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 (Features sind „cross-cutting“ (dateiübergreifend)). 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 (features are „cross-cutting“). 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 der Arbeit zugrunde liegenden Strukturen erläutert. Dazu gehören die Forschungsziele und Forschungsfragen, das verwendete Forschungsdesign, eine Übersicht der geplanten und tatsächlichen Zeitplanung sowie eine Erläuterung des Aufbaus der weiterführenden Kapitel 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 oder mehrere 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 [6]. 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 [3, 11]. Dieser kann dann anschließend verwendet werden, um eine akkurate Vorhersage zu erhalten, ob eine neu erfolgte Änderung an einer Software fehlerhaft oder frei von Fehlern ist.

Die Auswahl an Lernverfahren für Klassifikatoren ist groß. Studien zeigen, dass aus dem Pool von verfügbaren Verfahren sowohl Entscheidungsbaum-basierte (zum Beispiel J48, CART oder Random Forest) als auch Bayessche Verfahren die meistgenutzten sind [29]. Alternative Lernmethoden sind beispielsweise Regression, k-Nearest-Neighbors oder künstliche neuronale Netze [6]. 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 Features 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

[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.

sich dadurch aus, dass sie eine gemeinsame Menge von Features sowie eine gemeinsame Codebasis besitzen [33]. Durch das Vorhandensein verschiedener Features entlang der Softwareprodukte, kann eine breite Variabilität innerhalb einer Produktlinie erreicht werden. Eine detaillierte Einführung in den Themenkomplex von Software-Produktlinien und featurebasierter Softwareentwicklung kann in Abschnitt 2.1 gefunden werden. Bei der Entwicklung der Vorgersagetechnik wird die Implementation von Features mittels Präprozessor-Anweisungen, wie `#IFDEF` und `#IFNDEF` (auch Präprozessor-Direktiven genannt) betrachtet. Dieser bisher in wissenschaftlichen Arbeiten nur einmal im Rahmen einer Fallstudie betrachtete Ansatz [22] 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 featuresspezifischen Code enthält (insbesondere die sogenannten Feature-Interaktionen), ist möglicherweise fehleranfälliger als sonstiger Code.

Ein initiales und einfaches Beispiel für einen Softwarefehler innerhalb eines Features ist in Listing 1.1 dargestellt. Es ist zu erkennen, dass innerhalb des Codes des Features `print_time` die `printf`-Anweisung einen Schreibfehler enthält, der dazu führt, dass der Featurecode nicht ausgeführt werden kann. Ferner kann fehlerhafter Featurecode dazu führen, dass die gesamte Funktionalität des Features und gegebenenfalls der gesamten Software beeinträchtigt oder verhindert wird.

```
1 int test() {
2     #IFDEF print_time
3     prntf("Current time: %s", time(&now));
4     #ENDIF
5
6     printf("Hello World!");
7     return 0;
8   }
```

Listing 1.1: Exemplarische Darstellung eines fehlerhaften Features

Das zuvor genannte Ziel der Arbeit setzt sich aus mehreren Teilzielen zusammen. Dazu zählen die Erstellung eines Datensets unter Einbezug des Feature-Aspekts. Dieses Datenset dient wiederum zur Anlernung von einer repräsentativen Auswahl an Klassifikatoren mit anschließender vergleichender Evaluation dieser. Zusätzlich werden die featurebasierten Klassifikatoren mit klassischen dateibasierten Klassifikatoren verglichen, dessen Entwicklung aus einer wissenschaftlichen Arbeit entnommen wurde [17]. Ein genauer Überblick über die Forschungsziele befindet sich im nächsten Abschnitt.

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 durch Variabilität geprägten Codebasis zu erhalten.

1.1 Forschungsziele und Forschungsfragen

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 vorhergesehen, das Augenmerk auf Commits des Versionierungssystems 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, für ausstehende oder zukünftige Commits akkurate Vorhersagen zu treffen, ob diese Fehler beinhalten, um so 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. Die Beantwortung der Forschungsfragen erfolgt im weiteren Verlauf der Arbeit. Erkennbar sind die Antworten auf die Fragen an ihrer Einrahmung.

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: Evaluation 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:

- Vorbereitung
- Abschluss des ersten Forschungsziels (*Erstellung des Datensets*)
- Abschluss des zweiten Forschungsziels (*Training von Machine-Learning-Klassifikatoren*)
- Abschluss des dritten Forschungsziels (*Evaluation und Vergleich*)
- Nachbereitung

Diese Arbeitsphasen werden in den kommenden Abschnitten näher erläutert. Als finale Vorhersagetechnik wird jener Klassifikator verwendet, der sich im Rahmen der Gegenüberstellung im Verlauf der Evaluation als am „effektivsten“ erweist. Die Kriterien für die Beurteilung der Effektivität eines Klassifikators werden im Kapitel „Evaluation“ erläutert.

1.2 Forschungsdesign

Die für diese Arbeit gewählte Methodik basiert auf dem Prozessmodell „Cross-Industry Standard Process for Data Mining“, kurz CRISP-DM, nach Chapman et al.[7]. Es wird als Vorlage für die Arbeitsphasen zur Erreichung der Forschungsziele dieser Arbeit verwendet. Da sich der überwiegende praktische Teil dieser Arbeit auf Programmierung im Bereich des Machine Learning konzentriert, bildet das CRISP-DM Prozessmodell ein passendes vordefiniertes Vorgehen. Eine grafische Aufarbeitung des Prozessmodells mit seinen sechs zugehörigen Phasen sowie den Verbindungen zwischen ihnen ist in Abbildung 1.1 dargestellt.

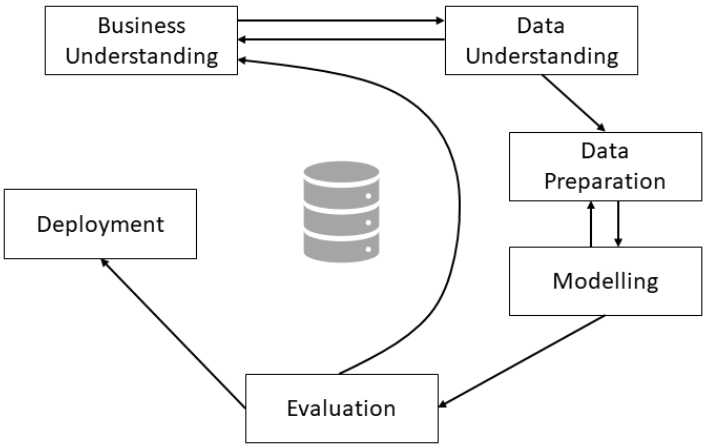


Abbildung 1.1: CRISP-DM Prozessmodell nach [7]

Das CRISP-DM-Prozessmodell wurde, wie der ausgeschriebene Name bereits andeutet, ursprünglich 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 1.2 dargestellt. Zusätzlich umfasst diese Abbildung die Zuordnung der Arbeitsphasen, die im vorherigen Abschnitt 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 der im Anschluss folgende Abschnitt.

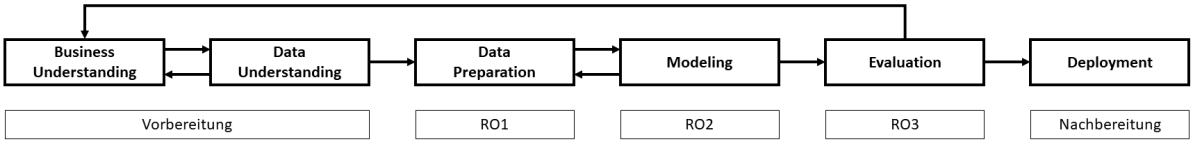


Abbildung 1.2: Phasen des CRISP-DM Prozessmodells nach [7] mit Zuordnung der Arbeitsphasen

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 begann und somit das Verfassen des Proposals sowie das Absolvieren der Einführungspräsentation zu dieser Phase gezählt werden können. Die darauffolgende Phase *Data Understanding* dient der Suche und

Einsicht von für den weiteren Verlauf der Phasen relevanten Daten und, falls vorhanden, vorgefertigten Datensets (es konnten keine Datensets gefunden werden). Da Commits als Datenbasis zur Erlernung der Klassifikatoren betrachtet werden, wird der überwiegende Teil der Suche nach Daten in Software-Repositories stattfinden, welche dem Versionierungssystem Git zugrunde liegen. 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 des featurebasierten 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 auf Machine-Learning-Algorithmen basierenden Klassifikatoren mithilfe des Datensets trainiert und anschließend getestet, um Anpassungen an den Algorithmen hinsichtlich einer höheren Genauigkeit der Vorhersagen vornehmen zu können. Diese Phase spiegelt somit die Erfüllung des zweiten Forschungsziels wider. 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 bzw. Finalisierung der schriftlichen Ausarbeitung sowie das Erstellen der Abschlusspräsentation und der anschließenden Vorführung dieser im Rahmen des Kolloquiums. Ferner können in Abbildung 1.2 Rückpfeile zwischen einzelnen Phasen erkannt werden. So können beispielsweise Erkenntnisse im Rahmen der Phase Data Understanding zu offenen Fragestellungen führen, die die Phase des Business Understanding betreffen. Ebenfalls können im Rahmen der Phase Modeling gewonnene Erkenntnisse dazu führen, dass ein Sprung zurück in die vorherige Phase Data Preparation nötig ist, da festgestellt worden ist, dass zusätzliche Bearbeitungsschritte der Daten erforderlich sind. Weiterhin können Erkenntnisse, die im Rahmen der Evaluationsphase gewonnen werden können, zuvor unbekanntes Wissen aus der ersten Phase erläutern.

1.3 Zeitplanung

ANPASSEN AN TATSÄCHLICHEN ABLAUF

Die nachfolgenden fünf Tabellen zeigen den im Rahmen der Vorbereitung der Arbeit geplanten Ablauf der zu erreichenden Ziele und Unterziele, aufgeteilt in die fünf Arbeitsphasen inklusive der Zuordnung der sechs Phasen des CRISP-DM-Prozessmodells. Die geschätzte und tatsächliche Dauer der jeweiligen Unterziele wird jeweils in Tagen, Wochen oder Monaten angegeben. Zur Verfügung stehen, gemäß der Vorgaben der Prüfungsordnung, 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. Eine Visualisierung des geplanten zeitlichen Verlaufs anhand eines Zeitstrahls als Gantt-Chart inklusive konkreter Datumsangaben ist in Abbildung 1.3 dargestellt. Der tatsächliche zeitliche Ablauf wird in Abbildung 1.4 gezeigt.

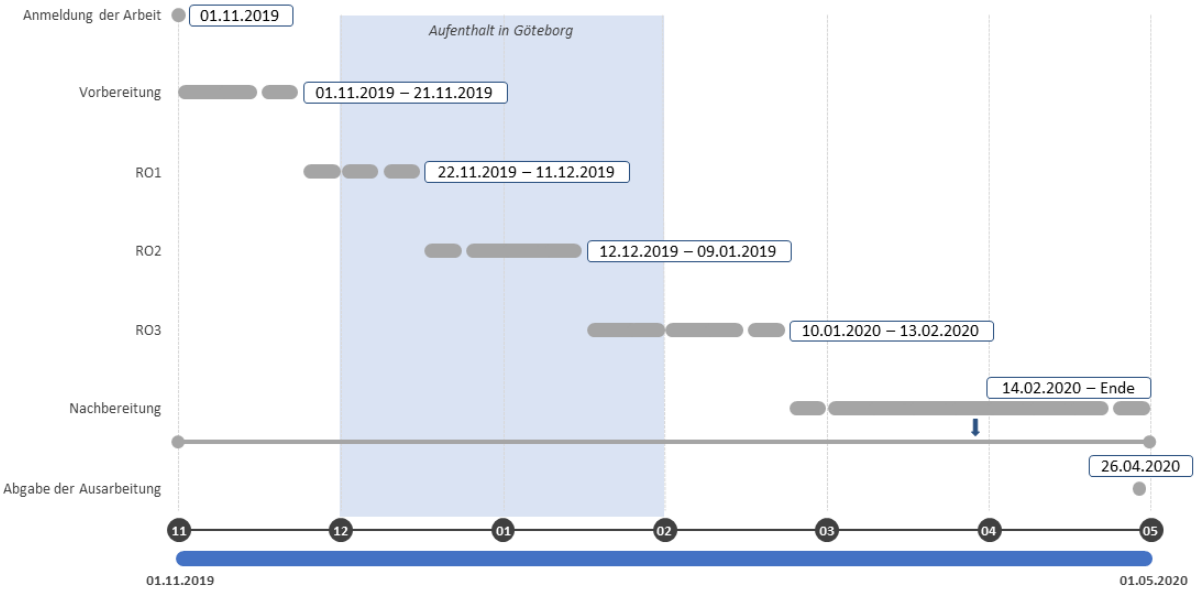


Abbildung 1.3: Geplanter zeitlicher Ablauf der Arbeit als Gantt-Chart

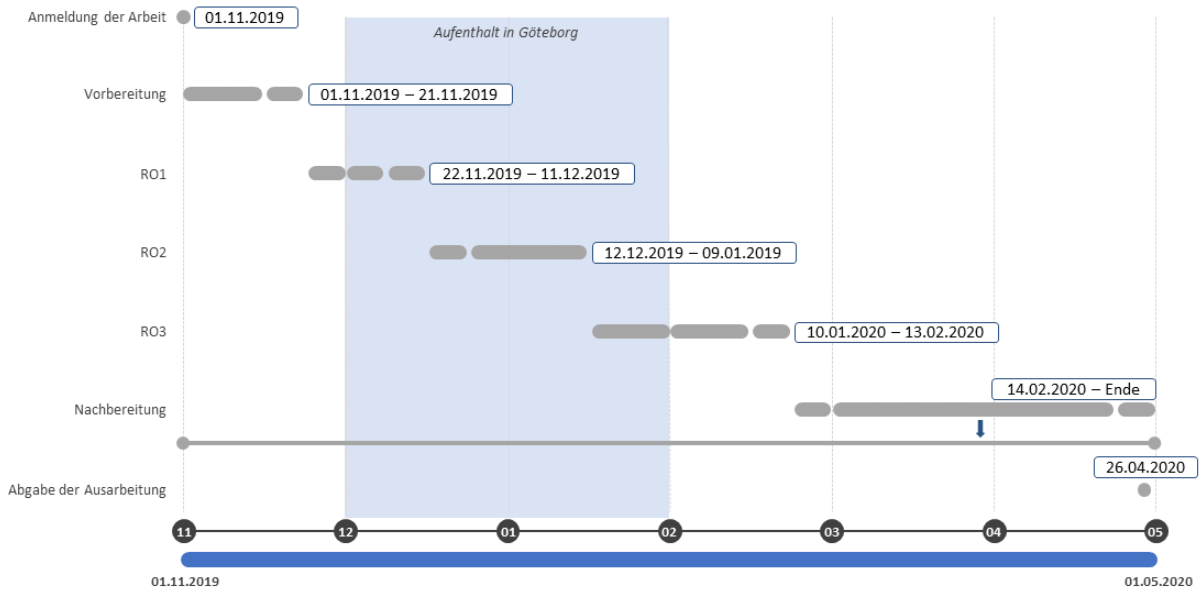


Abbildung 1.4: Tatsächlicher Ablauf der Arbeit als Gantt-Chart (TBD, einsetzen!)

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 und Tabellen zur verständlichen 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

Das zentrale Konzept hinter der featurebasierten Softwareentwicklung stellen sogenannte Software-Produktlinien dar. Wie bereits in der Einleitung erwähnt wurde, beschreiben Software-Produktlinien eine Menge von ähnlichen Softwareprodukten, welche eine gemeinsame Menge von Features sowie eine gemeinsame Codebasis besitzen und sich durch die Auswahl der verwendeten Features unterscheiden, sodass eine breite Variabilität innerhalb einer Produktlinie erreicht werden kann [4, 33].

Der zentrale Prozess der Generierung einer Software-Produktlinie ist in Abbildung 2.1 dargestellt. Aufgeteilt wird dieser Prozess 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 [4]. 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 Softwareprodukte basierend auf den bereits zuvor genannten Implementationstechniken für Features.

