FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Untersuchung des Zusammenhangs zwischen Software-Produktmetriken und dem Auftreten von Schwachstellen

Masterarbeit

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

Friedrich-Schiller Universität Jena
Fakultät für Mathematik und Informatik

eingereicht von

Celestino Madera Castro

Matrikelnummer: 153138
geboren am 17.11.1996 in Jena

Betreuer: Prof. Dr. Wolfram Amme, Tim Sonnekalb

Jena, 29. August 2023

Kurzfassung

Im Rahmen dieses Projekts wurde der Zusammenhang zwischen dem Auftreten von Schwachstellen und Software-Produktmetriken auf Grundlage künstlicher neuronaler Netze untersucht. Schwachstellen basieren auf Fehlern und Bugs eines Software-Produkts, wodurch eine Schnittstelle für Angreifer entsteht, durch deren Ausnutzung der Zugriff auf vertrauliche Informationen und die Manipulation von Daten ermöglicht werden kann. Dementsprechend kann die Existenz von Schwachstellen innerhalb einer Software-Anwendung die Qualität dieser Software beeinträchtigen. Mittels Software-Produktmetriken können beispielsweise die Eigenschaften des Quellcodes einer Software quantifiziert werden, wodurch die Verwendung solcher Metriken eine interessante Grundlage für Modelle der Schwachstellenerkennung repräsentiert. Unter Verwendung des CVEfixes-Datensatzes wurden mithilfe der Metriken-Extraktionstolls Understand und Analizo drei verschiedene Datensatze erstellt, die insgesamt über 80 verschiedene Metriken enthalten. Basierend auf der Methode der korrelationsbasierten Merkmalsauswahl sollte die Dimensionalität dieser Datensätze zusätzlich reduziert werden. Auf Grundlage dieser Datensätze wurden Modelle künstlicher neuronaler Netze zur Erkennung des Auftretens von Schwachstellen sowie zur Bewertung ihrer Schwere trainiert. Alle Modelle verfügen allerdings über eine schlechte Performanz aufgrund einer schlechten Datenqualität. Durch die Anwendung verschiedener Korrelationsanalysen konnten außerdem nur wenige, schwache Korrelationen zwischen den Metriken und den verwendeten Zielgrößen und viele, starke Korrelationen zwischen den einzelnen Metriken identifiziert werden. Daher wurde die Qualität der reduzierten Datensätze ebenfalls stark beeinträchtigt.

Inhaltsverzeichnis

Vorwort

An dieser Stelle möchte ich meine aufrichtige Dankbarkeit gegenüber all den Personen zum Ausdruck bringen, die mich während meiner Masterarbeit unterstützt und begleitet haben. Mein besonderer Dank gilt meinem Betreuer Tim Sonnekalb, der mich mit unermüdlichem Engagement, wertvollen Ratschlägen und fachlicher Expertise durch diese Arbeit geführt hat. Ihre Unterstützung und Ermutigung waren von unschätzbarem Wert und haben maßgeblich zum Gelingen dieser Arbeit beigetragen.

Ebenso möchte ich mich bei der Abteilung der sicheren Softwaretechnik des DLR-Instituts für Datenwissenschaften bedanken, die mir die Möglichkeit gegeben hat, diese Forschungsarbeit durchzuführen. Die Zusammenarbeit und die zur Verfügung gestellten Ressourcen haben es mir ermöglicht, meine Ideen zu verwirklichen und neue Erkenntnisse zu gewinnen.

Ein besonderer Dank gebührt meiner Familie, insbesondere meiner Mutter Sabine Madera Castro. Ihre bedingungslose Unterstützung, Liebe und Ermutigung haben mich in allen Phasen meines Studiums begleitet. Ohne sie wäre ich nicht in der Lage gewesen, diese Masterarbeit erfolgreich abzuschließen. Ihre Opfer und ihr Vertrauen in mich sind unermesslich, und ich bin unendlich dankbar dafür.

Vielen Dank!

1 Einleitung

Durch erfolgreiche Sicherheitsattacken auf Software-Anwendungen können potenzielle Gefahren für deren Benutzer und Eigentümer entstehen. In der Regel werden zur Ausübung solcher bösartigen Zwecke die Schwachstellen eines Systems verwendet, welche auf Fehlern und Bugs beruhen und beispielsweise durch Exploits ausgenutzt werden können. Mithilfe solcher Exploits können Angreifer unter anderem auf vertrauliche Informationen zugreifen und diese Daten manipulieren. Dabei bilden Fehler als Schwachpunkte Schnittstellen für Angreifer, welche beispielsweise innerhalb von Spezifikationen, den Entwicklungen oder Konfigurationen von Anwendungen auftreten und durch fehlende Kenntnisse bezüglich Sicherheitsthematiken entstehen können.

Die Existenz von Schwachstellen innerhalb einer Software-Anwendung stellen ein Sicherheitsrisiko dar, wodurch die Qualität einer solchen Software beeinflusst wird. Durch die Identifikation von Schwachstellen innerhalb einer Software-Anwendung wird dementsprechend die Qualität dieser Software beeinträchtigt. Für die Erkennung von Schwachstellen wird jedoch ein parametrisiertes Modell mit geeigneten Eingaben benötigt.

Software-Produktmetriken können beispielsweise die Eigenschaften des Quellcodes einer Software quantifizieren und stellen einen wesentlichen Bestandteil im Software-Messungsprozess dar. Solche Metriken messen beispielsweise die Größe und Komplexität einzelner Software-Komponenten oder spezifische Eigenschaften verwendeter Programmierparadigmen. Aufgrund dieser Eigenschaften repräsentiert die Verwendung solcher Metriken eine interessante Grundlage für Modelle der Schwachstellenerkennung. Auf Grundlage der Anwendung verschiedener Techniken und Werkzeuge lassen sich die Eigenschaften von Quellcode-Abschnitten berechnen, welche bereits identifizierte Schwachstellen enthalten. Diese gemessenen Eigenschaften können wiederum mit den Eigenschaften anderer Quellcode-Abschnitte verglichen werden, um weitere Schwachstellen zu identifizieren.

Im Rahmen dieses Projekts soll der Zusammenhang zwischen solchen Software-Produktmetriken und dem Auftreten von Schwachstellen untersucht werden. Dazu soll ein Datensatz bestehend aus Software-Produktmetriken erstellt werden, auf dessen Grundlage Modelle künstlicher neuronaler Netze trainiert werden. Mithilfe dieser Modelle soll sowohl das Auftreten von Schwachstellen innerhalb von Quellcode-Abschnitten erkannt als auch die Bewertung der Schwere dieser Schwachstellen realisiert werden. Auf Grundlage der Durchführung verschiedener Experiment soll letztendlich die Performanz dieser mittels Software-Produktmetriken trainierten Modelle

untersucht werden. Durch die Auswertung resultierender Ergebnisse soll wiederum der Zusammenhang zwischen diesen Metriken und des Auftretens von Schwachstellen bewertet werden. Darüber hinaus sollen die verwendeten Software-Produktmetriken unter der Berücksichtigung ausgewerteter Ergebnisse bezüglich eines Einsatzes innerhalb von Software-Qualitätsmodellen beurteilt werden.

Für die Definition der durchzuführenden Experimente werden in Kapitel 2 theoretische Grundlagen der Software-Entwicklung beschrieben. Die Theorie künstlicher neuronaler Netze wird wiederum in Kapitel 3 behandelt. Basierend auf diesen theoretischen Grundlagen werden in Kapitel 4 alle Prozesse zur Durchführung der Experimente beschrieben. In Kapitel 5 werden alle Implementierungsentscheidungen zur Realisierung dieser Experiment vorgestellt. Die Ergebnisse durchgeführter Experimente werden letztendlich in Kapitel 6 erläutert, ehe in Kapitel 7 die wesentlichen Informationen dieser Projektarbeit zusammengefasst und ausgewertet werden.

2 Grundlagen der Software-Entwicklung

In diesem Kapitel werden die für dieses Projekt relevanten theoretischen Grundlagen der Software-Entwicklung beschrieben. Dazu wird in Kapitel 2.1 zuerst die Thematik der Software-Sicherheit behandelt, ehe in Kapitel 2.2 das grundlegende Konzept der Software-Qualität beschrieben wird. In Kapitel 2.3 werden Software-Produktmetriken thematisiert. Dabei wird eine Taxonomie zur Kategorisierung von Software-Produktmetriken präsentiert und die Möglichkeiten zur Extraktion dieser Metriken unter Verwendung verschiedener Werkzeuge behandelt.

2.1 Grundlagen der Software-Sicherheit

Durch das Fehlverhalten von Software-Produkten können eine Vielzahl von Problemen ausgelöst werden, welche die Zuverlässigkeit, Verfügbarkeit und Sicherheit von Systemen beeinträchtigen. Um diese Probleme für bösartige Zwecke auszunutzen, versucht eine Reihe von Personen, dieses Fehlerverhalten aktiv zu erzeugen. Dabei werden durch diese Personen, welche auch als Hacker bezeichnet werden können, keine Sicherheitslücken erzeugt. Stattdessen existieren diese Sicherheitslücken bereits und werden durch solche Personen entsprechend ausgenutzt. Somit gelten Sicherheitslücken als wahre Ursachen der Problemen und repräsentieren das Ergebnis einer schlechten Software-Architektur und -Implementierung [1].

Für einen angemessenen Umgang bezüglich der Gefahren einer Sicherheitslücke benötigen Sicherheitsexperten entsprechende Informationen. Diese Informationen werden beispielsweise als Einträge in der *National Vulnerability Database* (*NVD*) archiviert, welche von der *National Institue of Standards and Technology* (*NIST*) betrieben und regelmäßig aktualisiert wird. Solche Einträge werden als *Common Vulnerabilites and Exposures* (*CVE*) bezeichnet und repräsentieren öffentlich bekannte Schwachstellen bereits veröffentlichter Software-Produkte. Neben einem grundlegenden Überblick bezüglich der jeweiligen Schwachstelle stellen CVE-Einträge weitere Informationen mithilfe von Referenzierungen bereit. Während die Kategorisierung einer Schwachstelle mittels Referenzierung zur *Common Weakness Enumeration* (*CWE*) bereitgestellt wird, ist die Schwere einer Schwachstelle durch die Referenzierung und Einordnung in das *Common Vulnerability Scoring System* (*CVSS*) verfügbar. Darüber hinaus können weitere Referenzierungen bestehen, welche zum Beispiel Möglichkeiten zur Ausnutzung oder Behebung der entsprechenden Schwachstelle enthalten. Mithilfe solcher Datenbanken können durch Sicherheitsexperten und verschiedene Werkzeuge, weitere Schwachstellen identifiziert, überwacht und behoben werden [2].

Im Rahmen dieses Kapitels wird die Definition von Software-Sicherheit in Kapitel 2.1.1 behandelt. In Kapitel werden 2.1.2 aktuelle Herausforderungen der Software-Sicherheit thematisiert. Kapitel 2.1.3 beschreibt eine grundlegende Kategorisierung von Sicherheitsattacken und Kapitel 2.1.4 erläutert die Ziele der Software-Sicherheit.

2.1.1 Definition der Software-Sicherheit

Im Rahmen des deutschen Begriffs der Sicherheit werden zwei voneinander unabhängige Aspekte namens *Safety* und *Security* von englischsprachigen Ländern darunter zusammengefasst [3]. Mittels des Safety-Begriffs werden Aspekte der Betriebssicherheit für den Schutz von Menschen und der Umwelt vor physischen Schaden thematisiert. Der Security-Begriff behandelt hingegen die Informationssicherheit für den Schutz von Daten. Aus gesetzlicher Sicht ist Safety zwingend erforderlich und wird in der Regel statisch umgesetzt, sodass Anpassungen häufig erst nach längeren Zeiträumen notwendig sind [3]. Security stellt in diesem Zusammenhang eher eine freiwillige und durch wirtschaftliche Faktoren beeinflusste Komponente dar. Darüber hinaus ist die Security sehr schnelllebig aufgrund der Identifikation neuer, unbekannter Schwachstellen, welche schnelle Entscheidungen und Handlungen zur Sicherstellung benötigen. Im Rahmen dieser Projektarbeit wird für den Begriff der Software-Sicherheit der Security-Begriff verwendet.

