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 April 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.

„Der wird kein Koch, sondern was Ordentliches!“
Für dich.
In Erinnerung.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung und Motivation

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 (zum Beispiel Naïve Bayes (NB), Bernoulli-NB oder multinomieller NB) die meistgenutzten sind [32]. 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 sich dadurch aus, dass sie eine gemeinsame Menge von Features sowie eine gemeinsame Codebasis besitzen [36]. 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 Vorhersagetechnik wird die Implementation von Features mittels Präprozessor-Anweisungen,

[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.

wie `#IFDEF` und `#IFNDEF` (auch Präprozessor-Direktiven genannt) betrachtet. Dieser bisher in wissenschaftlichen Arbeiten nur einmal im Rahmen einer Fallstudie betrachtete Ansatz [24] 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 (Kapitel 3). Dieses Datenset dient wiederum zur Anlernung von einer repräsentativen Auswahl an Klassifikatoren (Kapitel 4) mit anschließender vergleichender Evaluation (Kapitel 5) dieser. Zusätzlich werden die featurebasierten Klassifikatoren mit klassischen dateibasierten Klassifikatoren (Abschnitt 5.3) verglichen, dessen Entwicklung aus einer wissenschaftlichen Arbeit entnommen wurde [18]. 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 beschreibt dabei die Freischaltung von Veränderungen an einer oder mehreren Dateien. Als Datenbasis für das Trainieren der Klassifikatoren dienen dann Commits, für die auf Grundlage eines automatisierten Verfahrens aus

der Literatur eine Klassifikation in "fehlerhaftünd "fehlerfrei"verfügbar ist. 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.

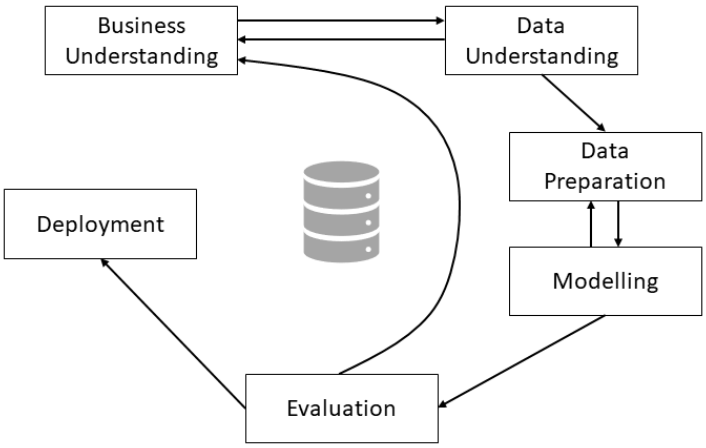


Abbildung 1.1: CRISP-DM Prozessmodell nach [7]

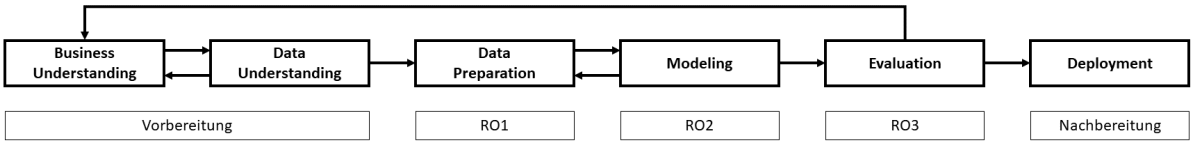


Abbildung 1.2: Phasen des CRISP-DM Prozessmodells nach [7] mit Zuordnung der Arbeitsphasen

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.

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.

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 Aufbau der Arbeit

Diese Ausarbeitung ist in sechs Kapitel unterteilt. Kapitel 1, welches mit diesem Abschnitt abgeschlossen wird, diente zur Einführung in das Thema der Masterarbeit. Ebenso stellte es die theoretischen Rahmenbedingungen der Arbeit vor. Kapitel 2 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. Kapitel 4 und Kapitel 5 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 in Kapitel 5 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 in Kapitel 6.

Zusätzlich wird die Ausarbeitung von zahlreichen Abbildungen und Tabellen zur verständlichen Verdeutlichung von Zusammenhängen ergänzt.

Kapitel 2

Hintergrund

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 entstehen kann [4, 36].

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) erstellt, welches die wählbaren Features und Constraints für mögliche Selektionen beschreibt [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 einigen 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.

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 und Listing 2.2 zu sehen. Sie wurden jeweils aus der wissenschaftlichen

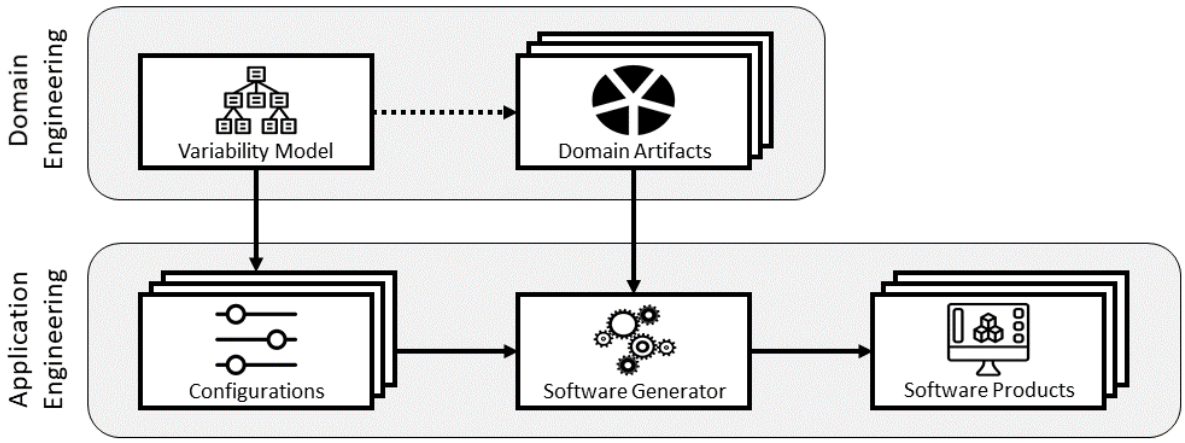


Abbildung 2.1: Generierung von Software-Produktlinien nach [36]

Literatur entnommen [17, 23]. Die Direktive `#IFDEF` leitet in `#IFDEF` den Code des Features `__unix__` ein, welcher mit der Anweisung `#ENDIF` endet. Der in den Zeilen 2 bis 6 angegebene Codeteil wird genau dann nur ausgeführt, wenn das Feature `__unix__` im Rahmen der Konfiguration des Softwareproduktes definiert beziehungsweise aktiviert ist [35]. In diesem Fall wird die Bedingung der Direktive erfolgreich erfüllt [35]. Sie schlägt fehl, wenn das Feature nicht definiert beziehungsweise aktiviert ist [35]. Die Direktive `#IFNDEF` wird für Code verwendet, der ausgeführt werden soll, wenn ein Feature nicht definiert ist. Im Falle des Beispiels in Listing 2.2 wird der in Zeile 3 angedeutete Code nur ausgeführt, wenn `NO_XMALLOC` nicht aktiviert wurde. Es besteht zudem die Möglichkeit, Features bzw. ihren Code zu verschachteln. Ein Beispiel dafür ist in Listing 2.3 angegeben. Es ist zu erkennen, dass sich der Code von `FEAT_MZSCHEME` innerhalb der bedingten Gruppe von `USE_XSMP` befindet. Der in Zeile 5 angedeutete Code kann somit nur ausgeführt werden, wenn `USE_XSMP` aktiviert ist. Im Fall von Verschachtelung beendet ein `#ENDIF` immer das nächstgelegene `#IFDEF` oder `#IFNDEF` [35]. Es besteht zudem die Möglichkeit, Direktiven mittels „und“ (`&&`, `and`) oder „oder“ (`||`, `or`) zu erweiterten Bedingungen zu verknüpfen, die zudem Negation in Form des `!`-Operators (anstelle von `#IFNDEF`) enthalten können [35, 25]. Dargestellt ist dies in Listing 2.4.

```
1 #IFDEF __unix__
2   #include "directorySelection.h"
3   #include "directoryNames.h"
4   void getDirectoryName(char* dirname
      ) {
5     getHomeDirectory(dirname);
6   }
7 #ENDIF
```

Listing 2.1: Beispieleinsatz von `#IFDEF` nach [23]

```
1 int test = 1;
2 #IFNDEF NO_XMALLOC
3   test = memory != NULL;
4 #ENDIF
5 if (test){
6  // Lines of code here..
7  }
```

Listing 2.2: Beispieleinsatz von `#IFNDEF` nach [17]

```
bool time = msec > 0;
#IFDEF USE_XSMP
  time = time && xsmp_icefd != -1;
  #IFDEF FEAT_MZSCHEME
    time = time || p_mzq > 0;
  #ENDIF
#ENDIF
if (time)
 gettime(&start_tv);
```

Listing 2.3: Beispiel eines verschachtelten Einsatzes von `#IFDEF` nach [17]

```
#IFDEF FEATURE_A && FEATURE_B
  (...)
#ENDIF
(...)
#IFDEF !FEATURE_A && FEATURE_C
  (...)
#ENDIF
```

Listing 2.4: Beispiele von erweiterten Bedingungen nach [25]

Die in den Listings 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). Ebenfalls kann ein solcher Fehler dazu führen, dass die Funktionalität des gesamten Sourcecodes beeinträchtigt wird.

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). Die Methoden in diesen Teilgebieten 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 [30]. 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 [30]. 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 2.2 anhand eines Beispiels dargestellt.

Die Abbildung zeigt den Prozess des überwachten Machine Learnings anhand des Beispiels der Erlernung eines Klassifikators zur Erkennung beziehungsweise Vorhersage von geometrischen Formen. Der Prozess beginnt mit den „gelabelten“ Eingabedaten (A). Die Werte der Label (kategorial oder numerisch) stellen dabei die zu vorhersagende Zielklasse dar. In diesem

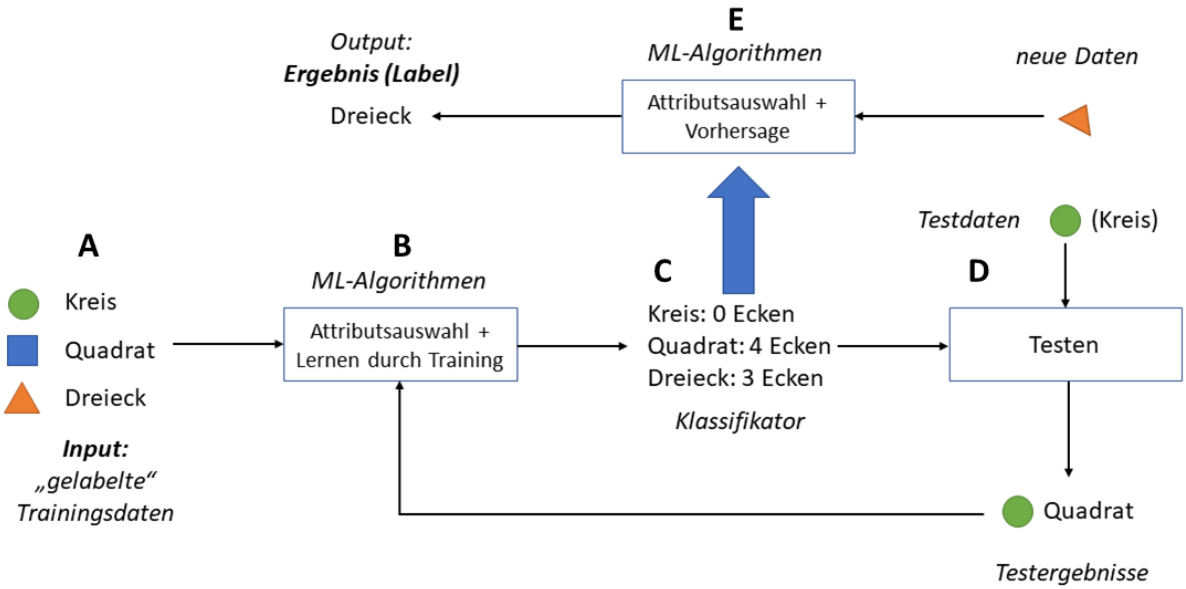


Abbildung 2.2: Allgemeiner Prozess des überwachten Machine Learnings dargestellt anhand eines Beispiels (vereinfacht)

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 Metrik „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). Die in der Regel verwendeten Verhältnisse (englisch: Split-Ratio) zwischen Trainings- und Testdaten betragen 80:20 (basierend auf dem Paretoprinzip) oder 75:25 (zum Beispiel [24]). 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. Im Voraus des Testens mit den Testdaten (D) werden in manchen Fällen zudem sogenannte Validierungsdaten verwendet. Dabei handelt es sich um eine eigenständige Teilmenge der Trainingsdaten, welche verwendet wird, um die Klassifikatoren nach jeder Erlernung zu evaluieren, um die Auswahl der Attribute hinsichtlich der Performanz auf Eignung zu prüfen [30]. Die Anwendung der Testdaten erfolgt dann im Anschluss.

Der in Abbildung 2.2 dargestellte Klassifikator stellt einen multinomiellen oder multi-class Klassifikator dar, da er zu drei oder mehr Werten der Zielklasse zuordnen kann [30]. Für viele praktische Anwendungen genügt jedoch ein binärer Klassifikator, welcher Vorhersagen zu

zwei Werten der Zielklasse trifft. Dies trifft auch auf die Klassifikatoren dieser Arbeit zu.

2.3 Fehlervorhersage mittels Machine Learning

Der Hintergrund der Fehlervorhersage mittels Machine Learning basiert auf dem zuvor vorgestellten Konzept des überwachten Machine Learnings. Als Grundlage dienen dabei meist Daten, die aus Software-Repositories entnommen beziehungsweise extrahiert werden. Viele Studien und wissenschaftliche Arbeit setzen jedoch auch auf vorgefertigte Datensets, wie zum Beispiel von der NASA oder von Eclipse [32]. Die zugehörigen Label der Datensets lauten in der Regel „fehlerfrei“ und „defekt“ und können auf verschiedene Weisen ermittelt werden. Eine gängige Methode ist die Identifizierung von korrektiven und fehlereinführenden Commits als Entscheidungsgrundlage für das Label. Die üblichen Vorgehensweisen setzen auf die Einbindung von Bugtracking-Systemen oder die Analyse von Commit-Nachrichten zur Identifizierung der korrektiven Commits [24, 39]. Fehlereinführende Commits können anschließend unter Verwendung von Git-Kommandos oder durch die Anwendung des sogenannten SZZ-Algorithmus ermittelt werden. Dieser Algorithmus wird in dieser Arbeit verwendet und in Abschnitt 3.2 erläutert. Auf Basis dieser Daten werden die Attribute bestimmt. Dabei handelt sich sich in der Regel um Metriken, die entweder die Charakteristika des Sourcecodes (Codemetriken) oder Aktivitäten und Prozesse im Bezug auf Software-Repositories (Prozessmetriken) beschreiben [32, 26]. Mithilfe dieser Attribute werden die Klassifikatoren erlernt. Eine Studie, welche 156 wissenschaftliche Arbeiten zum Thema der Fehlervorhersage mittels Machine Learning analysierte, ergab, dass besonders Entscheidungsbaum-basierte, Bayessche Verfahren, Regression und künstliche neuronale Netze als Klassifikationsalgorithmen zur Anwendung kommen [32]. Diese Algorithmen wurden unter anderem auch in dieser Arbeit verwendet. Erläuterungen können in Abschnitt 4.1 gefunden werden. Die fertig erlernten Klassifikatoren können dann auf Basis neuer Daten Vorhersagen zum Zustand einer Software treffen. Die Vorhersagen beruhen in der Regel auf defekte Dateien im Umfeld von Commits oder Releases.

