging, NER und syntaktisches Parsen. Semantische bedeutende Terme werden häufig anhand syntaktischer Muster basierend auf den POS-Tags von Termen extrahiert [2, 13, 21, 17]. Diese können als Label für Software-Artefakte verwendet werden. Syntaktischen Muster sind z.B. Nomen-Verb-Paare oder Nomen-Nomen-Paare, welche z.B.

durch die Kombination der Penn-Treebank-Labels *NN* und *VB* bzw. *NN* und *NN* dargestellt werden können. Extrahierte Terme können zudem durch verschiedene Techniken validiert werden z.B. durch Nachschlagen der Terme in Wörterbüchern oder dem Verwenden von (domänenspezifischen) Stoppwörtern [21].

Der statistische Ansatz [10, 3, 16, 31, 11, 30, 32, 25, 23] umfasst die Analyse von Termhäufigkeiten und die Verwendung statistischer Modelle. Basierend auf den Termhäufigkeiten werden Kennzahlen abgeleitet, die eine Aussage über die Wichtigkeit von Termen für die Semantik eines natürlichsprachlichen Software-Artefakts machen. Anhand der Kennzahlen können Terme, die sich als potenzielle Labels eignen, identifiziert werden [3, 32, 25, 23]. Oft eingesetzt werden die Kennzahlen *tf*, *idf* oder deren Kombination *tf-idf*. Eine weitere Technik ist die Verwendung von statistischen Modellen aus dem Bereich des Topic Modeling [10, 3, 16, 31, 11, 30, 23]. Diese erlauben das Identifizieren latenter Themen innerhalb von natürlichsprachlichen Software-Artefakten. Einzelnen Artefakten können so latente Themen zugewiesen werden. Die latenten Themen werden durch eine Menge von beschreibenden Termen definiert. Anhand der Termmenge kann ein Label definiert werden. Häufig genutzt werden die statistischen Modelle LDA und LSI bzw. deren Varianten wie Labeled-LDA.

Arbeit	[2]	[10]	[13]	[27]	[3]	[16]	[31]	[11]	[30]	[32]	[21]	[17]	[25]	[23]
Ansatz														
Linguistisch	✓	-	✓	✓	✓	✓	-	-	-	-	✓	✓	-	✓
Statistisch	-	✓	-	-	✓	✓	✓	✓	✓	✓	-	-	✓	✓
Machine Learning	-	✓	-	✓	✓	-	-	-	-	✓	-	-	✓	✓
Technik														
POS-Muster	✓	-	✓	✓	✓	✓	-	-	-	-	✓	✓	-	✓
Topic Modeling	-	✓	-	-	✓	✓	✓	✓	✓	-	-	-	-	✓
Ähnlichkeits-maße und semantische Kennzahlen	-	-	-	-	✓	-	-	-	-	✓	-	-	✓	✓
Klassifikation	-	✓	-	-	-	-	-	-	-	✓	-	-	-	-
Clustering	-	-	-	✓	✓	-	-	-	-	-	-	-	✓	✓

Tabelle 3.2.: Vergleich der verwandten Arbeiten anhand der Forschungsfrage.

Ein weiterer Ansatz beinhaltet die Verwendung von Modellen aus dem Bereich des ML [10, 27, 3, 32, 25, 23]. Dabei können zwei Verfahren unterschieden werden: Im überwachten Lernen werden Software-Artefakte zunächst manuell, etwa durch einen Domänenexperten, mit Labels versehen. Dieser Trainingsdatensatz wird genutzt, um ein Klassifikationsmodell zu trainieren [10, 32]. Das Modell erlaubt

es dann unbekannte Artefakte zu klassifizieren und ihnen das Label der jeweiligen Klasse zuzuweisen. Im nicht überwachten Lernen werden Software-Artefakte in Cluster ähnlicher Artefakte aufgeteilt [27, 3, 25, 23]. Anhand der identifizierten Cluster können aussagekräftige Labels definiert werden.

In hybriden Ansätzen werden Techniken aus den bereits genannten Ansätzen kombiniert, um die Aussagekraft der zu vergebenen Labels zu erhöhen [10, 27, 3, 16, 32, 25, 23]. Beispielsweise können zunächst semantisch bedeutende Terme mithilfe syntaktischer Muster aus den natürlichsprachlichen Software-Artefakten extrahiert werden. In einem nächsten Schritt erfolgt dann mithilfe von ML die Aufteilung der Terme in Cluster [27]. Neben den bereits genannten Techniken wurden in den hybriden Ansätzen auch Graphen [23] und externe Wissensquellen [10, 3] genutzt. So wurden extrahierte Terme und latente Themen als Knoten in einem Graphen dargestellt. Gewichtete Kanten beschrieben die semantische Abhängigkeit zwischen Termen und Themen. Das graphentheoretische Konzept der Zentralität eines Knotens wurde angewandt, um semantisch bedeutende Terme zu identifizieren [23]. Externen Wissensquellen wurden genutzt, um über eine Menge von extrahierten Termen zu abstrahieren oder die semantische Abhängigkeit zwischen Termen festzustellen. Im ersten Fall wurde eine Taxonomie definiert, die einer Menge von Termen ein Label in Form einer Qualitätsanforderung zuweist [10]. Im zweiten Fall wurden die Relationen von Termen in WordNet analysiert, um deren semantische Abhängigkeit zu bestimmen [3].

3.5. Besonderheiten der Ansätze und Techniken der Suchtreffer

Während der Analyse der verwandten Arbeiten konnten einige Besonderheiten der jeweils vorgestellten Ansätze und Techniken identifiziert werden, die bei der Entwicklung eines Ansatzes im Kontext dieser Arbeit berücksichtigt werden müssen. Diese Besonderheiten sind in der Tabelle 3.3 aufgeführt und werden im Folgenden beschrieben.

Viele der vorgestellten Methoden und Techniken benötigen einen möglichst umfangreichen Datensatz von natürlichsprachlichen Software-Artefakten, um optimale Ergebnisse zu erzielen [10, 27, 3, 16, 31, 11, 30, 32, 25, 23]. Darunter fallen diejenigen Methoden, die auf ML oder der statistischen Analyse der Software-Artefakte beruhen. So sind z.B. die Vorhersagen eines Klassifikationsmodells umso genauer, je besser der Trainingsdatensatz die Menge der Software-Artefakte repräsentiert. Gleichzeitig erlaubt ein umfangreicher Datensatz präzisere statistische Aussagen als ein kleiner Datensatz. Termhäufigkeiten und abgeleitete Maße sowie Kennzahlen können somit genauer die Ähnlichkeit bzw. semantische Wichtigkeit von

Termen und Artefakten repräsentieren. Statistische Modelle für Topic Modeling modellieren im Kontext dieser Arbeit häufig Wahrscheinlichkeitsverteilungen von Termen für latente Themen und Wahrscheinlichkeitsverteilungen von Themen für Anforderungen. Ein umfassender Datensatz kann auch diesbezüglich für präzisere Aussagen sorgen.

Arbeit	[2]	[10]	[13]	[27]	[3]	[16]	[31]	[11]	[30]	[32]	[21]	[17]	[25]	[23]
Besonderheiten														
umfangreiches (Trainings-) datenset ideal	-	✓	-	✓	✓	✓	✓	✓	✓	✓	-	-	✓	✓
anwendbar auf einzelnen Anforderungen	✓	-	✓	✓	✓	✓	-	-	-	-	✓	✓	-	✓
externe Wissensquelle notwendig	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-
domänenspezifische Ergebnisse	-	✓	-	-	-	-	✓	✓	✓	✓	-	-	-	-
Abstraktionsschritt	-	✓	-	-	-	-	✓	✓	✓	✓	-	-	-	-

Tabelle 3.3.: Vergleich der verwandten Arbeiten anhand ihrer Besonderheiten im Kontext dieser Arbeit.

Mit der idealerweise erfüllten Voraussetzung, dass ein möglichst umfangreicher Datensatz von natürlichsprachlichen Software-Artefakten vorhanden ist, entstehen Einschränkungen: In länger bestehenden und ausgereiften Softwareprojekten kann es z.B. durchaus eine umfangreiche Anforderungsspezifikation geben. Die genannten Methoden sind in diesen Fällen anwendbar. In kleineren oder neu begonnenen Projekten kann es jedoch sein, dass nur wenige textuelle Anforderungen existieren. Die im Kontext dieser Arbeit entwickelte Methode soll aber auch in diesen Fällen anwendbar sein. Methoden, die einen möglichst umfangreichen Datensatz benötigen, sind daher nicht ohne Vorbehalt anzuwenden. Diese Einschränkung haben Methoden mit einem linguistischen Ansatz basierend auf syntaktischen Mustern nicht. Die Extraktion semantisch wichtiger Terme mithilfe von syntaktischen Mustern ist auch auf einer einzelnen oder wenigen Anforderungen anwendbar [2, 13, 27, 3, 16, 21, 17, 23].

Zudem ist problematisch, dass Software-Artefakte oft sehr domänenspezifisch sind. Ein Modell, das auf einem Datensatz von domänenspezifischen Artefakten trainiert wurde, kann nur beschränkt in anderen Domänen als der Trainingsdomäne eingesetzt werden [10, 31, 11, 30, 32]. Dies erschwert die Entwicklung einer allgemein einsetzbaren Methode.

Eine weitere Besonderheit ergibt sich bzgl. der zu vergebenen Labels. Diese stellen im Kontext dieser Arbeit Software-Features dar und sollen für Anforderungen in Form von User Stories vergeben werden. Wie in dem Beispiel in Abschnitt 1.2 bereits deutlich wird, werden Software-Features aber häufig nicht explizit in den Anforderungen genannt. Um ein Feature-Label zu definieren, muss über die in der User Story beschriebene Funtkionalität abstrahiert werden. Viele der vorgestellten Methoden berücksichtigen diesen Abstraktionsschritt nicht bzw. können ihn aufgrund ihrer Funktionsweise nicht leisten. Dies betrifft Methoden aus dem linguistischen Ansatz, wenn sie allein auf dem Einsatz von syntaktischen Mustern basieren. In diesem Fall müssen extrahierte Terme direkt als Label verwendet werden [2, 13, 21, 17]. Betroffen sind auch Methoden aus dem statistischen Ansatz, wenn sie Labels allein anhand von Ähnlichkeitsmaßen und semantischen Kennzahlen von Termen definieren. In diesem Fällen muss der Term mit dem höchsten Wert bzgl. eines Maßes oder einer Kennzahl als Label vorgeschlagen werden [3, 25, 23]. Methoden aus dem Bereich des ML und Topic Modeling berücksichtigen den Abstraktionsschritt. Das Identifizieren potenzieller Labels erfolgt jedoch auch hier semi-automatisch: Zur Erstellung eines Trainingsdatensatzes für ein Klassifikationsmodell, müssen zunächst Anforderungen manuell durch einen Domänenexperten mit einem Label versehen werden [10, 31, 11, 30, 32]. Cluster von extrahierten Termen müssen ebenfalls manuell mit einem Label versehen werden, wenn dazu nicht auf die beschränkten linguistischen Methoden zurückgegriffen werden soll [27]. Latente Themen, die durch Topic Modeling identifiziert wurden, stellen in erster Linie eine Menge von Termen dar. Auch hier muss manuell über die Termmenge abstrahiert werden, um ein aussagekräftiges Label zu definieren [16].

3.6. Erkenntnisse aus der Literaturrecherche

Die Literaturrecherche hat gezeigt, dass es noch wenige Arbeiten gibt, die sich konkret mit der Vergabe von Feature-Labels für User Stories beschäftigen. Es konnten jedoch einige Arbeiten identifiziert werden, die sich im weiteren Sinne mit der Vergabe von Labels für natürlichsprachliche Anforderungen auseinandersetzen. In Bezug auf die Beantwortung der Forschungsfrage 3.1 hat die Literaturrecherche die folgenden Ergebnisse gebracht: Die Arbeiten aus der Literatur verfolgen im Allgemeinen vier Ansätze. Diese sind 1) der linguistische Ansatz, 2) der statistische Ansatz, 3) Ansätze, die auf ML basieren und 4) hybride Ansätze. Im Rahmen dieser Ansätze werden die folgenden Techniken eingesetzt: Im linguistischen Ansatz werden häufig POS-Muster eingesetzt, um semantisch wichtige Terme aus den Anforderungen zu extrahieren. Diese können dann als Labels für die Anforderungen verwendet werden. Innerhalb des statistischen Ansatzes werden Modelle für das Topic Modeling eingesetzt, um latente Themen innerhalb der Anforderungen zu identifizieren. Diese Themen können dann als Labels für die Anforderungen dienen. Außerdem werden statistische Ähnlichkeitssmaße und semantisch Kennzahlen

verwendet, um semantisch wichtige Terme in den Anforderungen zu identifizieren, welche dann als Labels dienen können. Die Ansätze, die auf maschinellem Lernen basieren, setzen unterschiedliche ML-Modelle zur Klassifikation oder zum Clustering der Anforderungen ein. In hybriden Ansätzen werden die Techniken aus den übrigen Ansätzen kombiniert.

Die Ansätze, die auf ML basieren zeigen im Rahmen der jeweiligen Arbeiten gute Ergebnisse. Diese Arbeiten verwenden jedoch umfangreiche Datensätze für das Training der eingesetzten ML-Modelle. Auch sind die jeweils erzielten Ergebnisse durch das Training mit spezifischen Datensätzen oft domänenspezifisch und lassen sich nicht generalisieren. Der Einsatz von ML-Modellen muss im Kontext dieser Arbeit daher genau evaluiert werden. Dies gilt ebenso für die statistischen Ansätze: Arbeiten, die Topic Modeling oder statistische Ähnlichkeitsmaße einsetzen, zeigen im Rahmen der jeweiligen Problemstellung gute Ergebnisse. Aber auch diese Arbeiten basieren häufig auf umfangreichen Datensätzen. Die linguistischen Ansätze, die POS-Muster zur Extraktion semantisch bedeutender Terme verwenden, zeigen im Rahmen der jeweiligen Arbeiten ebenfalls positive Ergebnisse. Diese haben zudem den Vorteil, dass sie auf einzelnen User Stories bzw. einer kleinen Menge von User Stories anwendbar sind. Die eingesetzten Muster sind jedoch häufig auf ein Artefakt wie z.B. Benutzerhandbücher, Geschäftsprozessdokumente oder Nutzerfragen und -kommentare optimiert. Die Anwendbarkeit dieser Muster für User Stories muss daher genau geprüft werden. Werden POS-Muster eingesetzt, müssen diese in jedem Fall auf die Verwendung mit User Stories hin optimiert werden. Die Arbeiten, die externe Wissensquellen verwenden, zeigen für ihre jeweilige Problemstellung ebenfalls positive Ergebnisse. Allerdings ist der Beitrag der externen Wissensquellen vernachlässigbar, da diese häufig nur einen kleinen Teil des in den jeweiligen Arbeiten entwickelten Ansatzes ausmachen. Aus diesem Grund und um eine möglichst breite Anwendbarkeit des in dieser Arbeit entwickelten Ansatzes sicherzustellen, werden externen Wissensquellen im Rahmen dieser Arbeit nicht berücksichtigt. Zudem zeigt die Literaturrecherche, dass die verwandten Arbeiten sich häufig auf einen Aspekt fokussieren: Entweder es werden semantisch wichtige Terme aus Artefakten extrahiert (linguistischer Ansatz) oder es werden Label mithilfe von ML-Modellen (ML-basierter Ansatz) vorgeschlagen. Eine Kombination verschiedener Ansätze zur Abdeckung beider Aspekte ist im Rahmen dieser Arbeit anzustreben, um die Vorteile einzelner Ansätze zu nutzen und deren Nachteile durch andere Ansätze auszugleichen.

4. (Semi-)automatische Vergabe von Feature-Labels

Im folgenden Kapitel wird das in dieser Arbeit entwickelte Modell zur (semi-) automatischen Vergabe von Feature-Labels vorgestellt. Abschnitt 4.1 gibt eine Übersicht des entwickelten Modells. Die darauf folgenden Abschnitte beschreiben die einzelnen Komponenten des Modells. In Abschnitt 4.2 wird die Erfassung und das Aufbereiten der Eingabedaten erläutert. Die Extraktion von feature-relevanten Informationen wird in Abschnitt 4.3 beschrieben. Abschnitt 4.4 beschreibt das Vorschlagen von Feature-Labels. In Abschnitt 4.5 wird die eigentliche Vergabe der Feature-Labels erläutert.

4.1. Übersicht des Modells

Wie bereits in den Abschnitten 1.2 und 3.5 dargestellt, ist die voll-automatische Identifikation von Software-Features innerhalb von User Stories schwierig. Software-Features sind oft nicht explizit im Text der User Story genannt, sondern nur implizit beschrieben. Es ist weiterhin oftmals ein Abstraktionsschritt nötig, der von einzelnen semantisch wichtigen Textbestandteilen auf die Ebene der Software-Features abstrahiert. Daher wurde im Rahmen der Problemstellung dieser Arbeit die Annahme gemacht, dass eine voll-automatische Vergabe von Feature-Labels für User Stories nicht praktikabel und ein (semi-)automatisches Modell zweckdienlicher ist. Unter Berücksichtigung der Problemstellung wurde das Modell jedoch mit der Absicht erstellt, dem Nutzer eine möglichst umfangreiche Unterstützung bei der Vergabe von Feature-Labels für User Stories zu geben. Dieses Ziel wird durch das Modell auf zwei verschiedene Weisen erreicht: 1) Zum Einen extrahiert es feature-relevante Informationen aus User Stories. 2) Zum Anderen schlägt es passende Feature-Labels für User Stories vor.

Das Modell extrahiert feature-relevante Informationen, sowohl aus einer User Story selbst als auch aus solchen User Stories, die semantisch ähnlich zu dieser User Story sind. Die Extraktion von feature-relevanten Informationen soll die Vergabe von Feature-Labels für eine User Story folgendermaßen unterstützen: Feature-relevante Information wird dazu genutzt, das durch das Modell vorge-

schlagene Feature-Label zu validieren. Die Validierung erfolgt dabei durch den Nutzer anhand eines manuellen Abgleichs des vorgeschlagenen Feature-Labels mit den feature-relevanten Informationen. Im Falle einer Validierung mit positivem Ergebnis, d.h. der Nutzer kommt zu der Einschätzung, dass die feature-relevanten Informationen das Software-Feature des Labels treffend beschreiben, kann das vorgeschlagene Label übernommen und vergeben werden. Im Falle einer Validierung mit negativem Ergebnis, d.h. der Nutzer meint, dass die feature-relevanten Informationen semantisch nicht mit dem vorgeschlagenen Label übereinstimmen, können die feature-relevanten Informationen dennoch bei der manuellen Definition eines aussagekräftigeren Feature-Labels unterstützen. Die Unterstützung wird durch die Synthese der tatsächlich feature-relevanten Textbestandteile der User Story und der Filterung nicht feature-relevanter Textbestandteile erreicht. Weiterhin steht dem Nutzer durch die Bündelung von feature-relevanten Informationen aus der User Story selbst und jenen aus semantisch ähnlichen User Stories nicht nur insgesamt mehr feature-relvante Information zur Verfügung, sondern diese ist auch breiter gestreut. Dies unterstützt insbesondere bei Projekten mit einer Vielzahl von User Stories die Definition eines passenden Feature-Labels. Das Vorschlagen eines Feature-Labels für eine User Story erfolgt mithilfe eines KNN, das anhand bekannter und bereits gelabelter User Stories trainiert wurde. Eine unbekannte User Story wird somit unter Berücksichtigung der bereits vergebenen Feature-Labels klassifiziert und es kann so ein mögliches Label für diese User Story vorgeschlagen werden. Das Modell arbeitet dabei iterativ. Für eine unbekannte User Story wird ein Feature-Label durch das KNN vorgeschlagen. Außerdem werden die feature-relevanten Informationen für die User Story durch das Modell extrahiert. Der Nutzer kann eine Validierung durchführen, d.h. das vorgeschlagene Label übernehmen oder anhand der gebündelten feature-relevanten Informationen ein treffenderes Label definieren. Die User Story wird zusammen mit dem letztlich vergebenen Feature-Label der Menge der bekannten und gelabelten User Stories hinzugefügt. Anhand dieser Menge wird das KNN erneut für die nächste Iteration des Modells trainiert.

Das entwickelte Modell ist im Detail in Abbildung 4.1 dargestellt. Es lässt sich in die folgenden Komponenten unterteilen: (1) Erfassen und Aufbereiten der Eingabedaten, (2) Extraktion feature-relevanter Informationen, (3) Vorschlagen von Feature-Labels und (4) (semi-)automatische Vergabe eines Feature-Labels. Die ersten drei Komponenten arbeiten dabei voll-automatisch. Die vierte Komponente erfordert dagegen die Interaktion eines Nutzers. Die Komponenten haben dabei die folgenden Verantwortlichkeiten: Die erste Komponente erfasst und bereitet die nötigen Eingabedateien für die Verarbeitung innerhalb des Modells auf. Die zweite Komponente extrahiert feature-relevante Information. Die dritte Komponente beinhaltet das KNN und schlägt passende Feature-Labels vor. Die vierte Komponente umfasst die durch den Nutzer ausgeführte Validierung und somit die (semi-)automatische Vergabe von Feature-Labels. In den folgenden Abschnitten werden die einzelnen Komponenten des Modells genauer beschrieben.