Software-Sicherheit repräsentiert in diesem Zusammenhang eine Eigenschaft des Verhaltens eines kompletten Systems in einer bestimmten Umgebung, welche erweiterte Planungen und ein sorgfältig durchdachtes Konzept benötigt. Systeme können mit einer ausreichenden Sicherheit für einzelne Umgebungen bereitgestellt werden, während sie völlig unsicher in anderen Umgebungen ausgeführt werden können. Durch die Spezifikation von Umgebungen wie beispielsweise durch die Auswahl einer Linux-Distribution entstehen dementsprechend zusätzliche Herausforderungen. Daher wird wird die Software-Sicherheit als ein Prozess des Risikomanagements definiert, in dem Software als die Wurzel aller bekannnten Sicherheitsproblemen von Computern gilt [1]. Dieser Prozess wird in allen Entwicklungsphasen einer Software benötigt, um die Sicherheit durch proaktive Identifikation und Auslöschung von Problemen innerhalb der Software zu erhöhen [4]. Ein von Grund auf erstelltes Sicherheitskonzept stellt im Gegensatz zu dem Hinzufügen einzelner Sicherheitsaspekte zu einem bereits bestehenden Entwurf einen effektiveren Ansatz dar.

In Abhängigkeit des Kontexts verfügt der Sicherheitsbegriff jedoch für verschiedene Personen über unterschiedliche Bedeutungen. Beispielsweise kann Sicherheit auch als die Durchsetzung einer Richtlinie aufgefasst werden, in der Regeln für den Zugriff auf Ressourcen beschrieben werden [1]. Eine Sicherheitsverletzung re-

präsentiert in diesem Fall zum Beispiel die unautorisierte Anmeldung eines Benutzers in einem System. Aufgrund weitreichend geteilter, impliziter Richtlinien wird eine explizite Sicherheitsrichtlinie nicht benötigt.

Darüber hinaus werden die Kriterien der Zuverlässigkeit und Sicherheit aufgrund vieler Gemeinsamkeiten miteinander verglichen. Die Zuverlässigkeit wird als die Messung der Robustheit eines Systems hinsichtlich des Auftretens von Fehlern definiert. Demzufolge kann Software-Sicherheit als Teilmenge der Zuverlässigkeit angesehen werden, bei der die Messung der Robustheit auf Grundlage einer Sicherheitsrichtlinie stattfindet [1].

2.1.2 Relevanz der Software-Sicherheit

Generell verfügen alle komplexen Systeme über mehrere Risiken, wobei Software-Systeme grundlegend als komplex definiert werden. Besonders komplexe Software-Systeme verfügen darüber hinaus über eine erschwerte Nachvollziehbarkeit und Analysierbarkeit, wodurch Sicherheitsrisiken meist durch die interne Komplexität versteckt werden [1]. Dementsprechend sind Computersysteme anfällig für versteckte Risiken, welche meistens zu spät identifiziert werden, sodass fatale Folgen entstehen können. Beispielsweise kann durch bösartige Programmierer Systemsoftware unbemerkt modifiziert werden oder durch unwissende Programmierer während der Realisierung wichtiger Funktionalitäten Fehler integriert werden.

Durch die steigende Konnektivität von Computernetzwerken, welche vor allem durch die Technologie des Internets verursacht wurde, werden zusätzlich die Möglichkeiten potenzieller Angriffe sowohl erhöht als auch vereinfacht. Größere Risiken entstehen beispielsweise aufgrund der wachsenden Abhängigkeit netzwerkbasierender Kommunikation für Menschen, Geschäfte und Regierungen. Dadurch wird mithilfe des Internets zur Ausnutzung von Software-Schwachstellen kein physikalischer Zugriff zwingend benötigt. Web-Services und service-orientierte Architekturen als Anwendungen zwischen verschiedenen Netzwerken stellen in diesem Kontext eine große Gefahr dar [4].

Ein gut strukturierter Prozess der Softwareentwicklung mittels dem Entwurf von Anforderungen, der Ausarbeitung einer detaillierten Spezifikationen bis hin zur tatsächlichen Implementierung sollte diesen Problemen entgegen wirken. Aufgrund des erhöhten wirtschaftlichen Drucks zur Produktion neuer Systeme wird der Entwicklungsprozess jedoch komprimiert, sodass einige essenzielle Methoden der Softwareentwicklung gar nicht erst verwendet werden [1]. Eine effektive Risikobewertung der Software-Sicherheit kann allerdings nur durch Expertenwissen und dem Einsatz umfangreicher Techniken vollzogen werden, sodass Risiken zwischen architekturellen

Problemen und Implementierungsfehlern unterschieden werden müssen.

Die Erweiterbarkeit von Software in Kombination mit dem Internet-Phänomen verschärft die Herausforderungen der Software-Sicherheit ebenfalls [1]. Durch die Verwendung von Zwischenrepräsentationen werden beispielsweise Abbildungen des Quellcodes mehrfach über verschiedene Netzwerke transferiert, ehe sie ausgeführt werden, wodurch ein besonders hohes Risiko entsteht. Deshalb sollte aufgrund der Komprimierung des Lebenszyklusses der Softwareentwicklung die Verwendung solcher Techniken vor allem bei der Integration von Software-Erweiterungen und Aktualisierungen stets sorgfältig bedacht sein. Meistens werden Funktionalitäten inkrementell in einem System entwickelt, wodurch die potenzielle Gefahr der Einführung neuer Sicherheitslücken sehr hoch sein kann. Diese Risiken verschlimmern sich vor allem durch die Verwendung von Low-Level-Programmiersprachen wie *C* oder *C++*, welche keinen Schutz gegenüber einfachen Arten von Attacken wie *Buffer-Overflows* bereitstellen. Selbst ein fehlerfreies System, welche in einer solchen Programmiersprachen formuliert wurde, ist durch unzureichende Konfigurationen angreifbar.

2.1.3 Kategorisierung von Schwachstellen

Probleme der Software-Sicherheit lassen sich grundlegend auf Fehler innerhalb von Entwürfen und Implementierungen zurückführen. *Defekte* gelten als Schwachstellen, welche sich zwischen Implementierungen und Entwürfen befinden und teilweise als Problem für viele Jahre in einem System schlummern, ehe durch ihr Auftauchen in bestimmten System riesige Konsequenzen entstehen [4]. Solche Schwachstellen werden prinzipiell anhand verschiedener Granularitätsebenen unterschieden. Defekte der niedrigsten Ebene gelten beispielsweise als Implementierungsfehler innerhalb einzelner Code-Zeilen oder -Abchnitte. Schwachstellen der mittleren Ebene entstehen wiederum durch Interaktionen mehrerer Code-Zeilen oder -Abschnitte und können beispielsweise durch die Untersuchung des Verhaltens mehrerer Funktionen bezüglich des Teilens globaler Variablen erkannt werden. Darüber hinaus existieren Schwachstellen auf Entwurfsebene, welche über eine viel größerer Reichweite verfügen und meistens nur durch eine große Expertise erkannt werden können.

In diesem Zusammenhang gelten *Bugs* als Software-Probleme auf Implementierungsebene, deren Quellcode teilweise gar nicht ausgeführt werden kann. Solche Probleme können in der Regel leicht erkannt und behoben werden. *Flaws* gelten hingegen als Probleme tieferer Ebenen und werden als einzelne und viel subtilere Fehler wie beispielsweise innerhalb von Array-Referenzierungen oder durch die Verwendung falscher Systemaufrufe charakterisiert. Zur Bewertung des potenziellen Schadens, welcher durch Bugs und Flaws verursacht werden kann wird, wiederum

ein *Risiko* definiert. Dieses Risiko repräsentiert den Einfluss einer Schwachstelle auf den Nutzen einer Software in Abhängigkeit von der Wahrscheinlichkeit, dass diese Schwachstelle ausgenutzt wird [4].

2.1.4 Ziele der Software-Sicherheit

Für den Begriff der Software-Sicherheit existiert keine reduzierte Beschreibung aufgrund der Repräsentation als dynamsiche Eigenschaft von Software-Systemen und der fehlenden Akzeptanz einer konkreten Definition durch die Allgemeinheit [1]. Stattdessen ist Sicherheit stets relativ und verfügt bezüglich verschiedener Kontexte eine unterschiedliche Bedeutung. Daher beruht diese Thematik eher auf den Fragen, was beschützt werden muss und gegenüber wem diese Elemente geschützt werden sollen. Auf Grundlage dieser Fragen beruht ein besseres Verständnis der Software-Sicherheit anhand ihrer Ziele, welche in den folgenden Absätzen erklärt werden.

Durch die Einführung des Internets und die Vermehrung der Gefahren aufgrund komprimierter Software-Lebenszyklen spielt die *Vermeidung* von Angriffen eine immer relevantere Rolle. Gefundene Schwachstellen können beispielsweise durch das Internet deutlich schneller verbreitet werden. Mittels der Einbettung von Attacken in einfache Skripte wird deren Wiederverwendung für Angreifer vereinfacht. Automatisierte und internet-basierte Attacken auf Software gelten dabei als ernsthafte Gefahren, welche in Abschätzungen des Risikomanagements einbezogen werden müssen [1].

Ein weiteres Ziel der Software-Sicherheit stellt die *Rückverfolgbarkeit und Rechenschaftspflicht* dar. Aufgrund der fehlenden Garantie vollständiger Sicherheit können Attacken auf Software nicht vollumfänglich vermieden werden [1]. In diesem Zusammenhang dient die Rechenschaftspflicht nicht als direkte Technologie der Vermeidung von Software-Attacken. Stattdessen soll dieses Konzept zur Abschreckung potenzieller Angreifer basierend auf seiner bloßen Existenz dienen. Gute Messungen bezüglich der Rückverfolgbarkeit und Rechenschaftspflicht sind essenziell für die Forensik, um entsprechende Attacken zu erkennen, zu zerlegen und zu veranschaulichen. In Gerichtsverfahren können dadurch entscheidende Beweise geliefert werden, um zu verdeutlich, wer wann etwas praktiziert hat. Die Rechenschaftspflicht stellt jedoch innerhalb der Softwareentwicklung aufgrund der eigenen Anfälligkeit solcher Systeme eine technologische Herausforderung dar. Eine Verifikation von Protokollen der Rechenschaftspflicht ist in diesem Rahmen allerdings sehr komplex, obwohl die Rechenschaftspflicht einen essenziellen Aspekt der Software-Sicherheit darstellt [1].

Die *Überwachung* eines Software-Systems in Form einer Echtzeitprüfung stellt ein weiteres Sicherheitsziel dar. Als einfache Überwachungssysteme existieren bei-

spielsweise Einbruchmeldesysteme, die auf der Überwachung von Prozessen und deren Ressourcenverbrauch basieren. Die Überwachung von Programmen ist in diesem Kontext auf vielen Ebenen möglich. Einfache Ansätze basieren beispielsweise auf der Beobachtung bekannter Signaturen und der Erkennung gefährlicher Muster bei Aufrufen der niedrigen System-Ebenen, wodurch letztendlich der Fortschritt von Angriffen identifiziert werden kann [1]. Komplexere Ansätze verwenden hingegegen spezielle Überwacher, welche innerhalb des Codes in Form von Behauptungen platziert werden [1].

Die Sicherhstellung der *Privatsphäre und Vertraulichkeit* gelten ebenfalls als Ziele der Software-Sicherheit. In den meisten Anwendungsfällen bestehen eindeutige Gründe für die Existenz von Geheimnissen. Während Geschäfte beispielsweise den Schutz ihrer Handelsgeheimnisse gegenüber anderen Wettbewerbern benötigen, möchten viele Webbenutzer zum Beispiel ihre Online-Aktivitäten gegenüber invasiven Marketing-Maschinen schützen. Für die Entwicklung eines Software-Produkts werden Entwürfe konzipiert, um durch die Ausführung auf Maschinen nützliche Aufgaben zu bewältigen. Jede Maschine verfügt jedoch über Möglichkeiten zur Extraktion versteckter Geheimnisse aus den Bestandteilen der Software [1]. Im Zusammenhang mit dieser Gefahr enthalten die Konzepte von Software-Produkten in der Regel keine Ansätze zur Sicherstellung der Privatsphäre. Daher sollte beispielsweise die Speicherung verwendeter Passwörter innerhalb des Quellcodes vermieden werden.