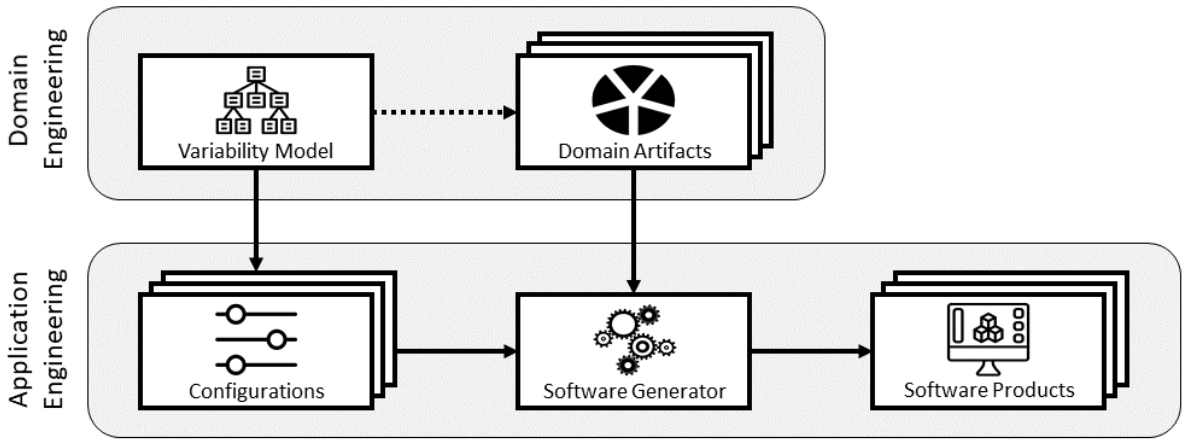


Abbildung 2.1: Generierung von Software-Produktlinien nach [33]

Die in dieser Arbeit betrachtete Implementierungstechnik von Features basiert auf Anweisungen beziehungsweise Bedingungsdirektiven des C-Präprozessors. Die für diese Arbeit relevanten Direktiven lauten `#IFDEF` und `#IFNDEF`. Einfache Beispieleinsätze für beide Direktiven sind in Listing 2.1 zu sehen. Sie befinden sich beide innerhalb einer Methode. Die Direktive `#IFDEF` leitet den Code der bedingten Gruppe des Features `feature1` ein, welche mit der Anweisung `#ENDIF` endet. Der mit `//Code feature1` angegebene Codeteil wird genau dann nur ausgeführt, wenn das Feature `feature1` im Rahmen der Konfiguration des Softwareproduktes definiert beziehungsweise aktiviert ist [32]. In diesem Fall wird die Bedingung der Direktive erfolgreich erfüllt [32]. Sie schlägt fehl, wenn das Feature nicht definiert beziehungsweise aktiviert ist [32]. Die Direktive `#IFNDEF` wird für Code verwendet, der ausgeführt werden soll, wenn ein Feature nicht definiert ist. Im Falle des Beispiels wird der in Zeile 7 angedeutete Code nur ausgeführt, wenn `feature2` nicht aktiviert wurde. Es besteht zudem die Möglichkeit, Features bzw. ihren Code zu verschachteln. Ein Beispiel dafür ist in Listing 2.2 angegeben. Es ist zu erkennen, dass sich der Code von `feature2` innerhalb der bedingten Gruppe von `feature1` befindet. Der in Zeile 5 angedeutete Code kann somit nur ausgeführt werden, wenn `feature1` aktiviert ist. Im Fall von Verschachtelung beendet ein `#ENDIF` immer das nächstgelegene `#IFDEF` oder `#IFNDEF` [32]. Es besteht zudem die Möglichkeit, Direktiven mittels „und“ (`&&`, `and`) oder „oder“ (`||`, `or`) zu erweiterten Bedingungen zu verknüpfen [32].

```
1  void example(){
2  #IFDEF feature1
3    // Code feature1
4  #ENDIF
5  // Code
6  #IFNDEF feature2
7    // Code
8  #ENDIF
9  // Code
10 }
```

Listing 2.1: Beispieleinsätze von `#IFDEF` und `#IFNDEF`

```
1  void example(){
2  #IFDEF feature1
3    // Code feature1
4    #IFDEF feature2
5        // Code feature1 & feature2
6    #ENDIF // Ende Code feature2
7    // Code feature1
8  #ENDIF // Ende Code feature1
9  // Code
10 }
```

Listing 2.2: Beispiel eines verschachtelten Einsatzes von `#IFDEF`

Die in Listing 2.1 und Listing 2.2 gezeigten Beispiele zeigen jeweils nur den Featurecode in einer Methode beziehungsweise in einer Datei. Fragmente des Featurecodes erstrecken sich jedoch nicht nur möglicherweise mehrfach über eine Datei sondern über mehrere Dateien - der Featurecode ist somit verstreut (englisch: code scattering), um eine Funktionalität des Features in der Gesamtheit der Software zu ermöglichen. Ein Defekt innerhalb eines Fragmentes des

Featurecodes kann allerdings dazu führen, dass die gesamte Funktionalität des Features beeinträchtigt oder unterbunden wird, da der Fehler übergreifend wirkt (englisch: cross-cutting). Grafisch dargestellt ist dies in Abbildung 2.2. Dort wirkt sich der Defekt eines Fragments von Feature 3 in einer Datei auf alle weiteren Dateien aus, sodass die Funktionalität des Features beeinträchtigt wird.

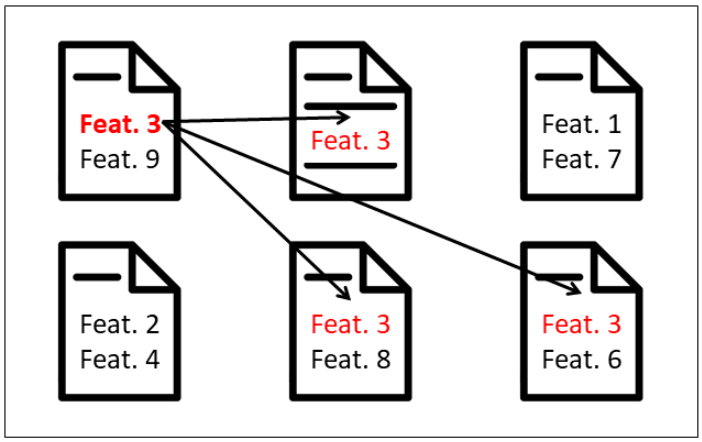


Abbildung 2.2: Auswirkungen eines Defekts in einem Fragment eines Features

2.2 Machine-Learning-Klassifikation

Das Themengebiet des Machine Learnings (ML) ist in zwei Teilgebiete unterteilt - das unüberwachte ML (englisch: unsupervised ML) und das überwachte ML (englisch: supervised ML). Beide Methoden verfolgen unterschiedliche Ziele. Im Rahmen des unüberwachten ML werden Prozesse durchgeführt, welche dazu dienen, die Struktur einer unbekannten Eingabemenge an Daten zu erlernen und anschließend zu repräsentieren [27]. Eine gängige Anwendung des unüberwachten ML ist das Clustering. Das überwachte ML beschreibt wiederum einen Prozess, welcher beabsichtigt, Vorhersagen über unbekannte Eingabedaten auf Basis der Erlernung einer Abbildungsfunktion zu treffen [27]. Die Attribute „unüberwacht“ und „überwacht“ erhalten die Methoden aufgrund ihrer Art des Lernens. In der Anwendung des unüberwachten ML werden die Eingabedaten erfasst, gegebenenfalls vorverarbeitet, um dann auf deren Basis ein Modell zu erlernen, welches die Darstellung beziehungsweise Repräsentation der Eingabedaten bestimmt [2]. Auf der anderen Seite, wird unter Anwendung des überwachten ML, ein Modell auf Basis eines sogenannten „gelabelten“ (beschrifteten) Datensatzes durch Merkmalsextraktion in Form von Attributen und Lernen auf der Grundlage der extrahierten Merkmale erstellt [2]. Der Datensatz, welcher zur Erlernung verwendet wird, wird im gängigen Sprachgebrauch des Machine Learning Datenset (englisch: dataset) genannt. Das aus der Erlernung resultierende Modell wird Klassifikator (englisch: classifier) genannt. Gängige Anwendungen des überwachten ML sind Regression und Klassifikation. In dieser Arbeit kommt die Klassifikation als Anwendung des überwachten ML zum Einsatz. Der grundlegende Prozess der Machine-Learning-Klassifikation ist in Abbildung 2.3 anhand eines Beispiels dargestellt.

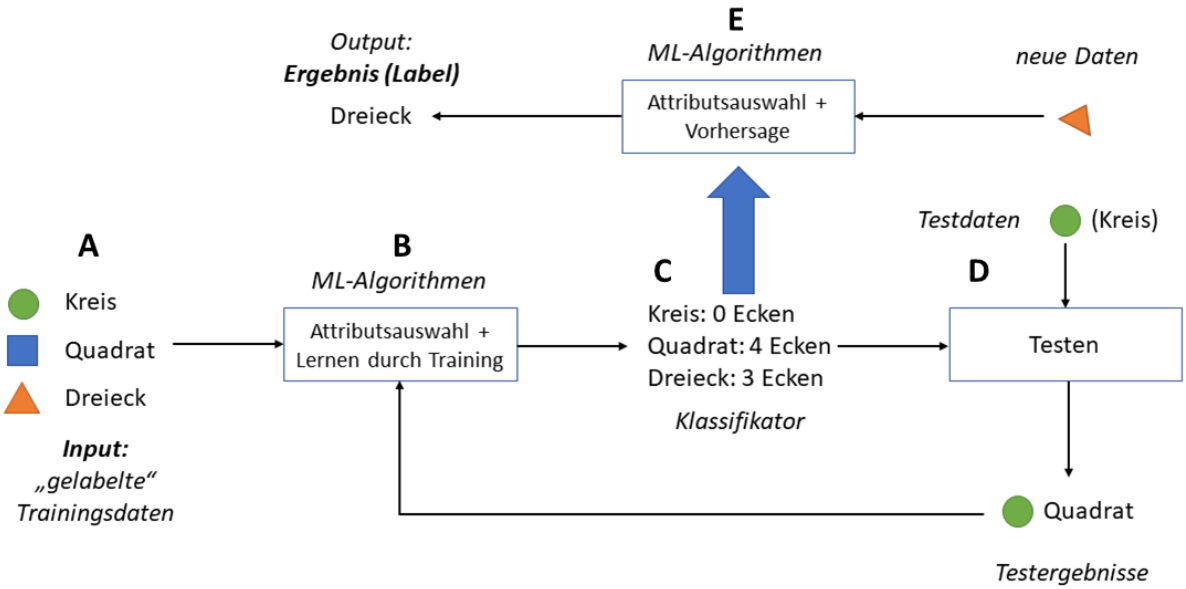


Abbildung 2.3: Allgemeiner Prozess des überwachten Machine Learnings dargestellt anhand eines Beispiels (vereinfacht)

Die Abbildung zeigt den Prozess des überwachten Machine Learnings anhand des Beispiels der Erlernung eines Klassifikators zur Erkennung beziehungsweise Vorhersage von geometrischen Formen. Die gezeigten Abläufe sind stark vereinfacht dargestellt. Der Prozess beginnt mit den „gelabelten“ Eingabedaten (A). Die Werte der Label (kategorisch oder numerisch) stellen dabei die zu vorhersagende Zielklasse dar. In diesem Falle bilden die Namen der geometrischen Formen die Label als kategorischen Wert. Die Rohdaten der Eingabemenge bestehen aus den geometrischen Formen selbst. Beide Datenmengen bilden das Datenset. Um nun einen Klassifikator anlernen zu können, müssen Merkmale der Eingangsdaten ausgewählt werden, anhand derer diese identifiziert werden können (B). Diese zu identifizierenden Charakteristika der Daten werden Attribute genannt. Diese Attribute können bereits vor dem Erlernen festgelegt werden oder automatisiert extrahiert werden. Im vorliegenden Fall wurde die Anzahl der Ecken der geometrischen Formen als Attribut zur Erlernung ausgewählt. Das Ergebnis ist der fertig trainierte Klassifikator, welcher das erlernte Wissen auf neue Daten abbilden kann (C). Ein Teil des Datensets wird in der Regel verwendet, um den Klassifikator nach dessen Erstellung zu testen (D). Das übliche Verhältnis (englisch: Split-Ratio) zwischen Trainings- und Testdaten beträgt 75:25. Diese Testdaten werden dem Klassifikator als Eingabemenge zur Klassifikation ohne Label zur Verfügung gestellt. Die Label sollten jedoch nicht verworfen werden, da sie als Vergleichsgrundlage für die Verhersageperformanz des Klassifikators dienen. Sie bilden die sogenannte „Ground Truth“(deutsch: Grundwahrheit). Dazu werden die vom Klassifikator vorhergesagten Label mit denen der Ground Truth verglichen. Sollte dieser Vergleich ergeben, dass die Label große Abweichungen zeigen, so kann der Klassifikator erneut erlernt werden mit anderen Attributen oder einer veränderten Split-Ratio. Erfüllt der Klassifikator die Anforderungen an die Performanz der Vorhersagen, so ist dieser bereit Vorhersagen auf Basis neuer Eingabedaten zu treffen (E). Dazu müssen von den neuen Daten die Attribute ermittelt werden. Auf Basis dieser trifft der Klassifikator die Vorhersage und liefert als Ausgabe das Label des Wertes der Zielklasse.

Der in Abbildung 2.3 dargestellte Klassifikator stellt einen multinomiellen oder multi-class Klassifikator dar, da er zu drei oder mehr Werten der Zielklasse zuordnen kann [27]. Die Regel bilden jedoch binäre Klassifikatoren, welche Vorhersagen zu zwei Werten der Zielklasse treffen. Dies trifft auch auf die Klassifikatoren dieser Arbeit zu.

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. [22] 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 zwei Abbildungen, welche den in der Arbeit von Queiroz et al. vorgestellten Prozess in zwei Teilen visualisieren. Der erste Teil ist in Abbildung 2.4 dargestellt.

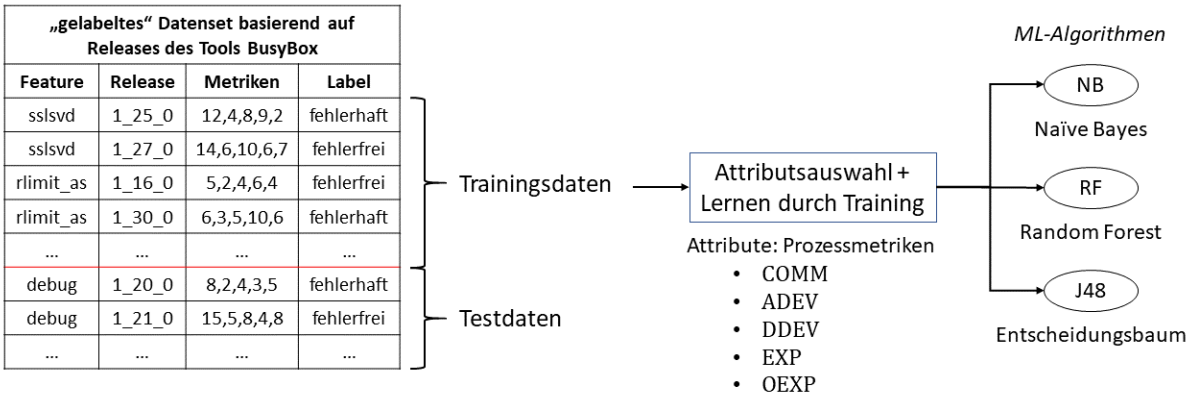


Abbildung 2.4: Teil 1: Featurebasierter Prozess des überwachten Machine Learnings nach [22]

Die Datenbasis des Datensets bilden historische 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 [28, 36]. 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/`

Die Trainingsdaten werden dann den Klassifikatoren zur Erlernung zur Verfügung gestellt. Als Attribute dienen fünf Prozessmetriken mit spezifischer Betrachtung von Software-Features. Einen Überblick über die Beschreibungen dieser gibt Tabelle 2.1.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.

Tabelle 2.1: Übersicht der von [22] 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.	

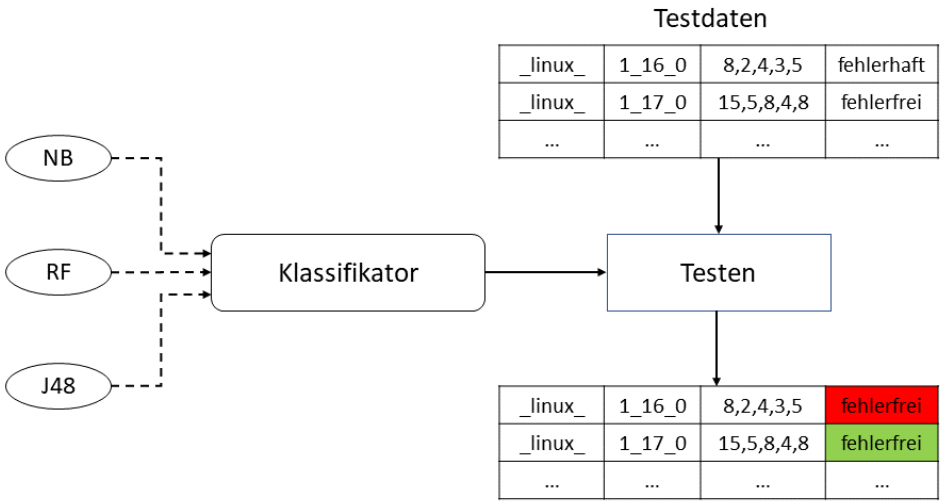


Abbildung 2.5: Teil 2: Featurebasierter Prozess des überwachten Machine Learnings nach [22]

Wie in Abbildung 2.5 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.

