UNIVERSITÄT KOBLENZ · LANDAU

Fachbereich 4: Informatik

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering

Featurebasierte Fehlererkennung mittels Methoden des Machine Learnings

Masterarbeit

zur Erlangung des Grades Master of Science (M.Sc.)
im Studiengang Informatik

vorgelegt von

Stefan Hermann Strüder

Erstgutachter: Prof. Dr. Jan Jürjens
Institut für Softwaretechnik

Zweitgutachter: Dr. Daniel Strüber
Chalmers University of Technology - Göteborg, Schweden (bis 02.2020)
Radboud-Universität - Nijmegen, Niederlande

Koblenz, im März 2020

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐

.....

(Ort, Datum) (Unterschrift)

Kurzfassung

Softwarefehler sind ein großes Ärgernis in der Softwareentwicklung und können nicht nur zu Rufschädigungen sondern auch zu erheblichen finanziellen Schäden für Unternehmen führen. Aus diesem Grund wurden im vergangenen Jahrzehnt zahlreiche Techniken zur Erkennung und Vorhersage von Fehlern entwickelt, welche zum großen Teil auf Methoden des Machine Learnings basieren. Die übliche Herangehensweise dieser Techniken erfolgt auf der Vorhersage von Fehlern auf Dateiebene. Seit einigen Jahren steigt jedoch die Popularität von featurebasierter Softwareentwicklung: ein Paradigma welches auf Funktionsinkremente eines Softwaresystems (Features) setzt und somit für eine breite Variabilität des Softwareproduktes sorgt. Eine gängige Implementationstechnik für Features basiert auf Annotationen mit Präprozessoranweisungen, wie `#IFDEF` und `#IFNDEF`, deren Code sich über mehrere Dateien der Quellcodedateien der Software verteilt („code scattering)“. Ein Fehler in solchem Featureode kann aufgrund dessen weitreichende Folgen für die Funktionalität der gesamten Software haben. Weist ein Teil des Featurecodes Fehler auf, so wird die gesamte Funktion des Features fehlerhaft und führt unter Umständen zum Ausfall der gesamten Funktionalität der Software. An dieses Problem knüpft diese Arbeit an. Es wird eine Vorhersagetechnik für fehlerhafte Features entwickelt, welche auf Methoden des Machine Learnings basiert. Die Auswertung von acht Klassifikatoren, welche jeweils auf einem induviduellen Klassifikationsalgorithmus basieren, zeigt, dass mithilfe des für diese Arbeit erstellten featurebasierten Datensets, eine Genauigkeit von bis zu 92% für die Vorhersage von fehlerhaften oder fehlerfreien Features erreicht werden konnte. Es wird zudem gezeigt, wie der Aspekt der Featureorientierung im Rahmen der Erstellung des Datensets eingebunden wurde und welche Resultate im Vergleich zur herkömmlichen dateibasierten Methodik erzielt werden konnten.

Abstract

Software errors are a major nuisance in software development and can lead not only to damage of reputation but also to considerable financial losses for companies. For this reason, numerous techniques for detecting and predicting errors have been developed over the past decade, which are largely based on machine learning methods. The usual approach of these techniques is to predict errors at file level. For some years now, however, the popularity of feature-based software development has been increasing - a paradigm that relies on function increments of a software system (features) and thus ensures a wide variability of the software product. A common implementation technique for features is based on annotations with preprocessor instructions, such as `#IFDEF` and `#IFNDEF`, whose code is spread over several files of the software's source code files („code scattering“). A bug in such a feature code can have far-reaching consequences for the functionality of the entire software. If a part of the feature code contains errors, the entire function of the feature becomes faulty and may lead to the failure of the entire functionality of the Software. This problem is the subject of this thesis. A prediction technique for faulty features is developed, which is based on methods of machine learning. The evaluation of eight classifiers, each based on an individual classification algorithm, shows that the feature-based data set created for this thesis allows an accuracy of up to 92% for the prediction of faulty or error-free features. It is also shown how the feature orientation aspect was incorporated into the creation of the dataset and what results were achieved compared to the traditional file-based methodology.

Anmerkung

Diese Masterarbeit entstand in Teilen in Zusammenarbeit mit der Forschungsgruppe der Division of Software Engineering unter der Leitung von Thorsten Berger am Department of Computer Science and Engeneering der Chalmers Universität of Technology in Göteborg, Schweden.

Mein besonderer Dank gilt Thorsten Berger für die Ermöglichung und Finanzierung dieser Zusammenarbeit. Ebenfalls gilt mein Dank dem gesamten Team der Forschungsgruppe für die Unterstützung bei Problemen und Fragen zu meiner Arbeit. Ein weiterer Dank gilt Daniel Strüber für seine Initiative zur Ermöglichung der Zusammenarbeit.

Comment

This master thesis was partly written in cooperation with the research group of the Division of Software Engineering headed by Thorsten Berger at the Department of Computer Science and Engineering of Chalmers University of Technology in Gothenburg, Sweden.

My special thanks goes to Thorsten Berger for facilitating and financing this cooperation. I would also like to thank the entire team of the research group for their support in case of problems and questions concerning my work. A further thank you goes to Daniel Strüber for his initiative to make this cooperation possible.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung und Motivation

Ausblick: Dieses Kapitel dient zur allgemeinen Einführung in diese Masterarbeit. Dazu werden neben einer Einleitung und Motivation in das zugrundeliegende Thema, die grundlegenden Strukturen der Arbeit erläutert. Dazu gehören die Forschungsziele und Forschungsfragen, das verwendete Forschungsdesign, eine Übersicht der angedachten und tatsächlichen Zeitplanung sowie eine Erläuterung des Aufbaus der weiterführenden Teile dieser Arbeit.

Softwarefehler stellen einen erheblichen Auslöser für finanzielle Schäden und Rufschädigungen von Unternehmen dar. Solche Fehler reichen von kleineren „Bugs“ bis hin zu schwerwiegenden Sicherheitslücken. Aus diesem Grund herrscht ein großes Interesse daran, einen Entwickler zu warnen, wenn er aktualisierten Softwarecode veröffentlicht, der möglicherweise einen Fehler beinhaltet.

Zu diesem Zweck haben Forscher und Softwareentwickler im vergangenen Jahrzehnt verschiedene Techniken zur Fehlererkennung und Fehlervorhersage entwickelt, die zu einem Großteil auf Methoden und Techniken des *Machine Learnings* basieren. Diese verwenden in der Regel historische Daten von fehlerhaften und fehlerfreien Änderungen an Softwaresystemen in Kombination mit einer sorgfältig zusammengestellten Menge von *Attributen* (in der Regel Features genannt [1]), um einen gegebenen Klassifikator anzulernen beziehungsweise zu trainieren. Dieser dient dann dazu, ein akkurate Vorhersage zu treffen, ob eine neu erfolgte Änderung an einer Software fehlerbehaftet oder frei von Fehlern ist.

Die Auswahl an Lernverfahren für Klassifikatoren ist groß. Studien zeigen, aus diesem Pool von Verfahren sowohl Entscheidungsbaum-basierte (zum Beispiel J48, CART oder Random Forest) als auch bayessche Verfahren die meistgenutzten sind [24]. Alternative Lernmethoden sind Regression, k-Nearest-Neighbor oder künstliche neuronale Netze [7]. Anzumerken ist allerdings, dass es keinen Konsens über die beste verfügbare Lernverfahren gibt, da jedes Verfahren unterschiedliche Stärken und Schwächen für bestimmte Anwendungsfälle aufweist.

Das Ziel dieser Arbeit ist die Entwicklung einer solchen Vorhersagetechnik für Softwarefehler basierend auf Software-Features. Diese beschreiben Inkremente der Funktionalität eines Softwaresystems. Die auf diese Weise entwickelten Softwaresysteme heißen Software-Produktlinien und bestehen aus einer Menge von ähnlichen Softwareprodukten. Sie zeichnen sich dadurch aus, dass sie eine gemeinsame Menge von Features sowie eine gemeinsame Codebasis besitzen

[1] Um einem missverständlichen und doppeldeutigen Gebrauch des Feature-Begriffes vorzubeugen, wird für die hier verwendete Beschreibung der Charakteristika von Daten auch im weiteren Verlauf dieser Ausarbeitung der Begriff „Attribute“ verwendet.

[26]. Durch das Vorhandensein verschiedener Features entlang der Softwareprodukte, kann eine breite Variabilität innerhalb einer Produktlinie erreicht werden.

ÜBERARBEITEN!

Die nachfolgende Abbildung 1.1 zeigt den zentralen Prozess der Entwicklung einer Produktlinie. Aufgeteilt wird dieser in das Domain Engineering und das Application Engineering. Im Rahmen des Domain Engineerings wird ein sogenanntes Variabilitätsmodell (Variability Model) erzeugt, welches durch die Kombination der wählbaren Features beschrieben wird [3]. Gängige Implementationstechniken für Features reichen von einfachen Lösungen durch Annotationen basierend auf Laufzeitparametern oder Präprozessor-Anweisungen bis hin zu verfeinerten Lösungen basierend auf erweiterten Programmiermethoden, wie zum Beispiel Aspektorientierung. In Teilen dieser Implementierungstechniken wird jedes Feature wird als wiederverwendbares Domain Artifact modelliert und gekapselt, welches im Prozess des Application Engineerings in Form einer Konfiguration zusammen mit weiteren Features, im Hinblick auf die gewünschte Funktionalität der Software, ausgewählt werden kann. Ein Software Generator erzeugt dann die gewünschten Software Produkte basierend auf den bereits zuvor genannten Implementationstechniken für Features.

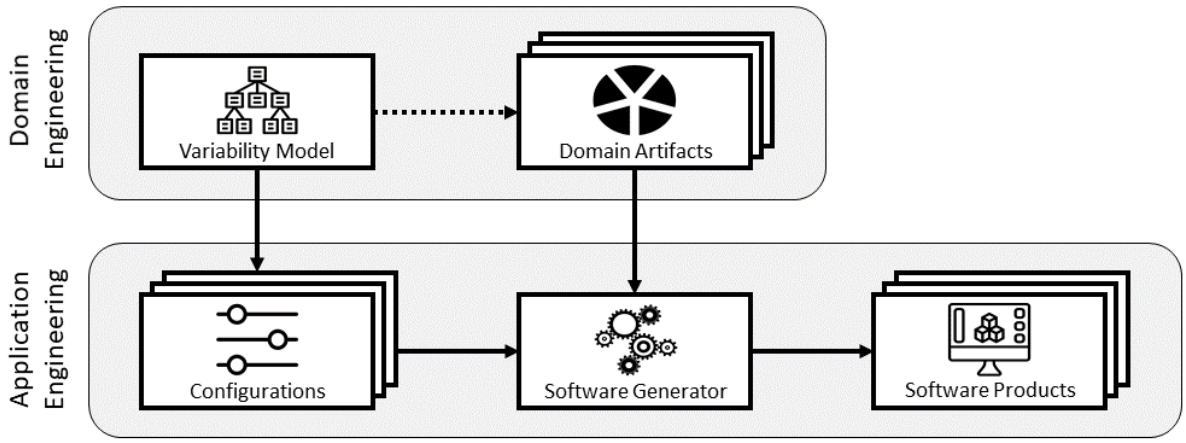


Abbildung 1.1: Generierung von Software-Produktlinien nach [26]

Das Ziel dieser Arbeit ist die Entwicklung einer auf Machine Learning gestützten Vorhersagetechnik für defekte Software-Features. Dieser bisher in wissenschaftlichen Papern nur einmal betrachtete Ansatz ist aufgrund mehrerer Gründe chancenreich:

1. Wenn ein bestimmtes Feature in der Vergangenheit mehr oder weniger fehleranfällig war, so ist eine Änderung, die das Feature aktualisiert, wahrscheinlich ebenfalls mehr oder weniger fehleranfällig.

2. Features, die mehr oder weniger fehleranfällig scheinen, könnten besondere Eigenschaften haben, die im Rahmen der Fehlervorhersage verwendet werden können.

3. Code, der viel Feature-spezifischen Code enthält (insbesondere die sogenannten Feature-Interaktionen), ist möglicherweise fehleranfälliger als sonstiger Code.

Das zuvor genannte Ziel der Arbeit setzt sich aus mehreren Teilzielen zusammen. Dazu zählen die Erstellung eines Datensets zum Trainieren von Machine-Learning-Klassifikatoren sowie

das Anlernen einer repräsentativen Auswahl an Klassifikatoren mit anschließender vergleichender Evaluation dieser. Ein genauer Überblick über die Forschungsziele befindet sich im nächsten Unterkapitel.

Sollte sich im Rahmen der Evaluation einer dieser Klassifikatoren als besonders effektiv erweisen, so würde diese Arbeit den Stand der Technik hinsichtlich der Fehlererkennung in Features vorantreiben und Organisationen erlauben, bessere Einblicke in die Fehleranfälligkeit von Änderungen in ihrer Codebasis zu erhalten.

1.1 Forschungsziele und Forschungsfragen

ÜBERARBEITEN!

Wie bereits in der Einleitung beschrieben, ist das übergeordnete Ziel dieser Arbeit die Entwicklung einer Vorhersagetechnik für Fehler in featurebasierter Software unter Zuhilfenahme von Methoden des Machine Learnings. Dazu ist vorgesehen, das Augenmerk auf Commits von Versionierungssystemen, wie beispielsweise Subversion oder Git, zu richten. Ein Commit bezeichnet dabei die zur Verfügungstellung einer aktualisierten Version einer Software. Als Datenbasis für das Trainieren der Klassifikatoren dienen dann fehlerhafte und fehlerfreie Commits von featurebasierter Software. Dies ermöglicht es, ausstehende defekte Commits vorherzusagen und das Risiko der Konsequenzen von Softwarefehlern zu senken.

Der Prozess der Entwicklung der Vorhersagetechnik ist in drei zu erreichende Forschungsziele eingeteilt. Jedem Forschungsziel werden Forschungsfragen zugeordnet, deren Aufklärung einen zusätzlichen Teil zur Erfüllung der Ziele beiträgt. Im Folgenden werden die Forschungsziele (RO – „research objective“) mit ihren zugehörigen Forschungsfragen (RQ – „research question“) vorgestellt.

RO1: Erstellung eines Datensets zum Trainieren von relevanten Machine-Learning-Klassifikatoren

RQ1a: Welche Daten kommen für die Erstellung des Datensets in Frage?
RQ1b: Wie weit müssen die Daten vorverarbeitet werden, um sie für das Training nutzbar zu machen?

RO2: Identifikation und Training einer Auswahl von relevanten Machine Learning Klassifikatoren basierend auf dem Datenset

RQ2: Welche Machine Learning Klassifikatoren kommen für die gegebene Aufgabe in Frage?

RO3: Evaluierung und Gegenüberstellung der Klassifikatoren sowie Vergleich zu modernen Vorhersagetechniken, die keine Features nutzen

RQ3a: Welche miteinander vergleichbaren Merkmale besitzen die Klassifikatoren?
RQ3b: Welche Metriken können für den Vergleich verwendet werden?
RQ3c: Welche Vor- und Nachteile besitzt ein Klassifikator?
RQ3d: Wie lassen sich die Klassifikatoren mit weiteren Vorhersagetechniken, die keine Features nutzen, vergleichen?

Zusätzlich zu den drei genannten Forschungszielen umfasst die Bearbeitung der Masterarbeit eine Vor- und Nachbereitung, sodass sich insgesamt fünf Arbeitsphasen ergeben. Diese werden in den weiteren Unterkapiteln näher erläutert. Als finale Vorhersagetechnik wird jener Klassifikator verwendet, der sich im Rahmen der Gegenüberstellung im Verlauf der Evaluation als am effektivsten erweist.

1.2 Forschungsdesign

Die für diese Arbeit gewählte Methodik basiert auf dem Prozessmodell Cross-Industry Standard Process for Data Mining, kurz CRISP-DM, nach [8]. Es wird als Vorlage für die Arbeitsphasen zur Erreichung der Forschungsziele dieser Arbeit verwendet. Da der überwiegende praktische Teil dieser Arbeit auf Programmierung im Bereich des Machine Learning konzentriert, bildet das CRISP-DM Prozessmodell ein passendes vordefiniertes Vorgehen. Die nachfolgende Abbildung 1.2 zeigt eine grafische Aufarbeitung des Prozessmodells mit seinen sechs zugehörigen Phasen sowie den Verbindungen zwischen diesen.

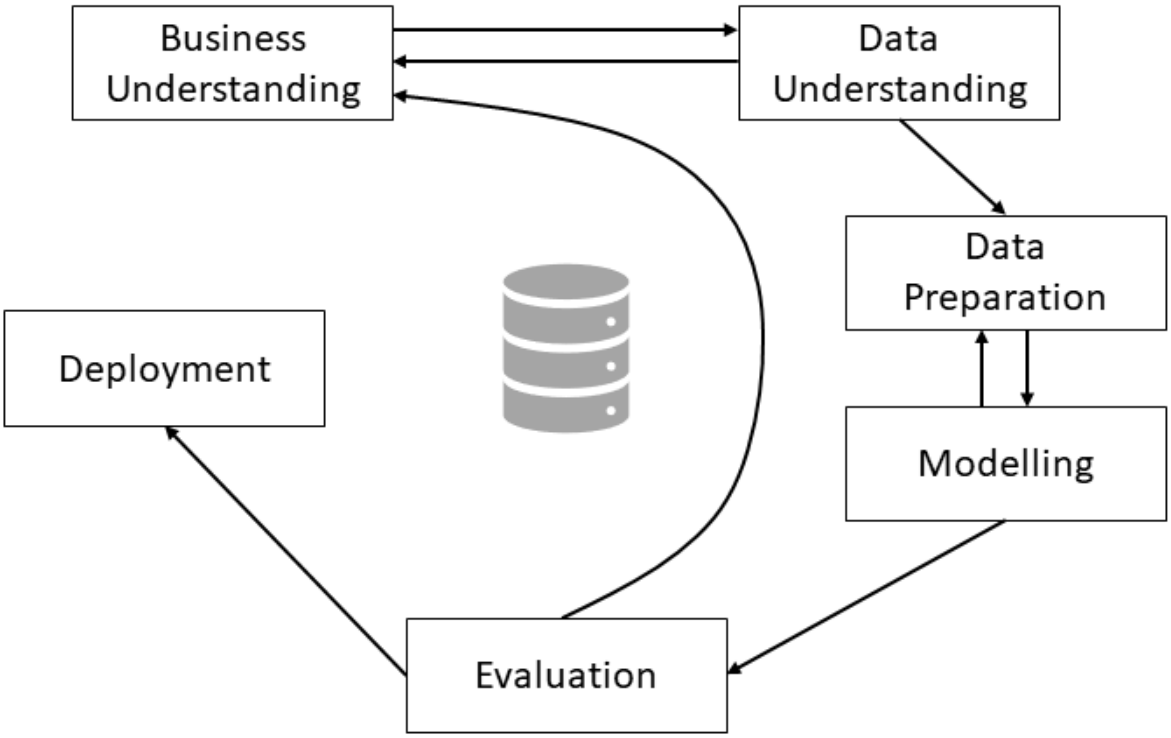


Abbildung 1.2: CRISP-DM Prozessmodells nach [8]

Das CRISP-DM Prozessmodell würde ursprünglich, wie der Name bereits andeutet, für die Erarbeitung von Data Mining Projekten entwickelt, eignet sich jedoch auch zur Verwendung im Rahmen eines Machine Learning Projektes, da sich die in beiden Bereichen verwendeten Methoden und Prozesse zu einem erheblichen Teil überlagern. Ein Überblick über die sechs Phasen des Prozessmodells ist in Abbildung 1.3 dargestellt. Zusätzlich umfasst diese Abbildung die Zuordnung der Arbeitsphasen, die im vorherigen Unterkapitel definiert wurden. Eine Erläuterung der Phasen des Prozessmodells erfolgt im Anschluss der Abbildung. Einen genauen Überblick über den konkreten Umfang der Arbeitsphasen, aufgeteilt in jeweilige Unterziele und zu erfüllende Aufgaben, bietet das im Anschluss folgende Unterkapitel.