Das Konzept der *mehrstufigen Sicherheit* wird aufgrund der Existenz unterschiedlich geheimer Informationen als weiteres Ziel deklariert. Regierungen klassifizieren ihre Informationen beispielsweise in mehreren Ebenen. Somit existieren Informationen, welche zum Beispiel als unklassifiziert definiert sind und Informationen des öffentlichen Gebrauchs sowie geheime oder auch streng geheime Informationen. Unternehmen verfügen ebenfalls über zu schützende Daten, welche teilweise auch vor den eigenen Angestellten geheim gehalten werden müssen. Dadurch werden unterschiedliche Ebenen des Schutzes zur Gewährung verschiedener Informationsebenen benötigt. Saubere Interaktionen von Software sind in Zusammenhang zu einem mehrstufigen Sicherheitssystem jedoch sehr komplex zu integrieren [1].

Ein weiteres Ziel der Software-Sicherheit ist die *Anonymität*, welche jedoch in der Praxis ein zweischneidiges Schwert darstellt. Zum einen existieren gute soziale Gründe für bestimmte Arten anonymer Handlungen und zum anderen bestehen ebenfalls gute soziale Gründe zur Vermeidung von Anonymität [1]. Daher sind gute Entscheidungen bezüglich der Anonymität in Kombination mit den Aspekten der Privatsphäre zu treffen, um eine angemessene Software-Sicherheit zu gewährleisten.

Beispielsweise sind Technologien zur Beeinträchtigung der Anonymität und Privatsphäre für die rechtliche Strafverfolgung nützlich. Cookies bieten hingegen eine gute Unterstützung in alltäglichen Handlungen, welche allerdings über ein Risiko hinsichtlich der Ausbeutung von Informationen verfügen. Somit werden gute Überlegungen durch Architekten, Entwickler und Manager benötigt, um die Folgen der Sammlung von Daten durch Programme zu bewerten.

Neben der Vertraulichkeit und Integrität zählt die *Authentifizierung* als eines der drei größten Sicherheitsziele. Es stellt ein zentrales Thema der Software-Sicherheit aufgrund des essenziellen Wissens bezüglich der Differenzierung zwischen vertraubaren und nicht vertraubaren Entitäten dar. Dementsprechend wird innerhalb einer Sicherheitsrichtlinie eine Definition benötigt, die festlegt, durch wen welche Handlungen ausgeführt werden können und welche Informationen geschützt werden müssen [1]. Ein großes Sicherheitsproblem stellt in diesem Zusammenhang die Anmeldung von Benutzern mit Passwörtern dar, welche es ermöglichen, Handlungen in einem sicherheitskritischen System auszuführen. Die Ausführung bestimmter Handlungen basiert in diesem Konzept auf der Verteilung von Benutzerrollen. Gerade in Systemen, welche finanzielle Transaktionen ausführen, stellt die Authentifizierung ein kritsiches und ernstzunehmendes Sicherheitsproblem dar, für das je nach Anwendungsfall starke Maßnahmen verwendet werden müssen.

Die *Integrität* thematisiert als Sicherheitsziel die Überprüfung einzelner Komponenten hinsichtlich ihrer Modifikation und Manipulation. In vielen Fällen ist für Benutzer das Vertrauen bezüglich korrekter Daten essenziell, da andernfalls fatale Foglen entstehen können. Durch das Fälschen von Informationen wird das Ziel der Integrität bei steigender Vertraulichkeitsstufe in Bezug zu korrekten Informationen immer kritischer angesehen [1].

2.2 Grundlagen der Software-Qualität

Qualität ist ein subjektives Konzept, welches von individuellen Bedürfnissen, Erwartungen und Anforderungen der Benutzer bezüglich eines Produkts, eines Prozesses oder einer Dienstleistung abhängt [5]. In der Wirtschaft kann die Bewertung der Qualität somit als Maßstab zur Erfüllung von Kundenbedürfnissen und -erwartungen betrachtet werden. Diese Bewertung basiert in der Regel auf der Gesamtheit an Merkmalen und Eigenschaften, welche geeignet sind, um den Grad der Erfüllung entsprechender Anforderungen zu bestimmen.

Die Qualität von Software ist dementsprechend ein entscheidender Faktor für den Erfolg von Unternehmen. Um individuelle Anforderungen und Erwartungen der Benutzer an einem Software-Produkt zu erfüllen, wurde die Norm *ISO/IEC 25010*

entwickelt. Die ISO/IEC 25010 ist ein Leitfaden, welcher auch bekannt als *Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE)* ist. Sie wird als internationaler Standard angesehen und definiert ein umfassendes Framework zur Bewertung und Verbesserung der Software-Qualität. Dazu werden acht Hauptqualitätskriterien festgelegt, welche verschiedene Aspekte der Software-Qualität abdecken [6]. Ein grundlegender Überblick über alle Hauptqualitätskriterien und ihren Aspekten kann in Abbildung 1 betrachtet werden. Eine Beschreibung der einzelnen Kriterien ist in den folgenden Unterkapiteln enthalten.

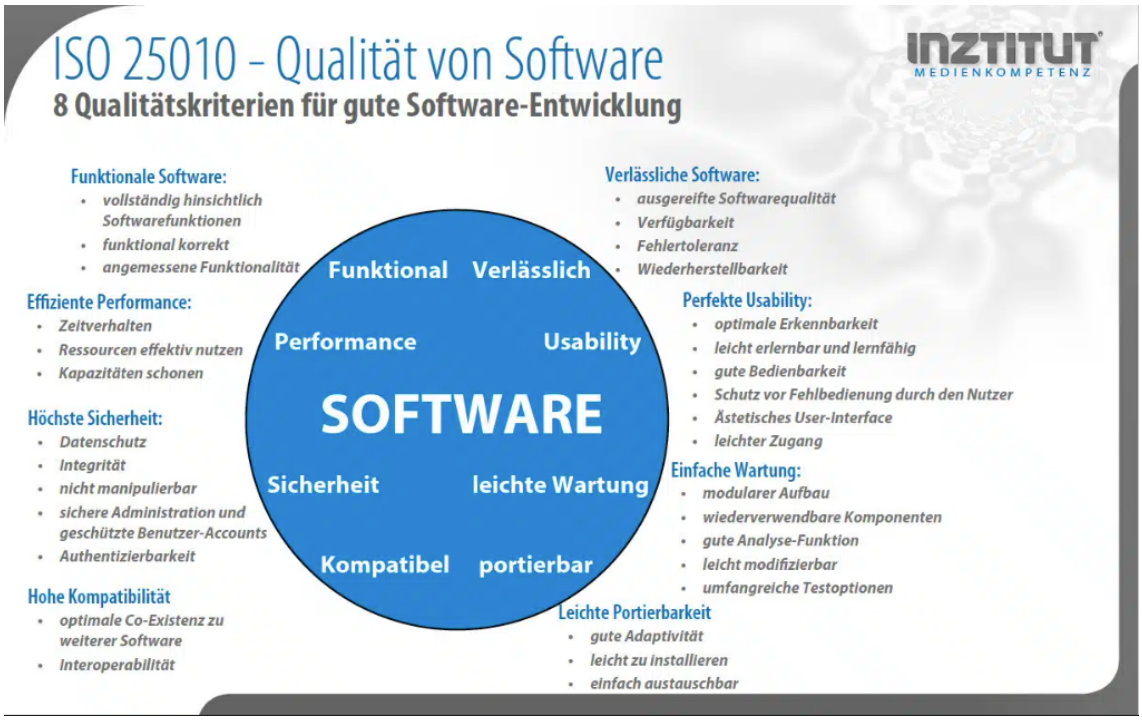


Abbildung 1: Überblick der Hauptqualitätskriterien und ihrer Aspekte nach ISO/IEC 25010 [6]

Zur Bewertung einzelner Aspekte der Software-Qualität enthält das Framework außerdem eine Reihe messbarer Attribute und Metriken. Diese Bewertung kann sowohl während der Entwicklung als auch nach der Bereitstellung der Software durchgeführt werden. Für die kontinuierlichen Verbesserung der Software-Qualität werden Richtlinien und bewährte Methoden bereitgestellt, welche beispielsweise die Identifizierung von Schwachstellen, die Planung und Umsetzung von Verbesserungsmaßnahmen sowie die Überwachung der Wirksamkeit dieser Maßnahmen umfassen.

2.2.1 Funktionalität

Das Kriterium Funktionalität beschreibt die Erfüllung funktionaler Anforderungen und der Bereitstellung erwarteter Funktionen. Ein Software-Produkt wird demnach

als funktionale Software angesehen, sofern die bereitgestellten Funktionen des Systems vorgegebene und implizite Bedürfnisse unter Verwendung spezifizierter Bedingungen erfüllen [7]. Die verschiedenen Aspekte der Funktionalität werden in Anhang 11.1.1 beschrieben.

2.2.2 Benutzbarkeit

Das Kriterium Benutzbarkeit beschreibt die Verwendung eines Systems oder Produkts durch spezifizierte Benutzer zur Erreichung spezifizierter Ziele mit Effektivität, Effizienz und Zufriedenheit in einem spezifizierten Anwendungskontext [7]. Software-Produkte mit optimaler Benutzbarkeit sind einfach zu erlernen, effizient zu bedienen und weisen gute Benutzererfahrungen auf. Die verschiedenen Aspekte der Benutzbarkeit werden in Anhang 11.1.2 beschrieben.

2.2.3 Zuverlässigkeit

Das Kriterium Zuverlässigkeit betrachtet die Durchführung spezifizierter Funktionen eines Produkts, eines Systems oder einer Komponente unter spezifizierten Bedingungen innerhalb einer spezifizierten Zeitperiode [7]. Verlässliche Software ist gegenüber Fehlern und Ausfällen stabil, zuverlässig und widerstandsfähig. Die verschiedenen Aspekte der Zuverlässigkeit werden in Anhang 11.1.3 beschrieben.

2.2.4 Wartbarkeit

Das Kriterium Wartbarkeit thematisiert die Effektivität und Effizienz für die Modifikation eines Produkts oder Systems zur Erweiterung, Korrektur oder Anpassung der Umgebung oder Anforderungen [7]. Ein Software-Produkt sollte demnach einfach zu warten, zu erweitern und anzupassen sein. Die verschiedenen Aspekte der Wartbarkeit werden in Anhang 11.1.4 beschrieben.

2.2.5 Effizienz

Das Kriterium Effizienz beschreibt die Performanz eines Software-Produkts in Relation zum Umfang verbrauchter Ressourcen unter vorgegebenen Bedingungen. Ein Software-Produkt verfügt über eine effiziente Performanz, sofern verfügbare Ressourcen effizient genutzt werden, während hohe Leistungen erzielt werden können [7]. Die verschiedenen Aspekte der Effizienz werden in Anhang 11.1.5 beschrieben.

2.2.6 Kompatibilität

Das Kriterium Kompatibilität beschreibt die Übermittlung von Informationen durch ein Produkt, ein System oder einer Komponente mit einem anderen System, einem anderen Produkt oder einer anderen Komponente. Dabei soll die Übermittlung der Informationen zur Durchführung benötigter Funktionen auch während des Teilens der gleichen Hardware oder der gleichen Software-Umgebung gewährleistet werden [7]. Ein Software-Produkt sollte eine hohe Kompatibilität mit anderen Systemen, Standards und Schnittstellen aufweisen. Die verschiedenen Aspekte der Kompatibilität werden in Anhang 11.1.6 beschrieben.

2.2.7 Portabilität

Das Kriterium behandelt die Effektivität und Effizienz zur Übertragung eines Systems, eines Produkts oder einer Komponente von einer Hardware, Software oder anderen Betriebs- oder Nutzungsumgebung zu anderen Umgebungen [7]. Ein Software-Produkt verfügt über eine leichte Portierbarkeit, sofern es auf verschiedenen Plattformen und Umgebungen problemlos eingesetzt werden kann. Die verschiedenen Aspekte der Portabilität werden in Anhang 11.1.7 beschrieben.

2.2.8 Sicherheit

Das Kriterium Sicherheit thematisiert den Schutz von Informationen und Daten durch ein Produkt oder ein System mit angemessenen Zugang auf Daten durch Personen und andere Produkte oder Systeme hinsichtlich ihrer Art und Ebene der Genehmigung bzw. Autorisation [7]. Software mit hoher Sicherheit schützt Daten und Systeme vor unbefugtem Zugriff und bietet eine sichere Umgebung. Dieses Hauptqualitätskriterium repräsentiert durch seine in Anhang 11.1.8 beschriebenen Aspekte die Ziele der Software-Sicherheit, welche in Kapitel 2.1.4 behandelt wurden.

