BACHELORARBEIT

Herr
Max Wittig

Intelligentes Software Lizenz Clearing mithilfe von maschinellen Lernverfahren

2017

Fakultät Angewandte Computer- und
Biowissenschaften

BACHELORARBEIT

Intelligentes Software Lizenz Clearing mithilfe von maschinellen Lernverfahren

Autor:
Herr Max Wittig

Studiengang:
Medieninformatik und Interaktives Entertainment

Seminargruppe:
MI14w2-B

Erstprüfer:
Prof. Dr. rer. nat.habil. Thomas Haenselmann

Zweitprüfer:
M. Sc. Dipl.-Inf. (FH) Knut Altroggen

Einreichung:
25. November 2017

Faculty Applied Computer Sciences and
Biosciences

BACHELOR THESIS

Intelligent software license clearing using machine learning algorithms

Author:
Mr Max Wittig

Course of Studies:
Media Informatics and Interactive Entertainment

Seminar Group:
MI14w2-B

First Examiner:
Prof. Dr. rer. nat.habil. Thomas Haenselmann

Second Examiner:
M. Sc. Dipl.-Inf. (FH) Knut Altroggen

Date of Submission:
25. November 2017

Inhaltsverzeichnis

Abkürzungsverzeichnis

GPL GNU General Public License

AGPL GNU Affero General Public License

LGPL GNU Lesser General Public License

MIT Massachusetts Institute of Technology

BSD Berkeley Software Distribution

EULA End User Licence Agreement (Endbenutzervereinbarung)

IDE Integrated Developmenent Environment (Integrierte Entwicklungsumgebung)

CSV Comma-separated values (Komma getrennte Werte)

USA Vereinigte Staaten von Amerika

ID Eindeutige Nummer

Abbildungsverzeichnis

Tabellenverzeichnis

Glossar

Agent ist ein Teil einer Soft- oder Hardware, welcher die Fähigkeit besitzt im Auftrag eines Nutzers Aufgaben auszuführen.[1]

Copyright auch Urheberrecht genannt, gibt dem Autor das Recht über die Verwertung seines Werkes zu entscheiden.[2]

Framework ist eine Struktur, welche die Basis für eine Software bildet.[3]

GitHub ist eine Plattform, welches Entwicklern ermöglicht Quellcode zu veröffentlichen und miteinander zu kollaborieren.[4]

Interpreter „[...] ist ein Programm, dass die Anweisungen eines in einer anderen Programmiersprache als der des verwendeten Computers geschriebenen Programms sofort ausführt. “[5]

Jaccard Index vergleicht Mitglieder von Mengen, um herauszufinden welche Mitglieder der Mengen gleich oder unterschiedlich sind. Daraus ergibt sich eine prozentuelle Anzahl. Je größer dieser Index, desto ähnlicher sind die Mengen.[6]

Obfuskation ist die beabsichtigte Veränderung von Quellcode, sodass dieser für Menschen schwer zu verstehen, bzw. nicht lesbar wird.[7]

[1] Nwana, H. S. (1996). Software agents: an overview. (Kap. 4, S. 5–6). Cambridge University Press, vgl.

[2] urheberrecht.de. (2017). Urheberrecht und Urheberrechtsgesetz. Zugriff 1. Juli 2017 unter http://www.urheberrecht.de/, vgl.

[3] Oxford University Press. (2017). framework — Definition of framework in English by Oxford Dictionaries. Zugriff 12. November 2017 unter https://en.oxforddictionaries.com/definition/framework, vgl.

[4] Brown, K. (2016). What Is GitHub, and What Is It Used For? Zugriff 10. September 2017 unter https://www.howtogeek.com/180167/htg-explains-what-is-github-and-what-do-geeks-use-it-for/, vgl.

[5] Verlag Bibliographisches Institut GmbH. (2017a). Duden — Interpreter — Rechtschreibung, Bedeutung, Definition, Herkunft. Zugriff 7. September 2017 unter http://www.duden.de/rechtschreibung/Interpreter.

[6] statisticshowto. (2016). Jaccard Index / Similarity Coefficient. Zugriff 28. September 2017 unter http://www.statisticshowto.com/jaccard-index/, vgl.

[7] Wikipedia. (2017). Obfuscation (software) - Wikipedia. Zugriff 13. August 2017 unter https://en.wikipedia.org/wiki/Obfuscation_(software), vgl.

Overfitting ist die zu starke Anpassung eines Algorithmusses an Trainingsdaten.[8]

PDF „[...] ist ein Dateiformat, dass die Präsentation und den Austausch von Dokumenten unabhängig von Original-Software, Hardware oder Betriebssystem ermöglicht.“[9]

PHP ist eine Scriptsprache, welche sich besonders gut für die Programmierung von Webseiten eignet.[10]

PostgreSQL ist ein quelloffenes Datenbanksystem.[11]

Postscript ist ein Produkt von Adobe, welches Dokumente in Druck umwandelt.[12]

Spam „[...] unerwünschte massenhaft per E-Mail oder auf ähnlichem Wege versandte Nachrichten.“[13]

SQL ist eine Programmiersprache, welche genutzt wird um Daten in Datenbanken zu speichern, zu verändern und abzurufen.[14]

Underfitting ist die zu schwache Anpassung eines Algorithmusses an Trainingsdaten.[15]

Vendor lock-in trifft zu, wenn der Kunde von einem einzigem Hersteller abhängig ist und nicht ohne erhebliche Kosten oder viel Aufwand zu einem anderen Anbieter wechseln kann.[16]

[8] Puget, J. F. (2015). Overfitting In Machine Learning (IT Best Kept Secret Is Optimization). Zugriff 24. November 2017 unter https://www.ibm.com/developerworks/community/blogs/jfp/entry/Overfitting_In_Machine_Learning?lang=en, vgl.

[9] Adobe Systems Incorporated. (2017b). Was ist Adobe PDF? PDF, Adobe PDF — Adobe Acrobat DC. Zugriff 7. September 2017 unter https://acrobat.adobe.com/ch/de/why-adobe/about-adobe-pdf.html.

[10] The PHP Group. (2017). PHP: What is PHP? - Manual. Zugriff 9. Juli 2017 unter http://php.net/manual/en/intro-whatis.php, vgl.

[11] The PostgreSQL Global Development Group. (2017). PostgreSQL: About. Zugriff 14. Oktober 2017 unter https://www.postgresql.org/about/, vgl.

[12] Adobe Systems Incorporated. (2017a). Adobe PostScript. Zugriff 7. September 2017 unter http://www.adobe.com/products/postscript.html, vgl.

[13] Verlag Bibliographisches Institut GmbH. (2017b). Duden — Interpreter — Rechtschreibung, Bedeutung, Definition, Herkunft. Zugriff 26. September 2017 unter http://www.duden.de/rechtschreibung/Spam.

[14] Refsnes Data. (2017). SQL Tutorial. Zugriff 24. November 2017 unter https://www.w3schools.com/sql/, vgl.

[15] Puget, J. F. (2015). Overfitting In Machine Learning (IT Best Kept Secret Is Optimization). Zugriff 24. November 2017 unter https://www.ibm.com/developerworks/community/blogs/jfp/entry/Overfitting_In_Machine_Learning?lang=en, vgl.

[16] The Linux Information Project. (2006). Vendor lock-in definition by The Linux Information Project (LINFO). Zugriff 15. September 2017 unter http://www.linfo.org/vendor_lockin.html, vgl.

1 Einleitung

1.1 Problemstellung

Im kompletten Themengebiet der Softwarentwicklung ist es meist notwendig auf vorheriges Wissen bzw. vorherige Software aufzubauen. Damit entsteht neue Software, welche die Basis von zukünftiger Software sein könnte. Allgemein wird das in diesem Themengebiet als Abhängigkeit von anderer Software oder Softwarekomponenten bezeichnet. Um als Projektteam in einem Unternehmen eine Softwarekomponente benutzen zu können sind zahlreiche Schritte nötig. Diese beginnen bei der Anmeldung der Komponente für das Software Clearing. Dabei müssen zahlreiche Informationen über diese Komponente bzw. Applikationen, wie z.B. Autor bzw. Quelle des Codes manuell eingetragen werden. Danach wird der Quellcode der Software mit Lizenzscannern von Fossology gescannt und abschließend wird manuell über die Lizenz der Software entschieden. Bei kritischen Lizenzen wird die Entscheidung auf das Software Clearing Team Meeting vertagt. Dabei werden spezifische Softwarekomponenten durch mehrere Personen untersucht, welche Experten in diesem Gebiet sind. Es gibt bereits Lizenzscanner mit denen das Toolkit Fossology schon gute Ergebnisse erzielt. Jedoch ist die Entscheidung von Menschen immer noch notwendig, da die Ergebnisse nicht immer richtig sind und nur auf einfacher Mustererkennung beruhen. Dabei stellt sich die Frage, ob die Notwendigkeit einer menschlichen Klassifizierung von Lizenzen im Quellcode durch einen intelligenten Lizenzscanner aufgehoben werden kann?

1.2 Motivation

Die Wahl des Themas fiel auf die Umsetzung eines Lizenzscanners, welcher mit machinellen Lernverfahren funktioniert. Grund dafür ist die Relevanz von Softwarelizenzen und Softwarelizenz Clearing in der aktuellen Tätigkeit des Autors als Software Entwickler. Erst die Arbeit mit und an der Software Fossology und die ständige Berührung mit Softwarelizenzen in Open Source Applikationen hat den Verfasser auf dieses Thema aufmerksam gemacht.

Der zweite Bestandteil des Bachelorthemas ist maschinelles Lernen, welches einen Trend in der heutigen Softwareentwicklung darstellt und sich damit als praktikabel herausstellt. Die Kombination dieser beiden Themengebiete stellt eine neue Entwicklung dar.
Durch die Erarbeitung und Umsetzung dieser Bachelorarbeit in Bezug zu Softwarelizenzen ist es möglich Kenntnisse im Bereich der Softwareentwicklung und des Software Lizenz Clearings zu festigen und zu vertiefen. Dies war ebenfalls für die Wahl des Themas ausschlaggebend. Zudem ist es für den Arbeitgeber des Verfassers eine interessante Entwicklung, welche diesem ermöglicht weitere Schritte zur der Automatisierung von Software Lizenz Clearing zu unternehmen.

2 Grundlagen

2.1 Software Lizenz Clearing mit Fossology

2.1.1 Software Lizenz Clearing

Begriffserklärung

Software Lizenz Clearing konzentriert sich auf das Finden und Zusammenfassen von Software Lizenzen, Urheberrechten und Autoreninformationen in Quellcode. Dabei werden Lizenztexte, Verweise auf Lizenzen und lizenzrelevante Aussagen berücksichtigt.[17]

Lizenzen im Vergleich

Copyleft Lizenzen

Copyleft Lizenzen haben das Ziel Quellcode für jeden frei zugänglich zu machen, aber verlangen dabei gleichzeitig alle modifizierten oder erweiterten Versionen des Programms frei zugänglich zu machen. Dies soll verhindern, dass frei zugänglicher Quellcode zu geschlossener Software wird, sobald dieser eingesetzt und modifiziert wird. Damit garantiert das Copyleft die Freiheit des Quellcodes. Beispiele für Copyleft Lizenzen sind GNU General Public License (GPL), GNU Affero General Public License (AGPL) oder GNU Lesser General Public License (LGPL).[18]

Freizügige Lizenzen

Freizügige Lizenzen, auch Permissive Lizenzen genannt sind das Gegenteil von Copyleft Lizenzen, denn diese erlauben die Modifikation von Quellcode und die Benutzung in kommerziell verteilter Software ohne die Änderungen an der Software selbst veröffentlichen

[17] Jaeger, M. C. (2016). FOSSology SPDX in HD. Zugriff 15. Juni 2017 unter http://events.linuxfoundation.org/sites/events/files/slides/%28OCS_Mr.%20Michael%20Jaeger%29OCS-2016-Jaeger%20036%20OSS%20fossy%2020161113%20slides%20spdxHD%203.pdf, vgl.

[18] Free Software Foundation, Inc. (2017). What is Copyleft? - GNU Project - Free Software Foundation. Zugriff 29. September 2017 unter https://www.gnu.org/licenses/copyleft.en.html, vgl.

zu müssen. Beispiele für Permissive Lizenzen ist die Massachusetts Institute of Technology (MIT) Lizenz oder die Familie der Berkeley Software Distribution (BSD) Lizenzen.[19]

Gründe

Rechtlich ist es für eine Verwendung von quelloffener Software notwendig die Lizenzen der verwendeten Komponenten auszuweisen und Autoren- und Copyrightinformationen anzugeben. Besitzt die Open Source Komponente keine Lizenz, so behält der Autor das volle Urheberrecht und die Komponente kann nicht ohne Erlaubnis kommerziell von anderen eingesetzt werden. Wird dies trotzdem getan, ohne die Rechte des Autors beachtet zu haben, kann es zu rechtlichen Konsequenzen kommen. Die Nutzung einer Lizenz ermöglicht es, dass Softwarekomponenten trotzdem ohne Nachfrage von anderen genutzt werden können. Jedoch legen Softwarelizenzen häufig auch gewisse Restriktionen und Nutzungsbedingungen fest. Rechtliche Konsequenzen sind deshalb auch bei der Nutzung von Softwarelizenzen möglich, falls Verpflichtungen der Lizenz nicht wahrgenommen werden.

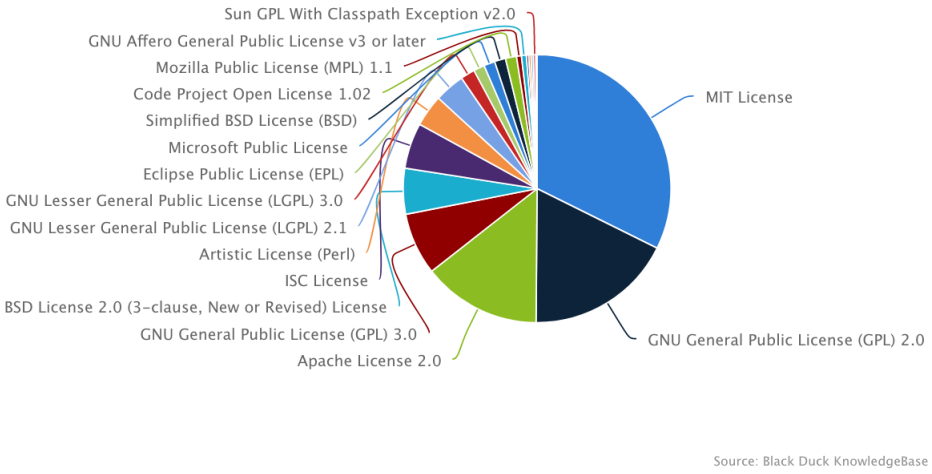

Abbildung 1: Verbreitung von Open Source Lizenzen

Quelle: Black Duck KnowledgeBase[20]

[19] The Open Source Initiative. (2017a). Frequently Answered Questions — Open Source Initiative. Zugriff 29. September 2017 unter https://opensource.org/faq#permissive, vgl.

[20] Black Duck KnowledgeBase. (2017a). Top Open Source Licenses. Zugriff 15. September 2017 unter https://www.blackducksoftware.com/sites/default/files/images/Top%5C%20Open%5C%20Source%5C%20Licenses.png.