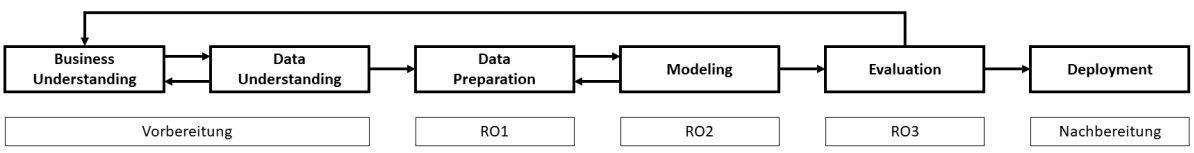


Abbildung 1.3: Phasen des CRISP-DM Prozessmodells nach [8] mit Zuordnung der Arbeitsphasen

Die ersten beiden Phasen *Business Understanding* und *Data Understanding* widmen sich der Vorbereitung der Arbeit. Die initiale Phase umfasst dabei die allgemeine Einarbeitung in das zugrundeliegende Thema und der Formulierung der Forschungsziele. Anzumerken ist, dass diese Phase bereits vor der sechsmonatigen Bearbeitungszeit der Arbeit beginnt und somit schon das Verfassen dieses Proposals als Teilaufgabe dieser Phase gezählt werden kann, da es auch eine grobe Einarbeitung in das Thema erfordert.

Die darauffolgende Phase *Data Understanding* dient der Suche und Einsicht von für den weiteren Verlauf der Phasen relevanten Daten und, falls vorhanden, vorgefertigten Datensets. Da Commits als Datenbasis zur Erlernung der Klassifikatoren betrachtet werden, wird der überwiegende Teil der Suche nach Daten auf dem Onlinedienst GitHub stattfinden, welchem das Versionierungssystemen Git zugrunde liegt. Für die weiteren Phasen ist es von besonderer Bedeutung, den Aufbau der Daten sorgfältig zu untersuchen.

Die dritte Phase *Data Preparation* kümmert sich um die Erstellung eines endgültigen Datensets und den dort hinführenden Prozessen. Diese Phase ist deckungsgleich mit den Anforderungen des ersten Forschungsziels.

Zur Anwendung kommt das im vorherigen Schritt erstellte Datenset in der Phase *Modeling*. In dieser werden die Data-Mining-Algorithmen und -Techniken gemäß den zugrundeliegenden Anforderungen auf das Datenset angewendet. Adaptiert an das Lernen der Machine Learning Klassifikatoren spiegelt dies die Arbeitsphase zur Erfüllung des zweiten Forschungsziels dar.

Die fünfte Phase umfasst die *Evaluation* der Resultate des zuvor erfolgten Schrittes und deckt somit die Erfüllung des dritten Forschungsziels ab.

Die Nachbereitung der Arbeit wird durch die Phase *Deployment* abgedeckt. Diese umfasst die Erstellung der finalen Ausarbeitung sowie der Abschlusspräsentation und der anschließenden Vorführung dieser im Rahmen des Kolloquiums.

Es ist zu erkennen, dass die Beschreibung der Arbeitsphasen weitestgehend auf einem theoretischen Level verfasst wurde. Es wird anhand der fünf CRISP-DM-Phasen gezeigt, was für den erfolgreichen Abschluss der Arbeit absolviert werden muss. Die Erörterung der Frage, wie die einzelnen zu erledigenden Aufgaben durchgeführt werden müssen, ist Teil der Vorbereitung der Arbeit. Im Rahmen der Phasen Business Understanding und Data Understanding wird nach einer eingehenden Recherche die genaue Arbeitsplanung festgelegt (siehe Unterziele der ersten Phase im folgenden Unterkapitel).

1.3 Zeitplanung

ANPASSEN AN TATSÄCHLICHEN ABLAUF

Im Nachfolgenden wird die vorläufige Ablaufplanung der Arbeit aufgezeigt. Die Ordnung erfolgt gemäß der Aufteilung in die fünf zuvor beschriebenen Arbeitsphasen. Die geschätzte Dauer der verschiedenen Unterziele wird jeweils in Tagen, Wochen oder Monaten angegeben. Zur Verfügung stehen insgesamt sechs Monate Bearbeitungszeit.

Phase 1: Vorbereitung

Unterziele	geplante Dauer	tatsächliche Dauer	
Strukturierte Literaturrecherche - Techniken der featurebasierten Softwareprogrammierung - Techniken zur Fehlererkennung in Software - Klassifikation mittels Machine Learning - Klassifikationsmethoden - Auswahl der Programmiersprache - Tool- und Libraryauswahl - Evaluationsmetriken	2 Wochen	TBD	Business Understanding
Recherche zur Bildung eines Datensets - Merkmale / Aufbau eines Datensets - Suche nach Datenquellen - Suche nach vorgefertigten Datensets - Prüfung der Daten / Datensets auf Eignung - Analyse des Aufbaus der Daten / der Datensets	1 Woche	TBD	Data Understanding
Total:	3 Wochen	TBD	

Phase 2: Forschungsziel 1 – Erstellung des Datensets (Data Preparation)

Unterziele	geplante Dauer	tatsächliche Dauer
finale Datenauswahl - Festlegung von Kriterien	1 Woche	TBD
Datenbereinigung - "Preprocessing"	1 Woche	TBD
finale Konstruktion des Datensets - Integration der Daten und des Feature-Aspekts - erneute abschließende Bereinigung sowie Formatierung - Teilung in Training-Set und Test-Set	1 Woche	TBD
Total:	3 Wochen	TBD

Phase 3: Forschungsziel 2 – Training der Machine Learning Klassifikatoren (Modeling)

Unterziele	geplante Dauer	tatsächliche Dauer
Auswahl geeigneter Klassifikatoren	1 Woche	TBD
Training der Klassifikatoren	3 Wochen	TBD
Total:	4 Wochen	TBD

Phase 4: Forschungsziel 3 – Evaluation und Vergleich der Machine Learning Klassifikatoren

Unterziele	geplante Dauer	tatsächliche Dauer
Evaluation der einzelnen Klassifikatoren - Festlegung der Bewertungsmetriken - Anwendung des Test-Sets - Berechnung der Bewertungsmetriken	2 Wochen	TBD
Vergleich der Klassifikatoren anhand der Metriken	2 Wochen	TBD
Vergleich mit weiteren Vorhersagetechniken, die nicht auf Features setzen	1 Woche	TBD
Total:	5 Wochen	TBD

Phase 5: Nachbereitung (Deployment

Unterziele	geplante Dauer	tatsächliche Dauer
Besprechung der vorangegangenen Arbeit mit Betreuer - Umsetzung möglicher Verbesserungsvorschläge	1 Woche	TBD
Erstellung der Ausarbeitung	7 Wochen	TBD
Erstellung der Abschlusspräsentation	1 Woche	TBD
Total:	9 Wochen	TBD

* Die Erstellung der Ausarbeitung ist ein laufender Prozess über den gesamten Verlauf der Bearbeitungszeit. Der hier erwähnte siebenwöchige Zeitraum dient unter Anderem zur Korrektur beziehungsweise Verbesserung hinsichtlich des Feedbacks des Betreuers und zur abschließenden Finalisierung.

Die nachfolgende Abbildung zeigt den zeitlichen Ablauf der Arbeit als Gantt-Chart inklusive konkreter Datumsangaben. Eine größere Version des Plans befindet sich im Anhang.

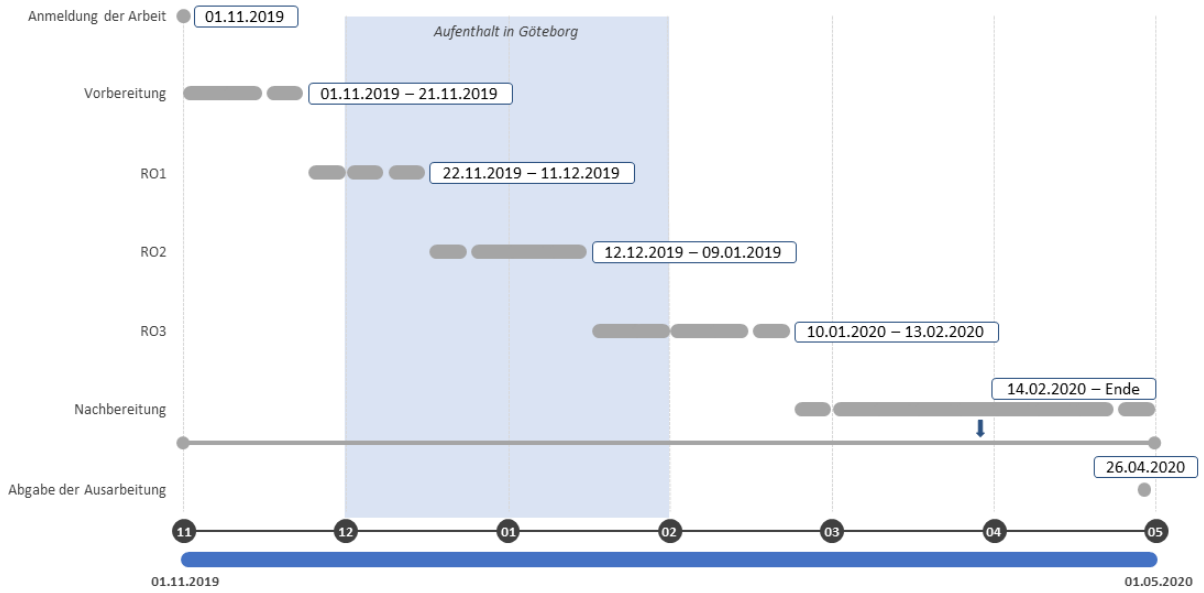


Abbildung 1.4: Zeitlicher Ablaufplan der Arbeit als Gantt-Chart

1.4 Aufbau der Arbeit

Diese Ausarbeitung ist in sechs Kapitel unterteilt. Das erste Kapitel, welches mit diesem Abschnitt abgeschlossen wird, diente zur Einführung in das Thema der Masterarbeit. Ebenso stellte es die theoretischen Rahmenbedingungen der Arbeit vor. Das zweite Kapitel „Hintergrund“ dient zur Vermittlung von Basiswissen zu den grundlegenden Themenkomplexen dieser Ausarbeitung. Dazu wird zunächst die featurebasierte Softwareentwicklung vorgestellt, ehe dann die Machine-Learning-Klassifikation sowie die darauf aufbauende Fehlervorhersage erläutert werden. Die zwei darauffolgenden Kapitel widmen sich der Auseinandersetzung des praktischen Teils dieser Masterarbeit in Form der Erstellung des featurebasierten Datensets sowie des Trainings der Machine-Learning-Klassifikatoren. Die Gegenüberstellung und Evaluation dieser Klassifikatoren erfolgt im fünften Kapitel inklusive eines Vergleiches zu nicht-featurebasierten Methoden zur Fehlererkennung. Eine abschließende Zusammenfassung sowie ein Ausblick auf weiterführende Projekte, die auf diese Masterarbeit aufbauen können, erfolgen im abschließenden sechsten Kapitel.

Zusätzlich wird die Ausarbeitung von zahlreichen Abbildungen zur verständlicheren Verdeutlichung von Zusammenhängen ergänzt.

Kapitel 2

Hintergrund

Ausblick: Zum besseren Verständnis der weiteren Verlaufs dieser Arbeit, dient dieses Kapitel zur Einführung in die zugrundeliegenden Themen. Dazu wird zunächst die featurebasierte Softwareentwicklung erläutert, ehe dann der Themenbereich des Machine Learnings vorgestellt wird. Dazu werden die Klassifikation und die Fehlervorhersage mittels Machine Learning erläutert. Unterstützt werden die Abschnitte von Grafiken zum besseren Verständnis der Zusammenhänge.

2.1 Featurebasierte Softwareentwicklung

2.2 Machine-Learning-Klassifikation

ÜBERARBEITEN!

Die Machine-Learning-Klassifikation unterliegt dem Teilgebiet des *überwachten Machine Learnings* (englisch: supervised Machine Learning). Die nachfolgende Abbildung 2.1 präsentiert den allgemeinen Prozess des überwachten Machine Learnings auf vereinfachter Weise anhand eines Beispiels. Anhand dieser werden die wichtigsten Informationen zum genannten Themengebiet erläutert.

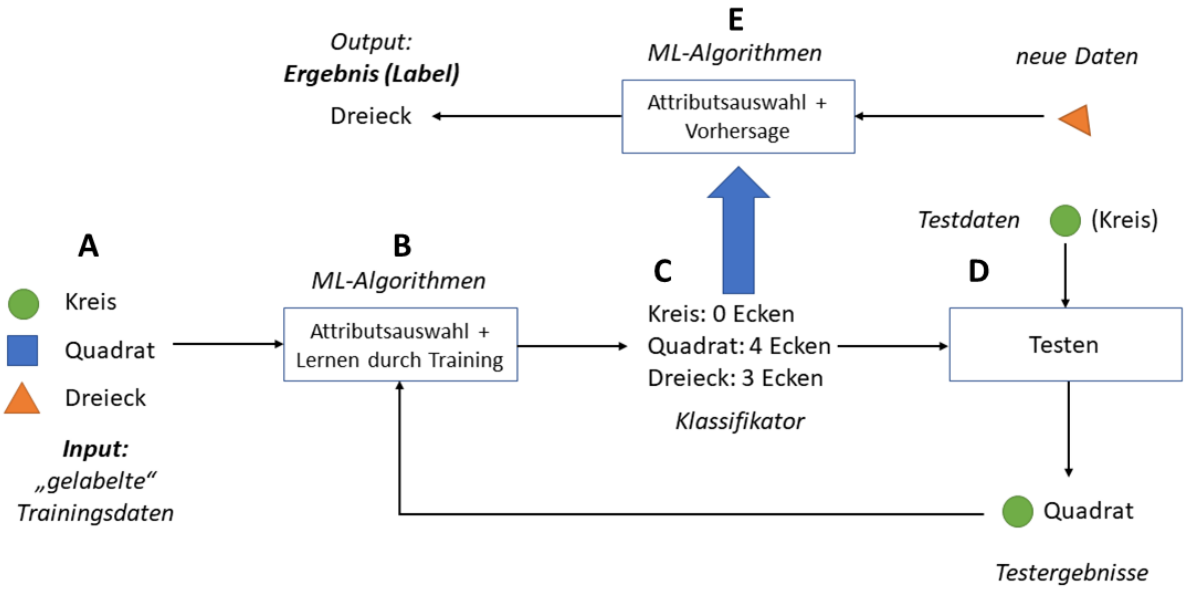


Abbildung 2.1: Allgemeiner Prozess des überwachten Machine Learnings dargestellt anhand eines Beispiels (vereinfacht)

Das in der Abbildung gezeigte Beispiel zeigt den Prozess der Entwicklung und Erlernung eines Klassifikators zur Erkennung von geometrischen Formen. Der Prozess beginnt mit der Erstellung eines Datensets, welches als Input für die Erlernung des Klassifikators dient.

ÜBERARBEITEN!

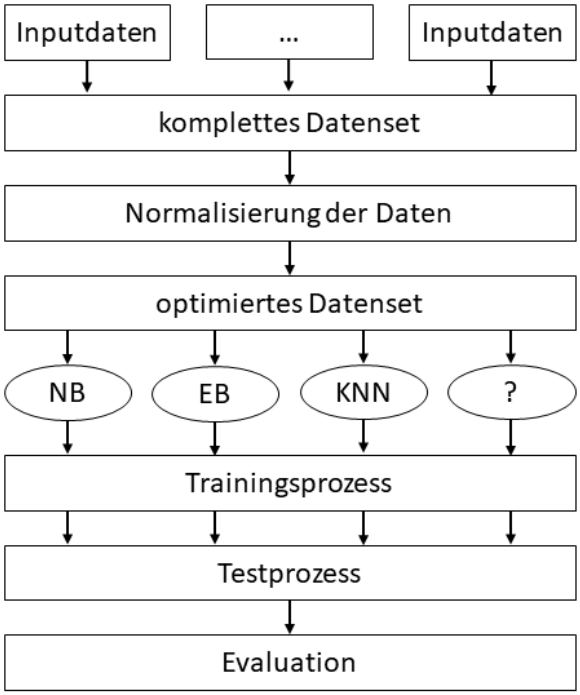


Abbildung 2.2: Angewendeter Prozess zur Durchführung der Klassfikation nach [6]

2.3 Fehlervorhersage mittels Machine Learning

Der Hintergrund zur Fehlervorhersage mittels Machine Learning wird anhand eines Beispiels aus der Literatur erläutert. Es stammt aus einer wissenschaftlichen Arbeit von Queiroz et al. [18] und widmet sich der Fehlervorsage von Features. Bei dieser ersten Fallstudie handelt es sich um die bisher einzige Arbeit mit Bezug zu Software-Features und stellt somit für diese Masterarbeit eine bedeutende literarische Grundlage dar. Der Ablauf des von Queiroz et al. angewandten Prozesses zur Erstellung eines featurebasierten Datensets und dessen Anwendung zur Erlernung von Klassifikatoren orientiert sich am zuvor vorgestellten allgemeinen Prozess des überwachten Machine Learnings.

Die Erläuterung des Beispiels erfolgt anhand von drei Abbildungen, welche den in der Arbeit von Queiroz et al. vorgestellten Prozess in drei Teilen visualisieren.