2.3 Software-Produktmetriken

Software-Produktmetriken erfassen die Eigenschaften von Software-Produkten durch die Abbildung verschiedener Charakteristiken gemessener Entitäten [8]. Für die Berechnung solcher Metriken werden Informationen aus dem Quellcode extrahiert und gegebenenfalls weiterverarbeitet. Die Abbildung von Charakteristiken gemessener Entitäten werden in der Regel als numerische Werte dargestellt. Ordinäre oder kategorische Werte können in besonderen Fällen ebenfalls eingesetzt werden. Solche Werte werden durch Kategorien repräsentiert, die im Falle ordinärer Werte eine

natürliche Ordnung oder Rangfolge aufweisen und im Fall kategorischer Werte keiner spezifischen Ordnung oder Rangfolge unterliegen. Auf der Grundlage bereits berechneter Metriken können außerdem weitere Metriken ermittelt werden [8].

Durch die Auswertung von Software-Produktmetriken können wiederum Aussagen über die Qualitätsaspekte eines Software-Artefakts getroffen werden. Da die Qualitätsbewertung Bestandteil eines Software-Produkts ist, stellen diese Metriken eine essentielle Komponente im Software-Messungsprozess dar, in dem die Eigenschaften der Entitäten jedes Zwischenprodukts überprüft werden. Diese Aufgabe wird in jeder Phase des Software-Entwicklungsprozesses benötigt, um die Qualität des letztendlichen Software-Produkts zu überwachen. Deshalb werden während jeder Phase der Software-Entwicklung mehrere Metriken für jedes Zwischenprodukt gemessen und ausgewertet, bis eine lieferbare Software über angemessene Eigenschaften verfügt [8].

Im Rahmen dieses Kapitels wird in Kapitel 2.3.1 ein grober Überblick über die verschiedenen Kategorien von Software-Produktmetriken beschrieben und anschließend in Kapitel 2.3.2 Methoden und Werkzeuge zur Extraktion dieser Metriken aus dem Quelcode vorgestellt.

2.3.1 Kategorien der Software-Produktmetriken

Für jedes Programmierparadigma existiert eine Vielzahl von Metriken, welche sich anhand der jeweiligen zugrunde liegenden Konzepte unterscheiden. Ein Großteil dieser Metriken kann jedoch anhand des entsprechenden Kontexts für andere Paradigmen adaptiert werden. Daher eignet sich zur Kategorisierung solcher Metriken eine Unterscheidung zwischen der Aussagekraft der jeweiligen Kontexte [8]. Ein Überblick über die verschiedenen Kategorien von Software-Produktmetriken wird in Anhang 11.2 erläutert.

2.3.2 Werkzeuge der Metriken-Extraktion

Werkzeuge der Extraktion von Software-Produktmetriken dienen zur automatisierten Berechnung und Bereitstellung ausgewählter Metriken. In der Regel unterstützt ein solches Tool lediglich die Konzepte eines Programmierparadigmas, sodass eine Unterscheidung zwischen diesen Tools dahingehend möglich ist. Ein Großteil der Extraktionswerkzeuge stellt dabei eine Reihe von Metriken für die Prinzipien der objektorientierten Programmierung bereit. Die Unterstützung anderer Programmierparadigmen kann hingegen nur durch sehr wenige Tools gewährleistet werden [9].

Viele dieser Werkzeuge implementieren Software-Produktmetriken abweichend von ihrer ursprünglichen Definition und Berechnungsgrundlage. Verschiedene Werk-

zeuge weisen deswegen unterschiedliche und inkonsistente Ergebnisse auf. Darüber hinaus sind diese Extraktionstools abhängig von der bereitgestellten Metriken-Menge und ihren unterstützten Programmiersprachen. Diese Werkzeuge lassen dabei nur durch hohen Aufwand hinsichtlich neuer Metriken und weiteren akzeptierten Programmiersprachen erweitern. Aufgrund dieser Gegebenheiten werden häufig individuelle Lösungen zur Extraktion bestimmter Software-Produktmetriken entworfen und implementiert [9].

Aktuelle Werkzeuge der Metriken-Extraktion entsprechen dadurch in der Regel nicht den aktuellen Anforderungen der Metrikenforschung. In der Praxis wird empfohlen, verschiedene Werkzeuge zur Erreichung von Zielen sowie zur Bewältigung spezifischer Herausforderungen zu verwenden. Im Rahmen dieser Forschungsarbeit werden zur Durchführung empirischer Studien zwei verschiedene Werkzeuge der Metriken-Extratkion verwendet. Dazu wird in Kapitel 2.3.2.1 ein Metriken-Tool namens Understand sowie in Kapitel 2.3.2.2 ein weiteres Werkzeug namens Analizo vorgestellt.

2.3.2.1 Understand

Understand ist ein von SciTools entwickeltes Werkzeug zur Metriken-Extraktion, welches zur Analyse von Quellcode verwendet werden kann. Es unterstützt eine Vielzahl von Sprachen wie beispielsweise Ada, Assembler, C, C++, C#, Delphi, Pascal, FORTRAN, JOVIAL, Java, Python und VHDL. Verschiedene Sprachen der Webentwicklung wie PHP, HTML, CSS, JavaScript, Typescript und XML werden ebenfalls unterstützt. Mithilfe dieser vielseitigen Unterstützung unterschiedlicher Programmiersprachen eignet sich Understand außerdem zur Analyse von Software-Projekten, welche mehrere Programmiersprachen verwenden [10].

Mittels Understand können über 100 Software-Produktmetriken bereitgestellt werden, welche die Eigenschaften des Quellcodes auf verschiedenen System-Ebenen erfassen. Der Großteil dieser Metriken können als Quantifizierungs- und Komplexitätsmetriken klassifiziert werden und basieren auf dem Konzept der objektorientierten Programmierung. Dabei können einige Metriken jedoch nur exklusiv für bestimmte Sprachen ermittelt werden [9]. Alle Metriken, welche in Understand verfügbar sind, werden in Anhang 11.3.1 beschrieben.

Zur Ausführung der Metriken-Extraktion stellt das Understand-Tool eine grafische Oberfläche bereit, über die statische Code-Analysen durchgeführt sowie Quellcode bearbeitet und restrukturiert werden kann. Mithilfe dieser grafischen Oberfläche können außerdem verschiedene Arten von Graphen wie beispielsweise Deklarationsgraphen, Hierarchiegraphen, Kontrollflussgraphen, UML-Klassengraphen,

Abhängigkeitsgraphen oder Baum-Karten erstellt werden. Diese Graphen können wiederum an individuelle Bedürfnisse angepasst werden. Darüber hinaus wird durch diese Oberfläche die Visualisierung von Ergebnissen ermöglicht [10].

Neben der grafischen Oberfläche existieren außerdem API-Schnittstellen, welche ebenfalls die Ausführung aller Funktionalitäten ermöglichen. Dazu existieren separate APIs für alle verschiedenen Graphen und unterstützten Programmiersprachen. Zusätzlich können alle Funktionalitäten durch entsprechende Befehle per Kommandozeile ausgeführt werden. Dadurch wird die Integration dieses Tools in die Prozesse der *Continuous Integration* und *Continuous Delivery* vereinfacht [10].

2.3.2.2 Analizo

Analizo ist ein Open-Source-Tool der Code-Analyse und -Visualisierung, welches die Programmiersprachen C, C++ und Java unterstützt. Die Unterstützung verschiedener Programmiersprachen innerhalb eines Projekts wird dabei ebenfalls ermöglicht. Mithilfe von Analizo können 15 verschiedene Metriken der Modul-Ebene und 7 verschiedene Metriken der Projekt-Ebene extrahiert werden [11]. Die meisten dieser Metriken können als Quantifizierungs- und Komplexitätsmetriken kategorisiert werden [9]. Alle Metriken, welche in Understand verfügbar sind, werden in Anhang 11.3.2 beschrieben.

Alle Funktionalitäten von Analizo sind als Kommandozeilenbefehle verfügbar und können in Docker-Containern integriert werden. Eine Extraktion von Metriken aus VCS-Repositories ist ebenfalls möglich. Dabei ist die Entwicklung von Metriken hinsichtlich von Änderungen für die gesamte Historie eines Projekts nachvollziehbar.

3 Grundlagen künstlicher neuronaler Netze

Künstliche neuronale Netze basieren auf den Vorgängen des zentralen Nervensystems höherer Lebewesen. Das menschliche Gehirn als Teil des zentralen Nervensystems ist die komplizierteste, bekannte Struktur, obwohl es nur aus einer Art von Zellen besteht [12]. Diese Zellen verfügen über die Fähigkeiten der Speicherung und Übertragung von Informationen und sind hochgradig untereinander verbunden. Mithilfe dieser komplexen Verknüpfungen von Nervenzellen können Signale empfangen und weitergeleitet werden. Eine Weiterleitung in Form eines Impulssignals findet in diesem Zusammenhang durch die Aufsummierung der durch Signale empfangenen Potenziale statt, sobald ein bestimmter Schwellenwert überschritten wurde [12]. Auf Grundlage dieser Eigenschaften können sich die einzelnen Zellen durch ihre gegebenen Fähigkeiten anpassen und Reaktionen erlernen.

Das Ziel künstlicher neuronaler Netze ist jedoch nicht die perfekte Modellierung des menschlichen Gehirns und seiner Funktionen. Stattdessen sollen mithilfe der Erkenntnisse über das menschliche Gehirn Annäherungen einer Funktion f^* ermittelt werden, welche über Eigenschaften statistischer Verallgemeinerungen verfügen [13]. Architekturen neuronaler Netze werden in vielen Bereichen zur Abschätzung einer Funktion $y = f^*(x)$ mittels der Abbildung der Eingabevariablen x über eine Zielfunktion f^* verwendet. Die beste Funktionsabschätzung $y = f(x; \theta)$ soll dabei durch einen Lernprozess ermittelt werden, indem die Parameter θ erlernt werden, welche die einzelnen Verbindungsgewichte symbolisieren.

Der Aufbau einzelner Neuronen und des gesamten Netzwerks wird in Kapitel 3.1 detaillierter beschrieben. In Kapitel 3.2 werden die Lernverfahren künstlicher neuronaler Netze konkreter charakterisiert und in Kapitel 3.3 häufig verwendete Aktivierungsfunktionen vorgestellt. Methoden der Normalisierung und Regularisierung werden in Kapitel 3.4 erläutert.

3.1 Architektur künstlicher neuronaler Netze

Künstliche neuronale Netzwerke werden grundsätzlich anhand ihrer einzelnen Neuronen und deren Verbindungen sowie ihrer zugrunde liegenden Lerntechnik charakterisiert [12]. Während in Kapitel 3.2 Lerntechniken thematisiert werden, beschreibt Kapitel 3.1.1 den Aufbau einzelner Neuronen und Kapitel 3.1.2 die Architektur des gesamten Netzes.

3.1.1 Aufbau eines Neurons

Zur Beschreibung neuronaler Netze wurde bereits im Jahr 1943 ein erster Ansatz durch McCulloh und Pitts entworfen [12]. Neuronen gelten als die Grundbausteine dieses Modells und werden als Knoten mit Verbindungen variabler Gewichte bezeichnet. Alle eingehenden Signale eines Knotens werden mit spezifischen, variablen Gewichten multipliziert und das Ergebnis einzelner Berechnungen als Zwischenresultate bereitgestellt. Diese Zwischenresultate werden unter Verwendung von Aktivierungsfunktionen und in Abhängigkeit individueller Schwellenwerte weitergeleitet.

