Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Nutzung natürlichsprachiger Informationen aus dem Quelltext zur Identifikation von Sicherheitslücken

Bachelorarbeit

im Studiengang Informatik

von

Marius Hilbich

Prüfer: Prof. Dr. Joel Greenyer
Zweitprüfer: Prof. Dr. Kurt Schneider
Betreuer: Dipl.-Inform. Stefan Gärtner

Hannover, 5. September 2014

Zusammenfassung

Im Bereich der Softwareentwicklung spielt Sicherheit eine immer größere Rolle, denn durch neues Wissen können auch als bisher sicher geltende Systeme angreifbar werden. Solche Sicherheitslücken können in der Regel auf Fehler im Quellcode zurückgeführt werden, jedoch sind diese fehlerhaften Codeabschnitte nicht offensichtlich und müssen erst durch erneutes studieren des Quellcodes aufgedeckt werden. Um diesen Vorgang zu erleichtern gibt es Analysewerkzeuge zur statischen Codeanalyse (z.B. PMD, FindBugs), welche auf mögliche Fehler hinweisen, indem sie den Daten- und Kontrollfluss des Programms analysieren. Diese Tools nutzen jedoch nicht alle Informationen, die sich in einem Quelltext befinden, so werden Namen von Klassen, Variablen und Methoden oder auch Kommentare bisher kaum in die Analyse einbezogen. In dieser Arbeit wurde ein Verfahren erarbeitet, welches unabhängig von bisherigen Vorgehensweisen Quelltexte auf Sicherheitslücken untersucht. Dafür werden zunächst die natürlichsprachigen Informationen des Quelltextes aufbereitet und anschließend unter verschiedenen Verfahren aus dem Bereich des maschinellen Lernens analysiert.

Abstract

In the field of software engineering the role of security has an increasing importance, because systems which apply to be safe so far can be vulnerable due to the gain of new knowledgement. These vulnerabilities are mainly based on code errors, but these errors aren't obvious and can only be revealed by studying the source code once again. To facilitate this process, there are analysis tools for static code analysis (eg. PMD, FindBugs), which indicate possible errors by analyzing the data and control flow of the program. But these tools do not use all information which are included in the source code, natural language information such as names of classes, variables and methods or comments aren't analyzed. In this work, a method was developed which examined source code for security flaws independently of previous procedures. Therefore the natural language information is first prepared and then analyzed by different approaches of the field of machine learning.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Die Sicherheit von Software spielt eine immer größere Rolle, denn sobald ein System angreifbar ist, können erhebliche wirtschaftliche Schäden für den Vertreiber entstehen. Diese Sicherheit kann allerdings niemals garantiert werden, denn es werden immer wieder Schwachstellen in Form von Programmierfehlern aufgedeckt, welche es möglich machen, nicht gewollte Abläufe eines Programms zu erzwingen. Daher muss Software regelmäßig auf Schwachstellen geprüft werden.
Da die manuelle Analyse allerdings ein enormer Aufwand ist, werden Analysewerkzeuge genutzt, um direkt mögliche Schwachstellen aufzudecken. Diese Werkzeuge nutzen allerdings noch kaum Informationen, welche der Programmierer in Form von Bezeichnern (engl. identifier) oder auch Kommentaren automatisch in den Quellcode integriert. Diese natürlichsprachigen Informationen besitzen jedoch ein hohes Potenzial.
So hat unter anderem Emily Hill 2008 in einer Arbeit[1] gezeigt, dass Wartungssoftware durch das Normalisieren von Abkürzungen im Quellcode unterstützt werden kann.

1.2 Ziel der Arbeit

Das Ziel der Arbeit ist zunächst eine eigene Aufbereitung für die natürlichsprachigen Informationen eines Quelltextes herzuleiten und zu evaluieren. Das bedeutet die Trennung von Bezeichnern und die anschließende Normalisierung von Abkürzungen. Software für ein solches Vorgehen ist nicht öffentlich Verfügbar, und Ansätze aus bisherigen Arbeiten basieren nicht auf Java-Quellcode, wodurch diese Verfahren anderen Grundvoraussetzungen unterliegen.
Das zweites Ziel ist, die extrahierten Informationen für eine natürlichsprachige Analyse zu nutzen und dabei verschiedene Ansätze zu evaluieren.

1.3 Struktur der Arbeit

Diese Arbeit enthält insgesamt acht Kapitel. Nach dieser kurzen Einleitung folgen die Grundlagen, welche nötig sind, um die erläuterten Verfahren dieser Arbeit nachvollziehen zu können.

Im dritten Kapitel wird das gesamte Verfahren einmal im Zusammenhang erläutert, das heißt, es werden die einzelnen Schritte, welche in den Kapiteln vier bis sieben erarbeitet und evaluiert werden in einem Datenfluss dargestellt. Dabei wird auch die Art der übergebenen Informationen von einem zum nächsten Schritt erläutert.

Im vierten Kapitel wird das Parsen des Quelltextes erläutert, sowie die einhergehenden Probleme für die Extrahierung der natürlichsprachigen Informationen. Ebenso wird in diesem Kapitel auf den möglichen Informationsgehalt der Bezeichner eingegangen.

Das fünfte Kapitel beschreibt die Vorgehensweise bei der Worttrennung der Bezeichner. Hierbei wird sowohl die Herleitung des Verfahrens, als auch die Evaluation des Ergebnisses beschrieben.

Das sechste Kapitel beschäftigt sich mit der Normalisierung von möglichen Abkürzungen, welche in den Bezeichnern enthalten sein können. Dabei wird auf das Problem eingegangen, eine Abkürzung auf exakt ein Wort abzubilden. Ebenso werden die Abkürzungen in zwei Kategorien eingeteilt, welche darauf untersucht werden, ob diese in einem gesamten Projekt gültig sind, oder ob es eine lokal begrenzte Gültigkeit für Abkürzungen gibt. Letztendlich wird in diesem Kapitel die Normalisierung ausgewertet, um die Güte der extrahierten Worte für die Analyse beurteilen zu können.

Im siebten Kapitel werden die bis dahin aufbereiteten Informationen analysiert. Dabei wird die Grundlage für die in dieser Arbeit durchgeführten Analysen beschrieben. Danach wird der Prozess der Analyse erläutert, welcher in einem externen Programm durchgeführt wird. Da noch keine vergleichbaren Sicherheitsanalysen durchgeführt wurden, werden alle möglichen Aufbereitungen verglichen, sowie vier verschiedene Klassifikatoren für die Analyse genutzt.

Im achten und letzten Kapitel werden die evaluierten Verfahren und Analysen nochmals zusammengefasst. Ebenso wird ein Ausblick auf mögliche Erweiterungen, Automatisierungen oder Kombinationen des Verfahrens mit bisherigen Verfahren geliefert.

2 Grundlagen

2.1 Sicherheitslücken

Als Sicherheitslücke wird eine Schwachstelle in einer Software bezeichnet, wenn diese von einem Angreifer ausgenutzt werden kann, um beispielsweise Schaden anzurichten oder sich Zugriffsrechte zu verschaffen, welche ihm normalerweise untersagt wären[2].

Zur Einteilung von Sicherheitslücken in einzelne Kategorien wird die *Common Weakness Enumaration* (CWE)[3] genutzt, welche nicht nur eine Auflistung der Arten von Sicherheitslücken ist, sondern auch Methoden zum Detektieren der einzelnen Schwachstellen beschreibt und Quellcode-Beispiele liefert. Zudem werden alle Schwachstellen mit einem Bewertungssystem verglichen, welches unter anderem die Größe des Schadens, die Einfachheit der Ausnutzung einer Schwachstelle und die Häufigkeit des Auftretens dieser Schwachstelle einbezieht.

Da das in dieser Arbeit erstellte Analyseverfahren bereits dokumentierte Schwachstellen benötigt, um weitere detektieren zu können, werden hier diejenigen CWE beschrieben, welche auch als Schwachstellen in dem Testprojekt vorliegen. Die Definitionen orientieren sich an denen der *Common Weakness Enumaration*[3].

CWE-20

Unsachgemäße Eingabevalidierung (engl. Improper Input Validation):

Eine Eingabe wird nicht korrekt validiert, wodurch schädlicher Quelltext auf dem System ausgeführt werden kann, welcher den Kontroll- und Datenfluss des Programms beeinflusst.

CWE-22

Pfad Durchquerung (engl. Path Traversal):

Eine externe Eingabe wird genutzt, um den Pfad zu einer Datei zu konstruieren, welche in einem Verzeichnis mit eingeschränktem Zugriff liegt. Jedoch wird die Eingabe nicht auf Sonderzeichen geprüft, welche den Zugriff außerhalb des eingeschränkten Verzeichnisses ermöglichen.
Diese Schwachstelle hat eine Bewertung von 69,3 von maximalen 100 und belegt somit den 13. Rang in der CWE-Top25-Auflistung der gefährlichsten Sicherheitslücken.

CWE-79

Website übergreifendes Scripting (engl. Cross-site Scripting):

Eine externe Eingabe für eine Verbindung (engl. link) wird vor dem Veröffentlichen nicht überprüft, wodurch schädliche Skripte auf dem Rechner einer Zielperson ausgeführt werden können, falls die Person die Verbindung nutzt, da sie vertrauenswürdig erscheint.
Diese Schwachstelle hat eine Bewertung von 77,7 von maximalen 100 und belegt somit den 4. Rang in der CWE-Top25-Auflistung der gefährlichsten Sicherheitslücken.

CWE-200

Informationsaufdeckung (engl. Information Exposure):

Informationen werden an einen Nutzer freigegeben, welcher nicht explizit autorisiert ist, diese Informationen abzurufen.

CWE-264

Berechtigungen, Zugriffsrechte und Zugriffskontrolle (engl. Permissions, Privileges, and Access Controls):

Schwachstellen dieser Kategorie stehen in Verbindung mit der Zugriffskontrolle, welche durch Berechtigungen, Zugriffsrechte oder Ähnlichem realisiert werden.

CWE-287

Unsachgemäße Authentifizierung (engl. Improper Authentication):

Wenn ein Akteur eine gegebene Identität beansprucht, wird entweder nicht geprüft, ob er dazu autorisiert ist, oder es wird unzureichend geprüft.

CWE-399

Ressourcen Verwaltungsfehler (engl. Resource Management Errors):

Schwachstellen dieser Kategorie stehen in Verbindung mit der unsachgemäßen Verwaltung von Systemressourcen. Beispiel: Pufferüberlauf (engl. buffer overflow)

2.2 Klassifikatoren

Klassifikatoren sind Algorithmen, welche Objekte in einzelne Klassen einteilen können (klassifizieren). Dafür müssen sie jedoch zunächst durch eine Trainingsmenge, welche bereits eingeteilte Objekte enthält, eingelernt werden. Im Folgenden werden die in dieser Arbeit genutzten Klassifikatoren erläutert[4].

k-NN (engl. k-Nearest-Neighbor)

Der k-Nächste-Nachbarn Algorithmus klassifiziert ein Objekt - wie der Name bereits sagt - nach seinen k nächsten Nachbarn. Dabei sind Nachbarn Einträge aus der Trainingsmenge, welche die geringste Distanz zu dem zu klassifizierenden Objekt aufweisen. Je nachdem wie die Nachbarn des Objekts klassifiziert sind, wird das Objekt selbst auch klassifiziert. Da die Distanz zwischen dem Objekt und seinen Nachbarn erst während der Klassifizierung berechnet wird, muss beim Einlernen des Klassifikators lediglich die Trainingsmenge abgespeichert werden[4].

SVM (engl. Support Vector Machine)

Eine SVM teilt alle Objekte der Trainingsmenge durch Vektoren in Ihre Klassen auf, dabei werden die Vektoren so gewählt, dass die Distanz zwischen ihnen und den Objekten maximal ist. Um ein neues Objekt zu klassifizieren, wird lediglich geprüft, auf welcher Seite der Vektoren das Objekt liegt. Das Objekt wird danach der entsprechenden Klasse dieser Seite zugewiesen[4].

Entscheidungsbaum (engl. Decision Tree)

Ein Entscheidungsbaum erstellt durch die Trainingsmenge, mit der er eingelernt wird, das Modell eines Baumes, welcher mit einem Wurzelknoten beginnt. An diesem Knoten wird ein Attribut des zu klassifizierenden Objektes abgefragt, und je nach dem Wert wird an einer der beiden Kanten weitergegangen. Diese führt entweder zu einem weiteren Knoten, oder aber nach einer beliebigen Anzahl von Knoten zu einer Klassifizierung des Objektes, welche nach den zuvor abgefragten Attributen getroffen wird[4].

Perzeptron (engl. perceptron)

Ein Perzeptron ist ein künstliches Neuron, welches durch Verschachtelung zu einem neuronalen Netz zusammengesetzt werden kann. Ein Perzeptron besitzt einen Schwellenwert, welcher erfüllt werden muss, damit ein Objekt als positiv klassifiziert wird. Der Wert, welcher den Schwellenwert erfüllen soll, wird durch die Attribute des Objekts berechnet, wobei diese zusätzlich eine Gewichtung erhalten. Sowohl der Schwellenwert, als auch die Gewichtung der Attribute wird aus der Trainingsmenge berechnet, wodurch das Einlernen dieses Klassifikators im Verhältnis zu den zuvor genannten relativ aufwendig ist[4].

2.2.1 Beurteilung eines Klassifikators

Die Beurteilung eines Klassifikators erfolgt durch die Anzahl der korrekt bzw. inkorrekt beurteilten Daten, die ihm übergeben werden. Dabei können alle relevanten Werte durch eine Wahrheitsmatrix wie in Tabelle 2.1 berechnet bzw. veranschaulicht werden.

	negativ	**positiv**
Vorhersage negativ	richtig negativ (RN)	falsch negativ (FN)
Vorhersage positiv	falsch positiv (FP)	richtig positiv (RP)

Tabelle 2.1: Wahrheitsmatrix eines Klassifikators

Die Daten, welche negativ sind und vom Klassifikator als negativ beurteilt werden, sind unter *richtig negativ* aufgeführt. Dem entsprechend sind alle Daten, welche positiv sind, aber dennoch vom Klassifikator als negativ beurteilt werden unter *falsch negativ* aufgeführt. Ebenso sind die Daten, welche vom Klassifikator als positiv beurteilt werden, und auch positiv sind, unter *richtig positiv* aufgeführt. Und alle Daten, welche zwar negativ sind, aber dennoch vom Klassifikator als positiv beurteilt werden unter *falsch positiv* aufgeführt. Aus diesen Beurteilungen lassen sich die folgenden Werte berechnen, welche der Bewertung eines Klassifikators dienen.

Trefferquote

Die Trefferquote gibt den Anteil der korrekt als positiv erkannten Daten im Vergleich zu allen tatsächlich positiven Daten an. Sie lässt sich durch folgende Formel berechnen:

$$Trefferquote = \frac{Anzahl\ richtig\ positiv}{Anzahl\ tatsächlich\ positiv} = \frac{\#RP}{\#RP + \#FN} \in [0,1]$$

Präzision

Die Präzision gibt den Anteil der korrekt als positiv erkannten Daten im Verhältnis zu allen als positiv vorhergesagten Daten an. Sie lässt sich durch folgende Formel berechnen:

$$Präzision = \frac{Anzahl\ richtig\ positiv}{Anzahl\ positiv\ vorhergesagt} = \frac{\#RP}{\#RP + \#FP} \in [0,1]$$

F_1-Maß

Da die Trefferquote und die Präzision unmittelbar zusammenhängen, wird mit dem F_1-Maß ein Wert berechnet, welcher die Beurteilung beider Werte durch einen gemeinsamen ermöglicht. Das F_1-Maß ist ein spezieller Fall des F_n-Maßes, welches die Trefferquote und Präzision gleich stark gewichtet. Es lässt sich durch folgende Formel berechnen:

$$F1 - Maß = 2 * \frac{Trefferquote * Präzision}{Trefferquote + Präzision} \in [0,1]$$

Falsch-Positiv-Rate

Die Falsch-Positiv-Rate (FPR) gibt den Anteil der tatsächlich negativen Daten an, welche fälschlicherweise als positiv beurteilt werden. Sie lässt sich durch folgende Formel berechnen:

$$FPR = \frac{Anzahl\ falsch\ positiv}{Anzahl\ tatsächlich\ negativ} = \frac{\#FP}{\#FP + \#RN} \in [0,1]$$

Falsch-Negativ-Rate

Die Falsch-Negativ-Rate (FNR) gibt den Anteil der tatsächlich positiven Daten an, welche fälschlicherweise als negativ beurteilt werden. Sie lässt sich durch folgende Formel berechnen:

$$FNR = \frac{Anzahl\ falsch\ negativ}{Anzahl\ tatsächlich\ positiv} = 1 - Trefferquote = \frac{\#FN}{\#FN + \#RP} \in [0,1]$$

3 Ansatz für die natürlichsprachige Analyse

Die natürlichsprachigen Informationen, welche in einem Quelltext vorliegen, sind in der Regel keine Fließtexte, welche aus Wörtern bestehen, die in einem Wörterbuch nachgeschlagen werden können. Eine Ausnahme können an dieser Stelle Kommentare darstellen, auf welche in Kapitel 6 näher eingegangen wird. Meist sind die Informationen aber in - von den Programmierern persönlich gewählten - Abkürzungen enthalten, welche erst durch eine Aufbereitung genutzt werden können. Diese Aufbereitung des Quelltextes kann in drei einzelne Schritte unterteilt werden, welche unabhängig voneinander erarbeitet und evaluiert werden. Zuletzt folgt die tatsächliche Analyse, welche auf der Aufbereitung aufbaut und ebenfalls einzeln evaluiert wird. Eine entsprechende Visualisierung ist in Abbildung 3.1 dargestellt.

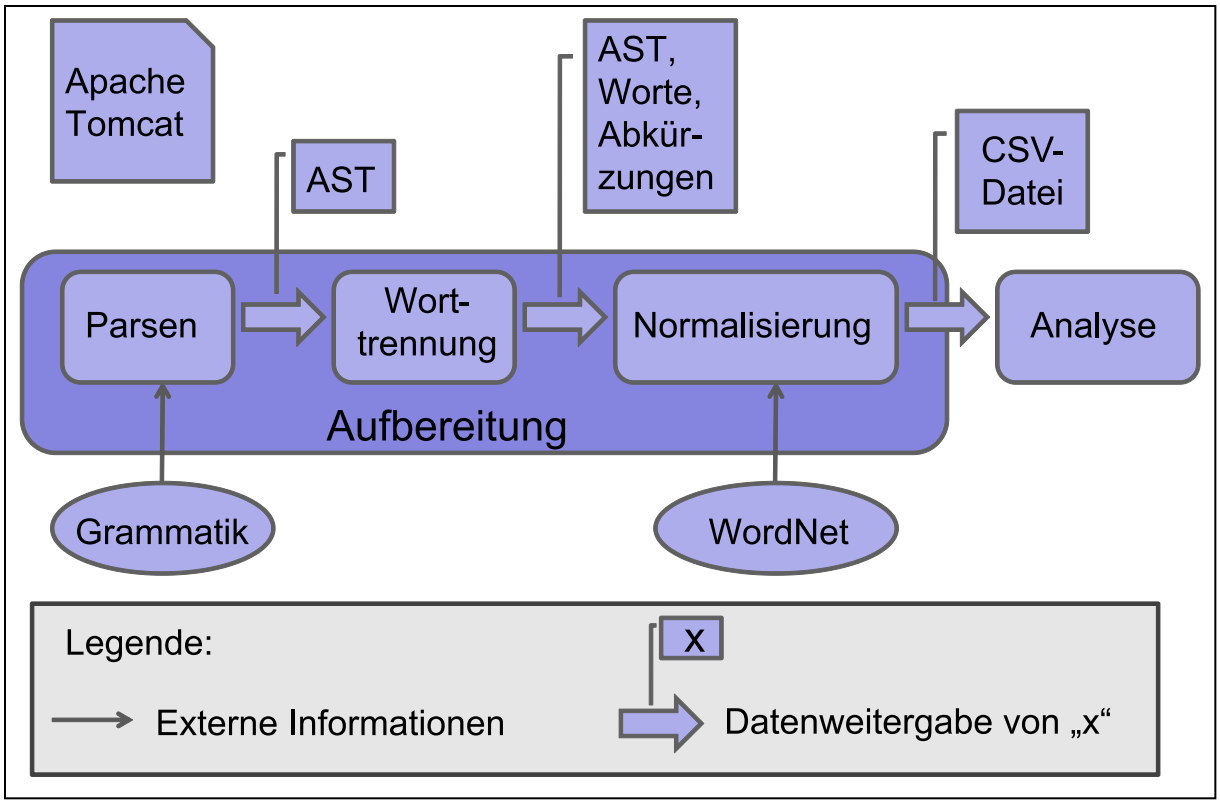

Abbildung 3.1: Visualisierung des Analyseansatzes

Der erste Schritt ist das Parsen des Quelltextes. Denn um den Quelltext korrekt auslesen zu können, muss dieser zunächst in einen abstrakten Syntaxbaum (AST)[5] umgeformt werden, aus welchem direkt die Bezeichner von Klassen, Methoden oder Variablen entnommen werden können. Damit der Parser den Quelltext in einen Syntaxbaum umformen kann, benötigt dieser eine Grammatik.