Rang	Lizenz	Anteil in Prozent
1	MIT Lizenz	32
2	GPL 2.0	18
3	Apache License 2.0	14

Tabelle 1: Top Open Source Lizenzen (Auswahl)
Quelle: Black Duck KnowledgeBase[21]

Ein Beispiel für die Verletzung von Lizenzrechten findet sich in einem Gerichtsurteil, welches sich mit der Einhaltung der GPL Lizenz befasst. Die GPL Lizenz ist eine sehr weit verbreitete Softwarelizenz, welche in einer Statistik (siehe Abbildung 1 und Tabelle 1) von Black Duck KnowledgeBase aus einer Zusammenstellung von über zwei Millionen Open Source Projekten einen Anteil von 18 Prozent belegt und somit die zweithäufig genutzte Lizenz aus dieser Auswahl ist (Stand 2017).
Die GPL Lizenz in Version 2.0 legt gewisse Bestimmungen fest. Das Weiterverbreiten und das Modifizieren von Software unter der GPL-2.0 Lizenz ist erlaubt, solange Änderungen im Quellcode festgehalten und auch unter der GPL-2.0 Lizenz veröffentlicht werden.[22]
Bei Nichteinhaltung dieser Bedingungen kann ein Lizenzverstoß festgestellt werden. So auch in dem Urteil, welches die GPL-2.0 Lizenz behandelt. Das Urteil bezieht sich auf die Software Ghostscript, welche ein quelloffener Postscript und PDF Interpreter ist. Darin hat das Unternehmen Artifex, der Hersteller von Postscript gegen die Verwendung der GPL 2.0 lizenzierten Software durch Hancom geklagt. Hancom hielt sich jedoch nicht an den Lizenzvertrag der GPL-2.0 Lizenz, denn dieses Unternehmen nutzte Postscript in vielen seiner Softwareprodukte, ohne den Quellcode für diese offen zu legen. In diesem Fall hatte Artifex auch die Zahlung einer Lizenzgebühr als Alternative zu der GPL-2.0 Lizenzierung angeboten. Allerdings wurde auch dieses Angebot von Hamcom nicht angenommen. Das Ergebnis des Urteils lautet, dass die GPL Lizenz ein gültiger Vertrag sei und Hancom somit rechtlich schuldig.[23]
Dieses Urteil ist jedoch nicht das Einzige in diesem Bereich. Auch ein deutsches Gericht hat die Wirksamkeit der GPL Lizenz mit einem ähnlichen Fall am 19. Mai 2004 bestätigt. Damit sind Softwarelizenzen als ernstzunehmende Verträge angesehen, welche geachtet und erfüllt werden sollten.[24]

Verfahrensweise

Ausgangslage

Damit die Einhaltung von Lizenzbestimmungen bei quelloffener Software möglich ist, müssen zunächst die

[21] Black Duck KnowledgeBase. (2017b). Top Open Source Licenses. Zugriff 15. September 2017 unter https://www.blackducksoftware.com/top-open-source-licenses.

[22] FOSSA, Inc. (2012). GNU General Public License v2.0 (GPL-2.0) Explained in Plain English - TLDRLegal. Zugriff 13. August 2017 unter https://tldrlegal.com/license/gnu-general-public-license-v2, vgl.

[23] Collins, K. (2017). A federal court has ruled that an open-source license is an enforceable contract. Zugriff 17. Juni 2017 unter https://qz.com/981029/a-federal-court-has-ruled-that-an-open-source-license-is-an-enforceable-contract/; Perens, B. (2017). Understanding the 'GPL is a Contract' court case. Zugriff 7. September 2017 unter https://perens.com/blog/2017/05/28/understanding-the-gpl-is-a-contract-court-case/, vgl.

[24] Bleich, H. (2004). Deutsches Gericht bestätigt Wirksamkeit der GPL. Zugriff 7. September 2017 unter https://www.heise.de/newsticker/meldung/Deutsches-Gericht-bestaetigt-Wirksamkeit-der-GPL-101616.html, vgl.

Lizenzen und deren Anforderungen gefunden werden. Dieser Prozess heißt Open Source Clearing. Dieses Clearing ist nur notwendig, wenn es sich um Open Source Komponenten handelt, welche nicht schon mit einem bestehendem Lizenzvertrag zwischen dem Urheber und dem Nutzer der Software geregelt sind. Jedoch wird Open Source Software Clearing bei der Nutzung von quelloffenen Komponenten durchaus benötigt.

Feststellung von Lizenzen

Um die Lizenzen von Software feststellen zu können kann auf verschiedene Faktoren geachtet werden. Zum Beispiel kann es sich um ein Open Source Projekt von GitHub oder einer ähnlichen Plattform handeln. GitHub ist eine große Plattform, welches jedem registrierten Benutzer ermöglicht, Quellcode hochzuladen, Quellcode von anderen einzusehen oder diesen durch Modifikationen zu verändern. Doch nur weil dieser Quellcode öffentlich einsehbar ist, bedeutet es trotzdem nicht, dass der Autor alle Rechte an diesem verliert. Auf dieser Plattform sind Software Lizenzen gekennzeichnet, wenn der Autor diese beilegt.

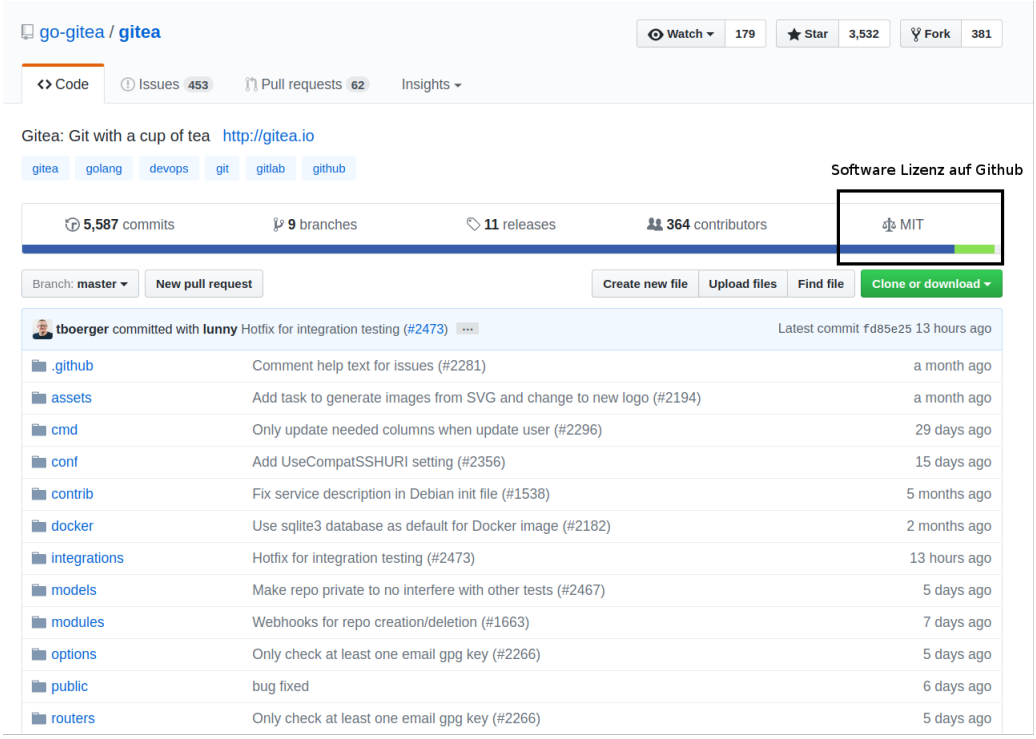

Abbildung 2: Gitea auf GitHub

Quelle: Bildschirmfoto von https://github.com/go-gitea/gitea

In Abbildung 2 ist ein beliebiges Open Source Projekt von GitHub zu sehen. Dort wird die Lizenz direkt abgebildet und ist für den Nutzer gut sichtbar. In diesem Fall ist dies die MIT Lizenz. Diese Anzeige in der Benutzeroberfläche von GitHub wird generiert, wenn in einem Open Source Projekt eine Datei mit dem Namen „LICENSE“ existiert. Das heißt, dass Softwareprojekte und seine Mitwirkenden entschieden haben eine gewisse Lizenz zu nutzen. Wurde bei einem Projekt auf GitHub keine Lizenz beigelegt, ist

davon auszugehen, dass das Urheberrecht des Autors weiterhin vollständig besteht.[25] Allerdings ist es nicht möglich den Angaben von Lizenzen auf GitHub und anderen Plattformen vollständig zu vertrauen. Dies folgt daraus, dass Software oft nicht von Grund auf neu erschaffen wird, sondern in den meisten Fällen von anderer Software abhängig ist, welche andere Lizenzbedingungen haben könnte. Diese könnten sogar inkompatibel mit der eigentlich vom Autor vorgesehenen Lizenz für seine Software sein. Das würde dann eine Lizenzverletzung darstellen.
Lizenzen sind miteinander kompatibel, wenn die Bedingungen der kombinierten Lizenzen bei einer Distribution zu einem größerem Werk immer noch eingehalten werden können. Dabei gibt es besondere Bestimmungen, welche für Lizenzen gelten, zum Beispiel dass nur Teile zusammen genutzt werden können, aber nicht der Quellcode direkt. Da die Installation von getrennten Programmen auf einem System diese nicht zu einem größerem Werk vereint, gilt in diesem Fall zum Beispiel keine Lizenzinkompatibilität.[26]
Inkompatibel sind z.B. die GPL-3.0 Lizenz und die GPL-2.0 Lizenz, weil beide Lizenzen den Copyleft Effekt aufeinander haben würden.[27]
Deshalb wird idealerweise Quellcode von Open Source Projekten nicht nur auf die Datei mit dem Namen LICENSE und dieser angegebenen Lizenz untersucht, sondern die Lizenzen werden für alle einzelne Dateien im Projekt festgestellt, welche kommerziell weiterbenutzt werden sollen. Die Erkennung von Lizenzen für jede Datei wird durch ein Kommentar im Kopf des Quellcodes möglich, welche eine Lizenz enthalten sollte. Im nachfolgenden auch Lizenzkopf genannt. Dabei können Dateien, welche z.B. nur zum Erstellen von Softwarekomponenten eingesetzt werden bei der Lizenzierung außer acht gelassen werden.

[25] Black Duck Software, Inc. (2017). Open Source License Compliance. Zugriff 24. Juni 2017 unter https://www.blackducksoftware.com/solutions/open-source-license-compliance; Brown, K. (2016). What Is GitHub, and What Is It Used For? Zugriff 10. September 2017 unter https://www.howtogeek.com/180167/htg-explains-what-is-github-and-what-do-geeks-use-it-for/, vgl.

[26] Free Software Foundation. (2017a). Frequently Asked Questions about the GNU Licenses - GNU Project - Free Software Foundation. Zugriff 5. November 2017 unter https://www.gnu.org/licenses/license-list.html#GPLCompatibleLicenses, vgl.

[27] Free Software Foundation. (2017b). Various Licenses and Comments about Them - GNU Project - Free Software Foundation. Zugriff 5. November 2017 unter https://www.gnu.org/licenses/gpl-faq.en.html#WhatIsCompatible, vgl.

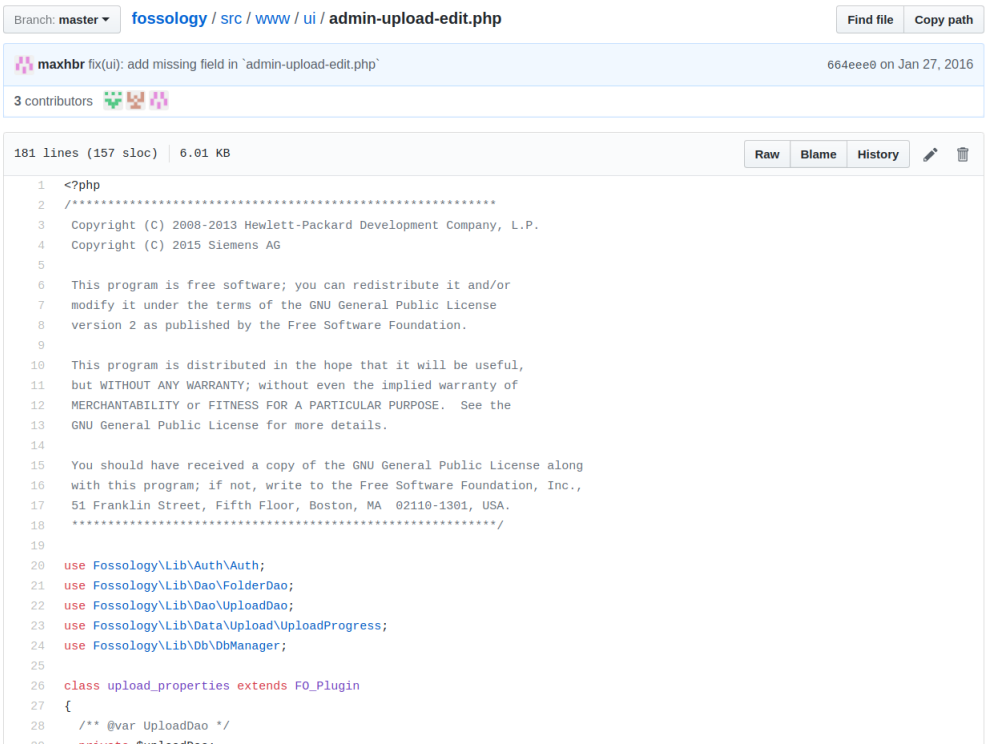

Abbildung 3: Ein GPL-2.0 Lizenzkopf
Quelle: Bildschirmfoto von
https://github.com/fossology/fossology/blob/master/src/www/ui/admin-upload-edit.php

In Abbildung 3 ist ein Kommentar in einer PHP Quellcodedatei sichtbar, welches die notwendigen Lizenzinformationen enthält. In diesem Beispiel enthält der Lizenzkopf den Lizenztext, Copyright und Informationen zum Autor der Datei. Diese Informationen reichen meist, um die Lizenz und deren verknüpfte Verpflichtungen dieser Datei festzustellen.

2.1.2 Open Source Software

Laut der Open Source Initiative muss Software einige Kriterien erfüllen, um als quelloffene Software gelten zu können. Die erste Bedingungen, welche vorgegeben wird, ist die freie Verbreitung von Software als Komponente. Dabei dürfen keine Restriktionen bezüglich dem Verkauf bzw. der Distribution gemacht werden.
Des Weiteren muss die Distribution des Programms in Quellcode, als auch in kompilierter Form erlaubt sein. Falls keine direkte Verteilung von Quellcode mit dem Produkt vorgesehen ist, sollte eine Verteilung auf Anfrage möglich sein. Dabei sollten nicht mehr als die Reproduktionskosten bzw. Kosten des Mediums angerechnet werden. Zudem sollte der Quellcode in roher Form vorliegen, das heißt ohne Code-Obfuskation oder andere Veränderungen, welche die Lesbarkeit beeinträchtigen würden.
Die Lizenz sollte die Verbreitung von Software erlauben, welche vom Quellcode der Software erstellt wurde. Diese kann dabei eine Änderung der Versionsnummer oder des Namens fordern.

Auch legt die Open Source Initiative fest, dass Software keine Diskriminierung gegen Personen, Gruppen oder andere Einsatzbereiche definieren darf. Beispielsweise sollte sie nicht nur von Firmen einsetzbar sein, sondern für alle Personen universell zugänglich sein. Zusätzlich sollte bei der Verteilung der Software auch die Lizenz übernommen werden. Ebenfalls sollten keine produktspezifischen Einschränkungen in Lizenzen enthalten sein und die Lizenz der Software sollte sich nicht auf nur eine Technologie wie beispielsweise Programmiersprache beschränken.[28]

Verwendung

Quelloffene Software werde laut Blackduck von 78 Prozent aller Unternehmen teilweise oder komplett genutzt. Zusätzlich geben 66 Prozent der Unternehmen an, dass sie Software für Endkunden produzieren würden, welche quelloffene Komponenten enthalte.[29]

Gründe

Es gibt viele verschiedene Gründe für Unternehmen quelloffene Software zu nutzen. Laut einer Umfrage von Blackduck glauben viele Firmen an verbessere Sicherheit durch quelloffene Software und eine bessere Skalierung auf eine größere Anzahl von Nutzern. Zusätzlich kann es eine persönliche Präferenz von Entwicklern des Unternehmens zu der Nutzung von quelloffener Software geben. Jedoch sind auch andere Faktoren für Unternehmen entscheidend.[30]

Zuverlässigkeit

Dadurch, dass bei Open Source Software der Quellcode für jeden zugänglich ist, besteht eine erhöhte Wahrscheinlichkeit, dass Probleme einer Software schneller behoben werden können. Jeder Einzelne kann die Software auf Fehler im Quellcode untersuchen, diese melden bzw. bei erforderlicher Kompetenz in dem Gebiet der Softwareentwicklung auch selbst beheben. Zusätzlich ist es für Unternehmen von Vorteil, Aktualisierungen von Software sofort zu nutzen, ohne auf eine offizielle Veröffentlichung zu warten, indem diese zum Beispiel selbst erstellt wird.
Dies steht im Gegensatz zu geschlossener Software. Bei dieser muss meist zuerst eine Fehlermeldung an das Unternehmen übermittelt werden, welches die Software betreut bzw. entwickelt. Auch kann ein auf diese Software angewiesenes Unternehmen nicht selbst eingreifen, sondern muss auf eine offizielle Verteilung der neuen Version der Software warten.[31]

Stabilität

Aufgrund von Fehlern, aktualisierten Anforderungen und eventuell auftretenden Sicherheitslücken ist die kontinuierliche Weiterentwicklung von Software in Form von neuen Versionen notwendig. Bei kommerzieller,

[28] The Open Source Initiative. (2017b). The Open Source Definition (Annotated) — Open Source Initiative. Zugriff 15. Juni 2017 unter https://opensource.org/osd-annotated, vgl.