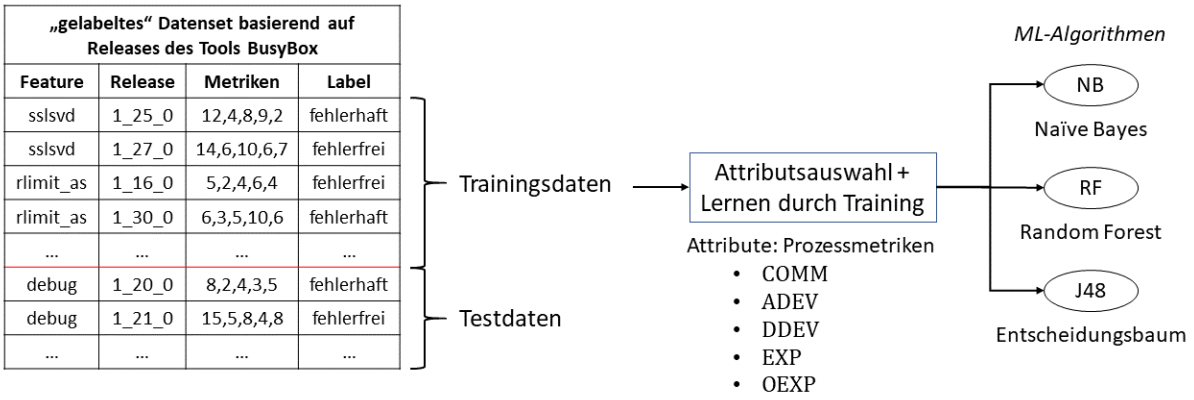


Abbildung 2.3: Teil 1: Featurebasierter Prozess des überwachten Machine Learnings nach [18]

Die Datenbasis des Datensets bilden Commits des UNIX-Toolkits BusyBox[1], dessen Quellcode frei verfügbar in einem Git-Repository[2] eingesehen und von dort geklont werden kann. Diese Commits wurden wiederum ihren entsprechenden Releases zugeordnet, welche auf der vergebenen Tag-Struktur des Repositories beruhen. Ferner wurden aus den Diffs der Commits die dort bearbeiteten Features extrahiert und anschließend zusammen mit den Release-Informationen in einer MySQL-Datenbank gespeichert. Zusätzlich enthält jeder Datenbankeintrag aggregierte Werte von fünf auf das Feature und den Release bezogenen Prozessmetriken (Erläuterung folgt) sowie das binäre Label, ob ein Feature in einem Release fehlerhaft oder fehlerfrei war. Ein Feature gilt in einem Release als fehlerhaft, sofern in einem Commit des darauffolgenden Releases ein fehlerbehebender Commit bezüglich des Features festgestellt werden konnte. Dies geschieht über die Analyse der Commit-Nachrichten. Sofern eine Commit-Nachricht die Begriffe „bug“ (Fehler), „error“ (schwerwiegender Fehler), „fail“ (fehlschlagen) oder „fix“ (beheben) enthält, werten die Autoren des Papers den Commit als fehlerbehebend. Alternative Methoden zur Durchführung dieser Analyse bestehen aus der Eindbindung von Daten aus Bug-Tracking-Systemen, die häufig an Software-Repositories angebunden sind, sowie aus der Anwendung des sogenannten SZZ-Algorithmus, welcher in dieser Arbeit verwendet wurde und in Abschnitt 3.2 erläutert wird [23, 29]. Wie im Rahmen des überwachten Machine Learning üblich, wird das Datenset in Trainings- und Testdaten in einem Verhältnis von 75:25 geteilt.

[1] `https://busybox.net/`
[2] `https://git.busybox.net/busybox/`

Tabelle 2.1: Übersicht der von [18] verwendeten Prozessmetriken

Metrik	Beschreibung
COMM	Anzahl der Commits, die in einem Release dem betreffenden Feature gewidmet sind.
ADEV	Anzahl der Entwickler, die das betreffende Feature in einem Release bearbeitet haben.
DDEV	kummulierte Anzahl der Entwickler, die das betreffende Feature in einem Release bearbeitet haben.
EXP	Geometrisches Mittel der „ Erfahrung“ aller Entwickler, die am betreffenden Feature in einem Release gearbeitet haben.
OEXP	„ Erfahrung“ des Entwicklers, der am meisten zum betreffenden Feature in einem Release beigetragen hat.
Erfahrung ist definiert als Summe der geänderten, gelöschten oder hinzugefügten Zeilen im zugehörigen Release.	

Die Trainingsdaten werden dann den Klassifikatoren zur Erlernung zur Verfügung gestellt. Als Attribute dienen fünf Prozessmetriken mit spezifischer Betrachtung von Software-Features. Tabelle XX gibt einen Überblick über die Beschreibungen dieser. Als Klassifikationsalgorithmen wurden Naïve Bayes, Random Forest und J48-Entscheidungsbäume gewählt. Diese Algorithmen wurden unter anderem auch in dieser Arbeit verwendet. Erläuterungen können in Abschnitt 4.1 gefunden werden.

Wie in Abbildung XX dargestellt ist, wird für jeden Klassifikationsalgorithmus ein Klassifikator erstellt, welcher anschließend getestet und evaluiert wird. Dazu werden die jeweiligen Klassifikatoren auf das Testdatenset angewendet, ohne jedoch die Werte der Zielklassen mit anzugeben. Diese werden im Anschluss an den Klassifikationsvorgang mit den vorhergesagten Werten auf Übereinstimmung verglichen. Anhand dieses Vergleiches können die Genauigkeit sowie weitere Metriken zur Bewertung der Leistung der Klassifikatoren gemessen werden. Eine Übersicht von Evaluationsmetriken kann in Abschnitt 5.2.1 gefunden werden.

Die so erstellten Klassifikatoren können dann zur Vorhersage von neuen Daten genutzt werden. Dazu müssen die fünf zuvor genannten Prozessmetriken der neuen Daten berechnet werden.

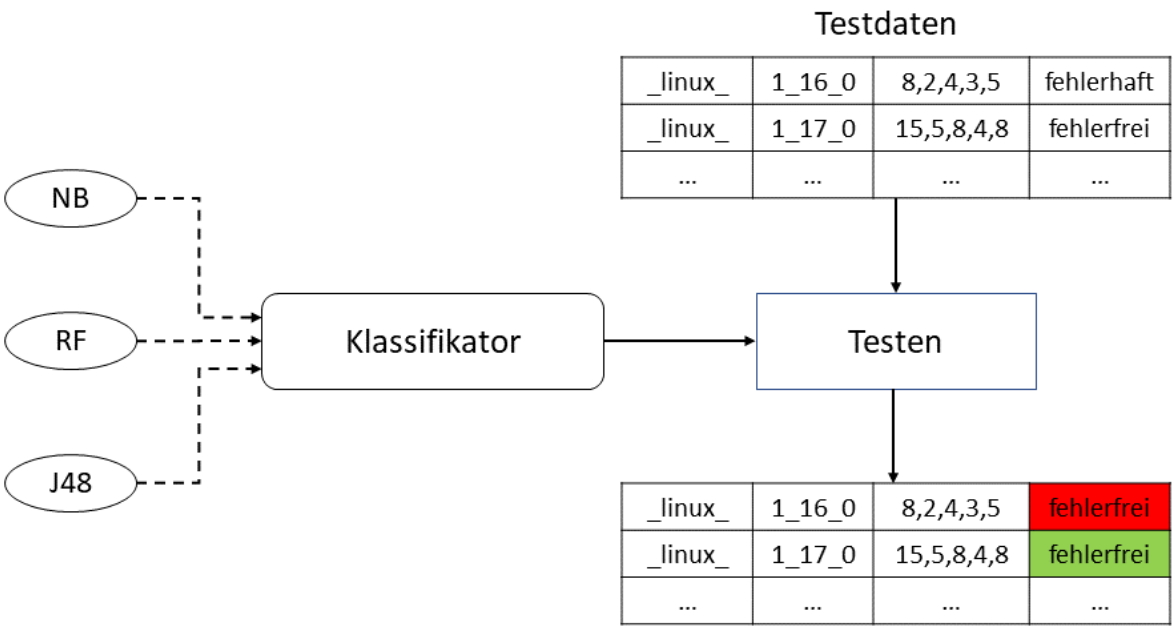


Abbildung 2.4: Teil 2: Featurebasierter Prozess des überwachten Machine Learnings nach [18]

Kapitel 3

Erstellung eines featurebasierten Datensets

Ausblick: Dieses Kapitel widmet sich der schrittweisen Erläuterung des Prozesses zur Erstellung des featurebasierten Datensets, welches zur Anlernung der Machine-Learning-Klassifikatoren dient. Dazu wird zunächst die Datenauswahl näher beleuchtet. Darauf folgt eine Darlegung der Konstruktion des Datensets sowie der Auswahl und Berechnung der Metriken, welche als Attribute (Features) im Rahmen der Anlernung der Klassifikatoren dienen. Eine Gliederung der Kapitel kann Abbildung XX entnommen werden.

Abbildung 3.1: Übersicht zur Gliederung des dritten Kapitels

3.1 Datenauswahl

Wie im vorangegangenen Kapitel bereits erwähnt wurde, bildet das Datenset die Grundlage für die Anlernung der Machine-Learning-Klassifikatoren und wird eigens für diese Arbeit auf Basis von Commits von 13 featurebasierten Software-Projekten erstellt. Die Auswahl der Software-Projekte erfolge anhand von vorheriger Verwendung in wissenschaftlicher Literatur [11, 12, 18]. Die für diese Arbeit verwendeten Software-Projekte sind samt ihres Einsatzzweckes und ihrer Datenquellen in Tabelle XX aufgeführt.

Zum Erhalt der Commit-Daten der Software-Projekte wurde die Python-Library PyDriller[2] verwendet [25]. Diese ermöglicht eine einfache Datenextraktion von Git-Repositories zum Erhalt von Commits, Commit-Nachrichten, Entwicklern, Diffs und mehr. Ein beispielhafter Sourcecode-Ausschnitt zur Konsolenausgabe von Metadaten eines Commits (Autor, Name der veränderten

[1]Links zu den Websites der Softwareprojekte und deren Repositories können im Anhang eingesehen werden.
[2]https://github.com/ishepard/pydriller

Tabelle 3.1: Übersicht der verwendeten Software-Projekten[1]

	Zweck	Datenquelle		Zweck	Datenquelle
Blender	3D-Modellierungstool	GitHub-Mirror	**libxml2**	XML-Parser	GitLab-Repository
Busybox	UNIX-Toolkit	Git-Repository	**lighttpd**	Webserver	Git-Repository
Emacs	Texteditor	GitHub-Mirror	**MPSolve**	Polynomlöser	GitHub-Repository
GIMP	Bildbearbeitung	GitLab-Repository	**Parrot**	virtuelle Maschine	GitHub-Repository
Gnumeric	Tabellenkalkulation	GitLab-Repository	**Vim**	Texteditor	GitHub-Repository
gnuplot	Plotting-Tool	GitHub-Mirror	**xfig**	Grafikeditor	Sourceforge-Repository
Irssi	IRC-Client	GitHub-Repository			

Dateien, Typ der Veränderung und jeweilige zyklomatische Komplexität der Dateien) ist in Listing 3.1 aufgeführt.

```
for commit in RepositoryMining("link_to_repo").traverse_commits():
  for m in commit.modifications:
    print(
          "Author {}".format(commit.author.name),
          " modified {}".format(m.filename),
          " with a change type of {}".format(m.change_type.name),
          " and the complexity is {}".format(m.complexity)
    )
```

Listing 3.1: Beispielhafter PyDriller-Code zur Ausgabe von Metadaten von Commits

Als Input der Python-Scripte zum Erhalt der Commit-Daten dienten jeweils die URLs zu den Git-Repositories der Software-Projekte. Weiterhin wurden die Daten in Commits je Release aufgeteilt. Durchgeführt wurde dies durch die Angabe von Release-Tags, basierend auf der Tag-Struktur von Git-Repositories, im PyDriller-Code. Für jede veränderte Datei innerhalb eines Commits und eines Releases wurden die folgenden Metadaten mit Hilfe von PyDriller abgerufen:

- Commit-Hash (eindeutiger Bezeichner des zugehörigen Commits)
- Autor des zugehörigen Commits
- zugehörige Commit-Nachricht
- Name der veränderten Datei
- Lines-of-Code der veränderten Datei
- zyklomatische Komplexität der veränderten Datei
- Anzahl der hinzugefügten Zeilen zur Datei
- Anzahl der entfernten Zeilen von der Datei
- Art der Änderung (ADD, REM, MOD)[3]
- Diff der Veränderung

Die auf diese Weise erhaltenen Daten wurden nach dem Abruf in einer MySQL-Datenbank gespeichert. Für jedes Software-Projekt wurde eine eigene Tabelle erstellt, in welcher neben der oben stehenden Metadaten zudem der Name des betreffenden Software-Projekts und die den Commits zugehörigen Release-Nummern gespeichert wurden. Jede veränderte Datei eines Commits erhält eine Zeile der Datenbank-Tabellen. In Tabelle XX kann eingesehen werden wie viele Releases je Software-Projekt zum Abruf einbezogen wurden und wie viele Commits daraus resultieren.

Diese „Rohdaten“ dienen zur weiteren Verarbeitung hinsichtlich der Erstellung des Datensets und der anschließenden Berechnung der Metriken. Eine Erläuterung der weiteren Verarbeitung der Daten folgt im kommenden Abschnitt.

[3]Diese Information fand in der weiteren Erstellung des Datensets keine Verwendung.

Tabelle 3.2: Übersicht der Anzahl der Releases und Commits je Software-Projekt

	#Releases	#Commits		#Releases	#Commits
Blender	11	19119	**libxml2**	10	732
Busybox	14	4984	**lighttpd**	6	2597
Emacs	7	12805	**MPSolve**	8	668
GIMP	14	7240	**Parrot**	7	16245
Gnumeric	8	6025	**Vim**	7	9849
gnuplot	5	6619	**xfig**	7	18
Irssi	7	253			

Tabelle 3.3: Übersicht der zur Erstellung des Datensets verwendeten Software-Projekten mit zugehörigen Werten

	Zweck	Datenquelle	#Releases	#Commits	#Korrektiv	#Fehlereinführend	#Features
Blender	3D-Modellierungstool	GitHub-Mirror	11	19119	8333	1418	1400
Busybox	UNIX-Toolkit	Git-Repository	14	4984	1408	142	628
Emacs	Texteditor	GitHub-Mirror	7	12805	6959	685	718
GIMP	Bildbearbeitung	GitLab-Repository	14	7240	1703	272	204
Gnumeric	Tabellenkalkulation	GitLab-Repository	8	6025	1591	136	637
gnuplot	Plotting-Tool	GitHub-Mirror	5	6619	880	1323	558
Irssi	IRC-Client	GitHub-Repository	7	253	77	1	9
libxml2	XML-Parser	GitLab-Repository	10	732	409	37	200
lighttpd	Webserver	Git-Repository	6	2597	1202	555	230
MPSolve	Polynomlöser	GitHub-Repository	8	668	158	69	54
Parrot	Virtuelle Maschine	GitHub-Repository	7	16245	3437	824	397
Vim	Texteditor	GitHub-Repository	7	9849	1033	2571	1158
xfig	Grafikeditor	Sourceforge-Repository	7	18	0	0	137

3.2 Konstruktion des Datensets

Die Konstruktion des Datensets gliedert sich in mehrere Phasen der Datenverarbeitung und -optimierung. Die erste Phase besteht aus der Extraktion der involvierten Features einer veränderten Datei. Dazu wurden mithilfe eines Python-Scripts die sogenannten Präprozessor-Direktiven `#IFDEF` und `#IFNDEF` in den Diffs der veränderten Dateien identifiziert und anschließend die den Direktiven folgende Zeichenfolge bis zum Ende der Codezeile als Feature gespeichert. Die Identifizierung erfolgte mittels regulären Ausdrücken. Gespeichert werden die pro Datei identifizierten Features in einer zusätzlichen Spalte in den jeweiligen MySQL-Tabellen der Software-Projekte. Konnte kein Feature identifiziert werden, wird entsprechend `none` gepseichert.

Dieser Weg der Identifizierung birgt einige Hindernisse. Diese können, neben dem Normalfall, in Abbildung XX gesehen werden. In einigen C-Programmierparadigmen ist es üblich, Header-Dateien mittels Präprozessor-Direktiven in Sourcecode einzubinden, sodass sie wie Features scheinen (siehe erster unerwünschter Fall in Abbildung XX). Diese "Header-Features", wie sie im weiteren Verlauf genannt werden, sollten jedoch ignoriert werden, da sie im Sourcecode keine Variabilität erzeugen. In der Regel sind diese Header-Features identifizierbar durch ihre Namensgebung in Form eines angehängten `_h_` an den Featurenamen, wie beispielsweise `featurename_h_`. Dieser angehängte Teil erlaubt es, die Header-Features mittels regulärer Ausdrücke zu erkennen und auszufiltern.

Ebenfalls besteht die Möglichkeit, dass „falsche“ Features identifiziert werden können. Beispiele dafür können von `#IFDEFs` stammen, welche in Kommentaren verwendet wurden (siehe zweiter unerwünschter Fall in Abbildung XX). Solche falschen Features wurden in einer manuellen Sichtung der identifizierten Features entfernt und durch `none` ersetzt.

```c
int test() {
       #IFDEF print_time
       printf("Current time: %s", time(&now));
       #ENDIF

       printf("Hello World!");
       return 0;
}
```

Normalfall
identifiziertes Feature: `print_time`

```c
#IFDEF time_h_
#INCLUDE <time.h>
#ENDIF

int test() {
       printf("Hello World!");
       return 0;
}
```

Unerwünschter Fall
identifiziertes Feature: `time_h_`
"Header-Features" werden ausgeschlossen

```c
// Maybe #IFDEF to make time optional?
int test() {
       printf("Current time: %s", time(&now));
       printf("Hello World!");
       return 0;
}
```

Unerwünschter Fall
identifiziertes Feature: `to make time optional?`
"falsche" Features werden manuell entfernt

Abbildung 3.2: Normalfall und unerwünschte Fälle bei der Identifizierung von Features

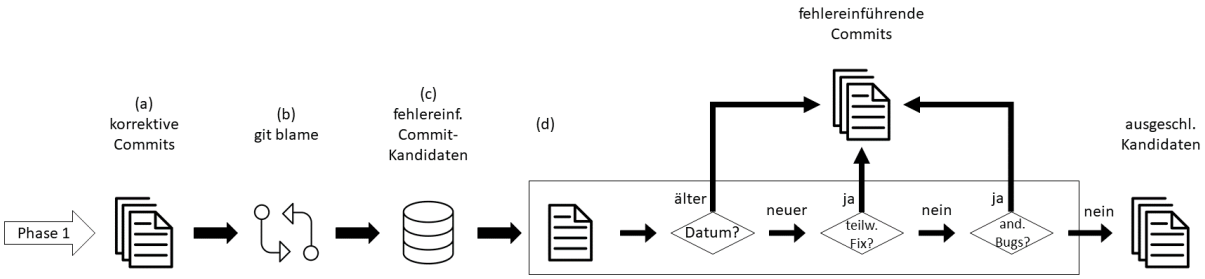


Abbildung 3.3: Ablauf der zweiten Phase des SZZ-Algorithmus (übersetzt, [4])

Die nächste Phase der Verarbeitung besteht aus der Identifizierung von korrektiven Commits. Eine dafür gängige Methode, die auch in dieser Arbeit Anwendung fand, besteht aus der Analyse der Commit-Nachrichten auf das Vorhandensein von bestimmten Schlagworten [29]. Bei den Schlagworten handelt es sich um „bug“, „error“, „fail“ und „fix“. Durchgeführt wurde die Analyse mittels Python-Skripte unter Zuhilfenahme von einfachen Formen des Natural Language Processings. Die Ergebnisse wurden in einer weiteren boole'schen Spalte der MySQL-Tabellen (true = korrektiv, false = nicht korrektiv) gespeichert.