Als Beispiel für zwei konkrete Anwendungen von Machine Learning gestützter Fehlervorhersage werden im Folgenden zwei wissenschaftliche Arbeiten vorgestellt. Die erste Arbeit „Comparative Analysis of the Efficiency of Change Metrics and Static Code Attributes for Defect Prediction“ von Moser et al. [18] stellt eine Methodik zur dateibasierten Fehlervorhersage vor. Die zweite Arbeit „Towards Predicting Feature Defects in Software Product Lines“ von Queiroz et al. [24] knüpft an den zuvor vorgestellten Ansatz der Software-Features an. Beide Literaturquellen werden im weiteren Verlauf dieser Arbeit eine Rolle spielen, die im jeweiligen Abschnitt erläutert wird.

Dateibasierte Fehlervorhersage

Das Beipsiel zur dateibasierten Fehlervorhersage stammt aus einer wissenschaftlichen Arbeit von Moser et al. [18]. Sie widmet sich einer vergleichenden Analyse von zwei verschiedenen Mengen von Metriken zur dateibasierten Fehlervorhersage mittels Methoden des Machine Learnings. Die Zuordnung erfolgt in „defekt“ und „defekt-frei“. Als Datenbasis dient ein vorgefertigtes Datenset von Eclipse. Auf Basis dieses Datensets wurden „produktbasierte“ Metriken (Codemetriken) und Prozessmetriken berechnet. Zur Anwendung kamen die Klassifikationsalgorithmen logistische Regression, Naïve Bayes und Entscheidungsbäume. Die Prozessmetriken stellen eine Besonderheit dieser Arbeit dar, da sie in dieser Arbeit zum ersten Mal näher betrachtet und auf ihre Eignung als Attribute zur Erlernung der Klassifikatoren erörtert

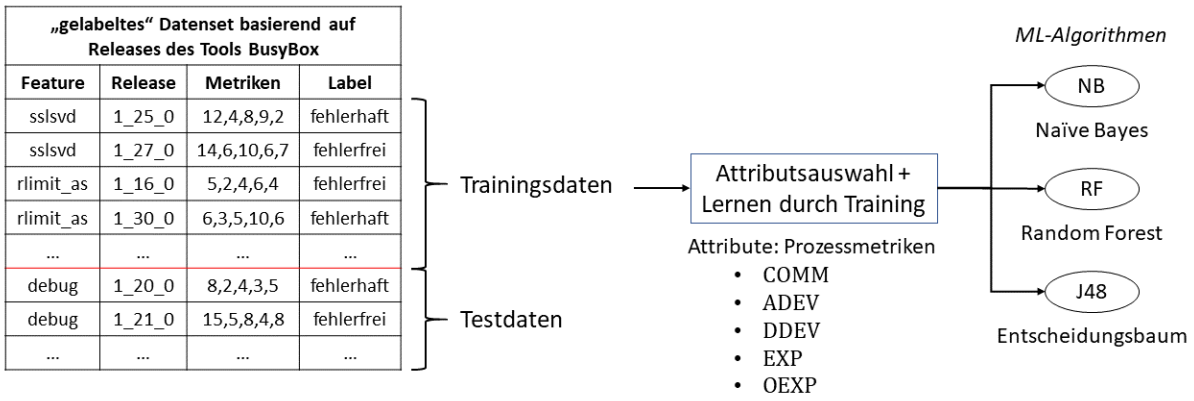


Abbildung 2.3: Teil 1: Featurebasierter Prozess des überwachten Machine Learnings nach [24]

wurden. Die Metriken berechneten unter anderem die Anzahl der Änderungen an einer Datei, die Anzahl der Autoren einer Datei, die Anzahl der hinzugefügten oder entfernten Zeilen einer Datei oder das Alter einer Datei. Das Resultat der Arbeit lautet, dass Prozessmetriken effektiver zur Fehlervorhersage genutzt werden können als Codemetriken.

Im Rahmen der Evaluation der für diese Arbeit erlernten featurebasierten Klassifikatoren wird ein zusätzliches dateibasiertes Datenset erstellt, welches auf diesen Prozessmtriken basiert und zum Performanzvergleich hinsichtlich der neuen featurebasierten Betrachtung hinzugezogen wird. Die vollständige Liste der Metriken ist in Abschnitt 5.3 aufgeführt.

Featurebasierte Fehlervorhersage

Das Beispiel zur featurebasierten Fehlervorhersage stammt aus einer wissenschaftlichen Arbeit von Queiroz et al. [24]. Bei dieser Fallstudie handelt es sich um die erste und bisher einzige Arbeit über Fehlervorhersage mit Bezug zu Software-Features. Sie 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 2.3 dargestellt.

Datenset

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

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

Tabelle 2.1: Übersicht der von [24] 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.	

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 [31, 39]. Wie im Rahmen des überwachten Machine Learning üblich, wird das Datenset in Trainings- und Testdaten in einem Verhältnis von 75:25 geteilt.

Metriken und Klassifikation

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.

Test der Klassifikatoren

Wie in Abbildung 2.4 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.

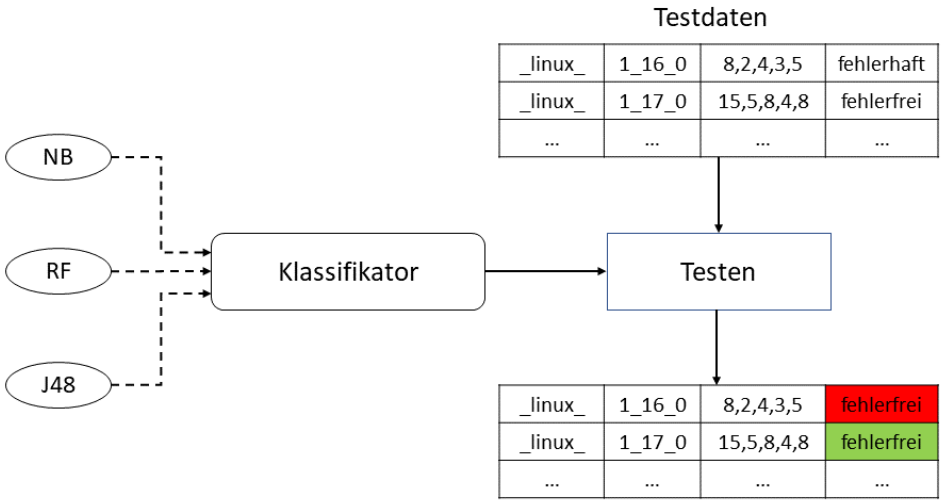


Abbildung 2.4: Teil 2: Featurebasierter Prozess des überwachten Machine Learnings nach [24]

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

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, 24]. Die für diese Arbeit verwendeten Softwareprojekte sind samt ihres Einsatzzweckes und ihrer Datenquellen in Tabelle 3.1 aufgeführt.

Um die Commit-Daten der Softwareprojekte zu erhalten wurde die Python-Library PyDriller[2] verwendet [33]. 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

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

Tabelle 3.1: Übersicht der verwendeten Softwareprojekte[1]

Projekt	Zweck	Datenquelle
Blender	3D-Modellierungstool	GitHub-Mirror
Busybox	UNIX-Toolkit	Git-Repository
Emacs	Texteditor	GitHub-Mirror
GIMP	Bildbearbeitung	GitLab-Repository
Gnumeric	Tabellenkalkulation	GitLab-Repository
gnuplot	Plotting-Tool	GitHub-Mirror
Irssi	IRC-Client	GitHub-Repository

Projekt	Zweck	Datenquelle
libxml2	XML-Parser	GitLab-Repository
lighttpd	Webserver	Git-Repository
MPSolve	Polynomlöser	GitHub-Repository
Parrot	virtuelle Maschine	GitHub-Repository
Vim	Texteditor	GitHub-Repository
xfig	Grafikeditor	Sourceforge-Repository

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, wie viele Releases je Softwareprojekt abgerufen wurden und wie viele Commits daraus resultieren.

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

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

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

Projekt	#Releases	#Commits
Blender	11	19.119
Busybox	14	4.984
Emacs	7	12.805
GIMP	14	7.240
Gnumeric	8	6.025
gnuplot	5	6.619
Irssi	7	253

Projekt	#Releases	#Commits
libxml2	10	732
lighttpd	6	2.597
MPSolve	8	668
Parrot	7	16.245
Vim	7	9.849
xfig	7	18

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, 24].

3.2 Konstruktion des Datensets

Die Konstruktion des Datensets gliedert sich in mehrere Phasen der Datenverarbeitung und -optimierung. Diese werden im folgenden vorgestellt.

Identifikation von Features

Die erste Phase besteht aus der Extraktion der involvierten Features einer veränderten 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. Für den Fall, dass ein Feature hinter der Direktive `#IFNDEF` identifiziert wurde, wird das Feature mit einem vorangestellten „not“ gespeichert. Es wird somit als eigenständiges Feature, neben seiner nicht-negierten Form, gespeichert. Kombinationen von Features werden in ihrer identifizierten Form gespeichert. 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.

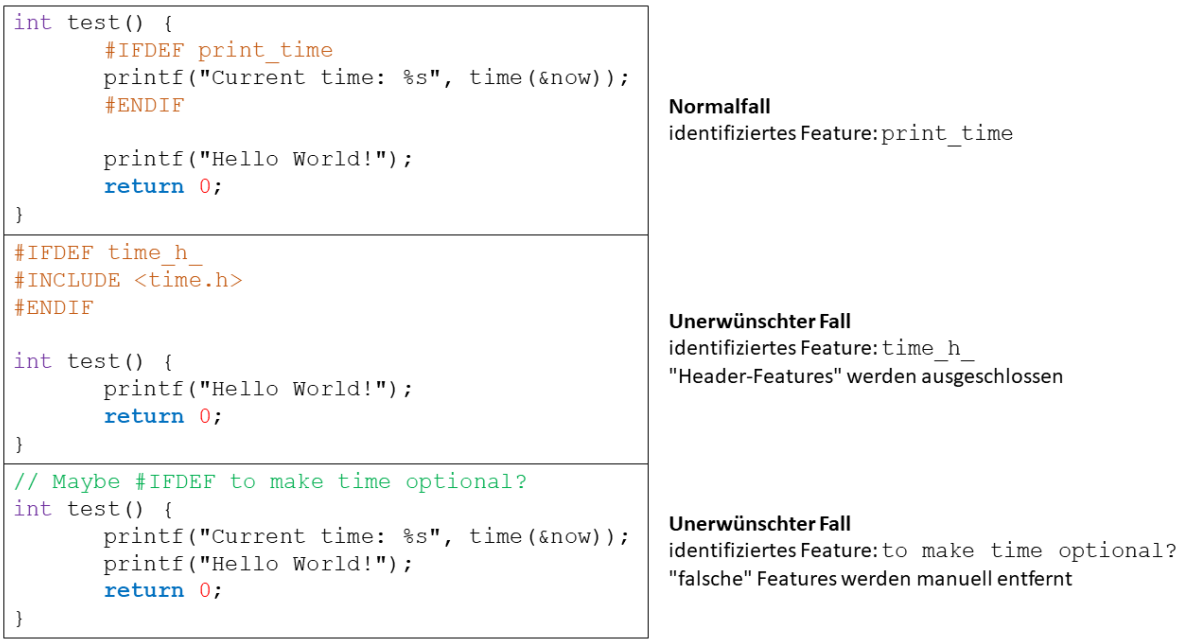


Abbildung 3.2: Normalfall und unerwünschte Fälle bei der Identifizierung der Features

Identifikation von korrektiven Commits

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 [39]. 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 Methoden des Natural Language Processings (NLP). Dabei wurde die Identifizierung auf die jeweils erste Zeile der Commit-Nachrichten beschränkt. Die Ergebnisse wurden in einer weiteren Spalte der MySQL-Tabellen (true = korrektiv, false = nicht korrektiv) gespeichert.

Identifikation fehlereinführender Commits

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 [31]. Dieser Algorithmus erlaubt es fehlereinführende Commits in lokal gespeicherten Software-Repositories zu finden [5]. Dafür setzt er voraus, dass die korrektiven Commits bereits identifiziert wurden, da sie als Eingabemenge des Algorithmus dienen [5]. Die Identifikation der fehlereinführenden Commits 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.

Der SZZ-Algorithmus, der 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

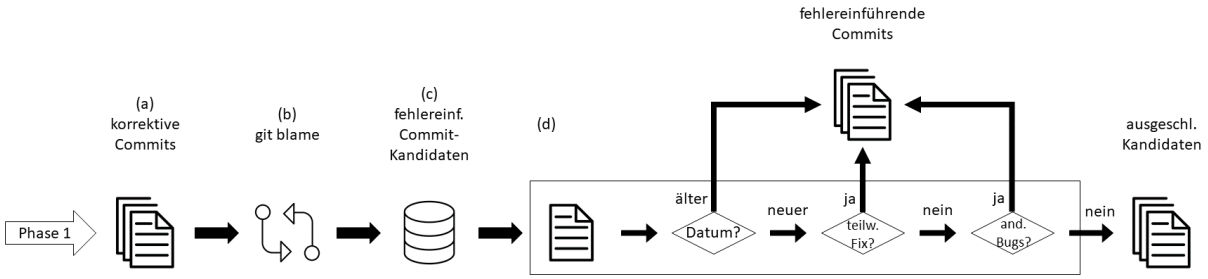


Abbildung 3.3: Ablauf der zweiten Phase des SZZ-Algorithmus (übersetzt, [5])

Tabelle 3.3: Anzahl der korrektiven und fehlereinführenden Commits sowie Anzahl der identifizierten Features je Softwareprojekt

Projekt	#korrektiv	#fehlereinführend	#Features
Blender	7.760	3.776	1.400
Busybox	1.236	802	628
Emacs	4.269	2.532	718
GIMP	1.380	854	204
Gnumeric	1.498	1.191	637
gnuplot	854	1.215	558
Irssi	52	22	9
libxml2	324	88	200
lighttpd	1.078	929	230
MPSolve	151	211	54
Parrot	3.109	3.072	397
Vim	371	696	1.158
xfig	0	0	137

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 der Anzahl der korrektiven und fehlereinführenden Commits sowie der Anzahl der identifizierten Features je Softwareprojekt ist in Tabelle 3.3 aufgeführt. Des weiteren ist eine Übersicht des Schemas der nun vollständigen initialen MySQL-Tabellen (im folgenden Haupttabellen genannt) 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.

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

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

Tabelle 3.4: Übersicht des Schemas der MySQL-Haupttabellen

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

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.