[29] Vaughan-Nichols, S. J. (2005). It's an open-source world: 78 percent of companies run open-source software — ZDNet. Zugriff 11. September 2017 unter http://www.zdnet.com/article/its-an-open-source-world-78-percent-of-companies-run-open-source-software/, vgl.

[30] Vaughan-Nichols, 2005, vgl.

[31] GBdirect. (2016). Benefits of Using Open Source Software. Zugriff 11. September 2017 unter http://open-source.gbdirect.co.uk/migration/benefit.html; Hartley, M. (2015). Why Use Open Source Software? - Datamation. Zugriff 11. September 2017 unter http://www.datamation.com/open-source/why-use-open-source-software-1.html, vgl.

geschlossener Software werden beispielsweise auch neue Dateiformate eingeführt, welche inkompatibel zu den älteren Dateien sind. Dies stellt ein Problem dar, da es durchaus vorkommen kann, dass der Betreiber der kommerziellen Software den Nutzern keine Möglichkeit bietet auf dieses neue Format umzusteigen. Bei quelloffener Software könnte jeder, aufgrund der offenen Natur der Software z.B. ein Konvertierungsprogramm erstellen und dies veröffentlichen. Auch das Unternehmen könnte sich entscheiden die ältere Version der Software weiter zu unterstützen, ohne auf den ursprünglichen Ersteller der Software angewiesen zu sein.[32]

Nachvollziehbarkeit

Da Open Source Software von jedem untersuchbar ist, muss dem Herausgeber der Software nicht getraut werden. Die Sicherheit der Software muss beispielsweise nicht auf bestmöglichem Stand sein, denn der Quellcode kann unabhängig darauf untersucht werden und eventuelle Fehler können verbessert werden. Im Gegensatz dazu steht geschlossene Software. Bei dieser wird dem Anwender keine Wahl gelassen. Da der Quellcode nicht einsehbar ist, muss der geschlossenen Software getraut werden, wenn man sie benutzen möchte. Es ist nicht einfach möglich, dass Unternehmen selbst die Sicherheit beurteilen oder beeinflussen können.[33]

Kosten

Die meist verfügbare quelloffene Software ist, sowohl von Kosten als auch von Lizenzgebühren befreit. Das bezieht sich meist nicht nur auf den Kaufpreis, sondern auch auf die finanziellen Ausgaben von Aktualisierungen und die längere Haltbarkeit der Software durch Pflege der Gemeinschaft.[34]

Flexibilität und Freiheiten

Flexibilität für Nutzer von quelloffener Software wird beispielsweise dadurch bereitgestellt, dass ein sogenannter „Vendor lock-in“ zu festgelegten Anbietern oder Technologien häufig vermieden werden kann. Quelloffene Softwareprojekte haben sehr selten als Gemeinschaft die Absicht Individuen in das Ökosystem einzuschließen, da dies für die Projekte keine kommerziellen Vorteile hätte.
Im Gegensatz dazu steht geschlossene Software. Beispielweise könnte ein Unternehmen als Anbieter einer geschlossenen Softwarelösung seinen Kunden die Möglichkeiten nehmen auf einen anderen Anbieter zu wechseln. Diese Strategie würde die Kunden fest an einen Anbieter binden, welches für das Unternehmen eine konstante Einnahmequelle für beispielsweise Lizenzgebühren darstellen könnte. Auch können dadurch weitere Nachteile entstehen, falls der Anbieter die Entwicklung der Software einstellt. Dies würde dann dazu führen, dass ein Unternehmen, welches Leistungen vom Anbieter bezogen hat nicht mehr auf Daten des alten Programms zugreifen kann, etwa bei einer Aktualisierungen des kompletten Systems, welches die Inkompatibilität der Software bewirkt. Bei geschlossener Software wäre dies problematisch, da es häufig nicht möglich ist diese Daten aus der alten Software in eine Alternative zu portieren.[35]

[32] GBdirect. (2016). Benefits of Using Open Source Software. Zugriff 11. September 2017 unter http://open-source.gbdirect.co.uk/migration/benefit.html; Hartley, M. (2015). Why Use Open Source Software? - Datamation. Zugriff 11. September 2017 unter http://www.datamation.com/open-source/why-use-open-source-software-1.html, vgl.

[33] GBdirect, 2016; Hartley, 2015, vgl.

[34] GBdirect, 2016; Hartley, 2015, vgl.

[35] GBdirect, 2016; Hartley, 2015, vgl.

Unterstützung und Verantwortlichkeit

Bei der Frage nach der Verantwortung im Fall eines Fehlers ist es denkbar, dass dies bei geschlossener Software besser geregelt ist. Allerdings hat häufig sowohl kommerzielle als auch freie Software eine Form von „End User Licence Agreement (Endbenutzervereinbarung) (EULA)“. Darin wird häufig jede Verantwortlichkeit für Schäden bei Benutzung der Software abgegeben. Somit gibt es bei geschlossener Software meist in diesem Aspekt keine Vorteile gegenüber freier Software. Auch bei der Betrachtung von direkter Unterstützung nach dem Kauf unterscheiden sich offene Komponenten nicht immer von kommerziellen. Einzelfälle gibt es, indem Autoren von kommerzieller Software auch direkte Unterstützung bereitstellen. Allerdings kann dies auch bei Open Source Programmen zutreffen.[36] Ein nennenswertes Beispiel dafür ist das Redhat Linux Betriebssystem. Das Unternehmen Redhat hat sich dazu entschieden 100 Prozent der kommerziell verfügbaren Produkte, Open Source zu halten und auch beim Einkauf von anderen Firmen Software deren quelloffen zu veröffentlichen. Das Geschäftsmodell von Redhat basiert auf quelloffener Software. Redhat erzielt Einnahmen, welche durch die direkte Unterstützung der Software für Wartung, Installation und genereller Unterstützung entstehen.[37]

Potenzielle Nachteile von quelloffener Software

Bedienbarkeit

Da bei quelloffener Software kein kommerzielles Produkt entstehen soll, könnte der Fokus der Entwicklung eher auf die Wünsche der Entwickler und weniger auf den Ausbau der Benutzeroberfläche und Optimierung der Bedienbarkeit fallen.[38]

Software Lizenzierung

Auch wenn die Benutzung von Open Source Software nicht direkt mit Kosten verbunden ist, könnten Kosten durch die nötigen Weiterentwicklungen oder die Unterstützung von Nutzern entstehen. Auch manche Softwarelizenzen, wie zum Beispiel die GPL Lizenz fordert gewisse Bedingungen, welche etwa bei dem Anpassen der Software an eigene Bedürfnisse bei gleichzeitiger kommerzieller Verwendung entstehen.[39]

Offenheit von Schwachstellen

Bei quelloffener Software werden häufig Fehler von der Gemeinschaft gefunden und behoben. Diese Fehler können auch kritische Sicherheitsschwachstellen sein. Dies würde beim Einsatz von anfälliger Software konstante Aktualisierungen mit dem Quellcode erfordern, um die Sicherheit zu gewährleisten. Denn anders als bei den meisten kommerziellen Lösungen, sind Fehler meist in der gesamten Entwicklungsgemeinschaft schon bekannt. Bei Vernachlässigung von Aktualisierungen könnten böswillige Nutzer Schwachstellen in

[36] GBdirect. (2016). Benefits of Using Open Source Software. Zugriff 11. September 2017 unter http://open-source.gbdirect.co.uk/migration/benefit.html; Hartley, M. (2015). Why Use Open Source Software? - Datamation. Zugriff 11. September 2017 unter http://www.datamation.com/open-source/why-use-open-source-software-1.html, vgl.

[37] Cormier, P. (2017). What makes us Red Hat. Zugriff 12. September 2017 unter https://www.redhat.com/de/blog/what-makes-us-red-hat; Levine, P. (2014). Why There Will Never Be Another RedHat: The Economics Of Open Source — TechCrunch. Zugriff 12. September 2017 unter https://techcrunch.com/2014/02/13/please-dont-tell-me-you-want-to-be-the-next-red-hat/, vgl.

[38] Bridge, R. (2013). Open source software - The advantages & disadvantages. Zugriff 12. September 2017 unter http://entrepreneurhandbook.co.uk/open-source-software/, vgl.

[39] Bridge, 2013, vgl.

der Software ausnutzen.[40]

2.1.3 Bestandteile von Fossology

Übersicht

Fossology ist ein Toolkit für Software Lizenz Clearing, das heißt es bietet viele Funktionen welche Software Lizenz Clearing ermöglichen sollen.

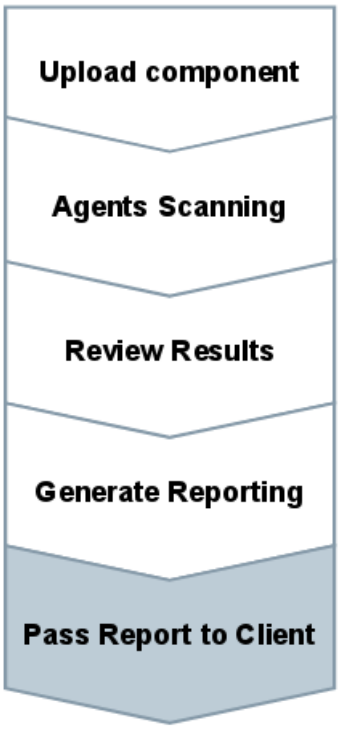

Abbildung 4: Funktionsweise von Fossology

Quelle: FOSSology SPDX in HD[41]

Hochladen von Komponenten

In Abbildung 4 ist der Arbeitsablauf von Fossology zu sehen. Der Ablauf beginnt, indem der Nutzer eine Komponente in Fossology hochlädt. Komponenten können Archive mit mehreren Dateien oder einzelne

[40] Bridge, R. (2013). Open source software - The advantages & disadvantages. Zugriff 12. September 2017 unter http://entrepreneurhandbook.co.uk/open-source-software/, vgl.

[41] Jaeger, M. C. (2016). FOSSology SPDX in HD. Zugriff 15. Juni 2017 unter http://events.linuxfoundation.org/sites/events/files/slides/%28OCS_Mr.%20Michael%20Jaeger%29OCS-2016-Jaeger%20036%20OSS%20fossy%2020161113%20slides%20spdxHD%203.pdf.

Dateien sein, welche Quellcode beinhalten.[42]

Scannen mit Agents

Software Agents können den im vorherigen Schritt hochgeladenen Quellcode auf Wunsch des Nutzers direkt nach Lizenzen, Copyright oder auch Exportkontroll- und Zollrestriktionen untersuchen.[43]

Nachprüfen der Ergebnisse

In diesem Schritt werden die Ergebnisse der diversen Agents überprüft und eventuelle Fehler manuell korrigiert.[44]

Exportieren der Resultate

Mit dem Exportieren der Resultate bzw. Weitergabe der Auswertung des Lizenz Clearings ist der Arbeitsablauf beendet. Exportiert werden können jetzt SPDX, Hinweis und Liesmich Dateien, sowie spezielle Copyright Formate für das Debian Paketverzeichnis.[45]

Vorraussetzungen

Fossology benötigt für den optimalen Betrieb eine Webserver Installation. Diese besteht aus einem Linux Betriebssystem, dem Webserver Apache2, dem Datenbanksystem PostgreSQL und der Programmiersprache PHP.

Oberfläche

Die Weboberfläche von Fossology ist in PHP geschrieben und benutzt das Symfony PHP Framework. Zusätzlich gibt es noch Werkzeuge für die Kommandozeile, welche mit der Bedienung der Agents helfen. Diese sind in den Programmiersprachen C, C++ und PHP geschrieben.

2.2 Maschinenlernen

2.2.1 Begriffserklärung

Es gibt verschiedene Definitionen was der Begriff Maschinenlernen bedeutet. Eine mögliche Definition kommt von Arthur Samuel.

> [Machine learning is the] field of study that gives computers the ability to learn without being explicitly programmed. (Arthur Samuel, 1959)

Arthur Samuel gilt als einer der Pioniere im Feld der künstlichen Intelligenz. Aus diesem Grund erhielt er 1987 den Computer Pionier Preis für anpassende, nicht nummerische Berechnungen. Die Basis für den Preis waren seine Ergebnisse zur Forschung über künstliche Intelligenz. Er konnte damit eine künstliche

[42] Jaeger, M. C. (2016). FOSSology SPDX in HD. Zugriff 15. Juni 2017 unter http://events.linuxfoundation.org/sites/events/files/slides/%28OCS_Mr.%20Michael%20Jaeger%29OCS-2016-Jaeger%20036%20OSS%20fossy%2020161113%20slides%20spdxHD%203.pdf, vgl.

[43] Jaeger, 2016, vgl.

[44] Jaeger, 2016, vgl.

[45] Jaeger, 2016, vgl.

Intelligenz konstruieren, welche gegen den viertbesten Damespieler in den Vereinigte Staaten von Amerika (USA) 1961 gewinnen konnte. Das Programm lernte dabei selbstständig welche Bewegungen zum Gewinn führen und welche sich eher negativ auswirken.[46]

2.2.2 Arten von Maschinenlernverfahren

Überwachtes Lernen

Überwachtes Lernen ist ein Maschinenlernverfahren, welches Eingangsvariablen (x) mit Ausgabevariablen (Y) mithilfe eines Algorithmus verbinden kann.

$$Y = f(x) \tag{2.1}$$

Aber überwachtes Lernen kann erneut in Klassifikations- und Regressionsprobleme unterteilt werden.[47]

Klassifikation

Ein Klassifikationsproblem ist dann vorhanden, wenn die Ausgabevariablen verschiedene Kategorien abbilden sollen. Ein Beispiel ist eine binäre Kategorisierung wie Ja/Nein, wahr/falsch.[48]

Regression

Regressionsprobleme im Bereich des überwachtem Lernen liegen vor, wenn die Ausgabevariablen einen realen Wert darstellen, wie zum Beispiel ein Gewicht oder eine Menge.[49]

Unüberwachtes Lernen

Bei unüberwachtem Lernen wird auf die Vorgabe von Ausgabevariablen verzichtet, das heißt dem Algorithmus werden nur Eingangsvariablen, jedoch keine Ergebnisse übergeben. Ein Beispiel dafür wäre die Erkennung von Themengebieten in Blogs. Dabei muss eine Zusammenfassung des Textes stattfinden, aber vorher ist nicht bekannt welche Themen es geben wird. Deshalb kann es keine bekannten Ausgabevariablen geben.[50]

Bestärkendes Lernen

Bestärkendes Lernen nutzt Beobachtungen, welche von Interaktionen mit der unmittelbaren Umgebung getan werden, um die Belohnung zu maximieren bzw. das Risiko zu verringern. Dies bedeutet, dass Entscheidungen die zu Aktionen des intelligenten Programmes werden, einige Schritte durchlaufen müssen. Der erste Schritt beginnt damit, dass der Eingangszustand von einem sogenannten Agent untersucht wird. Dieser führt im zweiten Schritt eine Funktion aus und observiert im dritten Schritt die Ergebnisse. Dieser

[46] IEEE. (2017). Arthur L. Samuel • IEEE Computer Society. Zugriff 15. September 2017 unter https://www.computer.org/web/awards/pioneer-arthur-samuel, vgl.

[47] Brownlee, J. (2016). Supervised and Unsupervised Machine Learning Algorithms. Zugriff 26. September 2017 unter https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/, vgl.

[48] Brownlee, 2016, vgl.