Der Suche nach korrektiven Commits folgt eine Analyse auf fehlereinführende Commits. Dazu wurde eine PyDriller-Implementierung des SZZ-Algorithmus nach Sliwerski, Zimmermann und Zeller verwendet [23]. Dieser ursprünglich für CVS-Versionskontrollsysteme entwickelte Algorithmus erlaubt es, in zwei Phasen fehlereinführende Commits in lokal gespeicherten Software-Repositories zu finden [4]. Die erste Phase besteht dabei aus der Identifizierung der korrektiven Commits. Dies kann entweder anhand der zuvor beschriebenen Analyse der Commit-Nachrichten geschehen oder durch die Analyse von Bug-Tracking-Systemen [4]. Die zweite Phase umfasst die Identifikation der fehlereinführenden Commits auf Basis der zuvor erkannten korrektiven Commits. Diese Phase ist in mehrere Schritte unterteilt und wird in Abbildung XX dargestellt. Die Erläuterungen der mit Buchstaben versehenen Schritte erfolgt im Anschluss. Die PyDriller-Implementierung des Algorithmus folgt dem gezeigten Ablauf.

Die zweite Phase des SZZ-Algorithmus, die als Input eine Liste der Commit-Hashes der zu-

Tabelle 3.4: Übersicht des Schemas der MySQL-Haupttabellen

Spaltenname	Beschreibung	Spaltenname	Beschreibung
name	Name des Softwareprojekts	lines_added	Anzahl der hinzugefügten Zeilen zur geänderten Datei
release_number	zugehörige Release-Version basierend auf vergebenen Tags	lines_removed	Anzahl der entfernten Zeilen von der geänderten Datei
commit_hash	eindeutiger Bezeichner eines Commits	change_type	Art der Änderung
commit_author	Autor eines Commits	diff	Diff der geänderten Datei
commit_msg	Nachricht eines Commits	corrective	Indikator, ob Commit fehlerbehebend war
filename	Name der geänderten Datei	bug_introducing	Indikator, ob Commit fehlereinführend war
nloc	„ Lines of code“ der geänderten Datei	feature	Namen der zugehörigen Features der geänderten Datei
cycomplexity	Zyklomatische Komplexität der geänderten Datei		

vor erkannten korrektiven Commits (a) erhält, beginnt mit der Ausführung eines `git blame` Befehls (b) zur Identifizierung sämtlicher Commits, in denen Veränderungen an den selben Dateien und Codezeilen vorgenommen wurden wie in den korrektiven Commits [4]. Daraus resultieren mögliche fehlereinführende Commit-Kandidaten (c). Für jeden dieser Commit-Kandidaten wird dann erörtert, ob er fehlereinführend ist (d). Dazu wird zunächst das Datum des Commit-Kandidaten mit dem zugehörigen korrektiven Commits verglichen. Liegt dieses vor dem Datum des korrektiven Commits, so gilt der Kandidat als tatsächlich fehlereinführend [4]. Liegt das Datum danach, so kann der Kandidat nur fehlereinführend sein, sofern er teilweise den vorhandenen Fehler löst (teilweiser Fix) oder für einen anderen Fehler verantwortlich ist, der nicht dem korrektiven Commit zugehörig ist (Kandidat ist Fehlerursache eines anderen korrektiven Commits) [4]. Die Ausgabe ist eine Liste von Commit-Hashes von fehlereinführenden Commits für jeden korrektiven Commit. Diese neuen Informationen werden in einer zusätzlichen boole'schen Spalte in den MySQL-Tabellen gespeichert (true = fehlereinführend, false = nicht fehlereinführend).

Eine Übersicht des Schemas der nun vollständigen initialen MySQL-Tabellen (im folgenden Haupttabellen genannt) ist in Tabelle XX aufgeührt. Wie bereits zuvor erwähnt, umfasst diese Tabellle für jede veränderte Datei eines Commits eine Ziele. Sollten in einem Diff einer veränderten Datei mehrere Features identifiziert worden sein, so wird für jedes Feature die entsprechende Zeile dupliziert.

Auf Basis der Daten der Haupttabellen können nun die für das Training der Klassifikatoren benötigten Metriken berechnet werden.

3.3 Metriken

Wie bereits in Kapitel XX erwähnt wurde, bilden sogenannte Metriken die Attribute zum Training der Machine-Learning-Klassifikatoren. Bei Metriken handelt sich sich um Zahlenwerte, die in Codemetriken und Prozessmetriken aufgeteilt sind und jeweils anhand der vorhandenen Daten des Datensets berechnet werden [19]. Codemetriken werden genutzt um Eigenschaften von Sourcecode, wie zum Beispiel „Größe“ oder Komplexität, zu messen [19]. Prozessmetriken dienen hingegen zur Messung von Eigenschaften, die anhand von Metadaten aus Software-Repositories erörtert werden können [19]. Beispiele dafür sind Anzahl der Veränderungen einer bestimmten Datei oder Anzahl der aktiven Entwickler an einem Projekt. Für diese Arbeit

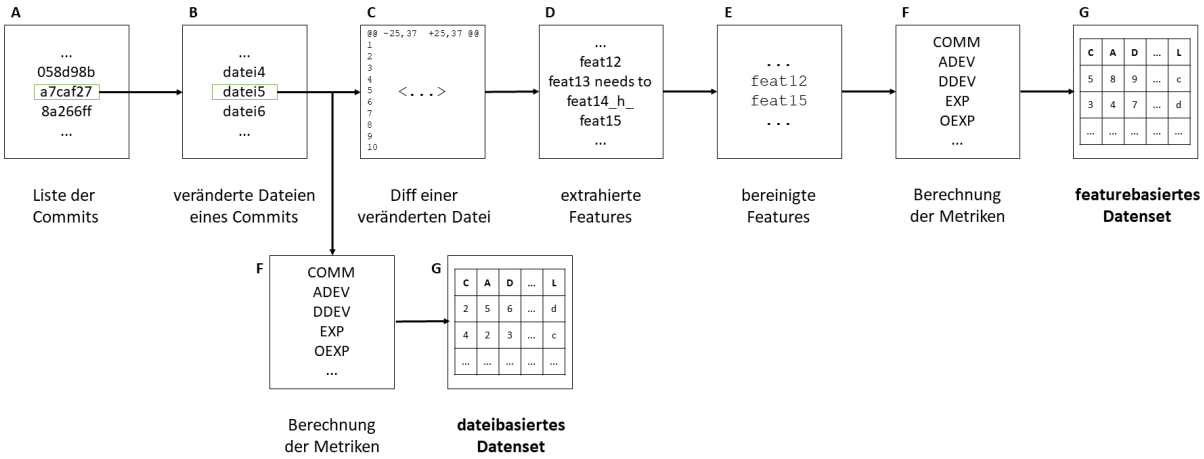


Abbildung 3.4: Visualisierung des Aufbaus und Unterscheidung der Datensets

wurden 11 Metriken errechnet, aufgeteilt in 7 Prozess- und 4 Codemetriken. Fünf der Prozessmetriken wurden aus wissenschaftlichen Arbeiten [19, 18] entnommen. Die weiteren sechs Metriken wurden auf Basis der von PyDriller erhaltenen Metadaten der Commits berechnet.

Im Hinblick auf die spätere Evaluation der Arbeit wurden die Metriken nicht nur auf Basis von Features sondern auch auf Basis von Dateien berechnet. Der letztgenannte Ansatz stellt die in der Machine-Learning-gestützten Fehlererkennung üblicherweise verwendete Methodik dar. Diese beiden Ansätze können somit im Rahmen der Evaluation verglichen werden. Für die Berechnung der dateibasierten Metriken mussten die Daten der Haupttabellen nicht weiter verarbeitet werden, da die erforderlichen Metadaten der Dateien bereits mit PyDriller abgerufen wurden, da sie die Grundlage der Identifikation der Features bildeten. In Abbildung XX wird die Abgrenzung zwischen feature- und dateibasierten Datensets anhand des Ablaufs der Verarbeitung der Rohdaten von PyDriller visualisiert. Es ist zu erkennen, dass für die Berechnung der dateibasierten Metriken keine weiteren Verarbeitungsschritte (Schritte A + B) nötig sind. Lediglich zur Herstellung des Featurebezugs sind weitere Schritte nötig, welche bereits im vorherigen Abschnitt erläutert wurden (Schritte C - E). Die finalen Datensets (G) bestehen aus den jeweils berechneten feature- und dateibezogenen Metriken (E) sowie den Labeln der Zielklasse. Eine Übersicht der berechneten Metriken samt Beschreibung befindet sich in Tabelle XX.

Ergänzen!!!! Feature und File betrachtung

Tabelle 3.5: Übersicht der berechneten Metriken

	Metrik	Beschreibung	Quelle
Prozessmetriken	Anzahl der Commits (COMM)	Anzahl der Commits, die dem Feature / der Datei in einem Release zugeordnet sind.	[19, 18]
	Anzahl der aktiven Entwickler (ADEV)	Anzahl der Entwickler, die innerhalb eines Releases das Feature / die Datei bearbeitet (geändert, gelöscht oder hinzugefügt) haben.	[19, 18]
	eindeutige Entwickleranzahl (DDEV)	kumulierte Anzahl der Entwickler, die innerhalb eines Releases das Feature / die Datei bearbeitet (geändert, gelöscht oder hinzugefügt) haben.	[19, 18]
	Erfahrung aller Entwickler (EXP)	geometrisches Mittel der „Erfahrung“ aller Entwickler, die innerhalb eines Releases das Feature / die Datei bearbeitet (geändert, gelöscht oder hinzugefügt) haben. Erfahrung ist definiert als Summe der geänderten, gelöschten oder hinzugefügten Zeilen in den dem Feature / der Datei zugeordneten Commits.	[19, 18]
	Erfahrung des meist beteiligten Entwicklers (OEXP)	„Erfahrung“ des Entwicklers, die innerhalb eines Releases das Feature / die Datei am häufigsten bearbeitet (geändert, gelöscht oder hinzugefügt) hat. Erfahrung ist definiert als Summe der geänderten, gelöschten oder hinzugefügten Zeilen in den dem Feature / der Datei zugeordneten Commits.	[19, 18]
	Grad der Änderungen (MODD)	Anzahl der Bearbeitungen (Änderung, Entfernung, Erweiterung) des Features / der Datei innerhalb eines Releases.	neu
	Umfang der Änderungen (MODS)	Anzahl der bearbeiteten Features / Dateien innerhalb eines Releases (Feature- bzw. dateiübergreifender Wert). Idee: Je mehr Features / Dateien in einem Release bearbeitet worden, desto fehleranfälliger scheinen diese zu sein.	neu
Codemetriken	Anzahl der Codezeilen (NLOC)	Durchschnittliche Anzahl der Codezeilen der dem Feature zugeordneten Dateien / der Datei innerhalb eines Releases.	neu
	Zyklomatische Komplexität (CYCO)	Durchschnittliche zyklomatische Komplexität der dem Feature zugeordneten Dateien / der Datei innerhalb eines Releases.	neu
	Anzahl der hinzugefügten Zeilen (ADDL)	Durchschnittliche Anzahl der hinzugefügten Codezeilen zu den dem Feature zugeordneten Dateien / zur Datei innerhalb eines Releases.	neu
	Anzahl der entfernten Zeilen (REML)	Durchschnittliche Anzahl der gelöschten Codezeilen von den dem Feature zugeordneten Dateien / von der Datei innerhalb eines Releases.	neu

Tabelle 3.6: Overview of used metrics

	Metric	Description	Source
Process metrics	Number of commits (COMM)	Number of commits associated with the feature/file in a release.	[19, 18]
	Number of active developers (ADEV)	Number of developers who have edited (changed, deleted or added) the feature / file within a release	[19, 18]
	Number of distinct developers (DDEV)	Cumultative number of developers who have edited (changed, deleted or added) the feature / file within a release	[19, 18]
	Experience of all develepoers (EXP)	geometric mean of the experience of all developers who have edited (changed, deleted or added) the feature / file within a release. Experience is defined as the sum of the changed, deleted or added lines in the commits associated with the feature / file.	[19, 18]
	Experience of the most involved developers (OEXP)	Experience of the developer who has edited (changed, deleted or added) the feature / file most often within a release. Experience is defined as the sum of changed, deleted, or added lines in the commits associated with the feature/file.	[19, 18]
	Degree of modifications (MODD)	Number of edits (change, removal, extension) of the feature / file within a release.	new
	Scope of modifications (MODS)	Number of edited features / files within a release (feature or file overlapping value). Idea: The more features / files have been edited in a release, the more error-prone they seem to be.	new
Code metrics	Lines of code (NLOC)	Average number of lines of code of the files associated with the feature / file within a release.	new
	Cyclomatic Complexity (CYCO)	Average cyclomatic complexity of the files associated with the feature / file within a release.	new
	Number of added lines (ADDL)	Average number of lines of code added to the files associated with the feature / file within a release.	new
	Number of removed lines (REML)	Average number of lines of code deleted from the files associated with the feature / from the file within a release	new

Tabelle 3.7: Übersicht des Schemas der Metrics-Tabellen des Datensets

Spaltenname	Beschreibung	Spaltenname	Beschreibung
name	Name des Softwareprojekts	oexp	Ërfahrung"des Entwicklers, der am meisten zum betreffenden Feature / zur betreffenden Datei in einem Release beigetragen hat
release_number	zugehörige Release-Version basierend auf vergebene Tags	scat	Scattering Degree des betreffenden Features / der betreffenden Datei
feature / filename	betreffendes Feature / betreffende Datei	tang	Tangling Degree des betreffenden Features / der betreffenden Datei
comm	Anzahl der Commits, die in einem Release dem betreffenden Feature / der betroffenen Datei gewidmet sind	nloc	Durchschnittliche Lines of Code der Bearbeitungen des betreffenden Features / der betreffenden Datei in einem Release
adev	Anzahl der Entwickler, die das betreffende Feature / die betreffende Datei in einem Release bearbeitet haben	cyco	Durchschnittliche zyklomatische Komplexität der Bearbeitungen des betreffenden Features / der betreffenden Datei in einem Release
ddev	kummulierte Anzahl der Entwickler, die das betreffende Feature / die betreffende Datei in einem Release bearbeitet haben	addl	Durchschnittliche Anzahl der hinzugefügten Zeilen des betreffenden Features / der betreffenden Datei in einem Release
exp	Geometrisches Mittel der Ërfahrung" aller Entwickler, die am betreffenden Feature / an der betreffenden Datei in einem Release gearbeitet haben	reml	Durchschnittliche Anzahl der entfernten Zeilen des betreffenden Features / der betreffenden Datei in einem Release

Kapitel 4

Training und Test der Machine-Learning-Klassifikatoren

Ausblick: Dieses Kapitel gibt einen detaillierten Einblick in das Training der Machine-Learning-Klassifikatoren. Dazu werden zunächst die verwendeten Klassifikatoren und deren initiale Auswahl erläutert. Anschließend werden der Trainingsprozess sowie die zum Einsatz kommenden Softwarewerkzeuge beschrieben.

4.1 Auswahl der Werkzeuge und Klassifikationsalgorithmen

Durch die Wahl der Programmiersprache Python, war die Entscheidung zur Auswahl eines Machine-Learning-Werkzeugs bereits absehbar. Zur Anwendung kommt die Python-Library scikit-learn[1], die im Jahr 2007 von Pedregosa et. al entwickelt wurde [16]. Das Werkzeug bietet eine große Auswahl an Machine-Learning-Algorithmen für überwachtes und unüberwachtes Lernen und ermöglicht darüber hinaus eine einfache Implementation sowie eine einfache Einbindung weiterer Python-Libraries, wie beispielsweise die Matplotlib zur Erstellung von mathematischen Darstellungen [16].

Ebenfalls wird der WEKA-Workbench[2] als weiteres Machine-Learning-Wekzeug verwendet. Im Rahmen der strukturierten Literaturanalyse zu Beginn der Erarbeitung der Masterarbeit, erwies sich dieses Werkzeug durch zahlreiche Zitierungen in wissenschaftlichen Arbeiten (unter anderem in [18]) ebenfalls als geeignet. Der WEKA-Workbench (WEKA als Akronym für Waikato Environment for Knowledge Analysis) wurde an der University of Waikato in Neuseeland entwickelt und bietet eine große Kollektion an Machine-Learning-Algorithmen und Preprocessing-Tools zur Verwendung innerhalb einer grafischen Benutzeroberfläche [10].

Die Verwendung von zwei Machine-Learning-Werkzeugen ermöglicht einen Vergleich der jeweiligen Implementierungen der verwendeten Klassifikationsalgorithmen in der anschließenden Evaluation. Eine Übersicht über die ausgewählten Klassifikationsalgorithmen befindet sich in Tabelle XX. Kurze Erläuterungen der Algorithmen befinden sich im Anschluss.