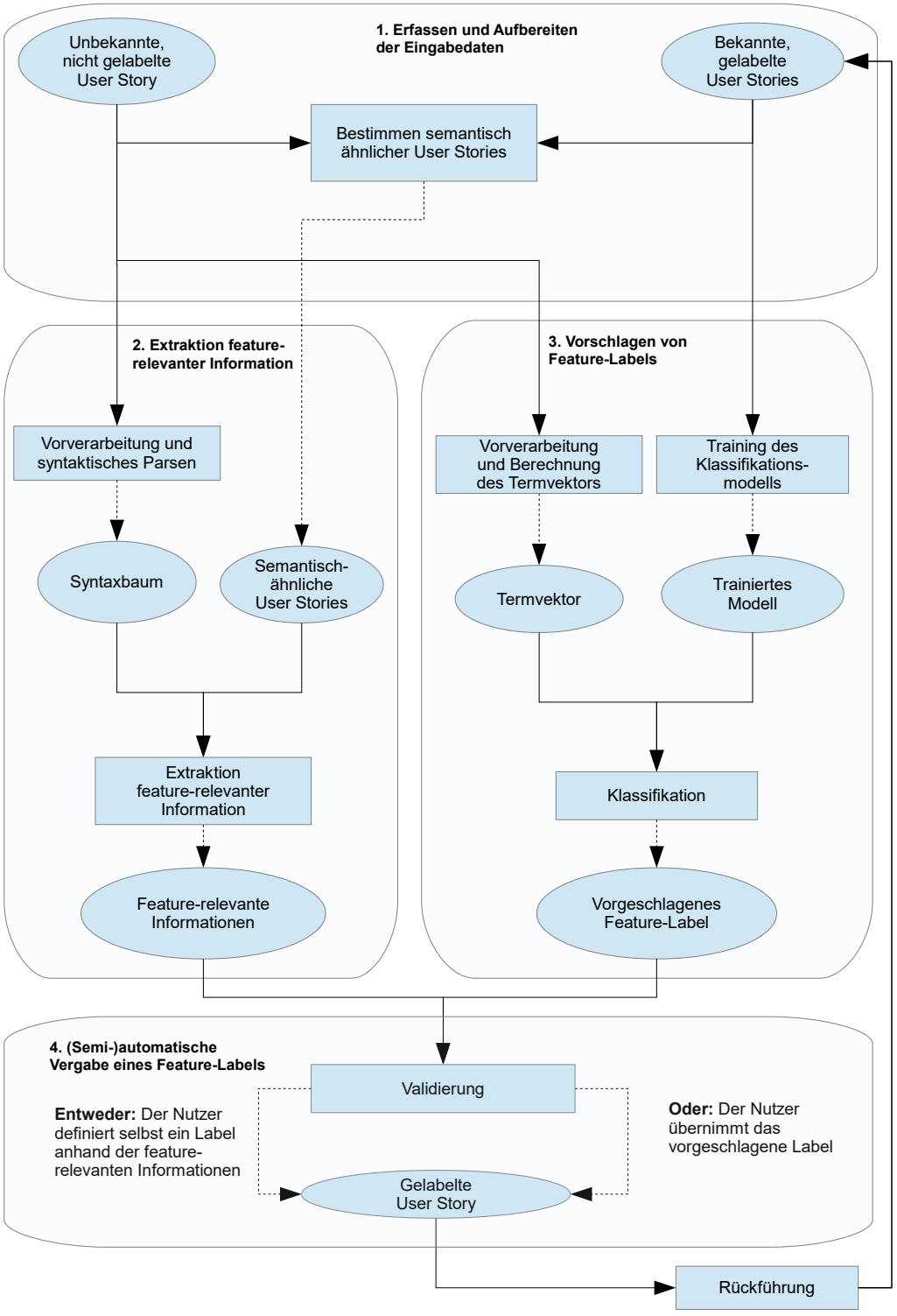

Abbildung 4.1.: Modell zur (semi-)automatischen Vergabe von Feature-Labels für User Stories.

4.2. Erfassen und Aufbereiten der Eingabedaten

In diesem Abschnitt wird die erste Modellkomponente beschrieben, welche sich mit dem Erfassen und Aufbereiten der Eingabedaten beschäftigt. Die primäre Eingabe in das Modell ist eine unbekannte User Story ohne ein zugewiesenes Feature-Label. Im Kontext dieser Arbeit wird angenommen, dass eine User Story aus einem Titel (Zusammenfassung) und dem eigentlichen Text (Beschreibung) der User Story besteht. Weiterhin ist die sekundäre Eingabe in das Modell die Menge der bereits bekannten und gelabelten User Stories. Eine bekannte User Story umfasst somit Titel, Beschreibung und ein Feature-Label. Die Menge der bekannten User Stories kann die leere Menge sein, wenn noch keine User Stories dem Modell hinzugefügt wurden. In diesem Fall arbeiten einige Komponenten des Modells nicht oder nur unvollständig. So kann z.B. kein Feature-Label für die unbekannte User Story vorgeschlagen werden, da es keine bekannten User Stories gibt anhand derer das KNN trainiert werden konnte.

Der wesentliche Schritt innerhalb dieser Modellkomponente ist das Identifizieren der Menge von semantisch ähnlichen User Stories aus den bereits bekannten User Stories für die unbekannte User Story. Die semantische Ähnlichkeit von User Stories wird im Kontext dieser Arbeit mithilfe der Kosinus-Ähnlichkeit gemessen, die in Abschnitt 2.3 vorgestellt wird. Die Kosinus-Ähnlichkeit wird anhand der gewichteten Termvektoren der User Stories berechnet. Die Gewichtung der Komponenten der Termvektoren erfolgt mithilfe des tf-idf-Gewichtungsschemas, welches ebenfalls in Abschnitt 2.3 beschrieben wird. Die Menge der semantisch ähnlichen User Stories bzgl. der unbekannten User Story wird dann folgendermaßen bestimmt: Zunächst wird für jede der bekannten User Stories und der unbekannten User Story die jeweilige semantische Ähnlichkeit berechnet. Gibt es noch keine bekannten User Stories, ist die Menge der semantisch ähnlichen User Stories die leere Menge. Gibt es bereits bekannte User Stories, ist die Top-N der ähnlichsten User Stories die Menge der semantisch ähnlichen User Stories. Der Parameter N wurde während der Entwicklung durch informelle Evaluation der extrahierten feature-relevanten Informationen aus den verwendeten Datensätzen auf $N = 3$ definiert. Während der Entwicklung wurde weiterhin festgestellt, dass pro User Story ungefähr 2 einzelne feature-relevante Informationen extrahiert werden. Dies ergibt innerhalb der zweiten Komponente des Modells ungefähr eine Menge von $(N + 1) \cdot 2 = 2N + 2$ feature-relevanten Informationen pro User Story. Dabei stammen 2 extrahierte feature-relevante Informationen aus der User Story selbst und $2N$ aus den semantisch ähnlichen User Stories. Um die Menge der feature-relevanten Informationen innerhalb der zweiten Komponente des Modells nicht zu umfangreich werden zu lassen, wurde der Parameter auf $N = 3$ bestimmt. Der Parameter ist grundsätzlich jedoch frei wählbar.

4.3. Extraktion feature-relevanter Informationen

Im folgenden Abschnitt wird die zweite Komponente des entwickelten Modells beschrieben, welche sich mit der Extraktion der feature-relevanten Informationen befasst. Unterabschnitt 4.3.1 beschreibt die syntaktischen Muster, auf denen die heuristischen Regeln zur Extraktion basieren. Weitere syntaktische Besonderheiten und sprachliche Phänomene, die einen Einfluss auf die Definition der Extraktionsregeln haben, werden in Unterabschnitt 4.3.2 vorgestellt. Unterabschnitt 4.3.3 erläutert die Extraktionsregeln selbst. Der gesamte Ablauf der Extraktion der feature-relevanten Informationen wird in Unterabschnitt 4.3.4 anhand eines Beispiels demonstriert.

Die Eingaben der zweiten Komponente sind (1) die unbekannte User Story, für die ein Feature-Label zu vergeben ist und (2) die Menge der semantisch ähnlichen User Stories, welche in der ersten Modellkomponente bestimmt wird. Aus beiden Eingaben extrahiert diese Komponente die jeweiligen feature-relevanten Informationen. Die Extraktion erfolgt durch Anwendung von heuristischen Regeln auf dem Syntaxbaum einer User Story. Syntaxbäume werden in Abschnitt 2.3 vorgestellt. Die heuristischen Regeln filtern und transformieren den Syntaxbaum zunächst, bevor sie die feature-relevanten Informationen extrahieren. Die eigentliche Extraktion erfolgt anhand bestimmter POS-Muster innerhalb des Syntaxbaumes, welche charakteristisch für das Auftreten von feature-relevanten Informationen in User Stories sind. POS-Muster in Syntaxbäumen werden in Abschnitt 2.3 vorgestellt. Die POS-Muster wurden durch Analyse der verwendeten Datensätze anhand häufig wiederkehrender syntaktischer Muster, die oft feature-relevante Information enthielten, identifiziert.

4.3.1. Syntaktische Muster für feature-relevante Informationen

Wie bereits in Abschnitt 2.1 erläutert folgen User Stories oft einer bestimmten Satzschablone. Diese teilt die User Story häufig in drei Bestandteile, welche jeweils den Nutzer, die Aktivität und das Ziel beschreiben. Während der Analyse der User Stories aus den verwendeten Datensätzen ist aufgefallen, dass der erste Bestandteil von User Stories für die Definition eines Feature-Labels bzw. Extraktion feature-relevanter Information meist nicht wichtig ist. Der Grund ist, dass aufgrund der Struktur von User Stories immer eine Aussage bzgl. eines Nutzer der Software gemacht wird. Für die Definition eines Features-Labels und insbesondere der Extraktion der feature-relevanten Informationen sind aber die auszuführenden Handlungen, Aktionen und Ziele des Nutzers von Bedeutung, nicht jedoch der eigentliche Nutzer selbst. Software-Features werden immer von den Nutzern bzw. Projektbeteiligten der Software verwendet, was für die Beschreibung des

eigentlichen Software-Features jedoch nicht immer relevant ist. Aus diesem Grund wurde bei der Herleitung der Extraktionsregeln im Wesentlichen der zweite und dritte Bestandteil von User Stories, also die beschriebene Aktivität und das Ziel berücksichtigt. Dies ist an den User Stories aus den verwendeten Datensätzen in Abbildung 4.2 erkennbar. Der erste Bestandteil der User Stories ist in beiden Fällen *As a human user*, welcher keine feature-relevante Information nach der Definition in Abschnitt 2.2 enthält.

As a human user, i want to enter a username and password, in order to log into the useraccount corresponding to the entered data.

Abbildung 4.2a: Beispiel für eine User Story aus den verwendeten Datensätzen.

As a human user, i want to enter an ID into a searchfield, in order to see all information to a specific patient.

Abbildung 4.2b: Beispiel für eine User Story aus den verwendeten Datensätzen.

Bei der Analyse der Syntaxbäume der User Stories aus den verwendeten Datensätzen ist weiterhin aufgefallen, dass die Bestandteile, die die Aktivität bzw. das Ziel beschreiben häufig aus Verbalphrasen bestehen. Verbalphrasen sind dadurch gekennzeichnet, dass sie aus einem Verb bzw. einer Verbform und weiteren Argumenten für diese bestehen. Die Argumente enthalten dabei ergänzende Informationen bzgl. des referenzierten Verbs bzw. der referenzierten Verbform. Innerhalb der Syntaxbäume der analysierten User Stories waren diese Argumente oftmals Nominalphrasen oder Präpositionalphrasen. Analog zu Verbalphrasen besteht eine Nominalphrasen oder Präpositionalphrase aus einem Nomen bzw. einer Präposition und ergänzenden Argumenten. Dies wird in den User Stories in Abbildung 4.2 deutlich. User Story *a)* enthält die Verbalphrasen *enter a username and password* und *log into the useraccount corresponding to the entered data*. Die erste Verbalphrase besteht aus dem Verb *enter* und der ergänzenden Nominalphrase *a username and password*. Die zweite Verbalphrase besteht aus dem Verb *log* und der ergänzenden Präpositionalphrase *into the Useraccount corresponding to the entered data*. Diese Bemerkung ist auch in User Story *b)* erkennbar: Hier gibt es die zwei Verbalphrasen *enter an ID into a searchfield* und *see all information to a specific patient*. Die erste Verbalphrase besteht dabei aus dem Verb *enter* und der ergänzenden Nominalphrase *an ID into a searchfield*. Die zweite Verbalphrase besteht aus dem Verb *see* und der ergänzenden Nominalphrase *all information to a specific patient*. Es ist erkennbar, dass derartige Verbalphrasen innerhalb einer User Story feature-relevante Informationen enthalten. Die heuristischen Regeln für die Extraktion feature-relevanter Informationen zielen somit darauf ab, Verbalphrasen zu extrahieren, welche aus einem Verb bzw. einer Verbform und einer ergänzenden Nominal- oder Präpositionalphrase bestehen.

4.3.2. Syntaktische Besonderheiten und sprachliche Phänomene

Während der Analyse der User Stories aus den verwendeten Datensätzen konnten, neben den bereits erwähnten POS-Mustern, einige syntaktische Besonderheiten und natürlichsprachliche Phänomene identifiziert werden, die bei der Definition der heuristischen Regeln berücksichtigt wurden. Diese Besonderheiten und Phänomene, sowie ihre Auswirkungen bei der Herleitung der Regeln, werden im folgenden Abschnitt beschrieben.

Abweichungen von der Satzschablone

Während der Analyse der User Stories aus den Datensätzen wurde festgestellt, dass diese teilweise von der genutzten Satzschablone abwichen (*As a..*, *i want to..*, *in order to..*). Zum Beispiel wurde nach dem dritten Bestandteil der User Story (Warum-Teil) ein weiterer, ergänzender Satz hinzugefügt. Auffällig ist, dass diese hinzugefügten Sätze oft Implementierungsdetails oder Spezifizierungen der beschriebenen Anforderungen enthalten. Dies ist für Implementierungs- bzw. Dokumentationszwecke sinnvoll und wichtig, jedoch nicht für die Definition eines Feature-Labels. Innerhalb der User Story in Abbildung 4.3 macht der letzte Satz eine genauere Aussage darüber, was bei der Löschung eines Profils geschehen soll, liefert jedoch keine weiteren Informationen für die Definition eines Feature-Labels, als es z.B. schon die Verbalphrase *delete the corresponding profile* macht. Aus diesem Grund wurden die heuristischen Regeln nur auf die eigentliche User Story angewandt, nicht jedoch auf eventuelle weitere ergänzende Sätze, die auf die User Story folgen. Dies wird im Rahmen der Implementierung des Prototypen insbesondere durch die Anwendung von Sentence-Segmentation sichergestellt. Sentence-Segmentation wird in Abschnitt 2.3 vorgestellt.

As an administrative user with certain access level, i want to use a delete button, in order to delete the corresponding profile. The profile should be deleted, but a „deprecated“ flag should be set.

Abbildung 4.3.: User Story mit einem weiteren ergänzenden Satz außerhalb der Satzschablone.

Aufzählungen, Ergänzungen und Kommentare

Einige der analysierten User Stories beinhalteten weiterhin Implementierungs- bzw. Spezifikationsdetails in Form von Aufzählungen, Ergänzungen und Kommentaren. Diese enthalten keine feature-relevanten Informationen und werden daher

durch die heuristischen Regeln gefiltert. Einige dieser User Stories werden im Folgenden beispielhaft aufgeführt. Die jeweiligen Abbildungen enthalten jeweils einmal die ursprüngliche User Story und einmal dieselbe User Story nachdem nicht feature-relevante Bestandteile entfernt wurden. Die User Story in Abbildung 4.4 beschreibt z.B., dass ein administrativer Nutzer in einem Profil erkennen möchte, welcher Nutzer jenes Profil erstellt hat. In den Klammern wird aufgezählt, welche Nutzerrollen ein Profil erstellen können und somit in der Profilansicht anzuzeigen sind. Dies ist ein Implementierungsdetail, welches nicht wichtig für die Definition eines Feature-labels ist.

As an administrative user with certain access level, i want to see the „created by“ entry of a profile, in order to check which user (administrator, organisation, practitioner) has created this patient profile.

Abbildung 4.4a: User Story mit einer nicht feature-relevanten Aufzählung von Nutzerrollen innerhalb der Klammer.

As an administrative user with certain access level, i want to see the „created by“ entry of a profile, in order to check which user has created this patient profile.

Abbildung 4.4b: User Story nach dem Entfernen der Aufzählung von Nutzerrollen innerhalb der Klammer.

Ebenfalls nicht von Bedeutung sind spezifizierende Ergänzungen, wie sie in den User Stories in den Abbildungen 4.5 und 4.6 gemacht werden. Im ersten Fall wird das genaue Datenformat für den zu implementierenden Service definiert. Im zweiten Fall werden die Parameter eines Matching-Algorithmus genauer beschrieben. Beides ist für die Definition eines Feature-Labels für die jeweiligen User Stories nicht relevant, da beide Informationen als Spezifikationsdetails anzusehen sind.

As a technical user with certain access level, i want to send a persons ID by POST to the service, in order to get the relationships of that person to other persons. (in JSON)

Abbildung 4.5a: User Story mit einer nicht feature-relevanten Ergänzung bzgl. des Datenformates innerhalb der Klammer.

As a technical user with certain access level, i want to send a persons ID by POST to the service, in order to get the relationships of that person to other persons.

Abbildung 4.5b: User Story nach dem Entfernen der Ergänzung bzgl. des Datenformates innerhalb der Klammer.

As an administrative user, i want to match new profiles with existing profiles, to connect profiles for the same person. Profiles, which have a matching factor in a certain range (below matching treshold but above a 2nd treshold) should be suggested for this Person.

Abbildung 4.6a: User Story mit einer nicht feature-relevanten Ergänzung bzgl. eines Parameters innerhalb der Klammer.

As an administrative user, i want to match new profiles with existing profiles, to connect profiles for the same Person. Profiles, which have a matching factor in a certain range should be suggested for this Person.

Abbildung 4.6b: User Story nach dem Entfernen der Ergänzung bzgl. eines Parameters innerhalb der Klammer.

Die User Story in Abbildung 4.7 betrifft Ergänzungen in Form von Kommentaren. In dieser User Story ergänzt der Autor, dass er sich nicht sicher ist, wie eine bestimmte Systemfunktion implementiert werden wird. Dies ist ebenfalls nicht relevant für die Definition eines Feature-Labels.

As an administrative user with certain access level, i want use a relate function (not sure how it will be implemented), in order to link the relationships between different persons (not profiles).

Abbildung 4.7a: User Story mit nicht feature-relevanter Kommentierung innerhalb der Klammer.

As an administrative user with certain access level, i want use a relate function, in order to link the relationships between different persons.

Abbildung 4.7b: User Story nach dem Entfernen der Kommentierung innerhalb der Klammer.

Allgemeine Phänomene natürlichsprachlicher Texte

User Stories stellen Anforderungsdokumente dar und werden als solche häufig in natürlicher Sprache verfasst. Aus diesem Grund treten besondere Charakteristiken von natürlicher Sprache, wie Ambiguitäten oder Redundanzen ebenfalls in User Stories auf. Während der Analyse der User Stories aus den verwendeten Datensätzen wurden teilweise ambige Formulierungen identifiziert. In der User Story in Abbildung 4.8 wird dies deutlich. In diesem Fall tritt die Ambiguität in der Formulierung *i want to POST keywords* auf. Aufgrund der Großschreibung

von *POST* ist davon auszugehen, dass mit *POST* eine besondere Form des HTTP-Requests gemeint ist (im Vergleich zu *GET*, *DELETE* etc.). Außerhalb dieses Kontextes kann *to post* aber auch im Sinne von *verschicken* verstanden werden.

As a technical user with certain access level, i want to POST keywords of a practitioner, in order to search and find practitioners. The output should be JSON.

Abbildung 4.8.: User Story mit ambiger Formulierung *POST keywords*.

Innerhalb der User Story in Abbildung 4.9 tritt eine ambige Formulierung auf, die Schwierigkeiten insbesondere beim syntaktischen Parsen der User Story verursachen kann: Für einen menschlichen Leser lässt sich aus dem Kontext der User Story schließen, dass *delete button* ein zusammengesetztes Nomen ist, *delete* kann jedoch durch den Parser als Verb *to delete* verstanden werden.

As an administrative user with certain access level, i want to use a delete button, in order to delete the corresponding profile. The profile should be deleted, but a „deprecated“ flag should be set.

Abbildung 4.9.: User Story mit ambiger Formulierung *delete button*.

Ambige Formulierungen sind nicht nur für menschliche Leser der User Story irreführend, sondern insbesondere auch für den syntaktischen Parser. Da ambige Formulierungen jedoch nur selten in den verwendeten Testdatensätzen auftraten, umfasst das entwickelte Modell keine Maßnahmen, um Ambiguitäten zu identifizieren und aufzulösen. Dasselbe gilt für das Auftreten von Redundanzen innerhalb einer User Story. Es ist in diesem Fall jedoch anzumerken, dass Redundanzen innerhalb der User Stories der verwendeten Datensätze nicht auftraten. Das liegt möglicherweise an der Nutzung von Satzschablonen und der Tatsache, dass User Stories Anforderungen in sehr komprimierter Form beschreiben.