[49] Brownlee, 2016, vgl.

[50] Guido, S. und Müller, A. (2016). *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly, vgl.

Agent bekommt dann von der Umgebung eine Belohnung. Informationen über die Belohnung werden anschließend gespeichert. Die gespeicherten Informationen ermöglichen bei erneuter Durchführung zu optimaleren Entscheidungen zu kommen.[51]

2.2.3 Anwendungsgebiete

Spracherkennung

Spracherkennung ermöglicht das Verstehen von menschlicher Sprache, so dass Computer diese erkennen und zu Text verarbeiten können. Einige Anbieter von Spracherkennungssystemen verstehen befehlsartige Anweisungen und können damit Aktionen durchführen oder dem Nutzer Informationen mitteilen. In naher Zukunft wird sich die Genauigkeit der Spracherkennung durch weitere Entwicklungen vermutlich noch erhöhen.[52]

Bilderkennung

Bilderkennung durch Algorithmen hat zahlreiche Anwendungsgebiete. Ein Beispiel dafür sind selbstfahrende Kraftfahrzeuge, welche durch die Existenz von ausreichenden Fähigkeiten der automatischen Bilderkennung erst möglich werden. Diese Fahrzeuge nutzen mehrere externe Sensoren um andere Fahrzeuge und Hindernisse zu erkennen. Auch in der Sicherheitstechnologie wird die intelligente Bilderkennung genutzt. Etwa am Berliner Bahnhof wurde 2017 eine Gesichtserkennung anhand von Freiwilligen getestet. Dies soll dazu genutzt werden Straftaten und Gefahrensituationen zu erkennen.[53]

Text Mining

Text Mining nutzt Maschinenlernverfahren, um Informationen aus Text zu extrahieren. Anwendungen dabei sind zum Beispiel die automatische Klassifizierung von Nachrichten. Diese könnten in verschiedene Relevanzen oder Kategorien unterteilt werden. Auch bei der E-Mail Filterung wird Text Mining genutzt, denn dadurch ist es möglich Spamnachrichten zu erkennen und diese von anderen E-Mails zu trennen. Weitere sind zum Beispiel geschäftliche Anwendungen, Sicherheit oder auch die Überwachung von Sozialen Medien.
Zusätzlich ist es möglich die Erkennung von Bildern und Texten zu verbinden. Dies wird zum Beispiel bei der Handschrifterkennung eingesetzt.[54]

[51] Guido, S. und Müller, A. (2016). *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly, vgl.

[52] Mohammed, M., Khan, M. und Bashier, E. (2016). *Machine Learning: Algorithms and Applications*. CRC Press, vgl.

[53] Borchers, D. (2017). Gesichtserkennung per Video: Bundespolizei sucht Berliner Freiwillige. Zugriff 26. September 2017 unter https://www.heise.de/newsticker/meldung/Gesichtserkennung-per-Video-Bundespolizei-sucht-Berliner-Freiwillige-3746657.html; Mohammed, M., Khan, M. und Bashier, E. (2016). *Machine Learning: Algorithms and Applications*. CRC Press, vgl.

[54] Mohammed et al., 2016, vgl.

3 Rahmenbedingungen

3.1 Ausgangssituation

Zu Beginn des Bearbeitungszeitraumes enthält die Software Fossology ca. 7500 Kontributionen auf der öffentlichen Seite des Projektes bei der Plattform GitHub (siehe https://github.com/fossology/fossology) und liegt in Version 3.1 in stabiler Fassung vor. Fossology enthält die zwei Lizenzscanner Monk und Nomos wobei es möglich ist den Lizenzscanner Ninka optional zu installieren. In dem Lizenztoolkit Fossology sind damit noch keine Lizenzscanner, welche mit maschinellen Lernverfahren ausgestattet sind. Damit kann Fossology nicht automatisch aus vorhandenen Lizenzentscheidungen lernen.

3.2 Zielsetzung

Die Zielsetzung wird in softwaretechnische Ziele und in zukunftsbezogene Ziele eingeteilt, welche nach dem Abschluss dieser Bachelorarbeit verfolgt werden können.

3.2.1 Softwaretechnische Ziele

Das Ziel dieser Bachelorarbeit ist eine Implementierung eines Lizenzscanners, welcher anhand von bestehenden Entscheidungen für Lizenzen von Quellcode lernt und damit neue Dateien mit einer Lizenz klassifizieren kann. Am Ende des Bearbeitungszeitraumes soll ein funktionsfähiger unabhängiger Lizenzscanner entstehen. Dieser kann mit vorher extrahierten Daten aus der Fossology Lizenzdatenbank angelernt und überprüft werden.

3.2.2 Zukunftsbezogene Ziele

Der Lizenzscanner kann anschließend nach Wunsch des betreuendes Unternehmens nach dem Bearbeitungszeitraum in die quelloffene Software Fossology integriert werden. Bevor dieser Aufwand jedoch betrieben wird, welcher auch das Design einer Benutzeroberfläche einschließen würde, wird zunächst nur der Lizenzscanner unabhängig von Fossology betrachtet, um dessen Leistung bei der Lizenzerkennung einschätzen zu können. Bei der Verwendung von Fossology soll dabei der Lizenzscanner in Zukunft bei Bedarf zusätzlich oder ergänzend zu den bestehenden Lizenzscannern ausgewählt werden können. Außerdem kann es aus technischer Sicht möglich sein diesen Lizenzscanner eine Lizenzentscheidung treffen zu lassen, wenn etwa das Ergebnis mit einem in Fossology etablierten Scanner, also Nomos oder Monk übereinstimmt. Aber auch, wenn dieser nicht ausgewählt ist könnte der Lizenzscanner von den Entscheidungen der menschlichen Lizenzentscheidungen lernen, um den Algorithmus im Laufe der Zeit zu verbessern. Dadurch ist es möglich den Lizenzerkennungvorgang stärker zu automatisieren und zu vereinfachen und den maschinenlernenden Lizenzscanner stetig in einer Produktionsumgebung zu verbessern.

3.3 Zeitplanung

Damit die Bearbeitungszeit eingehalten werden kann, ist eine ungefähre Zeitplanung im Voraus nötig. Diese ist als Gantt Diagramm realisiert. Dabei wird lediglich die reine Bearbeitungszeit aufgezeigt. Beachtet werden muss jedoch auch die Zeit der Themenfindung, Quellensuche und der Einrichtungszeit der Entwicklungs- und Schreibumgebung,
welches vor dieser stattfand.

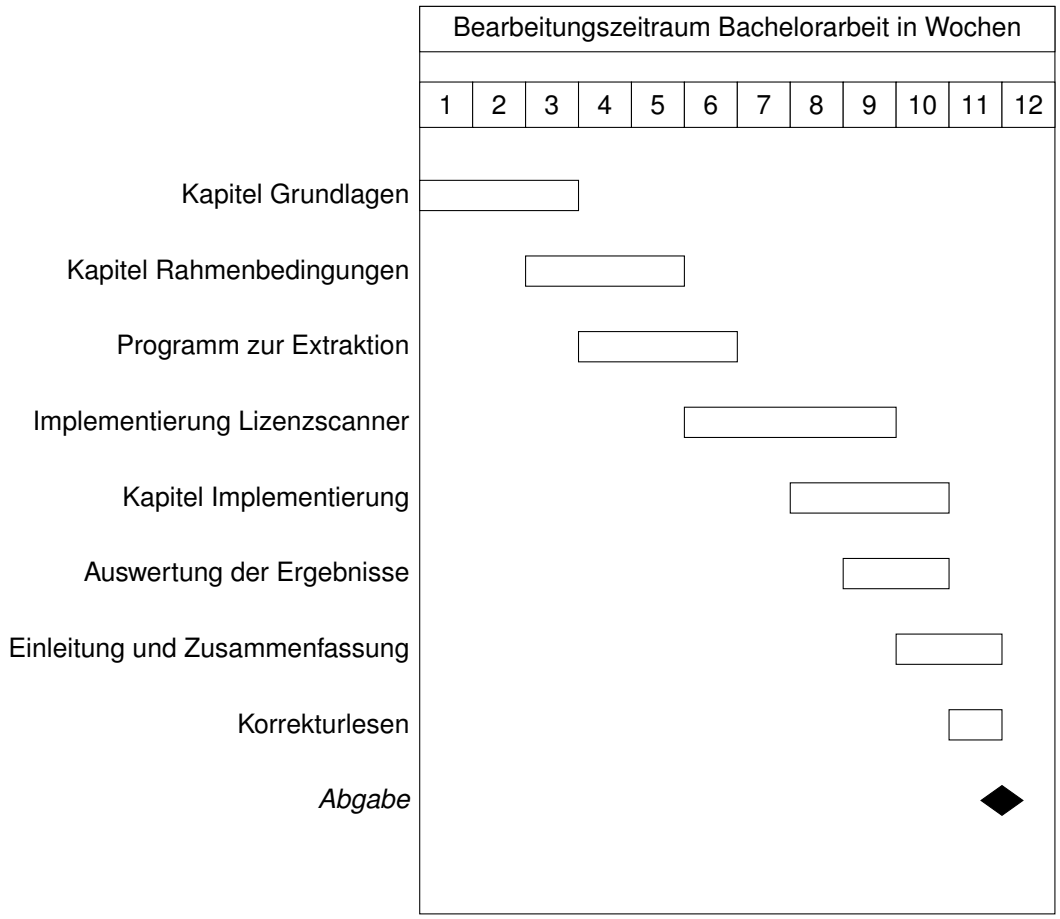

Abbildung 5: Zeitplanung der Bachelorarbeit

In Abbildung 5 ist das Gantt Diagram zu sehen, welche die Bearbeitungszeit einteilt. Da die Phase der Implementierung mit einem Entwicklungsaufwand verbunden ist, wird dafür eine größere Zeitspanne eingeplant.
Zur Überprüfung dieser Zeitplanung wird die aktive Bearbeitungszeit mit einem Program erfasst und die Zeitplanung gegebenenfalls adjustiert.

3.4 Entwicklungs- und Testumgebung

3.4.1 Entwicklungsumgebung

Als Entwicklungsumgebung wird Intellj IDEA Ultimate genutzt, da der Autor damit schon vertraut ist und dieses zahlreiche Funktionen bietet, wie zum Beispiel die Möglichkeit der Fehlerlokalisierung mit Haltepunkten, automatische Vorschläge beim Schreiben von Quellcode, Hervorhebung von Quellcode und zahlreiche andere Funktionen welche dem Nutzer Erleichterungen und schnellere Ergebnisse bringen sollen.

3.4.2 Versionskontrolle mit Git

Um die Änderungen von Quellcode zurückverfolgen zu können und sicher zu sein, dass diese gesichert sind gibt es für die Implementierungsphase dieser wissenschaftlichen Arbeit eine Versionskontrolle mit dem Versionsverwaltungssystem Git. Dies gilt sowohl für das Tool zur Extraktion der Daten aus Fossology, als auch für den Lizenzscanner. Diese sind als separate Projekte implementiert. Damit die Änderungen nachvollziehbar sind werden die Kontributionen (Commits) bei Git nach einem festen Schema benannt. Das Schema hält sich an die von Google, in dem Javascript Framework Angular festgelegten Richtlinien. Das ergründet sich daher, dass diese Konvention auch im betreuenden Unternehmen eingesetzt wird, welches eine spätere Integration in Fossology vereinfacht. Zusätzlich zu der lokalen Versionskontrolle wird in Git eine sogenannte Remote definiert, welche die Sicherung des Quellcodes auf einem Git Server erlaubt.

3.4.3 Testumgebung

Als Testumgebung für den Lizenzscanner kommt eine virtuelle Maschine zum Einsatz, welche eine modifizierte Debian Linux Distribution enthält. Der Einsatz dieses Betriebssystems ermöglicht unter anderem eine konsistente, einfach einzurichtende Test- und Entwicklungsumgebung durch den Einsatz der APT Paketverwaltung.
Diese bietet eine vielfältige breite Auswahl an Softwarepaketen an, welche bei der Entwicklung des Lizenzscanners hilfreich sind.

3.4.4 Bisherige Methoden der Lizenzfindung in Fossology

Monk

Monk ist ein Lizenzscanner in Fossology, welcher text-basierte Suchen durchführt und dafür gute Lizenztexte bzw. Muster für die Suche benötigt. Dazu verwendet Monk den Jaccard Index als Textvergleichsmetrik mit einer Gewichtung, welche Ergebnisse nach ihrer Größe sortiert. Diese Eigenschaft verschiedene Ergebnisse nach ihrer Größe zu ordnen, ist bei Monk besonders wichtig, denn dadurch kann zwischen verschiedenen Versionen von Lizenzen unterschieden werden. Bei Benutzung von Monk nutzt der Agent die Lizenztexte, welche schon in dem Fossology Server gespeichert sind. Jedoch gibt es auch die Möglichkeit neue unbekannte Lizenzen hinzuzufügen.[55]

[55] FOSSology Workgroup. (2017). Features — FOSSology. Zugriff 28. September 2017 unter https://www.fossology.org/features, vlg.

Nomos

Nomos findet Lizenzen dadurch, dass dieser kurze Phrasen, reguläre Ausdrücke und Heuristiken in Sätzen feststellt. Dabei muss sich diese Phrase in der Nähe einer anderen passenden Phrase befinden, um eine Lizenz festzustellen. Dies minimiert die Falscherkennung von Lizenzen. Bei der Lizenzerkennung nutzt Nomos verschiedene Phasen der Erkennung. Es beginnt mit dem Finden von Schlüsselwörtern, um lizenzrelevante Phrasen zu erkennen. Anschließend wird aus dem Lizenztext eine Baumstruktur erstellt, welche aus regulären Ausdrücken besteht um gewisse Lizenzen zu finden. Bei unvollständiger Erkennung gibt Nomos die Information zurück, dass eine nicht klassifizierte Lizenz gefunden wurde oder falls zutreffend eine Lizenzfamilie.[56]

Ninka

Ninka ist auch ein Softwarewerkzeug, welches Lizenzen in Quellcode identifizieren kann. Diese Identifikation basiert auf der Erkennung von Sätzen in Texten. Dabei ist es möglich mit Ninka verschiedene Lizenzen und deren Variationen zu unterscheiden.[57]

[56] FOSSology Workgroup. (2017). Features — FOSSology. Zugriff 28. September 2017 unter https://www.fossology.org/features, vlg.

[57] German, D. M. [Daniel M] und Manabe, Y. (2011). Ninka, a license identification tool for Source Code. Zugriff 29. September 2017 unter http://ninka.turingmachine.org/, vgl.

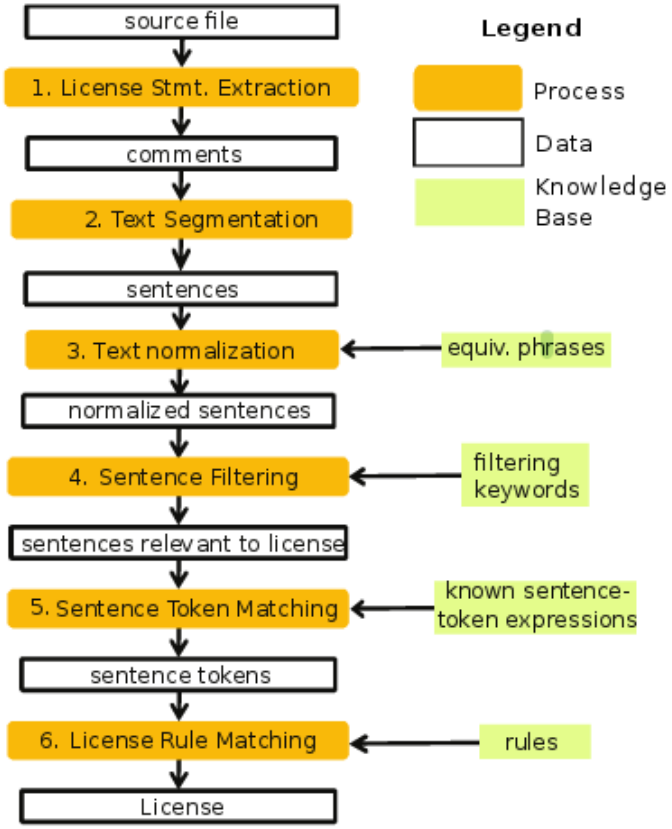

Abbildung 6: Ablauf Ninka Algorithmus

Quelle: A sentence-matching method for automatic license identification of source code files[58]

Ninkas Lizenzerkennungsprozess kann in sechs Schritte eingeteilt werden.[59]

Extraktion

Der erste Schritt beginnt mit der Extraktion von Kommentaren aus Lizenztexten mit speziellen Kommentarerkennungsmethoden. Falls es für eine Programmiersprache keine Methode gibt, Kommentare im Kopf des Quellcodes zu erkennen werden die ersten 1000 Zeilen Quellcode für die Analyse genutzt.