Nachdem ein Syntaxbaum erstellt wurde und auf alle Bezeichner zugegriffen werden kann, werden diese im nächsten Schritt in ihre Teilworte aufgetrennt (engl. splitting). Denn oftmals bestehen Bezeichner aus mehr als nur einem Wort und da sie keine Leerzeichen enthalten können, werden Teilworte durch andere Methoden getrennt.

Nach der Worttrennung folgt die Normalisierung von wohlmöglich vorhandenen Abkürzungen einzelner Teilworte. Um lediglich grammatikalisch korrekte Worte zuzulassen, wird an dieser Stelle ein Wörterbuch benötigt. In dieser Arbeit wird die Datenbank von WordNet[1] genutzt, da diese weitere Funktionen, wie das Unterteilen der Worte in die Wortarten Nomen, Adjektive und Verben bietet.

[1] http://wordnet.princeton.edu/, Version: 3.1

Die ersten drei Schritte machen die Aufbereitung des Quelltextes aus und finden komplett in einem Programm statt. Der letzte Schritt hingegen wird mit einem zweiten Programm realisiert, welches die Datenanalyse umsetzt. Dieses wird die extrahierten Worte, welche nach der Normalisierung in CSV-Dateien gespeichert werden auslesen, und anschließend mithilfe der in Kapitel 2.2 aufgeführten Klassifikatoren analysieren.

Als Ausgabe sollten die Namen der sicherheitsrelevanten Methoden des geparsten Quelltextes zurückgegeben werden. Diese müssen danach manuell nach den entsprechenden Sicherheitslücken überprüft und überarbeitet werden.

In den Folgenden Kapiteln wird Apache Tomcat[2] als Testprojekt genutzt. Dieses sollte einen repräsentativen Charakter haben, da an diesem über viele Jahre von mehreren Personen gearbeitet wurde. Außerdem liegen mir für Apache Tomcat dokumentierte Schwachstellen vor, welche das Auswerten der Sicherheitsanalyse erst ermöglichen.

[2] http://tomcat.apache.org/, Version: 6

4 Parsen des Quelltextes

Um mit dem Quelltext arbeiten zu können, muss dieser zunächst zu einem Syntaxbaum geparst werden. Zum Erstellen des Parsers wurde ANTLR[3] genutzt, denn dieses Tool benötigt lediglich eine Grammatik, welche alle syntaktischen Möglichkeiten durch Produktionen beschreibt. Für Java-Quellcode ist eine solche Grammatik auf GitHub[4] bereits öffentlich zugänglich. Diese Grammatik überließt allerdings alle Kommentare, da sie für die Struktur des Quelltextes nicht weiter relevant sind. Die Grammatik dahingehend zu erweitern, dass alle Kommentare geparst werden ist zwar möglich, aber da Kommentare an jeder Stelle des Quellcodes stehen können, wäre dies mit sehr hohem Aufwand verbunden. Deshalb wird vorerst auf die Nutzung der Kommentare verzichtet, jedoch wird diese Möglichkeit in Kapitel 6.5 aufgegriffen.

4.1 Auslesen des Syntaxbaums

Um auf den Syntaxbaum zuzugreifen wird ein Besucher (engl. Visitor) genutzt, dieser kann auf jeden einzelnen Knoten zugreifen. Erweitert wird er durch die Funktionalität, dass beim Besuchen einer Paket-, Klassen-, Methoden-, oder Variablendeklaration der Bezeichner ausgelesen und an die Worttrennung übergeben wird. Zur Veranschaulichung ist in Abbildung 4.1 der Syntaxbaum der folgenden Beispielklasse dargestellt. In diesem sind die Deklarationen, welche besucht werden grün unterstrichen. Von diesen kann entweder direkt oder über weitere Knoten auf die Bezeichner zugegriffen werden, welche rot eingekreist sind. Lediglich bei den Methodendeklarationen wird nicht nur auf den Methodennamen (III), sondern auch auf die übergebenen Parameter (IV) zugegriffen, wobei an dieser Stelle weitere Parameter stehen könnten.

BeispielKlasse.java

```
1   package testpackage;
2
3   public class BeispielKlasse {
4     public void beispielMethode(int wert){
5           String nameDesBezeichners = "test";
6     }
7   }
```

[3] http://www.antlr.org/, Version: 4.4
[4] https://github.com/antlr/grammars-v4/blob/master/java/Java.g4

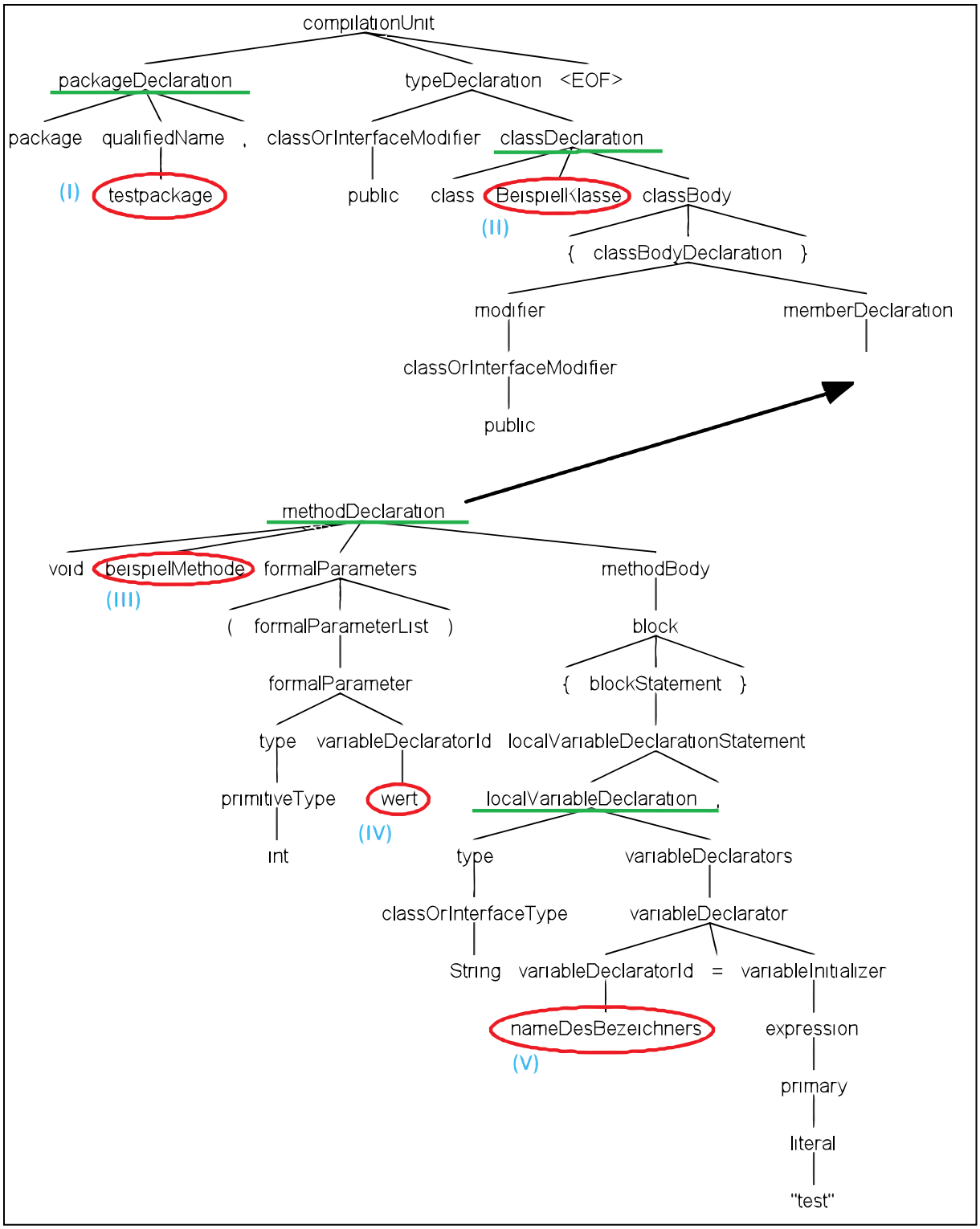

Abbildung 4.1: Syntaxbaum der Beispielklasse

4.2 Informationsgehalt der Bezeichner

Im Testprojekt[5] werden insgesamt 40578 Klassen, Methoden und lokale Variablen initialisiert, wodurch auch 40578 Bezeichner vorhanden sind. In Tabelle 4.1 ist zu sehen, dass von allen Bezeichnern 18,3% eine Länge von lediglich einem bis drei Zeichen haben. Das heißt, fast ein Fünftel der Bezeichner weist ohne weitere Aufbereitung lediglich codierte Informationen auf, daher ist eine Normalisierung der Bezeichner wichtig, um so viele natürlichsprachige Informationen wie möglich aufzubereiten und für die Analyse nutzen zu können. Die Bezeichner mit einer Länge von nur einem Zeichen sind jedoch in der Regel Abkürzungen von primitiven Datentypen oder temporäre Zählvariablen. Aus diesen lassen sich keine Informationen gewinnen, was somit bedeutet, dass bereits 7,66% der Bezeichner keinen Nutzen für die Analyse haben.

Bezeichnerlänge	Anzahl	Anteil
1	3107	7,66%
2	1367	3,37%
3	2950	7,27%
Summe	7424	18,30%

Tabelle 4.1: Prozentualer Anteil kleiner Bezeichnerlängen

[5] http://tomcat.apache.org/, Version: 6

5 Worttrennung

Die Bezeichner, welche mit Hilfe des Besuchers aus dem Quelltext extrahiert werden, bestehen oftmals noch aus mehreren Teilworten und Abkürzungen. Diese sind jedoch nicht wie in einem normalen Text durch ein Leerzeichen getrennt, da dies in einer Programmiersprache nicht möglich ist. Das heißt, es müssen zuerst in jedem Bezeichner die einzelnen Teilworte identifiziert und voneinander getrennt werden, bevor sie im nächsten Schritt, falls es Abkürzungen sind, zu grammatikalisch korrekten Worten normalisiert werden können.

Ein Standartverfahren hierfür ist die Camel-Case Teilung. Dabei werden Bezeichner immer dann getrennt, wenn auf ein Kleinbuchstabe ein Großbuchstabe folgt, denn zur besseren Lesbarkeit beginnen Programmierer oft einzelne Worte in Bezeichnern mit einem Großbuchstaben. Dies ist allerdings keine allgemeine Vorschrift, weshalb dieses Verfahren für die meisten Programmiersprachen keine guten Ergebnisse liefert.

Aus diesem Grund wurde 2011 in einer Arbeit ein Algorithmus namens TIDIER [6] erstellt, welcher Bezeichner in C-Quellcode durch ein effizienteres Verfahren in einzelne Worte trennt. Dabei werden trotzdem vorerst alle Bezeichner durch einen Camel-Case Teiler getrennt, jedoch werden die dadurch entstandenen Teilworte auf weitere Trennungen untersucht. Dafür wird eine Vielzahl von Wörterbüchern genutzt, mit welchen die Editierungsdistanz (Anzahl an Umformungen) zwischen einem Teilwort und einem möglichen Wort aus einem der Wörterbücher berechnet wird. Ist diese Distanz gleich 0, ist das Wort in einem der Wörterbücher enthalten und wird nicht weiter getrennt, ist sie jedoch größer 0 ist das Teilwort entweder eine Abkürzung oder es enthält weitere Teilworte. Dementsprechend wird die Distanz von möglichen Teilworten mit einer Länge größer drei berechnet und sollte die Summe ihrer Distanzen kleiner sein als die des vorherigen Teilworts, wird das Wort getrennt.

Zusätzlich wurde in dieser Arbeit auf die Code-Konventionen[7] in Java-Quellcode eingegangen, welche es möglich machen, dass sowohl der TIDIER Algorithmus, als auch eine einfache Camel-Case Teilung korrekte Trennungen von über 90% erreichen. Denn Java ist die erste Programmiersprache, unter welcher sich Code-Konventionen allgemein etabliert haben. Diese geben die Benennung von Variablen, Methoden und Klassen vor. Da der Quellcode zu TIDIER oder ähnlichem nicht öffentlich zugänglich ist und das in dieser Arbeit vorgestellte Analyseverfahren lediglich in Java evaluiert wird, wird ein einfacheres Trennungsverfahren hergeleitet, welches auf den Code-Konventionen basiert.

5.1 Herleitung des regulären Ausdrucks

Aus den Java Code-Konventionen können Regeln abgeleitet werden, unter welchen Voraussetzungen ein Bezeichner an einer bestimmten Stelle in mehrere Teilworte getrennt werden muss.
Folgende Konventionen dienen der Worttrennung als Grundlage:

- Jedes Teilwort - außer das erste in einem Bezeichner - beginnt mit einem Großbuchstaben (engl. camel case). Lediglich in Klassennamen beginnt auch das erste Wort mit einem Großbuchstaben.
- Konstanten werden komplett großgeschrieben und deren Teilwörter durch ein Unterstrich getrennt.

Daraus können vier Regeln für die Worttrennung abgeleitet werden. Zu jeder Regel ist entsprechend der Nummerierung ein Beispiel in Beispiel 5.1 angegeben.

1. Es wird hinter allen Kleinbuchstaben getrennt, auf die ein Großbuchstabe folgt.
2. Da einzelne Teilworte auch komplett großgeschrieben werden können, wird bei einer Folge von Großbuchstaben vor dem letzten getrennt, falls auf diesen ein Kleinbuchstabe folgt.
3. Es wird vor und hinter Unterstrichen getrennt.
4. Da auch Zahlen in Bezeichnern erlaubt sind, wird ebenso vor und hinter Zahlen getrennt.

Beispiel 5.1: Herleitung der Worttrennung

(1)	currentTime	→	current, time
(2)	ASTVisitor	→	ast, visitor
(3)	SPLIT_REGEX	→	split, regex
(4)	put10Entries	→	put, 10, entries

Aus diesen Regeln wird ein regulären Ausdruck erstellt, welcher in Abbildung 5.1 in Java-Code dargestellt ist. Dabei ist die Funktionalität der einzelnen Abschnitte des regulären Ausdrucks in der Abbildung in einzelnen Kommentaren erläutert.

```
String SPLIT_REGEX = "(?=[A-Z][a-z])(?!^)|"   // vor jedem Gb gefolgt von einem Kb (nicht vor erstem Zeichen)
        + "(?<=[a-z])(?=[A-Z0-9])|"           // hinter allen Kb gefolgt von einem Gb oder einer Z
        + "(?<=[0-9])(?=([A-Z]|[a-z]))|"      // hinter allen Z gefolgt von einem Gb oder Kb
        + "(?<=[A-Z])(?=[0-9])|"              // hinter allen Gb gefolgt von einer Zahl
        + "_";                                // vor und hinter unterstrichen
```

Abbildung 5.1: Regulärer Ausdruck der Worttrennung in Java

5.2 Evaluation

Um die Worttrennung des regulären Ausdrucks zu beurteilen, wird der Anteil der korrekt geteilten Worte des Testprojekts[6] genutzt. Da dieses jedoch 40578 Bezeichner von Klassen, Methoden und lokalen Variablen enthält, welche aus zeitlichem Aufwand nicht alle geprüft werden können, ob sie korrekt in Ihre Teilworte getrennt werden, wird eine zufällige Stichprobe (ohne Zurücklegen) des Testprojekts gewählt, um eine Aussage über die Güte der Worttrennung treffen zu können.

5.2.1 Berechnung des Stichprobenumfangs

Diese Stichprobe muss jedoch für eine begründete Aussage einen bestimmten Umfang haben, welcher sich durch folgende Formel berechnen lässt:

n: Stichprobenumfang
N: Gesamtumfang
z: Wert der Standartnormalverteilung
θ: Anteilswert
ΔΦ: Unsicherheitsbereich

$$n \geq \frac{z^2\theta(1-\theta)N}{(\Delta\phi)^2(N-1)+z^2\theta(1-\theta)} \quad [8]$$

Der Gesamtumfang sind alle Bezeichner des Projektes: **N = 40578**

Um die Aussage mit einer Sicherheit von 99% treffen zu können, muss ein Vertrauensintervall von 99% gewählt werden. Daraus ergibt sich als Wert der Standartnormalverteilung: **z = 2,58**

[6] http://tomcat.apache.org/, Version: 6

Der Anteilswert beschreibt, wie viel Prozent der Teilworte korrekt sein werden, umso näher dieser Wert an 50% liegt, desto größer wird die Stichprobe. Wie bereits erwähnt erzielt ein simpler Camel-Case Trenner (welcher nicht weiter erläutert wurde) in der Arbeit zu TIDIER[6] auf Java-Code weit über 90% korrekte Trennungen. Daher wird der Anteilswert auf mindestens 90% geschätzt: $\boldsymbol{\theta = 0{,}9}$

Der Unsicherheitsbereich gibt an, wie viel Prozent der wahre Wert unter oder über dem Ergebnis der Stichprobe liegen kann. Da dieser Wert die Größe der Stichprobe quadratisch beeinflusst und lediglich die Abweichung nach unten relevant ist, da der schlechtesten Fall als Bewertung angenommen wird, wird ein Wert von 2% gewählt: $\boldsymbol{\Delta\Phi = 0{,}02}$

Mit Hilfe dieser Werte ergibt sich ein Stichprobenumfang von mindestens **n=1444** Bezeichnern. Das entspricht etwa 3,56% aller Bezeichner des Testprojektes. Um diese Anzahl mindestens zu erfüllen, werden für die Auswertung alle Bezeichner geparst und mit einer Wahrscheinlich von 3,7% per Zufallszahl ausgewählt und sowohl der ursprüngliche Bezeichner, als auch die durch den regulären Ausdruck erstellten Teilworte gespeichert. Dadurch wurden 1553 Bezeichner ausgewählt, was etwa 3,83% entspricht und somit die Mindestanzahl des zuvor berechneten Stichprobenumfangs erfüllt.

5.2.2 Auswertung der Stichprobe

Die Stichprobe wurde manuell geprüft, wobei jeder Bezeichner der mindestens eine falsche Worttrennung enthält, als falsch getrennt gilt. Die Ergebnisse sind in Tabelle 5.1 aufgeführt. Da es sein konnte, dass eine der drei Kategorien Klassen, Methoden oder Variablen eine abweichende Fehlerquote bei der Worttrennung hat, wurde zunächst das Verhältnis dieser Kategorien in der Stichprobe mit denen in dem gesamten Projekt Apache Tomcat verglichen. Diese Verhältnisse liegen nah beieinander und zudem sind die Fehlerquoten der einzelnen Kategorien ebenfalls nahezu identisch. Um jedoch eine begründete Aussage über eine einzelne Kategorie machen zu können, muss die Stichprobe noch größer sein.

	Apache Tomcat gesamt		**Stichprobe**			
	Anzahl	**Verhältnis**	**Anzahl**	**Verhältnis**	**Fehler**	**Fehlerquote**
Klassen	1304	3,21%	48	3,09%	1	2,08%
Methoden	13633	33,6%	512	32,97%	13	2,54%
Variablen	25641	63,19%	993	63,94%	26	2,62%
gesamt	40578	100%	1553	100%	40	2,58%

Tabelle 5.1: Stichprobenergebnis der Worttrennung

Da die Gesamt-Fehlerquote der Stichprobe bei 2,58% liegt, kann durch die zuvor gewählten Parameter mit einer Sicherheit von 99% gesagt werden, dass der reguläre Ausdruck bei der Worttrennung eine Fehlerquote zwischen 0,58% und 4,58% aufweist. Beziehungsweise, werden zwischen 95,42% und 99,42% der Bezeichner korrekt getrennt. Mit einer Worttrennung von mindestens 95% Genauigkeit kann bereits eine fundierte Analyse bzw. im nächsten Schritt die Normalisierung durchgeführt werden, jedoch werden im Folgenden trotzdem noch die fehlerhaften Worttrennungen analysieren, um zu überprüfen, ob das Ergebnis noch verbessert werden kann.

5.3 Ursache von falschen Worttrennungen

Für die Analyse der falschen Worttrennungen auf Merkmale stehen zwar nur 40 Einträge, welche aus der Stichprobe hervorgegangen sind zur Verfügung, dennoch konnten zwei bzw. drei verschiedene Fehlerarten als häufigste Ursachen ausgemacht werden.