[1] https://scikit-learn.org/
[2] https://www.cs.waikato.ac.nz/ml/weka/

Tabelle 4.1: Zum Training verwendete Klassifikationsalgorithmen

scikit-learn	**WEKA**
Decision Trees	J48-Decision-Trees
k-Nearest-Neighbors	k-Nearest-Neighbors
Ridge Classifier	Logistic Regression
Naïve Bayes	Naïve Bayes
künstliche neuronale Netze	künstliche neuronale Netze
Random Forest	Random Forest
Stochastic Gradient Descent	Stochastic Gradient Descent
Support Vector Machines	Support Vector Machines

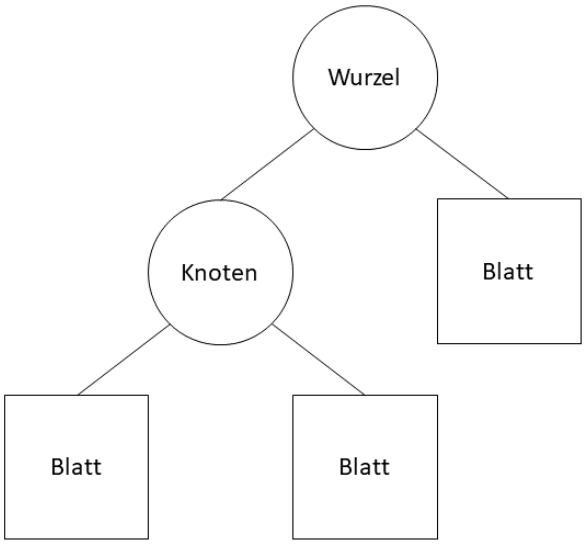


Abbildung 4.1: Grundsätzlicher Aufbau eines Decision Trees

Decision Trees

Überarbeiten?

Decision Trees (deutsch: Entscheidungsbäume) zählen zu den meistverwendeten Klassifikatoren im Bereich des supervised Machine Learnings. Studien belegten, dass sie hinsichtlich der Verwendung im Kontext von Fehlererkennung am häufigsten Anwendung finden [24]. Decision Trees sind gerichtete und verwurzelte Bäume, die als rekursive Partition der Eingabemenge des Datensets aufgebaut wird [21]. Den Ursprung des Baumes bildet die Wurzel, welche keine eingehenden Kanten besitzt - alle weiteren Knoten besitzen jedoch eine eingehende Kante [21]. Diese Knoten teilen wiederum die Eingabemenge anhand einer vorgegebenen Funktion in zwei oder mehr Unterräume der Menge auf [21]. Meist geschieht dies anhand eines Attributs, sodass die Eingabemenge anhand der Werte des einzelnen Attributs geteilt wird [21]. Die Blätter des Baumes bilden die Zielklassen ab. Eine Klassifizierung kann folglich durchgeführt werden, indem man von der Wurzel bis zu einem Blatt den Kanten anhand der entsprechenden Werte der Eingangsmenge folgt. Es existieren verschiedene Algorithmen zur Erstellung von Decision Trees. Bekannte Stellvertreter dieser sind ID3, C4.5 (J48) und CART [21]. Der grundlegende Aufbau eines Decision Trees ist in Abbildung XX dargestellt.

Eine Besonderheit von Decision Trees stellen sogenannte Random Forests dar. Diese beschreiben eine Lernmethode von Klassifikatoren, bei der mehrere einzelne Decision Trees gleichzeitig erzeugt werden und deren Ergebnisse anschließen aggregiert werden [1]. Dazu erhält jeder De-

$$D(p,q) = \sqrt{\sum_{1}^{n} (p_n - q_n)^2}$$

Abbildung 4.2: Formel zur Berechnung der Euklidischen Distanz (n = Anzahl der Attribute)

cision Tree eine Teilmenge der Eingabemenge des Datensets [1]. Random Forests eigenen sich besonders zur Anwendung, wenn viele Attribute im Datenset vorhanden sind [1].

k-Nearest-Neighbors

Ein k-Nearest-Neighbor-Klassifikator (deutsch: k-nächste-Nachbarn) basiert auf zwei Konzepten [28]. Das erste basiert auf der Abstandsmessung zwischen den Werten der zu klassifizierenden Datenmenge und den Werten der Attribute des Datensets [28]. Die Abstandmessung erfolgt in der Regel durch die Berechnung der Euklidischen Distanz (siehe Abbildung XX). Das zweite Konzept bildet der Parameter k, der angibt, wie viele nächste Nachbarn zum Vergleich der zuvor berechneten Abstände in Betracht gezogen werden. Bei einem k > 1 wird diejenige Zielklasse gewählt, deren Auftreten innerhalb der nächsten Nachbarn überwiegt.

Künstliche neuronale Netze

Künstliche neuronale Netze (KNN) verwenden nicht-lineare Funktionen zur schrittweisen Erzeugung von Beziehungen zwischen der Eingabemenge und den Zielklassen durch einen Lernprozess [13]. Sie sind angelehnt an die Funktionsweise von biologischen Nervensystemen und bestehen aus einer Vielzahl von einander verbundenen Berechnungsknoten, den Neuronen [15]. Der grundsätzliche Aufbau eines künstlichen neuronalen Netzes kann in Abbildung XX eingesehen werden. Der Lernprozess besteht aus zwei Phasen - einer Trainingphase und einer Recall-Phase [13]. In der Trainingsphase werden die Eingabedaten, meist als multidimensionaler Vektor, in den Input-Layer geladen und anschließend an die Hidden-Layer verteilt [15]. In den Hidden-Layers werden dann Entscheidungen anhand der Beziehungen zwischen den Eingabedaten und Zielklassen sowie die den Verbindungen zuvor zugewiesenen Gewichtsfaktoren getroffen [13]. **HIER!** [13]

HIER!

Naïve Bayes

Naïve-Bayes-Klassifikatoren zählen zu den linearen Klassifikatoren und basieren auf dem Satz von Bayes. Die Bezeichnung "naivërhält der Klassifikator durch die Annahme, dass Attribute der Eingabemenge unabhängig voneinander sind (diese Annahme wird häufig verletzt, dennoch erzielt der Klassifikator eine hohe Performanz) [20]. Der Klassifikator gilt als effizient, robust, schnell und einfach implementierbar [20]. Die zur Durchführung einer Klassifikation mittels Naïve Bayes benötigte Formel nach Thomas Bayes ist in Abbildung XX samt Erläuterung aufgeführt.

Es existiert zudem eine Mehrzahl an Varianten des Naïve-Bayes-Klassifikators, die verschiedene Annahmen über die Verteilung der Attribute der Eingabemenge machen. Beispiele dafür

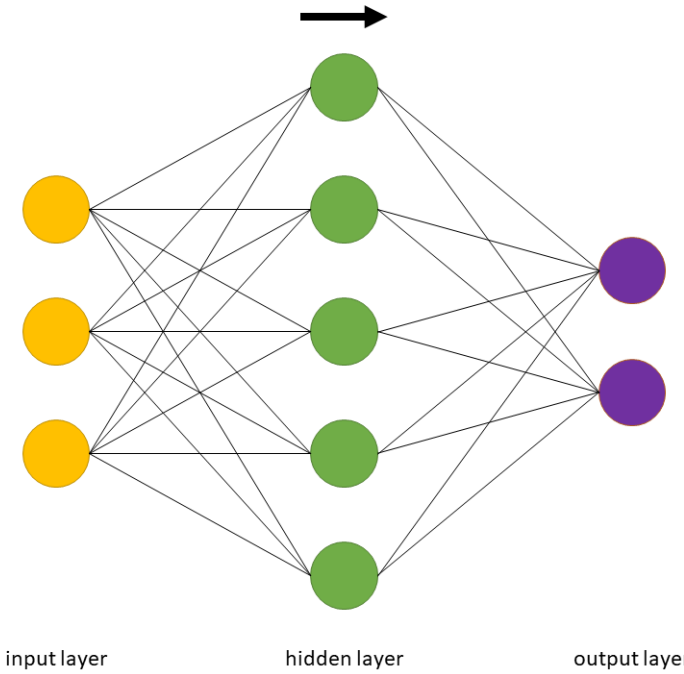


Abbildung 4.3: Grundsätzlicher Aufbau eines KNN mit 4 Input-Neuronen, 5 Hidden-Neuronen und 2 Output-Neuronen

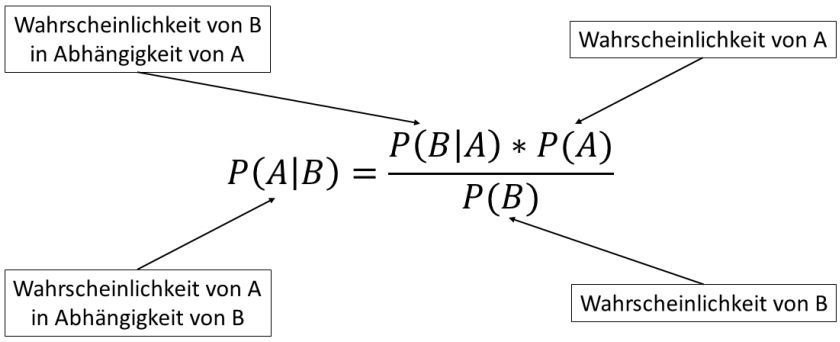


Abbildung 4.4: Satz von Bayes als Grundlage des Naïve-Bayes-Klassifikators

Tabelle 4.2: Zuordnung der verwendeten Abkürzungen

Abkürzung	Klassifikator	Abkürzung	Klassifikator
DT / J48	Decision Trees	RC	Ridge Classifier
KNN	k-Nearest-Neighbor	RF	Random Forest
LR	Logistic Regression	SGD	Stochastic Grandient Descent
NB	Naïve Bayes	SVM	Support Vector Machines
NN	künstliche neuronale Netze		

sind der Gauß'sche-Naïve-Bayes (normalverteilte Attribute), der multinomiale Naïve-Bayes (multinomiale Verteilung der Attribute) sowie der Bernoulli-Naïve-Bayes (unabhängige binäre Attribute).

Logistic Regression

Logistische Regressions-Klassifikatoren basieren auf dem mathematischen Konzept des Logits, welcher den natürlichen Logarithmus eines Chancenverhältnisses beschreibt [17]. Am besten geeignet ist dieser Klassifikator für eine Kombination aus kategorischen oder kontinuierlichen Eingabedaten und kategorischen Zielklassen [17].

HIER

Stochastic Gradient Descent

[5]

Support Vector Machines

Support Vector Machines verfolgen das Ziel, linear separierbare Klassen [27]

4.2 Analyse des Testprozesses

KONFIGURATION DER KLASSIFIKATOREN ERLÄUTERN + FINALE KONFIG ALS TABELLE
SMOTE

Im weiteren Verlauf dieses Abschnitts und im Rahmen der Evaluation im folgenden Kapitel, werden die Namen der Klassifikatoren auf Abbildungen abgekürzt. Die Abkürzungen können Tabelle XX entnommen werden.

Die Analyse des Testprozesses zeigte zudem, dass das dateibasierte Datenset stark unbalanciert hinsichtlich der Zielklasse ist. Mit einem Wert von etwa 98% existieren weitaus mehr Einträge, die dem Label „fehlerfrei" zugeordnet sind. Balanciertheit, also ein ausgeglichenes Verhältnis (50:50 ist im binären Fall nicht zwingend notwendig) innerhalb der Zielklassen, ist jedoch eine Voraussetzung für das korrekte Erlernen der meisten Klassifikatoren. Eine Nichtbeachtung dieses Problem kann zu einer irreführenden Accuracy führen, da die meisten Datensätze korrekt der überrepräsentierten Klasse zugeordnet werden. Als Lösung dieses Problems wurde der sogenannte SMOTE-Algorithmus auf das dateibasierte Datenset angewendet [9]. Der

Algorithmus, dessen Akronym für **S**ynthetic **M**inority **O**ver-sampling **Te**chnique steht, führt ein Oversampling der unterrepräsentierten Klasse durch [9]. Anhand von nächste-Nachbarn-Berechnungen auf Basis der Euklidischen Distanz zwischen den Attributwerten der einzelnen Datensätze des Datensets, werden neue synthetische Datensätze hinzugefügt (Oversampling), sodass sich die Anzahl der Datensätze der relevanten Klasse erhöht [9]. Im hier durchgeführten Fall wurde der Prozentsatz für die Generierung der synthetischen Datensätze auf 3000 festgelegt, sodass für jeden vorhandenen Datensatz der unterrepräsentierten Klasse 30 zusätzliche synthetische Datensätze erzeugt wurden. So konnte der Anteil der Datensätze mit dem Label „fehlerhaft“ auf etwa 27% erhöht werden. In Abbildung XX ist dargestellt, welchen Einfluss die Anwendung des SMOTE-Algorithmus auf die Accuracies der Klassifikatoren des datenbasierten Datensets im Rahmen des Testprozesses hatte. Das mit „vorher“ deklarierte Diagramm zeigt, dass nahezu alle Klassifikatoren eine Accuracy von nahezu 100% besitzen, was das zuvor beschriebene Problem widerspiegelt. Das Diagramm, welches die Testergebnisse nach Anwendung des SMOTE-Algorithmus darstellt, weist hingegen wesentlich realistischere Accuracies auf.

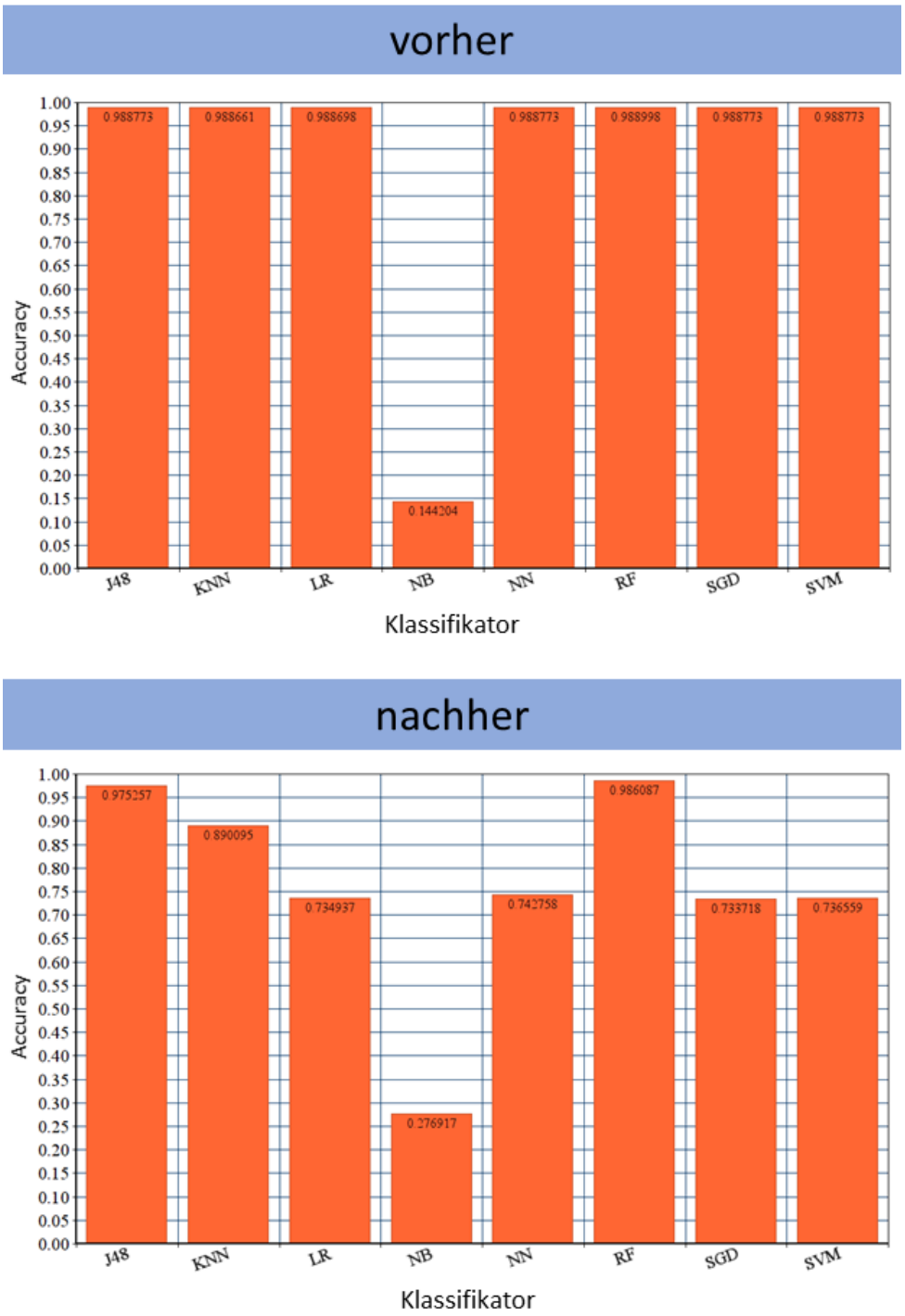


Abbildung 4.5: Vergleich der Accuracies je Klassifikator vor und nach der Anwendung des SMOTE-Algorithmus

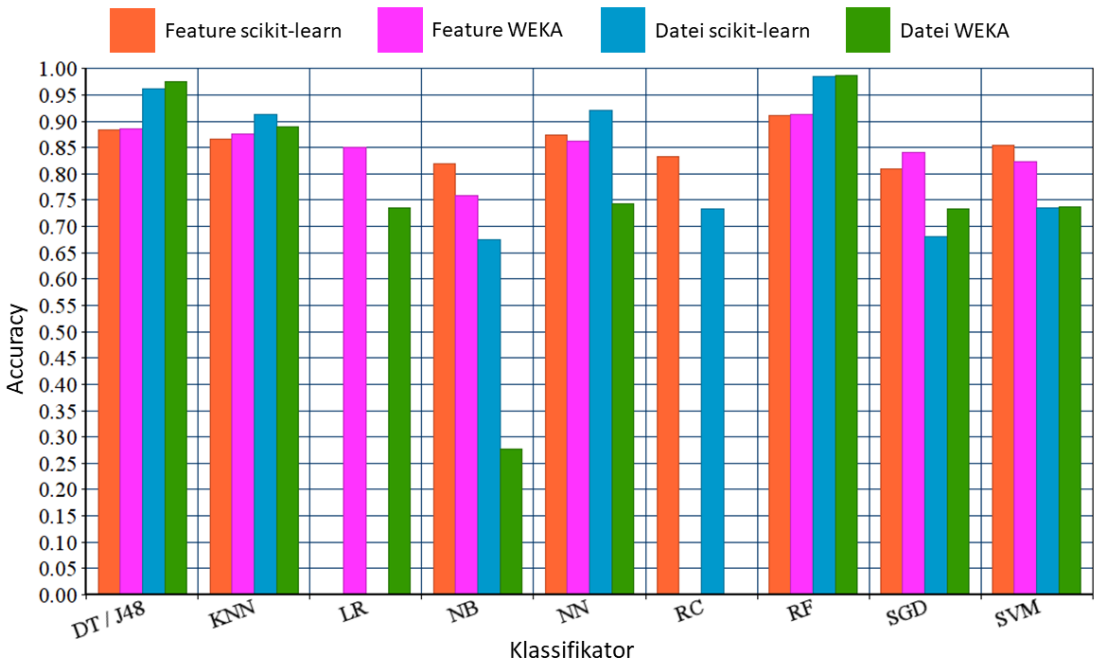


Abbildung 4.6: Vergleich der Klassifikatoren und Werkzeuge im Hinblick auf ihre Accuracies

Kapitel 5

Evaluation

Ausblick: Dieses Kapitel dient der Evaluation der im vorangegangenen Kapitel erläuterten Klassifikationen. Dies geschieht durch verschiedene Evaluationsmetriken, welche in diesem Kapitel vorgestellt werden. Ebenfalls umfasst dieses Kapitel einen Vergleich der Klassifikatoren zu einer nicht-featurebasierten Methode und eine Erläuterung der Herausforderungen und Limitationen, die mit der Erarbeitung der vorangegangenen Kapitel einhergingen.