Textsegmentierung

Im zweiten Schritt werden die Kommentare mithilfe eines Text Segmentierungsprozesses in eine Abfolge

[59] German, D. M. [Daniel M] und Manabe, Y. (2011). Ninka, a license identification tool for Source Code. Zugriff 29. September 2017 unter http://ninka.turingmachine.org/; German, D. M. [Daniel M.], Manabe, Y. und Inoue, K. (2010). A sentence-matching method for automatic license identification of source code files. In *Proceedings of the IEEE/ACM international conference on Automated software engineering* (S. 437–446). ASE '10. Antwerp, Belgium: ACM. doi:http://doi.acm.org/10.1145/1858996.1859088, vgl.

von Aussagen zerlegt. Dieser Vorgang dient dazu Quellcode und Kommentarnotationen wie zum Beispiel (# oder //) aus Kommentaren zu entfernen.

Normalisierung von Sätzen

Bei der Normalisierung von Sätzen werden bekannte Sätze durch eine normalisierte Version ersetzt. Das bedeutet, dass zum Beispiel die Silbentrennung von Wörtern aufgehoben wird.

Satzteil Vergleich

In diesem Vorgang wird der Satzteil gesucht, welcher bereits in der Datenbank existiert und mit diesem übereinstimmt. Falls es keine Übereinstimmung gibt, wird „UNKNOWN“ genutzt. Einige reguläre Ausdrücke können Stellen enthalten, in denen sich optionale Bestandteile in der Lizenz befinden. Dies könnten bei der Familie der BSD Lizenzen zum Beispiel Autor oder Copyright Informationen sein.

Lizenzregelmatching

Im sechsten Schritt wird durch Ninka die Lizenz mithilfe der vorher verarbeitenden Daten gefunden oder eine Liste von nicht gefundenen Lizenzen erstellt.[60]

Bisherige Implementierungen

Zum Zeitpunkt der Ausarbeitung dieser wissenschaftlichen Arbeit existieren verschiedene Verfahren, um Lizenzen in Software zu finden. Fossology selbst bietet die vorher genannten Lizenzscanner an, welche mit unterschiedlichen Vorgehensweisen arbeiten und an einer Datei herausfinden, welche Lizenz oder Lizenzfamilie in der Datei enthalten sein könnte. Auch gibt es viele Anwendungen von Maschinenlernen bzw. Text Mining, womit zum Beispiel Text kategorisiert oder klassifiziert werden kann. Jedoch gibt es nach den Recherchen des Verfassers bisher keine Lösung, welche die Suche von Lizenzen mit intelligenten Text Mining Verfahren kombiniert.

3.5 Mögliche Verfahrensweisen der intelligenten Lizenzfindung

3.5.1 Bag of Words

Bag of Words ist ein Verfahren, welches natürliche Sprache verarbeitet. Dabei ist das Ziel des Bag of Word Verfahrens meist Dokumente zu klassifizieren und Ähnlichkeiten oder Themen zu erkennen. Dabei wird gezählt wieviele Wörter es in einem Dokument gibt und wie häufig diese Vorkommen. Danach werden aus den Wörtern, anhand der Häufigkeit in einem Text, Vektoren gebildet. Damit wird anschließend auf eine Klasse geschlossen.[61]

[60] German, D. M. [Daniel M.], Manabe, Y. und Inoue, K. (2010). A sentence-matching method for automatic license identification of source code files. *Proceedings of the IEEE/ACM international conference on Automated software engineering* (S. 437–446). ASE '10. Antwerp, Belgium: ACM. doi:http://doi.acm.org/10.1145/1858996.1859088; German, D. M. [Daniel M], Manabe, Y. und Inoue, K. (2011). A sentence-matching method for automatic license identification of source code files. Zugriff 29. September 2017 unter http://ninka.turingmachine.org/dmg2010ninka.pdf, vgl.

[61] The Open Source Initiative. (2014). Explanation Bag of Words (BoW) – Natural Language Processing. Zugriff 7. Oktober 2017 unter https://ongspxm.wordpress.com/2014/12/05/explanation-bag-of-words-

3.5.2 Naive Bayes

Der Naive Bayes Algorithmus funktioniert damit, dass konditionelle Wahrscheinlichkeiten berechnet werden. Konditionelle Wahrscheinlichkeit ist die Wahrscheinlichkeit, dass etwas passiert nachdem etwas anderes schon geschehen ist. Dadurch ist es möglich die Wahrscheinlichkeit eines Ereignisses basierend auf vorherigem Wissen zu berechnen.

$$P(H|E) = \frac{(P(E|H) * P(H)}{P(E)} \tag{3.1}$$

Die Formel 3.1 beschreibt die a posteriori Wahrscheinlichkeit einer Hypothese. Dabei stellt $P(H)$ die Annahme über den Wahrheitsgehalt der Hypothese dar. $P(E|H)$ bestimmt die Wahrscheinlichkeit der Beobachtung, falls die Hypothese H wahr ist. $P(E)$ stellt die priore Wahrscheinlichkeit der Klasse dar.
Die Vorteile des Bayes'sches Klassifikators sind Einfachheit und Geschwindigkeit der Klassifizierung. Zusätzlich ist der Naive Bayes Algorithmus auch für Mehrklassenprobleme, wie Lizenzfindung geeignet. Allerdings setzt der Algorithmus unbedingt die Unabhängigkeit aller Merkmale voraus, weil sich sonst die Leistung stark verschlechtert.[62]

3.5.3 Entscheidungsbaum

Ein Entscheidungsbaum ist ein Model, welches auch Vorhersagen treffen kann. Es kann bei Klassifizierung und Regression eingesetzt werden. Bei dem Fall der Lizenzerkennung wird nur die Klassifizierung genutzt, weshalb eher Klassifizierungsbaum als Begriff genutzt wird, um den Algorithmus zu beschreiben. Diese Klassifizierungsbäume können Objekte, gewisse vordefinierte Klassen zuweisen.[63]

bow-natural-language-processing-2/, vgl.

[62] Saxena, R. (2017). How the Naive Bayes Classifier works in Machine Learning. Zugriff 26. Juli 2017 unter http://dataaspirant.com/2017/02/06/naive-bayes-classifier-machine-learning/, vgl.

[63] Maimon, O. und Rokach, L. (2014). *Data Mining With Decision Trees: Theory And Applications (2nd Edition)*. Series In Machine Perception And Artificial Intelligence. World Scientific Publishing Company.

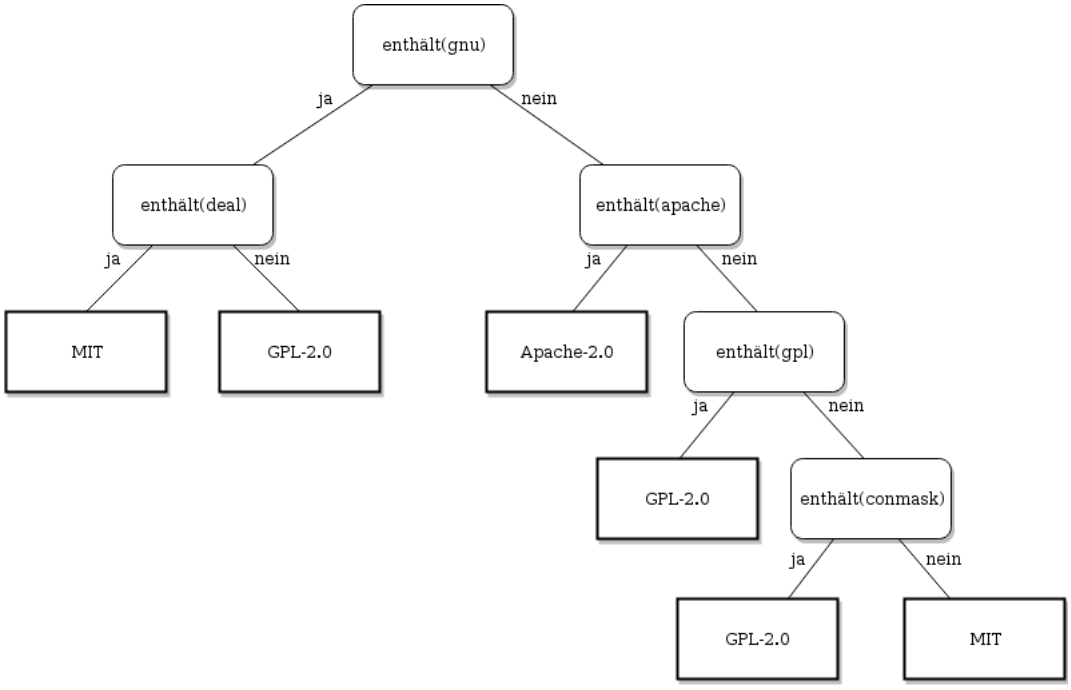

Abbildung 7: Beispiel Entscheidungsbaum eines Trainingdatensatzes mit 350 Dateien
Quelle: Basierend auf dem Entscheidungsbaum Algorithmus selbst erstellt

In Abbildung 7 wird sichtbar, wie der Entscheidungsbaumalgorithmus aus der Existenz einzelner Wörter Entscheidungen für die verwendete Lizenz treffen kann. In diesem Beispielbaum wird zunächst die Existenz des Wortes „gnu“ überprüft. Suche von Wörtern in Texten ist im Bereich der Softwareentwicklung eine gängige Praxis und stellt für die meisten Computer keine Herausforderung dar. Falls das Programm das Wort findet, wird der linke Zweig in Abbildung 7 genutzt. Nun überprüft der Algorithmus den Text auf die Existenz des Wortes „deal“. Wird das Wort „deal“ nun im Text gefunden wird der Text vom Entscheidungsbaumalgorithmus als MIT Lizenz klassifiziert. Damit endet die Klassifizierung des Textes in diesem Beispiel. Das Ende des Zweiges und somit das Ergebnis wird in Abbildung 7 durch eine schwarze Umrandung der Lizenz gekennzeichnet.

3.6 Implementierungsvorbereitung

3.6.1 Aufbau von Testdaten

Die Testdaten für die Implementierung bestehen aus Quellcode von Softwareprojekten. Diese sollten einen Lizenzkopf oder eine Referenz auf die Lizenz im Kommentar der Quellcode Datei enthalten.

Häufige Bestandteile von Kopfkommentaren in Quellcode

Die SPDX Foundation listet 345 Lizenzen auf, welche häufig genutzt werden (Stand Januar 2017). Das Kopfkommentar besteht meist zum größten Teil aus der verwendeten Lizenz für die Datei, jedoch können

auch andere Angaben darin beschrieben sein. Darunter zählt das Copyright des Autors, falls ein Kontributor Quellcode beigetragen hat, gibt es die Möglichkeit, dass sich dieser in das Kopfkommentar des Quellcodes einschreibt. Beispiele für diese Darstellung des Urheberrechts ist in der Apache2, GPL-2.0, MIT Lizenz zu finden.

```
Copyright [yyyy] [Name des Urheberrechtsinhabers]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

Apache2 Lizenz mit Urheberrecht[64]

3.6.2 Kriterien an Testdaten

Um ein gutes Model für den Maschinenlernalgorithmus zu finden, müssen die Testdaten möglichst sorgfältig ausgewählt werden. Zusätzlich müssen die Testdaten einige Kriterien erfüllen, damit Probleme vermieden werden.

Vielfalt der Quellen

Falls die benötigten Quellcodedateien für das Anlernen des Algorithmus nur von einer Software oder nur einem Autor kommen, besteht die Gefahr, dass es zu einer Überanpassung der Trainingsdaten kommt. Eine Überanpassung von Trainingsdaten, auch Overfitting genannt könnte sich dadurch verstärken, indem der Autor etwa das Urheberrecht konsistent gleich in den Quellcode der Software einfügt. Dies ist für ein Softwareprojekt nichts ungewöhnliches, könnte aber bei der Erkennung von Lizenzen den Algorithmus in der Hinsicht beeinflussen, dass dieser eher Kontributoren und Autoren von Quellcode voneinander trennt als die genutzten Lizenzen.
Außerdem könnte ein Kontributor zum Beispiel auch einen modifizierten Lizenzkopf benutzen bei dem das gleiche Problem von Overfitting auftritt, wenn die Testdaten und Trainingsdaten nur von einem Autor oder einem Softwareprojekt stammen. Deshalb ist es wichtig möglichst viele Dateien von verschiedenen Projekten zu nutzen.

[64] The Apache Software Foundation. (2004). Apache License, Version 2.0. Zugriff 10. Oktober 2017 unter https://www.apache.org/licenses/LICENSE-2.0.

Menge der Trainingsdaten

Damit der Algorithmus genau ist, das heißt, die richtigen Resultate liefern kann, müssen genügend Testdaten vorhanden sein, um die zu schlechte Anpassung, auch Underfitting genannt zu vermeiden. Andernfalls kann es sein, dass der Algorithmus viel zu wenig Daten hatte, um sich überhaupt ein aussagekräftiges Modell bilden zu können.

Reinheit der Testdaten

Um zu verhindern, dass der Algorithmus eher auf vorhandenen Quellcode oder andere Merkmale, statt Lizenzen angepasst wird sollten Trainingsdaten schon so weit aufbereitet sein, dass diese repräsentativ für Lizenzen in üblichen Softwareprojekten sind.

4 Implementierung

4.1 Datenextrahierung aus Fossology

4.1.1 Überblick

Da eine große Menge an Testdaten nötig ist, um sinnvolle Ergebnisse zu erzielen muss eine große Datenbank von schon klassifizierten Dateien mit den zugeordneten Lizenzen vorhanden sein. Dies dient dazu den Algorithmus zu trainieren. Der Arbeitgeber des Verfassers benutzt Fossology um Quellcode zu scannen und dieses mit entsprechenden Lizenzen zu klassifizieren. Daher ist es möglich die Datenbank von Lizenzen zu benutzen. Allerdings gibt es noch keine Möglichkeit die Quellcodedateien mit ihren Lizenzen ohne manuelle Arbeit zu extrahieren.

4.1.2 Analyse der Fossology Datenbank

Damit kein Aufwand für die Realisierung einer Oberfläche und andere, für diesen Fall nicht notwendigen Lösungen, erstellt werden muss und damit die Bearbeitungszeit eingehalten werden kann, wird der Quellcode mit zugehörigen Lizenzen aus der Fossology PostgreSQL Datenbank extrahiert.
Um Quellcode mit den zugehörigen Lizenzen automatisch aus der Datenbank zu extrahieren, muss zunächst eine explorative Einsicht in diese genommen, um die Datenstruktur herauszufinden die von Fossology genutzt wird. Dazu wird eine lokale Fossology Instanz in der Testumgebung aufgesetzt. In diese werden zunächst Testdaten in Fossology hochgeladen und manuell mit Lizenzen identifiziert. Das heißt es wird Software Lizenz Clearing durchgeführt, damit sich die Datenbank mit Daten füllt. Danach wird mithilfe eines Programms zur Datenbankeinsicht, die Verbindung zwischen hochgeladenen Dateien und identifizierten Lizenzen erkannt.

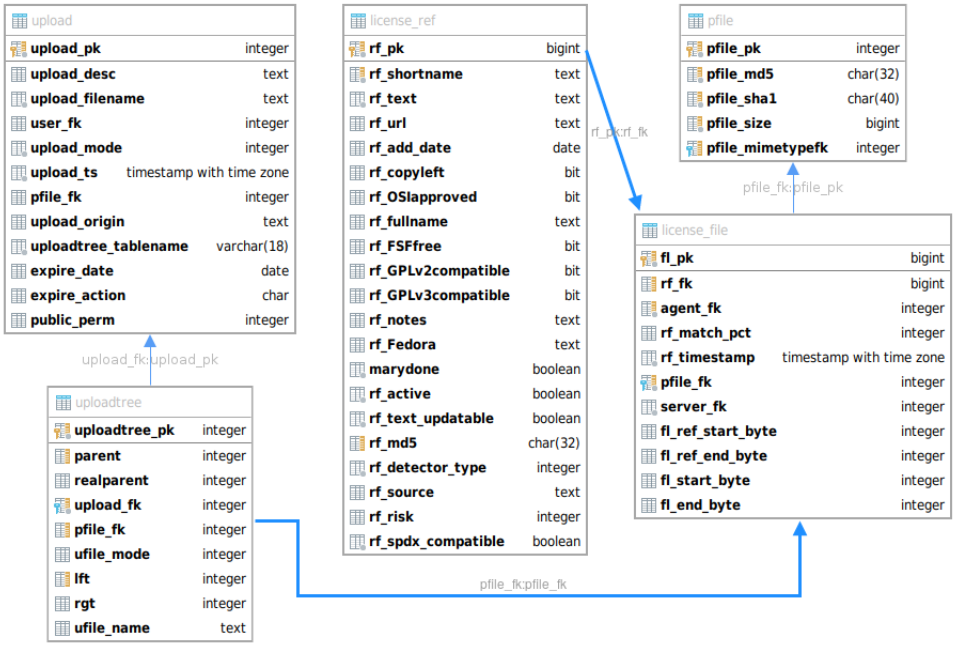

Abbildung 8: Darstellung der Ergebnisse der Fossology Datenbankanalyse(Ausschnitt)
Quelle: Abbildung aus eigenen Aufnahmen

In Abbildung 8 ist ein Auszug eines Entity Relationship Diagramms mit Tabellen aus der Fossology Datenbank sichtbar. Erkennbar ist dabei, dass die Tabellen
„license_ref “und„ license_file“ durch die Beziehung von rf_pk und rf_fk verbunden sind. Rf steht für die Referenz zu der Lizenz, welche als Eindeutige Nummer (ID) abgebildet ist. Verknüpft mit beiden Tabellen ist demzufolge zudem die Tabelle pfile, welche in diesem Fall für die Extraktion und Abgleichung von Datei IDs genutzt wird. Dennoch sind weitere Verknüpfungen nötig, um Lizenzdateien mit den klassifizierten Lizenzen extrahieren zu können. Auf diese wird nachfolgend näher eingegangen.

4.1.3 Explorative Datenanalyse mithilfe von SQL Abfragen

Nachdem die manuelle Datenanalyse abgeschlossen ist, stellt sich heraus dass die Daten in verschiedene Tabellen unterteilt sind. Diese beinhalten beispielsweise den Hashwert der Datei oder den jeweiligen primären bzw. Fremdschlüssel, welcher aus einer ID besteht.
Deshalb wird zunächst die ID der hochgeladenen Dateien benötigt. Diese findet sich in der Tabelle „license_file “zusammen mit dem Fremdschlüssel für die zugehörige Lizenz.
Das Ergebnis in Tabelle 2, wird durch diese SQL Abfrage erstellt.