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, dienen Attribute zur Erlernung der Machine-Learning-Klassifikatoren. Im hier vorliegenden Szenario werden sogenannte Metriken als Attribute verwendet. Bei Metriken handelt sich sich um Zahlenwerte, die Eigenschaften eines

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

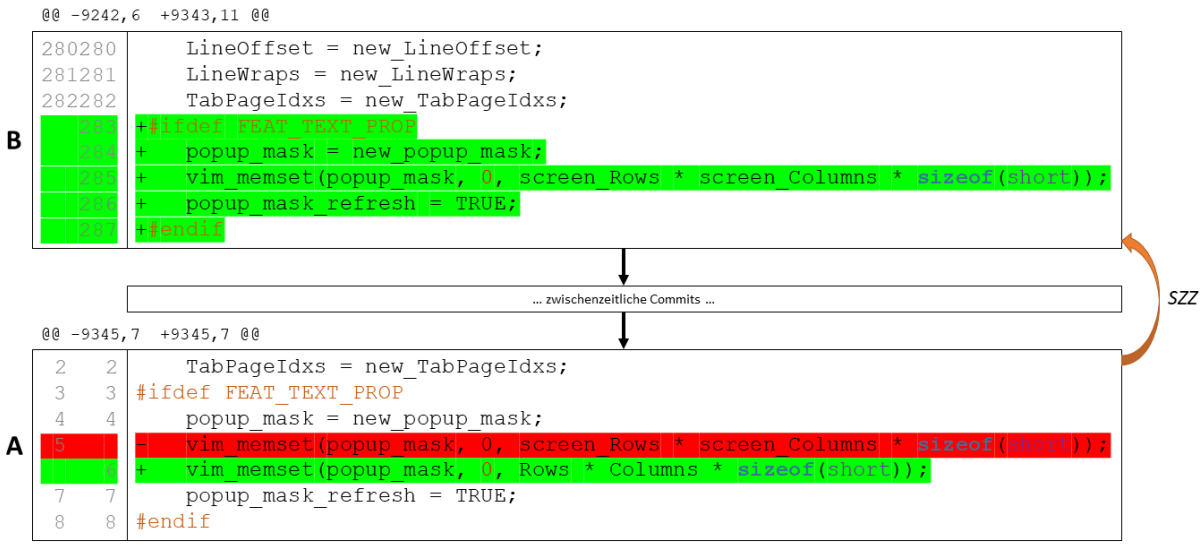


Abbildung 3.4: Reales Beispiel eines Fehlers mit korrektivem (A) und fehlereinführendem (B) Commit

Softwareprojekts quantifizieren. Die Metriken werden im hier vorliegenden Fall in die üblichen Kategorien Codemetriken und Prozessmetriken aufgeteilt und jeweils anhand der vorhandenen Rohdaten der Haupttabellen berechnet [26]. Codemetriken werden genutzt um Eigenschaften von Sourcecode, wie zum Beispiel „Größe“ oder Komplexität, zu messen [26]. Prozessmetriken dienen hingegen zur Messung von Eigenschaften, die anhand von Metadaten aus Software-Repositories erörtert werden können [26]. 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 [26, 24] 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. Dies bedeutet, dass für den Fall, dass ein Feature oder eine Datei in einem Release mehrfach bearbeitet wurden (d.h. es wird in mehreren Commits bearbeitet), der Durchschnittswert der jeweiligen Metriken innerhalb des Releases berechnet und gespeichert wird. Das Verfahren zur Bestimmung des Labels erfolgt anhand des folgenden Musters:

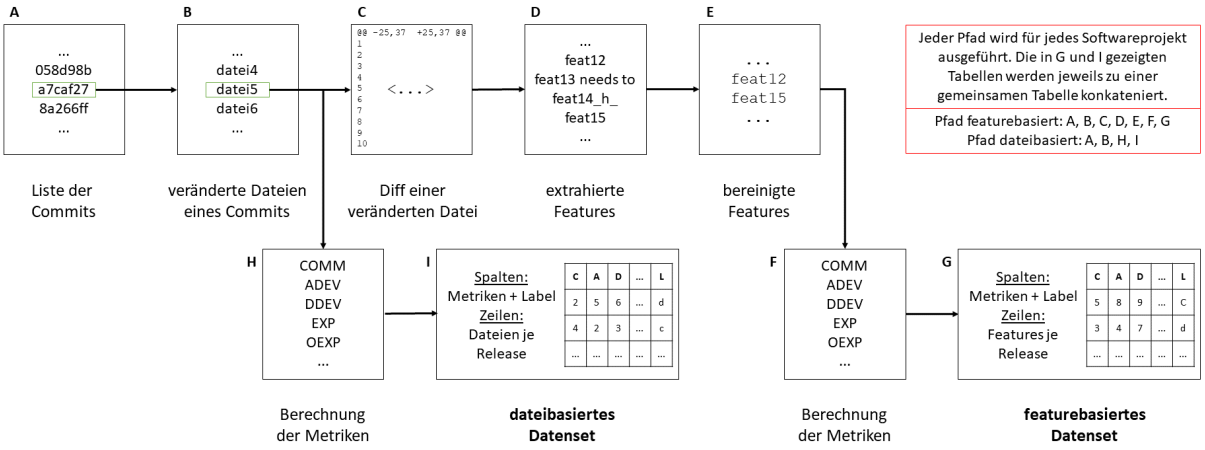


Abbildung 3.5: Visualisierung des Aufbaus und der Unterscheidung der Datensets

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.

Die so erstellten einzelnen Tabellen werden anschließend zu einer gemeinsamen Tabelle konkateniert, sodass eine umfangreiche Auflistung an Metriken inklusive der zugehörigen Labels entsteht. Diese Auflistung gibt an, welche Charakteristika ein Feature beziehungsweise eine Datei aufweisen muss, um als „fehlerfrei“ oder „defekt“ eingestuft zu werden und dient als Lerngrundlage der Klassifikatoren für zukünftige Vorhersagen. 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.

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.

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.	[26, 24]
	Anzahl der aktiven Entwickler (ADEV)	Anzahl der Entwickler, die innerhalb eines Releases das Feature bearbeitet (geändert, gelöscht oder hinzugefügt) haben.	[26, 24]
	eindeutige Entwickleranzahl (DDEV)	kumulierte Anzahl der Entwickler, die innerhalb eines Releases das Feature bearbeitet (geändert, gelöscht oder hinzugefügt) haben.	[26, 24]
	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.	[26, 24]
	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.	[26, 24]
	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.*			

Kapitel 4

Training und Test der Machine-Learning-Klassifikatoren

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 [21]. 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 [21].

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, 24, 28]) 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

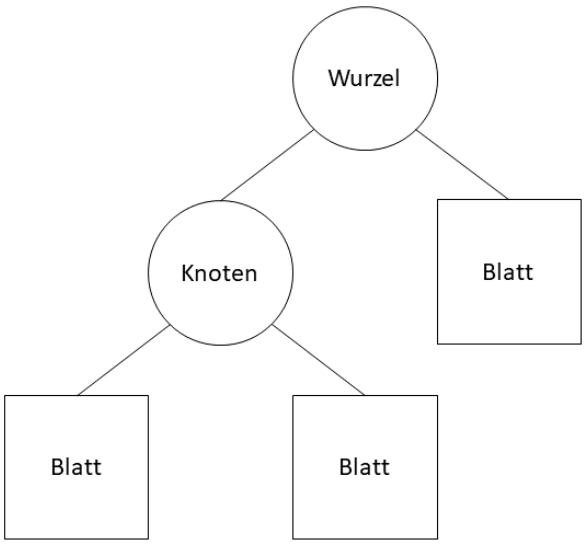


Abbildung 4.1: Grundsätzlicher Aufbau eines Decision Trees

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 [32]. Decision Trees sind gerichtete und verwurzelte Bäume, die als rekursive Partition der Eingabemenge des Datensets aufgebaut werden [29]. Den Ursprung des Baumes bildet die Wurzel, welche keine eingehenden Kanten besitzt - alle weiteren Knoten besitzen jedoch eine eingehende Kante [29]. Diese Knoten teilen wiederum die Eingabemenge anhand einer vorgegebenen Funktion in zwei oder mehr Unterräume der Menge auf [29]. Meist geschieht dies anhand eines Attributs, sodass die Eingabemenge anhand der Werte des einzelnen Attributs geteilt wird [29]. 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 [29]. Der grundlegende Aufbau eines Decision Trees ist in Abbildung 4.1 dargestellt.

Eine Besonderheit von Decision Trees stellen sogenannte Random Forests dar. Diese beschreiben eine Lernmethode von Klassifikatoren, bei der mehrere einzelne Decision Trees gleichzeitig erzeugt werden und deren Ergebnisse anschließen aggregiert werden [1]. Dazu erhält jeder Decision Tree eine Teilmenge der Eingabemenge des Datensets [1]. Random Forests eigenen sich besonders zur Anwendung, wenn viele Attribute im Datenset vorhanden sind [1].

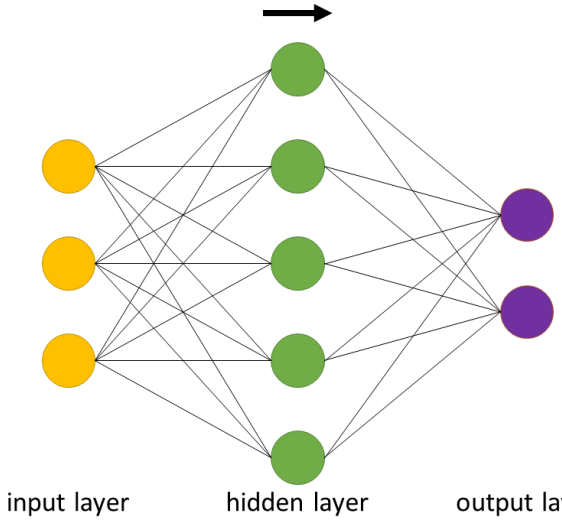


Abbildung 4.2: Grundsätzlicher Aufbau eines KNN mit drei Input-Layer-Neuronen, fünf Hidden-Layer-Neuronen und zwei Output-Layer-Neuronen

k-Nearest-Neighbors

Ein k-Nearest-Neighbor-Klassifikator (deutsch: k-nächste-Nachbarn) basiert auf zwei Konzepten [38]. Das erste Konzept basiert auf der Abstandsmessung zwischen den Werten der zu klassifizierenden Datenmenge und den Werten der Attribute des Datensets [38]. 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 [38]. 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 [20]. 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 [20]. In den Hidden-Layers werden dann Entscheidungen anhand der Beziehungen zwischen den Eingabedaten und Zielklassen sowie die den Verbindungen zuvor zugewiesenen Gewichtsfaktoren getroffen [15, 20]. 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].

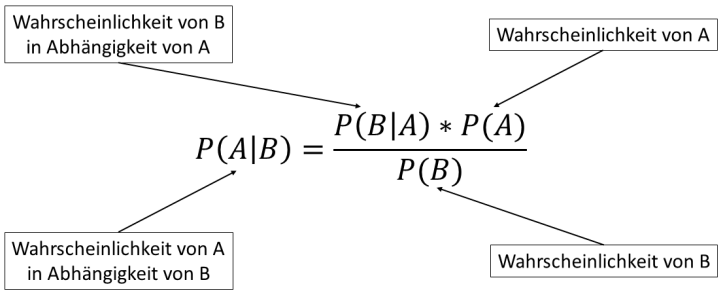


Abbildung 4.3: Satz von Bayes als Grundlage des Naïve-Bayes-Klassifikators

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 [27]. Diese Annahme wird zwar in der realen Verwendung des Klassifikators häufig verletzt, dennoch erzielt er in der Regel eine hohe Performanz [27]. Der Klassifikator gilt als effizient, robust, schnell und einfach implementierbar [27]. Die zur Durchführung einer Klassifikation mittels Naïve Bayes benötigte Formel nach Thomas Bayes ist in Abbildung 4.3 samt Erläuterung der einzelnen Faktoren aufgeführt.

Es existiert zudem eine Mehrzahl an Varianten des Naïve-Bayes-Klassifikators, die verschiedene Annahmen über die Verteilung der Attribute der Eingabemenge machen. Beispiele dafür 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 [22]. 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 kategorialen oder numerischen Eingabedaten und kategorischen Zielklassen [22]. 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 [34]. 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

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		

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.

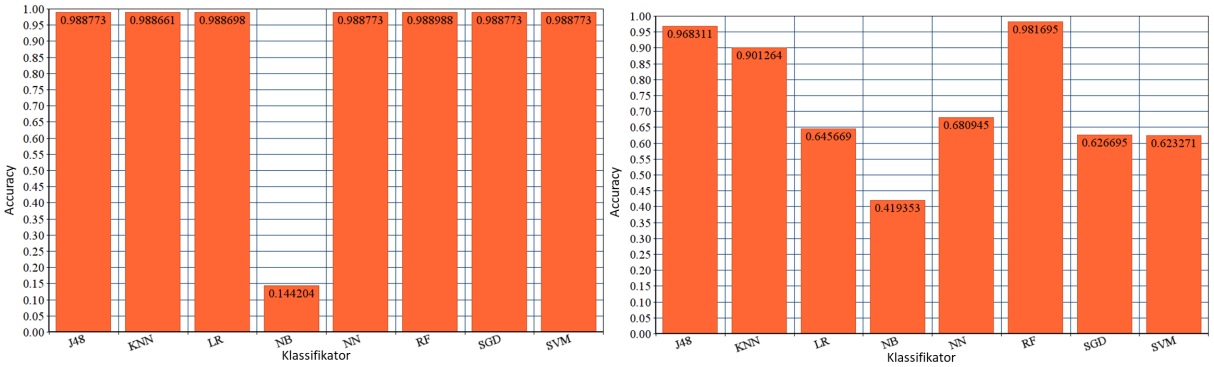
RQ2: WELCHE MACHINE-LEARNING-KLASSIFIKATOREN KOMMEN FÜR DIE GEGEBENE AUFGABE IN FRAGE?

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 [32].

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.

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



(a) vorher **(b)** nachher

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

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. Die erwähnte Abbildung zeigt pro Klassifikator die höchste Accuracy die mithilfe der fünf Split-Ratios 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.

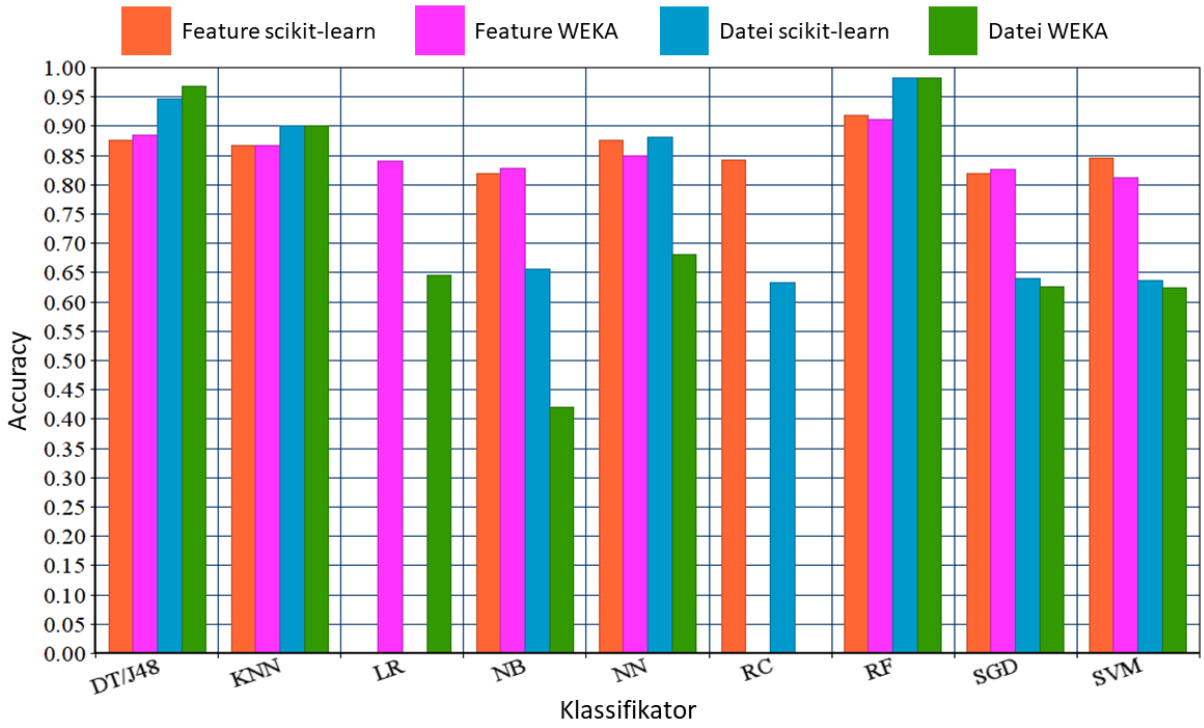


Abbildung 4.5: Vergleich der Klassifikatoren und Werkzeuge im Hinblick auf ihre Accuracies