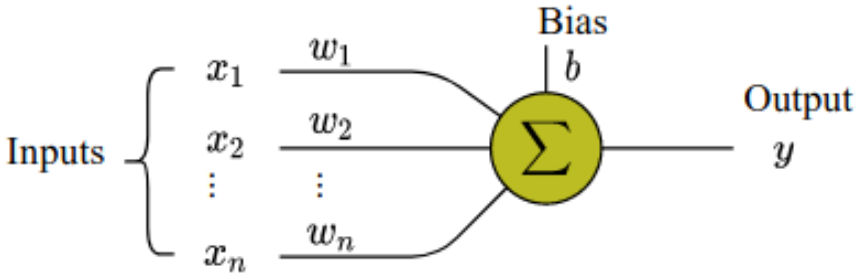


Abbildung 2: Aufbau eines einzelnen Neurons [14]

Abbildung 2 zeigt den schematischen Aufbau eines einzelnen Neurons, welches über seinen Aktivierungszustand y zu einem bestimmten Zeitpunkt definiert werden kann. In dem Beispiel verfügt das Neuron über n verschiedene Eingangsverbindungen, über die die Werte $x_1, x_2, ..., x_n$ empfangen werden. Diesen Verbindungen werden spezifische, unabhängige Gewichtungen $w_1, w_2, ..., w_n$ zugewiesen. Zur Berechnung des Aktivierungswertes werden alle Eingangswerte x_i mit ihren entsprechenden Gewichten w_i multipliziert und anschließend miteinander addiert. Zusätzlich wird diese gewichtete Summe mit einem Bias b addiert, wodurch die letztendliche Formel zur Berechnung des Aktivierungswertes $\Sigma(x) = b + \sum_{i=1}^{n} w_i * x_i$ lautet [13]. Auf Grundlage der Berechnung dieses Wertes kann unter Verwendung einer ausgewählten Aktivierungsfunktion ϕ der Aktivierungszustand $y = \phi(\Sigma(x))$ ermittelt werden. Die Aktivierung eines Neurons wird also letztendlich durch die Formel $y = \phi(\Sigma(b + \sum_{i=1}^{n} w_i * x_i))$ berechnet, dessen Ergebnis bei entsprechender Aktivierung letztendlich weitergeleitet wird [13].

Für die Repräsentation der Aktivierungszustände werden Werte eines kontinuierlichen Bereichs ausgewählt [12]. Diese Darstellung ist gut und intuitiv nachvollziehbar für die Abbildung dichotomer Variablen. Zur Abbildung kontinuierlicher Werte innerhalb dieses Bereichs ist jedoch eine Normalisierung beispielsweise durch die sogenannte z-Transformation erforderlich. Andernfalls verfügen kontinuierliche Werte aufgrund ihres größeren Wertebereichs in den Lernprozessen des neuronalen Netzes über mehr Einfluss als die binären Werte dichotomer Variablen. Die Abbildung

nominaler Werte erfolgt hingegen durch mehrere Neuronen, welche jeweils einzelne nominale Werte als dichotome Variablen repräsentieren [12].

3.1.2 Topologie künstlicher neuronaler Netze

Alle Neuronen eines neuronalen Netzes werden in drei verschiedene Arten von Schichten strukturiert, welche als Eingabe-, verborgene und Ausgabeschichten bezeichnet werden [12]. Abbildung 3 zeigt einen beispielhaften Aufbau eines solchen Netzes, welches über neun Neuronen und drei Schichten verfügt. Die erste Schicht beinhaltet drei Neuronen und repräsentiert die Eingabeschicht. In der zweiten Schicht stellt eine verborgene Schicht dar und enthält vier Neuronen, die durch Verbindungen mit jedem Neuron ihrer vorherigen und folgenden Schicht verbunden sind. Die dritte Schicht besteht aus zwei Neuronen und verkörpert die Ausgabeschicht.

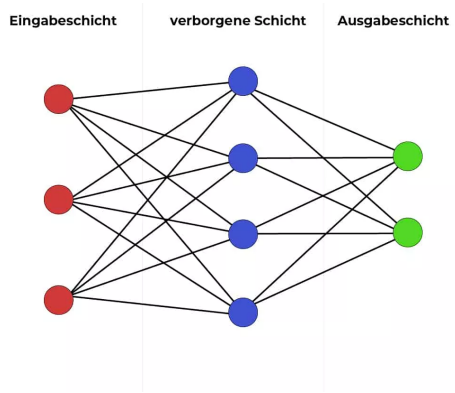


Abbildung 3: Architektur eines künstlichen neuronalen Netzes [15]

Während jedes neuronale Netz nur über jeweils eine Eingabe- und Ausgabeschicht verfügt, können zwischen diesen beiden Komponenten mehrere verborgene Schichten existieren. Durch die diese Aufteilung verfügen Neuronen über unterschiedliche Funktionalitäten. Der Informationsfluss dieses Modells beginnt in der Eingabeschicht zur Bereitstellung der Informationen aller Eingabevariablen. Neuronen der Eingabeschicht empfangen die Werte externer Variablen zur Repräsentation einzelner Merkmale. Die Anzahl der Eingabeneuronen entspricht in der Regel der Anzahl an Eingabevariablen. Zur Erhöhung der Selektivität und Invarianz von Repräsentationen werden die zugrunde liegenden Eingabewerte in der Regel transformiert. Auf Grundlage der Transformationen dieser Eingaben werden mehrere nichtlineare Schichten extrem komplizierter Funktionen entwickelt. Diese Funktionen sollen gegenüber kleinsten Details sensitiv und unempfindlich gegenüber größeren, irrelevanten Veränderungen sein [16]. Dazu werden die transformierten Werte dieser Variablen an mindestens ein Neuron einer verborgenen Schicht weitergeleitet.

Zur Definition der Zielfunktion f^* werden diese Informationen durch Berechnungen innerhalb der verborgenen Schichten ausgewertet und weitergeleitet [13]. Neuronen verborgener Schichten dienen zur Verarbeitung empfangener Informationen sowie zur anschließenden internen Weiterleitung berechneter Zwischenergebnisse. Innerhalb verborgener Schichten besteht grundsätzlich eine strukturelle Verknüpfung, sodass jedes Neuron einer verborgenen Schicht mit jedem Neuron der folgenden Schicht verbunden ist. Verbindungen können dabei durch Gewichtungen des Faktors null temporär unterbrochen werden [12]. Die letzte verborgene Schicht leitet ihre Ergebnisse wiederum an Neuronen der Ausgabeschicht unter Verwendung einer nichtlinearen Funktion weiter. Die finalen Ergebnisse werden somit letztendlich über die Neuronen der Ausgabeschicht bereitgestellt.

Neuronale Netze mit struktureller Verknüpfung benachbarter Schichten ohne rückläufigen Verbindungen zwischen den Zwischenergebnissen einzelner Neuronen und den Neuronen vorheriger Schichten werden als Feedforward-Netzwerke bezeichnet. Ein solches Netz kann entweder als mehrschichtiger Stapel einfacher Module zur Berechnung nichtlinearer Abbildungen zwischen Eingabe- und Ausgabevariablen auf Grundlage des Lernens repräsentiert werden [16]. Eine Darstellung als eine Zusammensetzung verschiedener Funktionen als gerichteter, azyklischer Graph ist ebenfalls möglich [13]. Die Tiefe dieses Modells wird über die maximale Länge an Verkettungen zwischen Eingabe- und Ausgabeschicht ermittelt werden. Die Breite des Modells wird wiederum über die Dimensionalität der versteckten Schichten beschrieben [13].

Eine optimale Anzahl aller verwendeten Neuronen und Schichten ist von der jeweiligen Problemstellung abhängig. Es sollte stets ein guter Ausgleich zwischen tolerablen Ergebnissen und der Beeinträchtigung der Generalisierungsfähigkeit angestrebt werden. Zur Ermittlung der optimalen Netzwerkstruktur existiert allerdings kein deterministischer Ansatz, sodass ein Testen verschiedener Netzwerktopolgien in der Praxis erforderlich ist [12].

3.2 Lernverfahren künstlicher neuronaler Netze

Ergebnisse neuronaler Netze unterliegen der Anforderung der Unempfindlichkeit gegenüber irrelevanten Variationen der Eingabe bei gleichzeitiger Sensitivität gegenüber kleinen, relevanten Details. Diese Herausforderung wird als Sleketivitäts-Invarianz-Dilemma bezeichnet und erfordert eine gute Merkmalsextraktion zur Realisierung oberflächlicher Klassifikatoren [16]. Mittels der Extraktion von Merkmalen werden Aspekte von Beispielen als relevante Merkmale zur Bestimmung von Zielgrößen mit Invarianz bezüglich irrelevanten Aspekten selektiert.

Generische, nicht-lineare Merkmale eignen sich in diesem Zusammenhang zur

Erhöhung der Performanz von Klassifikatoren bei einem bestehenden Risiko der fehlenden Verallgemeinerung des Lerners. Basierend auf umfangreichen Fähigkeiten und Expertenwissen können als konventioneller Ansatz geeignete Merkmalsextraktoren eigenständig konzipiert werden. Dieser zusätzliche Aufwand kann jedoch durch automatisches Lernen unter Verwendung einer verallgemeinerten Lernprozedur vermieden werden [16].

In Kapitel 3.2.1 wird der verallgemeinerte Lernprozess des überwachten Lernens für künstliche neuronale Netze vorgestellt. Kapitel 3.2.2 thematisiert ein Verfahren zur Optimierung von Parametern durch die Berechnung von Gradienten unter Verwendung des Backpropagation-Algorithmus.

3.2.1 Überwachtes Lernen künstlicher neuronaler Netze

Die am häufigsten verwendete Methode für den Lernprozess neuronaler Netze, welcher auch als Training bezeichnet wird, ist das überwachte Lernen [16]. Dazu wird als Datensatz eine Sammlung an Beispielen erstellt, welche für jedes Beispiel die repräsentativen Eingabewerte x enthält. Zusätzlich werden für alle Beispiele die erwarteten Ausgaben y in Form von entsprechenden Werten der Zielgrößen annotiert.

In der Praxis werden diese Beispieldaten in drei verschiedene Mengen unterteilt. Zum einen wird ein Trainingsdatensatz definiert, welcher zur Erlernung optimaler, interner Gewichtungen des neuronalen Netzes verwendet wird. Dabei soll der Vorhersagefehler bei mehrfacher Wiederholung des Trainingsprozesses für diesen Datensatz minimiert werden. Zum anderen wird ein Validationsdatensatz konstruiert, welcher das Training überwachen und diesen Prozess bei optimaler Generalisierung stoppen soll. Außerdem wird ein Testdatensatz erstellt, welcher die Performanz von Vorhersagen nach Abschluss des Trainings auswerten soll und dem künstlichen neuronalen Netz während des Trainings unbekannt ist [16].

Um Ergebnisse zu erhalten, die einen tolerablen Fehler aufweisen, wird eine Gewichtungsmatrix bestehend aus den Gewichten einzelner Verbindungen erzeugt, auf der Lernregeln angewendet werden [12]. Diese Lernregeln modifizieren im Rahmen wiederholter Auswertung verschiedener Beispiel-Elemente die erzeugte Gewichtungsmatrix, um den Fehler zwischen erwarteter und tatsächlicher Ausgabe sukzessive zu minimieren. Als Betrag der Veränderung von Gewichtungen in korrekte Richtungen werden Lernkonstanten oder Lernraten verwendet. Dabei ist die Auswahl geeigneter Konstanten entscheidend für die resultierende Performanz eines Netzwerks [12]. Durch zu hoch ausgewählte Konstanten kann eine Optimierung durch Anpassung der Gewichte ausschließlich auf den aktuell untersuchten Trainingsbeispielen erfolgen. Dadurch kann eine zu hohe Anpassung für Extremfälle ohne Filterung

geeigneter Mittelwerte erfolgen, wodurch letztendlich die Generalisierungsfähigkeit beeinträchtigt wird. Eine zu niedrige Lernrate kann jedoch den gesamten Prozess verzögern, sodass eine geeignete Optimierung nach Verarbeitung aller Beispieldaten möglicherweise nicht ermittelt werden konnte.

Neben der Auswahl geeigneter Lernparameter existieren weitere Herausforderungen während des Trainings künstlicher neuronaler Netze. Die Menge der Beispieldaten verfügt in der Regel über keine optimale Grundgesamtheit, wodurch eine Garantie zur Identifikation optimaler Interpolationen durch Trainingsalgorithmen fehlt [12]. Darüber hinaus besteht das Risiko der Überanpassung als Folge zu genauen Lernens eines kleinen Trainingsdatensatzes oder bei zu großer Anzahl verwendeter Neuronen. Durch die Überanpassung eines Netzwerks werden sehr gute Ergebnisse während des Trainingsprozesses erzielt, während schlechte Ergebnisse für unbekannte Daten aufgrund fehlender Generalisierung entstehen. Daher wird der Validationsdatensatz zur regelmäßigen Überprüfung des Trainings eingesetzt. Dieser Datensatz stellt eine vom Trainingsdatensatz unabhängige Menge dar und wird zur Identifikation des minimalen Fehlers bei geeigneter Generalisierung verwendet.