```
SELECT pfile_fk as "Datei Fremdschlüssel",
          rf_fk as "Lizenz Fremdschlüssel"
FROM license_file;
```

Datei Fremdschlüssel	Lizenz Fremdschlüssel
5	317
6	507
7	507
8	317

Tabelle 2: Auszug aus dem Ergebnis der SQL Abfrage in license_file

Damit erhält man nun die Fremdschlüssel, also die ID der Datei und die ID der zugeordneten Lizenz. Um den Fremdschlüssel der Datei und den Fremdschlüssel der zugeordneten Lizenz in deren Namen aufzulösen, benötigt man jedoch noch andere Abfragen. Damit der Vorgang vereinfacht werden kann, werden sogenannte Joins eingesetzt. Diese dienen dazu Tabellen miteinander zu verknüpfen. Da die Informationen über den Dateinamen in der Tabelle „uploadtree“ und die Informationen über den Namen der Lizenz in der Tabelle „license_ref“ zu finden sind, müssen diese kombiniert werden.

```
SELECT ufile_name as "Dateiname", rf_shortname as "Lizenz"
FROM license_file, uploadtree, license_ref
WHERE license_file.pfile_fk=uploadtree.pfile_fk
AND license_file.rf_fk=license_ref.rf_pk;
```

Diese Abfrage an die PostgreSQL Datenbank (siehe oben) stellt jeder Datei in der Fossology Datenbank mit ihrer zugehörigen Lizenz dar.

Dateiname	Lizenz
fo-restore-s3	GPL-2.0
fo-backup-s3	GPL-2.0
Makefile	No_license_found
unique.php	GPL-2.0
autoopts.c	BSD

Tabelle 3: Auswahl der Ergebnisse aus einer SQL Abfrage, welche Dateinamen und Lizenzen darstellt

Um die Gültigkeit der Ergebnisse der Abfrage zu überprüfen, werden in der Fossology Weboberfläche Stichproben durchgeführt. Das bedeutet, dass einzelne Abfrageergebnisse mit Anzeigen in Fossology verglichen werden. Aber auch bei dieser Abfrage sind weitere Optimierungen möglich, welche zum Beispiel durch den Ausschluss von Dateien ermöglicht werden kann, welche zu keiner Lizenz assoziiert sind.
Nachdem anschließend die richtige Abfrage zur Extraktion der Dateinamen mit der zugeordneten Lizenz gefunden ist, muss der Inhalt der Datei extrahiert werden. Um herauszufinden, in welcher Verzeichnisstruktur diese Dateien gespeichert werden, wird der Quellcode von Fossology und die Verzeichnisstruktur untersucht. Anhand eines Kommentars, dessen Quellcode und der Untersuchung des Verzeichnisses wird die Struktur offensichtlich. Daraufhin wird eine weitere Analyse in der Datenbank durchgeführt, welches die Tabelle „pFile“ offenlegt.

Datei ID	MD5 Hash	SHA1 Hash	Dateigröße in Bytes
3	1625E925290422E 7F2C57753BA802CC2	6E8D6BADEF58FB94EB19 E58E5C216EE9810FF97D	15592299
4	6D3D7B30D1FD36B 94C9A9939AB964972	83266C61032F7C0267FF 5DD4D9F87393F2D09CFA	5542394
5	56C560F8065967B C83FB3937F4EC0FFB	FFBCEC2D98A6F2018B22 4F266FCE0C3D50CD5F3A	1302
6	F7E14E5B4570421 64708015F8355BC9F	8B655BCF1902CE3AD20E FE9DAFB69040A4FBAF9B	2166
7	641EE801D1C00C9 CCA-DAE989AE88D1E1	299847771B61939172D90 5F34A57209F45E76D47	316

Tabelle 4: Auswahl der Ergebnisse aus der SQL Abfrage, welche Datei ID, MD5 Hash, SHA1 Hash und Dateigröße einer Datei darstellt

Diese Tabelle „pFile“ (siehe Tabelle 4) speichert die eindeutige Kennzeichnung einer Datei, den Hash des Dateinamens im Format MD5 und SHA1 und zusätzlich noch die Dateigröße in Bytes. Der Hauptordner der Fossology Daten befindet sich in

/srv/fossology/repository/localhost/files/

Nach weiteren Untersuchungen liegen Dateien in drei Unterverzeichnissen, welche durch die Teilung der SHA1 Prüfsumme in Zeichenketten der Länge zwei benannt sind. Um die Daten zu extrahieren wird der Abfrage in der Datenbank ein Filter hinzugefügt, welche die angezeigten Lizenzen einschränken soll. Dies erlaubt eine Extraktion von mehreren Lizenzen eines Typs. Zusätzlich werden die Ergebnisse in der Abfrage zufällig sortiert, um möglichst zufällige Daten für den Lizenzscanner zu erhalten.

4.1.4 Automatisierung der Extraktion

Um manuelle Extraktionsarbeit, Interaktionen mit der Weboberfläche und Zeit zu sparen, wird anhand der herausgefundenen Eigenschaften über die Fossology Datenbank ein Programm erstellt, welches Dateien aus der Fossology Datenbank mit den zugehörigen Lizenzen extrahieren kann. Dies stellt den ersten Schritt in Richtung eines automatischen Lizenzscanners dar, denn dieser benötigt genaue Trainingsdaten. Bei der Fossology Datenbank, welche intern im betreuenden Unternehmen genutzt wird, ist dies der Fall und eignet sich deshalb zum Trainieren des Maschinenlernalgorithmusses. Damit die Dateien, welche von verschiedensten quelloffenen Projekten stammen, noch zu der gefundenen Lizenz zugeordnet sind, werden von dem Fossology Extraktionsprogramm Ordner erstellt welche den gleichen Namen tragen wie die Lizenz mit denen diese in der Fossology Datenbank eingetragen sind. Dadurch ist es möglich diese Ordnerstruktur, bzw. den Namen des übergeordneten Ordners als Klassifizierung zu nutzen.
Dieses Extraktionsprogramm wird in der Programmiersprache Python implementiert, da der Verfasser bereits ein hohes Maß an Erfahrung mit dieser Programmiersprache vorweisen kann, was zu einer schnellen Implementierung und Minimiierung der Wahrscheinlichkeit für auftretende Fehler führt.

4.2 Implementierung des maschinenlernenden Lizenzscanners

Nachdem die Daten nun mit einem Programm zur Extraktion aus der Fossology Datenbank entnommen sind, muss der intelligente Lizenzscanner implementiert werden. Auch dieser ist in Python geschrieben. Damit der Vorgang der Implementierung strukturiert erfolgt und die Daten verwertbar sind, wird nach Vorbild des Crisp-DM Model verfahren. Dieses ist ein Data-Mining Verfahren, welches den Prozess in verschiedene Phasen einteilt.[65]

4.2.1 Geschäftsverständnis

In der Phase des Geschäftsverständnis werden Ziele und Anforderungen des Algorithmusses festgelegt und die konkrete Aufgabenstellung formuliert. In diesem Fall geht es darum Quellcodedateien nach ihren Lizenzen klassifizieren zu können.[66]

4.2.2 Datenverständnis

Datenverständnis bedeutet, dass Daten untersucht werden und mögliche Probleme in der Datenqualität erkannt werden. Dieses wurde bei der Untersuchung der Fossology Datenbank durchgeführt. Dabei wurden keine großen Probleme in der Qualität der Daten festgestellt, weil diese Daten frühere Produktionsdaten des betreuenden Unternehmens sind.[67]

4.2.3 Vorbereitung der Daten

Auch wenn die Daten im Schritt vorher aus der Fossology Datenbank extrahiert wurden, enthalten diese viele Informationen, welche die Leistung des Algorithmus verschlechtern bzw. eine Erkennung von Lizenzen unmöglich machen könnten. Da der Quellcode für das Trainieren des Algorithmus irrelevant ist, wurden einige Parameter in den Trainingsvorgangs des Lizenzscanners eingebaut, welche diese filtern. Diese sind als optionale Parameter definiert und können ein- und ausgeschaltet werden.[68] Im folgenden werden diese kurz beschrieben.

Stoppwörter

Ein möglicher Parameter, der aktiviert oder deaktiviert werden kann, ist die Entfernung von Stoppwörtern. Stoppwörtern sind Wörter, welche sehr häufig auftreten und daher keine Relevanz für den Lizenzscanner haben sollten. Diese werden häufig bei überwachtem Maschinenlernen verwendet, welches hier auch genutzt wird.[69] Beispiele für Stoppwörter sind

[65] Riepl, W. (2012). CRISP-DM: Ein Standard-Prozess-Modell für Data Mining — Statistik Dresden. Zugriff 27. Juli 2017 unter http://statistik-dresden.de/archives/1128, vgl.
[66] Riepl, 2012, vgl.
[67] Riepl, 2012, vgl.
[68] Riepl, 2012, vgl.
[69] Ganesan, K. (2014). All About Stop Words for Text Mining and Information Retrieval. Zugriff 23. Oktober 2017 unter http://text-analytics101.rxnlp.com/2014/10/all-about-stop-words-for-text-mining.html, vgl.

```
aber, abermals, acht, ähnlich etc.

siehe https://github.com/solariz/german_stopwords
/blob/master/german_stopwords_full.txt
```

Da Lizenzen aber meist in englischer Sprache verfasst sind, wird bei dem Filter eine Datenbank von englischen Stoppwörtern genutzt, welche von den Lizenztexten extrahiert werden. Dadurch werden Wörter gefiltert, welche nicht als Entscheidungsmerkmal für einen bestimmten Typ von Lizenz gelten können.

Wortstammbildung

Um die Entscheidungsmerkmale für den Algorithmus weiter zu reduzieren und nicht relevante Worteigenschaften auszuschließen, welche den Algorithmus in seiner Erkennung verschlechtern könnten, wird von allen Wörtern in den Lizenzen der Wortstamm gebildet. Dadurch wird verhindert, dass Variationen von Lizenzen den Entscheidungsprozess des Algorithmus beeinflussen.

Trennung in Phrasen

Statt dem Algorithmus ganze Lizenztexte aus Quellcodedateien zu übergeben, besteht auch die Möglichkeit im Trainierungsteil des Lizenzscanners dem Algorithmus nur Sätze zu übergeben welche trotzdem mit einer Lizenz klassifiziert sind.

Alphanumerischer Filter

Durch die Nutzung von regulären Ausdrücken ist es möglich Zeichen aus dem Trainingstext zu entfernen, welche keine Buchstaben und keine Zahlen sind. Dies entfernt nicht nur Sonderzeichen, sondern auch Zeichen welche häufig in bestimmten Programmiersprachen vorkommen. Dies sind Kommentarzeichen wie