4.3.3. Regeln zur Extraktion feature-relevanter Informationen

Im Folgenden werden die Regeln für die Extraktion feature-relevanter Informationen genauer vorgestellt. Die Extraktionsregeln können in drei Klassen unterteilt werden. Diese Klassen sind Filter-, Transformations- und die eigentlichen Extraktionsregeln. Die Anwendung der Regeln erfolgt genau in der genannten Reihenfolge. Filterregeln werden im Modell nach der Vorverarbeitung und dem syntaktischen Parsen der User Stories angewandt und entfernen für die Extraktion nicht relevante Satzbestandteile und Interpunktion. Transformationsregeln transformieren bestimmte syntaktische Muster innerhalb des Syntaxbaumes, was die Identifikation

von feature-relevanten Informationen unterstützt. Die Extraktionsregeln dienen der eigentlichen Extraktion der feature-relevanten Informationen.

Filterregeln

Filterregeln dienen insbesondere dazu die in Abschnitt 4.3.2 genannten syntaktischen Besonderheiten und sprachlichen Phänomenen aus den Syntaxbäumen herauszufiltern. Daher wurden Regeln zum Entfernen von Klammerungen und Aufzählungen mit Kommata definiert. Außerdem wurden Regeln definiert, um Artikel und die verschiedenen Arten von Pronomen wie Personal- und Possessivpronomen zu entfernen. Weiterhin wurden domänenspezifische Stoppwörter innerhalb der Testdatensätze identifiziert, die für die Extraktion feature-relevanter Informationen nicht relevant sind. Stoppwörter werden in Abschnitt 2.3 vorgestellt. Zum Einen beschreiben diese Stoppwörter Handlungen, die der Nutzer der User Story mithilfe oder anhand einer Benutzeroberfläche durchgeführt wie *click*, *enter* und *select*. Zum Anderen umfassen diese technische Systemfunktionen wie *store* und *send* oder beschreiben diese auf einem sehr hohen Abstraktionsniveau wie z.B. *use*. Dies ist für die Extraktion von feature-relevanten Informationen zu unspezifisch. Zuletzt wurden Filterregeln für Stoppwörter definiert, die sich auf die Satzschablonen von User Stories beziehen. Im Falle der in Abschnitt 2.1 vorgestellten Satzschablonen sind diese Stoppwörter *i*, *want*, *in*, *order* und *to*.

Transformationsregeln

Transformationsregeln wurden definiert, um bestimmte Knoten von Syntaxbäumen zu transformieren. Die Regeln betreffen diejenigen Knoten, welche durch den Parser mit dem Penn-Treebank-Label *SBAR* oder *S* versehen wurden. Penn-Treebank-Labels werden in Abschnitt 2.3 vorgestellt. Im Anhang B befindet sich zudem eine Übersicht aller definierten Labels. Das Label *SBAR* beschreibt einen untergeordneten Nebensatz, während das Label *S* einen Hauptsatz bestimmt. Während der Herleitung der Extraktionsregeln hat sich herausgestellt, dass es sich bei Satzbestandteilen, die mit dem Label *SBAR* oder *S* versehen wurden, oft gleichzeitig um Nominalphrasen (*NP*) handelt, die als Argument eines Verbs (*VB*) bzw. einer Verbform (*VB**) dienen. Wie in Abschnitt 4.3.2 beschrieben zielen die Extraktionsregeln für feature-relevante Informationen u.A. genau auf solche Verbalphrasen aus einem gemeinsamen Verb bzw. einer Verbform und einer ergänzenden Nominalphrase ab. Somit wurde durch die inkorrekte Vergabe der Labels *SBAR* oder *S* die Extraktion einiger feature-relevanter Information verhindert. Daher wurden Transformationsregeln definiert, die einen mit *SBAR* oder *S* gelabelte Knoten y in einen mit *NP* gelabelten Knoten, d.h. in eine Nominalphrase transformieren. Allerdings nur unter zwei Voraussetzungen: 1) Der

Knoten x, welcher auf der selben Ebene des Baumes und direkt links von y liegt, muss ein Label der Form *VB* oder *VB** haben und 2) beide Knoten x, y müssen direkte Kinder eines Knoten mit dem Label *VP*, d.h. einer Verbalphrase sein. Die Anwendung dieser Regeln wird in Abschnitt 4.3.4 nocheinmal verdeutlicht. Es entsteht somit häufig eine Verbalphrase aus einem Verb bzw. einer Verbform mit einer ergänzenden Nominalphrase als Argument. Diese kann dann wieder anhand der syntaktischen Muster extrahiert werden. Durch die Definition dieser Transformationsregeln konnte somit die Extraktion feature-relevanter Informationen verbessert werden. Es ist anzumerken, dass umfangreiche Transformationsregeln innerhalb des Syntaxbaumes wie sie in Quirchmayr et al. [27] beschrieben werden nicht angewandt wurden, da sich diese auf die Anwendung in Syntaxbäumen von Sätzen aus Benutzerhandbüchern beziehen, welche nicht so stark strukturiert sind, wie die anhand von Satzschablonen definierten User Stories.

Extraktionsregeln

Die Extraktionsregeln dienen dem eigentlichen Extrahieren der feature-relevanten Informationen innerhalb der User Stories. Wie in Abschnitt 4.3.1 beschrieben, werden im Kontext dieser Arbeit feature-relevante Informationen durch Verbalphrasen definiert, die aus einem Verb bzw. einer Verbform und einer Nominal- bzw. Präpositionalphrase als Argument bestehen. Die Extraktionsregeln extrahieren daher zwei Knoten x, y aus dem Syntaxbaum unter den folgenden Voraussetzungen: 1) x hat das Label *VB* oder *VB** (Verb bzw. Verbform) und y hat entweder das Label *NP* (Nominalphrase) oder *PP* (Präpositionalphrase), 2) x und y liegen direkt nebeneinander und auf derselben Ebene des Baumes und 3) beide Knoten müssen direkte Kinder eines Knotens mit dem Label *VP* (Verbalphrase) sein. Die Anwendung dieser Regeln wird noch einmal in Abschnitt 4.3.4 verdeutlicht. Auf der Ebene der User Story entspricht dieses syntaktische Muster dann jeweils einer feature-relevanten Information.

4.3.4. Ablauf der Exraktion feature-relevanter Informationen

In diesem Abschnitt wird die Anwendung der vorgestellten Regeln verdeutlicht. Die User Story in Abbildung 4.10 soll dabei als Beispiel dienen. Der Syntaxbaum dieser User Story ist in Abbildung 4.11 dargestellt. Die User Story ist exakt den verwendeten Datensätzen entnommen, d.h. es wurden keine orthografischen Korrekturen wie in den vorherigen Beispielen vorgenommen, um die Anwendung der Extraktionsregeln an einem realen Beispiel zu verdeutlichen.

As an technical User I want to use function on all routes in order to log all actions made via these endpoints.

Abbildung 4.10.: Beispiel User Story für die Anwendung der Extraktionsregeln.

Vorverarbeitung und syntaktisches Parsen

Bevor die heuristischen Regeln für eine User Story angewendet werden können, muss der Syntaxbaum der User Story erstellt werden. Vor dem syntaktischen Parsen der User Story erfolgen zunächst eine Reihe von Vorverarbeitungsschritten, welche im Detail in 2.3 beschrieben werden. Diese Schritte werden in dieser Reihenfolge durchgeführt: 1) Sentence-Segmentation, 2) Tokenization, 3) Lemmatization und 4) NER. Im letzten Schritt erfolgt das eigentliche Parsen der syntaktischen Struktur der Sätze, sodass am Ende der Vorverarbeitung der Syntaxbaum einer User Story vorliegt. Es ist anzumerken, dass während der Vorverarbeitung keine Stoppwörter entfernt werden, da dies das korrekte Parsen der syntaktischen Struktur der User Stories verhindern würde. Die Entfernung von Stoppwörtern geschieht erst im Rahmen der Anwendung der Filterregeln.

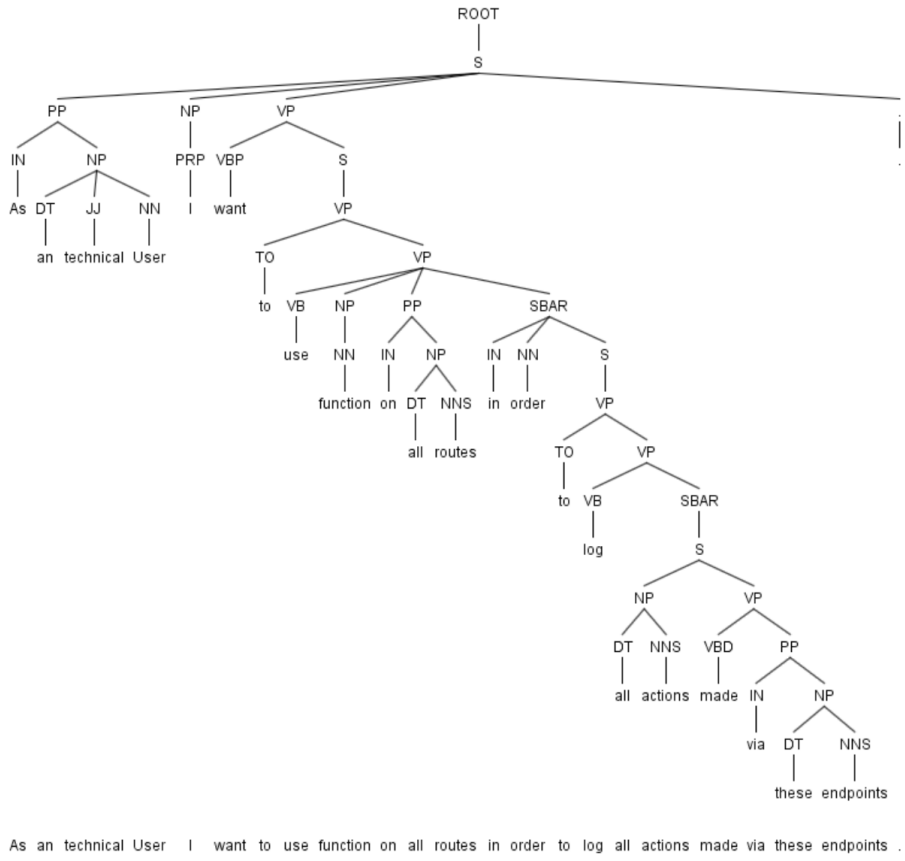

Abbildung 4.11.: Syntaxbaum der User Story.

Anwendung der Filterregeln

Wie bereits in Abschnitt 4.3.3 beschrieben, erfolgt zunächst die Anwendung der Filterregeln. In Abbildung 4.12 ist beispielhaft gezeigt, welche Knoten des Syntaxbaumes durch die Filterregeln aus der User Story entfernt werden. Dies sind diejenigen Knoten, die das Demonstrativpronomen *these*, das Indefinitpronomen *all* und den unbestimmten Artikel *an* enthalten. Zusammengefasst werden diese Knoten durch das Penn-Treebank-Label *DT*, welches Determinative repräsentiert. Ebenfalls entfernt werden Knoten, die ein Blatt haben, welches ein Stoppwort enthält. Im Falle des Beispiels sind dies die in der Satzschablone enthaltenden Wörter *in*, *order*, *i*, *want* und *to*. Ebenfalls entfernt wird der Knoten, der das Stoppwort *use* enthält. Entfernt wird auch der Punkt am Satzende. Abbildung 4.13 zeigt auf der linken Seite den Syntaxbaum der User Story aus dem Beispiel nach Anwendung der Filterregeln.

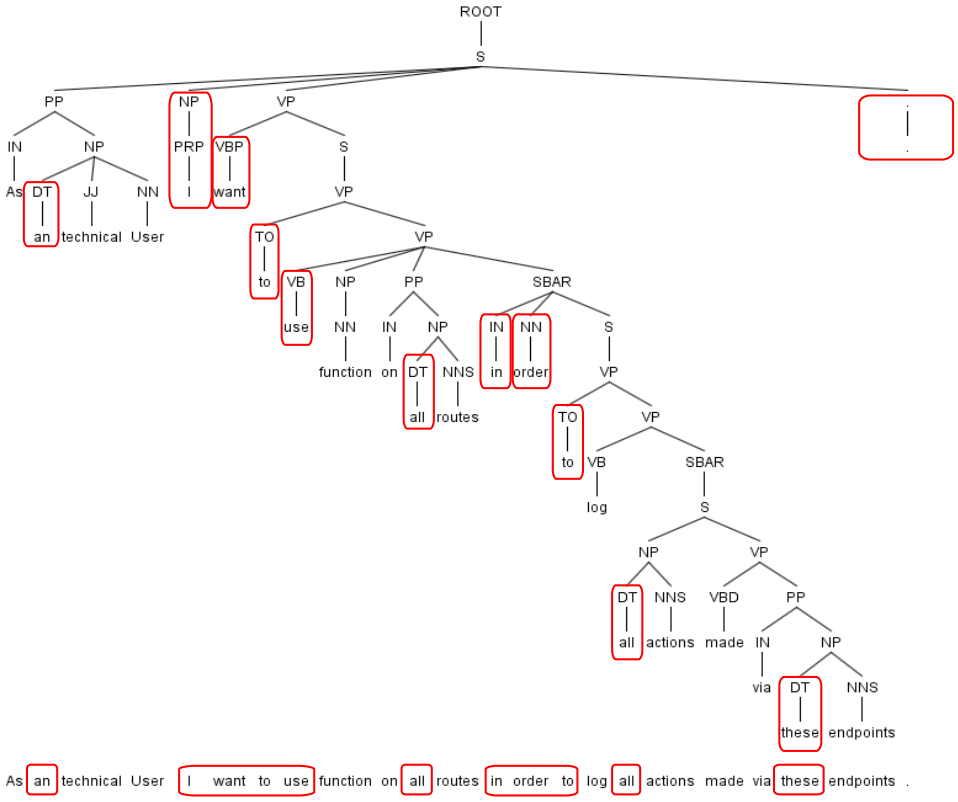

Abbildung 4.12.: Gefilterte Knoten des Syntaxbaumes der User Story.

Anwendung der Transformationsregeln

Nachdem die Filterregeln angewandt wurden, wird der Syntaxbaum mithilfe der Transformationsregeln umgeformt. In Abschnitt 4.3.3 wurde bereits die Funktionsweise der Transformationsregeln erläutert. Im Beispiels kann eine dieser Regeln angewandt werden. Der von der Regel betroffene Knoten ist in Abbildung 4.13 im linken Baum rot umrandet. Der Knoten, der den Satzteil *actions made via endpoints* umfasst, ist mit mit dem Label *SBAR* versehen und liegt auf derselben Ebene des Baumes und direkt rechts von dem Knoten, der das Token *log* umfasst und mit *VB* gelabelt wurde. Beide Knoten sind zudem direkte Kinder von einem Knoten, der mit Label *VP* versehen ist. Aus diesem Grund wird dieser Knoten vom ursprünglichen Label *SBAR* in das neue Label *NP* transformiert. Der transformierte Knoten ist in Abbildung 4.13 im rechten Syntaxbaum dargestellt und rot umrandet. Es ist anzumerken, dass der Knoten, welcher in Abbildung 4.13 im linken Syntaxbaum in gestrichelter Umrandung dargestellt ist, nicht von den Transformationsregeln betroffen ist. Aus dem folgenden Grund: Der direkt links von ihm gelegene Knoten auf der selben Ebene des Baumes, welcher den Satzbestandteil *on routes* umfasst, ist nicht mit dem Label *VB* oder *VB**versehen.

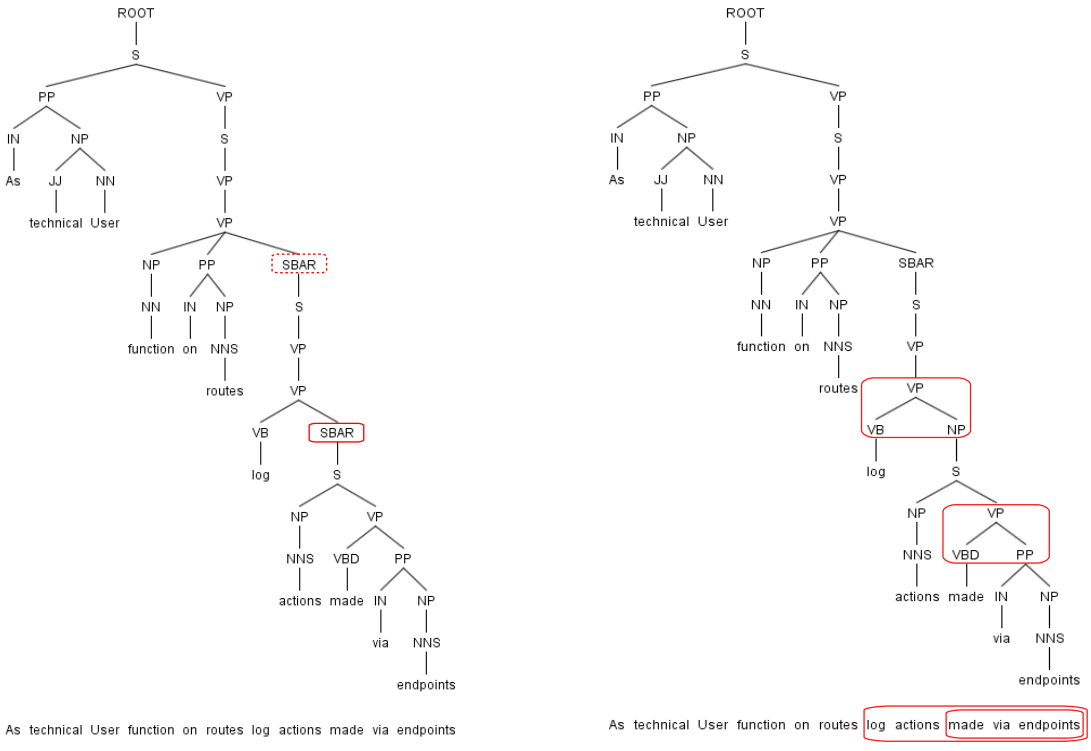

Abbildung 4.13.: Syntaxbaum nach Anwendung der Filterregeln und während der Anwendung der Transformationsregeln (links). Syntaxbaum nach Anwendung der Transformationsregeln und während der Anwendung der Extraktionsregeln (rechts).

Anwendung der Extraktionsregeln

Nach Anwendung der Filter- und Transformationsregeln werden nun die eigentlichen feature-relevanten Informationen extrahiert. Wie in Abschnitt 4.3.3 bereits beschrieben, definieren die Extraktionsregeln feature-relevante Informationen anhand von Verbalphrasen (*VP*), die aus jeweils einem Verb (*VB*) bzw. einer Verbform (*VB**) und einer Nominal- (*NP*) oder Präpositionalphrase (*PP*) bestehen. Im Syntaxbaum sind feature-relevante Informationen also durch einen Knoten mit dem Label *VP* gekennzeichnet, der zwei direkte Kinder mit den jeweiligen Labels *VB* bzw. *VB** und *NP* oder *PP* hat. Weiterhin müssen diese Kinder direkt nebeneinander liegen. Im Beispiel trifft dies auf zwei Knoten zu. Diese Knoten sind im Abbildung 4.13 im rechten Syntaxbaum dargestellt und rot umrandet. Die Blätter der extrahierten Knoten enthalten nun die eigentlichen feature-relevanten Informationen. Im Falle des Beispiels werden die feature-relevanten Informationen *log actions made via endpoints* und *made via endpoints* extrahiert, welche in Abbildung 4.13 rot umrandet sind. Auffällig ist, dass die extrahierten Informationen redundant sind, denn die Information *made via endpoints* ist bereits in *log actions made via endpoints* enthalten. Die redundanten Informationen werden in einem Nachbearbeitungsschritt entfernt. Die aus der User Story gewonnene feature-relevante Information ist somit *log actions made via endpoints*.

4.4. Vorschlagen von Feature-Labels

Im folgenden Abschnitt wird die dritte Komponente des Modells beschrieben, die für eine unbekannte User Story anhand bereits bekannter und gelabelter User Stories ein Feature-Label vorschlägt. Diese Komponente umfasst im Wesentlichen ein KNN, das anhand der bekannten und bereits gelabelten User Stories trainiert wird. Die Eingabedaten in die dritte Komponente des Modells sind eine unbekannte und zu klassifizierende User Story sowie die Menge der bekannten und bereits gelabelten User Stories. KNN werden in Abschnitt 2.4 vorgestellt.