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 im Rahmen der Anlernung der Klassifikatoren dienen. Eine Gliederung der Kapitel kann Abbildung 3.1 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 Softwareprojekten erstellt. Die Auswahl der Softwareprojekte erfolge anhand von vorheriger Verwendung in wissenschaftlicher Literatur [12, 14, 22]. Die für diese Arbeit verwendeten Softwareprojekte sind samt ihres Einsatzzweckes und ihrer Datenquellen in Tabelle 3.1 aufgeführt.

Tabelle 3.1: Übersicht der verwendeten Softwareprojekte[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			

Zum Erhalt der Commit-Daten der Softwareprojekte wurde die Python-Library PyDriller[2] verwendet [30]. Diese ermöglicht eine einfache Datenextraktion aus Git-Repositories zum Erhalt von Commits, Commit-Nachrichten, Entwicklern, Diffs und mehr (im weiteren Verlauf „Metadaten“ genannt). Ein beispielhafter Sourcecode-Ausschnitt zur Konsolenausgabe von Metadaten eines Commits (Autor, Name der veränderten 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 eigens erstellten Python-Scripte zum Erhalt der Commit-Metadaten dienten jeweils die URLs zu den Git-Repositories der Softwareprojekte. Weiterhin wurden die Metadaten in Commits je Release aufgeteilt. Ermöglicht wurde dies durch die Angabe von Release-Tags im PyDriller-Code, basierend auf der Tag-Struktur von Git-Repositories. 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 (Changeset) der Veränderung

Die auf diese Weise erhaltenen Daten wurden nach dem Abruf in einer MySQL-Datenbank gespeichert. Für jedes Softwareprojekt wurde eine eigene Tabelle erstellt, in welcher neben der oben stehenden Metadaten zudem der Name des betreffenden Softwareprojekts und die den Commits zugehörigen Release-Nummern gespeichert wurden. Jede veränderte Datei eines Commits erhält eine Zeile der Datenbank-Tabellen. In Tabelle 3.2 kann eingesehen werden,

[1] Links zu den Websites der Softwareprojekte und deren Repositories können im Anhang eingesehen werden.
[2] https://github.com/ishepard/pydriller
[3] Diese Information fand in der weiteren Erstellung des Datensets keine Verwendung.

wie viele Releases je Softwareprojekt abgerufen wurden und wie viele Commits daraus resultieren. Eine Übersicht der Anzahl der korrektiven und fehlereinführenden Commits sowie der Anzahl der identifizierten Features je Softwareprojekt ist in Tabelle 3.3 aufgeführt.

Tabelle 3.2: Übersicht der Anzahl der Releases und Commits je Softwareprojekt

	#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 Anzahl der korrektiven und fehlereinführenden Commits sowie zur Anzahl der identifizierten Features je Softwareprojekt

	#korrektiv	#fehlereinführend	#Features
Blender	8333	1418	1400
Busybox	1408	142	628
Emacs	6959	685	718
GIMP	1703	272	204
Gnumeric	1591	136	637
gnuplot	880	1323	558
Irssi	77	1	9
libxml2	409	37	200
lighttpd	1202	555	230
MPSolve	158	69	54
Parrot	3437	824	397
Vim	1033	2571	1158
xfig	0	0	137

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.

RQ1A: WELCHE DATEN KOMMEN FÜR DIE ERSTELLUNG DES DATENSETS IN FRAGE?

Es kommen die Daten von 13 featurebasierten Softwareprojekten zur Verwendung. Mithilfe der Python-Library PyDriller wurden die Metadaten der Commits, aufgeteilt nach Commits pro Release, aus den Git-Repositories extrahiert und in MySQL-Datenbanken gespeichert. Ausgewählt wurden die Softwareprojekte aufgrund einer vorherigen Verwendung in der wissenschaftlichen Literatur [12, 14, 22].

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 ver-

änderten Datei. Dazu wurden mithilfe eines Python-Scripts die Präprozessor-Anweisungen `#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 je Datei identifizierten Features in einer zusätzlichen Spalte in den jeweiligen MySQL-Tabellen der Metadaten der Softwareprojekte. Konnte kein Feature identifiziert werden, wird entsprechend „`none`“ gespeichert.

Dieser Weg der Identifizierung birgt einige Hindernisse. Diese können, neben dem Normalfall, in Abbildung 3.2 gesehen werden. In einigen C-Programmierparadigmen ist es üblich, Header-Dateien mittels Präprozessor-Direktiven im Sourcecode einzubinden, sodass sie wie Features scheinen (siehe erster unerwünschter Fall in Abbildung 3.2). Diese "Header-Features", wie sie im weiteren Verlauf genannt werden, sollten jedoch ignoriert werden, da sie im gesamten 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 3.2). Solche falschen Features wurden in einer manuellen Sichtung der identifizierten Features entfernt und durch „`none`“ ersetzt.

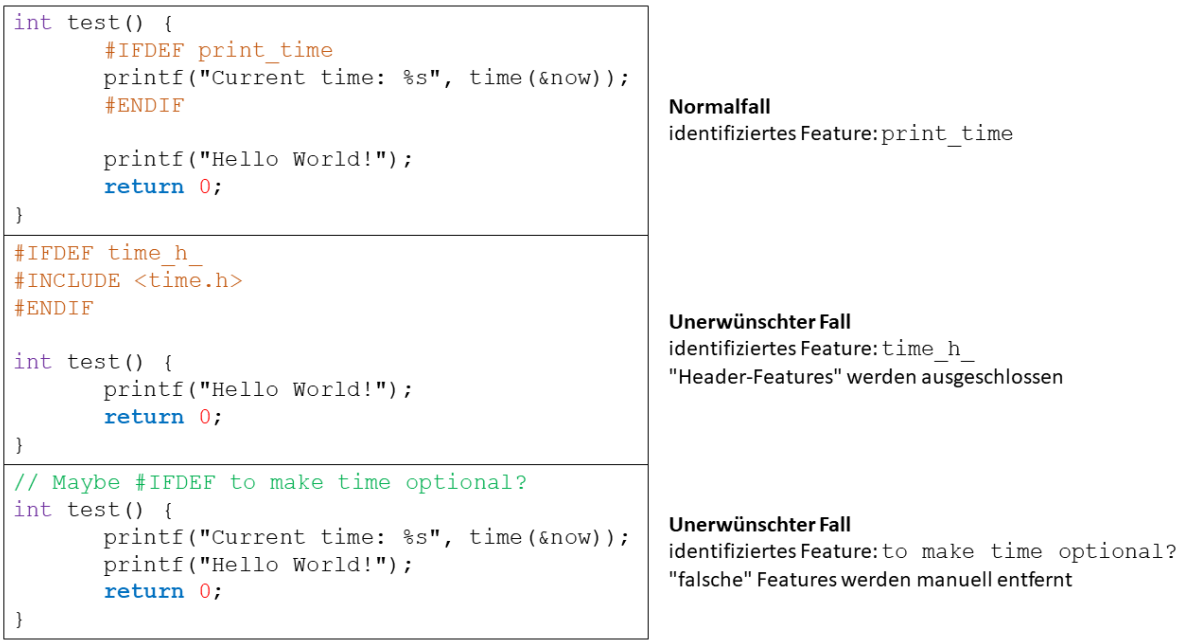


Abbildung 3.2: Normalfall und unerwünschte Fälle bei der Identifizierung der Features

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 Schlagwörtern [36]. Bei den Schlagwörtern handelt es sich um „bug“,„bugs“, „bugfix“, „error“, „fail“, „fix“, „fixed“ und „fixes“. Durchgeführt wurde die Analyse mittels eines Python-Skripts unter Zuhilfenahme von einfachen Formen des Natural Language Processings (NLP). Die Ergebnisse wurden in einer weiteren Spalte der MySQL-Tabellen (true = korrektiv, false = nicht korrektiv) gespeichert.

Der Suche nach korrektiven Commits folgt eine Analyse nach fehlereinführenden Commits. Dazu wurde eine PyDriller-Implementierung des SZZ-Algorithmus nach Sliwerski, Zimmermann und Zeller verwendet [28]. Dieser ursprünglich für CVS-Versionskontrollsysteme entwickelte Algorithmus erlaubt es, in zwei Phasen fehlereinführende Commits in lokal gespeicherten Software-Repositories zu finden [5]. 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 [5]. 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 3.3 dargestellt. Die Erläuterungen der mit Buchstaben versehenen Schritte erfolgt im Anschluss. Die für diese Arbeit verwendete PyDriller-Implementierung des Algorithmus folgt dem gezeigten Ablauf.

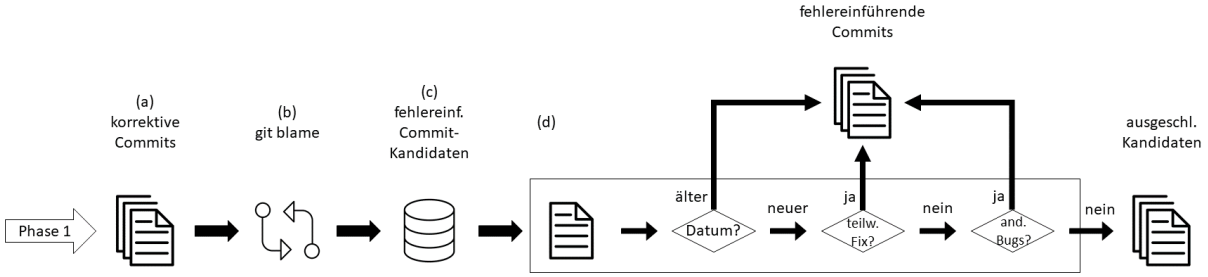


Abbildung 3.3: Ablauf der zweiten Phase des SZZ-Algorithmus (übersetzt, [5])

Die zweite Phase des SZZ-Algorithmus, die als Input eine Liste der Commit-Hashes der zuvor 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 [5]. 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 [5]. 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) [5]. Die Ausgabe ist eine Liste von Commit-Hashes von fehlereinführenden Commits für jede Datei eines korrektiven Commits. Diese neuen Informationen werden in einer zusätzlichen Spalte in den MySQL-Tabellen für jeden Eintrag gespeichert (true = fehlereinführend, false = nicht fehlereinführend).

Eine Übersicht des Schemas der nun vollständigen initialen MySQL-Tabellen (im folgenden Haupttabellen genannt) ist in Tabelle 3.4 aufgeführt. Wie bereits zuvor erwähnt, umfasst diese Tabelle 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.

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 (*ohne weitere Verwendung*)
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		

Ein reales Beispiel aus der Haupttabelle des Softwareprojektes Vim, welches die Diffs eines korrektiven (A) und eines fehlereinführenden (B) Commits zu einem Feature `FEAT_TEXT_PROP` zeigt, ist in Abbildung 3.4 dargestellt. Folgende Informationen können zu der betreffenden Datei (das Feature befindet sich sowohl im korrektiven als auch im fehlereinführenden Fall in der selben Datei) des korrektiven Commits[4] (A) aus den Daten der Haupttabellen entnommen werden:

- Datei: `screen.c`
- Commit-Hash: `1748c7f77ea864c669b7e5cfb2be0c34ce45e36e`
- Commit-Nachrich: `patch 8.1.1495: memory access` **`error`**`. problem: memory access` **`error`**`. solution: use the correct size for clearing the popup mask.`

Der Ausschnitt des Diffs zeigt zudem, dass der Methodenaufruf `vim_memset` mit abweichenden Argumenten ersetzt wurde. Laut zugehöriger Commit-Nachricht führte der ursprüngliche Methodenaufruf zu einem „Memory Access Error“. Dieser Commit wurde als korrektiv identifiziert, da die Commit-Nachricht das Schlagwort „error“enthält. Der entsprechende Eintrag in der Haupttabelle von Vim erhält somit in der Spalte „corrective“ den Wert `true`. Mithilfe des SZZ-Algorithmus konnte, unter Angabe des Commit-Hashes des korrektiven Commits, der fehlereinführende Commit[5] (B) der betroffenen Datei ermittelt werden. In dessen Ausschnitt des Diffs ist zu erkennen, dass mit diesem Commit das Feature `FEAT_TEXT_PROP` in der Datei mit dem fehlerhaften Methodenaufruf eingepflegt wurde. Folglich bekommt dieser in der Haupttabelle in der Spalte „bug_introducing“ den Wert `true` zugewiesen.

[4]Link zum Commit im Git-Repository von Vim: `https://github.com/vim/vim/commit/1748c7f77ea864c669b7e5cfb2be0c34ce45e36e`

[5]Link zum Commit im Git-Repository von Vim: `https://github.com/vim/vim/commit/33796b39b9f00b42ca57fa00dbbb52316d9d38ff`

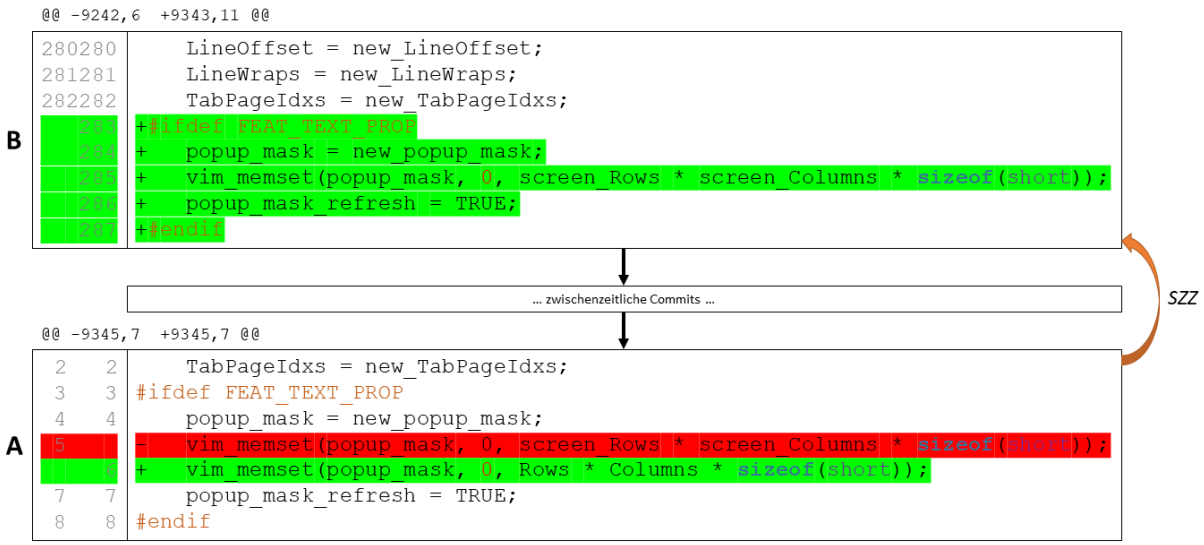


Abbildung 3.4: Reales Beispiel eines Fehlers mit korrektivem (A) und fehlereinführendem (B) Commit

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

3.3 Metriken

Wie bereits in Abschnitt 2.2 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 Rohdaten der Haupttabellen berechnet werden [23]. Codemetriken werden genutzt um Eigenschaften von Sourcecode, wie zum Beispiel „Größe“ oder Komplexität, zu messen [23]. Prozessmetriken dienen hingegen zur Messung von Eigenschaften, die anhand von Metadaten aus Software-Repositories erörtert werden können [23]. Beispiele dafür sind die Anzahl der Veränderungen einer bestimmten Datei oder die Anzahl der aktiven Entwickler an einem Projekt. Für diese Arbeit wurden elf Metriken errechnet, aufgeteilt in sieben Prozess- und vier Codemetriken. Fünf der Prozessmetriken wurden aus wissenschaftlichen Arbeiten [23, 22] 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 3.5 wird die Abgrenzung zwischen feature- und dateibasiertem Datenset 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 + I) bestehen aus den jeweils berechneten feature- (F) und dateibezogenen (H) Metriken sowie den

Labeln der Zielklasse. Berechnet werden die Werte der Metriken für die Daten jedes Softwareprojektes. Die daraus resultierenden Tabellen enthalten als Spalten die Werte der 13 Metriken sowie das Label (Zielklasse) „defekt“ oder „fehlerfrei“ und als Zeilen die Features bzw. Dateien aggregiert nach Release. Diese Tabellen werden jeweils zu einer gemeinsamen Tabelle konkateniert. Sollte ein Feature oder eine Datei in einem Release mehrfach bearbeitet worden sein (d.h. es wird in mehreren Commits bearbeitet), so wird der Durchschnittswert der jeweiligen Metriken berechnet und gespeichert. Das Verfahren zur Bestimmung des Labels erfolgt anhand des folgenden Musters:

- fehlereinführend + korrektiv = defekt
- fehlereinführend + nicht korrektiv = defekt
- nicht fehlereinführend + korrektiv = fehlerfrei
- nicht fehlereinführend + nicht korrektiv = fehlerfrei

Für den Fall, dass ein Feature oder eine Datei mehrfach innerhalb eines Releases bearbeitet sein sollte, erfolgt die Bestimmung des Labels anhand der folgenden Regeln:

- im Fall von Features wird überprüft, ob innerhalb des Releases das Feature mindestens ein Mal als „defekt“ markiert wurde. Ist dies der Fall, so wird angenommen, dass das Feature in dem betreffenden Release defekt ist. Tritt dieser Fall nicht ein, so gilt das Feature als fehlerfrei.
- im Fall von Dateien wird der jeweils letzte Commit der Dateien im betreffenden Release überprüft. Ist die Datei dort als „fehlerfrei“ markiert, so wird angenommen, dass sie fehlerfrei ist. Ist sie als „defekt“ markiert, so wird angenommen, dass sie in diesem Release defekt ist.