5.1 Herausforderungen und Limitationen

5.2 Vergleich der Klassifikatoren

Der Vergleich der Klassifikatoren erfolgt unter Zuhilfenahme von Evaluationsmetriken, die im nachfolgenden Abschnitt vorgestellt werden. Die Diskussion der Ergebnisse der Evaluation erfolgt in Abschnitt 5.2.2.

5.2.1 Evaluationsmetriken

Die zum Vergleich der Klassifikatoren erhobenen Evaluationmetriken entstammen dem Themengebiet des Information Retrieval und gelten als Standardmesswerte für ihren Einsatzzweck [22]. Ein Großteil dieser Metriken lässt sich anhand von Werten einer sogenannten Konfusionsmatrix berechnen. Im Falle einer binären Klassifikation, wie in dieser Arbeit, besteht diese Matrix aus vier Gruppen, deren Werte angeben, ob der jeweilige Klassifikator ein Objekt korrekt oder falsch einer der beiden Zielklassen zuordnen konnte [22]. Im Zusammenhang mit solchen Matrizen werden die beiden Zielklassen „positiv“ und „negativ“ genannt. Für diese Arbeit werden die positive Klasse dem Label „fehlerfrei“und die negative Klasse dem Label „defekt“ zugeordnet. Die Form einer allgemeinen Konfusionsmatrix ist in Abbildung XX dargestellt.

Sowohl scitkit-learn als auch WEKA besitzen die Option, Konfisionsmatrizen zu den durchgeführten Tests der Klassifikatoren auszugeben. Anhand der Werte der Zuordnungen zu den Gruppen wurden die folgenden Evaluationsmetriken berechnet:

- Treffergenauigkeit (Accuracy)
  Dieser Wert misst die Treffergenauigkeit der Vorhersagen des Klassifikators und gibt an,

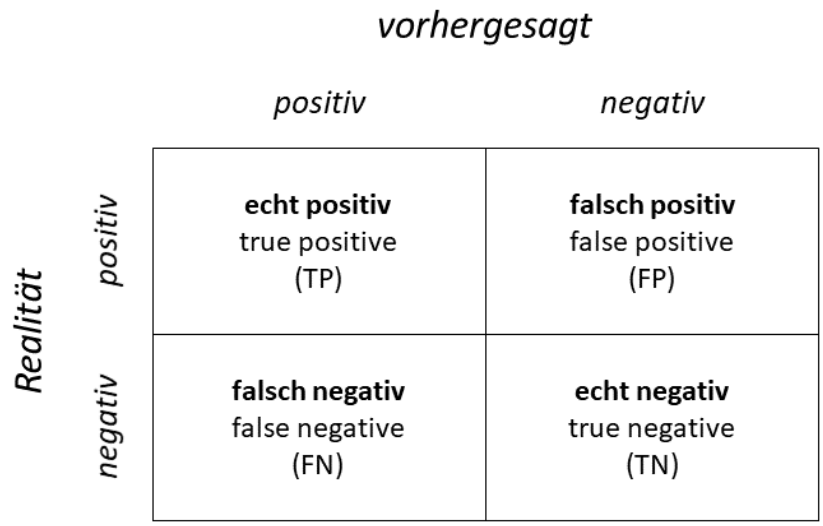


Abbildung 5.1: allgemeine Konfusionsmatrix

inwieweit dessen Vorhersagen mit der modellierten Realität übereinstimmen [22].

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- Echt-Positiv-Rate / Trefferquote (TP-Rate / Recall)
  Dieser Wert gibt den Anteil der korrekt als positiv gewerteten Vorhersagen an. [2]

$$TP - Rate = \frac{TP}{TP + FN}$$

- Falsch-Positiv-Rate (FP-Rate)
  Dieser Wert gibt den Anteil der fälschlicherweise als positiv gewerteten Vorhersagen an. [2]

$$FP - Rate = \frac{FP}{FP + TN}$$

- Positiver Vorhersagewert (Precision)
  Dieser Wert gibt die Anzahl der positiven Vorhersagen an, die auch tatsächlich zur positiven Klasse gehören [22].

$$Precision = \frac{TP}{TP + FP}$$

- F-Maß (F-Score)
  Dieser Wert berechnet das harmonische Mittel zwischen den Werten Precision und Recall und liegt somit zwischen diesen beiden Werten, jedoch näher am kleineren Wert [22].

$$F - Score = \frac{2TP}{2TP + FP + FN}$$

- ROC-Bereich (ROC-Area)
  Dieser Wert unterliegt der Messung der ROC-Kurve (ROC = Receiver operating characteristic, Betriebsverhalten des Empfängers), welche das Verhältnis zwischen der TP-Rate und der FP-Rate modelliert [22]. Die ROC-Area, auch AUC (area under curce, Bereich

Tabelle 5.1: Konfusionsmatrizen (scikit-learn)

		Feat-Datenset			File-Datenset		
	Ermittelt ->	**Fehlerfrei**	**Defekt**	**Total**	**Fehlerfrei**	**Defekt**	**Total**
DT	Realität fehlerfrei	2221	185	2406	13009	138	13147
	Realität defekt	224	388	612	726	4169	4895
	Total	2445	573	3018	13735	4307	18042
KNN	Realität fehlerfrei	2952	110	3062	20398	1592	21990
	Realität defekt	394	317	711	1125	6954	8079
	Total	3346	427	3773	21523	8546	30069
NB	Realität fehlerfrei	2489	533	3022	14618	7395	22013
	Realität defekt	695	56	751	2426	5630	8056
	Total	3184	589	3773	17044	13025	30069
NN	Realität fehlerfrei	4689	168	4857	16959	693	17652
	Realität defekt	480	481	961	1752	4652	6404
	Total	5169	649	5818	18711	5345	24056
RC	Realität fehlerfrei	3036	17	3053	13073	102	13175
	Realität defekt	637	83	720	4705	102	4807
	Total	3673	100	3772	17778	204	17982
RF	Realität fehlerfrei	2372	69	2441	13296	70	13366
	Realität defekt	165	412	577	188	4488	4676
	Total	2357	481	3018	13484	4558	18042
SGD	Realität fehlerfrei	1792	41	1833	7291	10296	17587
	Realität defekt	356	75	431	717	5752	6469
	Total	1833	116	2264	8008	16048	24056
SVM	Realität fehlerfrei	2996	45	3041	21948	210	22158
	Realität defekt	541	191	732	7704	207	7911
	Total	3537	236	3773	29652	417	30069

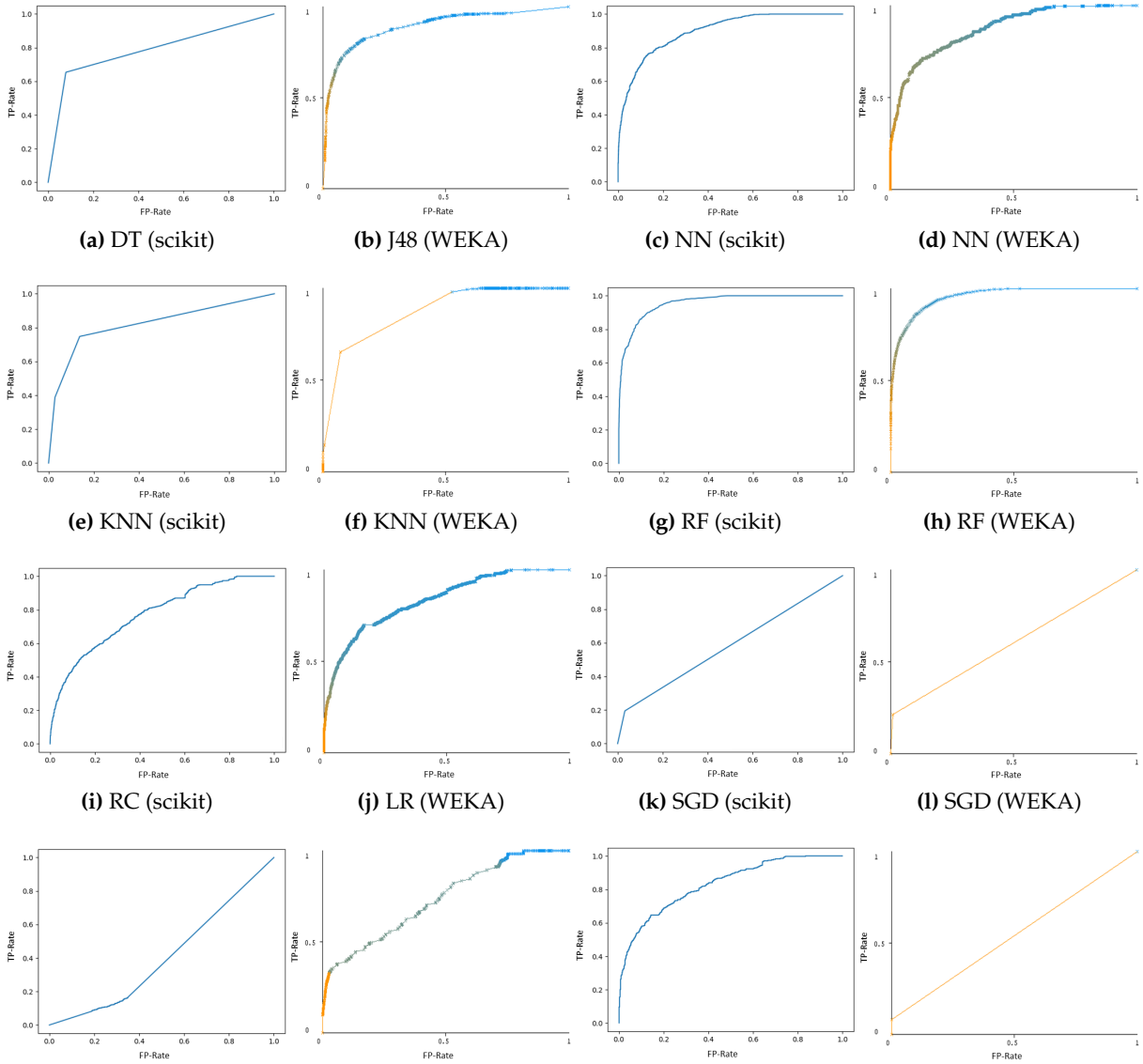
unter der Kurve) genannt, evaluiert den Bereich unter dieser Kurve mit einem Wert zwischen 0 (alle Negativwerte rangieren vor allen Positivwerten) und 1 (alle Positivwerte rangieren vor allen Negativwerten) [22].

- PRC-Bereich (PRC-Area)
  Dieser Wert unterliegt der Messung der PRC (Precision-Recall-Curve), welche die Werte der Precision und des Recalls gegenüberstellt. Die Messung des Bereiches unter dieser Kurve erfolgt analog zur ROC-Area.

5.2.2 Ergebnisse und Diskussion

5.3 Vergleich zu nicht-featurebasierten Methoden

Zum besseren Vergleich der Ergebnisse des Datensets mit neuartiger Fokussierung auf Software-Features, wird nicht nur das unter Zuhilfenahme der gleichen Metriken erstellte dateibasierte Datensets hinzugezogen, sondern zusätzlich ein Datenset, dessen Erstellung aus der wissenschaftlichen Literatur entnommen wurde und sich ebenfalls dem gängigen Weg der dateibasierten Fehlervorhersage widmet. Die von Moser et al. vorgestellte Methode umfasst die Berechnung von 17 Prozessmetriken, welche in Tabelle XX aufgeführt sind. [14]. Die Grundlage der Berechnungen bildeten die aus den Software-Repositories mittels PyDriller erhaltenen Daten. Zur Berechnung der `REVISIONS`-Metrik wurden die Commit-Nachrichten, analog zur Identifikation der fehlerbehebenden Commits, auf das Vorhandensein des Schlagwortes „refactor“ analysiert. Zur Berechnung der Metriken `AGE` und `WEIGHTED_AGE` wurde zudem für jeden Commit das zugehörige Datum der Ausführung abgerufen. Die Berechnung erfolgte entweder direkt mittels SQL-Abrufen oder mithilfe von Python-Skripten.



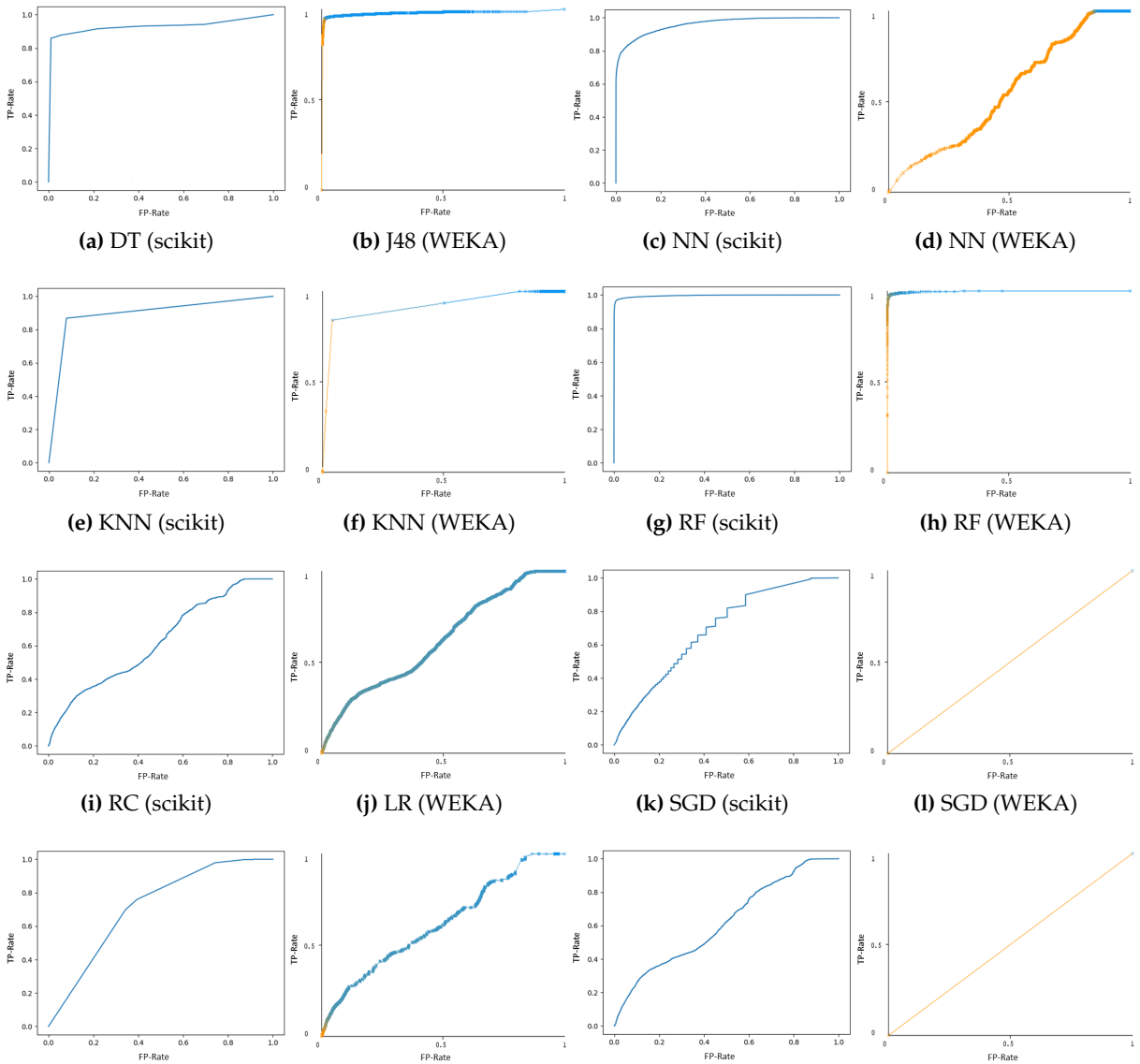
(a) DT (scikit) **(b)** J48 (WEKA) **(c)** NN (scikit) **(d)** NN (WEKA)

(e) KNN (scikit) **(f)** KNN (WEKA) **(g)** RF (scikit) **(h)** RF (WEKA)

(i) RC (scikit) **(j)** LR (WEKA) **(k)** SGD (scikit) **(l)** SGD (WEKA)

(m) NB (scikit) **(n)** NB (WEKA) **(o)** SVM (scikit) **(p)** SVM (WEKA)

Abbildung 5.2: ROC-Kurven der Klassifikatoren des featurebasierten Datensets



(a) DT (scikit) **(b)** J48 (WEKA) **(c)** NN (scikit) **(d)** NN (WEKA)

(e) KNN (scikit) **(f)** KNN (WEKA) **(g)** RF (scikit) **(h)** RF (WEKA)

(i) RC (scikit) **(j)** LR (WEKA) **(k)** SGD (scikit) **(l)** SGD (WEKA)

(m) NB (scikit) **(n)** NB (WEKA) **(o)** SVM (scikit) **(p)** SVM (WEKA)