3.2.2 Gradientenbasierte Optimierung durch Backpropagation

Der Trainings-Algorithmus künstlicher neuronaler Netze basiert in der Regel auf der Berechnung von Gradienten zur Reduzierung einer Verlustfunktion $\mathcal{L}(\hat{y}, y)$ [13]. In diesem Kontext werden Verlustfunktionen als Distanzmaß des Fehlers zwischen den ausgegebenen Zielgrößen $\hat{y} = f(x)$ und zuvor festgelegten Zielgrößen y definiert. Dieser Prozess kann auch als gradientenbasiertes Lernen bezeichnet werden und ist für neuronale Netze ähnlich zu jedem anderen Modell des maschinellen Lernens, welches Gradientenabstieg verwendet. Da das parametrische Model in den meisten Fällen einer Verteilung $p(y|x;\theta)$ unterliegt, wird in diesem Zusammenhang das Prinzip des Maximum-Likelihood angewandt. Dadurch wird als Verlustfunktion die Kreuzentropie zwischen Trainingsdaten und der Vorhersage des Modells verwendet [13].

In einigen anderen Fällen wird wiederum ein vereinfachter Ansatz zur Definition einer Verlustfunktion verwendet. Anstatt der Vorhersage vollständiger Wahrscheinlichkeitsverteilungen können einige Statistiken der Ausgabe y bedingt durch die Eingabevariablen x vorhergesagt werden [13]. Mithilfe spezifischer Verlustfunktionen können für solche Schätzungen entsprechende Prädiktoren trainiert werden. Mögliche Verlustfunktionen sind in diesem Kontext zum Beispiel der mittlere absolute Fehler oder der mittlere quadratische Fehler.

Die Reduzierung der Verlustfunktion $\mathcal{L}(\hat{y}, y) = \mathcal{L}(f_\theta(x), y)$ stellt im Rahmen dieses Lernprozesses ein Optimierungsproblem $\min_\theta \mathcal{L}(f_\theta(x), y)$ dar. Zur Berechnung

der Gradienten aller Verbindungsgewichte wird dementsprechend die erste Ableitung der Verlustfunktion $\frac{\delta J}{\delta \theta}$ benötigt. Dieser Vektor beschreibt die Erhöhung und Reduzierung des gesamten Fehlers bei Veränderung der einzelnen Verbindungsgewichte [13].

In der Praxis wird für dieses Verfahren meistens der *stochastische Gradientenabstieg* (*stochasitc gradient descend*) verwendet [16]. Zu Beginn des Trainings werden für gewöhnlich alle Gewichte mit geringen, zufälligen positiven Werten initialisiert. Während des Trainings werden die Eingabevektoren x einiger Beispiele dem neuronalen Netz zur Ermittlung der Ausgabe $\hat{y}$ sowie zur Berechnung ihres Fehlers $\mathcal{L}(\hat{y}, y)$ bereitgestellt.

Zur Berechnung der Gradienten und der Anpassung von Verbindungsgewichten wird das Verfahren der Backpropagation angewandt, bei dem zur Minimierung des Ausgabefehlers ein rückläufiges Korrektursignal verwendet wird [13]. Dieses Verfahren gilt als eine Verallgemeinerung Hebb-Regel, welche die Verbindungen zwischen zwei Neuronen bei gleichzeitiger Aktivierung verstärkt [12]. Durch die praktische Anwendung der Kettenregel von Ableitungen können die Gradienten auf Grundlage der Zielfunktion und der Gewichte des neuronalen Netzes berechnet werden. Dabei wird das Korrektursignal rekursiv durch alle Neuronen beginnend bei der Ausgabeschicht rückwärts durch alle verborgenen Schichten bis hin zur Eingabeschicht verarbeitet, um alle Verbindungsgewichte anzupassen.

Nach jedem Trainingsschritt werden auf Grundlage der Ergebnisse alle Gewichtsvektoren unter Verwendung der Backpropagation in die entgegengesetzte Richtung des berechneten Gradientenvektors angepasst [12]. Der negative Gradientenvektor stellt dabei einen Indikator der Richtung zur Annäherung an ein Minimum mit einem durchschnittlich geringen Fehler dar. Der Backpropagation-Algorithmus ist wiederum eine praktische Anwendung der Kettenregel von Ableitungen zur Berechnung der Gradienten der Zielfunktion hinsichtlich der Gewichte des neuronalen Netzwerks [16].

Nach Abschluss eines solchen Trainingsschrittes stellt die definierte Zielfunktion den Durchschnitt aller verwendeten Trainingsbeispiele im multidimensionalen Raum aller Werte von Verbindungsgewichten dar [16]. Dieser Prozess wird mit vielen kleinen Teilmengen der Trainingsmenge wiederholt, bis die durchschnittliche Zielfunktion nicht weiter reduziert werden kann. Dementsprechend werden alle Gewichtungen nach Auswertung einer Teilmenge zur schrittweisen Minimierung des gesamten Netzwerks angepasst. Nach Beendigung des Trainingsprozesses wird die Performanz des gesamten Systems durch den Testdatensatz überprüft. Dabei wird vor allem die Verallgemeinerungsfähigkeit des Netzes analysiert.

3.3 Aktivierungsfunktionen künstlicher neuronaler Netze

Aktivierungsfunktionen sind ein wesentlicher Bestandteil zur Modellierung der Nichtlinearität künstlicher neuronaler Netze. Ohne solche Aktivierungsfunktionen wären künstliche neuronale Netze lediglich lineare Modelle ohne Fähigkeiten zur Erlernung komplexer Muster. Mithilfe von Aktivierungsfunktionen kann auf Grundlage der gewichteten Summe aller empfangenen Eingabewerte bestimmt werden, ob eine Aktivierung stattfindet. Dazu wird eine nichtlineare Funktion auf der gewichteten Summe aller Eingaben ausgeführt, um die entsprechende Ausgabe eines Neurons zu berechnen.

Aktivierungsfunktionen der Ausgabeschichten werden anhand der Repräsentation ihrer Ausgabe selektiert [13]. Die Ergebnisse von Verlustfunktionen werden wiederum durch die Auswahl einer verwendeten Aktivierungsfunktion der Ausgabeschicht bedingt. Man unterscheidet grundsätzlich zwischen den binären und linearen Ausgaben sowie den Ausgaben mehrklassiger Klassifikationen. Je nach Anwendungsgebiet werden dafür zum Beispiel die Aktivierungsfunktionen Linear, Sigmoid, Tangens Hyperbolicus oder Softmax verwendet [13]. Diese Aktivierungsfunktionen können außerdem nicht nur für Neuronen der Ausgabeschicht verwendet werden. Neben einer Vielzahl weiterer Aktivierungsfunktionen wie der Rectified Linear Unit können diese Funktionen ebenfalls für Neuronen verborgener Schichten eingesetzt werden.

3.3.1 Lineare Aktivierungsfunktion

Die lineare Aktivierungsfunktion wird in Ausgabeschichten zur Erzeugung eines linearen Ausgabevektors $\hat{y} = W^T * h + b$ basierend auf affiner Transformation ohne Nichtlinearität verwendet [13]. Dabei repräsentiert der Vektor h verborgene Merkmale, welche die Ausgaben der vorherigen verborgenen Schicht repräsentieren und der Vektor W die jeweiligen spezifischen Verbindungsgewichte. Ein Beispiel einer linearen Aktivierungsfunktion wird in Abbildung 4 dargestellt.

Häufig werden lineare Aktivierungsfunktionen zur Berechnung des Mittelwerts bedingter Gauß'scher Verteilungen $p(y|x) = \mathcal{N}(y; \hat{y}, I)$ verwendet [13]. Die Maximierung des Log-Likelihood ist in diesem Fall äquivalent zur Minimierung des mittleren quadratischen Fehlers. Da lineare Aktivierungsfunktionen über keine Sättigung verfügen, bestehen ebenso wenige Schwierigkeiten bezüglich der Verwendung gradientenbasierter Optimierungen. In Kombination mit linearen Aktivierungsfunktionen kann dementsprechend eine Vielzahl von Optimierungsalgorithmen verwendet werden.

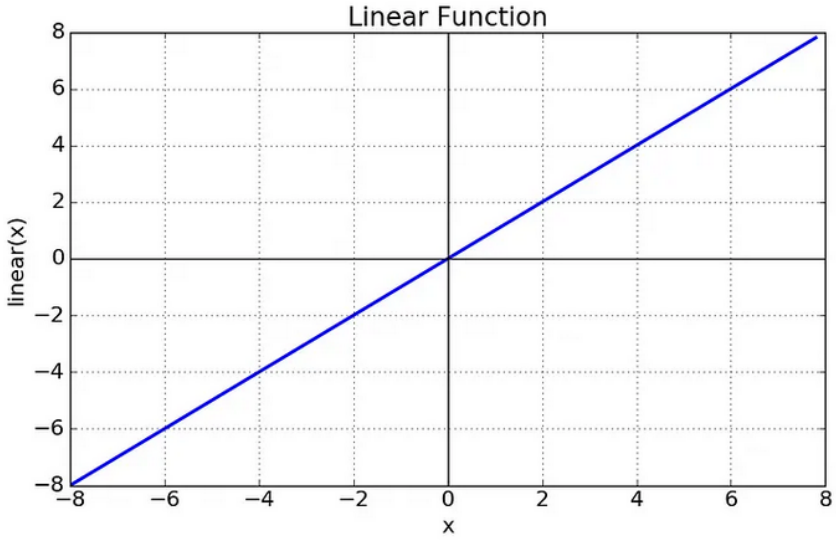


Abbildung 4: Darstellung einer linearen Aktivierungsfunktion [17]

3.3.2 Sigmoid-Funktion

Die Sigmoid-Funktion dient zur binären Klassifizierung durch die Vorhersage des Wertes einer binären Variable y. Unter Verwendung des Ansatzes des Maximum-Likelihood wird für y eine Bernoulli-Verteilung bedingt durch x definiert. Durch die Definition der Bernoulli-Verteilung als einzelne Zahl im Intervall $[0, 1]$ benötigt das künstliche neuronale Netz zur Vorhersage die Berechnung des Terms $P(y = 1|x)$ [13].

Zur Bereitstellung einer gültigen Wahrscheinlichkeit kann beispielsweise eine lineare Aktivierungsfunktion inklusive eines Schwellenwerts verwendet werden. Die Vorhersage basiert unter Verwendung dieses Ansatzes auf der folgenden Formel $P(y = 1|x) = \max\{0, \min\{1, w^T * h + b\}\}$. Das Training mittels Gradientenabstieg ist in diesem Fall jedoch nicht sehr effektiv. Sobald $z = w^T * h + b$ einen Wert außerhalb des vorgegebenen Intervalls erreicht, lautet der Gradient hinsichtlich seiner Parameter 0. Auf Grundlage eines solchen Gradienten verfügt der Lernalgorithmus über keine Anhaltspunkte zur Verbesserung entsprechender Parameter [13].

Ein besserer Ansatz zur Bereitstellung stärkerer Gradienten basiert auf der Kombination von Sigmoid-Funktionen $\sigma(z) = \frac{1}{1+\exp(-z)}$ mit Maximum-Likelihood. Diese Aktivierungsfunktion wird durch die Formel $\hat{y} = \sigma(w^T * h + b)$ repräsentiert, bei der σ die logistische Sigmoid-Funktion darstellt und zur Konvertierung der linearen Ergebnisse in Wahrscheinlichkeiten dient [13]. Die Darstellung der Sigmoid-Funktion wird beispielhaft in Abbildung 5 visualisiert.