Eine Übersicht der berechneten Metriken samt Beschreibung befindet sich in Tabelle 3.5. Das Datenset besteht aus der konkatenierten Tabelle mit den Metriken der elf Softwareprojekte und kann zur Erlernung der Klassifikatoren genutzt werden. Dieser Vorgang wird im folgenden Kapitel erläutert.

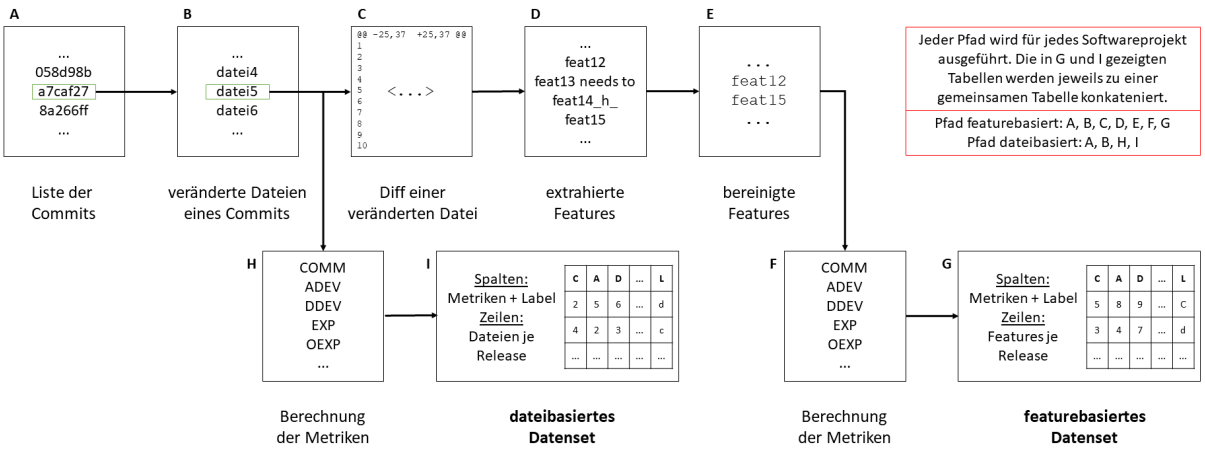


Abbildung 3.5: Visualisierung des Aufbaus und der Unterscheidung der Datensets

Tabelle 3.5: Übersicht der berechneten Metriken

	Metrik	Beschreibung	Quelle
Prozessmetriken	Anzahl der Commits (COMM)	Anzahl der Commits, die dem Feature in einem Release zugeordnet sind.	[23, 22]
	Anzahl der aktiven Entwickler (ADEV)	Anzahl der Entwickler, die innerhalb eines Releases das Feature bearbeitet (geändert, gelöscht oder hinzugefügt) haben.	[23, 22]
	eindeutige Entwickleranzahl (DDEV)	kumulierte Anzahl der Entwickler, die innerhalb eines Releases das Feature bearbeitet (geändert, gelöscht oder hinzugefügt) haben.	[23, 22]
	Erfahrung aller Entwickler (EXP)	geometrisches Mittel der „Erfahrung“ aller Entwickler, die innerhalb eines Releases das Feature 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.	[23, 22]
	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.	[23, 22]
	Grad der Änderungen (MODD)	Anzahl der Bearbeitungen (Änderung, Entfernung, Erweiterung) des Features / der Datei innerhalb eines Releases.	*
	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 sind, desto fehleranfälliger scheinen diese zu sein.	*
Codemetriken	Anzahl der Codezeilen (NLOC)	Durchschnittliche Anzahl der Codezeilen der dem Feature zugeordneten Dateien / der Datei innerhalb eines Releases.	*
	Zyklomatische Komplexität (CYCO)	Durchschnittliche zyklomatische Komplexität der dem Feature zugeordneten Dateien / der Datei innerhalb eines Releases.	*
	Anzahl der hinzugefügten Zeilen (ADDL)	Durchschnittliche Anzahl der hinzugefügten Codezeilen zu den dem Feature zugeordneten Dateien / zur Datei innerhalb eines Releases.	*
	Anzahl der entfernten Zeilen (REML)	Durchschnittliche Anzahl der gelöschten Codezeilen von den dem Feature zugeordneten Dateien /~ von der Datei innerhalb eines Releases.	*
** Diese Werte wurden auf Basis der mit PyDriller erhaltenen Metadaten berechnet. Die Berechnung der Metriken auf Feature-Level erfolgte auf Basis der Metadaten der ihnen zugrunde liegenden Dateien.*			

RQ1B: WIE WEIT MÜSSEN DIE DATEN VORVERARBEITET WERDEN, UM SIE FÜR DAS TRAINING NUTZBAR ZU MACHEN?

Eine umfassende Vorverarbeitung der Daten aus den Repositories ist nicht nötig. Die Verarbeitungsschritte bestanden aus: Extraktion der Features inklusive Bereinigung, Identifizierung von korrektiven Commits mittels Analyse der Commit-Nachrichten, Analyse nach fehlereinführenden Commits unter Zuhilfenahme des SZZ-Algorithmus sowie Berechnung von elf Metriken, welche als Attribute für die Erlernung der Klassifikatoren dienen.

Kapitel 4

Training und Test der Machine-Learning-Klassifikatoren

Ausblick: Dieses Kapitel gibt einen detaillierten Einblick in das Training und den Testprozess der Machine-Learning-Klassifikatoren. Dazu werden zunächst die Auswahl der verwendeten Werkzeuge und der Klassifikationsalgorithmen erläutert. Anschließend findet eine Analyse der Trainings- und Testprozesse der Klassifikatoren statt. Dies umfasst außerdem die Auflistung der finalen Konfigurationen der jeweiligen Klassifikatoren.

4.1 Auswahl der Werkzeuge und der Klassifikationsalgorithmen

Durch die Wahl der Programmiersprache Python war die Entscheidung zur Auswahl eines bestimmten Machine-Learning-Werkzeugs bereits absehbar. Zur Anwendung kommt die Python-Library scikit-learn[1], die im Jahr 2007 von Pedregosa et. al entwickelt wurde [20]. Das Werkzeug bietet eine große Auswahl an Machine-Learning-Algorithmen für überwachtes und unüberwachtes Lernen und ermöglicht darüber hinaus eine einfache Verwendung sowie eine einfache Einbindung weiterer Python-Libraries, wie beispielsweise die Matplotlib zur Erstellung von mathematischen Darstellungen [20].

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 [11, 22, 25]) ebenfalls als geeignet für die zugrundeliegende Aufgabe. Der WEKA-Workbench (WEKA als Akronym für **W**aikato **E**nvironment for **K**nowledge **A**nalysis) 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 [9]. Es existieren zudem Schnittstellen für die Programmiersprache Java [9].

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 je Werkzeug befindet sich in Tabelle 4.1. 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

Decision Trees

Decision Trees (deutsch: Entscheidungsbäume) zählen zu den meistverwendeten Klassifikatoren im Bereich des supervised Machine Learnings. Studien belegen, dass sie hinsichtlich der Verwendung im Kontext der Fehlererkennung die häufigste Anwendung finden [29]. Decision Trees sind gerichtete und verwurzelte Bäume, die als rekursive Partition der Eingabemenge des Datensets aufgebaut werden [26]. Den Ursprung des Baumes bildet die Wurzel, welche keine eingehenden Kanten besitzt - alle weiteren Knoten besitzen jedoch eine eingehende Kante [26]. Diese Knoten teilen wiederum die Eingabemenge anhand einer vorgegebenen Funktion in zwei oder mehr Unterräume der Menge auf [26]. Meist geschieht dies anhand eines Attributs, sodass die Eingabemenge anhand der Werte des einzelnen Attributs geteilt wird [26]. 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 [26]. Der grundlegende Aufbau eines Decision Trees ist in Abbildung 4.1 dargestellt.

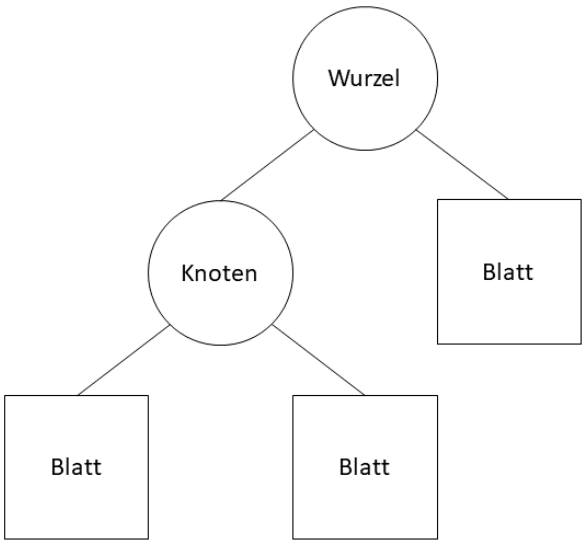


Abbildung 4.1: Grundsätzlicher Aufbau eines Decision Trees

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 Decision 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 [35]. Das erste Konzept basiert auf der Abstandsmessung zwischen den Werten der zu klassifizierenden Datenmenge und den Werten der Attribute des Datensets [35]. Die Abstandmessung erfolgt in der Regel durch die Berechnung der Euklidischen Distanz $D(p, q)$:

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

Die Anzahl der Attribute wird durch den Parameter n wiedergegeben, p und q repräsentieren jeweils die Werte der zu klassifizierenden Datenmenge und die Werte der Attribute des Datensets. Das zweite Konzept bildet der Parameter k, der angibt, wie viele nächste Nachbarn zum Vergleich der zuvor berechneten Abstände in Betracht gezogen werden [35]. Bei einem $k > 1$ wird diejenige Zielklasse gewählt, deren Auftreten innerhalb der nächsten Nachbarn überwiegt.

Künstliche neuronale Netze

Künstliche neuronale Netze (englisch: Artificial Neural Networks) verwenden nicht-lineare Funktionen zur schrittweisen Erzeugung von Beziehungen zwischen der Eingabemenge und den Zielklassen durch einen Lernprozess [15]. Sie sind angelehnt an die Funktionsweise von biologischen Nervensystemen und bestehen aus einer Vielzahl von einander verbundenen Berechnungsknoten, den Neuronen [19]. Der grundsätzliche Aufbau eines künstlichen neuronalen Netzes kann in Abbildung 4.2 eingesehen werden. Der Lernprozess besteht aus zwei Phasen - einer Trainingphase und einer Recall-Phase [15]. In der Trainingsphase werden die Eingabedaten, meist als multidimensionaler Vektor, in den Input-Layer geladen und anschließend an die Hidden-Layer verteilt [19]. In den Hidden-Layers werden dann Entscheidungen anhand der Beziehungen zwischen den Eingabedaten und Zielklassen sowie die den Verbindungen zuvor zugewiesenen Gewichtsfaktoren getroffen [15, 19]. Im Rahmen der Recall-Pahse werden die wird die Vorhersage basierend auf der zu klassifizierenden Datenmenge anhand der zuvor getroffenen Entscheidungen der Hidden-Layers getroffen und an die jeweiligen Output-Layer, welche die Werte der Zielklasse repräsentieren, weitergeleitet[15].

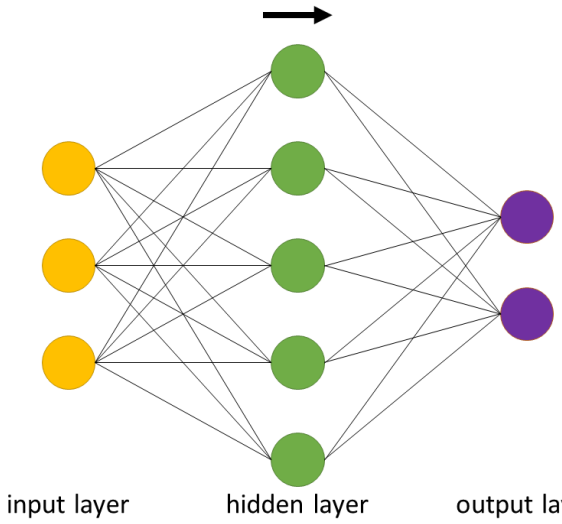


Abbildung 4.2: Grundsätzlicher Aufbau eines KNN mit drei Input-Layer-Neuronen, fünf Hidden-Layer-Neuronen und zwei Output-Layer-Neuronen

Naïve Bayes

Naïve-Bayes-Klassifikatoren zählen zu den linearen Klassifikatoren und basieren auf dem Satz von Bayes. Die Bezeichnung „naiv“ erhält der Klassifikator durch die Annahme, dass die Attribute der Eingabemenge unabhängig voneinander sind [24]. Diese Annahme wird zwar in der realen Verwendung des Klassifikators häufig verletzt, dennoch erzielt er in der Regel eine hohe Performanz [24]. Der Klassifikator gilt als effizient, robust, schnell und einfach implementierbar [24]. Die zur Durchführung einer Klassifikation mittels Naïve Bayes benötigte Formel nach Thomas Bayes ist in Abbildung 4.3 samt Erläuterung der einzelnen Faktoren aufgeführt.

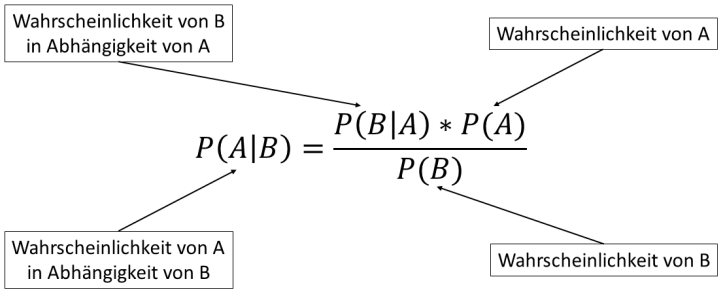


Abbildung 4.3: Satz von Bayes als Grundlage des Naïve-Bayes-Klassifikators

Es existiert zudem eine Mehrzahl an Varianten des Naïve-Bayes-Klassifikators, die verschiedene Annahmen über die Verteilung der Attribute der Eingabemenge machen. Beispiele dafür sind der Gaußsche-Naïve-Bayes (normalverteilte Attribute), der multinomiale Naïve-Bayes (multinomiale Verteilung der Attribute) sowie der Bernoulli-Naïve-Bayes (unabhängige binäre Attribute).

Logistische Regression

Logistische-Regressions-Klassifikatoren (englisch: Logistic Regression) basieren auf dem mathematischen Konzept des Logits, welcher den natürlichen Logarithmus eines Chancenverhältnisses beschreibt [21]. Seine Formel lautet:

$$logit(Y) = ln(\frac{\pi}{1-\pi})$$

Y beschreibt dabei die zu klassifizierende Datenmenge, wohingegen π die Verhältnisse der Wahrscheinlichkeiten der Werte der Attribute der Eingabemenge bezeichnet. Am besten geeignet ist dieser Klassifikator für eine Kombination aus kategorischen oder kontinuierlichen Eingabedaten und kategorischen Zielklassen [21]. Der von scikit-learn zur Verfügung stehende Ridge Classifier basiert ebenfalls auf dem Konzept der logistischen Regression.