Abbildung 5.3: ROC-Kurven der Klassifikatoren des dateibasierten Datensets

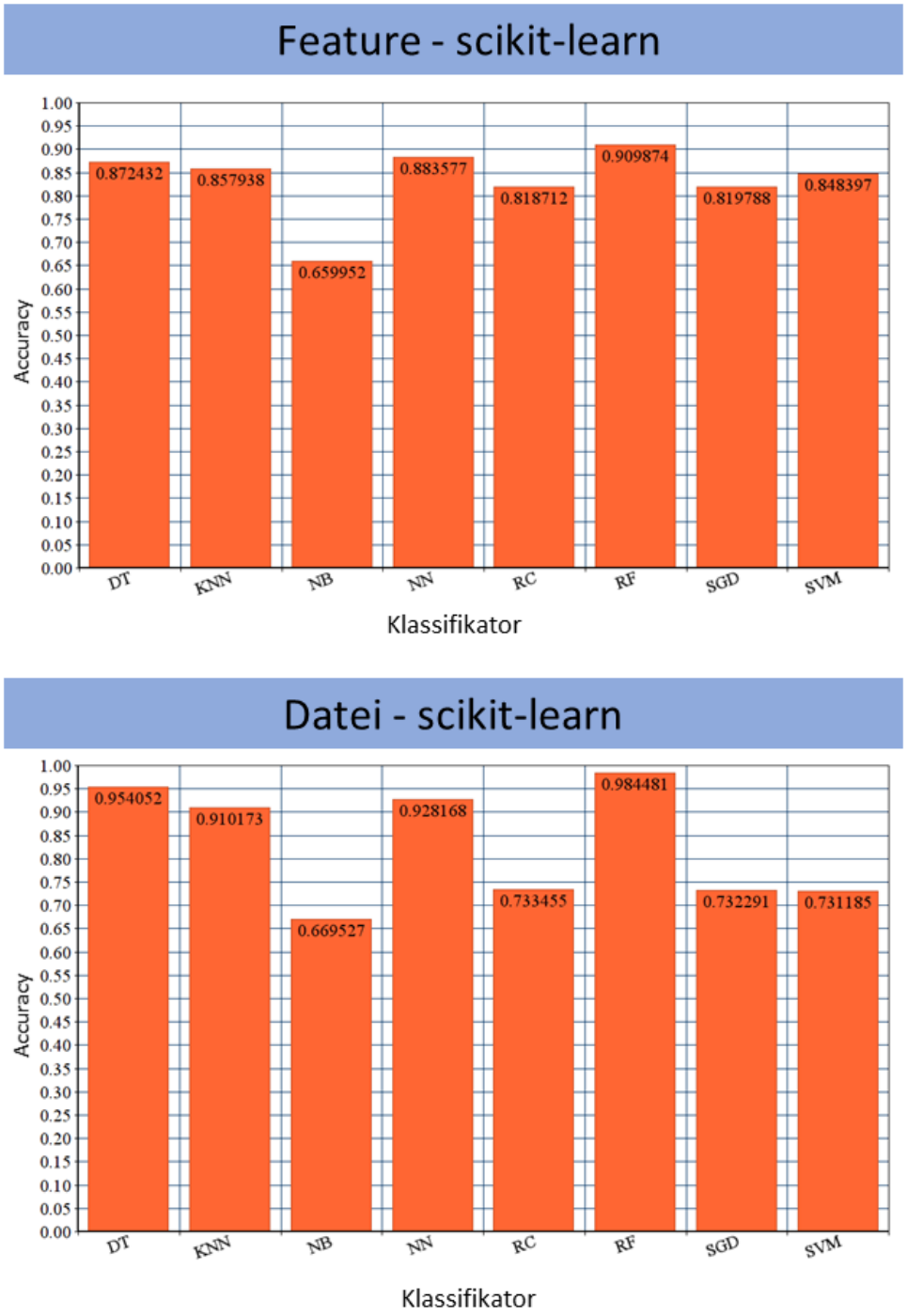
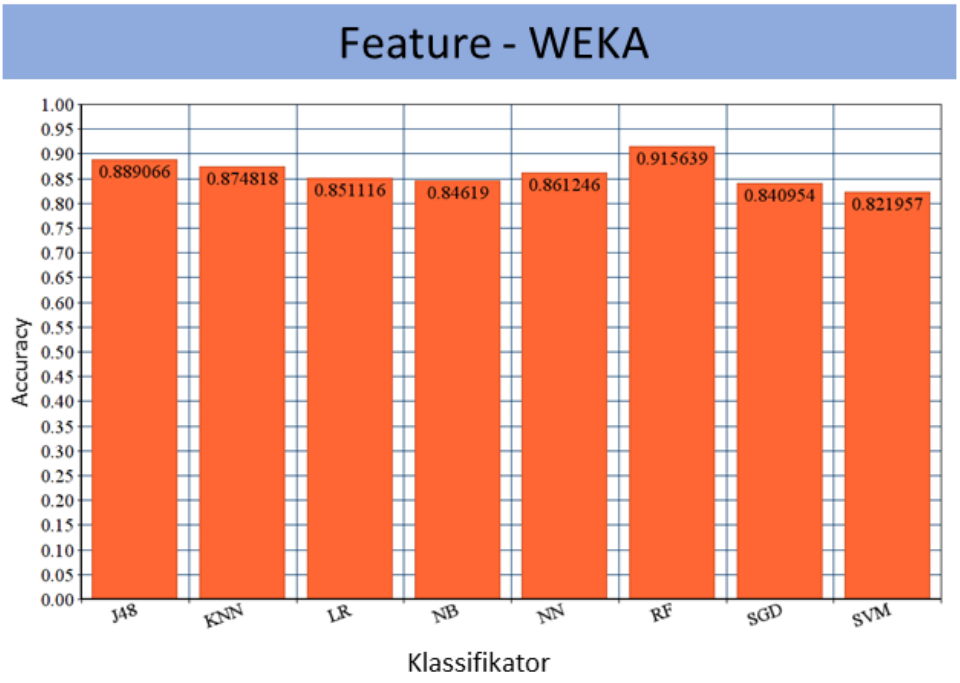


Abbildung 5.4: Vergleich der Accuracies zwischen den Datensets der scikit-Klassifikatoren



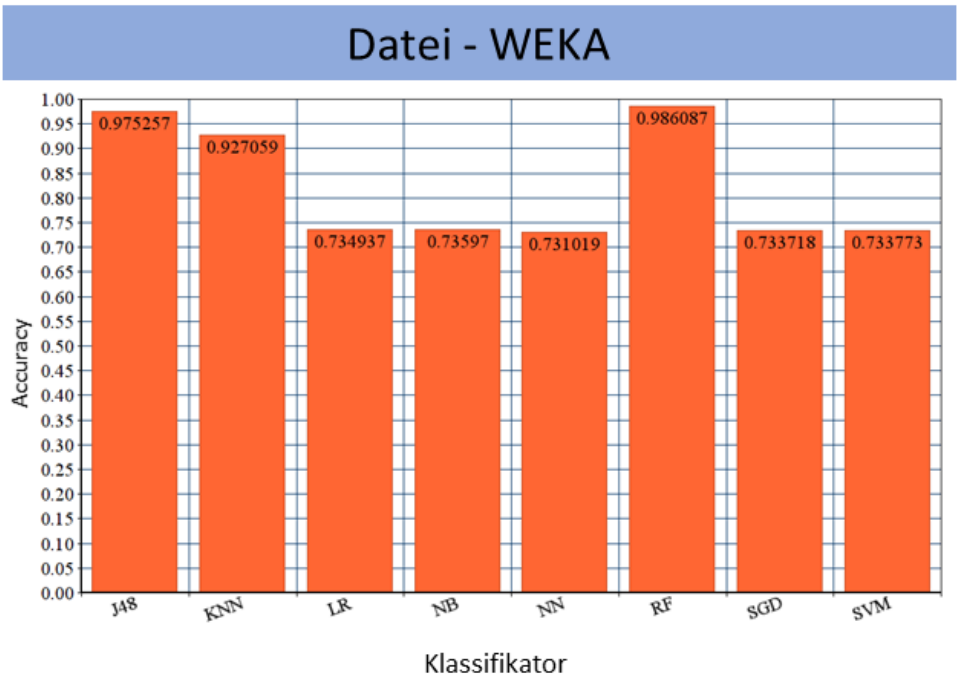


Abbildung 5.5: Vergleich der Accuracies zwischen den Werkzeugen der WEKA-Klassifikatoren

Tabelle 5.2: Konfusionsmatrizen (WEKA)

		Feat-Datenset			File-Datenset		
	Ermittelt ->	**Fehlerfrei**	**Defekt**	**Total**	**Fehlerfrei**	**Defekt**	**Total**
J48	Realität fehlerfrei	11569	580	12149	86805	1231	88036
	Realität defekt	1094	1848	2942	1745	30495	32240
	Total	12663	2428	15091	88550	31726	120276
KNN	Realität fehlerfrei	11284	865	12149	84426	3610	88036
	Realität defekt	1024	1917	2941	5163	27077	32240
	Total	12308	2782	15087	89589	30687	120276
LR	Realität fehlerfrei	3554	103	3657	13081	157	13238
	Realität defekt	571	299	870	4625	178	4803
	Total	4125	402	4527	17706	335	18041
NB	Realität fehlerfrei	11806	343	12149	25807	534	26341
	Realität defekt	1978	963	2941	8993	749	9742
	Total	13784	1306	15090	34800	1283	36083
NN	Realität fehlerfrei	1671	120	1791	21981	0	21981
	Realität defekt	194	278	472	8088	0	8088
	Total	1865	398	2263	30069	0	30069
RF	Realität fehlerfrei	11741	408	12149	13161	77	13238
	Realität defekt	865	2076	2941	174	4629	4803
	Total	12606	2484	15090	13335	4706	19841
SGD	Realität fehlerfrei	3623	34	3657	13237	1	13238
	Realität defekt	686	184	870	4803	0	4803
	Total	4319	218	4537	18040	1	18041
SVM	Realität fehlerfrei	3652	5	3657	13238	0	13238
	Realität defekt	801	69	870	4803	0	4803
	Total	4453	74	7427	18041	0	18041

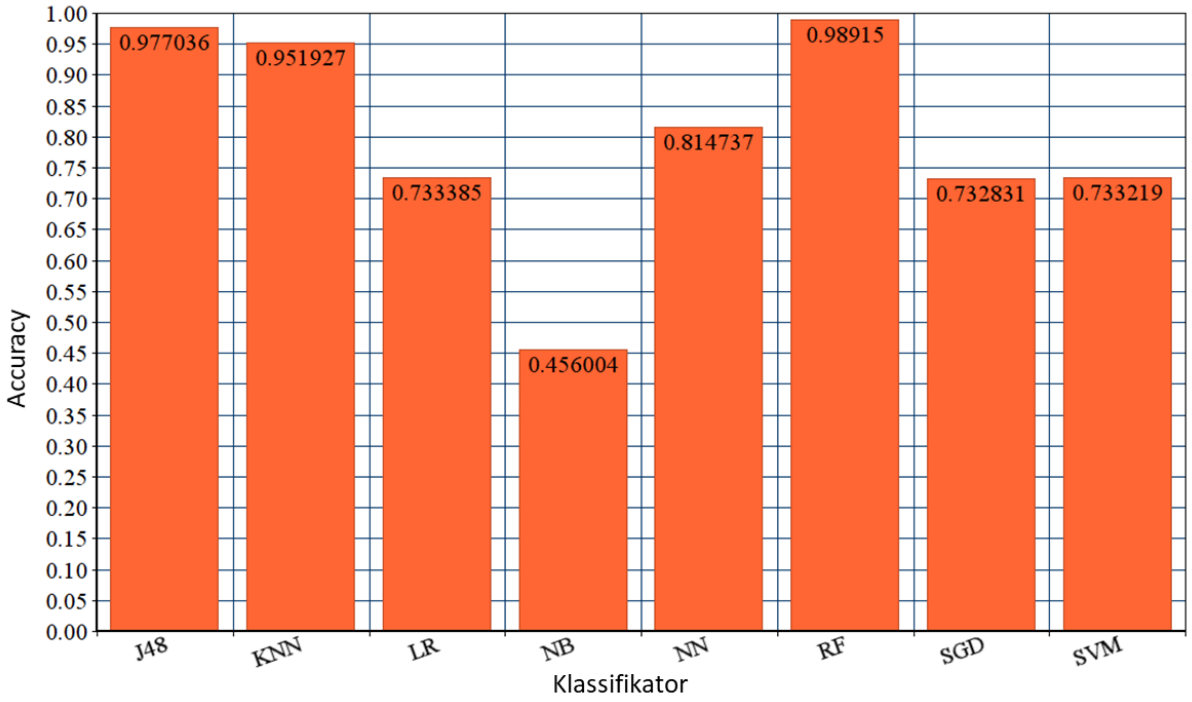


Abbildung 5.6: Übersicht der Accuracies der jeweilgen Klassifikatoren

Tabelle 5.3: Ergebnisse der Evaluationsmetriken auf Basis der Konfusionsmatrix (scikit-learn)

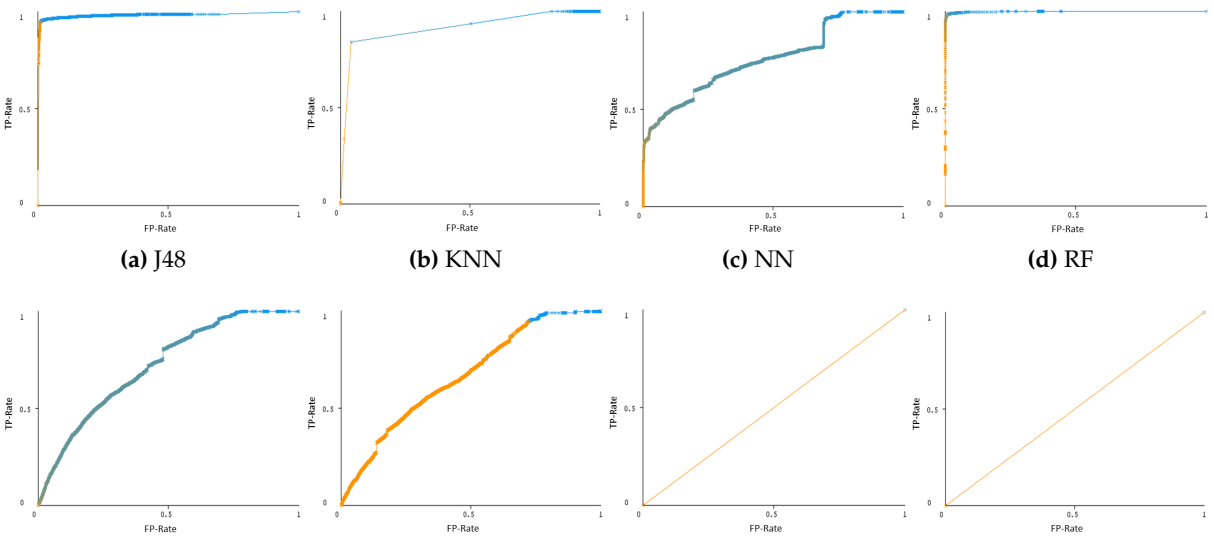
		Feat-Datenset			File-Datenset		
		Fehlerfrei	**Defekt**	**gew. Mittel**	**Fehlerfrei**	**Defekt**	**gew. Mittel**
DT	TP-Rate	0,92	0,63	0,86	0,99	0,85	0,95
	FP-Rate	0,37	0,08	0,31	0,15	0,01	0,11
	Precision	0,91	0,68	0,86	0,95	0,97	0,95
	Recall	0,92	0,63	0,86	0,99	0,85	0,95
	F-Score	0,92	0,65	0,86	0,97	0,91	0,95
	ROC-Area	0,78	0,78	0,78	0,92	0,92	0,92
	PRC-Area	0,76	0,50	0,71	0,72	0,86	0,76
KNN	TP-Rate	0,96	0,45	0,87	0,93	0,86	0,91
	FP-Rate	0,55	0,04	0,46	0,14	0,07	0,12
	Precision	0,88	0,74	0,86	0,95	0,81	0,91
	Recall	0,96	0,45	0,87	0,93	0,86	0,91
	F-Score	0,92	0,56	0,85	0,94	0,84	0,91
	ROC-Area	0,82	0,82	0,82	0,89	0,89	0,89
	PRC-Area	0,79	0,44	0,72	0,69	0,74	0,71
NB	TP-Rate	0,82	0,07	0,67	0,66	0,70	0,67
	FP-Rate	0,93	0,18	0,78	0,30	0,34	0,31
	Precision	0,78	0,10	0,65	0,86	0,43	0,74
	Recall	0,82	0,07	0,67	0,66	0,70	0,67
	F-Score	0,80	0,08	0,66	0,75	0,53	0,69
	ROC-Area	0,40	0,40	0,40	0,72	0,72	0,72
	PRC-Area	0,82	0,19	0,69	0,68	0,38	0,59
NN	TP-Rate	0,97	0,50	0,89~	0,96	0,73	0,90
	FP-Rate	0,50	0,03	0,42	0,27	0,04	0,21
	Precision	0,91	0,74	0,88	0,91	0,87	0,90
	Recall	0,97	0,50	0,89	0,96	0,73	0,90
	F-Score	0,94	0,60	0,88	0,93	0,79	0,90
	ROC-Area	0,90	0,90	0,90	0,95	0,95	0,95
	PRC-Area	0,81	0,45	0,76	0,71	0,71	0,71
RC	TP-Rate	0,99	0,16	0,83	0,99	0,02	0,73
	FP-Rate	0,88	0,01	0,72	0,98	0,01	0,72
	Precision	0,83	0,83	0,83	0,73	0,50	0,67
	Recall	0,99	0,12	0,83	0,99	0,02	0,73
	F-Score	0,90	0,20	0,77	0,84	0,04	0,63
	ROC-Area	0,77	0,77	0,77	0,62	0,62	0,62
	PRC-Area	0,81	0,26	0,70	0,73	0,27	0,61
RF	TP-Rate	0,97	0,71	0,82	0,99	0,96	0,99
	FP-Rate	0,29	0,03	0,24	0,04	0,01	0,03
	Precision	0,93	0,86	0,92	0,99	0,98	0,99
	Recall	0,97	0,71	0,92	0,99	0,96	0,99
	F-Score	0,95	0,78	0,92	0,99	0,97	0,99
	ROC-Area	0,96	0,96	0,96	0,98	0,98	0,98
	PRC-Area	0,79	0,67	0,77	0,74	0,96	0,79
SGD	TP-Rate	0,98	0,17	0,82	0,41	0,89	0,54
	FP-Rate	0,83	0,02	0,67	0,11	0,59	0,54
	Precision	0,83	0,65	0,80	0,91	0,36	0,76
	Recall	0,98	0,17	0,82	0,41	0,89	0,54
	F-Score	0,90	0,27	0,78	0,57	0,51	0,55
	ROC-Area	0,58	0,58	0,58	0,68	0,68	0,68
	PRC-Area	0,80	0,27	0,70	0,68	0,35	0,59
SVM	TP-Rate	0,99	0,26	0,84	0,99	0,03	0,74
	FP-Rate	0,74	0,01	0,84	0,97	0,01	0,72
	Precision	0,85	0,81	0,84	0,74	0,50	0,68
	Recall	0,99	0,26	0,84	0,99	0,03	0,74
	F-Score	0,91	0,39	0,81	0,85	0,05	0,64
	ROC-Area	0,83	0,83	0,83	0,62	0,62	0,62
	PRC-Area	0,80	0,35	0,71	0,73	0,27	0,61

Tabelle 5.4: Ergebnisse der Evaluationsmetriken auf Basis der Konfusionsmatrix (WEKA)