1. Bei der Benennung der Bezeichner wurde die Vorgabe der Nutzung vom Camel-Case nicht eingehalten. In Beispiel 5.2 sind unter dem ersten Punkt drei Beispiele aufgeführt; der erste Fall trat am häufigsten auf. Dabei wird ein Wort durch einen einzelnen Buchstaben abgekürzt (in diesem Fall das "o") und vor ein anderes angeheftet.
2. Sobald ein Klassenname die Camel-Case Vorgabe nicht einhält, wird der Fehler in alle Methoden und Variablen übernommen. In Beispiel 5.2 ist im zweiten Unterpunkt die Klasse angegeben welche fälschlicherweise nach "send" kleingeschrieben wurde. Unter den falschen Worttrennungen waren mehrere Methoden, welche diesen Fehler übernommen haben, da sie auf die Klasse verweisen.
3. Der reguläre Ausdruck behandelt nicht alle Fälle korrekt. Denn sobald ein komplett großgeschriebenes Teilwort ein kleines endständiges Plural-S enthält, wird dieses gemeinsam mit dem letzten Großbuchstaben als eigenes Wort interpretiert. In Beispiel 5.2 ist im dritten Unterpunkt eine Worttrennung angegeben, bei welcher das Wort "JARs" in "ja" und "rs" getrennt wurde, obwohl dieses bereits ein Wort war.

Beispiel 5.2: Falsche Worttrennungen

(1)	Kein Camel-Case:	oname, memread, thefile		
(2)	Klasse mit Camel-Case Fehler :	SendfileData		
(3)	Fehler im regulären Ausdruck:	openJARs	→	open, JA, Rs

Um die fehlenden Trennungen aus dem ersten Punkt zu identifizieren, könnten die Bezeichner zusätzlich nach dem Trennen auf anfängliche oder endständige Worte untersucht werden. Diese müssten dann mit einem Wörterbuch geprüft und mit dem ursprünglichen Wort verglichen werden. Da Wörterbücher aber sehr umfangreich sind und viele Akronyme enthalten, könnten wiederum fehlerhafte Treffer gefunden werden. Der zweite Punkt würde auf dieselbe Weise gelöst werden.
Der dritte Punkt könnte mit einer Fallunterscheidung im regulären Ausdruck behandelt werden, bei der keine Trennung durchgeführt wird, falls nach einer Folge Großbuchstaben nur ein einzelnes Plural-S folgt. Dieser Fall ist jedoch sehr selten, weshalb die Fallunterscheidung nicht weiter untersucht wird.

Eine übergeordnete Ursache für die falschen Trennungen bzw. die nicht eingehaltenen Code-Konventionen könnte darin bestehen, dass sich lediglich einzelne Autoren nicht an diese halten. Das kann die Ursache haben, dass ein Autor sich einen Programmierstil angeeignet hat, welcher den Code-Konventionen widerspricht, aber in anderen Programmiersprachen (z.B. C-Code) erlaubt ist.
Um dies zu untersuchen, werden alle Pakete (engl. packages), Klassen und Autoren, in denen die falschen Trennungen aus der Stichprobe auftraten miteinander verglichen. Als erstes fällt auf, dass 30% aller falschen Trennungen aus lediglich einem Paket stammen. Ebenso stammen 30% (mit einer Überschneidung zu den vorherigen) aus lediglich drei unterschiedlichen Klassen. Und der am meisten beteiligte Autor an diesem Paket und den Klassen, ist Mladen Turk, welcher insgesamt an 27,5% der Bezeichner beteiligt ist, welche nicht den Code-Konventionen entsprechen. An den restlichen falschen Trennungen waren zum Vergleich 13 weitere Autoren beteiligt.
Das zeigt, dass ein einzelner Autor einen großen Einfluss auf die Stichprobe hatte und das Ergebnis vielleicht sogar noch besser ist, als es die Stichprobe vermuten lässt.

6 Normalisierung der Abkürzungen

Für die folgende Analyse könnten bereits die nicht normalisierten Teilworte verwendet werden, jedoch werden die Abkürzungen der Bezeichner von Programmierern frei gewählt, wodurch ein Wort innerhalb eines Projektes zwei unterschiedliche Abkürzungen haben kann. Diese unterschiedlichen Abkürzungen würden ohne eine Normalisierung bei der Analyse nicht in Verbindung gebracht werden, wodurch die Ergebnisse verfälscht werden.
Im Schritt der Normalisierung werden somit die Teilworte, die nach der Worttrennung vorliegen zunächst darauf geprüft, ob sie bereits grammatikalisch korrekte Worte und keine Abkürzungen sind. Falls dies nicht zutrifft werden die Teilworte zu existierenden Worten normalisiert. Da jedoch der Wortschatz eines Wörterbuchs viel zu umfangreich ist, und somit zu oft Abkürzungen zu den falschen Worten normalisieren würde, wird ein eigener Wortschatz aus dem Quelltext des zu analysierenden Projektes erstellt.

6.1 Generieren eines Wortschatzes

Dafür wird das gesamte Projekt einmal geparst und alle Bezeichner in ihre Teilworte getrennt. Diese werden mit WordNet überprüft, ob sie im Wörterbuch enthalten sind. Dabei wird unterschieden, ob das Teilwort aus einem Klassennamen, einer Variable oder einem Methodennamen stammt. Denn laut Code-Konventionen sollen Klassen- und Variablennamen aus Nomen bestehen und Methoden aus Verben. Da sich Methoden oftmals auf Klassen oder Variablen beziehen und daher deren Namen enthalten können, können diese auch Nomen enthalten. Hinzu kommt das alle drei Kategorien zusätzlich Adjektive enthalten können um die Aussagekraft des Bezeichners zu erhöhen. Somit ergibt sich, dass alle drei Kategorien Nomen oder Adjektive enthalten und Methoden zusätzlich aus Verben bestehen können. Außerdem werden auch die Namen der Pakete und der Datentypen von Methoden sowie von Variablen genutzt, welche ebenfalls keine Verben enthalten.
Nachdem nun ein Teilwort in der Datenbank von WordNet einen Treffer erzielt, und je nach Herkunft die zuvor genannten Kriterien erfüllt, wird dieses Wort in eine Wortliste eingetragen, welche den Wortschatz symbolisiert. Dabei werden Verben in eine gesonderte Liste eingetragen, auf die, bei der Normalisierung nur für Methodennamen zurückgegriffen wird. Da WordNet eine Vielzahl von Akronymen enthält, welche nicht in die Wortliste übernommen werden sollen, müssen diese zusätzlich gefiltert werden. Dies ist jedoch nicht weiter schwer, da sie in der Datenbank komplett großgeschrieben werden, und somit alle komplett großgeschriebenen Nomen der Wortliste nicht hinzugefügt werden.
Um im weiteren Verfahren die Normalisierung von Nomen auf Ihre Pluralform zu verhindern, werden alle Nomen zusätzlich darauf geprüft ob sie in der Singularform vorliegen. Auch hier kann auf eine Funktion von WordNet zurückgegriffen werden, welche alle möglichen Grundformen zu einem Nomen liefert. Diese Funktion ist allerdings nicht sehr zuverlässig, da sie beispielsweise Singularformen die auf einem "s" enden, als Pluralform erkennt und das endständige "s" abschneidet. Daher werden die Singularformen, welche diese Funktion zurückgibt erneut mit WordNet geprüft, und falls es dieses Wort in der Datenbank nicht gibt, ist es sehr wahrscheinlich, dass keine Grundform existiert. In diesem Fall wird lediglich das ursprüngliche Nomen in die Wortliste aufgenommen, andernfalls wird sowohl das ursprüngliche Nomen, als auch dessen Singularform in der Wortliste gespeichert.
Zusätzlich zu der Wortliste und der gesonderten Liste mit Verben wird eine Stoppwortliste[7] genutzt, welche - wie der Name bereits verrät - aus Stoppworten besteht. Stoppworte sind Worte die häufig auftreten und daher kaum inhaltliche Relevanz haben. Diese Liste soll verhindern, dass solche Worte im weiteren Verfahren normalisiert werden oder Abkürzungen zu Stoppworten normalisiert werden und das Ergebnis verfälschen.
Nach der Fertigstellung der Wortlisten wird der Quelltext ein zweites Mal geparst und die Bezeichner getrennt, um für alle Teilworte, die eine Abkürzung und somit nicht in den Wortlisten sind, eine Normalisierung zu finden. Dabei werden die Abkürzungen nochmals in konventionelle Abkürzungen und Akronyme unterteilt.

[7] http://www.ranks.nl/stopwords, Stand: 01.08.2014

6.2 Akronyme

Akronyme sind eine bestimmte Art von Abkürzungen. Da sie ein Sonderfall der konventionellen Abkürzungen sind und somit nur einen geringen Anteil aller Abkürzungen ausmachen, werden diese der allgemeinen Normalisierung vorgezogen.
Akronyme bestehen aus den Anfangsbuchstaben der Worte, die sie abkürzen sollen. Das macht es einfach, sie zu Normalisieren, da sich die Akronyme in der Regel auf den Datentyp beziehen mit dem sie initialisiert werden. Es wird also bei allen Initialisierungen zunächst die Wortlänge des Bezeichners mit der Anzahl an Teilworten im Datentyp verglichen (außer der Bezeichner steht bereits in einer Wortliste), falls diese übereinstimmen wird nacheinander jeweils der erste Buchstabe des Bezeichners mit dem Anfangsbuchstaben des ersten Teilworts verglichen, dann der zweite und so weiter. Falls alle übereinstimmen wird dieser Bezeichner als Akronym für den Datentypen (bzw. die Worte) zu einer Akronymliste hinzugefügt, welche alle Akronyme und deren ausgeschriebene Bezeichnung enthält.
Da durch dieses Verfahren auch ein einzelner Buchstabe als Akronym eines Wortes erkannt wird, welches aus nur einem Teilwort besteht und den gleichen Anfangsbuchstaben hat, wird die Mindestwortlänge für Akronyme auf zwei gesetzt. Denn eine solche Normalisierung ist zwar zunächst korrekt, aber da Variablen mit einer Länge von nur einem Zeichen sehr oft auftreten (siehe Tabelle 4.1) ist es sehr wahrscheinlich, dass sich zwei Variablen mit der Bedeutung widersprechen. Dies könnte verhindert werden, indem die Normalisierung von Akronymen auf die Methode beschränkt wird, in der sie erkannt wurde, da die letztendliche Analyse jedoch zumindest auf Methodenebene stattfindet und in der Methode das Wort bereits gestanden haben muss, um das Akronym zu erkennen, hat dieses Vorgehen keinen weiteren Nutzen.
Hinzu kommt, dass Variablen mit einer Wortlänge von nur einem Zeichen meistens Abkürzungen für primitive Datentypen wie *Integer*, *String* oder *Double* sind, welche nicht viel Informationsgehalt besitzen.

6.3 Abkürzungen

Als Abkürzungen werden letztendlich alle restlichen Teilworte bezeichnet, welche nicht bereits in den Wortlisten stehen oder Akronyme sind. Außerdem werden Abkürzungen nur zu Worten aus den generierten Wortlisten normalisiert, das heißt, dass eine Abkürzung, dessen ursprüngliches Wort nie im Quelltext verwendet wird, nicht korrekt normalisiert werden kann.
Teilworte müssen zunächst drei Kriterien erfüllen, damit sie als Abkürzung eines Wortes aus der Wortliste gelten und sie mit einer hohen Korrektheit normalisiert werden können:

1. Die Anfangsbuchstaben des Teilwortes und des Wortes, zu dem es normalisiert werden soll, müssen übereinstimmen. Die gesamte Normalisierung wurde einmal ohne dieses Kriterium durchgeführt, und alle erkannten Normalisierungen, bei denen die Anfangsbuchstaben unterschiedlich waren, waren inkorrekt.
2. Alle Buchstaben des Teilwortes müssen in dem Wort, zu dem es normalisiert werden soll, enthalten sein. Denn es ergibt keinen Sinn andere Buchstaben in einer Abkürzung zu nutzen als diejenigen, welche in dem ursprünglichen Wort enthalten sind.
3. Das Teilwort muss eine Mindestwortlänge von drei haben. Denn bei einer Länge von lediglich zwei Buchstaben trifft die Abkürzung auf zu viele Worte zu, was eine hohe Fehlerrate zur Folge hätte.

Nach diesen Kriterien ist es trotzdem oft der Fall, dass eine Abkürzung auf mehrere Worte normalisiert werden kann. Daher wird ein Auswahlkriterium erstellt, welches sich berechnen lässt und nur eine Möglichkeit liefert. Die Abkürzung *conn* ist aus dem Quellcode von Apache Tomcat hervorgegangen und unter den möglichen Normalisierungen, welche die zuvor genannten Kriterien erfüllen, sind einige im Beispiel 6.1 aufgeführt.

Beispiel 6.1: Normalisierung der Abkürzung conn

conn	→	container
conn	→	connection
conn	→	content

Intuitiv würde *connection* für die richtige Normalisierung gehalten werden, was auch, wie es dem Kontext des Quellcodes entnommen werden kann, korrekt ist. Das liegt daran, dass bei *connection* alle vier Buchstaben aus *conn* direkt zusammenhängen. Also kann einfach die Länge der längsten direkt zusammenhängenden Buchstabenkette als Vergleichswert genommen werden. Jedoch wird ein aussagekräftigerer Wert benutzt, welcher im Folgenden als Ähnlichkeitsmaß bezeichnet wird, und der berechnet wird, indem die Länge jeder einzelnen zusammenhängenden Buchstabenkette der Abkürzung in der Normalisierung quadriert und aufaddiert wird. Das Ähnlichkeitsmaß wäre in diesem Beispiel bei *connection* 16 ($=4^2$) und sowohl bei *container* als auch *content* 10 ($=3^2+1^2$). Durch diese Berechnung werden auch zwei Normalisierungen unterschieden, welche als längste gemeinsame Wortkette die gleiche Anzahl haben, jedoch als zweilängste eine unterschiedliche Länge.
Es kann jedoch immer noch auftreten, dass zwei oder mehr Worte nach dieser Berechnung den gleichen Wert erhalten, wie im Beispiel 6.2. Die korrekte Normalisierung ist in diesem Fall *count*, jedoch wird durch das Ähnlichkeitsmaß lediglich die Normalisierung *convert* ausgeschlossen, da sie ein Ähnlichkeitsmaß von 3 hat. Die Normalisierungen *current* und *count* haben jedoch beide ein Ähnlichkeitsmaß von 5.

Beispiel 6.2: Normalisierung der Abkürzung cnt

			Ähnlichkeitsmaß
cnt	→	current	$1^2 + 2^2 = 5$
cnt	→	count	$1^2 + 2^2 = 5$
cnt	→	convert	$1^2 + 1^2 + 1^2 = 3$

Um jetzt eine Normalisierung auszuwählen, wird eine zusätzliche Priorität berechnet, welche das Verhältnis der Länge der Abkürzung zu der Länge der Normalisierung ist, wobei die Normalisierung mit der höchsten Priorität ausgewählt wird. Das hat zur Folge, dass kürzere Normalisierungen bevorzugt werden. Die Begründung dafür ist, dass es wahrscheinlicher ist, dass eine Buchstabenfolge in einem langen Wort auftritt. Das heißt, wenn die gleiche Buchstabenfolge in einem kürzeren Wort auftritt, ist es unwahrscheinlicher, dass es ein Zufall ist. In diesem Beispiel ist die Priorität von *current* 42% (3/7) und von *count* 60% (3/5), wodurch *count* hier korrekter Weise als Normalisierung von *cnt* bevorzugt wird.
Sollten zwei Normalisierungen jedoch sowohl das gleiche Ähnlichkeitsmaß, als auch die gleiche Priorität haben, wie im Beispiel 6.3, dann wird lediglich der erste Treffer genutzt und der zweite ignoriert. Denn wenn sich zwei Normalisierungen dermaßen ähnlich sind, ist es meistens nicht einmal manuell möglich die richtige auszuwählen, ohne den Kontext der Abkürzung zu kennen.

Beispiel 6.3: Normalisierung der Abkürzung str

			Ähnlichkeitsmaß	**Priorität**
str	→	string	$3^2 = 9$	3/6 = 50%
str	→	stream	$3^2 = 9$	3/6 = 50%

6.4 Ansatz der lokalen Normalisierung

Bisher wurde davon ausgegangen, dass alle Normalisierungen global (für alle Klassen in dem Projekt) gültig sind. Jedoch könnte es sein, dass eine Abkürzung in zwei Klassen zwei vollkommen unterschiedliche Bedeutungen hat, oder auch bereits in zwei Methoden innerhalb einer Klasse. Daher wird die Normalisierung sowohl von Abkürzungen, als auch von Akronymen zusätzlich einmal lokal für jede Klasse und lokal für jede Methode durchgeführt. Bei den Akronymen entsteht dadurch lediglich der Unterschied, dass sie nur noch lokal gültig sind, und somit gleiche Akronyme nicht global "überschrieben" werden, die restlichen Abkürzungen allerdings können durch dieses Vorgehen zusätzlich seltener normalisiert werden, da die Wortlisten lediglich aus Wörtern der Klasse bzw. der Methode bestehen. Die dadurch entstehenden Normalisierungen werden jeweils einmal für das Verfahren auf Klassenebene, als auch für das Verfahren auf Methodenebene gespeichert. Zusätzlich wird zu jeder Abkürzung das Paket, die Klasse und für die Normalisierung auf Methodenebene auch der Methodenname, in denen sie auftreten gespeichert.

6.5 Zusätzliches Nutzen der Kommentare

In Kapitel 4 wurde bereits darauf hingewiesen, dass die genutzte Grammatik keine Kommentare parst und somit alle Informationen aus den Kommentaren verloren gehen. Um diese Informationen trotzdem nutzen zu können, werden beim generieren des Wortschatzes zusätzlich alle Klassen Zeile für Zeile ausgelesen. Jede Zeile die mit "//" oder "/*" beginnt bzw. darauf folgende Zeilen die mit einem "*" beginnen, sind Kommentare und werden somit zunächst in die einzelnen Worte getrennt. Da dies ein Fließtext ist, werden die Worte einfach an allen Leerzeichen getrennt. Durch dieses Verfahren gehen zwar Kommentare verloren welche innerhalb des Quelltextes stehen, jedoch sollte dies den kleinsten Anteil ausmachen, was es vernachlässigbar macht. Die getrennten Worte werden mit WordNet geprüft und je nachdem, ob sie Nomen, Adjektive oder Verben sind in die entsprechenden Wortlisten eingefügt. Die darauf folgende Normalisierung der Abkürzungen wird genauso durchgeführt wie zuvor und alle Normalisierungen in einer Tabelle gespeichert. Das gleiche Verfahren wird auch ohne das Nutzen der Kommentare durchgeführt und die Normalisierungen in einer weiteren Tabelle gespeichert. Dadurch kann geprüft werden, ob durch die Kommentare weitere Informationen vorhanden waren, und somit mehr Abkürzungen korrekt normalisiert werden konnten.

6.6 Evaluation

Um die Normalisierung bewerten zu können, wird die Liste mit den Abkürzungen und ihren Normalisierungen gespeichert, und um eventuell den Kontext von nicht eindeutigen Normalisierungen einzusehen, wird für die Evaluation zusätzlich zu jeder Normalisierung deren Klasse gespeichert, in der sie auftritt. In den folgenden Abschnitten werden die zuvor erläuterten Ansätze einzeln evaluiert bevor in Kapitel 6.6.4 das Verfahren beurteilt wird, welches sich als das bestmögliche herausgestellt hat.

6.6.1 Lokale Normalisierung

Die größte Auffälligkeit beim Prüfen der lokalen Normalisierungen ist, dass sobald eine Abkürzung (auch Akronyme) innerhalb einer Klasse in mehreren Methoden normalisiert wird, sie auch in allen Methoden zum gleichen Wort normalisiert wird. Somit sind alle Abkürzungen zumindest innerhalb einer Klasse gültig. In der Tabelle für einzelne Klassen hingegen gibt es unterschiedliche Normalisierungen für einzelne Abkürzungen zwischen verschiedenen Klassen. Um diese Einträge besser beurteilen zu können und zu wissen, welche korrekt sind, wird für jeden Eintrag der Kontext innerhalb der Klasse betrachtet. Bei allen Fällen in denen sich Akronyme wiedersprechen, sind beide lokal in ihrer Klasse korrekt. Dies sind immerhin 14 Fälle, was bei insgesamt 122 verschiedenen Akronymen innerhalb des Testprojektes ein großer Anteil ist. Somit dürfen Akronyme nur innerhalb

einer Klasse gelten, weshalb für jede Klasse, falls sie ein Akronym besitzt, eine eigene Akronymliste erstellt wird und all diese Listen mit dem Paket und dem Namen der Klasse als Schlüssel in einer weiteren Liste gespeichert werden. So kann für die Aufbereitung der Analyse jedes Akronym einfach normalisiert werden, indem lediglich der Paket- und Klassenname, sowie das Akronym übergeben wird.