4.4.1. Vorverarbeitung und Berechnung von Termvektoren

Ähnlich zu den Vorverarbeitungsschritten innerhalb der Komponente für das Extrahieren feature-relvanter Informationen, werden die Eingabedaten in diese Komponente vorverarbeitet. Im Rahmen der Vorverarbeitung werden die folgenden Schritte durchgeführt, welche in Abschnitt 2.3 beschrieben werden: (1) Tokenization, (2) Stemming und (3) Stopword-Removal. Es gibt jedoch einige Unterschiede bzgl. der Vorverarbeitungsschritte der zweiten Komponente. So werden in dieser Komponente z.B. Stoppwörter entfernt. Für das Parsen und die Extraktion

feature-relevanter Informationen ist die syntaktische Struktur der User Stories von Bedeutung, welche durch das Entfernen von Stopwörtern zerstört werden würde. Dagegen ist bei der Klassifikation der User Stories die Semantik von Interesse. Stoppwörter liefern diesbezüglich keine Informationen und können daher entfernt werden. Ein weiterer Unterschied ist, dass das entwickelte Modell für die Klassifikation die gesamte User Story samt ergänzender Sätze berücksichtigt. Außerdem wird für die Klassifikation auch der Titel der User Story genutzt. Der Titel stellt meistens eine Art Überschrift bzw. Zusammenfassung der User Story dar und enthält wichtige semantische Informationen. Aus dem Titel, der eigentlicher User Story und evtl. weiterer ergänzender Sätze wird ein User-Story-Dokument erstellt. Dazu werden ergänzende Sätze und Titel an den Text der User Story angehängt. Während der Entwicklung des Ansatzes konnte festgestellt werden, dass diese künstlichen User-Story-Dokumente für bessere Klassifikationsergebnisse sorgen als die alleinige Nutzung der eigentlichen User Stories. Wie in Abschnitt 4.3.3 beschrieben wurden für die Extraktion feature-relevanter Informationen dagegen zusätzliche Sätze verworfen und nur die eigentliche User Story selbst ohne ihre Überschrift bzw. ihren Titel genutzt. Hier trug die Berücksichtigung des Titels und ergänzender Sätze zu weniger aussagekräftigen extrahierten feature-relevanten Informationen bei, während sie im Falle der Klassifikation wichtige semantische Informationen liefern. Nach Durchführung der genannten Vorverarbeitungsschritte werden die erstellten User-Story-Dokumente in Termvektoren umgeformt, wobei die einzelnen Terme nach dem tf-idf-Gewichtungsschema gewichtet werden.

4.4.2. Klassifikation für das Vorschlagen von Feature-Labels

Die Menge der bekannten und gelabelten User Stories wird nach Durchführung der vorgestellten Vorverarbeitungsschritte als Trainingsdatensatz für das KNN genutzt. Dabei stellen die gewichteten Termvektoren der einzelnen User-Story-Dokumente zusammen mit den vergebenen Feature-Labels die Eingabe in das KNN für das Training dar. Die einzelnen Klassen sind durch die vergebenen Feature-Labels definiert. Es handelt sich hierbei um ein Klassifikationsproblem mit mehreren Klassen. Zur Lösung von Klassifikationsproblemen gibt es im Bereich des ML verschiedene Modelle. Es wurden während der Entwicklung des Modells verschiedene Klassifikationsmodelle berücksichtigt und evaluiert, darunter der Naive-Bayes-Klassifikator und KNN. Die KNN zeigten dabei bessere Klassifikationsergebnisse als der Naive-Bayes-Klassifikator, sodass diese im vorliegenden Modell eingesetzt werden. Die Struktur des genutzten KNN besteht aus einer Eingabe- und Ausgabeschicht sowie einer versteckten Schicht mit 25 Knoten. Die Ausgabeschicht hat dabei genau so viele Knoten, wie es Klassen d.h. Feature-Labels in der Menge der bekannten User Stories gibt. Die Anzahl der Knoten der versteckten Schicht wurde während der Entwicklung des Modells durch informelle Evaluation der Klassifikationsergebnisse definiert. Eine Erhöhung, sowohl der Anzahl der Knoten als auch der Menge der

versteckten Schichten führte im Rahmen der Evaluation mit den verwendeten Datensätzen nicht zu besseren Klassifikationsergebnissen. Ein Feature-Label für eine unbekannte User Story wird folgendermaßen vorgeschlagen: Die Eingabe in das KNN besteht aus dem gewichteten Termvektor des unbekannten User-Story-Dokuments. Die Ausgabe des KNN entspricht einer Wahrscheinlichkeitsverteilung über die Menge der bekannten Feature-Labels. Das in dieser Arbeit entwickelte Modell schlägt genau das Feature-Label für die unbekannte User Story vor, dessen Wahrscheinlichkeit am höchsten ist.

Während der Entwicklung des vorliegenden Modells wurden zur Identifikation semantisch ähnlicher User Stories ebenfalls Clustering-Ansätze und Topic Modelling Ansätze berücksichtigt. Aus dem Bereich des nicht überwachten Lernens wurde das k-means-Verfahren getestet, um Cluster semantisch ähnlicher User Stories zu identifizieren. Es stellte sich jedoch heraus, dass das Vorschlagen von Feature-Labels anhand der Labels von User Stories aus demselben Cluster deutlich schlechtere Ergebnisse lieferte, als das Nutzens eines KNN. Ebenfalls untersucht wurde das Vorschlagen von Labels anhand der Feature-Labels von User Stories, die im Rahmen des Topic Modeling eine ähnliche Themenverteilung erhielten. Auch hier zeigte sich ein schlechteres Ergebnis bzgl. der vorgeschlagenen Feature-Labels im Vergleich zur Nutzung eines Klassifikationsmodells. Außerdem sind in beiden Ansätzen Parameter zu definieren, die A-priori-Wissen über die Menge der bekannten User Stories erfordern. Im Falle von Clustering ist dies die Anzahl der vorhandenen Cluster und im Falle des Topic Modeling die Anzahl der beschriebenen Themen. Beides entspricht im Kontext dieser Arbeit der Anzahl der tatsächlich durch die User Stories beschriebenen Features.

4.5. Validierung und Vergabe eines Feature-Labels

Innerhalb der vierten Komponente des Modells erfolgt schließlich die Vergabe des Feature-Labels für die unbekannte User Story. Das Modell extrahiert aus der unbekannten User Story und den zu ihr semantisch ähnlichen User Stories die feature-relevanten Informationen. Außerdem schlägt das Modell ein Feature-Label durch das anhand der bekannten User Stories trainierte KNN vor. Dem Nutzer des Modells werden, sowohl die feature-relevanten Informationen, als auch das vorgeschlagene Feature-Label präsentiert. Die feature-relevanten Informationen enthalten ein Relevanz-Maß, das auf der Kosinus-Ähnlichkeit der unbekannten User Story mit jeweils einer der zu ihr semantisch ähnlichen User Stories basiert. Das vorgeschlagene Feature-Label erhält durch das KNN eine Wahrscheinlichkeit, die die Sicherheit des KNN für das vorgeschlagene Label angibt. Der Nutzer kann das vorgeschlagene Feature-Label anhand dessen Wahrscheinlichkeit und der feature-relevanten Informationen sowie deren Relevanz-Maß validieren. Die Validierung kann dabei das Abgleichen der feature-relevanten Informationen mit

dem vorgeschlagenen Feature-Label umfassen. Stimmen die feature-relevanten Informationen dabei semantisch mit dem vorgeschlagenen Feature-Label überein, d.h. diese beschreiben das Software-Feature, welches durch das Label definiert wird, kann der Nutzer sich entscheiden das vorgeschlagene Label für die unbekannte User Story zu übernehmen. Tritt beim Abgleichen ein Konflikt auf, d.h. die feature-relevanten Informationen passen semantisch nicht zu dem vorgeschlagenen Feature-Label, muss der Nutzer abwägen: Treffen die feature-relevanten Informationen oder das vorgeschlagene Feature-Label das eigentliche durch die User Story beschriebene Software-Feature besser. Im ersten Fall muss der Nutzer anhand der feature-relevanten Informationen manuell ein Label für die User Story vergeben. Im zweiten Fall kann der Nutzer das vorgeschlagene Label wiederum übernehmen. In allen beschriebenen Fällen erfolgt die Vergabe des Feature-Labels für eine unbekannte User Story somit manuell, jedoch mit erheblicher Unterstützung durch das entwickelte Modell. Nach der (semi-)automatischen Vergabe des Feature-Labels wird die unbekannte User Story in jedem Fall der Menge der bekannten User Stories hinzugefügt. Somit fließt das durch den Nutzer bei der Vergabe des Labels genutzte semantische Wissen iterativ in das Modell ein: Zum Einen bei der Bestimmung der Menge der semantisch ähnlichen User Stories für eine unbekannte User Story. Zum Anderen beim Training des KNN, für welches die Menge der bekannten und gelabelten User Stories genutzt wird.

5. Implementierung des Prototypen

In dem folgenden Kapitel wird die Implementierung des in Kapitel 4 vorgestellten Modells als Jira-Prototyp beschrieben. Das Kapitel ist in die folgenden Abschnitte gegliedert: Abschnitt 5.1 beschreibt die Anforderungserhebung für den Prototypen. Abschnitt 5.2 erläutert die Herleitung des Entwurfs aus den Anforderungen. In Abschnitt 5.3 wird die eigentliche Implementierung beschrieben. Abschnitt 5.4 fasst die zur Qualitätssicherung durchgeführten Tests zusammen.

5.1. Anforderungserhebung

Vor dem Entwurf wurden die Anforderungen an die Implementierung des Modells als Jira-Prototyp erhoben. Sowohl die funktionalen als auch die Qualitätsanforderungen wurden in Form von User Stories dokumentiert. Diese sind in dem zu dieser Arbeit zugehörigen Jira-Projekt einsehbar. Die wesentlichen Aspekte, die von den erhobenen Anforderungen abgedeckt werden, sind folgend beschrieben. Das in dieser Arbeit entwickelte Modell stellt einen (semi-)automatischen Ansatz zur Vergabe von Feature-Labels für User Stories vor. Dies erreicht das Modell durch 1) die Extraktion der feature-relevanten Informationen für eine User Story und 2) dem Vorschlagen von Feature-Labels für User Stories. Die funktionalen Anforderungen an die Implementierung fokussieren sich deshalb im Wesentlichen auf die Umsetzung dieser Funktionalität. Speziell für die Implementierung des Jira-Prototypen beziehen sich die funktionalen Anforderungen aber auch auf Aspekte der Darstellung und Anzeige der feature-relevanten Informationen und vorgeschlagenen Feature-Labels.

Die Qualitätsanforderungen konzentrieren sich auf die Bereiche Benutzbarkeit, Effizienz und Wartbarkeit. Die Benutzbarkeit ist insbesondere wichtig, da das Modell auf die Unterstützung von Nutzern bei der Vergabe von Feature-Labels für User Stories ausgerichtet ist. Die Implementierung des Modells als Jira-Prototyp muss somit in hohem Maße nutzerorientiert erfolgen. Insbesondere sind Aspekte der Aufgabenangemessenheit entscheidend wie z.B. das einfache Zuweisen von Feature-Labels an User Stories und die übersichtliche Darstellung der Relevanz von feature-relevanten Informationen und der Wahrscheinlichkeit von vorgeschlagenen Feature-Labels. Die Implementierung des Modells muss weiterhin effizient sein, da

z.B. hohe Wartezeiten für die Nutzer während der Extraktion der feature-relevanten Informationen und dem Vorschlagen der Feature-Labels nicht praktikabel sind. Bezüglich der Wartbarkeit sind die folgenden Aspekte von Bedeutung: Das in dieser Arbeit entwickelte Modell setzt zur Extraktion der feature-relevanten Informationen heuristische Regeln ein, welche auf spezifischen POS-Mustern basieren. Es ist anzunehmen, dass diese je nach Projektkontext bzw. verwendetem Datensatz von User Stories angepasst werden müssen, um qualitativ hochwertige Ergebnisse bei der Extraktion feature-relevanter Informationen zu erzielen. Die Implementierung des Modells muss Entwicklern daher die Möglichkeit bieten, unkompliziert weitere Extraktionsregeln zu definieren. Für das Vorschlagen von Feature-Labels nutzt das Model ein KNN. Die Qualität der vorgeschlagenen Feature-Labels ist dabei sehr von dem verwendeten Klassifikationsmodell abhängig. Daher muss es Entwicklern auf eine einfache Art und Weise möglich sein, weitere Klassifikationsmodelle zu implementieren, um das Modell an einen spezifischen Datensatz anzupassen oder für den zukünftigen Gebrauch mit diesem zu optimieren.

5.2. Herleitung des Entwurfs

Nach der Anforderungserhebung wurde ein Entwurf der Implementierung aus den Anforderungen abgeleitet. Es wird dabei die Implementierung des eigentlichen Modells und die Implementierung des Jira-Plugins unterschieden. Die Klassen des Modells befinden sich im Paket *de.schindler.ma.freshman.model*. Eine Übersicht der Klassen aus diesem Paket gibt Abbildung 5.1. Im Paket *de.schindler.ma.freshman.plugin* befinden sich die Klassen des Plugins. Diese werden in Abbildung 5.4 dargestellt. Die Verantwortlichkeiten sind dabei wie folgt: Die Klassen des Pakets *de.schindler.ma.freshman.model* implementieren die wesentliche Funktionalität des in dieser Arbeit entwickelten Modells. Die Klassen aus dem Paket *de.schindler.ma.freshman.plugin* implementieren plugin-spezifische Funktionalität wie das Anzeigen der Ausgabegrößen des Modells auf der Benuteroberfläche und die Weiterleitung von Eingabe-Events durch Nutzer an das Modell.

Die wesentliche Funktionalität des in dieser Arbeit entwickelten Modells ist 1) die Extraktion feature-relevanter Informationen und 2) das Vorschlagen von Feature-Labels. Während des Entwurfs konnten vier (Teil-)Funktionalitäten identifiziert werden, die zur Umsetzung der übergeordneten Funktionalität des Modells notwendig sind: 1) Das syntaktische Parsen von User Stories, 2) die Extraktion feature-relevanter Informationen, 3) die Berechnung der Termvektoren von User Stories sowie die Bestimmung semantisch ähnlicher User Stories und 4) das Vorschlagen von Feature-Labels. Diese Funktionalität wurde jeweils auf die vier wesentlichen Klassen *SyntaxParser*, *FeatureRelevantInformationStore*, *WordVectorStore* und *Classifier* abgebildet. Die eigentliche Funktionalität wurde dabei häufig mithilfe

externer Bibliotheken implementiert. Die jeweils verwendeten Bibliotheken sind in der zugehörigen Klassenbeschreibung in Unterabschnitt 5.3.1 aufgeführt. Um einen hohen Grad an Wartbarkeit zu erreichen, wurden die Klassen so entworfen, dass ihre jeweilige (Teil-)Funktionalität unabhängig voneinander aufgerufen werden kann. Die Methoden der Klassen sind so definiert, dass sie eine übersichtliche Schnittstelle definieren. Die Klassen zeigen somit einen hohen Grad an Kohäsion und einen geringen Grad der Kopplung.

Um jedoch einen einfachen Zugriff auf die übergeordnete Funktionalität des Modells zu erreichen, wurde die Klasse *LabelAndFeatureInformationManager* definiert, die den Zugriff auf die beschriebenen Klassen kapselt und somit eine zentrale Schnittstelle für Clients darstellt. Um die Wartbarkeit der Implementierung weiter zu erhöhen, wurden Klassen des Modells, die sich in zukünftigen Versionen des Modells ändern können, als abstrakte Klassen definiert. Abstrakte Klassen können nicht instanziiert werden, beschreiben aber eine allgemeine Schnittstelle von Methoden, die abgeleitete Klassen implementieren müssen. Klassen, die als abstrakte Klassen definiert wurden sind *ParseTreeTransformationRule*, *ParseTreeExtractionRule* und *Classifier*. Dies ist hauptsächlich in den Anforderungen begründet, die eine einfache Erweiterbarkeit des Modells bzgl. weiterer Extraktionsregeln für feature-relevante Information und weiterer Klassifikationsmodelle für das Vorschlagen von Feature-Labels beschreiben.

Die Ausgabe auf der Benutzeroberfläche des Plugins wurde so entworfen, dass sowohl die feature-relevante Information, als auch die vorgeschlagenen Feature-Labels als Balkendiagramme angezeigt werden. Die Höhe und die Farbgebung der Balken entspricht dabei dem Relevanz-Maß einer feature-relevanten Information bzw. der Wahrscheinlichkeit eines vorgeschlagenen Feature-Labels. Die Ausgabe eines Relevanz-Maßes und einer Wahrscheinlichkeit ergibt sich dabei aus den Anforderungen. Durch diese Art der Präsentation wird ein hoher Grad an Aufgabenangemessenheit erreicht, da sowohl eine schnelle Beurteilung der tatsächlichen Relevanz einer feature-relavanten Information, als auch die schnelle Identifikation des wahrscheinlichsten Feature-Labels ermöglicht wird.

Um der Qualitätsanforderung der Effizienz gerecht zu werden, wurden Klassen, deren Initialisierung besonders ressourcenaufwendig z.B. in Bezug auf den Speicherbedarf oder die Laufzeit ist, mithilfe des Singleton-Architektur-Musters implementiert. Das Singleton-Muster verhindert, dass mehrere Instanzen einer Klasse gebildet werden können und erzwingt, dass immer dieselbe Referenz auf eine einzige Instanz der Klasse zurückgegeben wird. Somit wird ein übermäßiger Verbrauch von Ressourcen durch mehrfache Initialisierung verhindert und die Wiederverwendung einer einzigen Instanz gefördert. Klassen, die das Singleton-Muster implementieren, sind insbesondere die Klassen *SytnaxParser* und *LabelAndFeatureInformationManagerStore*. Aus den Qualitätsanforderungen bzgl. der Effizienz des Plugins ergibt sich weiterhin, dass die Extraktion von feature-

relevanten Informationen und das Vorschlagen von Feature-Labels in hinreichend schneller Zeit erfolgen muss. Daher verwenden Klassen, die diese Funktionalität implementieren, Caching-Mechanismen, um einmal extrahierte feature-relevante Informationen und berechnete Termvektoren vorzuhalten. Insbesondere sind dies die Klassen *FeatureRelevantInformationStore*, *WordVectorStore* und die Klasse *ArtificialNeuralNetwork*, welche die abstrakte Klasse *Classifier* implementiert.

5.3. Beschreibung der Implementierung

Im folgenden Abschnitt wird die eigentliche Implementierung des Modells und des Plugins beschrieben. In Unterabschnitt 5.3.1 wird die Implementierung des Modells vorgestellt. Die Interaktion der Klassen des Modells wird in Unterabschnitt 5.3.2 erläutert. Unterabschnitt 5.3.3 beschreibt die Implementierung des Plugins. In Abschnitt 5.3.4 wird die Interaktion der Klassen des Plugins mit denen des Modells dargestellt.

5.3.1. Implementierung des Modells

Das eigentliche Modell wurde im Rahmen dieser Arbeit in der Programmiersprache Java implementiert. Die Implementation des Modells umfasst alle Klassen aus dem Paket *de.schindler.ma.freshman.model*. Die statische Struktur der wesentlichen Klassen der Implementierung des Modells ist in dem Klassendiagramm in Abbildung 5.1 dargestellt. Die einzelnen Klassen werden im Folgenden erläutert.

SyntaxParser

Die Klasse *SyntaxParser* erlaubt das Parsen der syntaktischen Struktur einer User Story und die Rückgabe eines entsprechenden Syntaxbaumes. Dies wird durch die Funktion *getParseTreeOf(userStoryDescription)* realisiert, deren Argument ein String mit dem Inhalt der zu parsenden User Story ist. Die Rückgabe der Funktion ist der geparste Syntaxbaum der übergebenden User Story. Das syntaktische Parsen und die dazu notwendigen Vorverarbeitungsschritte sind mithilfe der StanfordNLP-Bibliothek [19] implementiert. Der genutzte Parser extrahiert den Syntaxbaum einer User Story im Rahmen der Phrasenstrukturgrammatik. Die StanfordNLP-Bibliothek enthält zu diesem Zweck ein vortrainiertes Modell für die englische Sprache. Die Konstruktion dieses Modells ist bzgl. der Laufzeit und des Speicherverbrauchs teuer, sodass die Klasse *SyntaxParser* mithilfe des Singleton-Musters implementiert wurde. Eine entsprechende Instanz der Klasse kann mit der Funktion *getInstance()* referenziert werden.

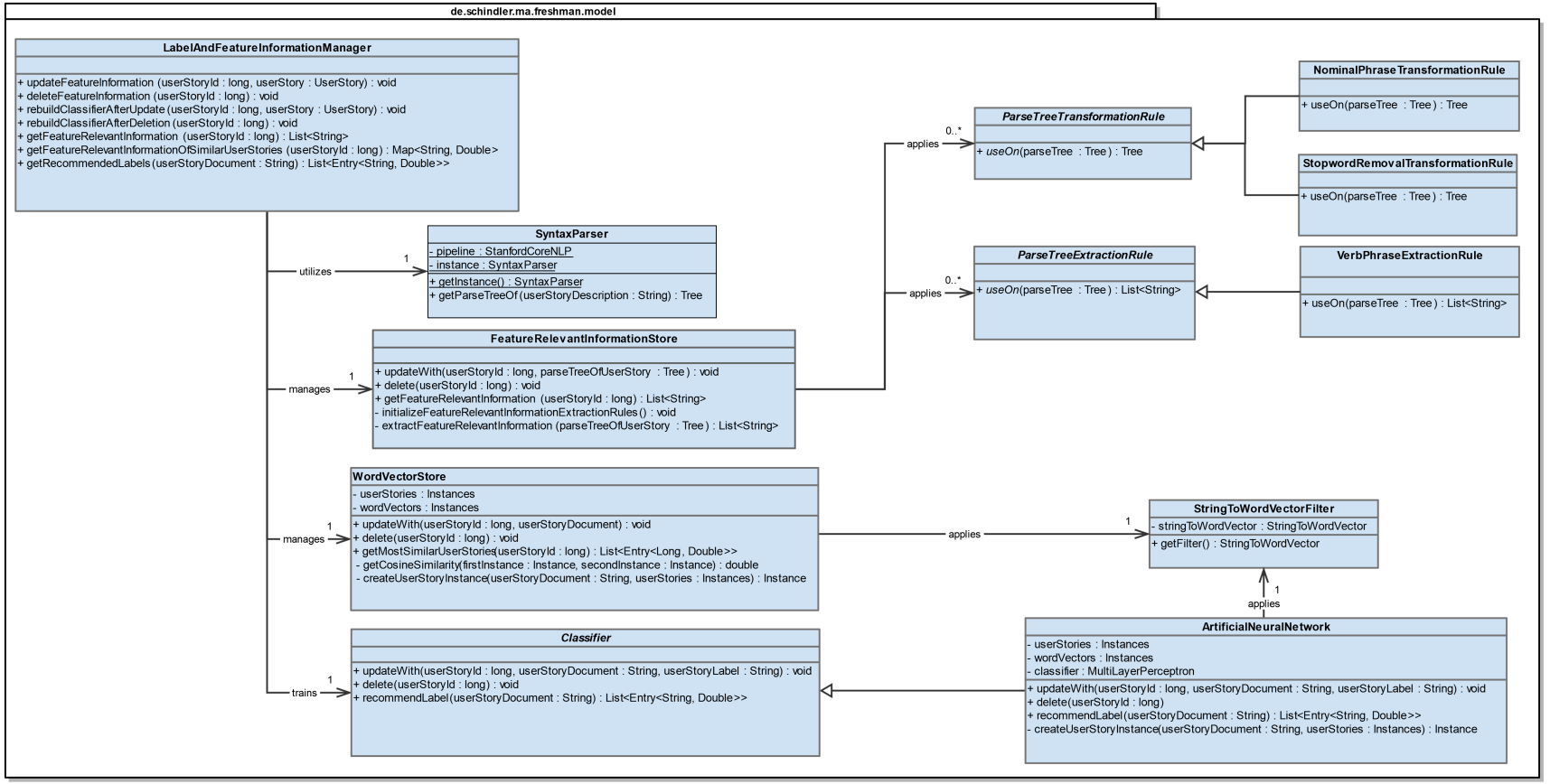

Abbildung 5.1.: Klassendiagramm der Implementierung des Modells.