```
""", // oder #
```

Dies soll eine Spezialisierung des Algorithmus auf die Erkennung von Programmiersprachen statt auf die Lizenz im Quellcode verhindern.

Extraktion von Kommentaren

Durch die Python Bibliothek „Pygments“ ist es möglich Quellcode hervorzuheben. Aber es ist auch möglich gewisse Teile einer Quellcodedatei zu extrahieren. Bei der Aktivierung dieser Option im Lizenzscanner wird die Pygment Bibliothek genutzt, welche alle Kommentare aus Quellcode zahlreicher Sprachen extrahiert. Übersprungen werden beim Trainieren Dateien, bei denen die Programmiersprache nicht erkannt werden kann. Wenn diese Option deaktiviert wird, liest der Lizenzscanner nur die ersten 40 Zeilen einer Quellcodedatei, da die längsten Lizenzen nach explorativer Analyse bis zu 40 Zeilen lang sein können.

Variationen der Anzahl an gescannten Lizenzen

Da das Datenset, welches eingesetzt wird nicht alle Lizenzen in großen Anzahlen gleich viel enthält und auch nur eine begrenzte Bearbeitungszeit zur Verfügung steht, beschränkt sich die Erkennung von Lizenzen

auf die Variationen der Erkennung der beliebtesten Softwarelizenzen. Zuerst werden nur Apache-2.0 und GPL-2.0 Lizenzen genutzt. In anderen Durchläufen werden anschließend auch zusätzlich Quellcodedateien mit MIT Lizenz hinzugezogen.

Variationen der Anzahl von trainierten Dateien

Damit erkennbar ist, wie viele Dateien für ein zuverlässiges Ergebnis des Lizenzscanners nötig sind, wird bei dem Testdurchlauf die Anzahl der gelernten Dateien variiert. Die Variationen welche getestet werden sind 500 und 2500 Dateien. Mehr Dateien sind aus Gründen des Speichermangels nicht möglich. Allerdings werden jeweils nur 60 Prozent dieser als eigentliche Trainingsdaten für das Model verwendet. Die übriggebliebenen 40 Prozent werden zur Verifikation des Modelles durch die Testdaten verwendet. Daraus ergeben sich Variationen als Parameter von 300 und 1500 welche als Trainingsdaten verwendet werden können.

4.2.4 Modellierung

In der Modellierungsphase werden Maschinenlernalgorithmen angewendet und ein Modell gebildet. Dieses wird nach dem Training auch auf der Festplatte abgelegt, um ein erneutes Training und die damit verbundene Ressourcenauslastung zu vermeiden. Dies beschleunigt auch den Vorgang beim Überprüfen von Testdaten beim nächsten Durchlaufen des Lizenzscanner, denn dieser muss somit kein Modell mehr berechnen.

Wahl der Bibliothek

Aufgrund der Einfachheit und Kompatibilität zu der NLTK Library wurde Textblob gewählt. Textblob ist eine Python Bibliothek, welche den Einstieg in natürliche Sprachanalyse geben soll. Sie verbindet die Funktionalitäten von NLTK und pattern. Interessant für die Verarbeitung von natürliche Sprache mit Maschinenlernverfahren ist die Integration mit NLTK. Diese Bibliothek ist speziell darauf ausgerichtet natürliche Sprache zu verarbeiten. Da Lizenztexte auch zu natürlicher Sprache zählen, ist dies ideal für den Anwendungsfall der Lizenzerkennung in Quellcode.[70]

Auswahl der Algorithmen

Damit möglichst genaue Ergebnisse erzielt werden und diese verglichen werden können, werden zwei Maschinenlernverfahren ausgewählt. Die Textblob Bibliothek stellt einige Verfahren zur Verfügung. Für diese wissenschaftliche Arbeit werden zwei davon
in ihrer Leistung mithilfe von realen schon klassifizierten Quellcodedateien verglichen. Die Algorithmen die hierbei verglichen werden sind Naive Bayes und das Entscheidungsbaumverfahren.

4.3 Auswertung der Ergebnisse

Die Auswertung der Ergebnisse kann als Evaluierung, also fünfte Phase des CRISP-DM Verfahrens verstanden werden.

[70] Loria, S. (2017). TextBlob: Simplified Text Processing. Zugriff 23. Oktober 2017 unter https://textblob.readthedocs.io/en/dev/, vgl.

4.3.1 Vorgehensweise

Um die Qualität eines Algorithmus und damit auch die Qualität der Daten zu verstehen und diese verbessern zu können, müssen zusätzlich zu den Trainingsdaten, welche dem Algorithmus antrainiert werden auch Testdaten ausgewählt werden, welche genauso wie die Trainingsdaten bereits klassifiziert sind. Jedoch sind diese nicht in dem Trainingsset des Algorithmus enthalten. Diese dienen lediglich zur Überprüfung der Genauigkeit.
Beim Einlesen der groben, ungefilterten Quellcodedateien werden diese eingeteilt in 60 Prozent Trainingsdaten, welche der Algorithmus zum Erlernen benötigt und 40 Prozent Testdaten. Dabei lernt der Algorithmus nicht die Testdaten und liest diese auch nicht ein. Nachdem dieser die Trainingsdaten gelernt hat, wird die Genauigkeit mithilfe der Testdaten berechnet. Das heißt, überprüft wird, ob die erkannte Klassifizierung auch der zugeordnete Klassifizierung der 40 Prozent Testdaten entspricht. Dazu müssen die Testdaten lediglich von dem Algorithmus klassifiziert werden. Die richtig erkannten Daten (E) werden dann prozentual in Relation mit der totalen Datenmenge (T) gestellt. Daraus wird dann auf die Genauigkeit (G) geschlossen.

$$G = \frac{E}{T} * 100 \tag{4.1}$$

Damit die Daten ausgewertet werden können, werden diese bei der Laufzeit des Programms in eine Comma-separated values (Komma getrennte Werte) (CSV) Datei geschrieben, welche die unterschiedlichen Testläufe mit verschiedenen Parametern und unterschiedlichen Ergebnissen dokumentiert.

4.3.2 Rückschlüsse

Rückschlüsse werden bei dem CRISP-DM Verfahren Bereitstellung genannt, weil in dieser Phase die Ergebnisse des Maschinenlernens aufbereitet und ausgewertet werden sollen.[71]
Dadurch dass die Ergebnisse des Trainingsvorgangs in eine CSV Datei geschrieben werden, sind diese maschinenlesbar, welches die Auswertung vereinfacht. Da die Algorithmen Naive Bayes und Entscheidungsbaum mit unterschiedlich gefilterten Trainings- und Testdaten durchlaufen wurden, wird dieser Einfluss nachfolgend ausgewertet.

Durchführung mit 250 Quellcodedateien

Von diesen 250 Quellcodedateien wurden 150 für das Trainieren des Algorithmus verwendet und 100 für die Überprüfung der Genauigkeit, das heißt als Testdaten. Dabei lies sich feststellen, dass bei keiner Datenfilterung und der Verwendung von drei unterschiedlichen Lizenzen und zehn Durchläufen eine Genauigkeit von 96 Prozent an richtig erkannten Lizenzen entsteht.
Bei der Nutzung von nur zwei Typen an Lizenzen im Quellcode ergab sich eine Genauigkeit von 98 Prozent. Diese Ergebnisse bereits ohne jegliche Datenfilterung sehr aussagekräftig. Ähnliche Ergebnisse konnten durch das Entscheidungsbaumverfahren erzielt werden. Bei der Nutzung von Datenfilterungsverfahren, wie zum Beispiel der Extraktion von Kommentaren statt der Nutzung der ersten 40 Zeilen, wurde das Ergebnis nicht in großem Maße beeinflusst. Jedoch sind diese Ergebnisse stark vom Zufall abhängig, denn bei zwei Lizenzen gibt es nur eine Wahrscheinlichkeit von 50 Prozent, dass die Klassifikation des Algorithmus falsch

[71] Riepl, W. (2012). CRISP-DM: Ein Standard-Prozess-Modell für Data Mining — Statistik Dresden. Zugriff 27. Juli 2017 unter http://statistik-dresden.de/archives/1128, vgl.

ist. Zusätzlich wurden nicht viele Trainings- und Testdaten eingesetzt. Dies macht das Ergebnis mit 250 Quellcodedateien trotz guter prozentualer Genauigkeiten nicht repräsentativ.

Durchführung mit 2500 Quellcodedateien

Auch hier wird die Teilung von 60 zu 40 genutzt, welche die Daten in Trainings- und Testdaten zu je 60 Prozent und 40 Prozent einteilt. Bei der Erstellung eines Models mit zwei Lizenzdatensätzen kann eine Genauigkeit von durchschnittlich 98 Prozent mit zehn Durchläufen des Lizenzscanners über 1000 Dateien gemessen werden. Diese wird durch die weitere Filterung von Dateien wie zum Beispiel Filterung von Kommentaren nicht viel weiter verbessert.
Bei der Nutzung von drei Lizenzen ist eine durchschnittliche Genauigkeit von 97 Prozent für die Erkennung der richtigen Lizenzklasse erkennbar. Die meisten fehlklassifizierten Dateien wurden als GPL-2.0 klassifiziert, obwohl diese MIT sind. Bei der Extraktion von Kommentaren und Nutzung des Filters, welcher ungültige Zeichen löscht sind ähnliche Ergebnisse festzustellen. Bemerkenswert ist auch die durchaus nachvollziehbare Extraktion der wichtigsten Entscheidungsmerkmale durch den Naive Bayes Algorithmus mit 2500 Testdaten und Kommentarextraktion.

```
contains(shall) = True          MIT : Apache
contains(warranty) = True       MIT : Apache
contains(portions) = True       MIT : Apache
contains(limitation) = True     MIT : Apache
contains(limitations) = True    Apache : MIT
contains(applicable) = True     Apache : MIT
contains(obtain) = True         Apache : MIT
contains(writing) = True        Apache : MIT
contains(ownership) = True      Apache : GPL-2.0
contains(permission) = True     MIT : Apache
```

Die Darstellung des Entscheidungsbaum Algorithmus stellt einige Probleme dar.

```
if contains(apache) == False:
if contains(substantial) == False:
  if contains(mit) == False:
    if contains(ffi.h) == False: return 'GPL-2.0'
    if contains(ffi.h) == True: return 'MIT'
  if contains(mit) == True: return 'MIT'
if contains(substantial) == True: return 'MIT'
if contains(apache) == True: return 'Apache-2.0'
```

Zwar sind durchaus Wörter erkennbar, welche in Verbindung mit Lizenzen stehen, jedoch kommt es trotzdem zum Overfitting, das heißt zu einer Überanpassung der Testdaten. Denn ffi.h ist eine Datei aus der Fossology Datenbank, welche den Namen ffi.h im Text enthält. Jedoch kann ein Dateiname generell nicht als Aussage zu der verwendeten Lizenz genutzt werden. Das heißt der Algorithmus hat sich zu sehr an die Daten angepasst.

4.3.3 Fehlerverständnis

Um die von dem Algorithmus falschklassifizierten Quellcodedateien herauszufinden und um die Trainingsdaten weiter verbessern zu können, wird eine Konfusionsmatrix genutzt, welche die falsch klassifizerten Lizenzen aufzeigen soll.

		Eigentliche Klasse		
		MIT	Apache-2.0	GPL-2.0
Vorhergesagte Klasse	MIT	306	0	0
	Apache-2.0	0	322	0
	GPL-2.0	18	10	344

Tabelle 5: Beispiel Konfusionsmatrix des Naive Bayes Algorithmus mit 2500 Dateien

In Tabelle 5 wird deutlich wie eine Konfusionsmatrix die Fehler des Algorithmuses bzw. des Modells verdeutlichen kann. In diesem Beispiel wurden 2500 Dateien genutzt. 1500 davon als Trainingsdaten und die verbleibenden 1000 Dateien als Testdaten für das Modell. Daraus ergab sich eine Genauigkeit von ca. 97 Prozent.

Fehlerdiagnose anhand des Algorithmuses

Naive Bayes

Die Textblob Bibliothek bietet die Möglichkeit an die informativsten Merkmale anzuzeigen, um zu einer Lizenz per Quellcode zu schließen. Bei 2500 Dateien können diese Informationen extrahiert werden.

```
        contains(apache) = False           GPL-2.0      : Apache-2.0
         contains(person) = True           MIT          : Apache-2.0
    contains(restriction) = True           MIT          : Apache-2.0
           contains(whom) = True           MIT          : Apache-2.0
contains(merchantability) = True           MIT          : Apache-2.0
        contains(fitness) = True           MIT          : Apache-2.0
            contains(gnu) = True           GPL-2.0      : Apache-2.0
    contains(permissions) = True           Apache-2.0   : MIT
       contains(language) = True           Apache-2.0   : MIT
            contains(2.0) = True           Apache-2.0   : GPL-2.0
```

Diese Form der Auflistung ermöglicht einen Einblick in die Funktionsweise des Modells, welches aus dem Naive Bayes Algorithmus entsteht. Beispielsweise gibt es eine höhere Wahrscheinlichkeit, dass es sich um eine GPL-2.0 Lizenz handelt, wenn das Wort „apache“ nicht enthalten ist. Hier wird sichtbar, dass der Algorithmus sehr stark an die Existenz eines Lizenztextes gebunden ist. Außerdem werden auch Fehler in der Klassifizierung sichtbar, denn ein Enthalten von zum Beispiel „2.0“ bestimmt noch nicht die Lizenz, denn dieser Wortteil ist sowohl in der Apache-2.0 Lizenz, als auch in der GPL-2.0 Lizenz enthalten.

Entscheidungsbaum

Bei dem Entscheidungsbaumverfahren ist es möglich die Fehler in dem Entscheidungsbaum selbst zu

erkennen und dadurch eventuelle Maßnahmen einzuleiten, denn aus den Ergebnissen lässt sich dieser Pseudocode mithilfe der textblob Bibliothek generieren.

```
if contains(sell) == False:
  if contains(apache) == False:
    if contains(mit) == False:
      if contains(www.apache.org/licenses/license-2.0)
          == False: return 'GPL-2.0'
      if contains(www.apache.org/licenses/license-2.0)
          == True: return 'Apache-2.0'
    if contains(mit) == True: return 'MIT'
  if contains(apache) == True: return 'Apache-2.0'
if contains(sell) == True: return 'MIT'
```

Dieser Pseudocode ist auch grafisch als Entscheidungsbaum darstellbar, weil es sich hierbei um eine einfache Abfolge von Anweisungen handelt. Erkennbar ist an diesem Beispiel mit 2500 Dateien Quellcode, dass der Pseudocode auch für Menschen sehr gut nachvollziehbar ist und damit den Algorithmus bildlich darstellt.

Fehler durch informellen Lizenzkopf

Bei der Betrachtung der falsch klassifizierten Dateien fällt auf, dass diese häufig entweder keinen Lizenzkopf haben oder eher informelle Lizenzinformationen enthalten.

```
// licensed to the .net foundation under one or more agreements.
// the .net foundation licenses this file to you under the mit license.
// see the license file in the project root for more information.
```

Zwar ist hier die Lizenz für eine Person erkennbar, weil die MIT Lizenz hier mit Namen genannt wird, jedoch enthält diese Datei nicht den kompletten formellen Lizenzkopf. Dieser sieht meist so aus wie nachfolgend und enthält sehr viele Informationen in Form von Wörtern und Formulierungen welche direkt zu der MIT Lizenz zugeordnet sind.

```
/*****************************************************************************
* The MIT License
* Copyright (c) 2003 Novell Inc.  www.novell.com
*
* Permission is hereby granted, free of charge, to any person obtaining  a copy
* of this software and associated documentation files (the Software), to deal
* in the Software without restriction, including  without limitation the rights
* to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
* copies of the Software, and to  permit persons to whom the Software is
* furnished to do so, subject to the following conditions:
*
* The above copyright notice and this permission notice shall be included in
```

```
* all copies or substantial portions of the Software.
*
* THE SOFTWARE IS PROVIDED AS IS, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
* AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
* LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
* SOFTWARE.
```

Die MIT Lizenz mit Nodell Copyright
Quelle: Fossology Lizenzdatenbank

Fehler durch Duale Lizenzen

```
//  ================================================================================
//  copyright (c) 1995-2015 mort bay consulting pty. ltd.
//  --------------------------------------------------------------------------------
//  all rights reserved. this program and the accompanying materials
//  are made available under the terms of the eclipse public license v1.0
//  and apache license v2.0 which accompanies this distribution.
//
//      the eclipse public license is available at
//      http://www.eclipse.org/legal/epl-v10.html
//
//      the apache license v2.0 is available at
//      http://www.opensource.org/licenses/apache2.0.php
//
//  you may elect to redistribute this
//  code under either of these licenses.
```

Beispiel für ein dualen Lizenzkopf
Quelle: Fossology Lizenzdatenbank

Auch in dem Beispiel, ist es für den Algorithmus durch zwei Faktoren schwer die Lizenz zu klassifizieren. Zum Einen ist die Lizenz nicht mit ihrem vollen Lizenzkopf erwähnt und zum Anderen handelt es sich bei dieser Datei, um eine dual lizenzierte Datei. Das bedeutet, dass die Datei nach Bedarf unter einer der beiden Lizenzen eingesetzt werden kann.
Behebbar wäre dies, indem der Algorithmus speziell zusätzlich auf Duale Lizenzen klassifiziert wird, ohne die Erkennung von Dateien mit einfachen Lizenzen einzuschränken. Da die Fossology Datenbank, jedoch noch nicht genug Dateien mit Dualen Lizenzen besitzt, wird diese Verbesserung im Rahmen dieser Bachelorarbeit nicht weiter verfolgt.

Fehler bei der Extrahierung von Kommentaren

Eine Möglichkeit der Filterung bei dem Lizenzscanners des Autors ist die Option der Kommentarextrahierung. Diese findet Lizenzen, welche sich nicht nur an oberster Stelle des Quellcodes befinden. Die Annahme ist, dass die Genauigkeit dadurch verbessert werden kann. Allerdings verschlechtert diese sich im Vergleich mit der Methode, welche keine Filterung vornimmt und nur die ersten 40 Zeilen der Quellcodedatei betrachtet. Dies entsteht dadurch, dass auch Kommentare aus dem Quellcode extrahiert werden, welche keine Lizenz darstellen.
Vermeidbar ist dies nur schwer, jedoch kann das Modell so optimiert werden, dass diese zusätzlichen unötigen Informationen nicht ins Gewicht fallen.