Mittels dieser Funktion wird eine nicht normalisierte Wahrscheinlichkeitsverteilung $\tilde{P}(y)$ konstruiert, welche nicht zu 1 aufsummiert werden kann. Zur Erhaltung

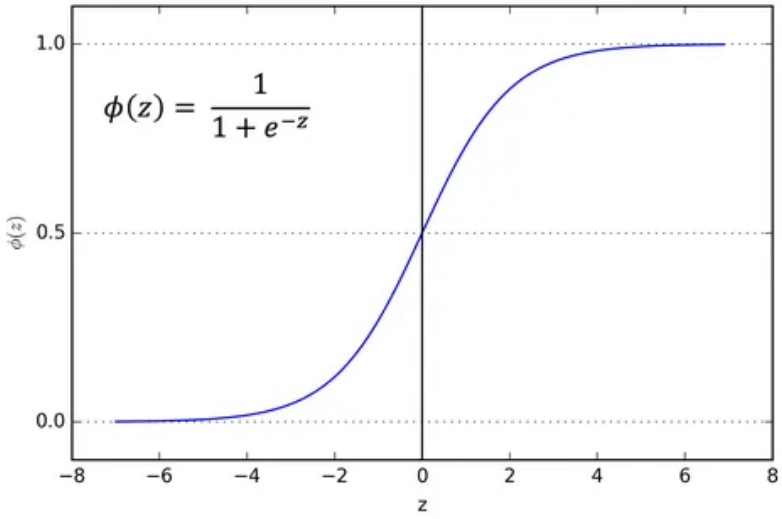


Abbildung 5: Darstellung einer Sigmoid-Aktivierungsfunktion [17]

einer gültigen Wahrscheinlichkeitsverteilung wird diese Summe anschließend durch eine angemessene Konstante dividiert. Mittels Exponentierung der nicht normalisierten Log-Wahrscheinlichkeit und anschließender Normalisierung ergibt sich wiederum eine Bernoulli-Verteilung unter Anwendung der sigmoid'schen Transformation. Dabei ist der Logarithmus von Sigmoid $P(y) = \sigma((2y-1) * z)$ stets definiert und endlich aufgrund der Rückgabe von Werten innerhalb des Intervalls $(0, 1)$ [13].

In Kombination mit einer Verlustfunktion des Maximum-Likelihood findet nur dann eine Sättigung statt, sobald das Modell die richtige Antwort $y = 1$ ermittelt und z einen sehr hohen positiven Wert repräsentiert oder die richtige Antwort $y = 0$ ist und z einen sehr niedrigen negativen Wert darstellt. Unter Verwendung anderer Verlustfunktionen wie den mittleren quadratischen Fehler kann eine Sättigung von $\sigma(z)$ jederzeit auftreten. Dabei resultiert eine Sättigung der Aktivierungsfunktion zu 0 bei einem sehr niedrigen negativen Wert von z und eine Sättigung zu 1 bei einem sehr hohen positiven Wert von z. Der Einsatz von Sigmoid-Funktionen eignet sich daher weniger als Aktivierungsfunktionen verborgener Schichten [13]. Stattdessen wird die Verwendung in Ausgabeschichten in Kombination mit angemessenen Verlustfunktionen empfohlen, welche die Sättigung ausgleichen.

3.3.3 Tangens Hyperbolicus

Die Aktivierungsfunktion des hyperbolischen Tangens basiert auf der Formel $g(z) = \tanh(z)$ mit $z = w^T * h + b$ als gewichtete Summe aller Eingänge des jeweiligen Neurons [13]. Diese Funktion verfügt über eine enge Verwandtschaft zur sigmoid'schen Aktivierungsfunktion aufgrund der Eigenschaft $\tanh(z) = 2 * \sigma(2z) - 1$, welches

durch Abbildung 6 verdeutlicht werden soll.

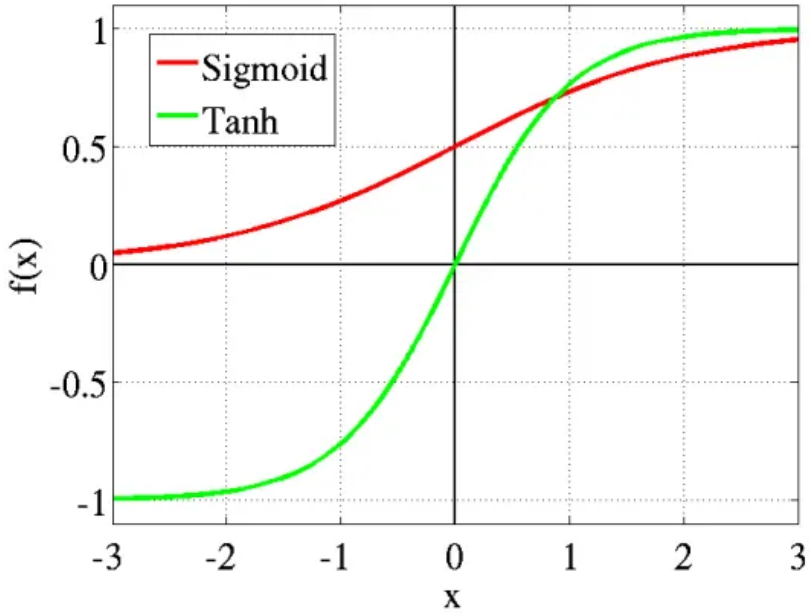


Abbildung 6: Darstellung der Aktivierungsfunktion des Tangens Hyperbolicus [17]

Aufgrund der Sättigungseigenschaften der Sigmoid-Funktion und einer höheren Performanz ist die Aktivierungsfunktion des hyperbolischen Tangens in vielen Anwendungsfällen besser geeignet [13]. Darüber hinaus ähnelt diese Funktion eher einer Identitätsfunktion, da $tanh(0) = 0$ ergibt, während $\sigma(0) = \frac{1}{2}$ ist.

3.3.4 Softmax-Funktion

Die Softmax-Funktion wird häufig zur Darstellung von Wahrscheinlichkeitsverteilungen diskreter Variablen mit n möglichen Werten verwendet. Diese Funktion stellt eine Verallgemeinerung der Sigmoid-Funktion dar, in der die Wahrscheinlichkeitsverteilung einer binären Variable repräsentiert wird [13]. Dazu wird ein Vektor $\hat{y}$ mit der Eigenschaft $\hat{y}_i = P(y = i|x)$ erzeugt. Die Werte einzelner Elemente $\hat{y}_i$ müssen dabei nicht zwingend in einem Wertebereich zwischen 0 und 1 liegen. Stattdessen muss der gesamte Vektor aufsummiert den Wert 1 ergeben, sodass gültige Wahrscheinlichkeitsverteilungen repräsentiert werden können.

Anschließend kann der gleiche Ansatz zur Berechnung von Bernoulli-Verteilungen für die Verallgemeinerung von Multinoulli-Verteilungen verwendet werden. Zur Vorhersage der nicht normalisierten Log-Wahrscheinlichkeiten wird eine lineare Schicht als Vektor $z = W^T * h + b$ mit $z_i = \log \tilde{P}(y = i|x)$ definiert [13]. Nach Exponentierung der Funktion und Normalisierung von z lautet die Formel der Softmax-Funktion $softmax(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$.

Eine Kombination der Softmax-Funktion mit der Verlustfunktion des Maximum-Likelihood eignet sich aufgrund ähnlicher Eigenschaften bezüglich der Sigmoid-Funktion [13]. Da eine Softmax-Funktion jedoch über verschiedene Ausgabewerte verfügt, erreicht diese Funktion eine Sättigung, sobald die Differenz zwischen den Ausgabewerten sehr hoch ist. Daher eignen sich andere Optimierungsalgorithmen nur bei einer Invertierung der Sättigungseigenschaft dieser Aktivierungsfunktion.

3.3.5 Rectified Linear Unit

Die Aktivierungsfunktion der Rectified Linear Unit ($ReLU$) lautet $g(z) = \max\{0, z\}$ [13]. Aufgrund der Ähnlichkeit zu linearen Aktivierungsfunktionen lassen sich diese Funktionen leicht optimieren. Der einzige Unterschied beruht auf der Ausgabe von 0 für die Hälfte des Wertebereichs, wodurch sich große und konsistente Gradienten ergeben. Eine Darstellung dieser Aktivierungsfunktion wird in Abbildung 7 repräsentiert.

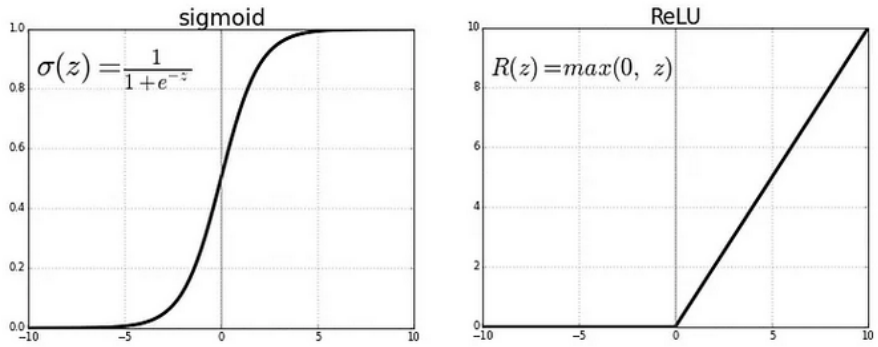


Abbildung 7: Darstellung einer ReLU-Aktivierungsfunktion [17]

In den meisten Fällen wird die ReLU-Aktivierungsfunktion basierend auf einer affinen Transformation $z = W^T * h + b$ angewendet. Dabei werden häufig die Parameter b aller Elemente mit geringen positiven Werten wie beispielsweise mit 0.1 initialisiert, um eine initiale Aktivierung der meisten Eingaben während des Trainings zu erzielen [13]. Ein Nachteil dieser Aktivierungsfunktion ist die fehlende Fähigkeiten des Lernens durch gradientenbasierte Methoden bei einem Aktivierungswert von 0. Basierend auf diesem grundlegenden Ansatz existieren jedoch verschiedene verallgemeinerte Ausprägungen wie beispielsweise *Absolute value rectification*, *leaky ReLU* oder *PReLU*.

3.4 Regularisierung

Ein zentrales Problem des maschinellen Lernens ist es, sowohl für Trainingsdaten als auch für neue, unbekannte Eingaben eine gute Performanz der Algorithmen zu gewährleisten [13]. Durch die Anwendung verschiedener Strategien als Modifikationen von Lern-Algorithmen soll dazu der Verallgemeinerungsfehler reduziert werden. Diese Strategien werden wiederum unter dem Begriff der Regularisierung zusammengefasst, denen verschiedene Ansätze zugrunde liegen.

Einerseits existieren Strategien zur Einschränkung von Parameterwerten. Andererseits bestehen Strategien auf der Grundlage zusätzlicher Terme innerhalb der Zielfunktionen, um weiche Einschränkungen von Parameterwerten zu definieren. Durch eine sorgfältige Auswahl entsprechender Elemente kann dabei die Performanz eines küsntlichen neuronalen Netzes auf der Testmenge erhöht werden [13]. Dabei können verwendete Einschränkungen auch spezifisches Vorwissen verschlüsseln oder generische Präferenzen einfacherer Modellklassen zur Förderung der Verallgemeinerung aufweisen. Andere Formen der Regularisierung können auch durch sogenannte Ensemble-Methoden als Kombinationen mehrerer Hypothesen zur Beschreibung von Trainingsdaten.

Die meisten Regularisierungsstrategien basieren auf der Regularisierung eines Schätzers, welche einen Ausgleich zwischen einem erhöhten Bias und reduzierter Varianz anstreben [13]. Effektive Regularisierer erzeugen dabei einen profitablen Ausgleich durch signifikante Reduzierung der Varianz, ohne den Bias übermäßig zu erhöhen. Im Rahmen dieses Kapitels werden die Regularisierungsstrategien des Dropouts in Kapitel 3.4.1 und der Batch-Normalisierung in Kapitel 3.4.2 vorgestellt.

3.4.1 Dropout

Dropout ist eine rechnerisch kostengünstige und leistungsstarke Methode zur Regularisierung einer breiten Modellfamilie. Mithilfe dieser Methode können Bagging-Strategien auf einer Sammlung sehr vieler und großer neuronaler Netze praktiziert werden [13]. Bagging stellt eine Technik der Regularisierung zur Reduzierung des Verallgemeinerungsfehlers durch die Kombination mehrerer Modelle dar. Dabei werden k unterschiedliche Modelle auf der Grundlage von k verschiedenen Datensätzen trainiert, welche durch Anwendung von Sampling basierend auf der Trainingsmenge erstellt worden. Jedes Modell wertet anschließend jedes Testbeispiel aus, um im Rahmen einzelner Abstimmungen die letztendlichen Ausgaben der Modell-Sammlung zu ermitteln.