FeatureRelevantInformationStore

Diese Klasse extrahiert und speichert die feature-relevanten Informationen von User Stories. User Stories können mithilfe der Methoden *updateWith(userStoryId, parseTreeOfUserStory)* bzw. *delete(userStoryId)* dem zugrunde liegenden Container hinzugefügt bzw. aus diesem entfernt werden. Anzumerken ist, dass die Funktion *updateWith()* den Syntaxbaum einer User Story erwartet. Dieser kann durch die oben beschriebene Klasse *SyntaxParser* erhalten werden. Wird eine User Story hinzugefügt, werden intern die feature-relevanten Informationen extrahiert und im Container gespeichert. Diese können dann mit der Funktion *getFeatureRelevantInformation(userStoryId)* abgefragt werden. Für die eigentliche Extraktion wurden die Werkzeuge *Tregex* und *Tsurgeon* aus der StanfordNLP-Bibliothek [19] genutzt. Mithilfe von *Tregex* können, ähnlich zu regulären Ausdrücken, bestimmte Muster innerhalb von Syntaxbäumen identifiziert werden. Knoten des Syntaxbaumes, die einem bestimmten Muster entsprechen, können dann durch *Tsurgeon* manipuliert werden. Klassen, die die in Abschnitt 4.3.3 beschriebenen Extraktionsregeln des Modells implementieren, nutzen die genannten Werkzeuge. Wie in Abschnitt 4.3.3 beschrieben umfasst das Modell drei Arten von verschiedenen Extraktionsregeln: 1) Filterregeln, 2) Transformationsregeln sowie 3) die eigentlichen Extraktionsregeln. Filter-und Transformationsregeln werden durch die Klassen *StopwordRemovalTransformationRule* bzw. *NominalPhraseTransformationRule* implementiert. Beide erben von der abstrakten Klasse *ParseTreeTransformationRule*, die die abstrakte Funktion *useOn(parseTree)* definiert. Diese erwartet als Argument einen Syntaxbaum und gibt den transformierten Syntaxbaum zurück. Die eigentlichen Extraktionsregeln für feature-relevante Information werden in der Klasse *VerbPhraseExtractionRule* implementiert. Diese Klasse erbt von der abstrakten Klasse *ParseTreeExtractionRule*. In dieser wird ebenfalls eine abstrakte Funktion *useOn(parseTree)* definiert. Auch diese erwartet als Argument den Syntaxbaum einer User Story. Im Unterschied zur oben genannten Funktion *useOn()* der abstrakten Klasse *ParseTreeTransformationRule*, gibt diese aber keinen transformierten Syntaxbaum zurück, sondern die extrahierten feature-relevanten Informationen einer User Story. Für die Extraktion greift die Klasse *FeatureRelevantInformationStore* auf Instanzen der Klassen *StopwordRemovalTransformationRule*, *NominalPhraseTransformationRule* und *VerbPhraseExtractionRule* zu.

WordVectorStore

Die Klasse *WordVectorStore* berechnet und speichert die in 4.2 beschriebenen Termvektoren von User Stories. Weiterhin realisiert sie das Identifizieren von semantisch ähnlichen User Stories. Mithilfe der Methoden *updateWith(userStoryId, userStoryDocument)* und *delete(userStoryId)* können User-Story-Dokumente dem

zugrunde liegenden Container hinzugefügt bzw. aus diesem entfernt werden. Wird ein User-Story-Dokument dem Container hinzugefügt, berechnet die Klasse *WordVectorStore* intern den jeweiligen Termvektor und speichert diesen im Container ab. Die eigentliche Berechnung der Termvektoren wird durch die Bibliothek WEKA [9] aus dem Bereich Data-Mining und ML realisiert. Die Bibliothek erlaubt die Definition von Filtern, mit denen Eingabedaten vorverarbeitet werden können. Im Rahmen dieser Arbeit wurde ein Filter implementiert, der User-Story-Dokumente unter Berücksichtigung der in Abschnitt 4.2 beschriebenen Vorverarbeitungsschritte in ihre jeweiligen Termvektoren transformiert. Für die Implementierung wurde auf bereits existierende Filter aus der Bibliothek WEKA wie z.B. einem Filter für tf-idf-Gewichtung zurückgegriffen. Der komplette Filter ist in der Klasse *StringToWordVectorFilter* implementiert. Nach dem Hinzufügen eines User-Story-Dokuments zum Container, können über die Methode *getMostSimilarUserStories(userStoryId)* semantisch ähnliche User Stories für eine User Story abgefragt werden. Dazu wird klassenintern die Kosinus-Ähnlichkeit der im Argument der Methode übergebenden User Story und aller im Container gespeicherten User Stories berechnet. Die Top-N der bzgl. der Kosinus-Ähnlichkeit semantisch ähnlichsten User Stories wird dann durch die Methode zurückgegeben. Wie in Abschnitt 4.2 beschrieben wurde im Kontext dieser Arbeit $N = 3$ definiert.

ArtificialNeuralNetwork

Die Klasse *ArtificialNeuralNetwork* realisiert das voll-automatische Vorschlagen von Feature-Labels für User Stories. Das Vorschlagen wird durch die Klassifikation von User Stories durch ein KNN erreicht. Das KNN wurde mithilfe der Klasse *MultiLayerPerceptron* aus der Bibliothek WEKA [9] implementiert. Mit den Methoden *updateWith(userStoryId, userStoryDocument, userStoryLabel)* und *delete(userStoryId)* können User-Story-Dokumente dem Trainingsdatensatz des KNN hinzugefügt bzw. aus diesem entfernt werden. Das Training des KNN wird dabei bei jedem Aufruf einer der beiden genannten Methoden erneut angestoßen. Dadurch wird erreicht, dass der Trainingszustand des KNN zu jedem Zeitpunkt dem aktuellen Datensatz von User Stories entspricht. Durch die Methode *recommendLabel(userStoryDocument)* kann dann ein Feature-Label für das übergebene User-Story-Dokument vorgeschlagen werden. Dazu gibt die Methode die vorgeschlagenen Feature-Labels sowie deren Wahrscheinlichkeiten zurück. Wie in Abschnitt 4.4.1 beschrieben, erfolgt das Training und der Klassifikationsvorgang des KNN mithilfe der Termvektoren von User Stories. Bevor also ein User-Story-Dokument dem Trainingsdatensatz durch die Methode *updateWith()* hinzugefügt oder ein Feature-Label mit der Methode *recommendLabel()* vorgeschlagen wird, werden die als Argument übergebenen User-Story-Dokumente in ihre jeweiligen Termvektoren transformiert. Dafür greift die Klasse *ArtificialNeuralNetwork* wie die in Abschnitt 5.3.1 beschriebene Klasse *WordVectorStore* auf die Klasse *StringToWordVectorFilter* zurück.

LabelAndFeatureInformationManager

Die Klasse *LabelAndFeatureInformationManager* kapselt den Zugriff auf die Funktionen der oben beschriebenen Klassen. Wird eine Instanz der Klasse *LabelAndFeatureInformationManager* erstellt, werden ebenfalls Instanzen der genannten Klassen erstellt und referenziert. Daher stellt sie für Clients die primäre Schnittstelle dar, um auf die im Modell implementierte Funktionalität zuzugreifen. Durch die Methoden *updateFeatureInformation(userStoryId, userStory)* und *deleteFeatureInformation(userStoryId)* können den Klassen *FeatureRelevantInformationStore* und *WordVectorStore* neue User Stories hinzugefügt bzw. User Stories aus diesen entfernt werden. Nach dem Hinzufügen einer User Story können die jeweiligen feature-relevanten Informationen abgefragt werden. Die Methode *getFeatureRelevantInformation(userStoryId)* gibt genau die feature-relevanten Informationen der übergebenen User Story zurück. Die Methode *getFeatureRelevantInformationOfSimilarUserStories(userStoryId)* gibt die feature-relevanten Informationen der zu der übergebenen User Story semantisch ähnlichsten User Stories zurück. Mithilfe der Methoden *rebuildClassifierAfterUpdate(userStoryId, userStory)* und *rebuildClassifierAfterDeletion(userStoryId)* können User Stories dem Trainingsdatensatz des KNN hinzugefügt bzw. aus diesem entfernt werden. Gleichzeitig wird das erneute Training des KNN anhand des veränderten Trainingsdatensatzes angestoßen. Nach dem Training des KNN können mit der Methode *getRecommendedLabels(userStoryDocument)* für das übergebene User-Story-Dokument Feature-Labels vorgeschlagen werden.

5.3.2. Interaktion der Klassen des Modells

Im folgenden Abschnitt werden die Kommunikation und der Austausch von Nachrichten zwischen den wesentlichen Klassen der Implementierung des Modells während der Ein- und Ausgabe beschrieben. Als Eingabe wird der Aufruf der Methoden *updateFeatureRelevantInformationen(id, userStory)* und *rebuildClassifierAfterUpdate(id, userStory)* der Schnittstellen-Klasse *LabelAndFeatureInformationManager* verstanden. Abbildung 5.2 zeigt das Sequenzdiagramm in dem Fall, dass ein Client diese Methoden aufruft. Als Ausgabe wird der Aufruf der Methoden *getFeatureRelevantInformation(id)*, *getFeatureRelevantInformationOfSimilarUserStories(id)* und *getRecommendedLabels(id)* verstanden. Das Sequenzdiagramm in Abbildung 5.3 zeigt den Fall, dass ein Client diese Methoden aufruft. Beide Sequenzdiagramme werden folgend im Detail beschrieben.

Interaktion während der Eingabe

Im Sequenzdiagramm der Abbildung 5.2 ruft ein Client zunächst die Methode *updateFeatureRelevantInformation(id, userStory)* der Klasse *LabelAndFeatureInformationManager* auf. Der Client möchte also eine User Story den Klassen *FeatureRelevantInformationStore* und *WordVectorStore* hinzufügen. Aus der Klasse *LabelAndFeatureInformationManager* wird jetzt die Methode *getParseTreeOf(userStory)* der Klasse *SyntaxParser* aufgerufen, um den Syntaxbaum der übergebenen User Story zu erhalten. Nach dem Parsen der User Story gibt diese den resultierenden Syntaxbaum und die Kontrolle an die Klasse *LabelAndFeatureInformationManager* zurück. Diese ruft nun die Methode *updateWith(id, syntaxTree)* der Klasse *FeatureRelevantInformationStore* auf, um feature-relevante Informationen aus dem übergebenen Syntaxbaum zu extrahieren und diese dann abzuspeichern. Die Klasse selbst ruft dazu intern die Methode *extractFeatureRelevantInformation(syntaxTree)* auf. Nach Ablauf der Methode, d.h. nach der Extraktion und Speicherung der feature-relevanten Informationen, gibt die Klasse *FeatureRelevantInformationStore* die Kontrolle wieder an die Klasse *LabelAndFeatureInformationManager* zurück. Diese ruft jetzt die Methode *updateWith(id, userStory)* der Klasse *WordVectorStore* auf, um den Termvektor der übergebenen User Story zu berechnen und diesen abzuspeichern. Die Klasse *WordVectorStore* ruft intern die Methode *useFilter(stringToWordVector)* auf, um die Berechnung und Speicherung des Termvektors anzustoßen. Nach Ablauf der Methode, d.h. der Berechnung und dem Abspeichern des Termvektors, gibt die Klasse *WordVectorStore* die Kontrolle wieder der Klasse *LabelAndFeatureInformationManager* zurück. Der Aufruf der Methode *updateFeatureInformation(id, userStory)* ist nun abgeschlossen.

Das Sequenzdiagramm in Abbildung 5.2 zeigt nun, dass der Client die Methode *rebuildClassifierAfterUpdate(id, userStory)* aufruft. Der Client möchte also die übergebene User Story dem Trainingsdatensatz in der Klasse *ArtificialNeuralNetwork* hinzufügen und das zugrunde liegende KNN neu trainieren. Die Klasse *LabelAndFeatureInformationManager* ruft somit die Methode *updateWith(id, userStory, label)* der Klasse *ArtificialNeuralNetwork* auf. Diese fügt zunächst die übergebene User Story und das vergebene Feature-Label dem Trainingsdatensatz hinzu und stößt durch das Aufrufen der internen Methode *buildClassifier()* das erneute Training des KNN an. Wiederum geht die Kontrolle nach Ablauf der Methode an die Klasse *LabelAndFeatureInformationManager* zurück. Die durch den Client aufgerufene Methode *rebuildClassifierAfterUpdate(id, userStory)* ist nun ebenfalls beendet.

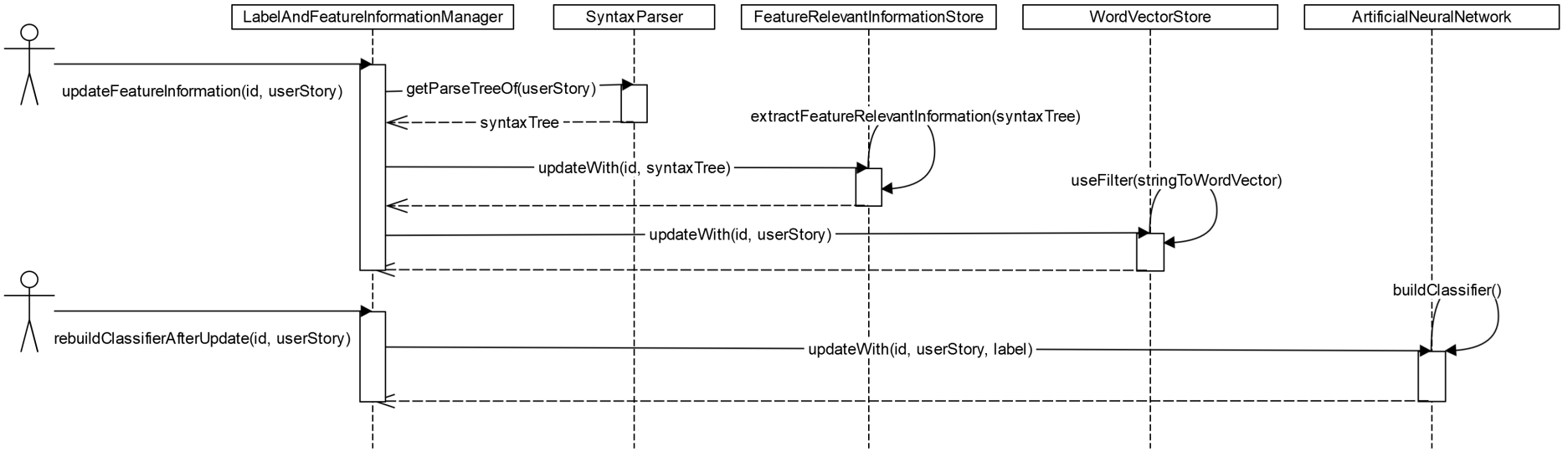

Abbildung 5.2.: Sequenzdiagramm für die Eingabe einer unbekannten User Story in das Modell.

Interaktion während der Ausgabe

Das Sequenzdiagramm in Abbildung 5.3 zeigt, dass ein Client zunächst die Methode *getFeatureRelevantInformation(userStoryId)* der Klasse *LabelAndFeatureInformationManager* aufruft, um die feature-relevanten Informationen für die übergebene User Story zu erhalten. Die Klasse *LabelAndFeatureInformationManager* gibt den Aufruf wiederum an die Klasse *FeatureRelevantInformationStore* weiter. Diese holt die für die übergebene User Story gespeicherte feature-relevante Information aus dem zugrunde liegenden Container und gibt sie an die Klasse *LabelAndFeatureInformationManager* zurück. Der Ablauf der Methode *getFeatureRelevantInformation(userStoryId)* ist damit beendet.

Das Sequenzdiagramm in Abbildung 5.3 zeigt nun, dass ein Client die Methode *getFeatureRelevantInformationOfSimilarUserStories(userStoryId)* aufruft, um auch die feature-relevanten Informationen der zu der übergebenen User Story semantisch ähnlichen User Stories zu erhalten. Die Klasse *LabelAndFeatureInformationManager* ruft dazu zunächst die Methode *getMostSimilarUserStories(userStoryId)* der Klasse *WordVectorStore* auf, um die Top-N der semantisch ähnlichsten User Stories für die übergebene User Story zu identifizieren. Die Klasse *WordVectorStore* ruft intern für alle im Container gespeicherten Termvektoren von User Stories die Methode *getCosineSimilarity(userStory, other)* auf, um die Kosinus-Ähnlichkeit des Termvektors der übergebenen User Story und der gespeicherten Termvektoren zu bestimmen. Nach Ablauf aller Methodenaufrufe gibt die Klasse *WordVectorStore* die Kontrolle wieder an die Klasse *LabelAndFeatureInformationManager* zurück. Für jede der N semantisch ähnlichsten User Stories ruft diese nun die Methode *getFeatureRelevantInformation(userStoryId)* der Klasse *FeatureRelevantInformationStore* auf, um die feature-relevanten Informationen der semantisch ähnlichen User Stories zu erhalten. Nach jedem Aufruf geht die Kontrolle an die Klasse *LabelAndFeatureInformationManager* zurück. Wurden alle N Aufrufe getätigt endet der Ablauf der Methode *getFeatureRelevantInformationOfSimilarUserStories(userStoryId)*.

Das Sequenzdiagramm in Abbildung 5.3 zeigt weiterhin, dass der Client die Methode *getRecommendedLabels(userStory)* der Klasse *LabelAndFeatureInformationManager* aufruft, um die vorgeschlagenen Feature-Labels für die übergebende User Story zu erhalten. Die Klasse *LabelAndFeatureInformationManager* ruft dazu die Methode *recommendLabels(userStory)* der Klasse *ArtificialNeuralNetwork* auf, um die Vorschläge für Feature-Labels zu beziehen. Diese klassifiziert die übergebene User Story durch das zugrunde liegende KNN und gibt die Feature-Labels der Klassifikation an die Klasse *LabelAndFeatureInformationManager* zurück. Damit endet der Ablauf der Methode *getRecommendedLabels(userStory)*.

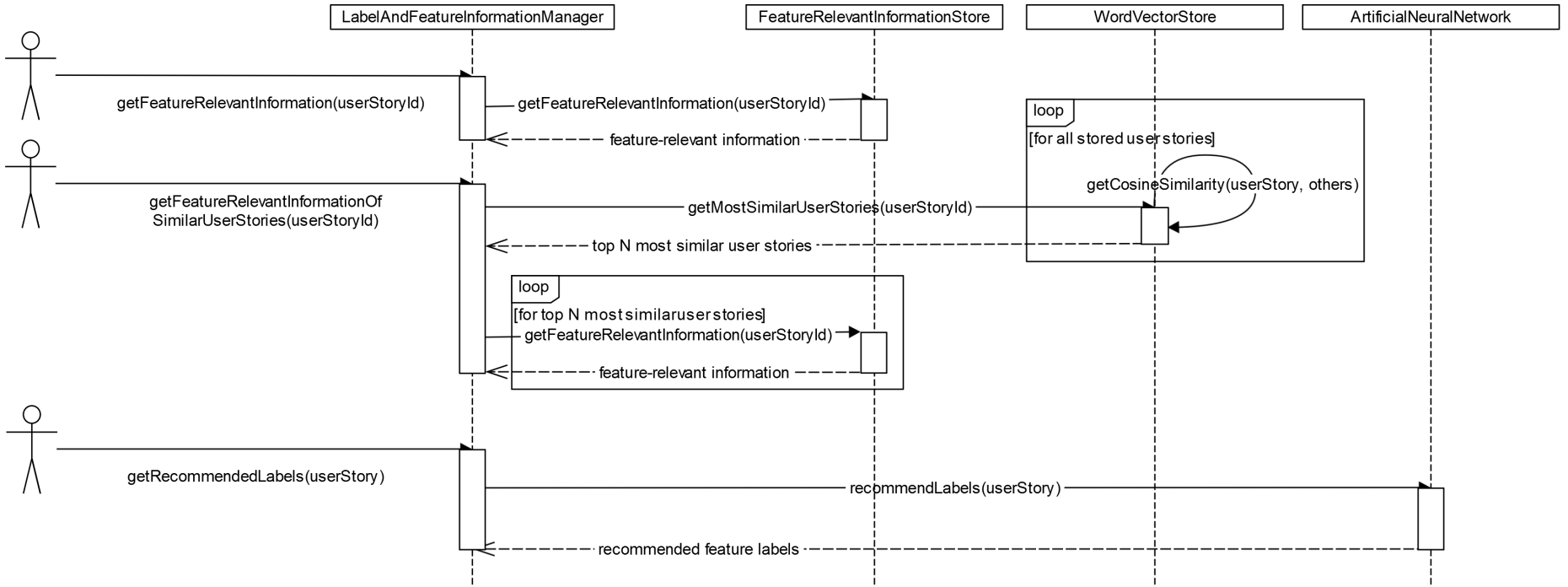

Abbildung 5.3.: Sequenzdiagramm für die Ausgabe der feature-relevanten Informationen und vorgeschlagenen Feature-Labels einer User Story durch das Modell.