		Feat-Datenset			File-Datenset		
		fehlerfrei	**defekt**	**gew. Mittel**	**fehlerfrei**	**defekt**	**gew. Mittel**
DT	TP-Rate	0,92	0,63	0,89	0,99	0,95	0,98
	FP-Rate	0,37	0,05	0,31	0,05	0,01	0,04
	Precision	0,91	0,76	0,88	0,98	0,96	0,98
	Recall	0,95	0,62	0,89	0,99	0,95	0,98
	F-Score	0,93	0,69	0,89	0,98	0,95	0,98
	ROC-Area	0,89	0,89	0,89	0,98	0,98	0,98
	PRC-Area	0,96	0,71	0,91	0,98	0,62	0,98
KNN	TP-Rate	0,93	0,65	0,88	0,96	0,84	0,93
	FP-Rate	0,35	0,07	0,29	0,16	0,04	0,13
	Precision	0,92	0,69	0,72	0,94	0,88	0,93
	Recall	0,93	0,65	0,88	0,96	0,84	0,93
	F-Score	0,92	0,67	0,87	0,95	0,86	0,93
	ROC-Area	0,87	0,87	0,87	0,91	0,91	0,91
	PRC-Area	0,95	0,59	0,88	0,95	0,80	0,91
LR	TP-Rate	0,97	0,34	0,85	0,99	0,04	0,74
	FP-Rate	0,66	0,03	0,54	0,96	0,01	0,71
	Precision	0,86	0,74	0,84	0,74	0,53	0,68
	Recall	0,92	0,34	0,85	0,99	0,04	0,74
	F-Score	0,91	0,47	0,83	0,85	0,07	0,64
	ROC-Area	0,83	0,83	0,83	0,62	0,62	0,62
	PRC-Area	0,95	0,63	0,89	0,83	0,38	0,71
NB	TP-Rate	0,92	0,33	0,85	0,98	0,08	0,74
	FP-Rate	0,67	0,03	0,55	0,92	0,02	0,68
	Precision	0,86	0,74	0,83	0,74	0,58	0,70
	Recall	0,97	0,33	0,85	0,98	0,08	0,74
	F-Score	0,91	0,45	0,82	0,88	0,14	0,65
	ROC-Area	0,73	0,73	0,73	0,62	0,62	0,62
	PRC-Area	0,91	0,51	0,83	0,82	0,38	0,70
NN	TP-Rate	0,93	0,59	0,86	1,00	0,00	0,73
	FP-Rate	0,41	0,07	0,34	1,00	0,00	0,73
	Precision	0,90	0,70	0,86	0,73	?	?
	Recall	0,93	0,59	0,86	1,00	0,00	0,73
	F-Score	0,91	0,64	0,86	0,85	?	?
	ROC-Area	0,82	0,82	0,82	0,55	0,55	0,55
	PRC-Area	0,96	0,72	0,91	0,80	0,30	0,67
RF	TP-Rate	0,97	0,71	0,92	0,99	0,96	0,99
	FP-Rate	0,29	0,03	0,24	0,04	0,01	0,03
	Precision	0,93	0,84	0,91	0,99	0,98	0,99
	Recall	0,97	0,71	0,92	0,99	0,96	0,99
	F-Score	0,95	0,77	0,91	0,99	0,97	0,99
	ROC-Area	0,96	0,96	0,96	0,99	0,99	0,99
	PRC-Area	0,99	0,87	0,97	1,00	0,99	1,00
SGD	TP-Rate	0,99	0,21	0,84	1,00	0,00	0,73
	FP-Rate	0,79	0,01	0,64	1,00	0,00	0,73
	Precision	0,84	0,84	0,84	0,73	0,00	0,538
	Recall	0,99	0,41	0,84	1,00	0,00	0,734
	F-Score	0,91	0,34	0,80	0,85	0,00	0,621
	ROC-Area	0,60	0,60	0,60	0,50	0,50	0,50
	PRC-Area	0,84	0,33	0,74	0,73	0,27	0,61
SVM	TP-Rate	1,00	0,08	0,82	1,00	0,00	0,73
	FP-Rate	0,92	0,00	0,74	1,00	0,00	0,73
	Precision	0,82	0,93	0,84	0,73	?	?
	Recall	1,00	0,08	0,82	1,00	0,00	0,73
	F-Score	0,90	0,15	0,76	0,85	?	?
	ROC-Area	0,54	0,54	0,54	0,50	0,50	0,50
	PRC-Area	0,82	0,25	0,71	0,73	0,27	0,61

Tabelle 5.5: Übersicht der berechneten Metriken nach [14]

Name	Abkürzung	Beschreibung
REVISIONS	revi	Anzahl der Revisionen (Bearbeitungen) der Datei.
REFACTORINGS	refa	Anzahl der Fälle, in denen die Datei in einem Refactoring involviert war. Basierend auf Analyse der Commit-Nachricht auf das Vorhandensein des Begriffs "refactor".
BUGFIXES	bugf	Anzahl der Fälle, in denen die Datei in einer Fehlerbehebung involviert war.
AUTHORS	auth	Anzahl der verschiedenen Autoren, die die Datei in das Repository eingecheckt haben.
LOC_ADDED	addl	Summe der zur Datei hinzugefügten Codezeilen über alle Revisionen.
MAX_LOC_ADDED	addm	Maximale Anzahl von Codezeilen, die für alle Revisionen hinzugefügt wurden.
AVE_LOC_ADDED	adda	Durchschnittlich hinzugefügte Codezeilen pro Revision.
LOC_DELETED	reml	Summe der von der Datei entfernten Codezeilen über alle Revisionen.
MAX_LOC_DELETED	remm	Maximale Anzahl von Codezeilen, die für alle Revisionen entfernt wurden.
AVE_LOC_DELETED	rema	Durchschnittlich entfernte Codezeilen pro Revision.
CODECHURN	cchl	Summe von (hinzugefügte Codezeilen - entfernte Codezeilen) über alle Revisionen.
MAX_CODECHURN	cchl	Maximaler CODECHURN für alle Revisionen.
AVE_CODECHURN	ccha	Durchschnittlicher CODECHURN pro Revision.
MAX_CHANGESET	maxc	Maximale Anzahl von Dateien, die gemeinsam committed wurden.
AVE_CHANGESET	avgc	Durchschnittliche Anzahl von Dateien, die gemeinsam committed wurden.
AGE	aage	Alter der Datei in Wochen (rückwärts zählend bis zu einem bestimmten Release).
WEIGHTED_AGE	wage	$WeightedAge = \frac{\sum_{i=1}^{N} Age(i) * LOC\_ADDED(i)}{\sum_{i=1}^{N} LOC\_ADDED(i)}$



(a) J48 **(b)** KNN **(c)** NN **(d)** RF

(e) LR **(f)** NB **(g)** SGD **(h)** SVM

Abbildung 5.7: ROC-Kurven der Klassifikatoren

Tabelle 5.6: Konfusionsmatrizen der Evaluation der Klassifikatoren des klassischen Datensets

	Ermittelt ->	**Fehlerfrei**	**Defekt**	**Total**
J48	Realität fehlerfrei	86965	1071	88036
	Realität defekt	1691	30549	32240
	Total	88656	31620	120276
KNN	Realität fehlerfrei	86121	1915	88036
	Realität defekt	3867	28373	32240
	Total	89988	30288	120276
LR	Realität fehlerfrei	13094	134	13228
	Realität defekt	4676	137	4813
	Total	17770	271	18041
NB	Realität fehlerfrei	7232	19102	26334
	Realität defekt	527	9222	9749
	Total	7759	28324	36083
NN	Realität fehlerfrei	30607	153	30760
	Realität defekt	7646	3691	11337
	Total	38253	3844	42097
RF	Realität fehlerfrei	87694	342	88036
	Realität defekt	936	31277	32213
	Total	88630	31619	120249
SGD	Realität fehlerfrei	13221	7	13228
	Realität defekt	4813	0	4813
	Total	18034	7	18041
SVM	Realität fehlerfrei	13228	0	13228
	Realität defekt	4813	0	4813
	Total	18041	0	18041

Tabelle 5.7: Ergebnisse der Evaluationsmetriken auf Basis der Konfusionsmatrix

		fehlerfrei	**defekt**	**gew. Mittel**
DT	TP-Rate	0,99	0,95	0,98
	FP-Rate	0,05	0,01	0,04
	Precision	0,98	0,97	0,98
	Recall	0,99	0,95	0,98
	F-Score	0,98	0,96	0,98
	ROC-Area	0,98	0,98	0,98
	PRC-Area	0,99	0,96	0,98
KNN	TP-Rate	0,98	0,88	0,95
	FP-Rate	0,12	0,02	0,10
	Precision	0,96	0,94	0,95
	Recall	0,98	0,88	0,95
	F-Score	0,97	0,91	0,95
	ROC-Area	0,95	0,95	0,95
	PRC-Area	0,97	0,88	0,95
LR	TP-Rate	0,99	0,03	0,73
	FP-Rate	0,92	0,01	0,72
	Precision	0,74	0,51	0,68
	Recall	0,99	0,03	0,73
	F-Score	0,85	0,05	0,63
	ROC-Area	0,72	0,72	0,72
	PRC-Area	0,88	0,44	0,77
NB	TP-Rate	0,28	0,95	0,46
	FP-Rate	0,05	0,73	0,24
	Precision	0,93	0,33	0,77
	Recall	0,28	0,95	0,46
	F-Score	0,42	0,48	0,44
	ROC-Area	0,66	0,66	0,66
	PRC-Area	0,85	0,40	0,73
NN	TP-Rate	1,00	0,33	0,82
	FP-Rate	0,67	0,01	0,49
	Precision	0,80	0,96	0,84
	Recall	1,00	0,33	0,82
	F-Score	0,89	0,49	0,78
	ROC-Area	0,76	0,76	0,76
	PRC-Area	0,88	0,66	0,82
RF	TP-Rate	1,00	0,97	0,99
	FP-Rate	0,03	0,00	0,02
	Precision	0,99	0,99	0,99
	Recall	1,00	0,97	0,99
	F-Score	0,99	0,98	0,99
	ROC-Area	1,00	1,00	1,00
	PRC-Area	1,00	0,99	1,00
SGD	TP-Rate	1,00	0,00	0,73
	FP-Rate	1,00	0,00	0,73
	Precision	0,73	0,00	0,54
	Recall	1,00	0,00	0,73
	F-Score	0,85	0,00	0,62
	ROC-Area	0,50	0,50	0,50
	PRC-Area	0,73	0,27	0,61
SVM	TP-Rate	1,00	0,00	0,73
	FP-Rate	1,00	0,00	0,73
	Precision	0,73	?	?
	Recall	1,00	0,00	0,73
	F-Score	0,85	?	?
	ROC-Area	0,50	0,50	0,50
	PRC-Area	0,73	0,27	0,61

Kapitel 6

Fazit

Ausblick: Das abschließende Kapitel dieser Arbeit dient zur Zusammenfassung der Ergebnisse der vorangegangenen Kapitel sowie zur Erläuterung der daraus gewonnenen Erkenntnisse. Ebenfalls wird ein Ausblick auf eine mögliche Weiterführung dieser Arbeit gegeben.

6.1 Zusammenfassung und Erkenntnisse

6.2 Ausblick

Literatur

[1] Mohammed S. Alam und Son T. Vuong. „Random Forest Classification for Detecting Android Malware“. In: *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*. IEEE, Aug. 2013. DOI: 10.1109/greencom-ithings-cpscom.2013.122.

[2] Ethem Alpaydin. *Introduction to Machine Learning*. Second Edi. Cambridge, Massachusetts: The MIT Press, 2010. ISBN: 9780262012430.

[3] Sven Apel u. a. *Feature-Oriented Software Product Lines*. Springer Berlin Heidelberg, 2013. DOI: 10.1007/978-3-642-37521-7.

[4] Markus Borg u. a. „SZZ unleashed: an open implementation of the SZZ algorithm - featuring example usage in a study of just-in-time bug prediction for the Jenkins project“. In: *Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation - MaLTeSQuE 2019*. ACM Press, 2019. DOI: 10.1145/3340482.3342742.

[5] Léon Bottou. „Large-Scale Machine Learning with Stochastic Gradient Descent“. In: *Proceedings of COMPSTAT'2010*. Physica-Verlag HD, 2010, S. 177–186. DOI: 10.1007/978-3-7908-2604-3_16.

[6] Evren Ceylan, F. Onur Kutlubay und Ayse B. Bener. „Software Defect Identification Using Machine Learning Techniques“. In: *32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO'06)*. IEEE, Aug. 2006. DOI: 10.1109/euromicro.2006.56.

[7] Venkata Udaya B. Challagulla u. a. „Empirical assessment of machine learning based software defect prediction techniques“. In: *International Journal on Artificial Intelligence Tools* 17.2 (2008), S. 389–400. ISSN: 02182130. DOI: 10.1142/S0218213008003947.

[8] Pete Chapman u. a. „CRISP-DM 1.0“. In: *CRISP-DM Consortium* (2000), S. 76. ISSN: 0957-4174. DOI: 10.1109/ICETET.2008.239.

[9] N. V. Chawla u. a. „SMOTE: Synthetic Minority Over-sampling Technique“. In: *Journal of Artificial Intelligence Research* 16 (Juni 2002), S. 321–357. DOI: 10.1613/jair.953.

[10] Eibe Frank, Mark A. Hall und Ian H. Witten. *The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques"*. Fourth Edition. Morgan Kaufmann, 2016.

[11] Claus Hunsen u. a. „Preprocessor-based variability in open-source and industrial software systems: An empirical study“. In: *Empirical Software Engineering* 21.2 (Apr. 2015), S. 449–482. DOI: 10.1007/s10664-015-9360-1.

[12] Jörg Liebig u. a. „An analysis of the variability in forty preprocessor-based software product lines“. In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE '10*. ACM Press, 2010. DOI: 10.1145/1806799.1806819.

[13] Roland Linder, Jeannine Geier und Mathias Kölliker. „Artificial neural networks, classification trees and regression: Which method for which customer base?“ In: *Journal of Database Marketing & Customer Strategy Management* 11.4 (Juli 2004), S. 344–356. DOI: 10.1057/palgrave.dbm.3240233.

[14] Raimund Moser, Witold Pedrycz und Giancarlo Succi. „A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction“. In: *Proceedings of the 13th international conference on Software engineering - ICSE '08*. ACM Press, 2008. DOI: 10.1145/1368088.1368114.

[15] Keiron O'Shea und Ryan Nash. „An Introduction to Convolutional Neural Networks“. In: (26. Nov. 2015). arXiv: http://arxiv.org/abs/1511.08458v2 [cs.NE].

[16] F. Pedregosa u. a. „Scikit-learn: Machine Learning in Python“. In: *Journal of Machine Learning Research* 12 (2011), S. 2825–2830.

[17] Chao-Ying Joanne Peng, Kuk Lida Lee und Gary M. Ingersoll. „An Introduction to Logistic Regression Analysis and Reporting“. In: *The Journal of Educational Research* 96.1 (Sep. 2002), S. 3–14. DOI: 10.1080/00220670209598786.

[18] Rodrigo Queiroz, Thorsten Berger und Krzysztof Czarnecki. „Towards predicting feature defects in software product lines“. In: *Proceedings of the 7th International Workshop on Feature-Oriented Software Development - FOSD 2016*. ACM Press, 2016. DOI: 10.1145/3001867.3001874.

[19] Foyzur Rahman und Premkumar Devanbu. „How, and why, process metrics are better“. In: *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, Mai 2013. DOI: 10.1109/icse.2013.6606589.

[20] Sebastian Raschka. „Naive Bayes and Text Classification I - Introduction and Theory“. In: (16. Okt. 2014). arXiv: http://arxiv.org/abs/1410.5329v4 [cs.LG].

[21] Lior Rokach und Oded Maimon. „Decision Trees“. In: *Data Mining and Knowledge Discovery Handbook*. Springer-Verlag, 2005, S. 165–192. DOI: 10.1007/0-387-25465-x_9.

[22] Claude Sammut und Geoffrey I. Webb, Hrsg. *Encyclopedia of Machine Learning and Data Mining*. Springer US, 2017. DOI: 10.1007/978-1-4899-7687-1.

[23] Jacek Śliwerski, Thomas Zimmermann und Andreas Zeller. „When do changes induce fixes?“ In: *ACM SIGSOFT Software Engineering Notes* 30.4 (Juli 2005), S. 1. DOI: 10.1145/1082983.1083147.

[24] Le Son u. a. „Empirical Study of Software Defect Prediction: A Systematic Mapping“. In: *Symmetry* 11.2 (Feb. 2019), S. 212. DOI: 10.3390/sym11020212.

[25] Davide Spadini, Mauricio Aniche und Alberto Bacchelli. „PyDriller: Python framework for mining software repositories“. In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2018*. ACM Press, 2018. DOI: 10.1145/3236024.3264598.

[26] Thomas Thüm u. a. „A Classification and Survey of Analysis Strategies for Software Product Lines“. In: *ACM Computing Surveys* 47.1 (Juni 2014), S. 1–45. DOI: 10.1145/2580950.

[27] Angelos Tzotsos. „A Support Vector Machine Approach for Object Based Image“. In: *Proceedings of 1st International Conference on Object-based Image Analysis* Negnevitsky (2006), S. 4–5.

[28] Zhongheng Zhang. „Introduction to machine learning: k-nearest neighbors“. In: *Annals of Translational Medicine* 4.11 (Juni 2016), S. 218–218. DOI: 10.21037/atm.2016.03.37.

[29] Thomas Zimmermann, Rahul Premraj und Andreas Zeller. „Predicting Defects for Eclipse“. In: *Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007)*. IEEE, Mai 2007. DOI: 10.1109/promise.2007.10.

Anhang A

Links der für die Erstellung des Datensets verwendeten Software-Projekte

	Link zur Website	Link zum Repository
Blender	https://www.blender.org/	https://github.com/sobotka/blender
Busybox	https://busybox.net/	https://git.busybox.net/busybox/
Emacs	https://www.gnu.org/software/emacs/	https://github.com/emacs-mirror/emacs
GIMP	https://www.gimp.org/	https://gitlab.gnome.org/GNOME/gimp
Gnumeric	http://www.gnumeric.org/	https://gitlab.gnome.org/GNOME/gnumeric
gnuplot	http://gnuplot.info/	https://github.com/gnuplot/gnuplot
Irssi	https://irssi.org/	https://github.com/irssi/irssi
libxml2	http://www.xmlsoft.org/	https://gitlab.gnome.org/GNOME/libxml2
lighttpd	https://www.lighttpd.net/	https://git.lighttpd.net/lighttpd/lighttpd1.4.git/
MPSolve	https://numpi.dm.unipi.it/software/mpsolve	https://github.com/robol/MPSolve
Parrot	http://parrot.org/	https://github.com/parrot/parrot
Vim	https://www.vim.org/	https://github.com/vim/vim
xfig	https://sourceforge.net/projects/mcj/	https://sourceforge.net/p/mcj/xfig/ci/master/tree/
Websites zuletzt abgerufen am 13. Januar 2020.		

Anhang B

Test 2

Lorem ipsum