Tabelle 4.3: Übersicht der Konfigurationen des featurebasierten Datensets

Klassifikator	Konfiguration scikit-learn	Konfiguration WEKA
DT / J48	• max_features = „sqrt“ • Split-Ratio: 85:15 • Features: alle außer COMM, DDEV, MODD	• Split-Ratio: 70:30 • Features: alle außer DDEV, OEXP, MODS, REML
KNN	• k = 1 • Split-Ratio: 75:25	• k = 1 • Split-Ratio: 65:35
LR		• Split-Ratio: 65:35
NB	• Multinomial-NB • Split-Ratio: 75:25	• Split-Ratio: 70:30 • Features: COMM,ADEV, EXP, OEXP, ADDL
NN	• hidden_layer_sizes = (13,13,13) • StandardScaler • max_iter = 500 • Split-Ratio: 85:15	• Hidden-Layer = (13,13,13) • Split-Ratio: 80:20
RC	• StandardScaler • Split-Ratio: 75:25 • Features: alle außer REML	
RF	• n_estimators = 200 • max_features = „sqrt“ • Split-Ratio: 75:25 • Features: alle außer COMM, DDEV, MODD, REML	• Iterationen: 200 • Split-Ratio: 80:20 • Features: alle außer ADEV, OEXP
SGD	• loss = „log“ • penality = „elasticnet“ • Split-Ratio: 70:30 • Features: EXP, OEXP, ADDL, REML	• Split-Ratio: 70:30 • Features: alle außer REML
SVM	• LinearSVC • max_iter = 20000 • StandardScaler • Split-Ratio: 65:35 • Features: alle außer DDEV, EXP, REML	• Split-Ratio: 65:35 • 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: 75:25 • Features: ADDL	• Split-Ratio: 75:25
KNN	• k = 1 • Split-Ratio: 80:20	• k = 1 • Split-Ratio: 85:15
LR		• Split-Ratio: 65:35
NB	• Bernoulli-NB • Split-Ratio: 85:15	• Split-Ratio: 75:15 • Features: DDEV, MODD, MODS, CYCO, ADDL
NN	• hidden_layer_sizes = (13,13,13) • max_ter = 500 • Split-Ratio: 75:25	• Hidden-Layer = (13,13,13) • Split-Ratio: 75:25
RC	• StandardScaler • Split-Ratio: 80:20 • Features: MODD, MODS, NLOC	
RF	• n_stimators = 200 • max_features = „sqrt“ • Split-Ratio: 80:20	• Iterationen: 200 • Split-Ratio: 85:15
SGD	• loss = „log“ • penality = „elasticnet“ • Split-Ratio: 65:35 • Features: alle außer DDEV	• Split-Ratio: 85:15
SVM	• LinearSVC • max_iter = 20000 • StandardScaler • Split-Ratio: 65:35	• Split-Ratio: 85:15

Kapitel 5

Evaluation

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

Identifikation von Features

Die grundsätzliche Frage, die im Rahmen der Identifikation der Features aufkam, war: „Was wird als Feature gezählt?“. Wie bereits in Abschnitt 3.2 erwähnt wurde, barg die Identifikation der Features einige Herausforderungen. So gestaltete sich die erste Herausforderung in der Ausfilterung von „Header-Features“, die in einigen Programmierparadigmen verwendet werden, um Header-Dateien im Sourcecode einzubinden. Diese Header-Features erzeugen jedoch keine Variabilität im Code, sodass sie unerwünscht sind. Identifizierbar waren die meisten dieser Header-Features an ihren vergebenen Namen, welche ein `_h_` aufwiesen. Auf diesem Weg konnten sie mittels regulärer Ausdrücke schnell ermittelt und ausgefiltert werden. Es besteht jedoch auch die Möglichkeit, dass in einigen Softwareprojekten die Header-Features nicht explizit durch ihre Namensgebung kenntlich gemacht werden. Sie lassen sich somit nur schwer identifizieren, beispielsweise durch eine manuelle Sichtung der Kontexte der Features im Sourcecode. Dies wäre jedoch im vorliegenden Fall aufgrund der großen Menge an Features sehr zeitaufwändig und wurde aus diesem Grund nicht durchgeführt. Die Entfernung der erkennbaren Header-Features zeigte, dass ein erheblicher Teil der zuvor identifizierten Features unerwünscht war. Diese Methode erwies sich somit als effektiv.

oder eine automatisierte Analyse des Kontexts des Features im Sourcecode > parsing.

falsche features, was ist wirklich feature?, header, keine tools, parsing

Einbindung des Bezugs zu Features

features unterliegen den Dateien, ... Keine in-depth Analyse des Codes

Heuristik zur Erkennung von korrektiven Commits

manuell stichprobe

Unpräzisiertheit des SZZ-Algorithmus

Eine Limitation, die im Laufe der Erstellung des Datensets durch Literaturrecherchen festgestellt wurde, bezieht sich auf den SZZ-Algorithmus. Dieser wurde genutzt, um fehlereinführende Commits auf Basis der Commit-Hashes der korrektiven Commits zu identifizieren. Analysen des Algorithmus ergaben, dass momentan verfügbare Implementationen und somit auch die des verwendeten Python-Tools PyDriller lediglich etwa 69% der tatsächlich existierenden fehlereinführenden Commits identifizieren können [37]. Darüber hinaus wurde herausgefunden, dass etwa 64% der identifizierten Commits falsch ermittelt wurden [37]. Der Algorithmus gilt somit als unpräzise [37]. Die Begründung dafür lautet wie folgt:

> The reason is that the implicit assumptions of the SZZ algorithm are violated by the insufficient file coverage and statement direct coverage between bug-inducing and bug-fixing commits. - [37]
>
> *Der Grund dafür ist, dass die impliziten Annahmen des SZZ-Algorithmus durch die unzureichende „file coverage“ und die unzureichende „statement direct coverage“ der Aussage zwischen fehlereinführenden und korrektiven Commits verletzt werden.*

Ferner stellten die Autoren der Studie in eigenen durchgeführten Tests fest, dass die Ergebnisse von acht von zehn früheren Studien durch den unpräzisen Algorithmus signifikant beeinflusst wurden [37]. Dies kann somit auch auf diese Arbeit zutreffen. Es existiert jedoch momentan keine alternative Methode zur Identifizierung von fehlereinführenden Commits. Sollte eine neue Methode oder eine verbesserte Version des SZZ-Algorithmus veröffentlicht werden, so würde es sich anbieten, die Hauptschritte dieser Arbeit unter Berücksichtigung der neuen Methode zu wiederholen, um sie mit den hier vorliegenden Ergebnissen zu vergleichen, um die Einflüsse des SZZ-Algorithmus herauszustellen.

Vorhersageziel

commits, releases? **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 [30]. Die meisten dieser Metriken lassen sich anhand von Werten einer sogenannten Konfusionsmatrix berechnen und messen allesamt die Performanz der Vorhersagen von Klassifikatoren unter verschiedenen Betrachtungsweisen. 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 [30]. 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.

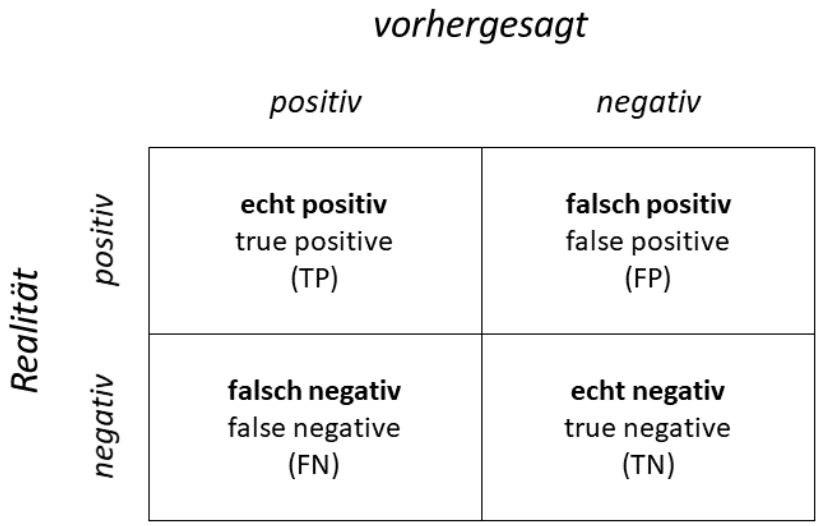


Abbildung 5.1: allgemeine Konfusionsmatrix

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 [30]. 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 sämtlicher als positiv gewerteter 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 [30]. 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 [30]. 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) [30, 19]. 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 Datenpunkt der positiven Klasse zugeordnet, wenn die Wahrscheinlichkeit einen Schwellenwert von 0,5 übersteigt - Datenpunkte, die diesen Schwellenwert unterschreiten werden wiederum der negativen Klasse zugeordnet [13]. Wird der Schwellenwert erhöht, so werden weniger Datenpunkte der positiven Klasse zugeordnet, wohingegen im Falle einer Absenkung des Schwellenwertes mehr Datenpunkte 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 [19]. Je höher dieser Wert ist (1,0 ist das Maximum), desto besser trifft der Klassifikator korrekte Vorhersagen [19].

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 sowie innerhalb sowie zwischen den Datensets verglichen und interpretiert.

RQ3a: WELCHE MITEINANDER VERGLEICHBAREN MERKMALE BESITZEN DIE KLASSIFIKATOREN?

Die Merkmale zum Vergleich stellen Evaluationsmetriken dar, welche auf Basis der Ergebnisse des Tests der Klassifikatoren errechnet werden und die Performanz der Vorhersagen auf verschiedene Weisen messen. Welche Metriken 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 von Konfusionsmatrizen berechnet werden. Die betrachteten Metriken lauten: Accuracy, TP-Rate, Precision und F-Score. Ebenfalls hinzugezogen werden die jeweiligen ROC-Kurven der Klassifikatoren inklusive der ROC-Bereiche. Diese Metriken stellen einen Standard für die Messung der Performanz der Vorhersagen von Klassifikatoren dar.

5.2.2 Ergebnisse und Diskussion

Konfusionsmatrizen

Als Basis der Ergebnisse der Evaluation der Klassifikatoren anhand der zuvor vorgestellten Metriken dienen die Konfusionsmatrizen. Die Matrizen des featurebasierten Datensets sind in

Tabelle 5.1: Konfusionsmatrizen des featurebasierten Datensets

		scikit-learn			WEKA		
	Ermittelt ->	**Fehlerfrei**	**Defekt**	**Total**	**Fehlerfrei**	**Defekt**	**Total**
DT	Realität fehlerfrei	1.627	122	1.749	3.285	188	3.473
	Realität defekt	146	268	414	388	463	851
	Total	1.773	390	2.163	3.673	651	4.324
KNN	Realität fehlerfrei	2.691	241	2.932	3.781	276	4.057
	Realität defekt	255	417	672	392	596	988
	Total	2.946	658	3.604	4.173	875	5.045
LR	Realität fehlerfrei				3.993	64	4.057
	Realität defekt				737	251	988
	Total				4.730	315	5.045
NB	Realität fehlerfrei	2.301	608	2.909	3.411	62	3.473
	Realität defekt	648	47	695	643	208	851
	Total	2.949	655	3.604	4.054	270	4.324
NN	Realität fehlerfrei	1.692	99	1.791	2.175	120	2.395
	Realität defekt	159	213	372	315	273	588
	Total	1.851	312	2.163	2.490	393	2.883
RC	Realität fehlerfrei	2.926	8	2.934			
	Realität defekt	603	67	670			
	Total	3.529	75	3.604			
RF	Realität fehlerfrei	2.851	99	2.950	2.234	61	2.295
	Realität defekt	213	441	654	195	393	588
	Total	3.064	540	3.604	2.429	454	2.883
SGD	Realität fehlerfrei	3.336	191	3.527	3.453	20	3.473
	Realität defekt	613	185	798	720	131	851
	Total	3.949	376	4.325	4.173	151	4.324
SVM	Realität fehlerfrei	4.039	58	4.097	4.055	2	4.057
	Realität defekt	722	226	948	945	43	988
	Total	4.761	284	5.045	5.000	45	5.045

Tabelle 5.1 aufgeführt, die Matrizen des dateibasierten Datensets können in Tabelle 5.2 gefunden werden. Beide Tabellen sind in den Spalten in die beiden verwendeten Tools scikit-learn und WEKA unterteilt. Ferner bilden die Spalten die von den Klassifikatoren vorhergesagten Label ab, wohingegen die Zeilen die „ground truth“, also die Realität, abbilden. Außerdem werden die ermittelten Werte der jeweiligen Klassen in den Spalten „Total“zusammengezählt. Die für die Klassifikatoren verwendeten Abkürzungen können Tabelle 4.2 entnommen werden. Anzumerken ist, dass sich die Gesamtsummen der Werte aufgrund der jeweiligen Konfigurationen der Klassifikatoren (insbesondere durch die Wahl der Split-Ratios) unterscheiden.

Accuracies

Die erste Metrik, die verglichen wird, ist die Accuracy. Diese gibt die Treffergenauigkeit der Klassifikatoren an. Dargestellt werden die Ergebnisse zum besseren Vergleich als Balkendiagramme, aufgeteilt in die jeweiligen Datenset und verwendeten Tools. Das Diagramm des feturebasierten Datensets ist in Abbildung 5.3 dargestellt. Der Vergleich des dateibasierten Datensets ist wiederum in Abbildung 5.4 abgebildet. Die konkreten Zahlenwerte können im Anhang eingesehen werden.

Der Vergleich der Accuracies des featurebasierten Datensets in Abbildung 5.3 zeigt, dass die Accuracies, bis auf eine Ausnahme, stets über 80% liegen und sich zwischen den verwendeten Tools nur marginal unterscheiden. Mit einer Accuracy von jeweils über 85% liegen die Entscheidungsbaum-basierten Klassifikatoren DT (scikit-learn), J48 (WEKA) und RF im Sinne der Performanz auf dem höchsten Niveau. Die weiteren Klassifikatoren reihen sich mit Werten

Tabelle 5.2: Konfusionsmatrizen des dateibasierten Datensets

		scikit-learn			WEKA		
	Ermittelt ->	**Fehlerfrei**	**Defekt**	**Total**	**Fehlerfrei**	**Defekt**	**Total**
J48	Realität fehlerfrei	21.704	121	21.825	21.320	464	21.784
	Realität defekt	1.905	12.782	14.687	693	14.034	14.727
	Total	23.609	12.903	36.512	22.013	14.498	36.511
KNN	Realität fehlerfrei	15.519	1.853	17.372	12.090	1.026	13.116
	Realität defekt	1.026	10.812	11.838	1.137	7.654	8.791
	Total	16.545	12.665	29.210	13.227	8.680	21.907
LR	Realität fehlerfrei				28.023	2.570	30.593
	Realität defekt				15.542	4.981	20.523
	Total				43.565	7.551	51.116
NB	Realität fehlerfrei	8.667	4.368	13.035	18.982	2.802	21.784
	Realität defekt	3.212	5.660	8.872	10.369	4.358	14.727
	Total	11.879	10.028	21.907	29.351	7.160	36.511
NN	Realität fehlerfrei	20.904	744	21.648	18.754	3.030	21.784
	Realität defekt	3.522	11.342	14.864	8.619	6.108	14.727
	Total	24.426	12.086	36.512	27.373	9.138	36.511
RC	Realität fehlerfrei~	15.874	1.438	17.312			
	Realität defekt	9.152	2.746	11.898			
	Total	25.026	4.184	29.210			
RF	Realität fehlerfrei	17.189	212	17.401	12.990	126	13.116
	Realität defekt	374	11.435	11.809	275	8.516	8.791
	Total	17.563	11.647	29.210	13.265	8.642	21.907
SGD	Realität fehlerfrei	20.600	10.037	30.637	12.698	418	13.116
	Realität defekt	11.187	9.293	20.480	7.760	1.031	8.791
	Total	31.787	19.330	51.117	20.458	1.449	21.907
SVM	Realität fehlerfrei	28.190	2.237	30.427	12.679	437	13.116
	Realität defekt	16.479	4.211	20.690	7.816	975	8.791
	Total	44.669	6.448	51.117	20.495	1.412	21.907

zwischen 80% und 87% ein. Eine Ausnahme bildet der Naïve-Bayes-Klassifikator von scikit-learn, welcher mit 65% die niedrigste Accuracy aufweist.