5.3.3. Implementierung des Plugins

Weiterhin wurde das Modell im Rahmen eines Jira-Plugins implementiert. Das Plugin ist ebenfalls in der Programmiersprache Java geschrieben. Die Implementierung des Plugins umfasst die Klassen des Pakets *de.schindler.ma.freshman.plugin*. Die Struktur und Schnittstellen dieser Klassen orientiert sich stark an der vorgegebenen Architektur für Jira-Plugins. Die Klassen implementieren außerdem keine Funktionalität des in dieser Arbeit entwickelten Modells, sondern greifen auf diese durch die Klasse *LabelAndFeatureInformationManager* aus dem eigentlichen Modell zu. Die statische Struktur der Klassen der Implementierung des Plugins ist in dem Klassendiagramm in Abbildung 5.4 dargestellt. Im Folgenden werden die einzelnen Klassen erläutert.

LabelAndFeatureInformationManagerStore

Die Klasse *LabelAndFeatureInformationeManagerStore* verwaltet verschiedene Instanzen der Klasse *LabelAndFeatureInformationManager* aus dem Modell. Wie bereits in Abschnitt 5.3.1 beschrieben, stellt die Klasse *LabelAndFeatureInformationManager* die zentrale Schnittstelle für Clients dar, um auf die Funktionalität des Modells zuzugreifen. Die Klasse verwaltet dazu jeweils eigene Instanzen der Klassen *FeatureRelevantInformationStore*, *WordVectorStore* und *ArtificialNeuralNetwork*, in denen die aktuellen feature-relevanten Informationen, Termvektoren und der aktuelle Trainingszustand des KNN vorgehalten werden. Daher ist pro Datensatz von User Stories ein eigener *LabelAndFeatureInformationManager* zu initialisieren, um eine Vermischung der vorgehaltenen Information über verschiedene Datensätze hinweg zu vermeiden. Aus diesem Grund ist die Klasse *LabelAndFeatureInformationManagerStore* verantwortlich für die Verwaltung der verschiedenen *LabelAndFeatureInformationManager* für die jeweiligen Projekte innerhalb einer Jira-Instanz. Die Klasse erlaubt durch die Methoden *put(projectId, manager)*, *get(projectId)* und *remove(projectId)* jeweils das Hinzufügen, Zurückgeben und Löschen einer Instanz der Klasse *LabelAndFeatureInformationManager* für das im Argument übergebene Projekt. Die Klasse ist weiterhin mithilfe des Singleton-Musters implementiert, um eine zentrale Stelle für die Verwaltung der Instanzen der Klasse *LabelAndFeatureInformationManager* zu schaffen. Durch Verwendung des Singleton-Musters ist sichergestellt, dass es nur eine Instanz der Klasse *LabelAndFeatureInformationManagerStore* innerhalb einer Jira-Instanz gibt. Auf diese Instanz kann mithilfe der Methode *getInstance()* zugegriffen werden.

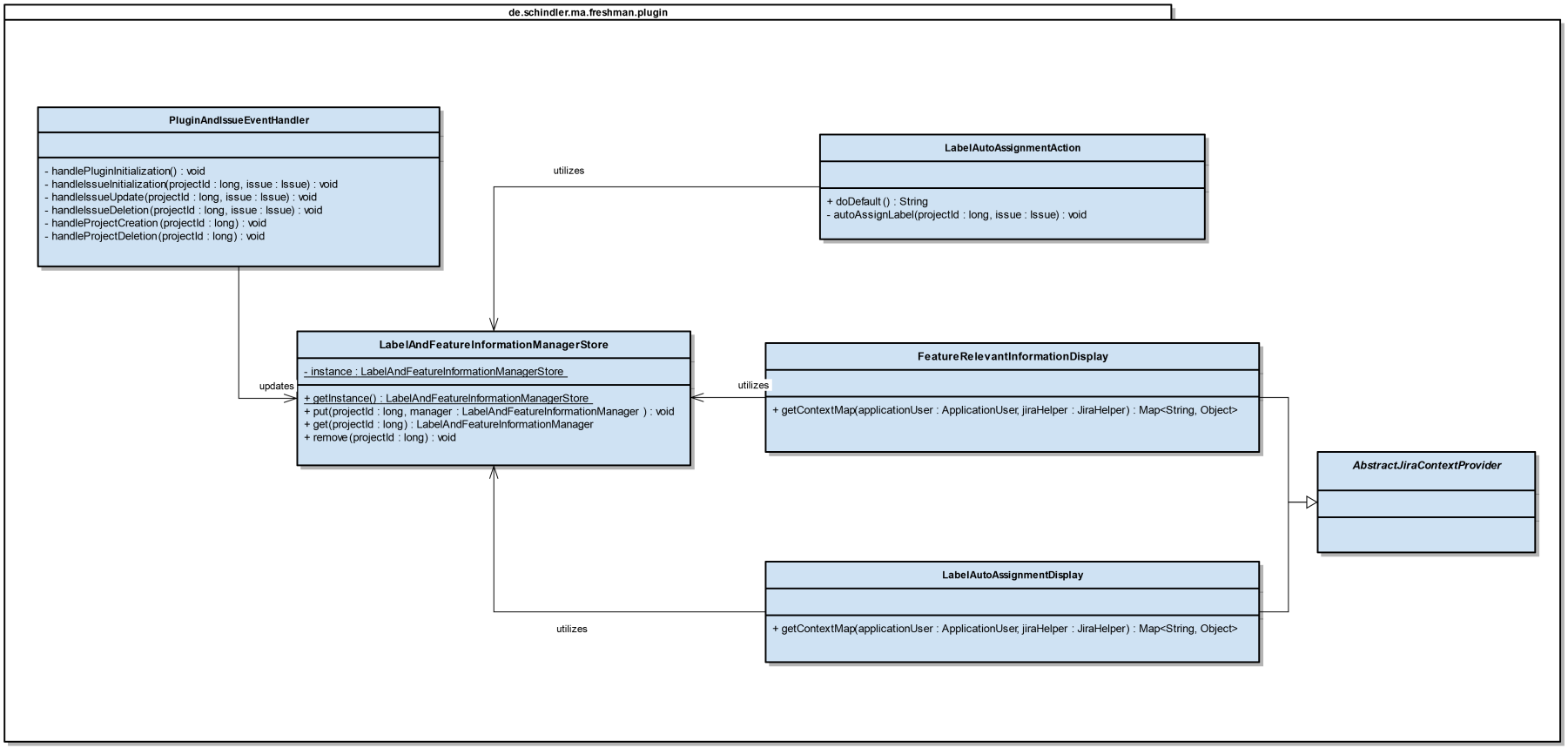

Abbildung 5.4.: Klassendiagramm der Implementierung des Plugins.

PluginAndIssueEventHandler

Die Klasse *PluginAndIssueEventHandler* stellt die Schnittstelle zwischen den Nutzereingaben auf der Benutzeroberfläche des Plugins und der entsprechenden Funktionalität des Modells dar. Die Klasse *PluginAndIssueEventHandler* fängt dazu relevante Eingabe-Events ab und leitet diese an den für das jeweilige Projekt zuständigen *LabelAndFeatureInformationManager* weiter, wo die Eingabe-Events dann verarbeitet werden. Relevante Eingabe-Events auf der Benutzeroberfläche des Plugins sind z.B. 1) das Erstellen einer neuen User Story, 2) das Aktualisieren einer User Story und 3) das Löschen einer User Story durch den Nutzer. Diese Events werden durch die Methoden *handleIssueInitialization(projectId, issue)*, *handleIssueUpdate(projectId, issue)* und *handleIssueDeletion(projectId, issue)* der Klasse *PluginAndIssueEventHandler* an den zuständigen *LabelAndFeatureInformationManager* weitergeleitet. Andere relevante Eingabe-Events betreffen die eigentlichen Projekte, in denen die User Stories verwaltet werden. So wird durch die Methoden *handleProjectCreation(projectId)* und *handleProjectDeletion(projectId)* das Modell über die Erstellung bzw. Löschung eines Projekts benachrichtigt. Zuletzt sind Ereignisse relevant, die den Lebenszyklus des Plugins betreffen: So meldet die Methode *handlePluginInitialization()* dem Modell, dass das Jira-Plugin durch den Nutzer aktiviert wurde.

LabelAutoAssignmentDisplay und FeatureRelevantInformationDisplay

Diese Klassen sind für die Darstellung der Balkendiagramme für die feature-relevanten Informationen bzw. vorgeschlagenen Feature-Labels auf der Benutzeroberfläche des Plugins verantwortlich. Der Aufbau und die Methoden dieser Klassen sind stark durch die vorgegebene Architektur für Jira-Plugins, insbesondere der Klasse *AbstractJiraContextProvider* bestimmt. Die Methoden *getContextMap(user, jiraHelper)* der jeweiligen Klassen tragen alle nötigen Informationen, die zur korrekten Darstellung der Diagramme auf der Benutzeroberfläche nötig sind, zusammen. Dies geschieht durch eine Abfrage der feature-relevanten Informationen bzw. vorgeschlagenen Feature-Labels an die Klasse *LabelAndFeatureInformationManager* aus dem Modell.

LabelAutoAssignmentAction

Diese Klasse implementiert die Vergabe des wahrscheinlichsten Feature-Labels für eine User Story. Auch der Aufbau und die Methoden dieser Klasse sind wesentlich von der vorgegebenen Architektur für Jira-Plugins bestimmt. Die Methode *autoAssignLabel(projectId, issue)* weist einer User Story ihr wahrscheinlichstes Feature-Label zu. Dieses wird durch eine Abfrage an die Klasse *LabelAndFeature-*

InformationManager aus dem Modells bestimmt. Besitzt die jeweilige User Story bereits ein Feature-Label, wird dieses jedoch nicht überschrieben. In diesem Fall hat der Aufruf der Methode *autoAssignLabel()* keine Wirkung.

5.3.4. Interaktion des Plugins mit dem Modell

Im Sequenzdiagramm der Abbildung 5.5 wird die Interaktion der Klassen des Plugins mit denen des Modells anhand der folgenden Benutzereingaben demonstriert: 1) Erstellen einer User Story, 2) Aktualisieren einer User Story und 3) Löschen einer User Story. Die Behandlung anderer Eingabe-Events läuft grundsätzlich ähnlich ab. Im Wesentlichen erfolgt die Interaktion des Plugins und des Modells durch die drei Klassen *PluginAndIssueEventHandler*, *LabelAndFeatureInformationManagerStore* und *LabelAndFeatureInformationManager*.

Zu Beginn des Sequenzdiagramm in Abbildung 5.5 wird eine neue User Story durch einen Nutzer erstellt. Es wird das zugehörige Eingabe-Event ausgelöst und die Methode *handleIssueInitialization(projectId, userStory)* der Klasse *PluginAndIssueEventHandler* aufgerufen. Diese bestimmt dann den für das im Argument übergebene Projekt zuständigen *LabelAndFeatureInformationManager*. Dies geschieht durch Aufruf der Methode *get(projectId)* der Klasse *LabelAndFeatureInformationManagerStore*. Nach Ablauf der Methode gibt dieser den tatsächlich zuständigen *LabelAndFeatureInformationManager* and den *PluginAndIssueEventHandler* zurück. Dieser kann nun die neu erstellte User Story an das Modell weiterleiten, was durch den sequentiellen Aufruf der Methoden *updateFeatureInformation(id, userStory)* und *rebuildClassifierAferUpdate(id, userStory)* der Klasse *LabelAndFeatureInformationManager* erfolgt. Nach Ablauf beider Methoden wurde das Modell so mit der im Argument der Methode *handleIssueInitialization(projectId, issue)* übergebenen User Story aktualisiert. Dem Sequenzdiagramm in Abbildung 5.5 lässt sich weiterhin entnehmen, dass die Interaktion der Klassen während der Aktualisierung einer User Story genauso abläuft, wie bei der Erstellung einer neuen User Story. Es wird jedoch in diesem Fall die Methode *handleIssueUpdate(projectId, userStory)* der Klasse *PluginAndIssueEventHandler* durch das Eingabe-Event aufgerufen.

Am Schluss des Sequenzdiagramms in Abbildung 5.5 ist erkennbar, dass ein Nutzer eine User Story löschen möchte. Das dazugehörige Eingabe-Event wird ausgelöst und die Methode *handleIssueDeletion(projectId, userStory)* der Klasse *PluginAndIssueEventHandler* aufgerufen. Diese bestimmt zunächst wieder den zu dem im Argument übergebenen Projekt gehörenden *LabelAndFeatureInformationManager*. Dies geschieht durch Aufruf der Methode *get(projectId)* der Klasse *LabelAndFeatureInformationManagerStore*. Nach Ablauf der Methode wird der jeweils zuständige *LabelAndFeatureInformationManager* an die Klasse *PluginAndIssueEventHandler* zurückgegeben.

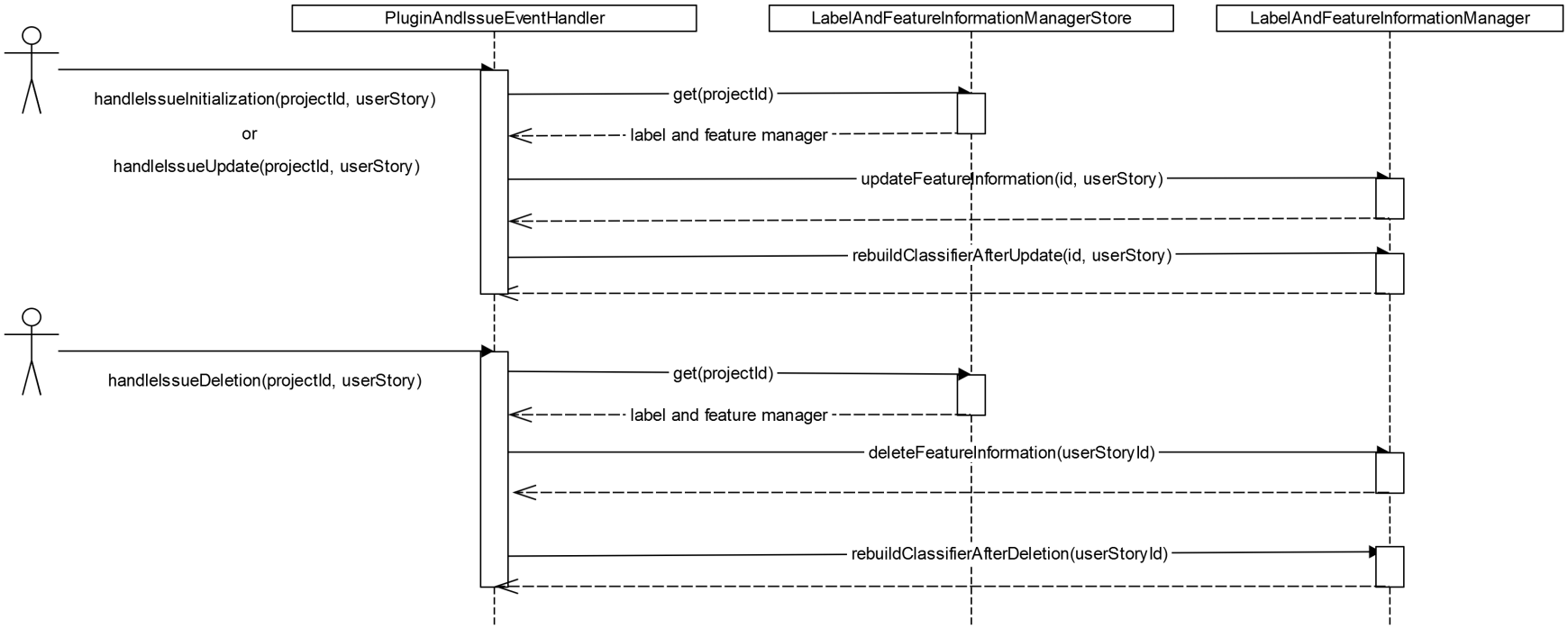

Abbildung 5.5.: Sequenzdiagramm der Interaktion zwischen dem Plugin und Modell während der Behandlung verschiedener Eingabe-Events von der Benutzeroberfläche.

Diese kann nun die im Argument der Methode *handleIssueDeletion(projectId, userStory)* übergebene User Story aus dem Modell löschen. Die Löschung wird durch den sequentiellen Aufruf der Methoden *deleteFeatureInformation(userStoryId)* und *rebuildClassifierAfterDeletion(userStoryId)* der Klasse *LabelAndFeatureInformationManager* erreicht. Nach dem Ablauf beider Methoden ist die Behandlung des Eingabe-Events bzgl. der Löschung einer User Story abgeschlossen.

5.4. Qualitätssicherung

Nach Abschluss der Implementierung des Modells als Jira-Prototyp wurden im Rahmen der Qualitätssicherung Komponenten- und Systemtests durchgeführt. Die folgenden Aspekte sind dabei von Bedeutung: Diejenigen Klassen, welche die wesentliche (Teil-)Funktionalität des Modells implementieren, rufen überwiegend Methoden von Klassen aus externen Bibliotheken auf. Dies sind im Besonderen die Klassen *SyntaxParser*, *FeatureRelevantInformationStore*, *WordVectorStore* und *ArtificialNeuralNetwork*. Aus diesem Grund wurden keine Komponententests für diese Klassen durchgeführt, da diese hauptsächlich die Klassen und Methoden der externen Bibliotheken getestet hätten. Deshalb wurden insbesondere auch keine konkreten Ausgabewerte wie 1) die Vorhersagewahrscheinlichkeiten des KNN, 2) die tf-idf-Gewichte der Termvektoren und 3) die Struktur der geparsten Syntaxbäume getestet, da diese Funktionalität in den Klassen der externen Bibliotheken implementiert ist. Wie bereits in Abschnitt 5.3 erwähnt, kapselt die Klasse *LabelAndFeatureInformationManager* den Zugriff auf die oben genannten Klassen und stellt die zentrale Schnittstelle für einen Client dar. Daher wurden Komponententest hauptsächlich für diese Klasse durchgeführt. Innerhalb des Pakets *de.schindler.ma.freshman.model* konnte so eine Anweisungsüberdeckung von 97% erreicht werden. Die übrigen 3% ergeben sich durch nicht genommene Verzweigungen innerhalb von bedingten Anweisungen. Es wurde eine Anweisungsüberdeckung von mindestens 90% innerhalb dieses Pakets angestrebt, da es die wesentliche Funktionalität des in dieser Arbeit entwickelten Modells bereitstellt.

Eine Ausnahme der obigen Entscheidung keine Komponententest für andere Klassen als *LabelAndFeatureInformationManager* durchzuführen, sind die Methoden *getMostSimilarUserStories()* und *getCosineSimilarity()* der Klasse *WordVectorStore*. Diese wurden ohne die Verwendung externer Bibliotheken implementiert und sind somit insbesondere anfällig für Fehler. Dies gilt weiterhin für die Methode *extractFeatureRelevantInformation()* der Klasse *FeatureRelevantInformationStore*. Diese wendet die im Rahmen dieser Arbeit definierten Extraktionsregeln auf den Syntaxbäumen der User Stories an. Um die Korrektheit der extrahierten featurerelevanten Information sicherzustellen, wurden für diese Methode anhand einer Stichprobe von User Stories aus den verwendeten Datensätzen Komponententests durchgeführt.

Für die Klassen aus dem Paket *de.schindler.ma.freshman.plugin* wurden ebenfalls keine Komponententests durchgeführt, da diese Klassen im Wesentlichen plugin-bezogene Funktionalität wie die Darstellung von Ausgabegrößen auf der Benutzeroberfläche oder die Weiterleitung von Eingabe-Events an das eigentliche Modell implementieren. Diese Klassen werden hauptsächlich durch die durchgeführten Systemtests abgedeckt. Der Fokus der Systemtests lag bzgl. der funktionalen Anforderungen auf der korrekten Anzeige von feature-relevanten Informationen und vorgeschlagenen Feature-Labels. In Bezug zu den Qualitätsanforderungen wurden insbesondere Aspekte der Aufgabenangemessenheit getestet: So z.B. die automatische Aktualisierung von Modellkomponenten im Hintergrund, wenn User Stories durch den Nutzer verändert werden oder die Behandlung von fehlerhaften Eingaben wie einer fehlenden User-Story-Beschreibung. Insgesamt wurden 28 Komponenten- und 13 Systemtests spezifiziert und positiv durchlaufen.