Bei fast allen Fällen, in denen sich die konventionellen Abkürzungen widersprochen haben, gab es jedoch nur eine korrekte Normalisierung. Dass heißt, in der Regel hat den Klassen das korrekte Wort in Ihrem Wortschatz gefehlt, wodurch ein falsches für die Normalisierung ausgewählt wurde. Es gab auch Fälle in denen keine Normalisierung korrekt war und auch einen Fall in dem sich zwei korrekte Normalisierungen widersprochen haben. Im Beispiel 6.4 ist dieser Fall aufgeführt, da dies jedoch nur ein Fall von insgesamt 170 verschiedenen Abkürzungen ist, kann angenommen werden, dass konventionelle Abkürzungen global gültig sind. Denn dadurch kann bei allen Abkürzungen der gesamte Wortschatz des Projektes genutzt werden, wodurch viel weniger Abkürzungen falsch normalisiert werden.

Beispiel 6.4: Normalisierung der Abkürzung att

att	→	attribute
att	→	attachment

6.6.2 Informationsgehalt der Kommentare

Im Vergleich zur bisherigen Normalisierung werden bei der Normalisierung mit dem Wortschatz der Kommentare, tatsächlich mehr Abkürzungen normalisiert, jedoch werden in etwa genauso viele Abkürzungen korrekt normalisiert wie ohne die Kommentare, wodurch der prozentuale Anteil der korrekten Normalisierungen gesunken ist. Daraus geht hervor, dass in dem Quelltext bereits alle relevanten Informationen enthalten sind, welche in den Kommentaren stehen. Denn die Kommentare können zwar einen großen Anteil der natürlichsprachigen Informationen des Quelltextes ausmachen, jedoch bestehen diese, da sie in einem Fließtext geschrieben sind, zu einem Großteil aus Stoppworten mit wenig Informationen. Hinzu kommt, dass sich die Kommentare in der Regel auf Methoden oder Variablen beziehen, und somit sich die Informationen mit denen, die bereits in den Bezeichnern stehen stark überschneiden und dadurch überflüssig werden. Die Wahrscheinlichkeit, dass die anderen Worte in den Kommentaren gerade den Namen eines abgekürzten Variablennamens beschreiben, welcher im restlichen Quelltext nie ausgeschrieben wird ist relativ gering.
Dem steht zusätzlich die "Verunreinigung" des Wortschatzes gegenüber, welcher durch die Kommentare mit weniger relevanten Worten erweitert wird, wodurch Abkürzungen, denen kein korrektes Wort zugeordnet werden konnte, zu den falschen Worten normalisiert werden. Somit hat die Nutzung der Kommentare ohne weitere Filterung eine negative Auswirkung auf die Normalisierung, weshalb darauf verzichtet wird.

6.6.3 Fehlerhafte Normalisierungen

Trotz des eigens generierten Wortschatzes und der Stoppwortliste besteht ein nicht vernachlässigbarer Anteil der Normalisierungen aus bereits korrekten Worten, welche trotzdem fälschlicherweise normalisiert werden. Dies sind jedoch keine beliebigen Worte, sondern einerseits Akronyme (z.B. HTML, URL), welche nicht als Worte erkannt werden, da die von WordNet gefundenen Akronyme nicht in die Wortlisten aufgenommen werden, da es ansonsten zu viele falsche Treffer gibt. Normalisiert werden sie aber auch nicht, da solche Akronyme bereits im normalen Sprachgebrauch anstelle der abgekürzten Worte verwendet werden. Andererseits sind ein Großteil dieser Worte Dateiendungen (z.B. XML, XSP), welche von WordNet gar nicht erkannt werden. Hinzu kommen Eigennamen des Projektes (z.B. Apache), welche von einem Wörterbuch ebenfalls nicht erkannt werden können. Alle diese Worte lassen sich unter dem Begriff domänenspezifische Worte zusammenfassen, was dazu führt, dass eine weitere Wortliste für eine gute Normalisierung benötigt

wird. Diese Domänenwortliste muss vor der Normalisierung an das zu analysierende Projekt angepasst werden. In dem Fall von Apache Tomcat ist die Domäne der Webserver, somit werden alle Akronyme, Dateiendungen und Eigennamen, welche auf diese Domäne zutreffen gesammelt und in einer Liste zusammengefasst. Durch diese zusätzliche Filterung kann die Normalisierung noch einmal erheblich verbessert werden, denn der Anteil der korrekten Normalisierungen zu allen Normalisierungen ist dadurch um ca. 10% gestiegen.

6.6.4 Resultierendes Verfahren

Die Normalisierung der Akronyme muss an dieser Stelle nicht mehr ausgewertet werden, denn im Unterkapitel 6.6.1 konnte nicht nur festgestellt werden, dass Akronyme mit einer Mindestlänge von zwei Zeichen immer lokal pro Klasse gültig sind, sondern auch, dass gefundene Akronyme immer korrekt normalisiert werden. Somit bezieht sich die folgenden Auswertung lediglich auf konventionelle Abkürzungen und nicht auf Akronyme.

Das erarbeitete Normalisierungsverfahren besteht letztendlich aus drei Durchläufen. Im ersten Durchlauf der Syntaxbäume werden die Wortlisten erstellt, das sind zuerst die allgemeine Wortliste, welche Nomen und Adjektive enthält, des Weiteren die Verbliste, welche ausschließlich aus Verben besteht. Zu diesen Wortlisten werden sowohl die Stoppwortliste, als auch die manuell erstellte Domänenwortliste hinzugefügt. Im zweiten Durchlauf werden alle Teilworte der Bezeichner, welche nicht bereits in einer der Listen vorkommen unter den in 6.3 erläuterten Kriterien normalisiert. Variablen und Parameter werden zusätzlich darauf geprüft, ob sie ein Akronym zu Ihrem Datentyp sind. Die Normalisierungen von Abkürzungen werden in einer Liste gespeichert, und die Normalisierungen von Akronymen in einer separaten Liste welche nur für die jeweilige Klasse gilt. Im dritten Durchlauf werden alle Teilworte der Bezeichner darauf geprüft, ob sie in einer der beiden Listen vorkommen, falls sie das tun, wird Ihre Normalisierung in einer CSV-Datei gespeichert, ansonsten wird das ursprüngliche Teilwort gespeichert.

Die Liste mit den Abkürzungen werden für die letztendliche Auswertung manuell geprüft, wobei lediglich 265 unterschiedliche Normalisierungen aus dem gesamten Projekt hervorgegangen sind. Diese treten jedoch insgesamt 4548 mal in einem Klassennamen, Methodennamen oder der Initialisierung einer Variable oder eines Parameters auf. Von diesen 265 Normalisierungen sind lediglich 170 korrekt, was ca. 64% entspricht. Das klingt zunächst weniger als erwartet, jedoch müssen die korrekten Normalisierungen auf die Anzahl ihres Auftretens bezogen werden. Und da die korrekten Normalisierungen insgesamt 3770 mal auftreten, machen sie somit 82,89% aller Normalisierungen aus, was ein annehmbarer Wert für die Analyse ist.
Jedoch dürfen nicht nur die normalisierten Abkürzungen betrachtet werden, sondern auch diejenigen, für welche kein Wort für die Normalisierung gefunden wurde. Außerdem ist der Anteil der Worte relevant, welche gar nicht erst normalisiert werden mussten, da es bereits Worte waren, welche in der Datenbank des Wörterbuchs gefunden werden konnten.
Zunächst werden lediglich Teilworte betrachtet, welche eine Mindestwortlänge von drei aufweisen, dass sind im gesamten Testprojekt 39584 Teilworte. Von dieser Anzahl werden bereits 77,79% als Worte erkannt und müssen nicht normalisiert werden (Tabelle 6.1, Abbildung 6.1). Von den 22,21% der Teilworte, welche normalisiert werden müssen, wird für 51,72% eine Normalisierung gefunden (Abbildung 6.2). Diese Zahl ist jedoch größer als sie zunächst klingt, denn von allen angeblichen Abkürzungen sind noch immer 30,75% normale Worte, welche nicht erkannt wurden, da es Verben sind, die in einem Klassen oder Variablennamen gestanden haben, und deren Wortliste keine Verben enthält, da es gegen die Code-Konventionen verstößt. Von diesen Verben wird allerdings nur ein sehr geringer Teil normalisiert (etwa 1% der Verben), und zusätzlich werden die Verben oftmals zu inhaltlich identischen Nomen normalisiert (z.B. *accept* zu *acceptor*), was diesen Umstand vernachlässigbar macht.

Alle Teilworte	39584
Erkannte Worte	30791
Abkürzungen	8793

Tabelle 6.1: Anteil der erkannten Worte

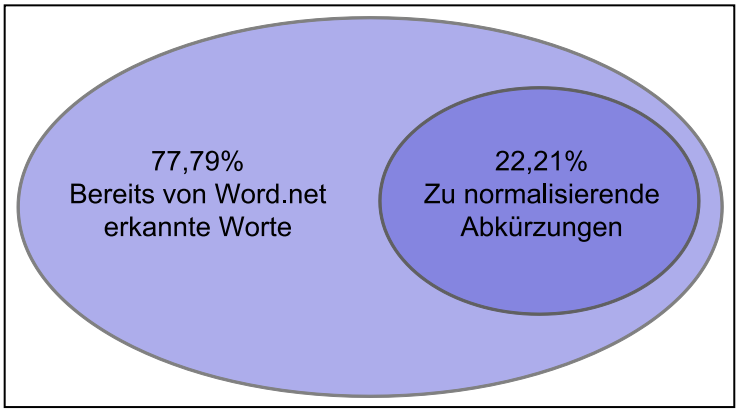

Abbildung 6.1: Anteil der erkannten Worte

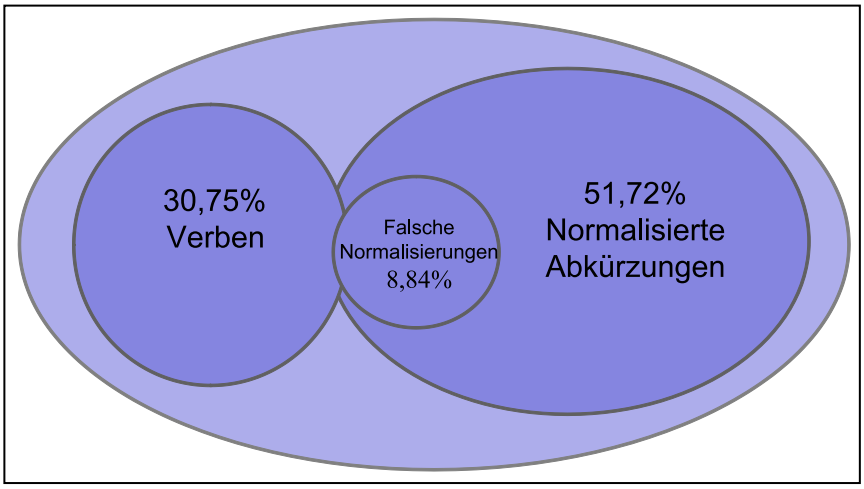

Abbildung 6.2: Anteil normalisierter Abkürzungen

	Anzahl	Prozent	Ohne nicht normalisierte Verben
Alle Abkürzungen	8793	100%	6330
Verben	2704	30,75%	-
Alle Normalisierungen	4548	51,72%	71,85%
Korrekte Normalisierungen	3770	42,88%	59,56%

Tabelle 6.2: Anteil korrekter Normalisierungen

Wenn nun also die nicht normalisierten Verben aus der Bewertung rausgenommen werden, werden von den Abkürzungen, die normalisiert werden müssten 71,85% normalisiert (Tabelle 6.2). Dementsprechend ist der Anteil der falschen Normalisierungen zu allen Abkürzungen, die normalisiert werden müssen 12,29% (Abbildung 6.3). Daraus ergibt sich - wie bereits erwähnt - ein Anteil von den korrekten Normalisierungen zu allen Normalisierungen von 82,89%.

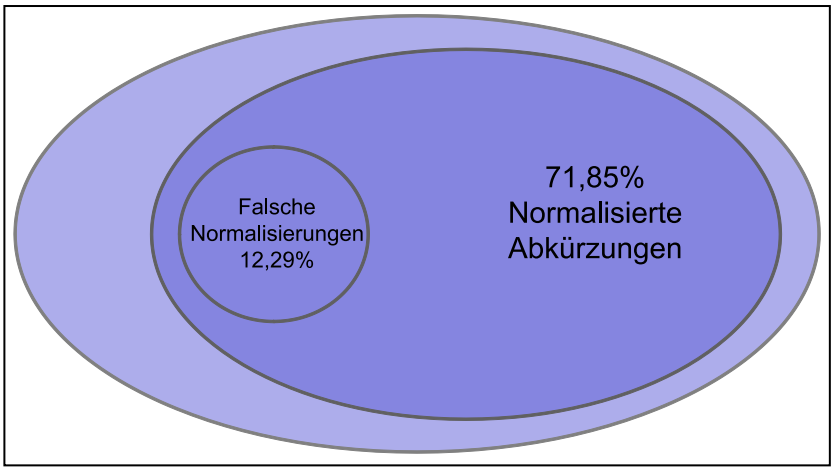

Abbildung 6.3: Anteil normalisierter Abkürzungen ohne Verben

Somit werden von allen Teilworten (mit einer Mindestlänge von drei) lediglich 6,47%[8] nicht oder falsch normalisiert, die restlichen 93,53% sind entweder bereits Worte oder werden durch die Normalisierung erweitert. Hinzu kommt, dass ein weiterer - wenn auch geringer - Teil von Akronymen normalisiert wird, welche auch Abkürzungen der Länge zwei umfassen.

Auf dieser Aufbereitung der natürlichsprachigen Informationen lässt sich im folgenden Kapitel eine fundierte Analyse durchführen. Denn von allen natürlichsprachigen Informationen wird ein Großteil korrekt aufbereitet, und die restlichen 6,47% können trotzdem analysiert werden, jedoch können diese nicht normalisierten Abkürzungen nicht mit anderen Abkürzungen in Verbindung gebracht werden, welche für das gleiche Wort stehen. Sollte für ein Wort innerhalb des Quellcodes jedoch lediglich eine Abkürzung genutzt werden, ist es kein Verlust wenn diese nicht normalisiert wird.

8 $\frac{Falsch\ oder\ nicht\ normalisierte\ Worte}{Alle\ Teilworte} = \frac{Alle\ Abkürzungen(ohne\ Verben) - korrekte\ Normalisierungen}{Alle\ Teilworte}$

$$= \frac{6330 - 3770}{39584} \approx 0{,}0647 \triangleq 6{,}47\%$$

7 Sicherheitsanalyse

Nach der Aufbereitung des Quelltextes können im letzten Schritt die natürlichsprachigen Informationen analysiert werden. Diese Analyse wird mit einem Programm zur Datenanalyse namens RapidMiner[9] durchgeführt. In diesem Programm wird ein Prozess erstellt, welcher unter 7.2 näher erläutert wird. Innerhalb dieses Prozesses wird einer der in 2.2 beschriebenen Klassifikatoren eingelernt und anschließend für die Analyse genutzt. Das Ziel der Validierung ist, verschiedene Klassifikatoren und Analyseverfahren mit denen in 2.2.1 erläuterten Formeln zu bewerten und zu vergleichen, um letztendlich zeigen zu können, ob natürlichsprachige Informationen zur Identifikation von Sicherheitslücken genutzt werden können.

7.1 Validierungsprozess

Der für die Analyse in RapidMiner erstellte Prozess ist in der kompletten Übersicht im Anhang auf Abbildung B.2 zu sehen, er verläuft von links nach rechts bzw. in der abgebildeten Form von unten nach oben. Dabei kann der Prozess in zwei Teilprozesse unterteilt werden, den oberen, welcher die Trainingsmenge analysiert und durch den Klassifikator in der Trainingsphase ein Modell erstellt, und den unteren, welcher das Modell in der Testphase auf die Testmenge anwendet und validiert. Zur besseren Veranschaulichung wird der Prozess an den drei Prozessausschnitten in Abbildung 7.1 bis Abbildung 7.3 beschrieben. In beiden Teilprozessen wird zunächst der Datensatz der Trainings- bzw. der Testmenge aus der Datenbank gelesen und in Dokumente umgewandelt (Abbildung 7.1, Blöcke: Retrieve CWE, Data to Documents). Danach wird für die Worte der Datensätze eine Relevanz berechnet und jedem Eintrag ein Wortvektor mit den entsprechenden Werten zugewiesen (Abbildung 7.1, Blöcke: Prozess Documents). Dabei wird bei dem Bearbeiten der Trainingsmenge eine Wortliste erstellt, welche alle Worte beinhaltet, die in der Trainingsmenge enthalten sind. Denn alle Worte die nicht in der Trainingsmenge enthalten sind, können in der Testmenge für die Analyse nicht genutzt werden, da keine Aussage über diese Einträge getroffen werden kann. Daher wird bei der Berechnung der Werte für die Testmenge diese Liste übergeben, um lediglich Werte für die Worte zu berechnen, über welche auch eine Aussage getroffen werden kann (Abbildung 7.1, Abbildung 7.2, Blöcke: Retrieve WordList, Store WordList).

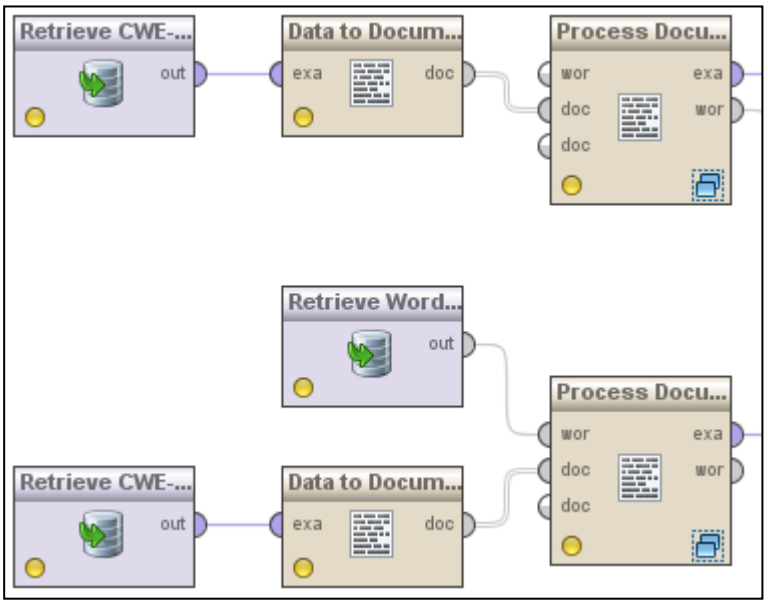

Abbildung 7.1: Prozessausschnitt 1

[9] http://rapidminer.com/, Version: 5.3.013

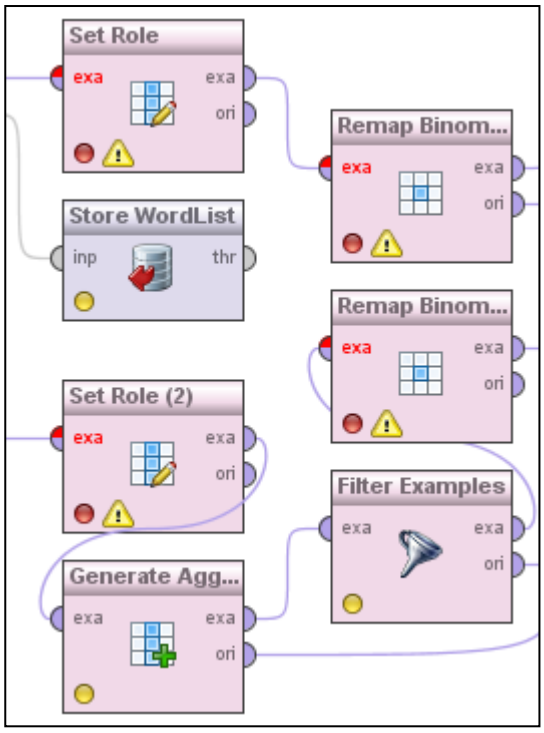

Abbildung 7.2: Prozessausschnitt 2

Im nächsten Schritt wird die Markierung, ob ein Eintrag sicherheitsrelevant ist oder nicht (sec/nonsec), welche zuvor manuell gesetzt wird, auf einen positiven bzw. negativen Wert abgebildet, um in der darauf folgenden Validierung die Ergebnisse auswerten zu können (Abbildung 7.2, Blöcke: Set Role, Remap Binomial). Im unteren Pfad des Prozesses werden zusätzlich alle Einträge gefiltert, welche keine Werte aufweisen, da sie keine Worte der Trainingsmenge enthalten. Denn über diese Einträge kann keine Aussage getroffen werden (Abbildung 7.2, Blöcke: Generate Aggregation, Filter Examples). Der zweite Filter in Abbildung 7.3 dient lediglich der Auswertung, denn dieser ist invers zu dem vorherigen Filter, das heißt er gibt lediglich diejenigen Einträge weiter, über welche keine Aussage getroffen werden kann. Dadurch kann direkt die Anzahl und Art der gefilterten Einträge abgelesen werden. Danach folgt die Validierung, welche in der Trainingsmenge durch den Klassifikator ein Modell erstellt (Abbildung 7.3, Blöcke: Validation). Der Teilprozess innerhalb der Validierung kann ebenfalls geändert werden und ist in Abbildung 7.4 dargestellt. An dieser Stelle kann auch der Klassifikator ausgetauscht werden (Block: k-NN). Das erstellte Modell wird zusätzlich auf der Trainingsmenge ausgeführt, um zu prüfen, ob dieses die Einträge der Trainingsmenge korrekt einschätzt. Die Validierung der Testmenge hingegen nutzt lediglich das Modell, welches mit der Trainingsmenge erstellt wurde, und analysiert die Einträge (Abbildung 7.3, Blöcke: Validation(2)). Dieser Teilprozess kann ebenfalls geändert werden und ist in Abbildung 7.5 dargestellt. Bei beiden Validierungen werden Tabellen entsprechend der in Kapitel 2.1 erstellt, aus welchen sich nun die Trefferquote, Präzision etc. ablesen bzw. berechnen lassen. Die zusätzliche Anwendung des Modells auf die Wortvektoren dient dem Verständnis der Analyse, denn dadurch werden alle Einträge mit ihrer Markierung und der Vorhersage des Modells angezeigt (Abbildung 7.3, Blöcke: Apply Model(4), Apply Model(5)). Dadurch lassen sich beispielsweise falsch-negativ Einträge betrachten und die Ursache dafür ermitteln.