Der Blick auf die Accuracies in Abbildung 5.4 des dateibasierten Datensets zeigt ein wesentlich differenzierteres Bild mit Werten zwischen 58% und 98%. Es ist jedoch auch wieder auffällig, dass die Entscheidungsbaum-basierten Klassifikatoren die höchsten Accuracies mit jeweils über 94% aufweisen. Die geringsten Accuracies weisen die Klassifikatoren SGD und SVM mit jeweils unter 63% auf. Die Accuracies der weiteren Klassifikatoren liegen zwischen den zuvor genannten Werten. Die Werte zwischen den Tools unterscheiden sich erneut nur marginal. Lediglich die NN-Klassifikatoren weisen eine Differenz von etwa 18% auf.

Eine Erklärung für die unterschiedlichen Streuungen der Ergebnisse zwischen den Datensets kann durch ihre jeweilige Größe begründet sein. Das dateibasierte Datenset weist eine etwa 10-fach höhere Anzahl an Einträgen auf. Es kann somit sein, dass bestimmte Klassifikatoren unter der Eingabe von größeren Datensets schlechter beziehungsweise besser performen.

Weitere Evaluationsmetriken

Dieser Abschnitt stellt die Ergebnisse der weiteren Evaluationsmetriken TP-Rate / Recall, FP-Rate, Precision und F-Score vor. Diese können in Tabelle 5.3 für das featurebasierte Datenset und in Tabelle 5.4 für das dateibasierte Datenset eingesehen werden. Aufgeteilt werden die Ergebnisse nach verwendetem Tool sowie nach den Werten der Zielklasse „fehlerfrei“ und „defekt“. Zudem wird das gewichtete Mittel angegeben. Dabei handelt es sich um den Mittelwert

Tabelle 5.3: Ergebnisse der Evaluationsmetriken des featurebasierten Datensets

		scikit-learn			WEKA		
		Fehlerfrei	**Defekt**	**gew. Mittel**	**Fehlerfrei**	**Defekt**	**gew. Mittel**
DT / J48	TP-Rate	0,93	0,65	0,88	0,95	0,54	0,87
	FP-Rate	0,65	0,07	0,18	0,46	0,05	0,38
	Precision	0,92	0,69	0,87	0,89	0,71	0,86
	F-Score	0,92	0,67	0,87	0,92	0,62	0,86
KNN	TP-Rate	0,92	0,62	0,86	0,93	0,60	0,87
	FP-Rate	0,62	0,08	0,18	0,40	0,07	0,33
	Precision	0,91	0,63	0,86	0,91	0,63	0,86
	F-Score	0,92	0,63	0,86	0,92	0,64	0,86
LR	TP-Rate				0,98	0,25	0,84
	FP-Rate				0,75	0,02	0,60
	Precision				0,84	0,80	0,83
	F-Score				0,91	0,39	0,81
NB	TP-Rate	0,79	0,07	0,65	0,98	0,24	0,84
	FP-Rate	0,07	0,21	0,18	0,76	0,02	0,61
	Precision	0,78	0,07	0,64	0,81	0,77	0,83
	F-Score	0,79	0,07	0,65	0,91	0,37	0,80
NN	TP-Rate	0,94	0,57	0,88	0,95	0,46	0,85
	FP-Rate	0,57	0,06	0,07	0,54	0,05	0,44
	Precision	0,91	0,68	0,87	0,87	0,70	0,84
	F-Score	0,93	0,92	0,88	0,91	0,56	0,84
RC	TP-Rate	1,00	0,10	0,83			
	FP-Rate	0,10	0,00	0,02			
	Precision	0,83	0,89	0,84			
	F-Score	0,91	0,18	0,77			
RF	TP-Rate	0,97	0,67	0,91	0,97	0,67	0,91
	FP-Rate	0,67	0,03	0,15	0,33	0,03	0,27
	Precision	0,93	0,82	0,91	0,92	0,87	0,91
	F-Score	0,95	0,74	0,91	0,95	0,75	0,91
SGD	TP-Rate	0,95	0,23	0,81	0,99	0,15	0,93
	FP-Rate	0,23	0,05	0,09	0,85	0,01	0,68
	Precision	0,84	0,49	0,78	0,83	0,87	0,84
	F-Score	0,89	0,32	0,81	0,90	0,26	0,78
SVM	TP-Rate	0,99	0,24	0,85	1,00	0,04	0,81
	FP-Rate	0,24	0,01	0,06	0,96	0,00	0,77
	Precision	0,85	0,80	0,84	0,81	0,96	0,84
	F-Score	0,91	0,37	0,81	0,90	0,08	0,74

der Ergebnisse beider Werte der Zielklasse unter Einbezug der Anzahl der korrekten Vorhersagen je Wert. Es zeigt somit die Performanz aggregiert für beide Werte an und liegt im Idealfall bei 1,00. Die vollständigen Tabellen der Ergebnisse der Evaluation, inklusive einer weiteren Metrik, können im Anhang gefunden werden.

Betrachtet man die Ergebnisse der Metriken des featurebasierten Datensets in Tabelle 5.3, so ergibt sich eine breite Streuung der Werte. Jeder Klassifikator zeigt durch seine Ergebnisse seine individuellen „Stärken“ und „Schwächen“. In der Gesamtheit der Kombination der Werte betrachtet,

TOP: RF, KNN, DT (eingeschränkt) Mittel: LR, NN **NÄHER BETRACHTEN**

ROC-Kurven und ROC-Bereiche

Die Interpretation der ROC-Kurven und ROC-Bereiche erfolgt anhand des in Abschnitt 5.2.1 vorgestellten Schemas. Die ROC-Kurven samt der Werte der ROC-Bereiche (repräsentiert durch

Tabelle 5.4: Ergebnisse der Evaluationsmetriken des dateibasierten Datensets

		scikit-learn			WEKA		
		fehlerfrei	**defekt**	**gew. Mittel**	**fehlerfrei**	**defekt**	**gew. Mittel**
DT / J48	TP-Rate	0,99	0,87	0,94	0,98	0,95	0,97
	FP-Rate	0,87	0,01	0,35	0,05	0,02	0,03
	Precision	0,92	0,99	0,95	0,97	0,97	0,97
	F-Score	0,96	0,93	0,94	0,97	0,96	0,97
KNN	TP-Rate	0,89	0,91	0,90	0,92	0,87	0,90
	FP-Rate	0,91	0,11	0,43	0,13	0,08	0,11
	Precision	0,94	0,85	0,90	0,91	0,88	0,90
	F-Score	0,92	0,88	0,90	0,92	0,88	0,90
LR	TP-Rate				0,92	0,24	0,65
	FP-Rate				0,76	0,08	0,49
	Precision				0,64	0,66	0,65
	F-Score				0,76	0,36	0,60
NB	TP-Rate	0,66	0,64	0,65	0,87	0,30	0,64
	FP-Rate	0,64	0,34	0,46	0,70	0,13	0,47
	Precision	0,73	0,56	0,66	0,65	0,61	0,63
	F-Score	0,70	0,60	0,66	0,74	0,40	0,60
NN	TP-Rate	0,97	0,76	0,88	0,86	0,41	0,68
	FP-Rate	0,76	0,03	0,33	0,59	0,14	0,41
	Precision	0,86	0,94	0,89	0,69	0,67	0,68
	F-Score	0,91	0,84	0,88	0,76	0,51	0,66
RC	TP-Rate	0,92	0,23	0,64			
	FP-Rate	0,23	0,08	0,14			
	Precision	0,63	0,66	0,64			
	F-Score	0,75	0,34	0,58			
RF	TP-Rate	0,99	0,97	0,98	0,99	0,97	0,98
	FP-Rate	0,97	0,01	0,40	0,03	0,01	0,02
	Precision	0,98	0,98	0,98	0,98	0,99	0,98
	F-Score	0,98	0,98	0,98	0,99	0,98	0,98
SGD	TP-Rate	0,67	0,45	0,58	0,97	0,12	0,63
	FP-Rate	0,45	0,33	0,38	0,88	0,03	0,54
	Precision	0,65	0,48	0,58	0,62	0,71	0,66
	F-Score	0,66	0,47	0,58	0,76	0,20	0,53
SVM	TP-Rate	0,93	0,20	0,63	0,97	0,11	0,62
	FP-Rate	0,20	0,07	0,12	0,89	0,03	0,55
	Precision	0,63	0,65	0,64	0,62	0,69	0,65
	F-Score	0,75	0,31	0,57	0,75	0,19	0,53

„AUC“) sind in Abbildung 5.5 (featurebasiertes Datenset) und Abbildung 5.6 dargestellt. Zu sehen sind jeweils die von den Tools scikit-learn und WEKA ausgegebenen und unveränderten Plots. Die in den Plots des Tools WEKA dargestellten Farbverläufe der Kurven verdeutlichen keine für diesen Zweck relevanten Informationen und können somit ignoriert werden.

Die ROC-Kurven des featurebasierten Datensets in Abbildung 5.5 zeigen, dass erneut der Entscheidungsbaum-basierte Klassifikator RF beider Tools nahezu den Idealfall der Performanz widerspiegeln. Dies bezeugen auch die ROC-Bereiche von 0,95. Die Klassifikatoren SGD und SVM (nur WEKA) zeigen hingegen den nahezu schlechtesten Fall mit winkelhalbierenden Kurven und ROC-Bereichen nahe des unerwünschten Wertes von 0,50. Dies zeigt, dass die Klassifikatoren die Werte der Zielklasse nicht unterscheiden können und somit „raten“ statt präzise vorherzusagen. Eine weitere Besonderheit stellt die ROC-Kurve des NB-Klassifikators von scikit-learn dar. Sie verläuft konvex und besitzt einen ROC-Bereich von 0,41, welcher unter dem unerwünschten Wert liegt. Dies bedeutet, dass der Klassifikator inverse Vorhersagen trifft. Im vorliegenden Fall bedeutet dies, dass statt dem Label „fehlerfrei“ das inverse, also gegenteilige, Label „defekt“ vorhersagt wird. Für den Fall eines ROC-Bereiches von 0,00 können somit die

Label einfach invertiert werden. Die weiteren Klassifikatoren besitzen ROC-Kurven, welche den Normalfall mit einer durchschnittlichen oder überdurchschnittlichen Performanz darstellen. Der Vergleich zwischen den verwendeten Tools zeigt, dass sich die Kurven und die Werte der Bereiche in den überwiegenden Fällen ähneln. Lediglich die Klassifikatoren NB und SVM weisen signifikante Unterschiede auf. Begründet werden kann dies durch verschiedene Implementierungsansätze der Klassifikationsalgorithmen in den Tools. So ist zum Beispiel nicht ersichtlich, welche Variation des NB-Klassifikators von WEKA verwendet wird. Diese kann somit gegebenenfalls nicht für die zugrundeliegenden Datensets optimiert sein und zu ungenauen oder falschen Ergebnissen führen.

Der Blick auf die ROC-Kurven des dateibasierten Datensets in Abbildung 5.6 zeigt, dass die Performanzen, gegenüber des featurebasierten Datensets, für die meisten Klassifikatoren höher ausfallen. Insbesondere die Klassifikatoren, welche in der ersten und zweiten Zeile der Abbildung dargestellt sind, mit Ausnahme des NN-Klassifikators von WEKA, besitzen ROC-Kurven, die nahezu oder vollständig dem Idealfall entsprechen. Die jeweiligen ROC-Bereiche mit Werten von über 0,90 bekräftigen dies. Einbußen in der Performanz können, mit Ausnahme des NB-Klassifikators von scikit-learn, bei den Klassifikatoren der dritten und vierten Zeile festgestellt werden. Keiner der Klassifikatoren konnte einen ROC-Bereich von 0,70 erreichen. Ihre Leistung ist somit auf ein unterdurchschnittliches Niveau gesunden. Die WEKA-Klassifikatoren SGD und SVM zeigen weiterhin den unerwünschten Fall.

Auch die Betrachtung der ROC-Kurven mit ihren zugehörigen Bereichen zeigte erneut, dass die RF-Klassifikatoren die beste Wahl für den in dieser Arbeit vorliegenden Klassifikationsfall darstellen. Sie zeigten in allen Gesichtspunkten der Evaluation die höchste Performanz.

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 [18]. Die Grundlage der Berechnungen bildeten die aus den Software-Repositories mittels PyDriller erhaltenen Daten. Zur Berechnung der `REVISIONS`-Metrik wurden die Commit-Nachrichten, analog zur Identifikation der fehlerbehebenden Commits, auf das Vorhandensein des Schlagwortes „refactor“ analysiert. Zur Berechnung der Metriken `AGE` und `WEIGHTED_AGE` wurde zudem für jeden Commit das zugehörige Datum der Ausführung abgerufen. Die Berechnung erfolgte entweder direkt mittels SQL-Abrufen oder mithilfe von Python-Skripten. Aufgrund der Unbalanciertheit des Datensets wurde der SMOTE-Algorithmus mit einem Wert von 3000 angewendet.

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

Tabelle 5.5: Übersicht der berechneten Metriken nach [18]

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

dieses Datenset als „vorhandenes Datenset“ bezeichnet, welches ebenfalls mit dem SMOTE-Algorithmus mit einem Wert von 300 erweitert wurde.

RQ3d: WIE LASSEN SICH DIE KLASSIFIKATOREN MIT WEITEREN VORHERSAGETECHNIKEN, DIE KEINE FEATURES NUTZEN, VERGLEICHEN?

Es wurde auf eine Methode zur Erstellung eines dateibasierten Datensets aus der wissenschaftlichen Literatur zurückgegriffen [18]. 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.