```
/* let's just use gettickcount64 if we can. */
/* greg hazel assures me that this works, that bittorrent has
* done it for years, and this it won't turn around and
* bite us.  he says they found it on some game programmers'
* forum some time around 2007.
*/
```

Ein Kommentar aus Quellcode
Quelle: Fossology Datenbank

Dieses Beispiel von Opensource Quellcode aus der Fossology Datenbank zeigt dieses Problem, denn dabei handelt es sich nicht um eine Beschreibung der Lizenz.

Fehler durch fehlende Lizenzen in Quellcodedateien

Es kommt teilweise vor, dass der Quellcode direkt keine Lizenz enthält, sondern dies durch eine projektbasierte Lizenz gelöst wird. Da der Lizenzscanner für das Untersuchen von Dateien erstellt wurde, kann das jedoch nicht untersucht werden. Probematisch ist dies, weil der Lizenzscanner zum Trainieren des Modells existierende Dateien aus der Fossology Datenbank nutzt, welche durch diese Weise identifiert sein könnten. Der Filter für Kommentarextrahierung vermeidet dieses Training von falschen Daten teilweise, denn dabei werden dem Lernalgorithmus nur die Kommentare im Quellcode übergeben.
Der Lizenzscanner könnte bei Nichterkennung einer Lizenz zu der Lizenz zurückfallen, welche am häufigsten in dem Open Source Projekt vorkommt. Dies ist jedoch als unabhängiger Lizenzscanner schwer und kann erst beachtet werden, nachdem der Lizenzscanner in Fossology integriert wurde.

Absturz durch hohe Speicherauslastung

Dadurch, dass der Computer bei dem Lernen von Daten einen großen Aufwand betreibt, indem Wahrscheinlichkeiten etc. berechnet werden, kommt es durchaus vor, dass bei der Berechnung des Modells von den Trainingsdaten der Arbeitsspeicher und der Auslagerungsspeicher zu sehr und zu schnell gefüllt werden. Dies führt zu einer Belastung des Systems oder sogar zu dem Absturz des Programms. Dies kann auch bei der Filterung von Text beobachtet werden. Gelöst wurde dies durch eine Erweiterung des Arbeitsspeichers in der virtuellen Maschine und die Einschränkung von Trainingsdaten auf maximal 2500 Dateien.
Aber auch nicht vermeidbare Fehler kommen vor. Denn der Lizenzscanner lernt zufällige Quellcodedateien, welche bereits klassifiziert wurden. Eine Erhöhung der Lerndatenmenge würde jedoch bessere Ergebnisse erzielen. Möglich ist dies durch die Nutzung einer leistungsfähigeren, besser ausgerüsteten Rechenmaschine.

5 Zusammenfassung

5.1 Fazit

Resümierend lässt sich feststellen, dass die Einordnung von Quellcode und Projekten in verschiedene Lizenzen sehr wichtig ist, weil quelloffene Software bei der Entwicklung neuer Produkte oder Software für Unternehmen immer prominenter wird. Dies ist eine Schlussfolgerung der Unternehmen, welche feststellen, dass der Einsatz bedeutende Vorteile mit sich bringt, zum Beispiel die Reduzierung von Entwicklungskosten durch die Nutzung von existierender Software oder die Flexibilität welche Unternehmen durch den Einsatz zahlreicher Open Source Projekte bekommen. Damit die Verbreitung und der Einsatz von Open Source Software zunimmt, muss auch das Bewusstsein für Open Source Lizenz Clearing wachsen, damit rechtliche Konflikte vermieden und die Rechte von Autoren eingehalten werden können. Durch Open Source Lizenz Clearing ist es möglich, dass Firmen selbstsicherer quelloffene Software verwenden können. Dies steigert die Produktivität, Effizienz und Transparenz der Software. Zusätzlich könnte es auch Firmen motivieren eigene Software unter einer quelloffenen Lizenz zu veröffentlichen und diese in der Gemeinschaft weiterzuentwickeln und zu pflegen. Damit profitiert nicht nur das Unternehmen, sondern auch jeder Einzelne, welcher etwa eine Softwarelösung für Probleme benötigt. Fossology als Software Clearing Toolkit enthält bereits verschiedene Lizenzscanner, welche bei der Einordnung von Dateien zu Lizenzen helfen können, jedoch beeinflussen die Ergebnisse vorheriger Lizenzentscheidungen nicht die Klassifizierung von neuen Lizenzen.
Mit dieser wissenschaftlichen Arbeit konnte gezeigt werden, dass es möglich ist einen unabhängigen Lizenzscanner zu programmieren, welcher mithilfe von maschinellen Lernverfahren wie zum Beispiel dem Naive Bayes oder dem Entscheidungsbaumverfahren Lizenzentscheidungen trifft. Diesem Lizenzscanner kann nicht komplett vertraut werden, denn im Rahmen dieser Bachelorarbeit konnte kein Resultat mit 100 prozentiger Genauigkeit bestimmt werden. Dies liegt in den meisten Fällen jedoch nicht an dem Lizenzscanner selbst, sondern an den Dateien welche beispielsweise keine Referenz auf eine Lizenz enthalten. In solchen Fällen muss es die Möglichkeit geben auf manuelle Lizenzklassifizierung zu bestehen oder auf die Einbeziehung von projektbasierten Lizenzen zurückzugreifen, um ein Ergebnis zu erhalten.

5.2 Ausblick

In Zukunft könnte dieser intelligente Lizenzscanner in Fossology integriert werden. Zunächst würde dieser Lizenzscanner nur von bestehenden Daten lernen, welche Menschen bereits eintragen haben. Das würde keinen Mehraufwand auslösen, weil der Lizenzscanner im Hintergrund Daten lernt. Bei genügend gelernten Lizenzdateien könnte dieser Lizenzscanner anschließend die bisherigen Mustererkennungsscanner ergänzen und ihre Ergebnisse bestätigten. Dies ermöglicht eine weitere Stufe von Selbstsicherheit für die Erkennung von Lizenzen. Bei genügend Selbstsicherheit des Unternehmens und nach der Abklärung von rechtlichen Problemen, könnte der intelligente Lizenzscanner mit der Nutzung von Maschinenlernverfahren die endgültigen Entscheidungen für Lizenzen in Quellcode treffen und so dem Menschen Arbeit abneh-

men. Bei Problemen könnte ein kleineres Team von Lizenzexperten eine Entscheidung treffen, welche anschließend dem Lizenzscanner zum Lernen zur Verfügung steht.

Literatur

Adobe Systems Incorporated. (2017a). Adobe PostScript. Zugriff 7. September 2017 unter http://www.adobe.com/products/postscript.html

Adobe Systems Incorporated. (2017b). Was ist Adobe PDF? PDF, Adobe PDF — Adobe Acrobat DC. Zugriff 7. September 2017 unter https://acrobat.adobe.com/ch/de/why-adobe/about-adobe-pdf.html

Black Duck KnowledgeBase. (2017a). Top Open Source Licenses. Zugriff 15. September 2017 unter https://www.blackducksoftware.com/sites/default/files/images/Top%5C%20Open%5C%20Source%5C%20Licenses.png

Black Duck KnowledgeBase. (2017b). Top Open Source Licenses. Zugriff 15. September 2017 unter https://www.blackducksoftware.com/top-open-source-licenses

Black Duck Software, Inc. (2017). Open Source License Compliance. Zugriff 24. Juni 2017 unter https://www.blackducksoftware.com/solutions/open-source-license-compliance

Bleich, H. (2004). Deutsches Gericht bestätigt Wirksamkeit der GPL. Zugriff 7. September 2017 unter https://www.heise.de/newsticker/meldung/Deutsches-Gericht-bestaetigt-Wirksamkeit-der-GPL-101616.html

Borchers, D. (2017). Gesichtserkennung per Video: Bundespolizei sucht Berliner Freiwillige. Zugriff 26. September 2017 unter https://www.heise.de/newsticker/meldung/Gesichtserkennung-per-Video-Bundespolizei-sucht-Berliner-Freiwillige-3746657.html

Bridge, R. (2013). Open source software - The advantages & disadvantages. Zugriff 12. September 2017 unter http://entrepreneurhandbook.co.uk/open-source-software/

Brown, K. (2016). What Is GitHub, and What Is It Used For? Zugriff 10. September 2017 unter https://www.howtogeek.com/180167/htg-explains-what-is-github-and-what-do-geeks-use-it-for/

Brownlee, J. (2016). Supervised and Unsupervised Machine Learning Algorithms. Zugriff 26. September 2017 unter https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/

Collins, K. (2017). A federal court has ruled that an open-source license is an enforceable contract. Zugriff 17. Juni 2017 unter https://qz.com/981029/a-federal-court-has-ruled-that-an-open-source-license-is-an-enforceable-contract/

Cormier, P. (2017). What makes us Red Hat. Zugriff 12. September 2017 unter https://www.redhat.com/de/blog/what-makes-us-red-hat

FOSSA, Inc. (2012). GNU General Public License v2.0 (GPL-2.0) Explained in Plain English - TLDRLegal. Zugriff 13. August 2017 unter https://tldrlegal.com/license/gnu-general-public-license-v2

FOSSology Workgroup. (2017). Features — FOSSology. Zugriff 28. September 2017 unter https://www.fossology.org/features

Free Software Foundation. (2017a). Frequently Asked Questions about the GNU Licenses - GNU Project - Free Software Foundation. Zugriff 5. November 2017 unter https://www.gnu.org/licenses/license-list.html#GPLCompatibleLicenses

Free Software Foundation. (2017b). Various Licenses and Comments about Them - GNU Project - Free Software Foundation. Zugriff 5. November 2017 unter https://www.gnu.org/licenses/gpl-faq.en.html#WhatIsCompatible

Free Software Foundation, Inc. (2017). What is Copyleft? - GNU Project - Free Software Foundation. Zugriff 29. September 2017 unter https://www.gnu.org/licenses/copyleft.en.html

Ganesan, K. (2014). All About Stop Words for Text Mining and Information Retrieval. Zugriff 23. Oktober 2017 unter http://text-analytics101.rxnlp.com/2014/10/all-about-stop-words-for-text-mining.html

GBdirect. (2016). Benefits of Using Open Source Software. Zugriff 11. September 2017 unter http://open-source.gbdirect.co.uk/migration/benefit.html

German, D. M. [Daniel M] & Manabe, Y. (2011). Ninka, a license identification tool for Source Code. Zugriff 29. September 2017 unter http://ninka.turingmachine.org/

German, D. M. [Daniel M.], Manabe, Y. & Inoue, K. (2010). A sentence-matching method for automatic license identification of source code files. In *Proceedings of the IEEE/ACM international conference on Automated software engineering* (S. 437–446). ASE '10. Antwerp, Belgium: ACM. doi:http://doi.acm.org/10.1145/1858996.1859088

German, D. M. [Daniel M], Manabe, Y. & Inoue, K. (2011). A sentence-matching method for automatic license identification of source code files. Zugriff 29. September 2017 unter http://ninka.turingmachine.org/dmg2010ninka.pdf

Guido, S. & Müller, A. (2016). *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly.

Hartley, M. (2015). Why Use Open Source Software? - Datamation. Zugriff 11. September 2017 unter http://www.datamation.com/open-source/why-use-open-source-software-1.html

IEEE. (2017). Arthur L. Samuel • IEEE Computer Society. Zugriff 15. September 2017 unter https://www.computer.org/web/awards/pioneer-arthur-samuel

Jaeger, M. C. (2016). FOSSology SPDX in HD. Zugriff 15. Juni 2017 unter http://events.linuxfoundation.org/sites/events/files/slides/%28OCS_Mr.%20Michael%20Jaeger%29OCS-2016-Jaeger%20036%20OSS%20fossy%2020161113%20slides%20spdxHD%203.pdf

Levine, P. (2014). Why There Will Never Be Another RedHat: The Economics Of Open Source — TechCrunch. Zugriff 12. September 2017 unter https://techcrunch.com/2014/02/13/please-dont-tell-me-you-want-to-be-the-next-red-hat/

Loria, S. (2017). TextBlob: Simplified Text Processing. Zugriff 23. Oktober 2017 unter https://textblob.readthedocs.io/en/dev/

Maimon, O. & Rokach, L. (2014). *Data Mining With Decision Trees: Theory And Applications (2nd Edition)*. Series In Machine Perception And Artificial Intelligence. World Scientific Publishing Company.

Mohammed, M., Khan, M. & Bashier, E. (2016). *Machine Learning: Algorithms and Applications*. CRC Press.

Nwana, H. S. (1996). Software agents: an overview. (Kap. 4, S. 5–6). Cambridge University Press.

Oxford University Press. (2017). framework — Definition of framework in English by Oxford Dictionaries. Zugriff 12. November 2017 unter https://en.oxforddictionaries.com/definition/framework

Perens, B. (2017). Understanding the 'GPL is a Contract' court case. Zugriff 7. September 2017 unter https://perens.com/blog/2017/05/28/understanding-the-gpl-is-a-contract-court-case/

Puget, J. F. (2015). Overfitting In Machine Learning (IT Best Kept Secret Is Optimization). Zugriff 24. November 2017 unter https://www.ibm.com/developerworks/community/blogs/jfp/entry/Overfitting_In_Machine_Learning?lang=en

Refsnes Data. (2017). SQL Tutorial. Zugriff 24. November 2017 unter https://www.w3schools.com/sql/

Riepl, W. (2012). CRISP-DM: Ein Standard-Prozess-Modell für Data Mining — Statistik Dresden. Zugriff 27. Juli 2017 unter http://statistik-dresden.de/archives/1128

Saxena, R. (2017). How the Naive Bayes Classifier works in Machine Learning. Zugriff 26. Juli 2017 unter http://dataaspirant.com/2017/02/06/naive-bayes-classifier-machine-learning/

statisticshowto. (2016). Jaccard Index / Similarity Coefficient. Zugriff 28. September 2017 unter http://www.statisticshowto.com/jaccard-index/

The Apache Software Foundation. (2004). Apache License, Version 2.0. Zugriff 10. Oktober 2017 unter https://www.apache.org/licenses/LICENSE-2.0

The Linux Information Project. (2006). Vendor lock-in definition by The Linux Information Project (LINFO). Zugriff 15. September 2017 unter http://www.linfo.org/vendor_lockin.html

The Open Source Initiative. (2014). Explanation Bag of Words (BoW) – Natural Language Processing. Zugriff 7. Oktober 2017 unter https://ongspxm.wordpress.com/2014/12/05/explanation-bag-of-words-bow-natural-language-processing-2/

The Open Source Initiative. (2017a). Frequently Answered Questions — Open Source Initiative. Zugriff 29. September 2017 unter https://opensource.org/faq#permissive

The Open Source Initiative. (2017b). The Open Source Definition (Annotated) — Open Source Initiative. Zugriff 15. Juni 2017 unter https://opensource.org/osd-annotated

The PHP Group. (2017). PHP: What is PHP? - Manual. Zugriff 9. Juli 2017 unter http://php.net/manual/en/intro-whatis.php

The PostgreSQL Global Development Group. (2017). PostgreSQL: About. Zugriff 14. Oktober 2017 unter https://www.postgresql.org/about/

urheberrecht.de. (2017). Urheberrecht und Urheberrechtsgesetz. Zugriff 1. Juli 2017 unter http://www.urheberrecht.de/

Vaughan-Nichols, S. J. (2005). It's an open-source world: 78 percent of companies run open-source software — ZDNet. Zugriff 11. September 2017 unter http://www.zdnet.com/article/its-an-open-source-world-78-percent-of-companies-run-open-source-software/

Verlag Bibliographisches Institut GmbH. (2017a). Duden — Interpreter — Rechtschreibung, Bedeutung, Definition, Herkunft. Zugriff 7. September 2017 unter http://www.duden.de/rechtschreibung/Interpreter

Verlag Bibliographisches Institut GmbH. (2017b). Duden — Interpreter — Rechtschreibung, Bedeutung, Definition, Herkunft. Zugriff 26. September 2017 unter http://www.duden.de/rechtschreibung/Spam

Wikipedia. (2017). Obfuscation (software) - Wikipedia. Zugriff 13. August 2017 unter https://en.wikipedia.org/wiki/Obfuscation_(software)

Selbstständigkeitserklärung

Der Verfasser Max Wittig erklärt, dass er die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt hat. Die aus fremden Quellen (einschließlich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind ausnahmslos als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Max Wittig

Luzern, 25. November 2017