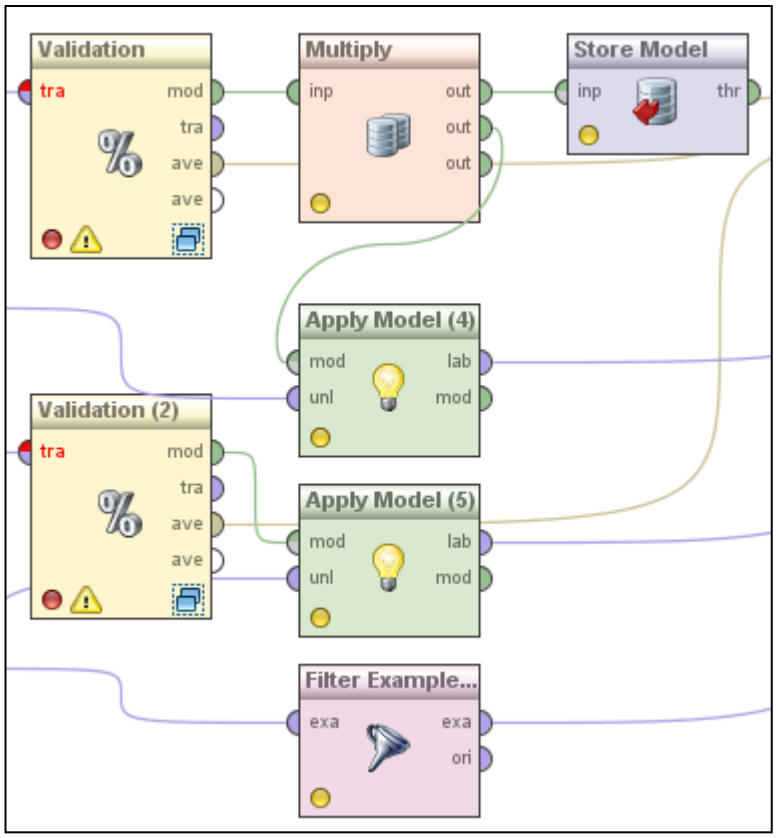

Abbildung 7.3: Prozessausschnitt 3

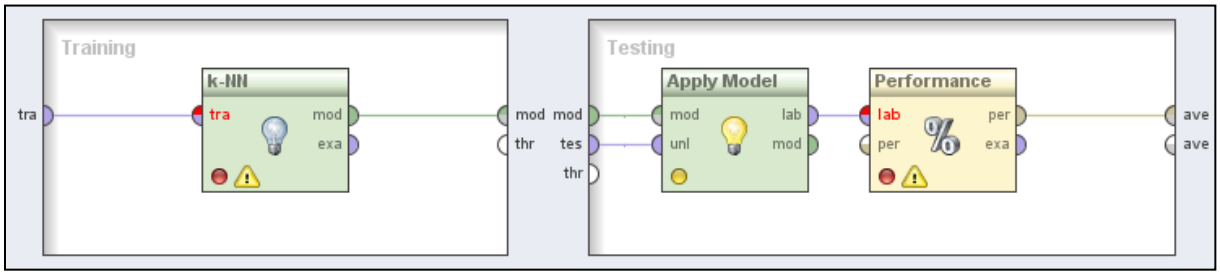

Abbildung 7.4: Detaillierter Validierungsblöck der Trainingsmenge

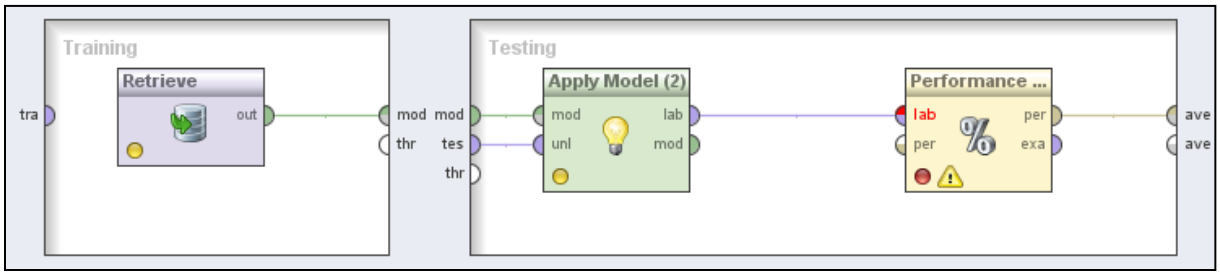

Abbildung 7.5: Detaillierter Validierungsblock der Testmenge

7.2 Evaluationsgrundlage

Um einen Klassifikator auf Worte sensibilisieren zu können, welche sicherheitsrelevant sind, werden dokumentierte Schwachstellen benötigt. Denn die Vermutung liegt darin, dass im Quelltext in sicherheitsrelevanten Methoden der gleiche Wortschatz bzw. die gleichen Schlüsselworte als Bezeichner genutzt werden.

Mir wurde, vom Fachgebiet für Software Engineering der Leibniz Universität Hannover, eine Liste zur Verfügung gestellt, welche Schwachstellen des Projektes Apache Tomcat dokumentiert. Erstellt wurde diese Liste aus CVE-Einträgen (Common Vulnerabilities and Exposures), aus Kommentare innerhalb des SVN von Apache Tomcat und durch Bugzilla-Einträge. Enthalten sind in der Liste zu jeder dokumentierten Schwachstellen folgende Werte: Die CVE-ID, das Veröffentlichungsdatum, die CVSS Bewertung (Common Vulnerability Scoring System), die betroffene Version von Apache Tomcat, die Überarbeitungsnummer (engl. revision) in der die Schwachstelle behoben wurde, verwandte Schwachstellen, die betroffenen Pakete, die Art der Auswirkung, die zugehörige CWE-ID und die Art der Schwachstelle.

Relevant sind für mich lediglich Einträge zur Version 6, da die Analyse an dieser Version durchgeführt wird. Des Weiteren wird die jeweilige Überarbeitungsnummer genutzt, über welche es möglich ist die Subversion (engl. subversion) der betroffenen Klassen dieser Schwachstellen im Zustand vor der Überarbeitung abzurufen. Diese Klasse, welche die dokumentierten Schwachstellen noch beinhalten, können für die Analyse genutzt werden.

7.3 Validierungen

Die Klasse, welche die dokumentierten Sicherheitslücken enthalten, werden in das aktuelle Projekt von Apache Tomcat eingefügt, und die, in den Kapiteln 3 bis 6 erläuterte Aufbereitung durchgeführt, und anschließend die aufbereiteten Worte der betroffenen Klassen in einer CSV-Datei gespeichert. Dabei werden alle Worte einer Methode in einer Zeile durch Leerzeichen getrennt gespeichert, und in der ersten Spalte wird der entsprechende Methodenname gespeichert. Im SVN kann zu der entsprechenden Überarbeitung eingesehen werden, welche Methoden von der Schwachstelle betroffen waren, und dementsprechend die Methodennamen in der erstellten CSV-Datei mit der Markierung *sec* (sicherheitsrelevant bzw. positiv) oder *nonsec* (nicht sicherheitsrelevant bzw. negativ) überschreiben werden. Des Weiteren kann der mir zur Verfügung gestellten Liste entnommen werden, unter welchem CWE diese Sicherheitslücke eingeordnet wird. Das macht es mir möglich, die Trainings- und Testmengen in einzelne Arten von Schwachstellen einzuteilen, wodurch der genutzte Wortschatz und die Schlüsselworte noch spezifischer und effektiver auszuwerten sind. Als letztes werden aus den erstellten CSV-Dateien zusätzlich Worte gefiltert, welche aus nur einem Zeichen bestehen, da diese in dem Aufbereitungsverfahren nicht normalisiert werden. Außerdem werden primitive Datentypen gefiltert, da diese zu häufig auftreten und somit eine gewisse Ungenauigkeit in die Analyse bringen könnten. Aus demselben Grund werden häufige Präfixe von Methodennamen gefiltert (z.B.: get, set, do). Ebenso werden Worte, die in einer Methode mehrfach auftreten auf einen Eintrag reduziert.

Die entstandenen CSV-Dateien werden für jedes CWE in eine Trainings- und eine Testmenge unterteilt, wobei die Trainingsmenge so gewählt wird, dass sie ca. ein Drittel der Gesamtmenge ausmacht, und möglichst aus den zeitlich frühsten Einträgen für dieses CWE besteht. Die entstandenen Mengen sind in Tabelle 7.1 aufgeführt. Die Methoden einer einzelnen Klasse wurden niemals in zwei Mengen aufgeteilt, wodurch es schwierig war, bei weniger Einträgen, wie beim CWE-287, eine angemessene Aufteilung durchzuführen. Zu diesem CWE sind lediglich vier Klassen mit Schwachstellen dokumentiert, was die Auswahl sehr einschränkt, wodurch die Trainingsmenge letztendlich größer gewählt wurde als die Testmenge. Da bei denjenigen CWE mit einer größeren Menge die Klassifikatoren mehr eingelernt werden, können an dieser Stelle auch bessere Ergebnisse der Analyse erwarten werden, daher sind in allen Tabellen die CWE nach der Anzahl - von oben nach unten abfallend - aufgelistet. Weiterhin ist auffällig, dass die Anzahl sicherheitsrelevanter Methoden

zu der Anzahl aller Methoden sehr gering ist. Das ist jedoch leicht zu erklären, denn die betroffenen Klassen bestehen aus durchschnittlich 40 Methoden und eine Sicherheitslücke betrifft durchschnittlich nur zwei bis drei Methoden. Da alle nicht betroffenen Methoden als nicht sicherheitsrelevant angesehen werden, ist dieser Anteil sehr hoch.

	In der Trainingsmenge			In der Testmenge		
CWE	**Einträge**	**sec-Einträge**	**%-Anteil sec**	**Einträge**	**sec-Einträge**	**%-Anteil sec**
20	357	10	2,8	588	18	3,06
200	230	16	6,96	447	22	4,92
22	121	11	9,09	442	22	4,98
264	187	12	6,42	262	15	5,73
399	145	12	8,28	282	26	9,22
79	76	5	6,58	146	9	6,16
287	63	6	9,52	36	6	16,67

Tabelle 7.1: Erstellte Trainings- und Testmenge

7.3.1 Inklusion durch sicherheitsrelevante Worte

Ein ähnliches Verfahren der Sicherheitsanalyse, welches auf natürlichsprachige Informationen zurückgreift, wurde bisher nicht durchgeführt. Daher wird als Vergleichswert das einfachste Analyseverfahren, welches auf der Basis der erstellten Daten durchgeführt werden kann genutzt. Dieses Verfahren ist eine simple Inklusion, welche in Java implementiert wird.
Dabei wird zunächst die gesamte Trainingsmenge ausgelesen und alle Worte, welche in einer als sicherheitsrelevant markierten Methode stehen, werden in einer Liste gespeichert. Danach wird die gesamte Testmenge ausgelesen und jede Methode, welche eines der Worte aus der Liste enthält, wird als sicherheitsrelevant markiert. Je nachdem, ob die Methode auch sicherheitsrelevant war oder nicht, oder nicht markiert wurde und trotzdem sicherheitsrelevant war, wird die Anzahl der Richtig-Positiven, Falsch-Positiven, Richtig-Negativen oder Falsch-Negativen Einträge erhöht. Danach wird mit diesen Werten wie in Kapitel 2.1 beschrieben die Trefferquote etc. berechnet. Dieses Analyseverfahren wurde für alle CWE durchgeführt; die Ergebnisse sind in Tabelle 7.2 aufgeführt. Die Werte in allen Tabellen dieses Kapitels sind auf zwei Nachkommastellen gerundet, das heißt die Falsch-Negativ-Rate ist nur tatsächlich gleich 0, wenn die Trefferquote gleich eins ist. Ebenso ist die Falsch-Positiv-Rate nur dann gleich null, wenn die Präzision gleich eins ist.
Als erstes fällt bei den Ergebnissen die hohe Trefferquote auf. Daran lässt sich ablesen, dass die sicherheitsrelevanten Methoden aus den Testmengen wenigstens einmal die gleichen Worte enthalten, wie die sicherheitsrelevanten Methoden der entsprechenden Trainingsmenge.

CWE	Trefferquote	Präzision	F_1-Maß	Falsch-Positiv-Rate	Falsch-Negativ-Rate
20	1	0,04	**0,08**	0,81	0
200	1	0,05	**0,1**	0,81	0
22	0,95	0,05	**0,1**	0,91	0,05
264	0,93	0,06	**0,11**	0,74	0,07
399	0,85	0,1	**0,18**	0,67	0,15
79	0,89	0,08	**0,15**	0,81	0,11
287	1	0,15	**0,26**	0,97	0

Tabelle 7.2: Ergebnisse der Inklusion

Da jedoch jede andere Methode, welche auch nur einmal ein sicherheitsrelevantes Wort aus der Trainingsmenge nutzt, ebenfalls als positiv markiert wird, ist die Präzision sehr gering. Das heißt, der Anteil an Falsch-Positiven Vorhersagen ist sehr hoch, was auch in der Falsch-Positiv-Rate widergespiegelt wird. Somit ist dieses Analyseverfahren lediglich als geringe Vorauswahl nützlich, es müssen aber trotzdem noch sehr viele Einträge manuell überprüft werden. Dieser relativ geringe Nutzen wird auch durch das geringe F1-Maß widergespiegelt, welches durchschnittlich nur knapp über 0,1 liegt.

7.3.2 Informationsgehalt von Methodennamen und Datentypen

Bei der Inklusion wurde bei der Aufbereitung die Informationen von allen Bezeichnern innerhalb der Methoden und der Methodennamen selbst genutzt. Dabei könnte es bereits genügen, lediglich die Methodennamen für die Analyse zu nutzen. Andererseits könnte es der Fall sein, dass durch das hinzufügen aller Datentypen (bis auf die gefilterten primitiven Datentypen) die Analyse verbessert wird. Um diese Herangehensweisen zu vergleichen werden die zu analysierenden Trainings- und Testmengen in drei verschiedenen Arten erstellt.
Einmal wird der Datensatz lediglich mit den Methodennamen, einmal mit allen Bezeichner und einmal mit allen Bezeichnern und ihren Datentypen erstellt. Diese Datensätze werden mit dem unter 7.1 erläuterten Prozess evaluiert und dabei alle Klassifikatoren, welche unter 2.2 erläutert wurden, in ihren Analysewerten verglichen; die Ergebnisse sind in Tabelle 7.3 aufgeführt. Einzelne Werte wie Trefferquote, Präzision etc. sind zu jedem Klassifikator im Anhang in den Tabelle A.1 bis Tabelle A.4 aufgeführt. Die Abkürzungen *mD* bzw. *oD* bedeuten, dass die Analyse entweder mit oder ohne Datentypen durchgeführt wurde. Für diesen ersten Vergleich werden lediglich zwei CWE genutzt, zum Einen CWE-20, da es die meisten Einträge aufweist, und zum Anderen CWE-79, da es sehr wenig Einträge aufweist, aber dennoch ein angemessenes Verhältnis der Trainings- und Testmenge hat.

		F1-Maß des Klassifikators			
CWE	**Grundlage**	**k-NN**	**SVM**	**Entscheidungsbaum**	**Perzeptron**
20	*Bezeichner mD*	0,55	0,53	0,63	0,69
20	*Bezeichner oD*	0,56	0,75	0,53	**0,81**
20	*Methoden*	0	0,43	0,43	0,47
79	*Bezeichner mD*	0,35	0,23	0,32	0,32
79	*Bezeichner oD*	0,4	0,36	0,36	0,28
79	*Methoden*	0,57	0,57	0	0,57

Tabelle 7.3: Ergebnisse der verschiedenen Datensätze

Zunächst ist es auffällig, dass diese Werte bereits weit über denen der Inklusion liegen, und bei der Nutzung von Bezeichnern ohne Ihre Datentypen wird beim CWE-20 mit Hilfe des Klassifikators Perceptron ein F1-Maß von 0,81 erreicht, was bereits ein sehr guter Wert ist. Für das CWE-20 wird bei der Analyse - basierend auf den Methodennamen - in allen Fällen das schlechteste Ergebnis erzielt, und der Klassifikator k-NN erreicht nicht einmal einen Treffer. Für das CWE-79 erreicht diese Analyse die besten Werte, wenn der Entscheidungsbaum außen vor gelassen wird. Wird das Ergebnis jedoch genauer betrachtet, können die gefilterten Einträge der Testmenge aus Tabelle 7.4 entnommen werden, welche nicht beurteilt werden können, da sie keine Worte enthalten, die in der Trainingsmenge vorkommen. Die Analyse, welche lediglich Methodennamen nutzt, hat einen sehr geringen Wortumfang, wodurch die Anzahl der gefilterten Worte sehr groß ist. Beim CWE-79 werden mehr als ein Drittel der Einträge gefiltert, und dabei sind fast die Hälfte aller sicherheitsrelevanten Einträge enthalten, wodurch das Ergebnis aus Tabelle 7.3 viel schlechter ist, als es die Zahlen zunächst vermuten lassen.

		In der Testmenge		gefilterte Einträge	
CWE	**Grundlage**	**Einträge**	**sec-Einträge**	**nonsec**	**sec**
20	*Bezeichner mD*	588	18	12	0
20	*Bezeichner oD*	588	18	17	0
20	*Methoden*	588	18	112	1
79	*Bezeichner mD*	146	9	4	1
79	*Bezeichner oD*	146	9	2	1
79	*Methoden*	146	9	56	4

Tabelle 7.4: Anzahl gefilterte Einträge verschiedener Datensätze

Diese erste Analyse zeigt, dass eine Analyse basierend auf den Methodennamen viel zu wenig Grundlage besitzt. Ob eine Analyse mit der Nutzung der Datentypen effektiver ist oder nicht, kann durch diese Ergebnisse jedoch noch nicht eindeutig beantwortet werden. Daher werden im Folgenden noch einmal alle CWE, des zur Verfügung stehenden Datensatzes, sowohl mit der Nutzung von Datentypen, als auch ohne analysiert.

7.3.3 Analyse mit Nutzung der Datentypen

Die Ergebnisse der Validierung mit der Nutzung von Datentypen sind in der Tabelle 7.5 aufgeführt; die einzelnen Teilwerte sind wiederum im Anhang in den Tabelle A.5 bis Tabelle A.8 aufgeführt.

	F1-Maß des Klassifikators			
CWE	**k-NN**	**SVM**	**Entscheidungsbaum**	**Perzeptron**
20	0,55	0,53	0,63	0,69
200	0,36	0,31	0,37	0,39
22	0,32	0,41	0,67	0,5
264	0,32	0,29	0,32	0,44
399	0,34	0,24	0,56	0,46
79	0,35	0,23	0,32	0,32
287	0,5	0,5	0,4	0,72
Durchschnitt	0,39	0,36	0,47	0,5

Tabelle 7.5: Analyse mit der Nutzung von Datentypen

Der Klassifikator k-NN hat einen Durchschnittswert von 0,39, welcher lediglich von dem Ergebnis der Analyse des CWE-20 und CWE-287 abweicht. Alle anderen CWE liegen bei einem stabilen Wert von ca. 0,35. Dabei kann den Tabellen aus dem Anhang entnommen werden, dass die Trefferquote für diesen Klassifikator durchschnittlich bei 0,62 liegt, also werden durch diesen sehr einfachen Klassifikator mehr als die Hälfte der Sicherheitslücken erkannt, jedoch mit einer geringen Präzision. Wobei die Präzision zusätzlich im Verhältnis zum Anteil der sicherheitsrelevanten Methoden betrachtet werden sollte. Denn wenn eine Präzision von 33% erreicht wird, sind von allen als sicherheitsrelevant markierten Methoden lediglich ein Drittel tatsächliche Treffer. Das heißt, es müssen maximal dreimal so viele Methoden kontrolliert werden wie es Sicherheitseinträge gibt, und da in den mir zur Verfügung stehenden Schwachstellen lediglich 6% aller Methoden sicherheits-relevant sind, müssen auch nur maximal 18% der zu analysierenden Methoden geprüft werden. Die anderen Klassifikatoren liefern stark schwankende Ergebnisse, wobei die Analyse durch das Perzeptron fast immer ein besseres F1-Maß liefert, als bei der Analyse durch k-NN, jedoch ist die Analyse mit einem Perzeptron sehr viel aufwendiger.