Konfusionsmatrizen

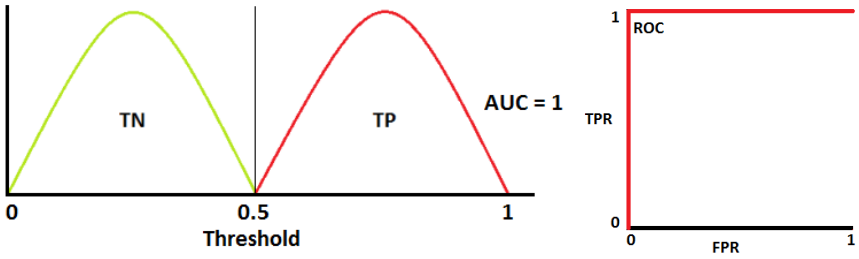
Accuracies

Weitere Evaluationsmetriken

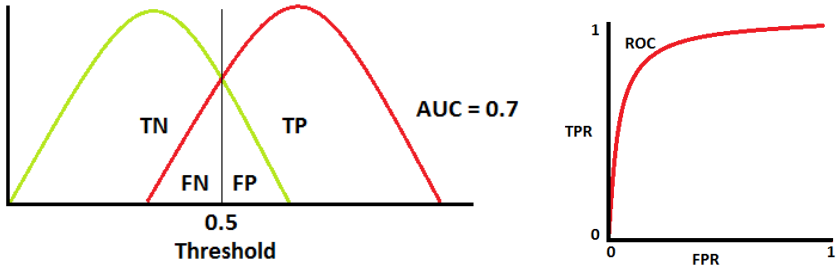
ROC-Kurven und ROC-Bereiche

Tabelle 5.6: Konfusionsmatrizen des nicht-featurebasierten Vergleichs

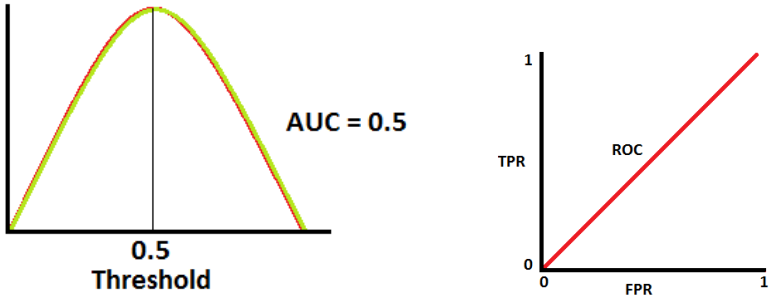
		Datenset nach [18]			vorhandenes Datenset		
	Ermittelt ->	**Fehlerfrei**	**Defekt**	**Total**	**Fehlerfrei**	**Defekt**	**Total**
J48	Realität fehlerfrei	12.845	233	13.078	2.378	191	2.569
	Realität defekt	394	5.586	5.980	258	597	855
	Total	13.239	5.819	19.058	2.636	788	3.424
KNN	Realität fehlerfrei	12.565	513	13.078	1.679	154	1.833
	Realität defekt	928	5.052	5.980	242	371	613
	Total	13.493	5.565	19.058	1.921	525	2.446
LR	Realität fehlerfrei	12.495	583	13.078	2.531	38	2.569
	Realität defekt	4.934	1.046	5.980	785	70	855
	Total	17.429	1.629	19.058	3.316	108	3.424
NB	Realität fehlerfrei	6.224	15.485	21.709	28	1.059	1.087
	Realität defekt	698	9.356	10.054	15	366	381
	Total	6.922	24.841	31.763	43	1.425	1.468
NN	Realität fehlerfrei	17.364	43	17.408	2.099	96	2.195
	Realität defekt	3.939	4.065	8.004	433	307	740
	Total	21.303	4.108	25.411	2.532	403	2.935
RF	Realität fehlerfrei	17.282	125	17.407	1.045	42	1.087
	Realität defekt	342	7.662	8.004	79	302	381
	Total	17.624	7.787	25.429	1.124	344	1.468
SGD	Realität fehlerfrei	29.882	660	30.542	2.565	4	2.569
	Realität defekt	12.522	1.405	13.927	841	14	856
	Total	42.404	2.065	44.469	3.406	18	3.424
SVM	Realität fehlerfrei	30.026	516	30.542	1.833	0	1.833
	Realität defekt	12.910	1.017	13.927	613	0	613
	Total	42.936	1.533	44.469	2.446	0	2.446



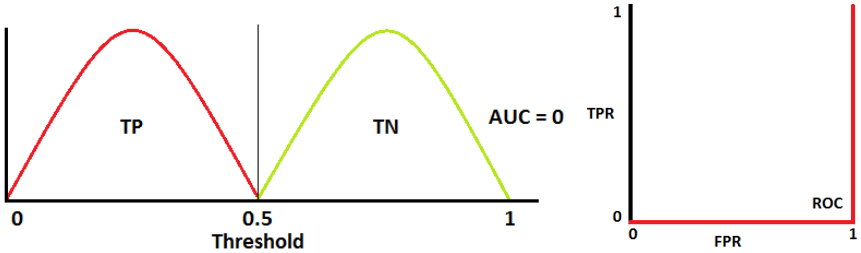
(a) Idealfall **(b)** Idealfall



(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) [19]

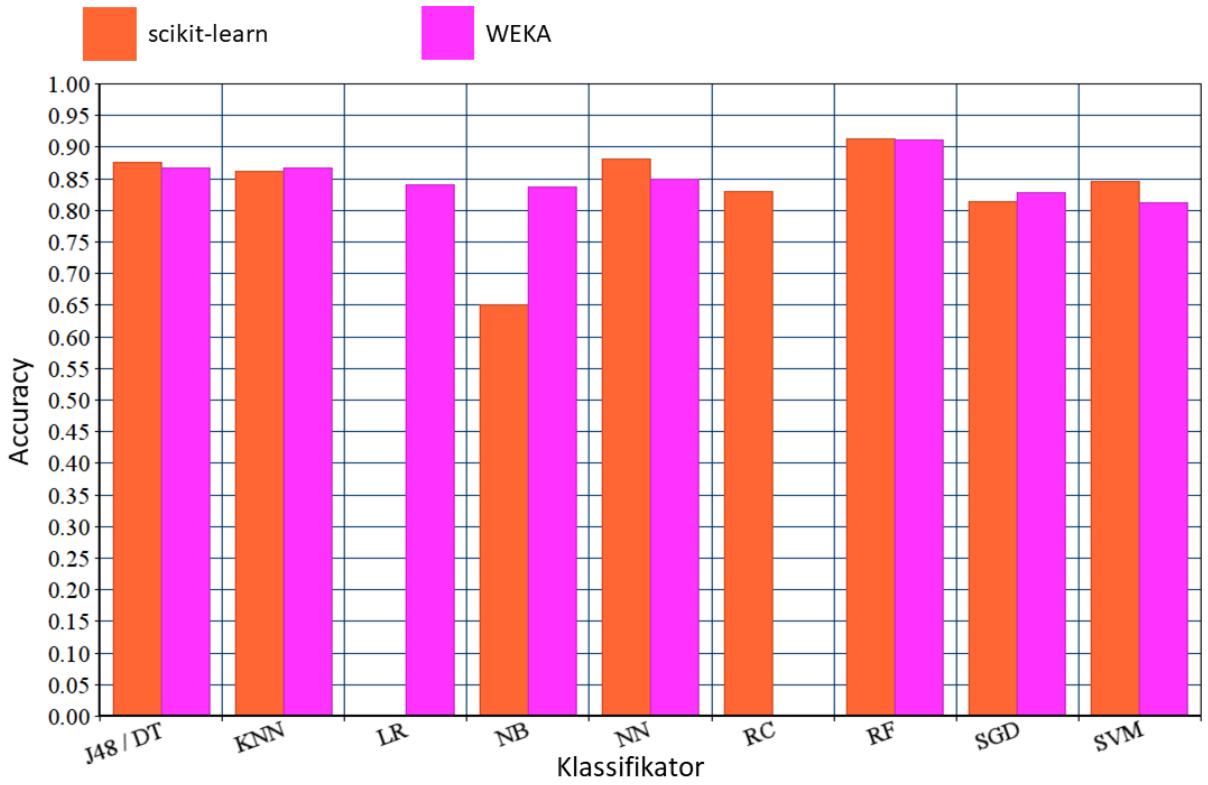


Abbildung 5.3: Vergleich der Accuracies des featurebasierten Datensets

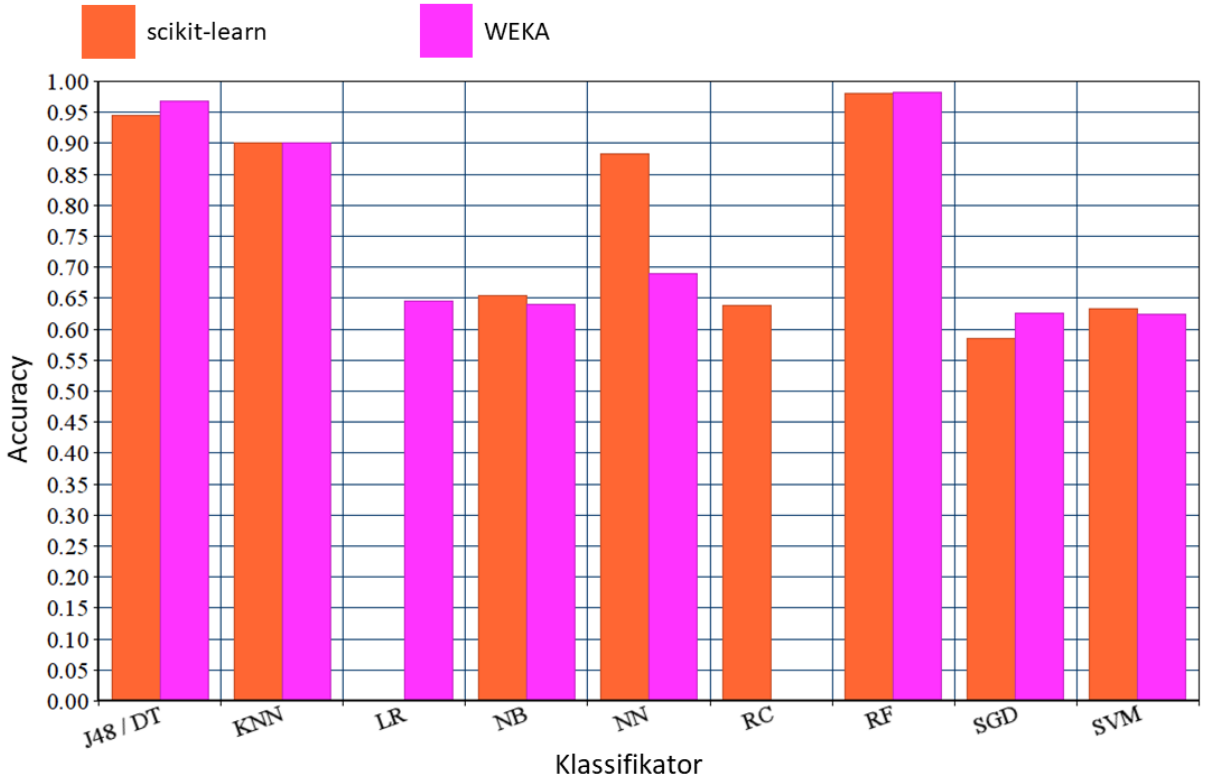
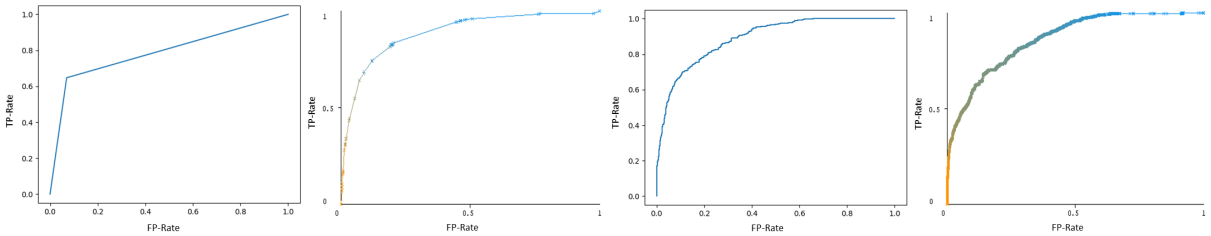