6. Evaluation

Im folgenden Kapitel wird das in dieser Arbeit entwickelte Modell evaluiert. Die dazu genutzten Metriken und das Vorgehen bei der Evaluation werden in Abschnitt 6.1 beschrieben. In Abschnitt 6.2 werden die Evaluationsergebnisse vorgestellt und diskutiert. Danach werden in Abschnitt 6.3 die Evaluationsergebnisse der Ansätze aus den verwandten Arbeiten vorgestellt und mit denen des in dieser Arbeit entwickelten Modells verglichen.

6.1. Vorgehen bei der Evaluation

Das Modell dieser Arbeit wird sowohl bzgl. der Qualität der extrahierten feature-relevanten Informationen als auch der vorgeschlagenen Feature-Labels evaluiert. Für die Evaluation wurden zwei Testdatensätze aus Studenten-Projekten im Bereich Software-Entwicklung genutzt. Die Projekte stammen aus den Domänen Gesundheitswesen bzw. einer Navigationslösung für Studenten. Tabelle 6.1 gibt einen Überblick der Testdatensätze. Das Projekt aus dem Gesundheitswesen (im Folgenden Projekt 1 genannt) besteht aus 59 User Stories, während das Projekt für die Navigationslösung (im Folgenden Projekt 2 genannt) 19 User Stories umfasst. Die User Stories aus Projekt 1 beschreiben insgesamt 7 verschiedene Features, während die User Stories aus Projekt 2 insgesamt 5 verschiedene Features beschreiben. Alle User Stories bestehen aus einem Titel (Zusammenfassung), dem eigentlichen Text (Beschreibung) und einem Feature-Label. Die vergebenen Feature-Labels werden dabei als Goldstandard für die Evaluation behandelt. Die User Stories der Testdatensätze umfassen keine Angaben bzgl. feature-relevanter Informationen. Daher wurde für die Evaluation der extrahierten feature-relevanten Informationen der Goldstandard durch den Autor definiert.

	Testdatensätze	
	Projekt 1	Projekt 2
Domäne	Gesundheitswesen	Navigationslösung
# User Stories	59	19
# Features	7	5

Tabelle 6.1.: Überblick der für die Evaluation genutzen Testdatensätze.

Die Evaluation erfolgte anhand der Metriken Precision P, Recall R und F1-Maß. Diese wurden im Kontext dieser Arbeit wie in Definition 6.1 gezeigt verwendet. Für eine User Story u_i beschreibt X_i dabei die durch das Modell zurückgegebene Ergebnismenge. Im Falle der feature-relevanten Information ist X_i also die Menge der aus User Story u_i extrahierten feature-relevanten Informationen. Im Falle der Label-Vorschläge ist X_i somit die Menge der vorgeschlagenen Feature-Labels für User Story u_i. Weiterhin beschreibt G_i die Ergebnismenge des Goldstandards für eine User Story u_i. Im Falle der feature-relevanten Information ist G_i also die Menge der tatsächlich feature-relevanten Informationen in User Story u_i. Im Falle der Label-Vorschläge ist G_i somit die Menge der tatsächlich vergebenen Feature-Labels für User Story u_i.

$$P = \frac{\sum_{u_i} |X_i \cap G_i|}{\sum_{u_i} |X_i|} \quad R = \frac{\sum_{u_i} |X_i|}{\sum_{u_i} |G_i|} \quad \text{F1-Maß} = 2\frac{P \cdot R}{P + R} \tag{6.1}$$

6.2. Vorstellung der Evaluationsergebnisse

Im folgenden Abschnitt werden die Evaluationsergebnisse des in dieser Arbeit entwickelten Modells vorgestellt und diskutiert. Die Evaluation wird jeweils separat für die extrahierten feature-relevanten Informationen und die vorgeschlagenen Feature-Labels durchgeführt.

6.2.1. Evaluation der feature-relevanten Informationen

Die Evaluation der extrahierten feature-relevanten Informationen erfolgte anhand aller User Stories der Testdatensätze, jedoch separat für die jeweiligen Projekte. Für jede User Story eines Projekts wurden durch das Modell die jeweiligen feature-relevanten Informationen extrahiert und manuell mit denen des definierten Goldstandards verglichen. Die Ergebnisse der Evaluation der extrahierten feature-relevanten Informationen sind in Tabelle 6.2 dargestellt. Die Evaluation zeigt, dass das Modell gute Ergebnisse bzgl. der extrahierten feature-relevanten Information liefert. So konnte eine Precision von 83% und ein Recall von 91% in Projekt 1 erreicht werden. Die Werte für die Precision und den Recall in Projekt 2 betragen 70% bzw. 75%. Daraus ergeben sich als F1-Maße 87% für Projekt 1 bzw. 72% für Projekt 2. An den Werten für den Recall ist zu erkennen, dass das Modell einen Großteil der feature-relevanten Informationen aus den User Stories in beiden Projekten extrahiert. In Projekt 1 konnten mit 91% sogar fast alle feature-relevanten Informationen extrahiert werden. Weiterhin zeigt die

Evaluation, dass die extrahierten feature-relevanten Informationen tatsächlich relevant für die Definition eines Feature-Labels sind. In Projekt 1 sind 83% der extrahierten Informationen tatsächlich als feature-relevant einzuschätzen, während dies in Projekt für 70% aller extrahierten Informationen der Fall ist. Auffällig ist, dass die Werte für alle Evaluationsmetriken in Projekt 1 höher sind als in Projekt 2. Dies ist wahrscheinlich darin begründet, dass hauptsächlich die User Stories aus Projekt 1 während der Herleitung der Extraktionsregeln analysiert wurden. Es ist anzunehmen, dass die Qualität der extrahierten feature-relevanten Information erheblich von bestimmten Parametern der zugrundeliegenden User Stories abhängt. Diese Parameter können z.B. die Domäne des jeweiligen Projektes oder die jeweils eingesetzte Satzschablone zur Formulierung der User Stories sein.

	Feature-relevante Information		
	Precision	Recall	F1-Maß
Projekt 1	0,83	0,91	0,87
Projekt 2	0,70	0,75	0,72

Tabelle 6.2.: Evaluationsergebnisse bzgl. der feature-relevanten Information.

6.2.2. Evaluation der Label-Vorschläge

Für die Evaluation der Label-Vorschläge wurden die Testdatensätze je Projekt im Verhältnis 80 zu 20 (80/20-Split) in einen Trainingsdatensatz und einen Testdatensatz aufgeteilt. Das KNN wurde dann jeweils für ein Projekt anhand des Trainingsdatensatzes trainiert und anhand des Testdatensatzes evaluiert. Die Aufteilung erfolgte dabei komplett zufällig, d.h. es wurde nicht auf eine gleichmäßige Verteilung der Goldstandard-Klassen innerhalb des Trainingsdatensatzes bzw. Testdatensatzes geachtet. Dem liegt die Annahme zugrunde, dass auch in der Praxis bestimmte Klassen (Feature-Labels) innerhalb der User Stories über- bzw. unterrepräsentiert sein können. Wird z.B. innerhalb eines Projektes in einem abgegrenzten Zeitraum verstärkt an einem bestimmten Feature gearbeitet, kann es sein, dass verfasste User Stories überproportional dieses Feature beschreiben. Insgesamt wurde die Evaluation des KNN je Projekt dreimal wiederholt, d.h. es wurde dreimal ein komplett zufälliger 80/20-Split des jeweiligen Testdatensatzes durchgeführt anhand dessen das KNN neu trainiert bzw. evaluiert wurde. Der Durchschnittswert aller Wiederholungen wurde für jede Evaluationsmetrik berechnet, um eine präzise Aussage bzgl. der Qualität der vorgeschlagenen Feature-Labels zu ermöglichen.

Projekt 1	Label-Vorschläge		
	Precision	Recall	F1-Maß
1. Wiederholung	0,83	1,00	0,91
2. Wiederholung	0,75	1,00	0,86
3. Wiederholung	0,75	1,00	0,86
⌀ Durchschnitt	0,78	1,00	0,87
Projekt 2	Label-Vorschläge		
	Precision	Recall	F1-Maß
1. Wiederholung	0,75	1,00	0,86
2. Wiederholung	0,75	1,00	0,86
3. Wiederholung	0,25	1,00	0,4
⌀ Durchschnitt	0,58	1,00	0,71

Tabelle 6.3.: Evaluationsergebnisse bzgl. der vorgeschlagenen Feature-Labels.

Die Ergebnisse der Evaluation der Label-Vorschläge sind in Tabelle 6.3 dargestellt. Anhand der Evaluation kann festgestellt werden, dass das Modell durchschnittliche bis gute Ergebnisse bzgl. der vorgeschlagenen Feature-Labels für User Stories erzielt. Das tatsächliche Ergebnis ist dabei jedoch stark vom jeweiligen Projekt abhängig. So konnte in Projekt 1 eine durchschnittliche Precision von 78% bei einem perfekten Recall von 100% erzielt werden. Der Durchschnittswert des F1-Maßes beträgt 87%. In Projekt 2 beträgt die durchschnittliche Precision dagegen 58% bei einem perfekten Recall von ebenfalls 100%. Der durchschnittliche Wert für das F1-Maß beträgt 71%. Es ist auffällig, dass der Recall in den einzelnen Wiederholungen und in beiden Projekten immer 100% beträgt. Die Werte ergeben sich daher, dass es für eine User Story immer nur ein einziges Feature-Label im Rahmen des Goldstandards gibt. Das KNN des Modells schlägt, wie in Abschnitt 4.4 beschrieben, ebenfalls nur ein Feature-Label pro User Story vor. Aufgrund der Definition 6.1 für den Recall, beträgt dessen Wert im Rahmen der genutzten Testdatensätze immer 100%.

Die Evaluation zeigt weiterhin, dass in Projekt 1 die vom Modell vorgeschlagenen Feature-Labels deutlich häufiger dem tatsächlichen Feature-Label einer User Story entsprechen als in Projekt 2. Dies kann anhand der Datensätze aus den jeweiligen Projekten erklärt werden, mit denen das KNN trainiert wurde. Wie in Tabelle 6.1 dargestellt, besteht der Datensatz von Projekt 1 aus 59 User Stories, während der Datensatz von Projekt 2 aus 19 User Stories besteht. Dem KNN steht in Projekt 1 somit ein deutlich umfangreicherer Datensatz für das Training zu Verfügung als dies in Projekt 2 der Fall ist. Aufgrund des größeren Trainingsdatensatzes ist zu erwarten, dass das KNN in Projekt 1 eine bessere Klassifikationsleistung erzielt als in Projekt 2. Dies wird durch die Evaluationsergebnisse bestätigt. So ist in Projekt 1 die durchschnittliche Precision mit 78% deutlich höher als in Projekt 2, wo diese 58% beträgt. Auch die Precision-Werte der einzelnen Wiederholungen schwanken

in Projekt 2 deutlicher als in Projekt 1. Während in Projekt 1 die Precision in den einzelnen Wiederholungen 75% bzw. 83% beträgt, hat Wiederholung 3 mit einer Precision von 52% in Projekt 2 eine erheblich schlechtere Precision als die beiden anderen Wiederholungen, in denen die Precision jeweils 75% beträgt. Im Allgemeinen ist zu erwarten, dass die Vorhersagegenauigkeit des KNN mit umfangreicheren Datensätzen als denen aus den Projekten 1 und 2 steigt.

6.3. Vergleich mit Evaluationsergebnissen verwandter Arbeiten

Im folgenden Abschnitt sollen die Evaluationsergebnisse des entwickelten Modells mit den Ergebnissen der Ansätze aus den verwandten Arbeiten verglichen werden. Tabelle 6.4 zeigt die erzielten Werte für die Evaluationsmetriken Precision, Recall und F1-Maß für die Ansätze aus den verwandten Arbeiten. Die Werten sind dabei direkt aus den jeweiligen Arbeiten übernommen.

Es ist anzumerken, dass nicht alle Ansätze sowohl feature-relevante Informationen extrahieren als auch Feature-Labels vorschlagen. In diesen Fällen sind nur die Werte für das jeweils in der Arbeit behandelte Problem angegeben. Die Werte für das jeweils nicht behandelte Problem sind mit einem X markiert. Beinhaltete eine Arbeit nur Werte für Precision bzw. Recall wurde das F1-Maß nach Definition 6.1 berechnet. In diesen Fällen ist der Wert für das F1-Maß fett gedruckt. Wurden Werte aus Diagrammen in den jeweiligen Arbeiten übernommen und konnten daher nicht genau bestimmt werden, sind diese Werte kursiv gedruckt. Werte, die mit einem - markiert sind, konnten aus verschiedenen Gründen nicht aufgeführt werden: Es wurden z.B. keine Angaben zu den jeweiligen Werten gemacht wie in den Arbeiten von [10], [32] und [21]. Bei der Arbeit von [3] handelt es sich um eine Übersichtsstudie, in der ebenfalls keine Angaben zu den jeweiligen Werten der referenzierten Arbeiten gemacht wurden. Ein weiterer Grund ist, dass eine andere Evaluationsmetrik als Precision, Recall und F1-Maß verwendet wurde wie das Kappa-Maß in der Arbeit von [25]. Zuletzt wurden in manchen Arbeiten andere Ausgabeartefakte evaluiert wie z.B. konzeptuelle Modelle und Cluster in den Arbeiten von [17] bzw. [23]. Haben Arbeiten mehrere Werte für Evaluationsmetriken aufgeführt z.B. separat für verschiedene Modellvarianten und -parameter oder Datensätze, wurde der jeweils beste Wert berücksichtigt. Wurde in den Arbeiten ein Durchschnittswert für die Evaluationsmetriken aufgeführt wurde dieser anstatt der jeweils besten Werte berücksichtigt.

Arbeit	Feature-relevante Information			Label-Vorschläge		
	Precision	Recall	F1-Maß	Precision	Recall	F1-Maß
[2]	*0.490*	*0.900*	**0.635**	X	X	X
[10]	X	X	X	-	-	0.480
[13]	0.559	0.434	0.458	X	X	X
[27]	0.941	0.969	0.955	X	X	X
[3]	-	-	-	X	X	X
[16]	0.862	0.835	0.845	X	X	X
[31]	X	X	X	0.570	0.470	**0.515**
[11]	X	X	X	0.564	0.668	**0.612**
[30]	X	X	X	0.353	0.956	**0.516**
[32]	X	X	X	-	0.763	-
[21]	-	-	-	X	X	X
[17]	-	-	-	X	X	X
[25]	X	X	X	-	-	-
[23]	X	X	X	-	-	-

Tabelle 6.4.: Evaluationsergebnisse der Ansätze aus verwandten Arbeiten.

Das in dieser Arbeit entwickelte Modell zeigt sich im Vergleich zu den Ansätzen aus den verwandten Arbeiten kompetitiv. Sowohl bei der Extraktion feature-relevanter Informationen als auch beim Vorschlagen von Feature-Labels sind die Ergebnisse mit denen aus den verwandten Arbeiten vergleichbar und teilweise sogar besser. Betrachtet man die Evaluation der feature-relevanten Informationen zeigt nur der Ansatz von Quirchmayr et al. [27] deutlich besere Ergebnisse als das in dieser Arbeit entwickelte Modell. So beträgt der Wert des F1-Maßes bei Quirchmayr et al. [27] 0.955, während er in dieser Arbeit bei 0.87 für den ersten Testdatensatz bzw. 0.72 für den zweiten Testdatensatz liegt. Auch die Werte für Precision und Recall sind mit 0.941 und 0.969 besser als die Werte in dieser Arbeit, welche bei 0.83 und 0.91 für den ersten Testdatensatz bzw. 0.7 und 0.75 für den zweiten Testdatensatz liegen. Im Vergleich zu den Ansätzen aus der verwandten Literatur zeigt der Ansatz von Quirchmayr et al. [27] insgesamt die besten Ergebnisse. Der Ansatz von Liu et al. [16] zeigt in Bezug zu dem in dieser Arbeit entwickelten Modell vergleichbare Ergebnisse. So betragen die Werte für Precision, Recall und F1-Maß in [16] jeweils 0.862, 0.835 und 0.845, was vergleichbar zu den Werten dieser Arbeit für den ersten Testdatensatz ist. Die Ansätze aus den Arbeiten von Arora et al. [2] und Johann et al. [13] zeigen insgesamt etwas schlechtere Ergebnisse als das in dieser Arbeit entwickelte Modell. Die Werte für Precision, Recall und F1-Maß betragen 0.49, 0.90 und 0.635 bei Arora et al. [2], während sie bei Johann et al. [13] bei 0.559, 0.434 und 0.458 liegen. Dies sind schlechtere Werte als die des zweiten Testdatensatzes dieser Arbeit. Hervorzuheben bei Johann et al. [13] ist der im Vergleich zur Precision besonders hohe Wert für den Recall von 0.9, was auch im Vergleich zu den anderen Arbeiten ein guter Wert ist.

Insgesamt zeigt sich unter den verwandten Arbeiten eine deutliche Variabilität bzgl. der Qualität der extrahierten feature-relevanten Informationen. Dies kann zum Einen an den jeweiligen zur Extraktion genutzten Heuristiken (einfache POS-Muster bzw. komplexe POS-Muster in Syntaxbäumen) selbst liegen. Zum Anderen sind die jeweiligen Artefakte, auf denen die Heuristiken arbeiten, sehr unterschiedlich (Benutzerhandbücher, App-Beschreibungen und natürlichsrachliche Anforderungen), was ebenfalls einen großen Einflus auf die Qualität der Extraktionsergebnisse hat. Die besonders guten Ergebnisse bei Quirchmayr et al. [27] können u.A. mit den umfangreichen Transformationsregeln erklärt werden, mit denen die Korrektheit von geparsten Syntaxbäumen sichergestellt wird. Solche Regeln werden in diesem Umfang in keiner der verwandten Arbeiten und auch nicht in dem in dieser Arbeit entwickelten Modell verwendet.

Betrachtet man die Ergebnisse bzgl. der vorgeschlagenen Labels, zeigt sich, dass das in dieser Arbeit entwickelte Modell bzgl. der Evaluationsmetriken die besten Ergebnisse liefert. Der höchste Wert für die Precision wird mit 0.57 in Wang et al. [31] erreicht. Im Falle des Recalls wird der höchste Wert in Wang et al. [30] mit 0.956 erreicht. In der Arbeit von Hong et al. [11] ist der Wert für das F1-Maß mit 0.612 am höchsten. Diese Werte werden alle von denen in dieser Arbeit übertroffen. Im ersten Testdatensatz liegen die Werte für Precision im Durchschnitt bei 0,78, für den Recall bei 1,00 und für das F1-Maß bei 0,87. Im zweiten Testdatensatz liegen diese durchschnittlich bei 0,58 für Precision, 1,00 für Recall und 0,71 für das F1-Maß. Diese guten Werte sind jedoch wahrscheinlich durch die hohen Werte für den Recall bedingt. Da im Kontext dieser Arbeit immer nur ein Label pro User Story vergeben wird, beträgt der Wert für den Recall immer 1,00. Die Arbeiten von Wang et al. [31], Hong et al. [11], Wang et al. [30] und Wang et al. [32] betrachten dagegen jedoch zumindest teilweise den Fall, dass pro Artefakt mehrere Labels vergeben werden können. Dies kann ein Grund sein, warum die Werte für die verschiedenen Evaluationsmetriken in manchen Arbeiten schlechter sind als die in dieser Arbeit. In Anbetracht, dass in diesen Arbeiten mehrere Labels vergeben werden, ist auch der hohe Wert für den Recall von 0.956 in der Arbeit von Wang et al. [30] hervorzuheben.

Insgesamt zeigt sich, dass alle Arbeiten im Bereich des Vorschlagens von Labels bzgl. der Ergebnisse dicht beieinander liegen. Zum Einen kann dies an den ähnlichen Techniken (Topic Modeling bzw. Verwendung von Klassifikationsmodellen), die im Rahmen dieser Ansätze verwendet werden, erklärt werden. Zum Anderen haben gerade die Arbeiten von Wang et al. [31], Hong et al. [11], Wang et al. [30] und Wang et al. [32] einen sehr ähnlichen Fokus bzgl. der Artefakte (Fragen von Software-Informationsseiten bzw. Software-Beschreibungen) für die Labels zu vergeben sind.

7. Fazit

Dieses Kapitel zieht ein Fazit bzgl. der vorliegenden Arbeit. In Abschnitt 7.1 wird ein Ausblick auf weitere Arbeiten gegeben. Abschnitt 7.2 fasst die Methoden und Ergebnisse der Arbeit zusammen.