Stochastic Gradient Descent

Ein Stochastic-Gradient-Descent-Klassifikator basiert auf dem Gradientenverfahren (englisch: Gradient Descent), welches das Ziel hat, zu einem gegebenen x das minimale y zu finden [31]. Im Falle der Klassifikation mittels Machine Learning bezeichnet x dabei die zu klassifizierende Eingabemenge und y das erwartete Ergebnis der Klassifikation. Minimiert werden sollen dabei die „Kosten“, die sich aus der Ermittlung der Ergebnisse ergeben.

Support Vector Machines

Support Vector Machines verfolgen das Ziel, eine sogenannte „Hyperplane“ in einem n-dimensionalen Raum (n = Anzahl der Attribute der Eingabemenge) zu finden, welche die Datenpunkte der Eingabemenge eindeutig klassifizieren kann [10]. Die Hyperplane beschreibt eine Trennlinie beziehungsweise Trennfläche, mit deren Hilfe die Daten der zu klassifizierenden Menge den Zielklassen zuordnen lassen [16]. Dabei gilt es, dass die Trennflächen, welche die Eingangsmenge anhand der Attribute in verschiedene Trennungsebenen unterteilen, einen möglichst großen Abstand ohne Datenpunkte voneinander haben [16]. Dies funktioniert sowohl für linear als auch nicht-lineare trennbare Mengen.

Alle zuvor vorgestellten Klassifikationsalgorithmen sind bereits in den Werkzeugen scikit-learn und WEKA integriert. Sie erhalten als Eingabe das finale Datenset, dessen Erstellung im vorherigen Kapitel erläutert wurde. Die 13 berechneten Metriken bilden dabei die Attribute, wohingegen die Zielklasse durch die Label „defekt“ und „fehlerfrei“ abgebildet wird.

RQ2: WELCHE MACHINE-LEARNING-KLASSIFIKATOREN KOMMEN FÜR DIE GEGEBENE AUFGABE IN FRAGE?

Es werden neun verschiedene Klassifikationsalgorithmen zur Anwendung kommen. Sieben Algorithmen werden sowohl mit scikit-learn als auch mit WEKA verwendet (DT / J48, KNN, NB, NN, RF, SGD, SVM). Jeweils ein Algorithmus ist werkzeugspezifisch (scikit-learn: RC, WEKA: LR), jedoch unterliegen beide Algorithmen dem Konzept der Regression. Das Hauptkriterium für die Auswahl sämtlicher Algorithmen war die vorherige Verwendung im Rahmen der wissenschaftlichen Literatur [29].

4.2 Analyse der Trainings- und Testprozesse

Im weiteren Verlauf dieses Abschnitts und im Rahmen der Evaluation im folgenden Kapitel, werden die Namen der Klassifikatoren auf Abbildungen und in Tabellen abgekürzt. Die Abkürzungen können Tabelle 4.2 entnommen werden.

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		

Die Analyse des Trainingsprozesses zeigte, 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 (Treffergenauigkeit des Klassifikators) 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 [8]. Der Algorithmus, dessen Akronym für **S**ynthetic **M**inority **O**versampling **Te**chnique steht, führt ein Oversampling der unterrepräsentierten Klasse durch [8]. 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 [8]. 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 4.4 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 und eine unrealistische Situation darstellt. Das Diagramm, welches die Testergebnisse nach Anwendung des SMOTE-Algorithmus darstellt, weist hingegen wesentlich glaubwürdigere Accuracies auf.

Der Testprozess diente zur Erarbeitung der korrekten Konfiguration der Klassifikatoren. Dazu zählen das Festlegen von möglichen Parametern, die die Klassifikationsalgorithmen beeinflussen können, das Festlegen des optimalen Verhältnisses zwischen Training- und Testdatenset (Split-Ratio) sowie die Auswahl der effektivsten Kombination der gegebenen elf Attribute. In Abbildung 4.5 ist zunächst dargestellt, welche Accuracies die jeweiligen Klassifikatoren pro Werkzeug und Datenset ohne eine angewendete Konfiguration erreichen. Gemessen wurden jeweils die Accuracies der Klassifikatoren für die Split-Ratios 85:15 (Training : Test), 80:20, 75:25, 70:30 und 65:35. Außerdem wurde für das Werkzeug WEKA die alternative Option zur Durchführung einer „10-fold-cross-validation“ berücksichtigt. Bei dieser Variante wird das Datenset in zehn gleich große Datensets aufgeteilt, welche dann jeweils mit einer Split-Ratio von 90:10 zur Anlernung eines Klassifikators dienen [34]. Die Accuracy dieses Vorgangs wird anschließend als Durchschnitt der Accuracies der 10 Klassifikatoren gewertet [34]. Die erwähnte Abbildung zeigt pro Klassifikator die höchste Accuracy die mithilfe der fünf Split-Ratios oder mithilfe der 10-fold-cross-validation gemessen werden konnte.

Zur Auswahl der effektivsten Kombination der Attribute je Klassifikator wurde auf die von scikit-learn und WEKA bereitgestellten Werkzeuge zurückgegriffen. Sowohl scikit-learn als auch WEKA konnten nicht für jeden Klassifikator der beiden Datensets Kombinationen ausgeben. Dies bedingt entweder die Funktionsweise des Klassifikationsalgorithmus, die fehlende Unterstützung der Werkzeuge für einen bestimmten Klassifikator oder eine endlose Durchführung der Ermittlung der Kombination. Eine Übersicht der Konfiguration je Klassifikator des featurebasierten Datensets ist in Tabelle 4.3 dargestellt. Die Übersicht für das dateibasierte Datenset befindet sich in Tabelle 4.4.

Mithilfe der Konfigurationen konnten die finalen Klassifikatoren erzeugt werden, welche als Grundlage für die Evaluation und den gegenseitigen Vergleich innerhalb und zwischen dem featurebasierten und dem dateibasierten Datenset dienen. Die Evaluation erfolgt im folgenden Kapitel.

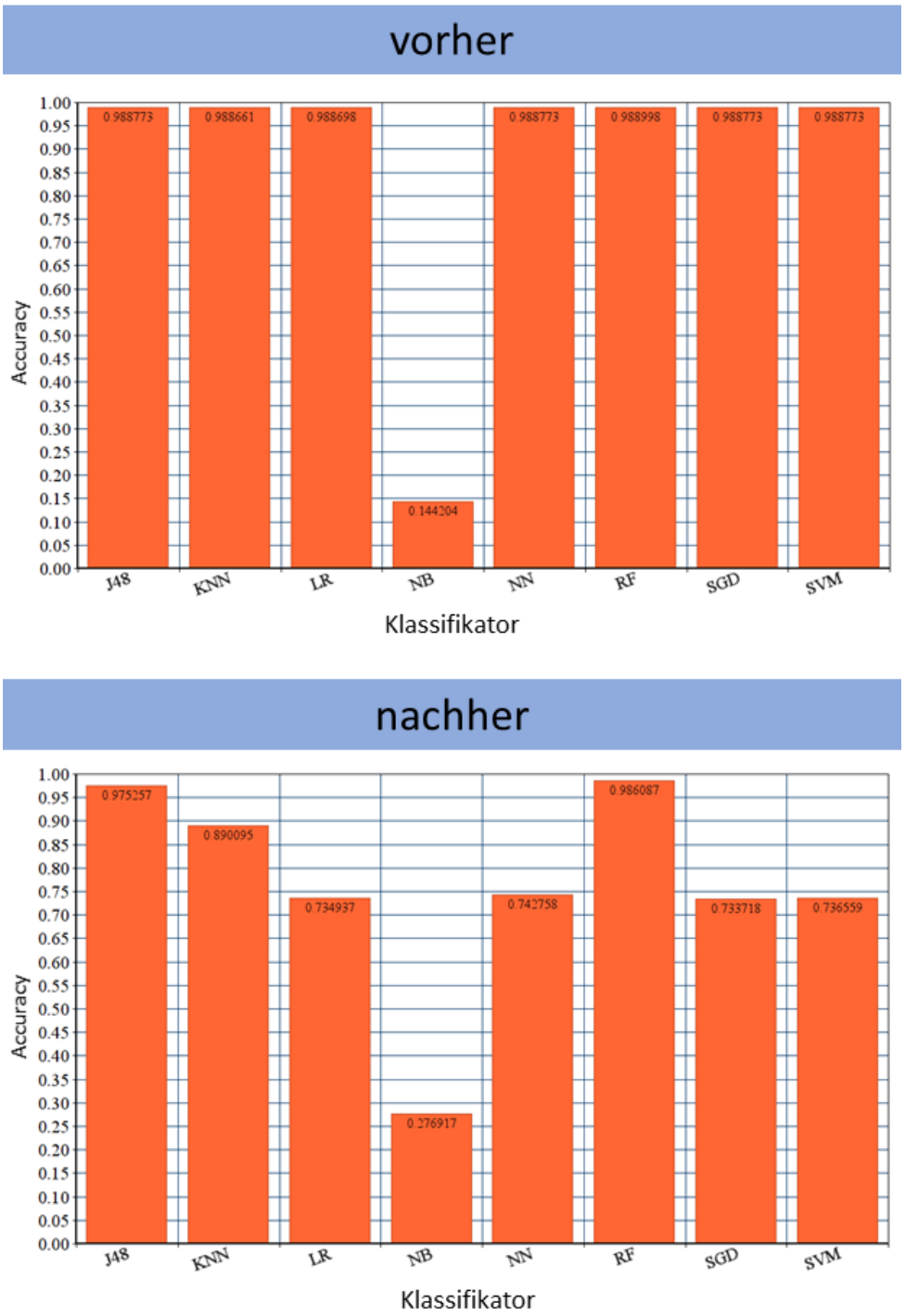


Abbildung 4.4: Vergleich der Accuracies je Klassifikator vor und nach der Anwendung des SMOTE-Algorithmus auf das dateibasierte Datenset

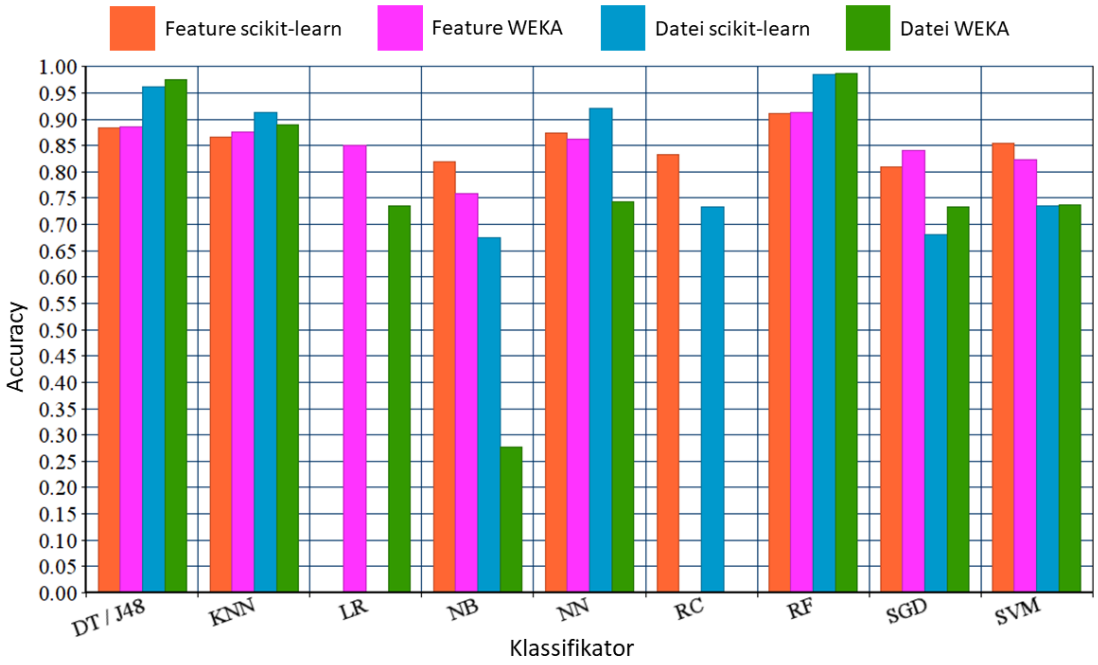


Abbildung 4.5: Vergleich der Klassifikatoren und Werkzeuge im Hinblick auf ihre Accuracies

Tabelle 4.3: Übersicht der Konfigurationen des featurebasierten Datensets

Klassifikator	Konfiguration scikit-learn	Konfiguration WEKA
DT / J48	• max_features = „sqrt“ • Split-Ratio: 80:20 • Features: alle außer ADEV, DDEV	• Split-Ratio: 10-x-v • Features: alle außer DDEV, OEXP, • MODD, NLOC
KNN	• k = 2 • Split-Ratio: 75:25	• k = 1 • Split-Ratio: 10-x-v
LR		• Split-Ratio: 70:30
NB	• Multinomial-NB • Split-Ratio: 75:25	• Split-Ratio: 10-x-v • Features: COMM, EXP, MODD, ADDL
NN	• hidden_layer_sizes = (13,13,13) • max_iter = 500 • Split-Ratio: 70:30	• Hidden-Layer = (13,13,13) • Split-Ratio: 85:15
RC	• StandardScaler • Split-Ratio: 75:25 • Features: alle außer EXP, CYCO	
RF	• n_estimators = 200 • max_features = „sqrt“ • Split-Ratio: 80:20 • Features: alle außer DDEV, MODD, REML	• Iterationen: 200 • Split-Ratio: 10-x-v • Features: alle außer DDEV, OEXP
SGD	• loss = „log“ • penality = „elasticnet“ • Split-Ratio: 85:15 • Features: EXP, OEXP	• Split-Ratio: 70:30 • Features: alle außer REML
SVM	• LinearSVC • max_iter = 20000 • StandardScaler • Split-Ratio: 75:25 • Features: alle außer DDEV, EXP, REML	• Split-Ratio: 70:30 • Features: alle außer REML

Tabelle 4.4: Übersicht der Konfigurationen des dateibasierten Datensets

Klassifikator	Konfiguration scikit-learn	Konfiguration WEKA
DT / J48	• max_features = „sqrt“ • Split-Ratio: 85:15 • Features: DDEV, EXP	• Split-Ratio: 10-x-v
KNN	• k = 1 • Split-Ratio: 75:25	• k = 10 • Split-Ratio: 10-x-v
LR		• Split-Ratio: 85:15
NB	• Bernoulli-NB • Split-Ratio: 75:25	• Split-Ratio: 70:30 • Features: MODS, CYCO
NN	• hidden_layer_sizes = (13,13,13) • max_iter = 500 • Split-Ratio: 80:20	• Hidden-Layer = (13,13,13) • Split-Ratio: 75:25
RC	• StandardScaler • Split-Ratio: 85:15 • Features: alle außer DDEV	
RF	• n_estimators = 200 • max_features = „sqrt“ • Split-Ratio: 85:15 • Features: alle außer EXP, OEXP	• Iterationen: 200 • Split-Ratio: 85:15
SGD	• loss = „log“ • penality = „elasticnet“ • Split-Ratio: 80:20 • Features: ADDL	• Split-Ratio: 85:15
SVM	• LinearSVC • max_iter = 20000 • StandardScaler • Split-Ratio: 75:25	• Split-Ratio: 85:15

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 und auf Werten von sogenannten Konfusionsmatrizen basieren. Zudem umfasst dieses Kapitel eine Erläuterung von Herausforderungen und Limitationen, die im Laufe der Bearbeitung der Arbeit festgestellt worden sind. Abschließen wird dieses Kapitel ein Vergleich der Klassifikatoren zu einer klassischen dateibasierten Methode, welche aus der wissenschaftlichen Literatur entnommen wurde.

5.1 Herausforderungen und Limitationen

szz
absprechen mit Daniel ...

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 [27]. Der Großteil dieser Metriken lässt sich anhand von Werten einer sogenannten Konfusionsmatrix berechnen und messen allesamt die Performanz der Vorhersagen von Klassifikatoren. 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 [27]. 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 5.1 dargestellt.