In Tabelle 7.6 sind die gefilterten Einträge aufgeführt. Im Gegensatz zur Analyse auf der Basis der Methodennamen, werden bei diesem Verfahren verhältnismäßig wenig Einträge gefiltert, jedoch ist der Anteil mit durchschnittlich 6,7% immer noch relativ hoch. Der größte Verlust an dieser Stelle sind aber die sicherheitsrelevanten Methoden, welche gefiltert wurden, da keine Informationen bzw. Worte der Trainingsmenge auf sie zutreffen. Dieser Fall tritt vor allem bei den CWE auf, welche verhältnismäßig wenig Einträge aufweisen, jedoch ist hier nicht die Anzahl der Einträge in der Testmenge entscheidend, sondern die Einträge der Trainingsmenge. Denn wie bereits erläutert muss die Trainingsmenge zunächst einen genügend großen Wortschatz besitzen, um über alle Methoden des Projektes ein Urteil fällen zu können.

	In der Testmenge		**gefilterte Einträge**	
CWE	**Einträge**	**sec-Einträge**	**nonsec**	**sec**
20	588	18	12	0
200	447	22	34	0
22	442	22	5	0
264	262	15	25	1
399	282	26	57	0
79	146	9	4	1
287	36	6	8	1

Tabelle 7.6: Gefilterte Einträge der Analyse mit Datentypen

Die bisherigen Analysen wurden auf die gefilterten Datensätzen angewandt, dabei wurden auch Mehrfacheinträge eines Wortes innerhalb einer Methode auf einen Eintrag reduziert. Um dieses Vorgehen zu rechtfertigen, wird mit dem obigen Datensatz eine weitere Analyse durchgeführt, ohne Duplikate aus den Methoden zu filtern. Die Ergebnisse sind relativ eindeutig und können in Tabelle 7.7 eingesehen und mit denen aus Tabelle 7.5 verglichen werden.

	F1-Maß des Klassifikators			
CWE	**k-NN**	**SVM**	**Entscheidungsbaum**	**Perzeptron**
20	0,42	0,36	0,21	0,74
200	0,32	0,28	0,38	0,35
22	0,29	0,23	0,67	0,42
264	0,26	0,23	0,21	0,31
399	0,21	0,2	0,52	0,36
79	0,3	0,15	0,32	0,35
287	0,44	0,43	0,47	0,55
Durchschnitt	0,32	0,27	0,4	0,44

Tabelle 7.7: Analyse mit Datentypen und Duplikaten

Alle Klassifikatoren liefern ein schlechteres F1-Maß, wenn die Duplikate nicht gefiltert werden. Es hätte angenommen werden können, dass der Bezeichner einer Variable, welche in einer Methode öfter geändert oder genutzt wird eine höhere Relevanz hat als andere, jedoch zeigt das Ergebnis, dass die Häufigkeit der Nutzung eines Bezeichners kein Verhältnis zu dessen Verbindung mit der Methode hat. Stattdessen genügt es, dass ein Bezeichner lediglich einmal in einer Methode genutzt wird, um für diese relevant zu sein.

7.3.4 Analyse ohne Datentypen

Um festzustellen, ob die Nutzung der Datentypen die Analyse tatsächlich aufgrund von mehr Informationen verbessert, folgt die identische Analyse mit gefilterten Einträgen, wobei die CSV-Dateien ohne Datentypen erstellt werden. Die Ergebnisse sind in Tabelle 7.8 aufgeführt und können mit denen in Tabelle 7.5 verglichen werden. Die einzelnen Teilwerte sind wiederum im Anhang in den Tabelle A.9 bis Tabelle A.12 aufgeführt.

	F1-Maß des Klassifikators			
CWE	**k-NN**	**SVM**	**Entscheidungsbaum**	**Perzeptron**
20	0,56	0,75	0,53	0,81
200	0,41	0,25	0,42	0,51
22	0,31	0,46	0,68	0,61
264	0,29	0,27	0,33	0,45
399	0,31	0,2	0,56	0,48
79	0,4	0,36	0,36	0,28
287	0,42	0,4	0,25	0,22
Durchschnitt	0,39	0,38	0,45	0,48

Tabelle 7.8: Analyse ohne die Nutzung von Datentypen

Die Ergebnisse der Analyse mit Datentypen und ohne Datentypen liegen in den Durchschnittswerten sehr nah beieinander. Jedoch können bei den Auswertungen der einzelnen Klassifikatoren (Tabelle A.5 bis Tabelle A.12) markante Unterschiede festgestellt werden. So ist die Trefferquote bei der Analyse mit Datentypen für die CWE mit verhältnismäßig wenigen Einträgen meist höher. Die Ursache hierfür ist klar, denn durch die weiteren Worte in der Trainingsmenge liegen mehr Worte vor, welche auch in den sicherheitsrelevanten Methoden der Testmenge auftreten. Dadurch können mehr Methoden detektiert werden, jedoch steigt dadurch auch die Anzahl der fälschlicherweise als sicherheitsrelevant beurteilten Einträge, wodurch die Präzision allgemein sinkt. Zudem scheinen die Durchschnitts-Ergebnisse mit Datentypen relativ stark von dem CWE-287 beeinflusst zu sein, wobei bei diesem CWE nur relativ wenige Einträge der Analyse zugrunde liegen, welche aus lediglich drei unterschiedlichen Klassen stammen. Daher sollten die Ergebnisse nicht ebenso stark gewichtet werden, wie beispielsweise die vom CWE-20, wessen Analyse auf mit Abstand am meisten Einträgen beruht. Nichts desto trotz scheinen beide Analysen, sowohl mit als auch ohne Datentypen, ihre Vor- und Nachteile zu besitzen, jedoch scheint die Wahl des Klassifikators einfacher zu sein.

Denn bei sehr kleinen Mengen liefert k-NN die besten Ergebnisse, wobei es immer schwierig ist, eine Menge von dokumentierten Sicherheitslücken als klein genug zu beurteilen. Für größere Mengen hingegen ist ein Entscheidungsbaum oder ein Perzeptron als Klassifikator am effektivsten. Jedoch basieren die Modelle des Entscheidungsbaums, welche in Abbildung 7.6 bis Abbildung 7.12 dargestellt sind, lediglich auf einem bis vier unterschiedlichen Worten. Das scheint nicht sehr zuverlässig zu sein, denn alle sicherheitsrelevanten Methoden ohne diese Worte können nicht erkannt werden. So ist die Beurteilung des Entscheidungsbaums des CWE-79 (Abbildung 7.12) lediglich davon abhängig, ob der Bezeichner http verhältnismäßig oft in einer Methode genutzt wird, was bei vielen Methoden eines Projektes zu einem Webserver der Fall sein kann, auch wenn sie nicht sicherheitsrelevant sind. Zu dem CWE-264 (Abbildung 7.10) konnte ebenfalls kein klares Modell erstellt werden, denn bereits die Klassifizierung der Trainingsmenge weist schwächen auf. Das ist daran zu erkennen, dass die untere Klassifizierung (zu sec) nach einem Wert für "required" von größer als 0,388 noch immer zu knapp einem Drittel falsch vorhersagt wird (zu erkennen an dem Blauanteil im Verhältnis zu dem Rotanteil). Ein gutes Beispiel hingegen ist der Entscheidungsbaum zum CWE-200 (Abbildung 7.6), dort wird nach einem geringen Wert für "authenticate" und einem hohen für "response" die Methode als sicherheitsrelevant eingeteilt, denn das könnte bedeuten, dass ein Nutzer

für eine Ausgabe nicht hinreichend autorisiert wird, was der Beschreibung des CWE-200 entspricht. Somit sollte für die Nutzung eines Entscheidungsbaums allgemein eine größere Menge an dokumentierten Sicherheitslücken vorliegen. Dies könnte erreicht werden, indem nicht die Aufteilung der Sicherheitslücken in die einzelnen CWE vorgenommen wird, sondern alle Sicherheitslücken gemeinsam für das Modell verwendet werden.

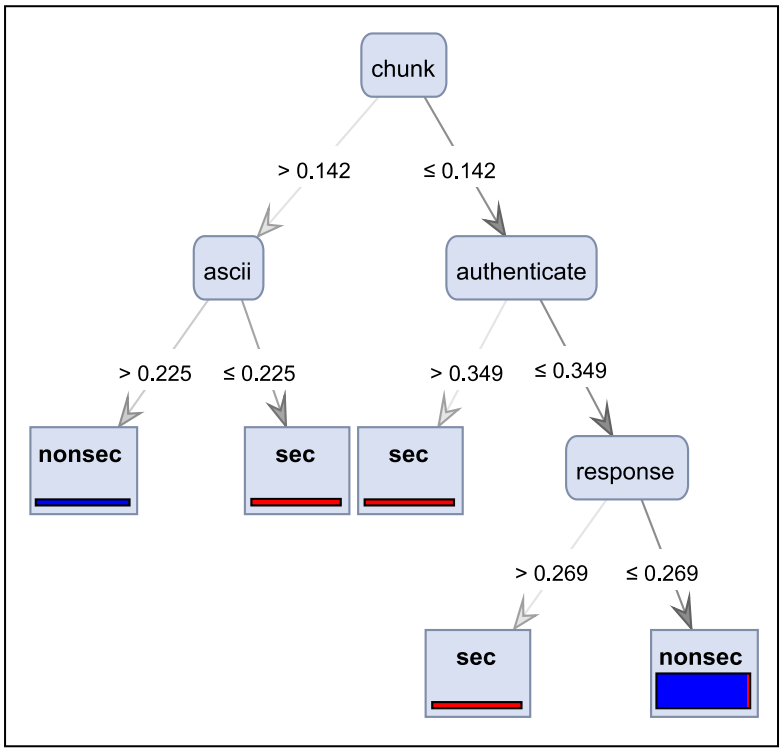

Abbildung 7.6: Entscheidungsbaum CWE-200

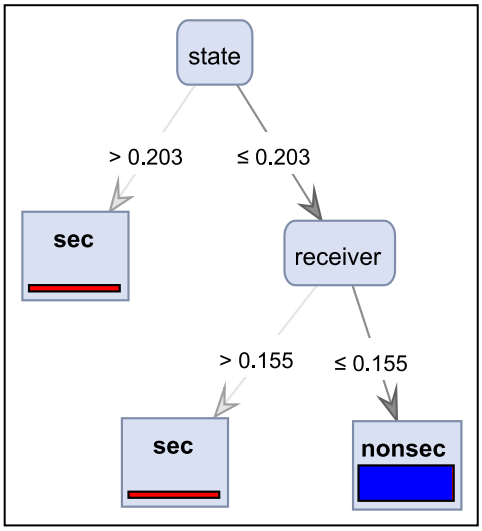

Abbildung 7.7: Entscheidungsbaum CWE-20

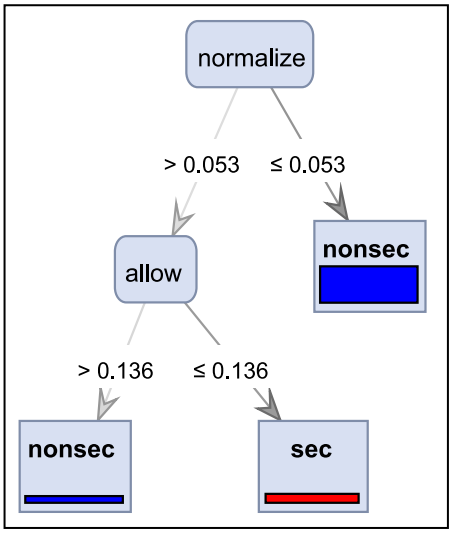

Abbildung 7.8: Entscheidungsbaum CWE-22

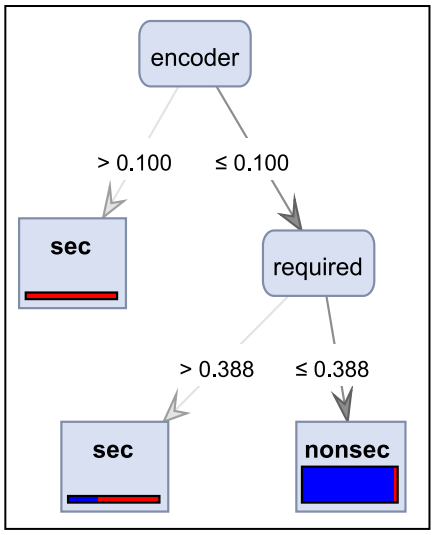

Abbildung 7.10: Entscheidungsbaum CWE-264

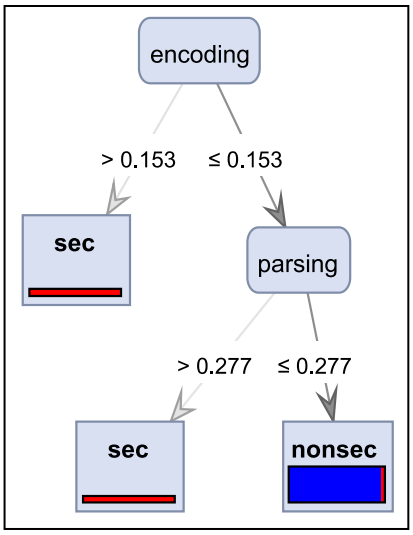

Abbildung 7.9: Entscheidungsbaum CWE-399

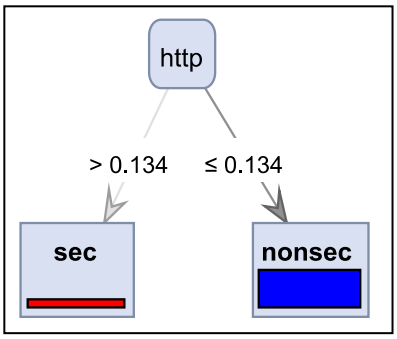

Abbildung 7.12: Entscheidungsbaum CWE-79

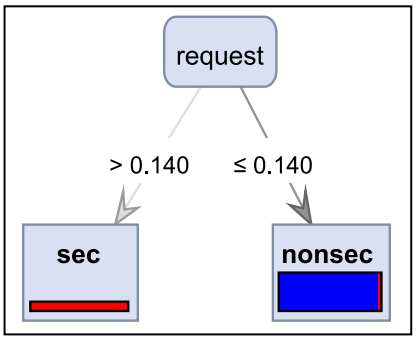

Abbildung 7.11: Entscheidungsbaum CWE-287

7.3.5 Analyse ohne CWE-Kategorien

Für diese Analyse werden lediglich alle Trainings- und Testmengen der einzelnen CWE in eine gemeinsame Trainingsmenge und eine gemeinsame Testmenge zusammengefügt. Um weiterhin beurteilen zu können, ob die Nutzung von Datentypen in diesem Fall einen Vorteil hat, wird diese Analyse sowohl mit, als auch ohne Datentypen durchgeführt. Die Ergebnisse sind in Tabelle 7.9 und Tabelle 7.10 aufgeführt.
Überraschenderweise erreicht der Entscheidungsbaum bei der Analyse mit Datentypen eine Trefferquote und Präzision von 0. Es konnte kein Modell bei dieser Analyse erstellt werden, da durch die Datentypen die Überschneidung von Worten in sicherheitsrelevanten und nicht sicherheitsrelevanten Methoden zu groß ist.

Klassifikator	Trefferquote	Präzision	F1-Maß	Falsch-Positiv-Rate	Falsch-Negativ-Rate
k-NN	0,37	0,42	**0,39**	0,03	0,63
SVM	0,27	0,28	**0,27**	0,04	0,73
Entscheidungsbaum	0	0	**0**	0	1
Perzeptron	0,61	0,25	**0,35**	0,1	0,39

Tabelle 7.9: Analyse aller CWE mit Datentypen

Klassifikator	Trefferquote	Präzision	F1-Maß	Falsch-Positiv-Rate	Falsch-Negativ-Rate
k-NN	0,55	0,6	**0,57**	0,02	0,45
SVM	0,46	0,4	**0,43**	0,05	0,54
Entscheidungsbaum	0,29	0,64	**0,4**	0,01	0,71
Perzeptron	0,67	0,25	**0,36**	0,13	0,33

Tabelle 7.10: Analyse aller CWE ohne Datentypen

Die restlichen Klassifikatoren werden bei einer solchen Menge von Einträgen durch die Datentypen ebenfalls verschlechtert. Der Entscheidungsbaum erreicht allerdings auch ohne Datentypen keine besseren Werte, was wie zuvor an der schlechten Trefferquote liegt. Jedoch ist das Modell des Entscheidungsbaums nun um einiges größer, wodurch es einen größeren Informationsgehalt besitzt, als die Entscheidungsbäume der einzelnen CWE. Die Abbildung 7.13 zeigt ein Bild des Entscheidungsbaums, wobei die Werte der Kanten entfernt wurden.

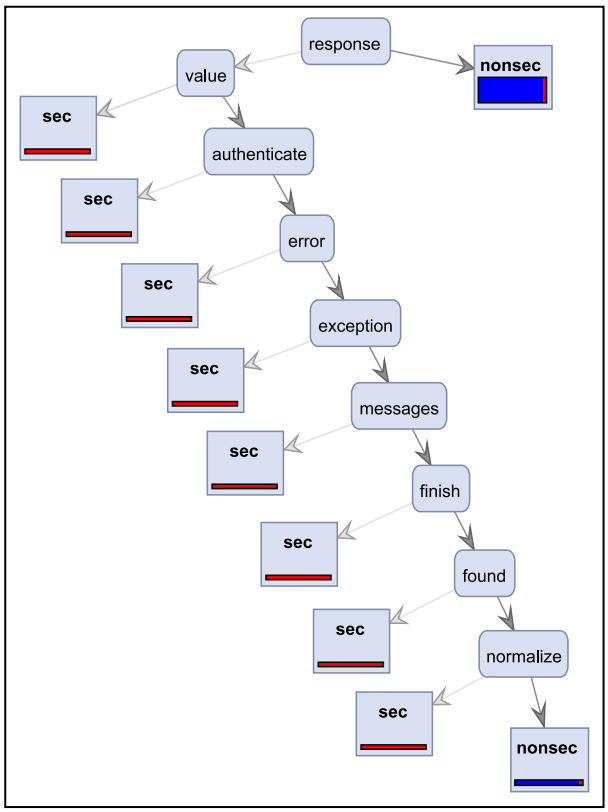

Abbildung 7.13: Entscheidungsbaum aller CWE

Weiterhin ist auffällig, dass die zuvor schlechteren Klassifikatoren k-NN und SVM nun ein höheres F_1-Maß erreichen, als die vorherigen Favoriten. Der Klassifikator k-NN hat für alle Sicherheitslücken ähnliche Werte, wie zuvor beim CWE-20, welches unter den CWE die besten Werte hat und auch am meisten Einträge hat. Wenn die Analyse also mit einer kleinen Menge von Sicherheitslücken trainiert wird, sollten die Einträge der Sicherheitslücken nicht in die CWE getrennt werden, sondern für alle gemeinsam mit dem Klassifikator k-NN analysiert werden. Jedoch wird bei der Analyse aller CWE gemeinsam die höchste Trefferquote durch das Perzeptron erreicht. Es hat zwar auch die höchste Falsch-Positiv-Rate, wodurch mehr Falschmeldungen durchgesehen werden müssen, jedoch werden dadurch auch die meisten Sicherheitslücken entdeckt. Sollte das Ziel also eine hohe Abdeckung aller Sicherheitslücken sein, wobei ein erhöhter Aufwand keine Rolle spielt, ist die Analyse mit einem Perzeptron das erfolgreichste Verfahren.

7.3.6 Analyse mit Perzeptron

Eine Analyse mit einem Perzeptron ist zu empfehlen, wenn für einzelne CWE eine hinreichend große Trainingsmenge vorliegt. Zur Veranschaulichung habe die Anzahl der mir zur Verfügung stehenden Einträge zu dem F_1-Maß der Analyse mit einem Perzeptron verglichen; die Ergebnisse sind in Abbildung 7.14 dargestellt.