7.1. Ausblick

Die Evaluation hat deutlich gemacht, dass die Werte der Evaluationsergebnisse je nach verwendetem Datensatz variieren. Daher muss das in dieser Arbeit entwickelte Modell anhand einer größeren Datenbasis evaluiert werden, um dessen generelle Anwendbarkeit zu validieren. Insbesondere ist zu vermuten, dass die zur Extraktion genutzten heuristischen POS-Muster stark auf die verwendeten Datensätze ausgerichtet sind. So könnte schon die Verwendung unterschiedlicher Satzschablonen oder der Bezug zu anderen Domänen die Qualität der aus den User Stories extrahierten feature-relevanten Informationen beeinflussen. Ebenso ist anzunehmen, dass die Qualität der vorgeschlagenen Feature-Labels deutlich von dem genutzten Klassifikationsmodell abhängig ist. Im Rahmen dieser Arbeit wurde bereits informell der Naive-Bayes Klassifikator als Alternative zu dem eingesetzten KNN evaluiert. Es existieren jedoch weitere Klassifikationsmodelle wie z.B. Entscheidungsbäume und verschiedene Architekturen für KNNs. Weitere formelle Evaluation muss daher in Abhängigkeit der jeweiligen Datenbasis das am besten geeignete Klassifikationsmodell bestimmen. Mögliche Verbesserungen des Modells ergeben sich außerdem bei der Wahl des semantischen Ähnlichkeitsmaßes für User Stories sowie der Herleitung tiefergehender und sprachwissenschaftlich fundierter POS-Muster. So können z.B. durch Verwendung der Levenshtein-Distanz im Rahmen der Bestimmung der semantischen Ähnlichkeit von User Stories auch lexikalische Unterschiede von Wörtern z.B. durch Tipp- oder Rechtschreibfehler berücksichtigt werden. In Bezug zu den POS-Mustern können z.B. komplexere Transformationsregeln wie sie Quirchmayr et al. [27] verwenden implementiert werden, um die Korrektheit von Syntaxbäumen sicherzustellen und mögliche Parserfehler auszuschließen.

7.2. Zusammenfassung

In dieser Arbeit wurde ein Modell zur (semi-)automatischen Vergabe von Labels in Form von Software-Features für User Stories entwickelt. Die eigentliche Vergabe der Feature-Labels geschieht dabei manuell durch einen Nutzer. Das Modell unterstützt den Nutzer durch die folgenden Aspekte 1) Extraktion und Bündelung von feature-relevanten Informationen aus User Stories und 2) Vorschlagen von passenden Feature-Labels anhand bereits vergebener Labels. Die bekannten und mit einem Feature-Label versehenen User Stories werden genutzt, um ein KNN zu trainieren. Dieses wird dann verwendet, um neue User Stories zu klassifizieren. Somit kann voll-automatisch ein Feature-Label für eine neue User Story vorgeschlagen werden. Zudem wird so die konsistente Verwendung bereits vergebener Labels begünstigt. Anhand der Kosinus-Ähnlichkeit wird für eine User Story aus den bekannten User Stories weiterhin eine Menge von semantisch ähnlichen User Stories bestimmt. Aus diesen werden mithilfe von NLP-Techniken und heuristischen POS-Mustern feature-relevante Informationen extrahiert. Durch die Bündelung feature-relevanter Informationen aus der neuen User Story und semantisch ähnlichen User Stories, wird die manuelle Vergabe von aussagekräftigen Feature-Labels unterstützt.

Eine vor der Entwicklung des Modells durchgeführte Literaturrecherche hat gezeigt, dass es in der Literatur bisher wenige Arbeiten gibt, die sich mit der (semi-)automatischen Vergabe von Labels für natürlichsprachliche Anforderungen beschäftigen. Während der Literaturrecherche hat sich herausgestellt, dass verwandte Arbeiten ihren Fokus häufig auf jeweils einen der oben genannten Aspekte legen. Sie stellen entweder einen Ansatz zur Extraktion von Software-Features bzw. feature-relevanten Informationen vor oder entwickeln Ansätze für das Vorschlagen von Labels. Die Kombination der beiden Aspekte wie sie in dieser Arbeit vorgestellt wurde, ist jedoch nur in wenigen der verwandten Arbeiten berücksichtigt worden. Für die Extraktion von Software-Features bzw. feature-relevanten Informationen verwenden die verwandten Arbeiten überwiegend NLP-Techniken und heuristische POS-Muster. Die Ansätze für das Vorschlagen von Labels basieren auf ML und statistischen Techniken wie Topic Modeling oder der Verwendung von Ähnlichkeitsmaßen.

Das in dieser Arbeit entwickelte Modell wurde anhand von zwei Testdatensätzen und den Metriken Precision, Recall und F1-Maß evaluiert. Die Evaluation hat gezeigt, dass durch das Modell eine konsistente Vergabe von aussagekräftigen Feature-Labels für User Stories erreicht werden kann. Mit maximalen Werten von 0,83 für Precision bzw. 0,91 für Recall kann das Modell sinnvoll eingesetzt werden, um feature-relevante Informationen aus User Stories zu extrahieren. Auch die Qualität der vorgeschlagenen Labels ist mit einem durchschnittlichen Wert von 0,78 für Precision bei einem perfekten Recall geeignet, um diese bei der eigentlichen Vergabe von Feature-Labels zu berücksichtigen. Der Vergleich der Evaluationsergebnisse mit denen aus den verwandten Arbeiten macht ersichtlich, dass das in dieser

Arbeit entwickelte Modell gleichwertige und oft sogar bessere Ergebnisse bzgl. der extrahierten feature-relevanten Information und vorgeschlagenen Feature-Labels erzielt. Weiterhin konnte durch eine prototypische Implementierung als Plugin für Jira die Praktikabilität des entwickelten Modells demonstriert werden.

Die Ergebnisse der vorliegenden Arbeit und die Erkenntnisse der verwandten Arbeiten aus der Literatur zeigen, dass die voll-automatische Vergabe von Feature-Labels für User Stories zurzeit nicht praktikabel ist. Ein menschlicher Nutzer kann die Semantik einer User Story besser erfassen und über diese abstrahieren. Somit wird der Nutzer ein treffenderes Feature-Label definieren als eine automatisierte Methode. Zumindest bei umfangreichen Datensätzen zeigen automatisierte Modelle aus dem Bereich des ML jedoch eine beachtliche Leistung beim Vorschlagen von passenden Feature-Labels. Jedoch benötigen auch diese zuvor einen manuell mit Labels versehenen Datensatz für das Training. Diese Arbeit hat gezeigt, dass eine konsistente Vergabe von aussagekräftigen Feature-Labels für User Stories mithilfe eines solchen (semi-)automatischen Modells effektiv möglich ist.

Literaturverzeichnis

1. Ameller, D., Farré, C., Franch, X., und Rufian, G.: A Survey on Software Release Planning Models. In: Abrahamsson, P., Jedlitschka, A., Nguyen Duc, A., Felderer, M., Amasaki, S., und Mikkonen, T. (Hrsg.) Product-Focused Software Process Improvement, S. 48–65. Springer International Publishing, Cham (2016)
2. Arora, C., Sabetzadeh, M., Briand, L., und Zimmer, F.: Improving Requirements Glossary Construction via Clustering: Approach and Industrial Case Studies. In: Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ESEM '14, 18:1–18:10. ACM, Torino, Italy (2014)
3. Bakar, N.H., Kasirun, Z.M., und Salleh, N.: Feature extraction approaches from natural language requirements for reuse in software product lines: A systematic literature review. Journal of Systems and Software 106, 132–149 (2015)
4. Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag, Berlin, Heidelberg (2006)
5. Bosch, J.: Design and Use of Software Architectures: Adopting and Evolving a Product-line Approach. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA (2000)
6. Chen, K., Zhang, W., Zhao, H., und Mei, H.: An approach to constructing feature models based on requirements clustering. In: 13th IEEE International Conference on Requirements Engineering (RE'05), S. 31–40 (2005)
7. Classen, A., Heymans, P., und Schobbens, P.-Y.: What's in a Feature: A Requirements Engineering Perspective. In: Fiadeiro, J.L., und Inverardi, P. (Hrsg.) Fundamental Approaches to Software Engineering, S. 16–30. Springer Berlin Heidelberg (2008)
8. Cohn, M.: User Stories Applied: For Agile Software Development. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA (2004)
9. Hall, M., Eibe, F., Holmes, G., Pfahringer, B., Reutemann, P., und Witten, I.H.: The WEKA Data Mining Software: An Update. SIGKDD Explorations 11(1) (2009)
10. Hindle, A., Ernst, N.A., Godfrey, M.W., und Mylopoulos, J.: Automated Topic Naming to Support Cross-project Analysis of Software Maintenance Activities. In: Proceedings of the 8th Working Conference on Mining Software Repositories. MSR '11, S. 163–172. ACM, Waikiki, Honolulu, HI, USA (2011)
11. Hong, B., Kim, Y., und Lee, S.H.: An Efficient Tag Recommendation Method Using Topic Modeling Approaches. In: Proceedings of the International Conference on Research in Adaptive and Convergent Systems. RACS '17, S. 56–61. ACM, Krakow, Poland (2017)
12. IEEE Standard Glossary of Software Engineering Terminology. IEEE Std 610.12-1990 (1990)
13. Johann, T., Stanik, C., Alireza, M., Alizadeh, B., und Maalej, W.: SAFE: A Simple Approach for Feature Extraction from App Descriptions and App Reviews. In: 2017 IEEE 25th International Requirements Engineering Conference (RE), S. 21–30 (2017)
14. Jurafsky, D., und Martin, J.H.: Speech and Language Processing (2nd Edition). Prentice-Hall, Inc., Upper Saddle River, NJ, USA (2009)

15. Kang, K.C., Kim, S., Lee, J., Kim, K., Shin, E., und Huh, M.: FORM: A Feature-oriented Reuse Method with Domain-specific Reference Architectures. Ann. Softw. Eng. 5(1), 143–168 (1998)
16. Liu, Y., Liu, L., Liu, H., Wang, X., und Yang, H.: Mining domain knowledge from app descriptions. Journal of Systems and Software 133, 126–144 (2017)
17. Lucassen, G., Robeer, M., Dalpiaz, F., Werf, J.M.E.M. van der, und Brinkkemper, S.: Extracting conceptual models from user stories with Visual Narrator. Requirements Engineering 22(3), 339–358 (2017)
18. Manning, C.D., Raghavan, P., und Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, New York, NY, USA (2008)
19. Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S.J., und McClosky, D.: The Stanford CoreNLP Natural Language Processing Toolkit. In: Association for Computational Linguistics (ACL) System Demonstrations, S. 55–60 (2014)
20. Marcus, M., Kim, G., Marcinkiewicz, M.A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K., und Schasberger, B.: The Penn Treebank: Annotating Predicate Argument Structure. In: Proceedings of the Workshop on Human Language Technology. HLT '94, S. 114–119. Association for Computational Linguistics, Plainsboro, NJ (1994)
21. Ménard, P.A., und Ratté, S.: Concept extraction from business documents for software engineering projects. Automated Software Engineering 23(4), 649–686 (2016)
22. Miller, G.A.: WordNet: A Lexical Database for English. Commun. ACM 38(11), 39–41 (1995)
23. Misra, J., Sengupta, S., und Podder, S.: Topic Cohesion Preserving Requirements Clustering. In: 2016 IEEE/ACM 5th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), S. 22–28 (2016)
24. Ngaopitakkul, A., und Bunjongjit, S.: Selection of Proper Activation Functions in Back-Propagation Neural Network Algorithm for Transformer and Transmission System Protection. In: Transactions on Engineering Technologies: International MultiConference of Engineers and Computer Scientists 2014. Hrsg. von G.-C. Yang, S.-I. Ao, X. Huang und O. Castillo, S. 279–291. Springer Netherlands, Dordrecht(2015)
25. Niu, N., Reddivari, S., Mahmoud, A., Bhowmik, T., und Xu, S.: Automatic labeling of software requirements clusters. In: 2012 4th International Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation (SUITE), S. 17–20 (2012)
26. Porter, M.F.: An Algorithm for Suffix Stripping. In: Readings in Information Retrieval. Hrsg. von K. Sparck Jones und P. Willett, S. 313–316. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA(1997)
27. Quirchmayr, T., Paech, B., Kohl, R., Karey, H., und Kasdepke, G.: Semi-automatic rule-based domain terminology and software feature-relevant information extraction from natural language user manuals. Empirical Software Engineering (2018)
28. Seiler, M., und Paech, B.: Using Tags to Support Feature Management Across Issue Tracking Systems and Version Control Systems. In: Grünbacher, P., und Perini, A. (Hrsg.) Requirements Engineering: Foundation for Software Quality, S. 174–180. Springer International Publishing, Cham (2017)
29. Taylor, A., Marcus, M., und Santorini, B.: The Penn Treebank: An Overview. In: Treebanks: Building and Using Parsed Corpora. Hrsg. von A. Abeillé, S. 5–22. Springer Netherlands, Dordrecht(2003)
30. Wang, S., Lo, D., Vasilescu, B., und Serebrenik, A.: EnTagRec ++: An enhanced tag recommendation system for software information sites. Empirical Software Engineering 23(2), 800–832 (2018)

31. Wang, T., Wang, H., Yin, G., Ling, C.X., Li, X., und Zou, P.: Tag recommendation for open source software. Frontiers of Computer Science 8(1), 69–82 (2014)

32. Wang, X.-Y., Xia, X., und Lo, D.: TagCombine: Recommending Tags to Contents in Software Information Sites. Journal of Computer Science and Technology 30(5), 1017–1035 (2015)

33. Wautelet, Y., Heng, S., Kolp, M., und Mirbel, I.: Unifying and Extending User Story Models. In: Jarke, M., Mylopoulos, J., Quix, C., Rolland, C., Manolopoulos, Y., Mouratidis, H., und Horkoff, J. (Hrsg.) Advanced Information Systems Engineering, S. 211–225. Springer International Publishing, Cham (2014)

Abbildungsverzeichnis

Tabellenverzeichnis

A. Ergebnisse der Suchanfragen in chronologischer Reihenfolge

A.1. Suchanfragen mit dem Suchterm 3.5

Suchort	Suchdatum	Such-beschränkungen	Suchanfrage	#Ergebnisse	#relevante Ergebnisse	verwendete Treffer	Bemerkungen
ACM	25.07.2018		+(feature +(extraction mining)) +("software requirement")	101	2	[2, 10]	
IEEE	25.07.2018		(feature AND (extraction OR mining)) AND software requirement	411	1	[13]	1) *software requirement* wurde nicht in Anführungszeichen gesetzt, um die Menge der Suchergebnisse zu erweitern
Wiley	25.07.2018		(feature AND (extraction OR mining)) AND "software requirement"	47	0		1) keine relevanten Suchtreffer
Springer	25.07.2018	Content Type: Article, Discipline: Computer Science	(feature AND (extraction OR mining)) AND "software requirement"	174	1	[27]	
ScienceDirect	25.07.2018		(feature AND (extraction OR mining)) AND "software requirement"	274	2	[3, 16]	

Tabelle A.1.: Suchergebnisse für Suchterm 3.5.

A.2. Suchanfragen mit dem Suchterm 3.2

Suchort	Suchdatum	Such-beschränkungen	Suchanfrage	#Ergebnisse	#relevante Ergebnisse	verwendete Treffer	Bemerkungen
ACM	26.07.2018		+((software requirement) (software repository)) +(tag label) +(recommend* assign*)	296	1	[11]	1) *software requirement* bzw. *software repository* wurde nicht in Anführungszeichen gesetzt, um die Menge der Suchergebnisse zu erweitern
IEEE	26.07.2018		(software requirement OR software repository) AND (tag OR label) AND (recommend* OR assign*)	75	0		1) *software requirement* bzw. *software repository* wurde nicht in Anführungszeichen gesetzt, um die Menge der Suchergebnisse zu erweitern 2) keine relevanten Suchtreffer
Wiley	26.07.2018		("software requirement"OR "software repository") AND (tag OR label) AND (recommend* OR assign*)	84	0		1) keine relevanten Suchtreffer
Springer	26.07.2018	Content Type: Article, Discipline: Computer Science	("software requirement"OR "software repository") AND (tag OR label) AND (recommend* OR assign*)	401	5	[31, 30, 32, 21, 17]	
ScienceDirect	26.07.2018	Article Type: Research Article	("software requirement"OR "software repository") AND (tag OR label) AND (recommend* OR assign*)	496	0		1) Suchergebnisse enthielten relevante Treffer, diese waren jedoch Duplikate bzgl. vorheriger Suchanfragen

Tabelle A.2.: Suchergebnisse für Suchterm 3.2.

A.3. Suchanfragen mit dem Suchterm 3.3

Suchort	Suchdatum	Such-beschränkungen	Suchanfrage	#Ergebnisse	#relevante Ergebnisse	verwendete Treffer	Bemerkungen
ACM	26.07.2018		+("software requirement""software repository") +(clustering annotation)	78	0		1) Suchergebnisse enthielten relevante Treffer, diese waren jedoch Duplikate bzgl. vorheriger Suchanfragen
IEEE	26.07.2018		(clustering OR annotation) AND (software requirement OR software repository)	959	2	[25, 23]	1) *software requirement* bzw. *software repository* wurde nicht in Anführungszeichen gesetzt, um die Menge der Suchergebnisse zu erweitern
Wiley	26.07.2018		(clustering OR annotation) AND ("software requirement"OR "software repository")	82	0		1) keine relevanten Suchtreffer
Springer	26.07.2018	Content Type: Article, Discipline: Computer Science	(clustering OR annotation) AND ("software requirement"OR "software repository")	396	0		1) Suchergebnisse enthielten relevante Treffer, diese waren jedoch Duplikate bzgl. vorheriger Suchanfragen
ScienceDirect	26.07.2018	Title, Abstract, Keywords	(clustering OR annotation) AND (software requirement OR software repository)	163	0		1) software requirement bzw. software repository wurde nicht in Anführungszeichen gesetzt, um die Menge der Suchergebnisse zu erweitern 2) Suchergebnisse enthielten relevante Treffer, diese waren jedoch Duplikate bzgl. vorheriger Suchanfragen

Tabelle A.3.: Suchergebnisse für Suchterm 3.3.

A.4. Suchanfragen mit dem Suchterm 3.4

Suchort	Suchdatum	Such-beschränkungen	Suchanfrage	#Ergebnisse	#relevante Ergebnisse	verwendete Treffer	Bemerkungen
ACM	27.07.2018		+(key* concept topic) +(model mining extract*) +("software requirement")	151	0		1) Suchergebnisse enthielten relevante Treffer, diese waren jedoch Duplikate bzgl. vorheriger Suchanfragen
IEEE	27.07.2018		(key* OR concept OR topic) AND (model OR mining OR extract*) AND "software requirement"	48	0		1) Suchergebnisse enthielten relevante Treffer, diese waren jedoch Duplikate bzgl. vorheriger Suchanfragen
Wiley	27.07.2018		(key* OR concept OR topic) AND (model OR mining OR extract*) AND "software requirement"	121	0		1) keine relevanten Suchtreffer
Springer	27.07.2018	Content Type: Article, Discipline: Computer Science	(key* OR concept OR topic) AND (model OR mining OR extract*) AND "software requirement"	457	0		1) Suchergebnisse enthielten relevante Treffer, diese waren jedoch Duplikate bzgl. vorheriger Suchanfragen
ScienceDirect	27.07.2018	Title, Abstract, Keywords	(key OR concept OR topic) AND (model OR mining OR extract) AND software requirement	379	0		1) *software requirement* bzw. *software repository* wurde nicht in Anführungszeichen gesetzt, um die Menge der Suchergebnisse zu erweitern 2) Suchergebnisse enthielten relevante Treffer, diese waren jedoch Duplikate bzgl. vorheriger Suchanfragen 3) der Wildcard-Operator * bei key* und extract* wird im Suchfeld Titel, Abstract, Keywords von ScienceDirect nicht unterstützt

Tabelle A.4.: Suchergebnisse für Suchterm 3.4.

B. Übersicht der Penn-Treebank-Labels

B.1. Labels für Wortarten

Label	Bedeutung
CC	Konjunktion
TO	Infinitiv *to*
CD	Kardinalzahl
UH	Interjektion
DT	Determinativ
VB	Verb, Grundform
EX	Demonstrativpronomen
VBD	Verb, Präteritum
FW	Fremdwort
VBG	Verb, Gerundium
IN	Präposition
VBN	Verb, Partizip
JJ	Adjektiv
VBP	Verb, *nicht* 3. Person Singular Präsens
JJR	Adjektiv, Komparativ
VBZ	Verb, 3. Person Singular Präsens
JJS	Adjektiv (Superlativ)
WDT	Determinativ (W-Wort)
LS	Aufzählungsmarker
WP	Pronomen (W-Wort)
MD	Modalverb
WP$	Possesivpronomen (W-Wort)
NN	Nomen, Singular oder Stoffname
WRB	Adverb (W-Wort)
NNS	Nomen, Plural
#	Pfund-Zeichen (Währung)
NNP	Eigenname, Singular
$	Dollar-Zeichen
NNPS	Eigenname, Plural
.	Punkt am Satzende
PDT	Indefinitpronomen
,	Komma
POS	Possessivaffix
:	Doppelpunkt, Semikolon
PRP	Personalpronomen
(	öffnende Klammer
PP$	Possessivpronomen
)	schließende Klammer
RB	Adverb

Tabelle B.1.: Penn-Treebank-Labels für Wortarten [20, 29].

B.2. Labels für syntaktische Einheiten

Label	Bedeutung
ADJP	Adjektivphrase
ADVP	Adverbialphrase
NP	Nominalphrase
PP	Präpositionalphrase
S	Hauptsatz
SBAR	Untergeordneter Nebensatz
SBARQ	Direkte Frage, eingeleitet durch ein W-Wort
SINV	Hauptsatz mit Inversion
SQ	Ja-Nein-Frage als Teil einer direkten Frage ohne W-Wort
VP	Verbalphrase
WHADVP	Adverbialphrase, eingeleitet durch ein W-Wort
WHNP	Nominalphrase, eingeleitet durch ein W-Wort
WHPP	Präpositionalphrase, eingeleitet durch ein W-Wort
X	Unbekanntes oder nicht einordbares Zeichen

Tabelle B.2.: Penn-Treebank-Labels für syntaktische Einheiten [20, 29].