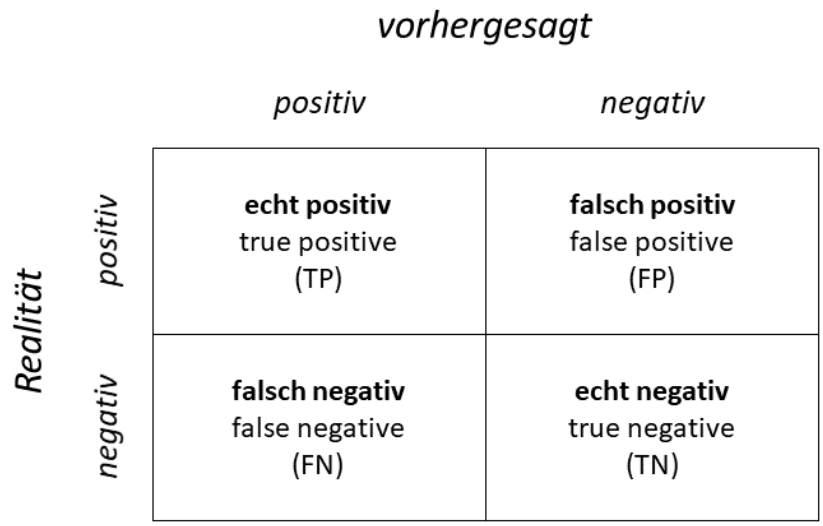


Abbildung 5.1: allgemeine Konfusionsmatrix

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

- Treffergenauigkeit (Accuracy)
  Dieser Wert misst die Treffergenauigkeit der Vorhersagen des Klassifikators und gibt an, inwieweit dessen Vorhersagen mit der modellierten Realität übereinstimmen [27]. Die Formel zur Berechnung der Accuracy lautet:

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

  Das Ergebnis der Berechnung ist ein prozentualer Wert. 100% stellen damit die bestmögliche Accuracy dar.

- Echt-Positiv-Rate / Trefferquote (TP-Rate / Recall)
  Dieser Wert gibt den Anteil der korrekt als positiv gewerteten Vorhersagen an [2]. Die Formel zur Berechnung der TP-Rate bzw. des Recalls lautet:

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

  Das Ergebnis der Berechnung ist ein prozentualer Wert. 100% stellen damit die bestmögliche TP-Rate bzw. den bestmöglichen Recall dar. Beide Begriffe werden parallel für die Berechnung der gezeigten Formel verwendet.

- Positiver Vorhersagewert (Precision)
  Dieser Wert gibt die Anzahl der positiven Vorhersagen an, die auch tatsächlich zur positiven Klasse gehören [27]. Die Formel zur Bestimmung der Precision lautet:

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

  Das Ergebnis der Berechnung ist ein prozentualer Wert. 100% stellen damit die bestmögliche Precision dar.

- F-Maß (F-Score)
  Dieser Wert berechnet das harmonische Mittel der Werte Precision und Recall und liegt somit zwischen diesen beiden Werten, jedoch näher am kleineren Wert [27]. Die Formel zur Berechnung des F-Scores lautet:

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

  Das Ergebnis der Berechnung ist ein prozentualer Wert. 100% stellen damit den bestmöglichen F-Score dar.

Darüber hinaus wurden die sogenannten ROC-Kurven (ROC curve) der einzelnen Klassifikatoren ermittelt. Diese Wahrscheinlichkeitskurven bzw. -graphen (ROC = Receiver Operating characteristic, Betriebsverhalten des Empfängers), beschreiben das Verhältnis zwischen der TP-Rate (y-Achse) und der FP-Rate (x-Achse) [27, 18]. Die Falsch-Positiv-Rate (FP-Rate) gibt dabei den Anteil der fälschlicherweise als positiv gewerteten Vorhersagen an [2]. Sie wird mittels der folgenden Formel berechnet:

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

Wie bei allen Metriken ist das Ergebnis der FP-Rate ein prozentualer Wert, der bestenfalls möglichst gering ausfallen sollte. Sowohl die TP-Rate als auch die FP-Rate geben nur singuläre Werte an, aus welchen sich keine Graphen herleiten lassen. Jeder Klassifikator errechnet jedoch im Rahmen der Vorhersage eines Datensatzes Wahrscheinlichkeiten, die die Zugehörigkeit zu den Werten der Zielklasse darstellen [13]. In der Regel wird ein Datensatz der positiven Klasse zugeordnet, wenn die Wahrscheinlichkeit einen Schwellenwert von 0,5% übersteigt - Datensätze, die diesen Schwellenwert unterschreiten werden wiederum der negativen Klasse zugeordnet [13]. Wird der Schwellenwert erhöht, so werden weniger Datensätze der positiven Klasse zugeordnet, wohingegen im Falle einer Absenkung des Schwellenwertes mehr Datensätze der positiven Klasse zugeordnet werden [13]. Für die Erstellung der ROC-Kurven werden somit die Werte der TP-Rate und der FP-Rate unter der Berücksichtigung der Schwellenwerte im Bereich von 0,0 bis 1,0 gegenübergestellt.

Eine weitere Metrik, die in Verbindung mit der ROC-Kurve auftritt, ist der ROC-Bereich (ROC area). Dieser Wert, der anhand der ROC-Kurve berechnet wird und auch AUC-Bereich (AUC = Area Under Curve, Bereich unter der Kurve) genannt wird, gibt an, in wieweit ein Klassifikator in der Lage ist, zwischen den Werten der Zielklassen zu unterscheiden [18]. Je höher dieser Wert ist (1,0 ist das Maximum), desto besser trifft der Klassifikator korrekte Vorhersagen [18].

Am Beispiel des Schwellenwertes 0,5 wird nun die Interpretation der ROC-Kurve und des ROC-Bereiches vorgenommen. Unterstützt wird dies durch grafische Beispiele, die in Abbildung 5.2 gezeigt werden. Der Idealfall ist in den (a) und (b) dargestellt. Die Wahrscheinlichkeitskurven der Zielklasse (a) weisen keine Überlappung auf. Die Werte der Zielklasse sind somit allesamt korrekt zugeordnet worden, sodass der Klassifikator korrekt zwischen diesen unterscheiden kann. Die diesem Fall entsprechende ROC-Kurve ist in (b) dargestellt. Der ROC-Bereich beträt in diesem Fall 1,0. Ein „Normalfall“ ist in (c) und (d) dargestellt. Es ist in (c) zu erkennen, dass die Wahrscheinlichkeitskurven überlappen, sodass falsche Zuordnungen getroffen werden. Die entsprechende ROC-Kurve ist in (d) dargestellt. Der entsprechende ROC-Bereich beträgt im hier gezeigten Fall 0,7. Dies bedeutet, dass 70% der Zuordnungen richtig getroffen werden. Verbessert werden können die Werte möglicherweise, wenn der Schwellenwert verändert wird. Der „Worst Case“, der bei der Performanzmessung mittels ROC-Kurven auftreten kann, ist in (e) und (f) dargestellt. In (e) ist zu erkennen, dass sich die Wahrscheinlichkeitskurven vollständig überlappen. Es findet somit eine willkürliche Zuordnung statt. Die

zugehörige ROC-Kruve (f) entspricht einer Winkelhalbierenden. Ein weiterer Fehlerfall ist in (g) und (h) dargestellt. Die Wahrscheinlichkeitskurven (g) zeigen, dass die Zuordnungen gegenteilig erfolgen und somit der jeweils dem anderen Wert der Zielklasse falsch zugeordnet werden. Der entsprechende ROC-Bereich beträgt 0, da wie in (h) zu sehen ist, keine Fläche unter der Kurve vorhanden ist.

Alle vorgestellten Metriken werden automatisiert von den Werkzeugen scikit-learn und WEKA berechnet. Ferner besitzen beide Werkzeuge die Fähigkeit, ROC-Kurven mit den entsprechenden ROC-Werten auszugeben. Die im Rahmen der Evaluation der Klassifikatoren ermittelten Werte der Metriken sowie die ROC-Kurven und Werte der ROC-Bereiche werden im folgenden Abschnitt aufgeführt und gedeutet.

RQ3A: WELCHE MITEINANDER VERGLEICHBAREN MERKMALE BESITZEN DIE KLASSIFIKATOREN?

Die Merkmale zum Vergleich stellen Evaluationsmetriken dar, welche anhand der Ergebnisse des Tests der Klassifikatoren errechnet werden und die Performanz der Vorhersagen auf verschiedene Weisen messen. Welche Metriken berechnet berechnet wurden beantwortet Forschungsfrage RQ3b.

RQ3B: WELCHE METRIKEN KÖNNEN FÜR DEN VERGLEICH VERWENDET WERDEN?

Für den Vergleich der Klassifikatoren werden klassische Evaluationsmetriken verwendet, welche auf Basis der Konfusionsmatrizen berechnet werden. Ebenfalls hinzugezogen werden die jeweiligen ROC-Kurven der Klassifikatoren mit ihren ROC-Bereichen. Diese Metriken stellen einen Standard für die Messung von Klassifikatoren dar.

5.2.2 Ergebnisse und Diskussion

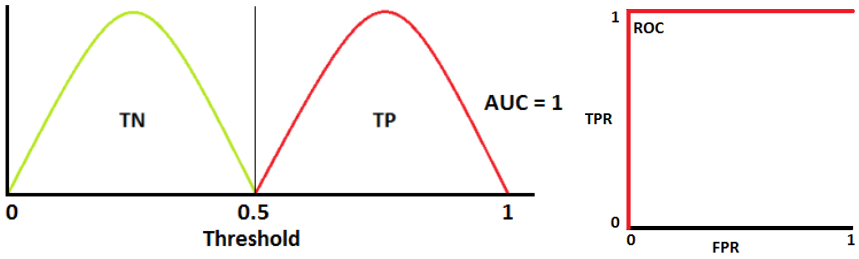
Dieser Abschnitt dient zur Präsentation der Ergebnisse der Evaluation der Klassifikatoren anhand der zuvor vorgestellten Evaluationsmetriken. Ferner werden die vorgestellten Ergebnisse innerhalb und zwischen der Datensets verglichen und interpretiert.

RQ3C: WELCHE VOR- UND NACHTEILE BESITZT EIN KLASSIFIKATOR?

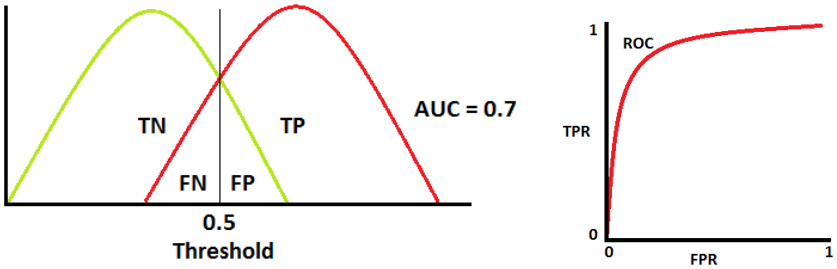
Hier soll mal so viel Text stehen, damit der ganze Text nicht nur in einer Zeile steht sondern in mindestens zwei oder mehr Zeilen, denn andernfalls werden wir nicht sehen können ob der Rahmen nur um die erste Zeile geht, oder wie wir wollen sich um den ganzen Absatz zieht.

5.3 Vergleich zu nicht-featurebasierten Methoden

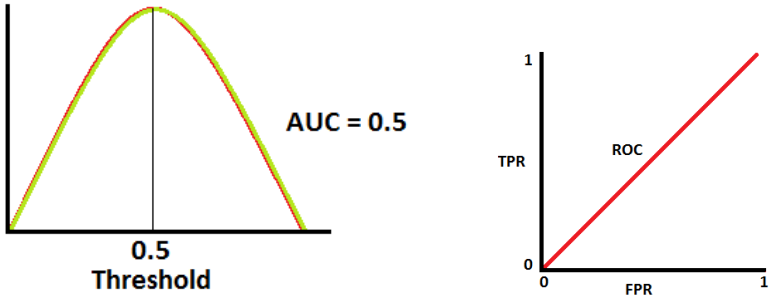
Zum besseren Vergleich der Ergebnisse des Datensets mit neuartiger Fokussierung auf Software-Features, wird nicht nur das unter Zuhilfenahme der gleichen Metriken erstellte dateibasierte Datensets hinzugezogen, sondern zusätzlich ein Datenset, dessen Erstellung aus der wissenschaftlichen Literatur entnommen 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 5.5 aufgeführt sind. [17]. 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



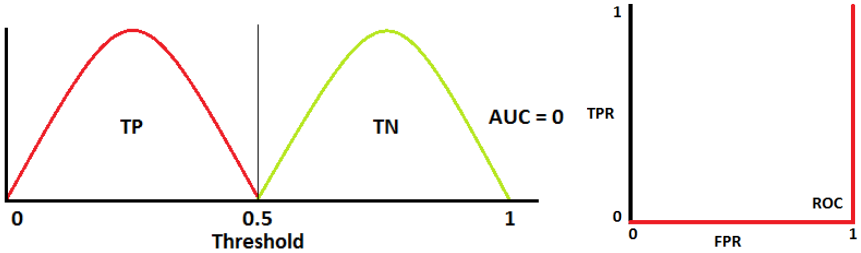
(a) Idealfall **(b)** Idealfall



(c) „Normalfall“ **(d)** „Normalfall“



(e) „Worst Case“ **(f)** „Worst Case“



(g) Fehlerfall **(h)** Fehlerfall

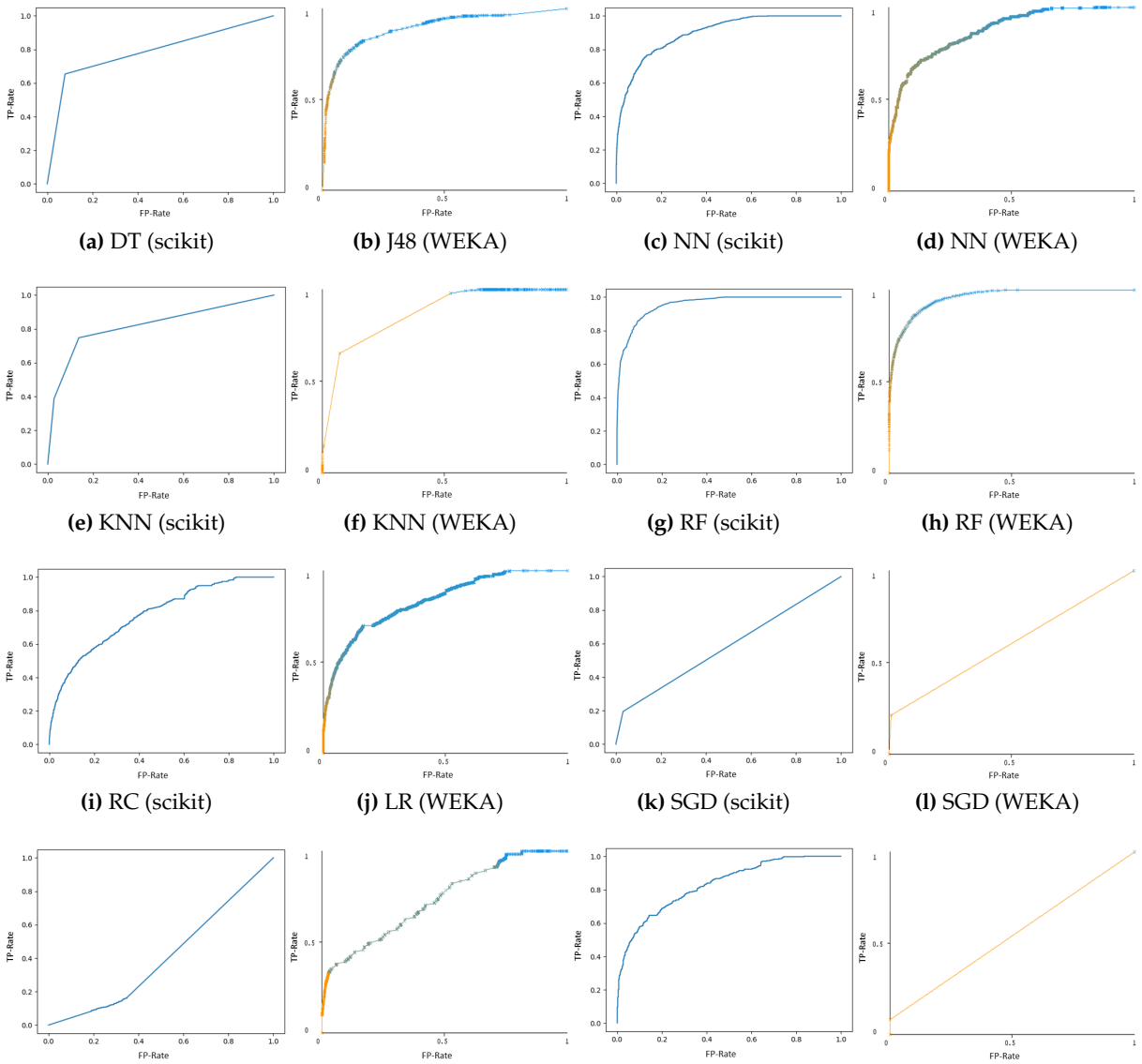
Abbildung 5.2: Beispiel zur Interpretation der ROC-Kurve und des ROC-Bereiches (TPR = TP-Rate, FPR = FP-Rate, Threshold = Schwellenwert) [18]