Abbildung 5.4: Vergleich der Accuracies des dateibasierten Datensets

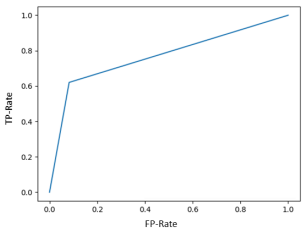


(a) DT (scikit)
AUC = 0,79

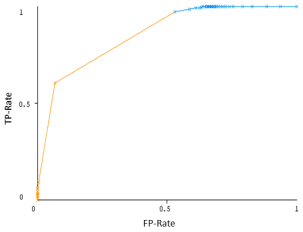
(b) J48 (WEKA)
AUC = 0,88

(c) NN (scikit)
AUC = 0,89

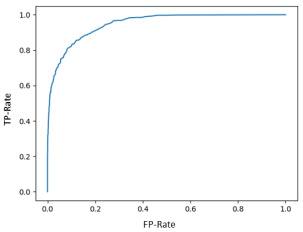
(d) NN (WEKA)
AUC = 0,86



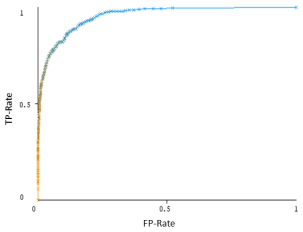
(e) KNN (scikit)
AUC = 0,77



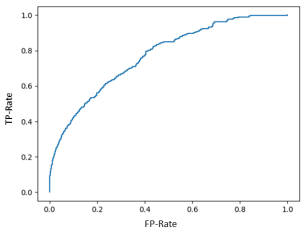
(f) KNN (WEKA)
AUC = 0,85



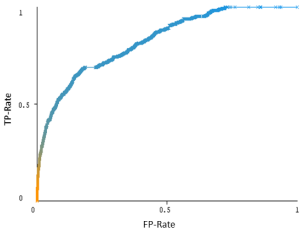
(g) RF (scikit)
AUC = 0,95



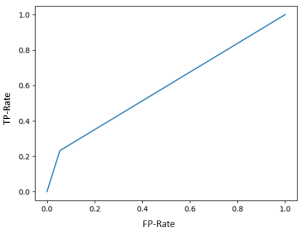
(h) RF (WEKA)
AUC = 0,95



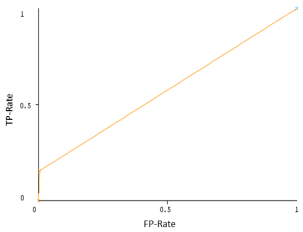
(i) RC (scikit)
AUC = 0,77



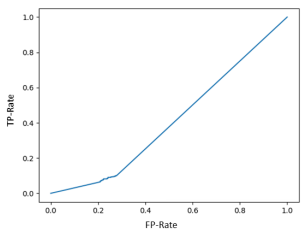
(j) LR (WEKA)
AUC = 0,82



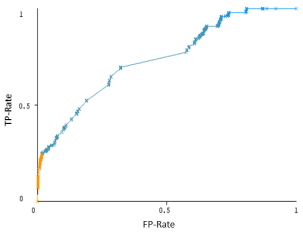
(k) SGD (scikit)
AUC = 0,59



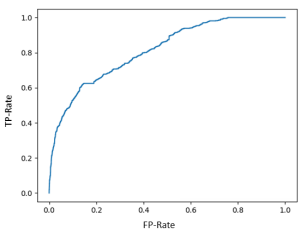
(l) SGD (WEKA)
AUC = 0,57



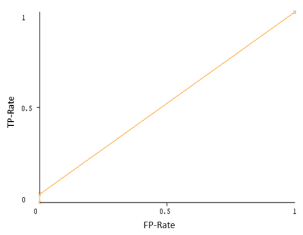
(m) NB (scikit)
AUC = 0,41



(n) NB (WEKA)
AUC = 0,74

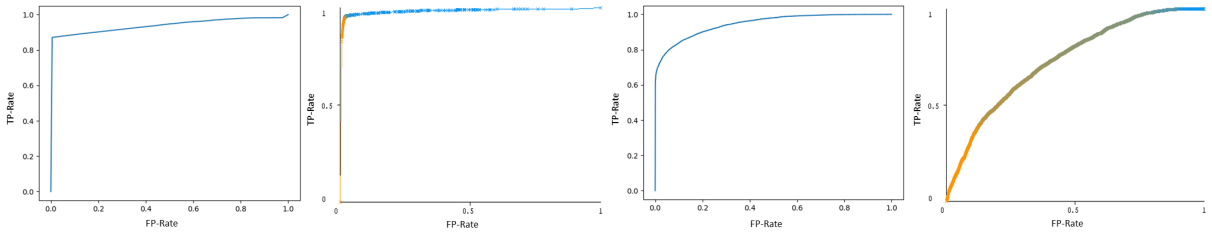


(o) SVM (scikit)
AUC = 0,81



(p) SVM (WEKA)
AUC = 0,52

Abbildung 5.5: ROC-Kurven der Klassifikatoren des featurebasierten Datensets

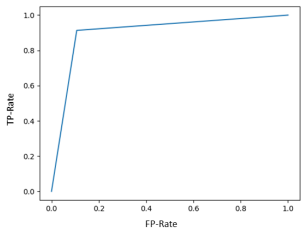


(a) DT (scikit)
AUC = 0,94

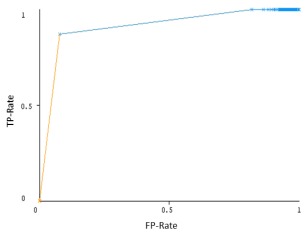
(b) J48 (WEKA)
AUC = 0,98

(c) NN (scikit)
AUC = 0,95

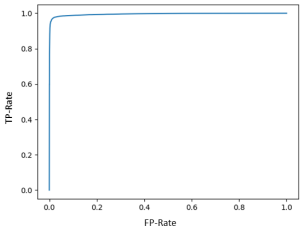
(d) NN (WEKA)
AUC = 0,73



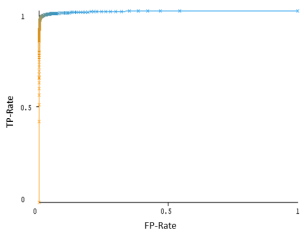
(e) KNN (scikit)
AUC = 0,90



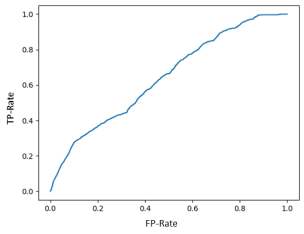
(f) KNN (WEKA)
AUC = 0,91



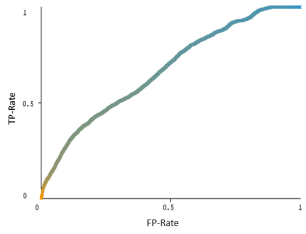
(g) RF (scikit)
AUC = 1,00



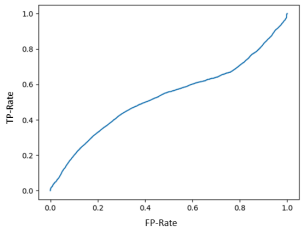
(h) RF (WEKA)
AUC = 1,00



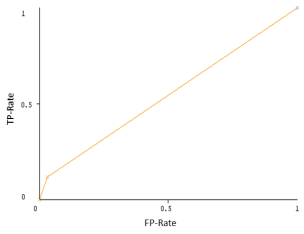
(i) RC (scikit)
AUC = 0,64



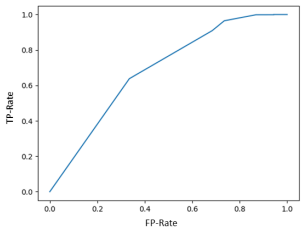
(j) LR (WEKA)
AUC = 0,67



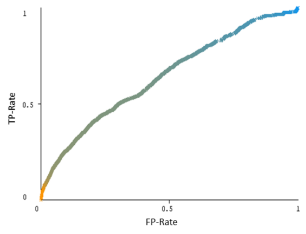
(k) SGD (scikit)
AUC = 0,53



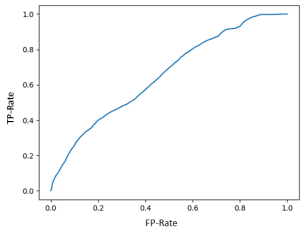
(l) SGD (WEKA)
AUC = 0,54



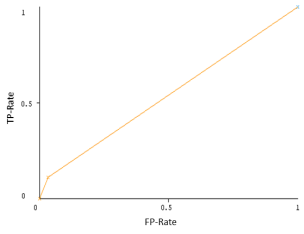
(m) NB (scikit)
AUC = 0,69



(n) NB (WEKA)
AUC = 0,64



(o) SVM (scikit)
AUC = 0,65



(p) SVM (WEKA)
AUC = 0,54

Abbildung 5.6: ROC-Kurven der Klassifikatoren des dateibasierten Datensets

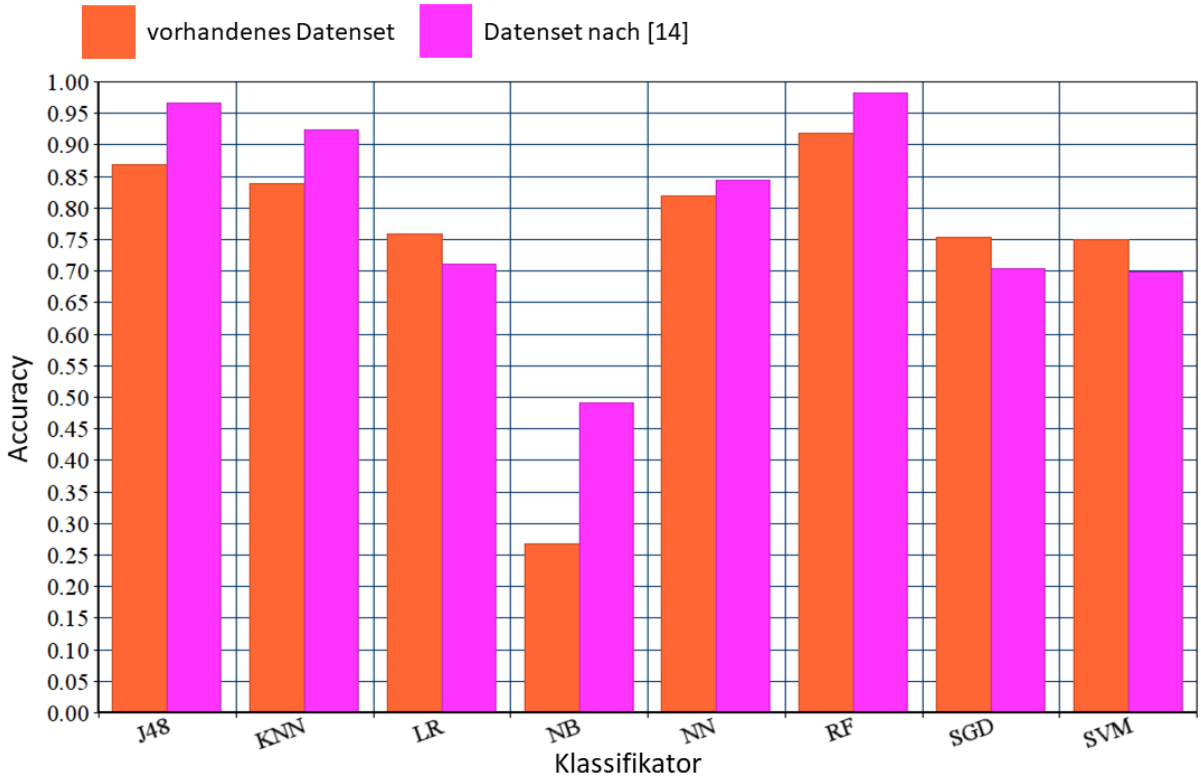
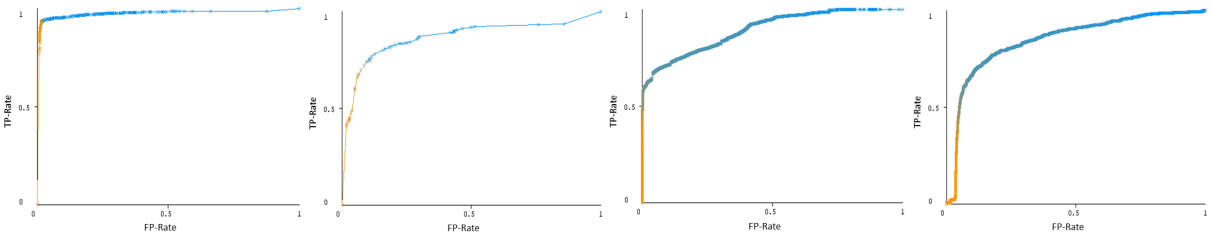


Abbildung 5.7: Übersicht der Accuracies des nicht-featurebasierten Vergleichs

Tabelle 5.7: Ergebnisse der Evaluationsmetriken des nicht-featurebasierten Vergleichs

		Datenset nach			**vorhandenes Datenset**		
		fehlerfrei	**defekt**	**gew. Mittel**	**fehlerfrei**	**defekt**	**gew. Mittel**
J48	TP-Rate	0,98	0,93	0,97	0,93	0,70	0,87
	FP-Rate	0,02	0,02	0,07	0,30	0,07	0,25
	Precision	0,97	0,96	0,97	0,90	0,76	0,87
	F-Score	0,98	0,95	0,97	0,91	0,73	0,87
KNN	TP-Rate	0,96	0,85	0,92	0,92	0,61	0,84
	FP-Rate	0,16	0,04	0,12	0,40	0,08	0,32
	Precision	0,93	0,91	0,92	0,87	0,71	0,83
	F-Score	0,95	0,88	0,92	0,90	0,65	0,83
LR	TP-Rate	0,96	0,18	0,71	0,99	0,08	0,76
	FP-Rate	0,83	0,05	0,58	0,92	0,02	0,69
	Precision	0,72	0,64	0,69	0,76	0,65	0,74
	F-Score	0,82	0,65	0,28	0,86	0,15	0,68
NB	TP-Rate	0,29	0,93	0,49	0,03	0,96	0,27
	FP-Rate	0,07	0,71	0,27	0,04	0,97	0,28
	Precision	0,90	0,38	0,73	0,65	0,26	0,55
	F-Score	0,44	0,54	0,47	0,05	0,41	0,14
NN	TP-Rate	1,00	0,51	0,84	0,96	0,42	0,82
	FP-Rate	0,49	0,00	0,34	0,59	0,04	0,45
	Precision	0,82	0,99	0,87	0,83	0,76	0,81
	F-Score	0,90	0,67	0,83	0,89	0,54	0,80
RF	TP-Rate	0,99	0,96	0,98	0,96	0,79	0,92
	FP-Rate	0,04	0,01	0,03	0,21	0,04	0,16
	Precision	0,98	0,98	0,98	0,93	0,88	0,92
	F-Score	0,99	0,97	0,98	0,95	0,83	0,92
SGD	TP-Rate	0,98	0,10	0,70	1,00	0,02	0,75
	FP-Rate	0,90	0,02	0,62	0,98	0,00	0,74
	Precision	0,71	0,68	0,70	0,75	0,78	0,76
	F-Score	0,82	0,18	0,62	0,86	0,03	0,65
SVM	TP-Rate	0,92	0,07	0,70	1,00	0,00	0,75
	FP-Rate	0,93	0,02	0,64	1,00	0,00	0,75
	Precision	0,70	0,66	0,69	0,75	?	?
	F-Score	0,82	0,13	0,60	0,86	?	?

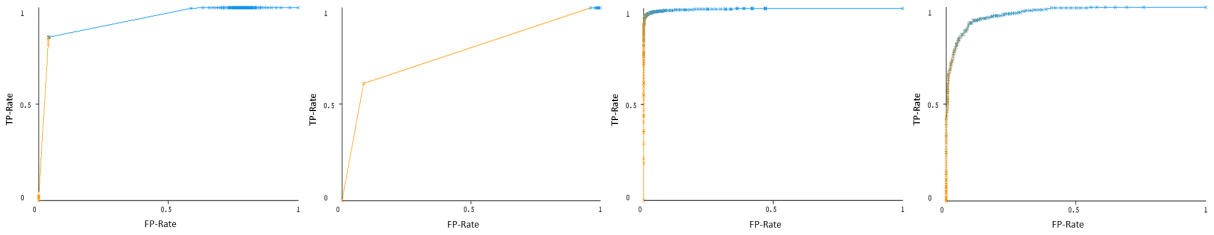


(a) J48 n. [18]
AUC = 0,97

(b) J48 (v. D.)
AUC = 0,86

(c) NN n. [18]
AUC = 0,89

(d) NN (v. D.)
AUC = 0,85

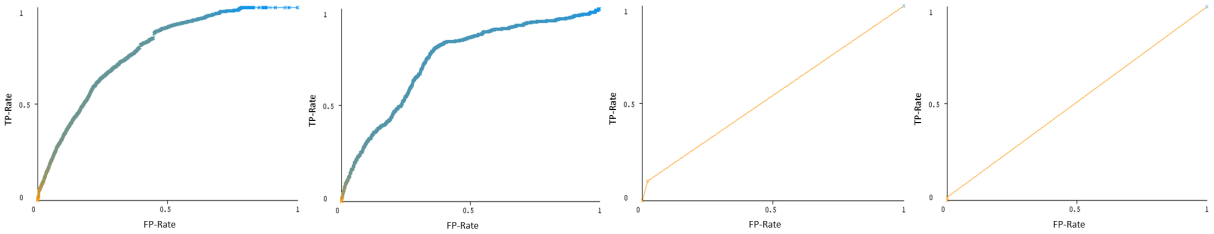


(e) KNN n. [18]
AUC = 0,93

(f) KNN (v. D.)
AUC = 0,77

(g) RF n. [18]
AUC = 1,00

(h) RF (v. D.)
AUC = 0,97

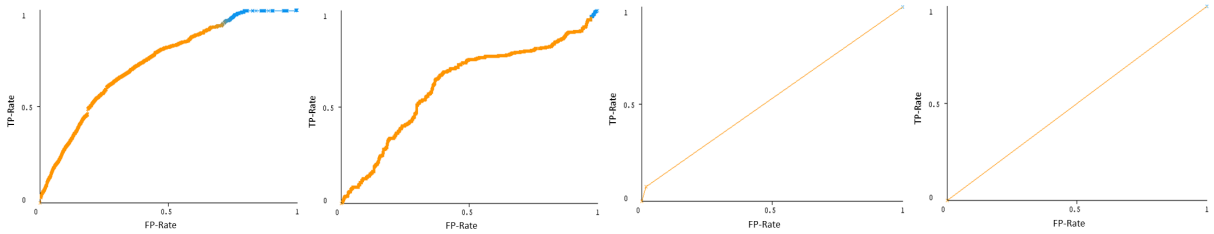


(i) LR n. [18]
AUC = 0,77

(j) LR (v. D.)
AUC = 0,73

(k) SGD n. [18]
AUC = 0,54

(l) SGD (v. D.)
AUC = 0,51



(m) NB n. [18]
AUC = 0,73

(n) NB (v. D.)
AUC = 0,61

(o) SVM n. [18]
AUC = 0,53

(p) SVM (v. D.)
AUC = 0,50

Abbildung 5.8: ROC-Kurven des nicht-featurebasierten Vergleichs
(v.D. = vorhandenes Datenset)

Kapitel 6

Fazit

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] Flavio Medeiros u. a. „Discipline Matters: Refactoring of Preprocessor Directives in the #ifdef Hell“. In: *IEEE Transactions on Software Engineering* 44.5 (Mai 2018), S. 453–469. DOI: 10.1109/tse.2017.2688333.

[18] 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.

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

[20] 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].

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

[22] 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.

[23] Christopher Preschern. „Patterns to escape the #ifdef hell“. In: *Proceedings of the 24th European Conference on Pattern Languages of Programs - EuroPLop '19*. ACM Press, 2019. DOI: 10.1145/3361149.3361151.

[24] 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.

[25] Rodrigo Queiroz u. a. „The shape of feature code: an analysis of twenty C-preprocessor-based systems“. In: *Software & Systems Modeling* 16.1 (Juli 2015), S. 77–96. DOI: 10.1007/s10270-015-0483-z.