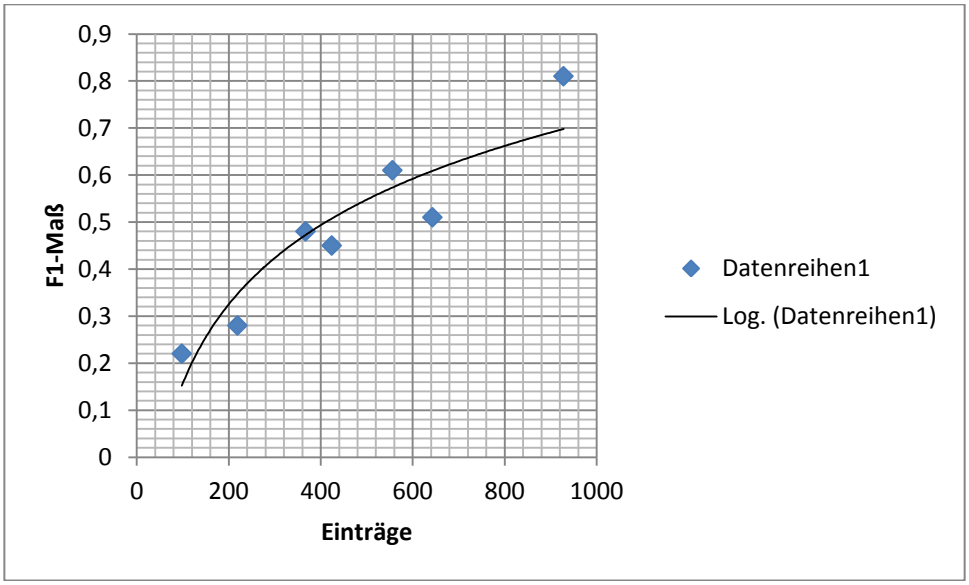

Abbildung 7.14: Verhältnis der Einträge zum F1-Maß

Die x-Achse gibt die Anzahl der Einträge des jeweiligen CWE an und auf der y-Achse ist das dem CWE entsprechende F_1-Maß, welches aus der Analyse durch das Perzeptron hervorgegangen ist dargestellt. Zur Veranschaulichung habe Ich durch die Ergebnisse eine logarithmische Regression gelegt, welche in etwa einen Trend angibt. Jedoch zeigt die erhöhte Abweichung der Daten der CWE mit einer hohen Anzahl von der Kurve, dass diese nicht einfach extrapoliert werden kann und somit ab einer bestimmten Anzahl Einträgen ein F_1-Maß nahe eins erreicht wird. Trotzdem gibt diese Kurve einen Trend an, durch welchen klar erkennbar ist, dass höhere Trainingsmengen, mit Perzeptron als Klassifikator ein höheres F_1-Maß liefern.
Des Weiteren habe Ich auch die Falsch-Positiv- und Falsch-Negativ-Rate mit der Anzahl der Einträge verglichen und erneut durch die Ergebnisse eine logarithmische Regression gelegt; die Ergebnisse sind in Abbildung 7.15 und Abbildung 7.16 dargestellt. Auch hier ist eine Extrapolation nicht direkt möglich, denn es gilt nicht, dass die Falsch-Positiv-Rate ab einer Anzahl von ca. 1200 Einträgen immer gleich 0 ist, jedoch kann angenommen werden, dass sie für solch hohe Anzahlen nah an 0 liegt. Das eine genaue Aussage durch diese Graphen nicht möglich ist, wird auch durch die hohe Abweichung der Werte in Abbildung 7.15 klar. Dennoch zeigt der Graph, dass bei höheren Trainingsmengen die Falsch-Positiv-Rate relativ gering ist. Die Falsch-Negativ-Rate hingegen, fällt nicht ganz so stark ab, das heißt, um tatsächlich alle bzw. möglichst viele Schwachstellen detektieren zu können, reicht es nicht aus, dieses Verfahren mit einer sehr großen Trainingsmenge durchzuführen.

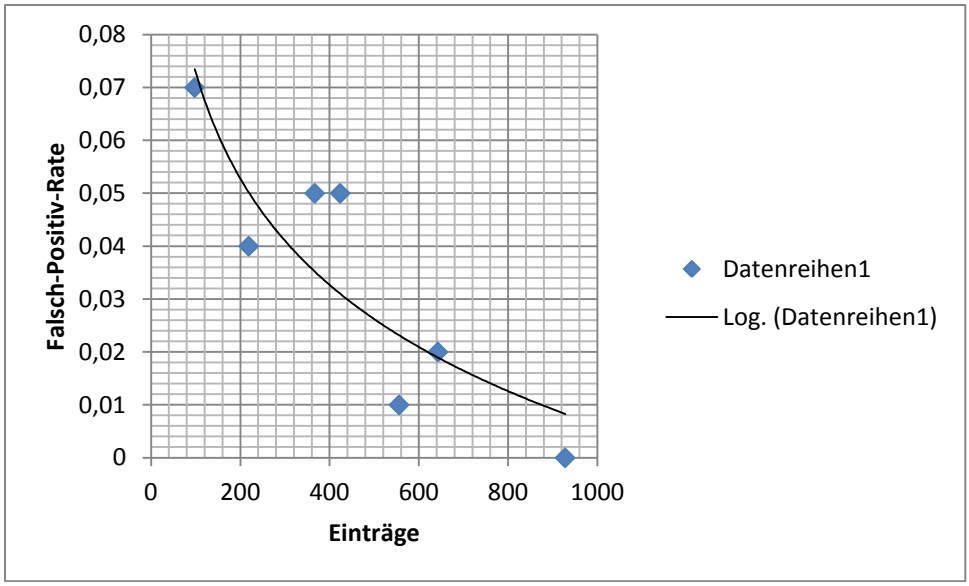

Abbildung 7.15: Verhältnis der Einträge zur Falsch-Positiv-Rate

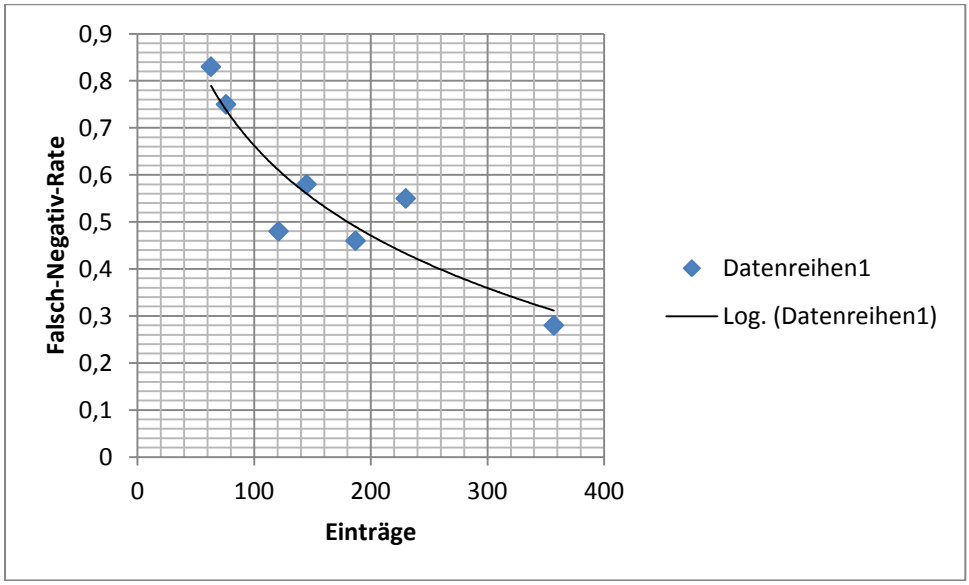

Abbildung 7.16: Verhältnis der Eintrage zur Falsch-Negativ-Rate

Das Ziel einer Sicherheitsanalyse ist jedoch, möglichst viele Schwachstellen aufzudecken, daher wird in Kapitel 7.3.7 ein weiterer Ansatz einer Analyse erstellt. Um zuvor jedoch nochmals auf die Relevanz der Größe der Trainingsmenge einzugehen, sind in Tabelle 7.11 drei unterschiedliche Analysen mit ihren Ergebnissen aufgeführt. Dabei wurde erneut ein Perzeptron als Klassifikator genutzt und als Grundlage wurde die Trainings- und Testmenge des CWE genutzt, für welches mir die meisten Einträge zur Verfügung stehen. Bei den bisherigen Analysen wurden bei diesem CWE 37,78% aller Einträge für die Trainingsmenge verwendet. In dieser Analyse wurde der prozentuale Anteil variiert, um auszumachen, ab welcher Anzahl Einträge die Trainingsmenge zu gering wird. Die genauen Anzahlen der Einträge der Trainings- und Testmenge sind in Tabelle 7.12 aufgeführt.

Trainigsmenge	Trefferquote	Präzision	F1-Maß	Falsch-Positiv-Rate	Falsch-Negativ-Rate
37,78%	0,72	0,93	**0,81**	0	0,28
22,65%	0,68	0,94	**0,79**	0	0,32
19,58%	0,29	0,47	**0,36**	0,01	0,71

Tabelle 7.11: Analyse mit variabler Trainingsmenge

	In der Trainingsmenge		In der Testmenge		gefilterte Einträge	
Trainigsmenge	Einträge	sec-Einträge	Einträge	sec-Einträge	nonsec	sec
37,78%	357	10	588	18	17	0
22,65%	214	6	731	22	29	0
19,58%	195	4	760	24	58	0

Tabelle 7.12: Einträge der Teilmengen bei variabler Trainingsmenge

Zuerst wurde die Größe der Trainingsmenge von 357 auf 214 verringert. Das ist eine ca. 40% kleinere Trainingsmenge, dementsprechend sind in der neuen Trainingsmenge auch nur noch sechs sicherheitsrelevante Methoden enthalten, im Gegensatz zu den ursprünglichen zehn. Trotz dieser drastischen Reduzierung der Trainingsmenge bleibt das Ergebnis der Analyse fast identisch, es verschlechtert sich nur sehr gering, jedoch werden nun 29 Einträge gefiltert, über welche keine Aussage gemacht werden kann, im Gegensatz zu den ursprünglichen 17. Jedoch werden noch immer keine sicherheitsrelevanten Einträge gefiltert. Im nächsten Schritt wurde die Trainingsmenge um lediglich 10% verringert, jedoch waren in diesen 10% zwei sicherheitsrelevante Methoden enthalten, wodurch die Anzahl dieser von sechs auf lediglich vier fällt. Das macht sich in den Analyseergebnissen auch stark bemerkbar. Sowohl die Trefferquote als auch die Präzision sinken um etwa die Hälfte. Zusätzlich werden in dieser Analyse bereits 58 Einträge gefiltert, was doppelt so viele sind, wie in der vorherigen Analyse.

Scheinbar waren in der ersten Trainingsmenge viele redundante Einträge enthalten, wodurch die drastische Reduzierung auf die Analyse keinen großen Einfluss hatte. Bei der zweiten Verkleinerung der Trainingsmenge wurden jedoch markante Worte entfernt, welche dadurch für die Analyse fehlten und das Ergebnis stark beeinflusst haben. Damit ist klar, dass vor allem der Umfang der Worte in der Trainingsmenge eine entscheidende Rolle für die Analyse spielt. In Tabelle 7.13 ist der Umfang des Wortschatzes der einzelnen Trainingsmengen aufgelistet. Dabei fällt auf, dass nach beiden Reduzierungen der Trainingsmenge die Anzahl der unterschiedlichen Worte in der Trainingsmenge um 50 gesunken ist, obwohl von der zweiten zur dritten Analyse viel weniger Einträge entfernt wurden.

Trainigsmenge	*Einträge*	unterschiedliche Worte
37,78%	357	360
22,65%	214	310
19,58%	195	260

Tabelle 7.13: Wortumfang unterschiedlich großer Trainingsmengen

7.3.7 Teilen der Mengen sicherheitsrelevanter Worte

Die bisher vorgestellten Analysen weisen meist eine durchschnittliche Trefferquote von 50% auf, jedoch zeigt die zuerst durchgeführte Analyse durch eine simple Inklusion, dass zumindest der Wortschatz für eine hohe Trefferquote in den sicherheitsrelevanten Methoden vorhanden ist. Da der Anteil der sicherheitsrelevanten Einträge zu allen Einträgen jedoch sehr gering ist, unterscheidet sich der Wortschatz der positiven von den negativen Einträgen nicht. Das heißt, alle Worte die in sicherheitsrelevanten Methoden enthalten sind, tauchen auch wenigsten einmal in einer nicht sicherheitsrelevanten Methode auf. Dadurch weist die Inklusion sehr viele Falsch-Positivs auf, wodurch eine sehr geringe Präzision und dementsprechend ein sehr geringes F1-Maß erzielt wird.
Um dies zu umgehen, werden für die folgende Analyse die positiven und negativen Einträge der Trainingsmenge in zwei disjunkte Mengen von Worten geteilt. Die Entscheidung, zu welcher Menge ein Wort hinzugefügt wird, fällt auf der Basis der Häufigkeit des Auftretens des Wortes in den sicherheitsrelevanten bzw. nicht sicherheitsrelevanten Methoden. Das heißt, es wird zunächst für jedes Wort eine prozentuale Auftretenshäufigkeit für positive, sowie für negative Einträge erstellt. Wenn beispielsweise ein Wort in drei von fünf sicherheitsrelevanten Methoden enthalten ist, erhält dieses Wort für positive Einträge einen Wert von 60%. Jedoch tritt das Wort auch in sechs von den restlichen 100 Methoden auf, wodurch es eine Wahrscheinlichkeit von 6% für negative Einträge erhält.
Um aus diesen Häufigkeiten zu entscheiden, ob das Wort sicherheitsrelevant ist oder nicht, wird die Differenz der beiden Werte berechnet. Sollte der Betrag der Differenz einen manuell gewählten Grenzwert überschreiten, wird das Wort der Teilmenge (positiv oder negativ) zugeteilt, welche die jeweils größere Auftretenshäufigkeit aufweist. In der folgenden Analyse wird als Grenzwert 30% gewählt, das heißt, wenn ein Wort 30% mal öfter in einer der beiden Teilmengen auftritt, wird es dieser zugeordnet. Ebenso bedeutet das aber auch, dass alle Worte, welche in beiden Mengen etwa gleich häufig auftreten (innerhalb einer Spanne von 30%) werden komplett aus der Trainingsmenge entfernt. Durch dieses Vorgehen sind die in Tabelle 7.14 dargestellten Mengen entstanden, wobei ein Eintrag in der neuen Trainingsmenge jetzt nur noch aus genau einem Wort besteht.

	Neue Trainingsmenge			Ursprüngliche Trainingsmenge		
CWE	**Einträge**	**sec-Einträge**	**%-Anteil sec**	**Einträge**	**sec-Einträge**	**%-Anteil sec**
20	307	21	6,84	357	10	2,8
200	199	6	3,02	230	16	6,96
22	210	3	1,43	121	11	9,09
264	231	3	1,3	187	12	6,42
399	95	1	1,05	145	12	8,28
79	120	18	15	76	5	6,58
287	101	13	12,87	63	6	9,52

Tabelle 7.14: Anzahlen der Einträge der erstellten disjunkten Trainingsmengen

In der neuen Trainingsmenge weicht der Anteil der sicherheitsrelevanten Einträge stark von der Anzahl der Einträge in der ursprünglichen Trainingsmenge ab. Zudem weisen die neuen Trainingsmengen von vier der sieben CWE weniger als sieben sicherheitsrelevante Einträge auf, was tatsächlich ebenso viele sicherheitsrelevante Worte sind. In den Tabelle 7.15 bis Tabelle 7.17 sind die Ergebnisse der Analyse durch drei der vier Klassifikatoren aufgeführt, lediglich Entscheidungsbäume konnten auf dieser Trainingsmenge, welche lediglich aus einzelnen Worten besteht, nicht erfolgreich durchgeführt werden.

Alle Tabellen enthalten einen weiteren Durchschnitt, welcher nur diejenigen CWE berücksichtigt, welche auch eine angemessene Anzahl an sicherheitsrelevanten Worten enthalten. Diesem Durchschnitt kann entnommen werden, dass die höchste Trefferquote durch die Klassifikation mit k-NN erreicht wird, jedoch ist die Präzision in dem Fall nur minimal höher als durch die ursprüngliche Inklusion. Eine höhere Präzision wird mit dem Klassifikator SVM erzielt, wobei die Trefferquote nur geringfügig kleiner im Verhältnis zu der Analyse mit k-NN ist.

CWE	Trefferquote	Präzision	F1-Maß	Falsch-Positiv-Rate	Falsch-Negativ-Rate
20	1	0,07	**0,13**	0,49	0
200	0,25	0,08	**0,12**	0,17	0,75
22	0,79	0,2	**0,32**	0,18	0,21
264	0,5	0,13	**0,21**	0,25	0,5
399	0,17	0,67	**0,27**	0,02	0,83
79	0,75	0,08	**0,14**	0,61	0,25
287	1	0,33	**0,5**	0,53	0
Durchhschnitt	0,64	0,22	0,24	0,32	0,36
Durchhschnitt CWE-20/79/287	0,92	0,16	0,26	0,54	0,08

Tabelle 7.15: Analyse mit disjunkter Trainingsmenge (k-NN)

CWE	Trefferquote	Präzision	F1-Maß	Falsch-Positiv-Rate	Falsch-Negativ-Rate
20	0,94	0,14	**0,24**	0,21	0,06
200	0,5	0,22	**0,31**	0,11	0,5
22	0,58	0,24	**0,34**	0,1	0,42
264	0,08	0,2	**0,11**	0,02	0,92
399	0,13	0,6	**0,21**	0,02	0,87
79	0,63	0,15	**0,24**	0,24	0,37
287	1	0,4	**0,57**	0,4	0
Durchhschnitt	0,55	0,28	0,29	0,16	0,45
Durchhschnitt CWE-20/79/287	0,86	0,23	0,35	0,28	0,14

Tabelle 7.16: Analyse mit disjunkter Trainingsmenge (SVM)

CWE	Trefferquote	Präzision	F1-Maß	Falsch-Positiv-Rate	Falsch-Negativ-Rate
20	0,67	0,14	**0,23**	0,15	0,33
200	0,95	0,17	**0,29**	0,28	0,05
22	0,95	0,17	**0,29**	0,27	0,05
264	0,08	0,07	**0,07**	0,09	0,92
399	0,39	0,82	**0,53**	0,02	0,61
79	0,25	0,2	**0,22**	0,07	0,75
287	1	0,31	**0,47**	0,6	0
Durchhschnitt	0,61	0,27	0,3	0,21	0,39
Durchhschnitt CWE-20/79/287	0,64	0,22	0,31	0,27	0,36

Tabelle 7.17: Analyse mit disjunkter Trainingsmenge (Perzeptron)

Da mehr als die Hälfte der analysierten CWE aber einer mangelhaften Trainingsmenge unterliegen, spiegelt dieses Ergebnis wahrscheinlich nicht das ganze Potential wieder, was durch eine abgewandelte Aufbereitung möglich wäre.
Um eine dem CWE angepasste Trainingsmenge zu erstellen, sollte der Grenzwert, der für die Aufteilung zuständig ist, dynamisch bestimmt werden. Dabei sollte das Ziel sein, dass eine gewisse Mindestanzahl an sicherheitsrelevanten Worten genutzt wird. Zugleich sollte der prozentuale Anteil der sicherheitsrelevanten Einträge der neuen Trainingsmenge nicht stark von dem prozentualen Anteil in der ursprünglichen Trainingsmenge abweichen. Doch selbst nach solch einer Aufbereitung werden wahrscheinlich ähnlich viele Worte aus beiden Mengen entfernt, wodurch für viele Einträge der Testmenge keine Aussage über deren Sicherheitsrelevanz getroffen werden kann. Die Anzahlen der gefilterten Einträge der zuvor durchgeführten Analyse sind in Tabelle 7.18 aufgeführt. Daraus geht hervor, dass bis zu 50% der Daten gefiltert wurden, da sie keine Worte der Trainingsmenge enthielten. Zudem werden bei mehreren CWE auch sicherheitsrelevante Einträge gefiltert, was nicht passieren sollte.

	In der Testmenge		gefilterte Einträge	
CWE	Einträge	sec-Einträge	nonsec	sec
20	588	18	67	0
200	447	22	92	0
22	442	22	92	3
264	262	15	83	3
399	282	26	143	3
79	146	9	22	1
287	36	6	15	2

Tabelle 7.18: Gefilterte Einträge der Analyse mit disjunkten Mengen

Um alle Probleme, die diese Analyse aufbringt, zunächst ohne großen Aufwand zu umgehen, wird wie in den vorherigen Analysen ebenfalls eine gemeinsame Analyse aller CWE durchgeführt. Dadurch wird zunächst die Anzahl der gefilterten Einträge erheblich verringert, wie in Tabelle 7.19 aufgeführt ist, sodass nur noch zu weniger als 3% der Testmenge keine Aussage mehr getroffen werden kann. Dabei werden zwar immer noch zwei sicherheitsrelevante Methoden gefiltert, jedoch erheblich weniger als zuvor. Ebenso weicht bei dieser Analyse der prozentuale Anteil der sicherheitsrelevanten Methoden der neuen Trainingsmenge nicht mehr so stark von dem Anteil in der ursprünglichen Trainingsmenge ab, wie in Tabelle 7.20 aufgeführt ist.