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

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



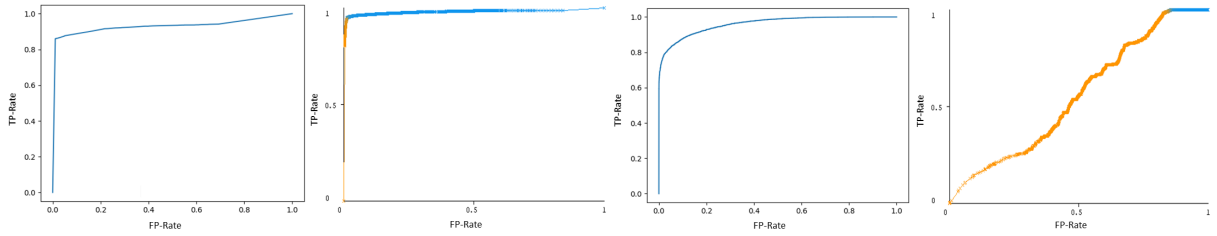
(a) DT (scikit) **(b)** J48 (WEKA) **(c)** NN (scikit) **(d)** NN (WEKA)

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

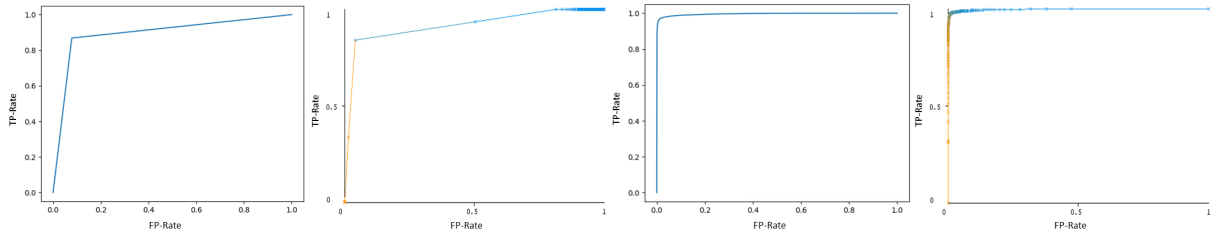
(i) RC (scikit) **(j)** LR (WEKA) **(k)** SGD (scikit) **(l)** SGD (WEKA)

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

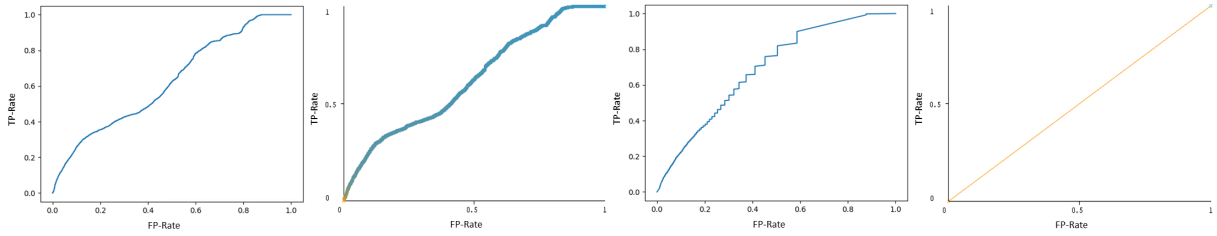
Abbildung 5.3: ROC-Kurven der Klassifikatoren des featurebasierten Datensets



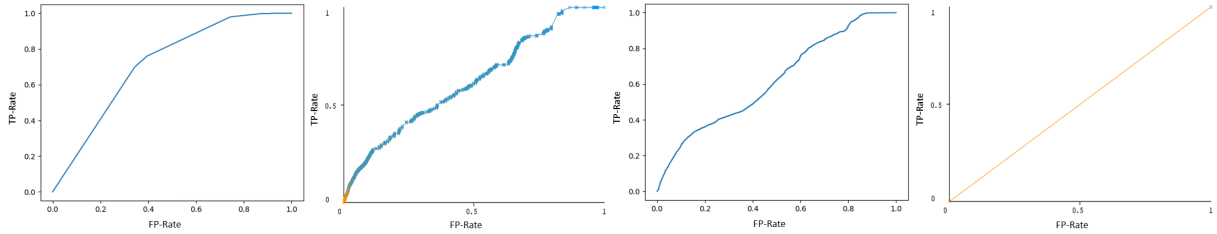
(a) DT (scikit) **(b)** J48 (WEKA) **(c)** NN (scikit) **(d)** NN (WEKA)



(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 5.4: ROC-Kurven der Klassifikatoren des dateibasierten Datensets

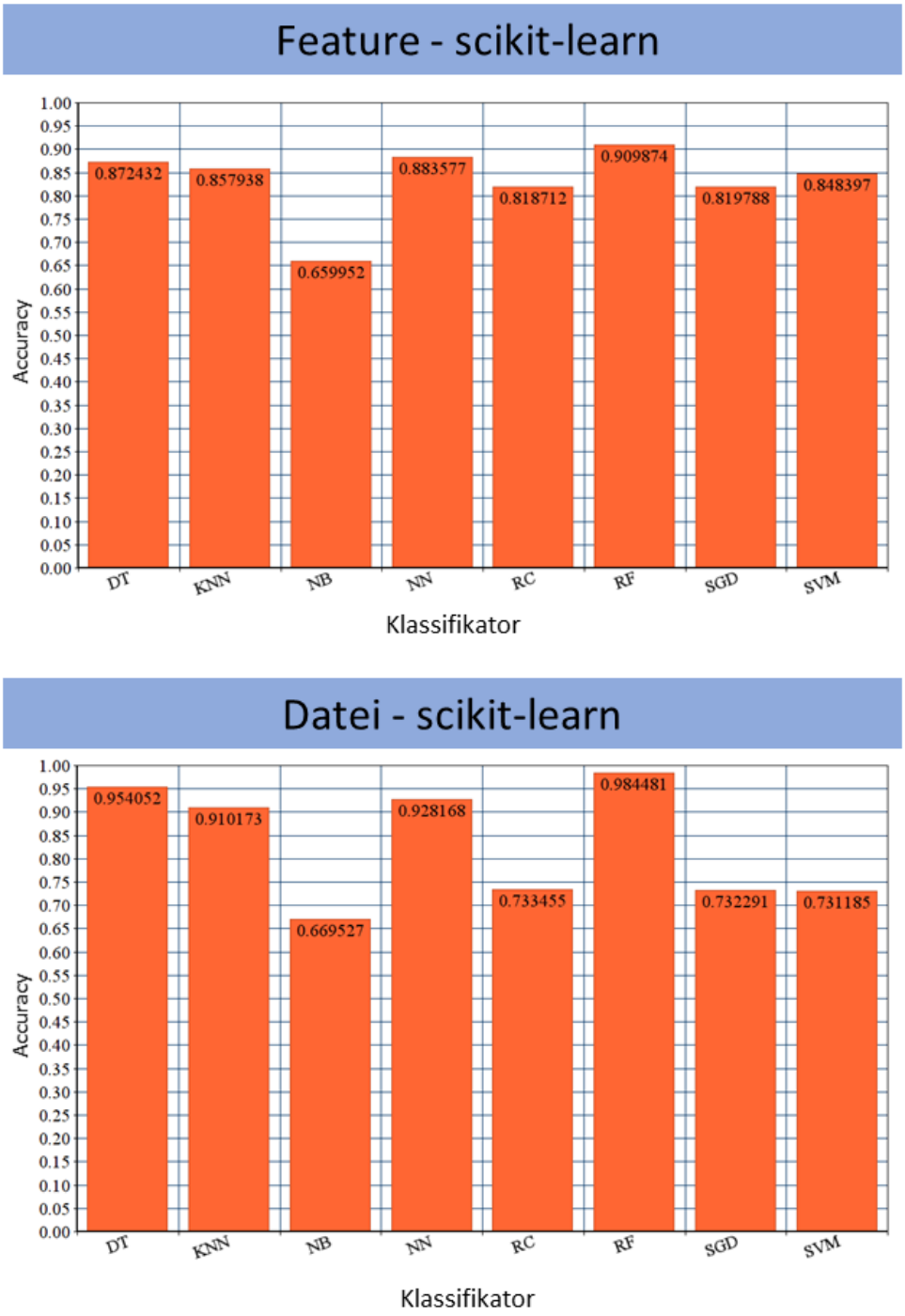


Abbildung 5.5: Vergleich der Accuracies zwischen den Datensets der scikit-Klassifikatoren

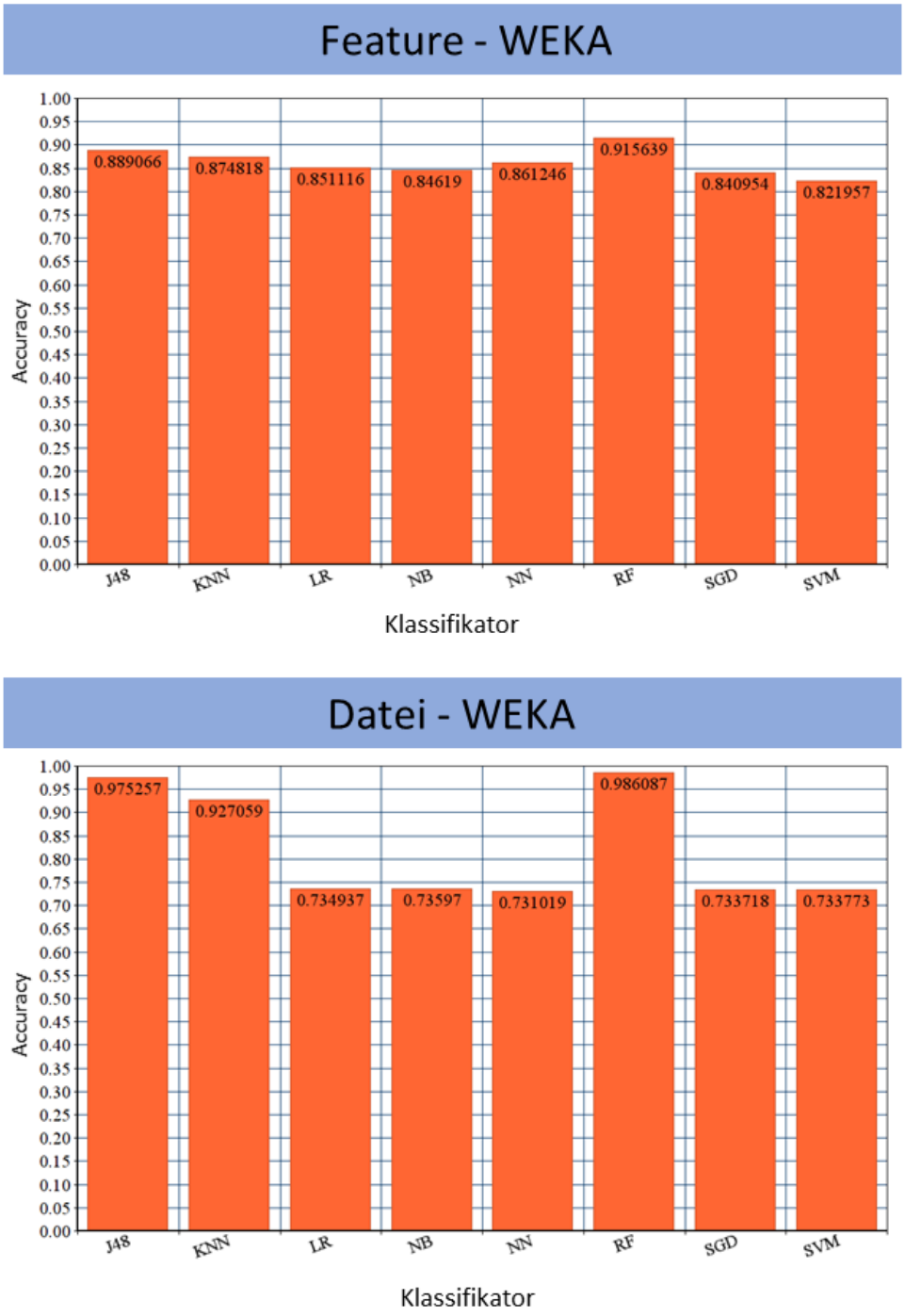


Abbildung 5.6: Vergleich der Accuracies zwischen den Werkzeugen der WEKA-Klassifikatoren

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

		featurebasiertes Datenset			**dateibasiertes Datenset**		
		Fehlerfrei	**Defekt**	**gew. Mittel**	**Fehlerfrei**	**Defekt**	**gew. Mittel**
DT	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
KNN	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
NB	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
NN	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
RC	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
RF	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
SGD	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
SVM	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

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

		featurebasiertes Datenset			**dateibasiertes Datenset**		
		fehlerfrei	**defekt**	**gew. Mittel**	**fehlerfrei**	**defekt**	**gew. Mittel**
DT	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
KNN	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
LR	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
NB	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
NN	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
RF	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
SGD	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
SVM	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

Tabelle 5.5: Übersicht der berechneten Metriken nach [17]

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	$WeightedAge = \frac{\sum_{i=1}^{N} Age(i) * LOC\_ADDED(i)}{\sum_{i=1}^{N} LOC\_ADDED(i)}$

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.

Zur besseren Vergleichbarkeit dieses Ansatzes, wurden die Metriken für ein weiteres Datenset unter Berücksichtigung des Feature-Aspekts berechnet. Dieses Datenset basiert ebenfalls aus den mit PyDriller erhaltenen Rohdaten und beinhaltet nur jede Dateien, in welchen ein Feature entfernt, hinzugefügt oder verändert wurde. Im weiteren Verlauf des Abschnitts wird dieses Datenset als „vorhandenes Datenset“ bezeichnet.

RQ3D: WIE LASSEN SICH DIE KLASSIFIKATOREN MIT WEITEREN VORHERSAGE-TECHNIKEN, DIE KEINE FEATURES NUTZEN, VERGLEICHEN?

Es wurde auf eine Methode zur Erstellung eines dateibasierten Datensets aus der wissenschaftlichen Literatur zurückgegriffen [17]. Dieses Datenset basiert auf den mittels PyDriller abgerufenen Daten und umfasst 17 Attribute. Zum besseren Vergleich wurde ein weiteres Datenset erstellt, welches nur jene Dateien umfasst, in denen sich Featurecode befindet.