[26] 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.

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

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

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

[35] 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.

[36] 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.

[37] Ming Wen u. a. „Exploring and exploiting the correlations between bug-inducing and bug-fixing commits“. In: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2019*. ACM Press, 2019. DOI: 10.1145/3338906.3338962.

[38] 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.

[39] 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

Accuracies der Klassifikatoren je Datenset

Tabelle B.1: Accuracies des featurebasierten Datensets

Klassifikator	Accuracy scikit-learn	Accuracy WEKA
J48 / DT	0,88	0,87
KNN	0,86	0,87
LR		0,84
NB	0,65	0,84
NN	0,88	0,85
RC	0,83	
RF	0,91	0,91
SGD	0,81	0,83
SVM	0,85	0,81

Tabelle B.2: Accuracies des dateibasierten Datensets

Klassifikator	Accuracy scikit-learn	Accuracy WEKA
J48 / DT	0,94	0,97
KNN	0,90	0,90
LR		0,65
NB	0,65	0,64
NN	0,88	0,68
RC	0,64	
RF	0,98	0,98
SGD	0,58	0,63
SVM	0,63	0,62

Tabelle B.3: Accuracies des nicht-featurebasierten Vergleichs

Klassifikator	Accuracy Datenset nach [18]	Accuracy vorhandenes Datenset
J48 / DT	0,97	0,87
KNN	0,92	0,84
LR	0,71	0,76
NB	0,49	0,27
NN	0,84	0,82
RF	0,98	0,92
SGD	0,70	0,75
SVM	0,70	0,75

Anhang C

Erweiterte Ergebnisse der Evaluationsmetriken

Die nachfolgenden Tabellen enthalten die Ergebnisse der Evaluation mit zusätzlichen Metriken. Die Metrik „PRC-Area“ ist wie folgt definiert:

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

Tabelle C.1: Erweiterte Ergebnisse der Evaluationsmetriken des featurebasierten Datensets

		scikit-learn			WEKA		
		Fehlerfrei	**Defekt**	**gew. Mittel**	**Fehlerfrei**	**Defekt**	**gew. Mittel**
DT / J48	TP-Rate	0,93	0,65	0,88	0,95	0,54	0,87
	FP-Rate	0,65	0,07	0,18	0,46	0,05	0,38
	Precision	0,92	0,69	0,87	0,89	0,71	0,86
	Recall	0,93	0,65	0,88	0,95	0,54	0,87
	F-Score	0,92	0,67	0,87	0,92	0,62	0,86
	ROC-Area	0,79	0,79	0,79	0,88	0,88	0,88
	PRC-Area	0,77	0,51	0,72	0,96	0,66	0,90
KNN	TP-Rate	0,92	0,62	0,86	0,93	0,60	0,87
	FP-Rate	0,62	0,08	0,18	0,40	0,07	0,33
	Precision	0,91	0,63	0,86	0,91	0,63	0,86
	Recall	0,92	0,62	0,86	0,93	0,60	0,87
	F-Score	0,92	0,63	0,86	0,92	0,64	0,86
	ROC-Area	0,77	0,77	0,77	0,85	0,85	0,85
	PRC-Area	0,78	0,46	0,72	0,94	0,55	0,87
LR	TP-Rate				0,98	0,25	0,84
	FP-Rate				0,75	0,02	0,60
	Precision				0,84	0,80	0,83
	Recall				0,99	0,25	0,84
	F-Score				0,91	0,39	0,81
	ROC-Area				0,82	0,82	0,82
	PRC-Area				0,95	0,60	0,88
NB	TP-Rate	0,79	0,07	0,65	0,98	0,24	0,84
	FP-Rate	0,07	0,21	0,18	0,76	0,02	0,61
	Precision	0,78	0,07	0,64	0,81	0,77	0,83
	Recall	0,79	0,07	0,65	0,98	0,24	0,84
	F-Score	0,79	0,07	0,65	0,91	0,37	0,80
	ROC-Area	0,41	0,41	0,41	0,74	0,74	0,74
	PRC-Area	0,83	0,18	0,71	0,92	0,48	0,83
NN	TP-Rate	0,94	0,57	0,88	0,95	0,46	0,85
	FP-Rate	0,57	0,06	0,07	0,54	0,05	0,44
	Precision	0,91	0,68	0,87	0,87	0,70	0,84
	Recall	0,94	0,57	0,88	0,95	0,46	0,85
	F-Score	0,93	0,92	0,88	0,91	0,56	0,84
	ROC-Area	0,89	0,89	0,89	0,86	0,86	0,86
	PRC-Area	0,80	0,46	0,71	0,96	0,67	0,90
RC	TP-Rate	1,00	0,10	0,83			
	FP-Rate	0,10	0,00	0,02			
	Precision	0,83	0,89	0,84			
	Recall	1,00	0,10	0,83			
	F-Score	0,91	0,18	0,77			
	ROC-Area	0,77	0,77	0,77			
	PRC-Area	0,81	0,26	0,71			
RF	TP-Rate	0,97	0,67	0,91	0,97	0,67	0,91
	FP-Rate	0,67	0,03	0,15	0,33	0,03	0,27
	Precision	0,93	0,82	0,91	0,92	0,87	0,91
	Recall	0,97	0,67	0,91	0,97	0,67	0,91
	F-Score	0,95	0,74	0,91	0,95	0,75	0,91
	ROC-Area	0,95	0,95	0,95	0,95	0,95	0,95
	PRC-Area	0,80	0,61	0,76	0,99	0,97	0,96
SGD	TP-Rate	0,95	0,23	0,81	0,99	0,15	0,93
	FP-Rate	0,23	0,05	0,09	0,85	0,01	0,68
	Precision	0,84	0,49	0,78	0,83	0,87	0,84
	Recall	0,95	0,23	0,81	0,99	0,15	0,83
	F-Score	0,89	0,32	0,81	0,90	0,26	0,78
	ROC-Area	0,59	0,59	0,59	0,57	0,57	0,57
	PRC-Area	0,80	0,26	0,70	0,83	0,30	0,72
SVM	TP-Rate	0,99	0,24	0,85	1,00	0,04	0,81
	FP-Rate	0,24	0,01	0,06	0,96	0,00	0,77
	Precision	0,85	0,80	0,84	0,81	0,96	0,84
	Recall	0,99	0,24	0,85	1,00	0,04	0,81
	F-Score	0,91	0,37	0,81	0,90	0,08	0,74
	ROC-Area	0,81	0,81	0,81	0,52	0,52	0,52
	PRC-Area	0,80	0,33	0,72	0,81	0,23	0,70

Tabelle C.2: Erweiterte Ergebnisse der Evaluationsmetriken des dateibasierten Datensets

		scikit-learn			WEKA		
		fehlerfrei	**defekt**	**gew. Mittel**	**fehlerfrei**	**defekt**	**gew. Mittel**
DT / J48	TP-Rate	0,99	0,87	0,94	0,98	0,95	0,97
	FP-Rate	0,87	0,01	0,35	0,05	0,02	0,03
	Precision	0,92	0,99	0,95	0,97	0,97	0,97
	Recall	0,99	0,87	0,94	0,98	0,95	0,97
	F-Score	0,96	0,93	0,94	0,97	0,96	0,97
	ROC-Area	0,94	0,94	0,94	0,98	0,98	0,98
	PRC-Area	0,59	0,91	0,72	0,97	0,97	0,97
KNN	TP-Rate	0,89	0,91	0,90	0,92	0,87	0,90
	FP-Rate	0,91	0,11	0,43	0,13	0,08	0,11
	Precision	0,94	0,85	0,90	0,91	0,88	0,90
	Recall	0,89	0,91	0,90	0,92	0,87	0,90
	F-Score	0,92	0,88	0,90	0,92	0,88	0,90
	ROC-Area	0,90	0,90	0,90	0,91	0,91	0,91
	PRC-Area	0,55	0,81	0,66	0,91	0,93	0,97
LR	TP-Rate				0,92	0,24	0,65
	FP-Rate				0,76	0,08	0,49
	Precision				0,64	0,66	0,65
	Recall				0,92	0,24	0,65
	F-Score				0,76	0,36	0,60
	ROC-Area				0,67	0,67	0,67
	PRC-Area				0,76	0,57	0,68
NB	TP-Rate	0,66	0,64	0,65	0,87	0,30	0,64
	FP-Rate	0,64	0,34	0,46	0,70	0,13	0,47
	Precision	0,73	0,56	0,66	0,65	0,61	0,63
	Recall	0,66	0,64	0,65	0,87	0,30	0,64
	F-Score	0,70	0,60	0,66	0,74	0,40	0,60
	ROC-Area	0,69	0,69	0,69	0,64	0,64	0,64
	PRC-Area	0,54	0,51	0,53	0,69	0,56	0,64
NN	TP-Rate	0,97	0,76	0,88	0,86	0,41	0,68
	FP-Rate	0,76	0,03	0,33	0,59	0,14	0,41
	Precision	0,86	0,94	0,89	0,69	0,67	0,68
	Recall	0,97	0,76	0,88	0,86	0,41	0,68
	F-Score	0,91	0,84	0,88	0,76	0,51	0,66
	ROC-Area	0,95	0,95	0,95	0,73	0,73	0,73
	PRC-Area	0,57	0,81	0,67	0,81	0,61	0,73
RC	TP-Rate	0,92	0,23	0,64			
	FP-Rate	0,23	0,08	0,14			
	Precision	0,63	0,66	0,64			
	Recall	0,92	0,23	0,64			
	F-Score	0,75	0,34	0,58			
	ROC-Area	0,64	0,64	0,64			
	PRC-Area	0,57	0,46	0,53			
RF	TP-Rate	0,99	0,97	0,98	0,99	0,97	0,98
	FP-Rate	0,97	0,01	0,40	0,03	0,01	0,02
	Precision	0,98	0,98	0,98	0,98	0,99	0,98
	Recall	0,99	0,97	0,98	0,99	0,97	0,98
	F-Score	0,98	0,98	0,98	0,99	0,98	0,98
	ROC-Area	1,00	1,00	1,00	1,00	1,00	1,00
	PRC-Area	0,59	0,96	0,74	1,00	0,99	1,00
SGD	TP-Rate	0,67	0,45	0,58	0,97	0,12	0,63
	FP-Rate	0,45	0,33	0,38	0,88	0,03	0,54
	Precision	0,65	0,48	0,58	0,62	0,71	0,66
	Recall	0,67	0,45	0,58	0,97	0,12	0,63
	F-Score	0,66	0,47	0,58	0,76	0,20	0,53
	ROC-Area	0,53	0,53	0,53	0,54	0,54	0,54
	PRC-Area	0,57	0,44	0,52	0,62	0,44	0,55
SVM	TP-Rate	0,93	0,20	0,63	0,97	0,11	0,62
	FP-Rate	0,20	0,07	0,12	0,89	0,03	0,55
	Precision	0,63	0,65	0,64	0,62	0,69	0,65
	Recall	0,93	0,20	0,63	0,97	0,11	0,62
	F-Score	0,75	0,31	0,57	0,75	0,19	0,53
	ROC-Area	0,65	0,65	0,65	0,54	0,54	0,54
	PRC-Area	0,58	0,46	0,53			

Tabelle C.3: Erweiterte Ergebnisse der Evaluationsmetriken des nicht-featurebasierten Vergleichs

		Datenset nach [18]			vorhandenes Datenset		
		fehlerfrei	**defekt**	**gew. Mittel**	**fehlerfrei**	**defekt**	**gew. Mittel**
J48	TP-Rate	0,98	0,93	0,97	0,93	0,70	0,87
	FP-Rate	0,02	0,02	0,07	0,30	0,07	0,25
	Precision	0,97	0,96	0,97	0,90	0,76	0,87
	Recall	0,98	0,93	0,97	0,93	0,70	0,87
	F-Score	0,98	0,95	0,97	0,91	0,73	0,87
	ROC-Area	0,97	0,97	0,97	0,86	0,86	0,86
	PRC-Area	0,98	0,96	0,97	0,92	0,75	0,88
KNN	TP-Rate	0,96	0,85	0,92	0,92	0,61	0,84
	FP-Rate	0,16	0,04	0,12	0,40	0,08	0,32
	Precision	0,93	0,91	0,92	0,87	0,71	0,83
	Recall	0,96	0,85	0,92	0,92	0,61	0,84
	F-Score	0,95	0,88	0,92	0,90	0,65	0,83
	ROC-Area	0,93	0,93	0,93	0,77	0,77	0,77
	PRC-Area	0,95	0,84	0,91	0,87	0,53	0,78
LR	TP-Rate	0,96	0,18	0,71	0,99	0,08	0,76
	FP-Rate	0,83	0,05	0,58	0,92	0,02	0,69
	Precision	0,72	0,64	0,69	0,76	0,65	0,74
	Recall	0,96	0,18	0,71	0,99	0,08	0,76
	F-Score	0,82	0,65	0,28	0,86	0,15	0,68
	ROC-Area	0,77	0,77	0,77	0,73	0,73	0,73
	PRC-Area	0,89	0,56	0,79	0,87	0,46	0,77
NB	TP-Rate	0,29	0,93	0,49	0,03	0,96	0,27
	FP-Rate	0,07	0,71	0,27	0,04	0,97	0,28
	Precision	0,90	0,38	0,73	0,65	0,26	0,55
	Recall	0,29	0,93	0,49	0,03	0,96	0,27
	F-Score	0,44	0,54	0,47	0,05	0,41	0,14
	ROC-Area	0,73	0,73	0,73	0,61	0,61	0,61
	PRC-Area	0,85	0,52	0,75	0,78	0,34	0,66
NN	TP-Rate	1,00	0,51	0,84	0,96	0,42	0,82
	FP-Rate	0,49	0,00	0,34	0,59	0,04	0,45
	Precision	0,82	0,99	0,87	0,83	0,76	0,81
	Recall	1,00	0,51	0,84	0,86	0,42	0,82
	F-Score	0,90	0,67	0,83	0,89	0,54	0,80
	ROC-Area	0,89	0,89	0,89	0,85	0,85	0,85
	PRC-Area	0,94	0,86	0,91	0,93	0,60	0,84
RF	TP-Rate	0,99	0,96	0,98	0,96	0,79	0,92
	FP-Rate	0,04	0,01	0,03	0,21	0,04	0,16
	Precision	0,98	0,98	0,98	0,93	0,88	0,92
	Recall	0,99	0,96	0,98	0,96	0,79	0,92
	F-Score	0,99	0,97	0,98	0,95	0,83	0,92
	ROC-Area	1,00	1,00	1,00	0,97	0,97	0,97
	PRC-Area	1,00	0,99	1,00	0,99	0,92	0,97
SGD	TP-Rate	0,98	0,10	0,70	1,00	0,02	0,75
	FP-Rate	0,90	0,02	0,62	0,98	0,00	0,74
	Precision	0,71	0,68	0,70	0,75	0,78	0,76
	Recall	0,98	0,10	0,70	1,00	0,02	0,75
	F-Score	0,82	0,18	0,62	0,86	0,03	0,65
	ROC-Area	0,54	0,54	0,54	0,51	0,51	0,51
	PRC-Area	0,70	0,35	0,59	0,75	0,26	0,63
SVM	TP-Rate	0,92	0,07	0,70	1,00	0,00	0,75
	FP-Rate	0,93	0,02	0,64	1,00	0,00	0,75
	Precision	0,70	0,66	0,69	0,75	?	?
	Recall	0,98	0,07	0,70	1,00	0,00	0,75
	F-Score	0,82	0,13	0,60	0,86	?	?
	ROC-Area	0,53	0,53	0,53	0,50	0,50	0,50
	PRC-Area	0,70	0,34	0,59	0,75	0,25	0,62