	In der Testmenge		gefilterte Einträge	
CWE	Einträge	sec-Einträge	nonsec	sec
Alle	2203	118	58	2

Tabelle 7.19: Gefilterte Einträge der Analyse aller CWE mit disjunkten Mengen

	urspr. Trainingsmenge			neue Trainingsmenge		
CWE	Einträge	sec-Einträge	%-Anteil sec	Einträge	sec-Einträge	%-Anteil sec
Alle	1179	72	6,11	1263	65	5,15

Tabelle 7.20: Verteilung der sicherheitsrelevanten Einträge

Die resultierenden Analyseergebnisse sind in Tabelle 7.21 aufgeführt, wobei der Klassifikator k-NN wie bei der Analyse im Kapitel 7.3.5 die besten Ergebnisse für CWE übergreifende Analysen liefert. Dieser erreicht nun eine Trefferquote für alle Sicherheitsrelevanten Methoden von 88%, jedoch liegt die Präzision immer noch bei lediglich 10%.

CWE	Trefferquote	Präzision	F1-Maß	Falsch-Positiv-Rate	Falsch-Negativ-Rate
k-NN	0,88	0,1	**0,18**	0,47	0,12
SVM	0,25	0,14	**0,18**	0,09	0,75
Perceptron	0,34	0,17	**0,23**	0,09	0,66

Tabelle 7.21: Ergebnis der Analyse mit disjunkten Mengen auf allen CWE

8 Zusammenfassung und Ausblick

In diesem Kapitel werden noch einmal die Ergebnisse der evaluierten Verfahren zusammengefasst, sowie ein Ausblick auf mögliche Erweiterungen, Automatisierungen oder Kombinationen des Verfahrens mit bisherigen Verfahren geliefert.

8.1 Zusammenfassung

Das erste Ziel dieser Arbeit war die Aufbereitung der natürlichsprachigen Informationen eines Quelltextes, dafür wurde zunächst ein Verfahren zur Worttrennung basierend auf den Java Code-Konventionen erarbeitet. Dieses Verfahren erzielte in der Stichprobe des Testprojekts eine korrekte Trennung der Teilworte der Bezeichner von 97,42%, wodurch mit einer 99% Wahrscheinlichkeit die Anzahl von korrekten Trennungen zwischen 95,42. und 99,42% liegt.

Auf der Basis dieser Worttrennung wurde ein Verfahren zur Normalisierung von Abkürzungen erarbeitet. Diese Normalisierungsverfahren konnte im Testprojekt für 71,85% von allen Abkürzungen eine Normalisierung erstellen, wobei der Anteil der korrekten Normalisierungen 82,89% beträgt. Da vor der Normalisierung bereits 84,01% aller Bezeichner, welche die in Kapitel 6.3 erläuterten Vorgaben erfüllen, korrekte Worte, werden von diesen Bezeichnern insgesamt 93,53% entweder zu korrekten Worten normalisiert oder sie sind bereits Worte.

Mit Hilfe dieser hochwertigen Aufbereitung wurden die Bezeichner von Klassen mit dokumentierten Schwachstellen in CSV-Dateien nach ihrer Aufbereitung gespeichert. Wobei jede Methode einzeln zeilenweise gespeichert wurde und, je nachdem, ob sie sicherheitsrelevant ist oder nicht, als solches markiert wurde. Auf dieser Grundlage wurde das zweite Ziel dieser Arbeit, die Analyse der natürlichsprachigen Informationen durchgeführt.

Eine Analyse, welche lediglich auf Methodennamen basiert, ist weniger erfolgreich und somit nicht empfehlenswert. Die Hinzunahme von Datentypen zu der Analyse könnte als hilfreich eingestuft werden, wenn nur eine sehr geringe Trainingsmenge vorliegt. Bei großen Trainingsmengen oder einer CWE übergreifenden Analyse sollte auf die Datentypen verzichtet werden. Sowohl bei kleinen Mengen, als auch bei der CWE übergreifenden Analyse hat der einfachste Klassifikator k-NN die besten Ergebnisse geliefert, bei einer hinreichend großen Trainingsmenge für ein einzelnes CWE konnte sich im Umfang dieser Arbeit das Perzeptron durchsetzen. Dabei ist bei dem Umfang der Trainingsmenge aber vor allem die Größe des Wortschatzes, das heißt der unterschiedlichen Worte relevant.

In einem weiteren Ansatz, bei welchem die Trainingsmenge in disjunkte Mengen geteilt wurde, konnten höhere Trefferquoten erzielt werden, wobei die Einteilung der Trainingsmenge wahrscheinlich dynamisch entschieden werden sollte. Um bei diesem Ansatz jedoch nicht zu viele Methoden filtern zu müssen, da wegen einer mangelnden Anzahl von Worten keine Aussage getroffen werden kann, sollte diese Analyse wiederum auf allen Sicherheitslücken gemeinsam ohne CWE-Einteilung durchgeführt werden. Für dieses Verfahren hat sich erneut k-NN als Klassifikator durchgesetzt, da er die höchste Trefferquote aufwies.

8.2 Ausblick

Die Ergebnisse dieser Arbeit zeigen zunächst das enorme Potential der natürlichsprachigen Informationen für die Sicherheitsanalyse, ein Nachteil ist jedoch, dass zunächst dokumentierte Schwachstellen vorliegen müssen, um weitere zu detektieren. Ein Ansatz für dieses Problem wäre einen Klassifikator durch eine Vielzahl von Projekten einzulernen, und zu testen, ob dieser Klassifikator auf wiederum anderen Projekten dokumentierte Sicherheitslücken detektiert. Falls dies der Fall ist, könnte ein allgemeines Analysetool eingelernt werden, welches für jedes Projekt genutzt werden könnte.

Der alternative Ansatz im Kapitel 7.3.7 scheitert einerseits an der Aufbereitung, für welche bereits ein mögliches Vorgehen erläutert wurde, und andererseits an der hohen Anzahl gefilterter Einträge. Um dies zu vermindern, könnten alle Einträge, welche den Grenzwert nicht in eine Richtung der beiden Mengen überschreiten entweder zu der Negativmenge hinzugefügt werden, oder das Modell wird um eine Schnittmenge erweitert, was bedeutet, dass all diese Einträge sowohl zu der positiven als auch zu der negativen Menge hinzugefügt werden.

Sollte eine der vorgestellten Analysemethoden den Erwartungen für eine Sicherheitsanalyse entsprechen, könnte das gesamte Verfahren in einem automatisierten Programm implementiert werden. Dieses Programm wäre von der Implementierung viel einfacher als eine Kontrollflussanalyse, wie sie in bisherigen Analysewerkzeugen verwendet wird.

Um jedoch das komplette Potential auszuschöpfen kann entweder die dargestellte Inklusion oder die Analyse mit disjunkten Mengen als Voranalyse für weitere Analysewerkzeuge dienen. Ebenso könnten die hier dargestellten Verfahren mit anderen Analysewerkzeugen verbunden werden, um ein noch höheres F_1-Maß mit einer sehr hohen Trefferquote zu erreichen.

Literaturverzeichnis

[1] E. Hill, Z. Fry, H. Boyd, and G. Sridhara, “AMAP: automatically mining abbreviation expansions in programs to enhance software maintenance tools,” *In MSR ’08: Proceedings of the Fifth InternationalWorking Conference on Mining Software Repositories*, pp. 79–88, 2008.

[2] MITRE, “FAQ,” 2014. [Online]. Available: http://cwe.mitre.org/about/faq.html. [Accessed: 10-Aug-2014].

[3] MITRE, “Common Weakness Enumeration,” 2014. [Online]. Available: http://cwe.mitre.org/index.html. [Accessed: 10-Aug-2014].

[4] E. Alpaydin, *Maschinelles Lernen*. 2008.

[5] A. V. Aho, M. S. Lam, R. Sethi, and J. C. Ullman, *Compiler*. 2008, p. 52.

[6] L. Guerrouj, M. Di Penta, G. Antoniol, and Y.-G. Guéhéneuc, “TIDIER: an identifier splitting approach using speech recognition techniques,” *J. Softw. Evol. Process*, vol. 25, no. 6, pp. 575–599, Jun. 2011.

[7] Sun Microsystems, “Naming Conventions,” 1999. [Online]. Available: http://www.oracle.com/technetwork/java/javase/documentation/codeconventions-135099.html#367.

[8] Bundesministerium des Innern/Bundesverwaltungsamt, “Handbuch für Organisationsuntersuchungen und Personalbedarfsermittlung,” 2013. [Online]. Available: http://www.orghandbuch.de/OHB/DE/ohb_pdf.pdf?__blob=publicationFile&v=6.

Abbildungsverzeichnis

Tabellenverzeichnis

A. Analysetabellen

CWE	Input	Trefferquote	Präzision	F1-Maß	Falsch-Positiv-Rate	Falsch-Negativ-Rate
20	*Bezeichner mD*	0,67	0,46	**0,55**	0,03	0,33
	Bezeichner oD	0,61	0,52	**0,56**	0,02	0,39
	Methoden	0	0	**0**	0,02	1
79	*Bezeichner mD*	0,63	0,24	**0,35**	0,12	0,37
	Bezeichner oD	0,38	0,43	**0,4**	0,04	0,62
	Methoden	0,4	1	**0,57**	0	0,6

Tabelle A.1: k-NN mit unterschiedlicher Grundlage

CWE	Input	Trefferquote	Präzision	F1-Maß	Falsch-Positiv-Rate	Falsch-Negativ-Rate
20	*Bezeichner mD*	0,44	0,67	**0,53**	0,01	0,56
	Bezeichner oD	0,67	0,86	**0,75**	0	0,33
	Methoden	0,29	0,83	**0,43**	0	0,71
79	*Bezeichner mD*	0,25	0,22	**0,23**	0,05	0,75
	Bezeichner oD	0,25	0,67	**0,36**	0,01	0,75
	Methoden	0,4	1	**0,57**	0	0,6

Tabelle A.2: SVM mit unterschiedlicher Grundlage

CWE	Input	Trefferquote	Präzision	F1-Maß	Falsch-Positiv-Rate	Falsch-Negativ-Rate
20	*Bezeichner mD*	0,56	0,71	**0,63**	0,01	0,44
	Bezeichner oD	0,56	0,5	**0,53**	0,02	0,44
	Methoden	0,29	0,83	**0,43**	0	0,71
79	*Bezeichner mD*	0,5	0,24	**0,32**	0,1	0,5
	Bezeichner oD	0,25	0,67	**0,36**	0,01	0,75
	Methoden	0	0	**0**	0	1

Tabelle A.3: Entscheidungsbaum mit unterschiedlicher Grundlage

CWE	Input	Trefferquote	Präzision	F1-Maß	Falsch-Positiv-Rate	Falsch-Negativ-Rate
20	*Bezeichner mD*	0,56	0,91	**0,69**	0	0,44
	Bezeichner oD	0,72	0,93	**0,81**	0	0,28
	Methoden	0,82	0,33	**0,47**	0,06	0,18
79	*Bezeichner mD*	0,5	0,24	**0,32**	0,1	0,5
	Bezeichner oD	0,25	0,33	**0,28**	0,04	0,75
	Methoden	0,4	1	**0,57**	0	0,6

Tabelle A.4: Perzeptron mit unterschiedlicher Grundlage

CWE	Trefferquote	Präzision	F1-Maß	Falsch-Positiv-Rate	Falsch-Negativ-Rate
20	0,67	0,46	**0,55**	0,03	0,33
200	0,3	0,46	**0,36**	0,02	0,7
22	0,5	0,23	**0,32**	0,09	0,5
264	0,71	0,21	**0,32**	0,17	0,29
399	0,5	0,26	**0,34**	0,19	0,5
79	0,63	0,24	**0,35**	0,12	0,37
287	1	0,33	**0,5**	0,45	0
Durchschnitt	0,62	0,31	0,39	0,15	0,38

Tabelle A.5: k-NN mit Datentypen

CWE	Trefferquote	Präzision	F1-Maß	Falsch-Positiv-Rate	Falsch-Negativ-Rate
20	0,44	0,67	**0,53**	0,01	0,56
200	0,3	0,33	**0,31**	0,03	0,7
22	0,32	0,58	**0,41**	0,01	0,68
264	0,29	0,29	**0,29**	0,05	0,71
399	0,19	0,33	**0,24**	0,05	0,81
79	0,25	0,22	**0,23**	0,05	0,75
287	0,8	0,36	**0,5**	0,32	0,2
Durchschnitt	0,37	0,4	0,36	0,07	0,63

Tabelle A.6: SVM mit Datentypen

CWE	Trefferquote	Präzision	F1-Maß	Falsch-Positiv-Rate	Falsch-Negativ-Rate
20	0,56	0,71	**0,63**	0,01	0,44
200	0,45	0,31	**0,37**	0,05	0,55
22	0,5	1	**0,67**	0	0,5
264	0,29	0,36	**0,32**	0,03	0,71
399	0,42	0,85	**0,56**	0,01	0,58
79	0,5	0,24	**0,32**	0,1	0,5
287	0,6	0,3	**0,4**	0,32	0,4
Durchschnitt	0,47	0,54	0,47	0,07	0,53

Tabelle A.7: Entscheidungsbaum mit Datentypen

CWE	Trefferquote	Präzision	F1-Maß	Falsch-Positiv-Rate	Falsch-Negativ-Rate
20	0,56	0,91	**0,69**	0	0,44
200	0,3	0,55	**0,39**	0,01	0,7
22	0,45	0,56	**0,5**	0,02	0,55
264	0,5	0,39	**0,44**	0,05	0,5
399	0,42	0,52	**0,46**	0,05	0,58
79	0,5	0,24	**0,32**	0,1	0,5
287	1	0,56	**0,72**	0,18	0
Durchschnitt	0,53	0,53	0,5	0,06	0,47

Tabelle A.8: Perzeptron mit Datentypen

CWE	Trefferquote	Präzision	F1-Maß	Falsch-Positiv-Rate	Falsch-Negativ-Rate
20	0,61	0,52	**0,56**	0,02	0,39
200	0,35	0,5	**0,41**	0,02	0,65
22	0,57	0,21	**0,31**	0,11	0,43
264	0,69	0,18	**0,29**	0,19	0,31
399	0,46	0,23	**0,31**	0,2	0,54
79	0,38	0,43	**0,4**	0,04	0,62
287	0,83	0,28	**0,42**	0,45	0,17
Durchschnitt	0,56	0,34	0,39	0,15	0,44

Tabelle A.9: k-NN ohne Datentypen

CWE	Trefferquote	Präzision	F1-Maß	Falsch-Positiv-Rate	Falsch-Negativ-Rate
20	0,67	0,86	**0,75**	0	0,33
200	0,35	0,2	**0,25**	0,07	0,65
22	0,33	0,78	**0,46**	0	0,67
264	0,31	0,24	**0,27**	0,06	0,69
399	0,15	0,29	**0,2**	0,05	0,85
79	0,25	0,67	**0,36**	0,01	0,75
287	0,67	0,29	**0,4**	0,34	0,33
Durchschnitt	0,39	0,48	0,38	0,08	0,61

Tabelle A.10: SVM ohne Datentypen

CWE	Trefferquote	Präzision	F1-Maß	Falsch-Positiv-Rate	Falsch-Negativ-Rate
20	0,56	0,5	**0,53**	0,02	0,44
200	0,4	0,44	**0,42**	0,03	0,6
22	0,52	1	**0,68**	0	0,48
264	0,31	0,36	**0,33**	0,03	0,69
399	0,42	0,85	**0,56**	0,01	0,58
79	0,25	0,67	**0,36**	0,01	0,75
287	0,17	0,5	**0,25**	0,03	0,83
Durchschnitt	0,38	0,62	0,45	0,02	0,62

Tabelle A.11: Entscheidungsbaum ohne Datentypen

CWE	Trefferquote	Präzision	F1-Maß	Falsch-Positiv-Rate	Falsch-Negativ-Rate
20	0,72	0,93	**0,81**	0	0,28
200	0,45	0,6	**0,51**	0,02	0,55
22	0,52	0,73	**0,61**	0,01	0,48
264	0,54	0,39	**0,45**	0,05	0,46
399	0,42	0,55	**0,48**	0,05	0,58
79	0,25	0,33	**0,28**	0,04	0,75
287	0,17	0,33	**0,22**	0,07	0,83
Durchschnitt	0,44	0,55	0,48	0,03	0,56

Tabelle A.12: Perzeptron ohne Datentypen

.

B. Evaluierungsprozess

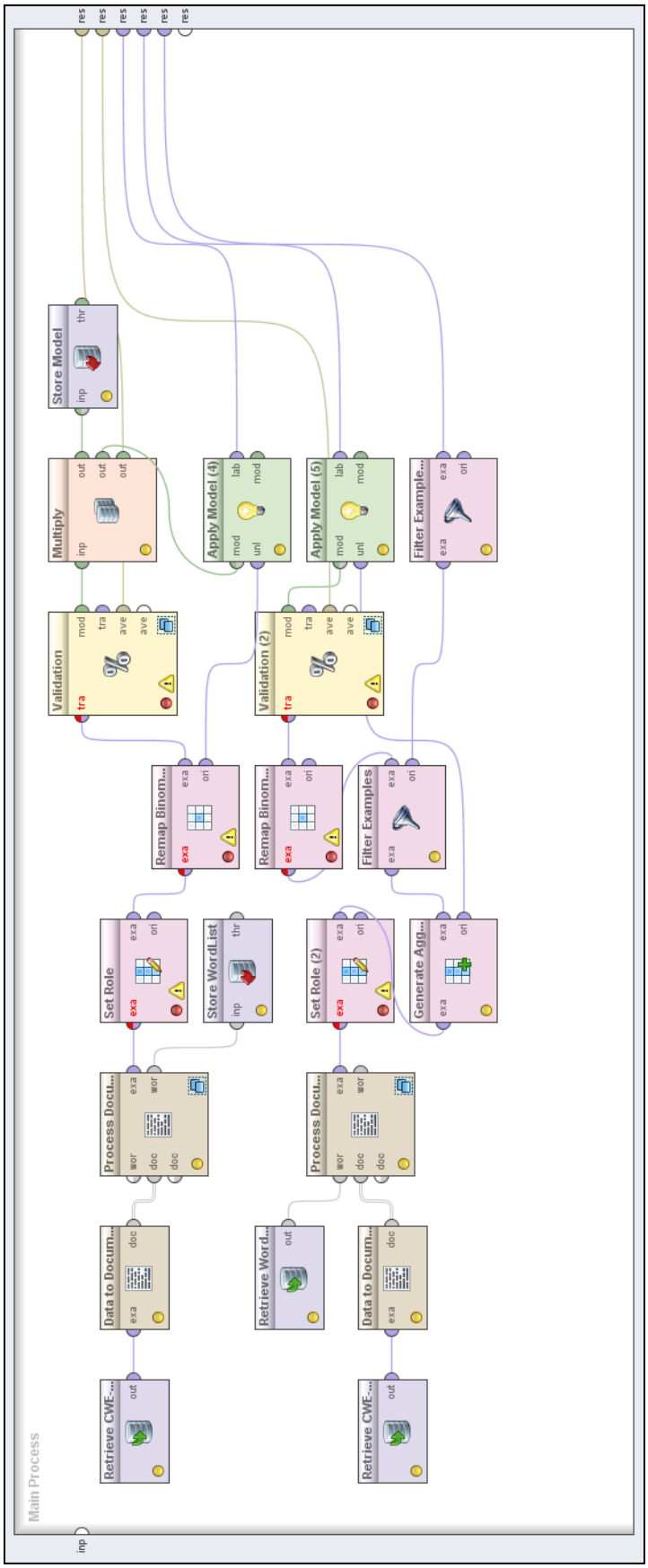

Abbildung B.1: RapidMiner Prozess

C. Inhalt der CD

Der gedruckten Version dieser Bachelorarbeit liegt eine CD mit folgendem Inhalt bei:

- Eine Version dieser Ausarbeitung im PDF und DOCX Format
- Die Tabellen der Stichprobe zur Worttrennung
- Alle Tabellen der unterschiedlichen Normalisierungsverfahren
- Der Quellcode:
  - für alle Schritte der Aufbereitung
  - zu den Evaluierungen der einzelnen Schritte
  - der genutzten Apache Tomcat Version
- Die Aufbereiteten Klassen aus Apache Tomcat
- Der Analyseprozess aus RapidMiner als XML-Datei

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 05.09.2014

_____

Marius Hilbich