Tabelle 5.6: Konfusionsmatrizen der Evaluation der Klassifikatoren des klassischen Datensets

		vorhandenes Datenset			Datenset nach [17]		
	Ermittelt ->	**Fehlerfrei**	**Defekt**	**Total**	**Fehlerfrei**	**Defekt**	**Total**
J48	Realität fehlerfrei	7085	345	7430	86965	1071	88036
	Realität defekt	723	1325	2048	1691	30549	32240
	Total	7808	1670	9478	88656	31620	120276
KNN	Realität fehlerfrei	6946	484	7430	86121	1915	88036
	Realität defekt	850	1198	2048	3867	28373	32240
	Total	7796	1682	9478	89988	30288	120276
LR	Realität fehlerfrei	7349	81	7430	13094	134	13228
	Realität defekt	1946	102	2048	4676	137	4813
	Total	9295	183	9478	17770	271	18041
NB	Realität fehlerfrei	65	1770	1835	7232	19102	26334
	Realität defekt	24	510	534	527	9222	9749
	Total	89	2280	2369	7759	28324	36083
NN	Realität fehlerfrei	1110	4	1114	30607	153	30760
	Realität defekt	204	104	308	7646	3691	11337
	Total	1314	108	1422	38253	3844	42097
RF	Realität fehlerfrei	7314	116	7430	87694	342	88036
	Realität defekt	583	1465	2048	936	31277	32213
	Total	7897	1581	9478	88630	31619	120249
SGD	Realität fehlerfrei	7429	1	7430	13221	7	13228
	Realität defekt	2043	5	2048	4813	0	4813
	Total	9472	6	9478	18034	7	18041
SVM	Realität fehlerfrei	1114	0	1114	13228	0	13228
	Realität defekt	308	0	308	4813	0	4813
	Total	1422	0	1422	18041	0	18041

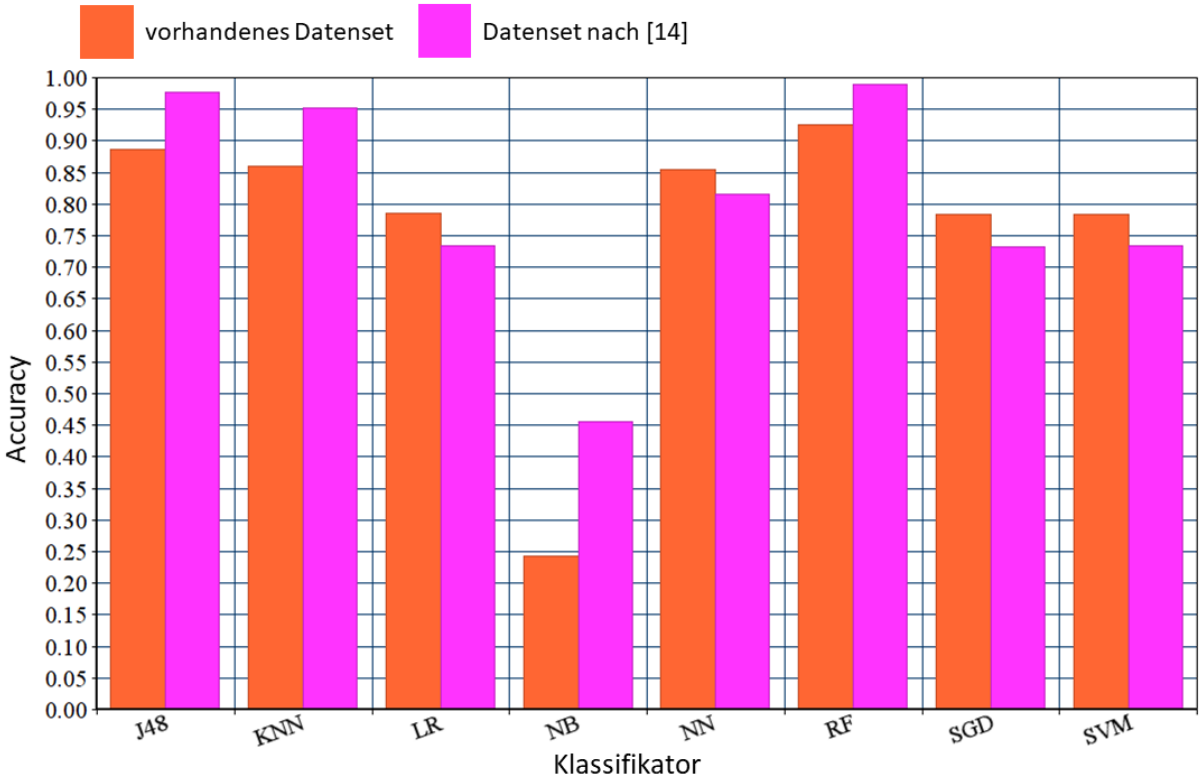
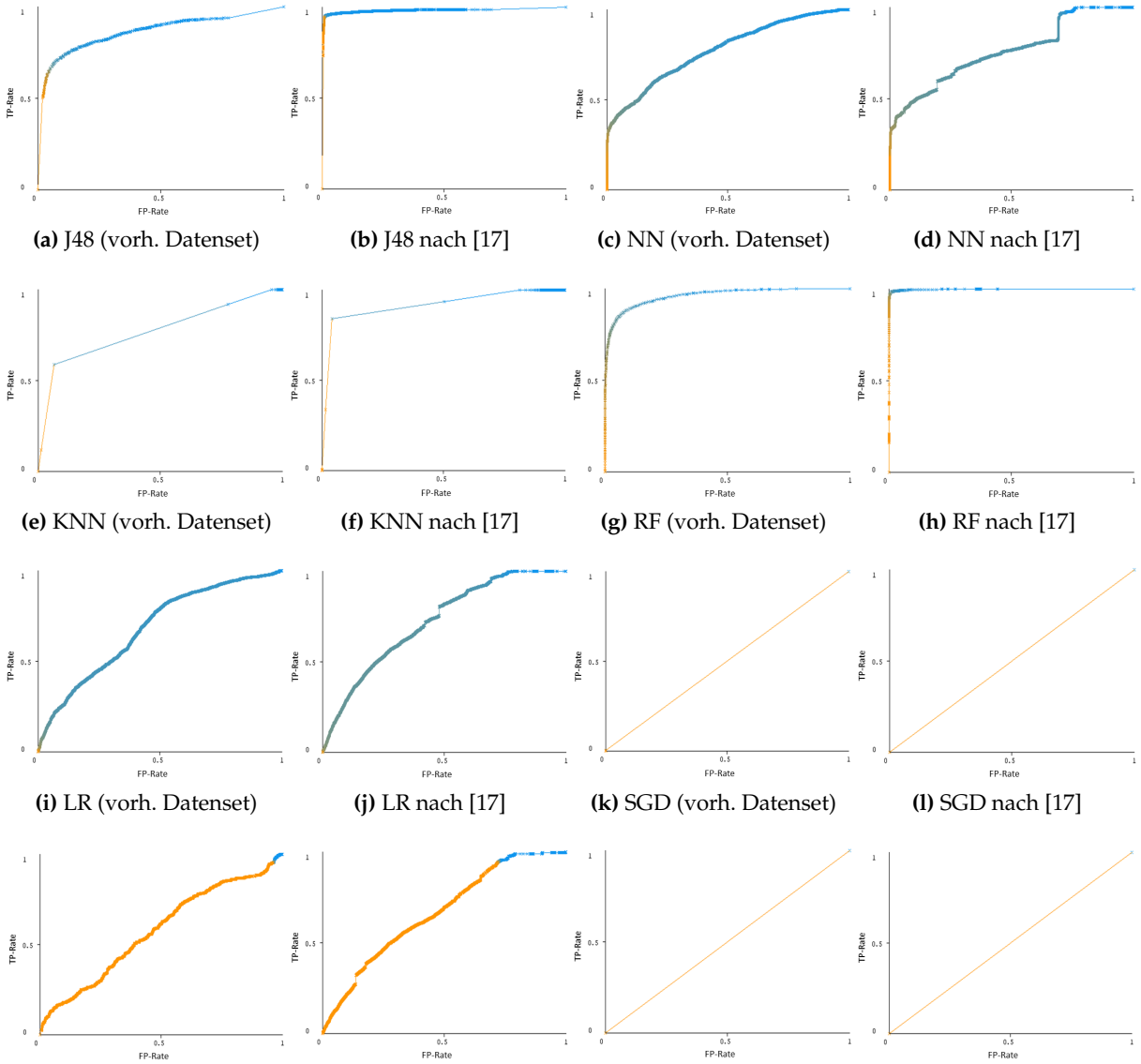


Abbildung 5.7: Übersicht der Accuracies der jeweiligen Klassifikatoren und Datensets



(a) J48 (vorh. Datenset) **(b)** J48 nach [17] **(c)** NN (vorh. Datenset) **(d)** NN nach [17]

(e) KNN (vorh. Datenset) **(f)** KNN nach [17] **(g)** RF (vorh. Datenset) **(h)** RF nach [17]

(i) LR (vorh. Datenset) **(j)** LR nach [17] **(k)** SGD (vorh. Datenset) **(l)** SGD nach [17]

(m) NB (vorh. Datenset) **(n)** NB nach [17] **(o)** SVM (vorh. Datenset) **(p)** SVM nach [17]

Abbildung 5.8: ROC-Kurven der Klassifikatoren und Datensets

Tabelle 5.7: Ergebnisse der Evaluationsmetriken auf Basis der Konfusionsmatrix

		vorhandenes Datenset			**Datenset nach**		
		fehlerfrei	**defekt**	**gew. Mittel**	**fehlerfrei**	**defekt**	**gew. Mittel**
DT	Precision	0,91	0,79	0,88	0,98	0,97	0,98
	Recall	0,95	0,65	0,89	0,99	0,95	0,95
	F-Score	0,93	0,71	0,88	0,98	0,96	0,98
	ROC-Area	0,86	0,86	0,86	0,98	0,98	0,98
KNN	Precision	0,89	0,71	0,85	0,96	0,94	0,95
	Recall	0,94	0,59	0,86	0,98	0,88	0,95
	F-Score	0,91	0,64	0,85	0,97	0,91	0,95
	ROC-Area	0,77	0,77	0,77	0,95	0,95	0,95
LR	Precision	0,79	0,58	0,74	0,74	0,51	0,68
	Recall	0,99	0,05	0,79	0,99	0,03	0,73
	F-Score	0,89	0,09	0,71	0,85	0,05	0,63
	ROC-Area	0,68	0,68	0,68	0,72	0,72	0,72
NB	Precision	0,73	0,22	0,62	0,93	0,33	0,77
	Recall	0,04	0,96	0,24	0,28	0,95	0,46
	F-Score	0,07	0,36	0,13	0,42	0,48	0,44
	ROC-Area	0,57	0,57	0,57	0,66	0,66	0,66
NN	Precision	0,85	0,96	0,87	0,80	0,96	0,84
	Recall	1,00	0,34	0,85	1,00	0,33	0,82
	F-Score	0,91	0,50	0,82	0,89	0,49	0,78
	ROC-Area	0,83	0,83	0,83	0,76	0,76	0,76
RF	Precision	0,93	0,93	0,93	0,99	0,99	0,99
	Recall	0,98	0,72	0,93	1,00	0,97	0,99
	F-Score	0,95	0,81	0,92	0,99	0,98	0,99
	ROC-Area	0,96	0,96	0,96	1,00	1,00	1,00
SGD	Precision	0,78	0,93	0,80	0,73	0,00	0,54
	Recall	1,00	0,00	0,78	1,00	0,00	0,73
	F-Score	0,88	0,01	0,69	0,85	0,00	0,62
	ROC-Area	0,50	0,50	0,50	0,50	0,50	0,50
SVM	Precision	0,78	?	?	0,73	?	?
	Recall	1,00	0,00	0,78	1,00	0,00	0,73
	F-Score	0,88	?	?	0,85	?	?
	ROC-Area	0,50	0,50	0,50	0,50	0,50	0,50

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] Abdullah Alsaeedi und Mohammad Zubair Khan. „Software Defect Prediction Using Supervised Machine Learning and Ensemble Techniques: A Comparative Study“. In: *Journal of Software Engineering and Applications* 12.05 (2019), S. 85–100. DOI: 10.4236/jsea.2019.125007.

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

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

[10] Rohith Gandhi. *Support Vector Machine — Introduction to Machine Learning Algorithms*. Online. Juni 2018. URL: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.

[11] Awni Hammouri u. a. „Software Bug Prediction using Machine Learning Approach“. In: *International Journal of Advanced Computer Science and Applications* 9.2 (2018). DOI: 10.14569/ijacsa.2018.090212.

[12] 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.

[13] KNIMETV. *What is an ROC Curve?* Video on YouTube. Sep. 2019. URL: https://www.youtube.com/watch?v=kWDHmroVWs8.

[14] 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.

[15] 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.

[16] Stefan Luber und Nico Litzel. *Was ist eine Support Vector Machine?* Online. Nov. 2019. URL: https://www.bigdata-insider.de/was-ist-eine-support-vector-machine-a-880134/.

[17] 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.

[18] Sarang Narkhede. *Understanding AUC - ROC Curve*. Online. Juni 2018. URL: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.

[19] 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].

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

[21] 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.

[22] 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.

[23] 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.

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

[25] Jacek Ratzinger, Thomas Sigmund und Harald C. Gall. „On the relation of refactorings and software defect prediction“. In: *Proceedings of the 2008 international workshop on Mining software repositories - MSR '08*. ACM Press, 2008. DOI: 10.1145/1370750.1370759.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] Aishwarya V Srinivasan. *Stochastic Gradient Descent — Clearly Explained !!* Online. Sep. 2019. URL: https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31.

[32] Richard M. Stallmann und Zachary Weinberg. *The C Preprocessor*. For GCC version 6.3.0. Free Software Foundation, Inc. 2016. URL: https://scicomp.ethz.ch/public/manual/gcc/6.3.0/cpp.pdf.

[33] 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.

[34] Ian H. Witten. *Data Mining with WEKA: Cross-valiadation*. Online-Video. URL: https://www.futurelearn.com/courses/data-mining-with-weka/0/steps/25384.

[35] 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.

[36] 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 Softwareprojekte

	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

Detaillierte Ergebnisse der Evaluationsmetriken

Tabelle B.1: Ergebnisse der Evaluationsmetriken auf Basis der Konfusionsmatrix (scikit-learn)

		featurebasiertes Datenset			**dateibasiertes 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 B.2: Ergebnisse der Evaluationsmetriken auf Basis der Konfusionsmatrix (WEKA)

		featurebasiertes Datenset			**dateibasiertes 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 B.3: Ergebnisse der Evaluationsmetriken auf Basis der Konfusionsmatrix

		vorhandenes Datenset			**Datenset nach [17]**		
		fehlerfrei	**defekt**	**gew. Mittel**	**fehlerfrei**	**defekt**	**gew. Mittel**
DT	TP-Rate	0,95	0,65	0,89	0,99	0,95	0,98
	FP-Rate	0,35	0,05	0,29	0,05	0,01	0,04
	Precision	0,91	0,79	0,88	0,98	0,97	0,98
	Recall	0,95	0,65	0,89	0,99	0,95	0,95
	F-Score	0,93	0,71	0,88	0,98	0,96	0,98
	ROC-Area	0,86	0,86	0,86	0,98	0,98	0,98
	PRC-Area	0,93	0,73	0,89	0,99	0,96	0,98
KNN	TP-Rate	0,94	0,59	0,86	0,98	0,88	0,95
	FP-Rate	0,41	0,07	0,34	0,12	0,02	0,10
	Precision	0,89	0,71	0,85	0,96	0,94	0,95
	Recall	0,94	0,59	0,86	0,98	0,88	0,95
	F-Score	0,91	0,64	0,85	0,97	0,91	0,95
	ROC-Area	0,77	0,77	0,77	0,95	0,95	0,95
	PRC-Area	0,89	0,52	0,81	0,97	0,88	0,95
LR	TP-Rate	0,99	0,05	0,79	0,99	0,03	0,73
	FP-Rate	0,95	0,01	0,75	0,92	0,01	0,72
	Precision	0,79	0,58	0,74	0,74	0,51	0,68
	Recall	0,99	0,05	0,79	0,99	0,03	0,73
	F-Score	0,89	0,09	0,71	0,85	0,05	0,63
	ROC-Area	0,68	0,68	0,68	0,72	0,72	0,72
	PRC-Area	0,87	0,36	0,76	0,88	0,44	0,77
NB	TP-Rate	0,04	0,96	0,24	0,28	0,95	0,46
	FP-Rate	0,05	0,97	0,25	0,05	0,73	0,24
	Precision	0,73	0,22	0,62	0,93	0,33	0,77
	Recall	0,04	0,96	0,24	0,28	0,95	0,46
	F-Score	0,07	0,36	0,13	0,42	0,48	0,44
	ROC-Area	0,57	0,57	0,57	0,66	0,66	0,66
	PRC-Area	0,80	0,30	0,69	0,85	0,40	0,73
NN	TP-Rate	1,00	0,34	0,85	1,00	0,33	0,82
	FP-Rate	0,66	0,00	0,52	0,67	0,01	0,49
	Precision	0,85	0,96	0,87	0,80	0,96	0,84
	Recall	1,00	0,34	0,85	1,00	0,33	0,82
	F-Score	0,91	0,50	0,82	0,89	0,49	0,78
	ROC-Area	0,83	0,83	0,83	0,76	0,76	0,76
	PRC-Area	0,93	0,73	0,89	0,88	0,66	0,82
RF	TP-Rate	0,98	0,72	0,93	1,00	0,97	0,99
	FP-Rate	0,29	0,02	0,23	0,03	0,00	0,02
	Precision	0,93	0,93	0,93	0,99	0,99	0,99
	Recall	0,98	0,72	0,93	1,00	0,97	0,99
	F-Score	0,95	0,81	0,92	0,99	0,98	0,99
	ROC-Area	0,96	0,96	0,96	1,00	1,00	1,00
	PRC-Area	0,99	0,91	0,97	1,00	0,99	1,00
SGD	TP-Rate	1,00	0,00	0,78	1,00	0,00	0,73
	FP-Rate	1,00	0,00	0,78	1,00	0,00	0,73
	Precision	0,78	0,93	0,80	0,73	0,00	0,54
	Recall	1,00	0,00	0,78	1,00	0,00	0,73
	F-Score	0,88	0,01	0,69	0,85	0,00	0,62
	ROC-Area	0,50	0,50	0,50	0,50	0,50	0,50
	PRC-Area	0,78	0,22	0,66	0,73	0,27	0,61
SVM	TP-Rate	1,00	0,00	0,78	1,00	0,00	0,73
	FP-Rate	1,00	0,00	0,78	1,00	0,00	0,73
	Precision	0,78	?	?	0,73	?	?
	Recall	1,00	0,00	0,78	1,00	0,00	0,73
	F-Score	0,88	?	?	0,85	?	?
	ROC-Area	0,50	0,50	0,50	0,50	0,50	0,50
	PRC-Area	0,78	0,22	0,66	0,73	0,27	0,61