Dieser Ansatz ist jedoch für viele große neuronale Netze aufgrund der hohen Ko-

sten des Trainings und der Auswertung hinsichtlich der Laufzeit und des Speicherverbrauchs unpraktisch [13]. Dropout-Strategien stellen in diesem Zusammenhang eine kostengünstige Alternative des Trainings zur Bewertung einer Sammlung exponentiell vieler neuronaler Netze dar. Dazu werden während des Trainings eine Sammlung aller formbaren Unternetzwerke verwendet, in denen Neuronen aus dem zugrundeliegenden Basis-Netzwerk entfernt werden, welche nicht zur Ausgabeschicht gehören. In Abbildung 8 wird ein Beispiel zur Erzeugung dieser Unternetzwerke vorgestellt. Das Basis-Netzwerk besteht in diesem Fall aus drei Schichten, welche insgesamt über fünf Neuronen verfügen. Da nur ein Neuron in der Ausgabeschicht enthalten ist, können durch die potenzielle Entfernung der restlichen Neuronen insgesamt 16 verschiedene Unternetzwerke konstruiert werden. In der Praxis wird dabei für das effektive Entfernen von Neuronen aus den Unternetzwerken ihre jeweiligen Ausgaben mit dem Wert 0 multipliziert.

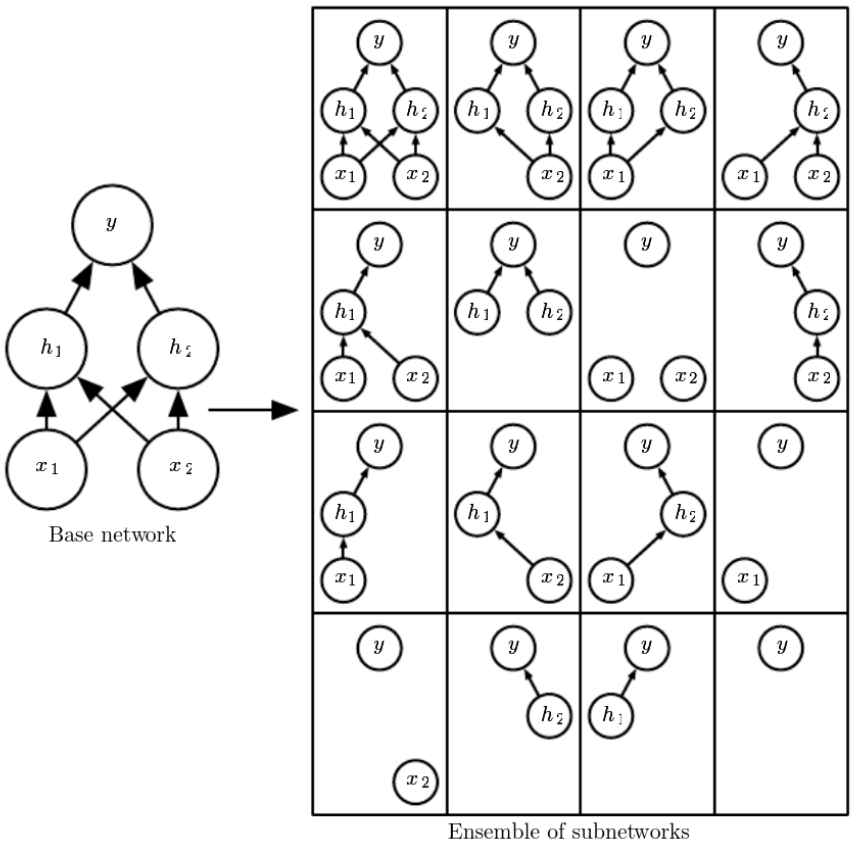


Abbildung 8: Konstruktion von Unternetzwerken anhand eines Basis-Netzwerks [13]

Die Anwendung der Dropout-Strategie beruht auf der Grundlage von minibatchbasierenden Lernalgorithmen, die über kurze Trainingsschritte verfügen [13]. Dazu eignet sich beispielsweise der stochastische Gradientenabstieg. Für jedes Beispiel eines Minibatches wird eine Binärmaske für alle Neuronen der Eingabeschichten sowie

der verborgenen Schichten erstellt. Diese Maske für dabei von jedem Neuron unabhängig voneinander verarbeitet, wobei der Wert 1 für die Berücksichtigung eines Neurons und der Wert 0 für das Entfernen eines Neurons innerhalb eines Unternetzwerks steht. Die Wahrscheinlichkeit des Maskenwerts 1 wird vor Trainingsbeginn durch einen Hyperparameter festgelegt und hängt dementsprechend nicht von einer Funktion der aktuellen Werte von Modellparametern oder der Eingabebeispiele ab. Typischerweise beträgt die Wahrscheinlichkeit des Maksenwerts 1 für Neuronen der Eingabeschicht 0.8 und für Neuronen der verborgenen Schicht 0.5 [13]. Auf Grundlage dieser Binärmaske findet die Forward-Propagation sowie die Backpropagation und die Anpassung der Modellparameter unter Verwendung wie gewöhnlich statt.

Dadurch stellt die Dropout-Strategie während des Trainings pro Beispiel und Aktualisierung zur Erzeugung von n zufälligen binären Zahlen und ihrer Multiplikation mit ihrem jeweiligen Aktivierungszustand eine rechnerisch sehr billige Methode mit Berechnung in $\mathcal{O}(n)$ dar. Je nach Implementierung benötigt diese Methode $\mathcal{O}(n)$ Speicherplatz zur Archivierung der ermittelten Binärzahlen bis zur Anwendung des Backpropagation-Algorithmus [13]. Darüber hinaus verfügt diese Technik über so gut wie keine Limitierungen bezüglich der Art von Modellen oder verwendeten Trainingsprozeduren. So kann dieser Ansatz für fast jedes Modell angewendet werden, welches über verteilte Repräsentationen verfügt und mit dem stochastischen Gradientenabstieg trainiert werden kann.

3.4.2 Batch-Normalisierung

Die Batch-Normalisierung ist eine Methode zur adaptiven Reparametrisierung basierend auf den Herausforderungen des Trainings sehr tiefer neuronaler Netze. Solche Modelle können als Kompositionen verschiedener Funktionen und Schichten angesehen werden. Dabei werden Gradienten zur Aktualisierung jedes Parameters unter der Annahme fehlender Veränderungen der Neuronen anderer Schichten verwendet. Aufgrund der simultanen Aktualisierungen in der Praxis können dadurch jedoch unerwartete Ergebnisse entstehen, da viele zusammengefügte Funktionen unter der Annahme konstant bleibender Funktionen simultan verändert werden [13].

Mittels der Batch-Normalisierung soll eine Reparametrisierung durchgeführt werden, welche das Problem der Koordinierung von Aktualisierungen zwischen vielen Schichten reduzieren soll. Diese Technik kann auf der Eingabeschicht und jeder verborgenen Schicht eines Netzwerks angewendet werden. Während des Trainings werden für zu normalisierende Schichten auf Grundlage des aktuellen Minibatches eine Matrix H erstellt, welche die jeweiligen resultierenden Aktivierungen enthält [13]. Dabei wird diese Matrix in Form einer Design-Matrix konstruiert, welche die Akti-

vierungen eines jeden Beispiels des Minibatches in einer Matrix-Zeile repräsentiert.

Eine Normalisierung dieser Matrix basiert auf der Formel $H' = \frac{H-\mu}{\sigma}$. Die Variable μ stellt in diesem Zusammenhang einen Vektor der Mittelwerte von Aktivierungen der einzelnen Neuronen und σ die entsprechenden Standardabweichungen dieser Aktivierungen dar. Der Mittelwert wird dabei auf Grundlage der Formel $\mu = \frac{1}{m} * \sum_i H_{i,j}$ und die Standardabweichung unter Anwendung der Formel $\sigma = \sqrt{\delta + \frac{1}{m} * \sum_i (H-\mu)_i^2}$ berechnet [13]. m repräsentiert in diesem Zusammenhang die Anzahl der Beispiele des Minibatches. Die Variable δ stellt wiederum einen kleinen positiven Wert wie beispielsweise 10^{-8} dar, welche verwendet wird, um undefinierte Gradienten von $\sqrt{z}$ bei $z = 0$ zu vermeiden.

Diese Vektoren werden auf jede Zeile der Matrix H übertragen, um jedes Element durch die Formel $H'_{i,j} = \frac{H_{i,j}-\mu_j}{\sigma_{i,j}}$ separat zu normalisieren [13]. Auf Grundlage der Matrix H' findet das restliche Training des Netzwerks analog zur gewohnten Verarbeitung der Matrix H statt. Entscheidend ist dabei die Berechnung des Mittelwerts sowie der Standardabweichung im Rahmen der Backpropagation zur Anwendung der Normalisierung.

Diese Normalisierungsoperationen werden grundlegend zur Vermeidung von Effekten der Erhöhung von Standardabweichungen und Mittelwerten benötigt [13]. Dementsprechend werden durch die Batch-Normalisierung Mittelwerte und Varianzen jedes Neurons standardisiert, um das Lernen bei möglichen Anpassungen der Beziehungen zwischen den Neuronen sowie bei der Anpassung nichtlinearer Statistiken einzelner Neuronen zu stabilisieren.

4 Wissenschaftliche Methodik

Im Rahmen dieses Kapitels wird das Konzept der Experimente zur Untersuchung des Zusammenhangs zwischen Software-Produktmetriken und dem Auftreten von Schwachstellen erläutert. Dazu werden in Kapitel 4.1 drei verschiedene Forschungsfragen aufgestellt, welche mittels empirischer Studien ausgewertet werden sollen. Der grundlegende Aufbau dieser Experimente wird wiederum in Kapitel 4.2 beschrieben und in den darauf folgenden Kapiteln konkretisiert.

4.1 Forschungsfragen

Aufgrund komprimierter Lebenszyklen der Softwareentwicklung und der steigenden Komplexität und Konnektivität der entstehenden Software-Produkte ist die Sicherheit und letztendlich auch die Qualität dieser Systeme ein kritischer Faktor. Schwachstellen gelten in diesem Kontext als die Ursache aller Sicherheitsrisiken, weshalb Techniken und Methoden zur frühzeitigen Erkennung dieser Gefahren benötigt werden, um die Sicherheit eines Software-Produkts zu erhöhen. Da Software-Produktmetriken die Eigenschaften des Quellcodes solcher Produkte quantifizieren, sollten diese Charakteristiken bezüglich der Existenz von Schwachstellen innerhalb von Abschnitten des Quellcodes analysiert werden. Mithilfe einer solchen Analyse sollen Muster und Korrelationen identifiziert werden, welche auf potenzielle Sicherheitsprobleme hinweisen können, um präventive Maßnahmen zur Minimierung und Vermeidung von Schwachstellen zu ergreifen.

Basierend auf einer umfangreichen Datenanalyse soll durch die Entwicklung statistischer Modelle auf Grundlage der Verwendung von Software-Produktmetriken Vorhersagen getroffen werden, um Schwachstellen identifizieren und bewerten zu können. Solche Vorhersagen können Entwicklern und Sicherheitsexperten als unterstützendes Werkzeug dienen, um effektiv und effizient die Sicherheit von Software-Produkten zu erhöhen. Auf Grundlage resultierender Ergebnisse sollen die untersuchten Software-Produktmetriken außerdem hinsichtlich ihrer Eignung zur Bewertung von Software-Qualität überprüft werden. Dementsprechend werden diese Metriken den Hauptqualitätskriterien sowie den jeweiligen Aspekten der Software-Qualität nach ISO/IEC 25010 gegenübergestellt. In diesem Kontext werden die konkreten Forschungsfragen wie folgt definiert:

- **RQ1:** Können Software-Produktmetriken die Existenz von Schwachstellen erkennen?

- **RQ2:** Können Software-Produktmetriken die Schwere von Schwachstellen vorhersagen?
- **RQ3:** Welche Software-Produktmetriken eignen sich zur Bewertung der Kriterien und Aspekte der Software-Qualität?

4.2 Architektur

Zur Ermittlung von Vorhersagen hinsichtlich der Erkennung des Auftretens von Schwachstellen sowie zur Bewertung ihrer Schwere sollen künstliche neuronale Netze verwendet werden. Unter Anwendung der Methode des überwachten Lernens wird ein Datensatz benötigt, in dem jeder Eintrag mit entsprechenden Zielgrößen gekennzeichnet ist. Zur Erstellung eines solchen Datensatz wird wiederum der Quellcode bereits bekannter und verifizierter Schwachstellen benötigt, um durch den Einsatz von Metriken-Extraktionstools die entsprechenden Software-Produktmetriken zu ermitteln. Darüber hinaus werden außerdem Beispiele ohne Schwachstellen benötigt, um den Datensatz entsprechend zu balancieren. Aufgrund dieser Anforderungen besteht das gesamte Experiment aus verschiedenen Schritten, deren Ablauf in Abbildung 9 beispielhaft visualisiert wurde.

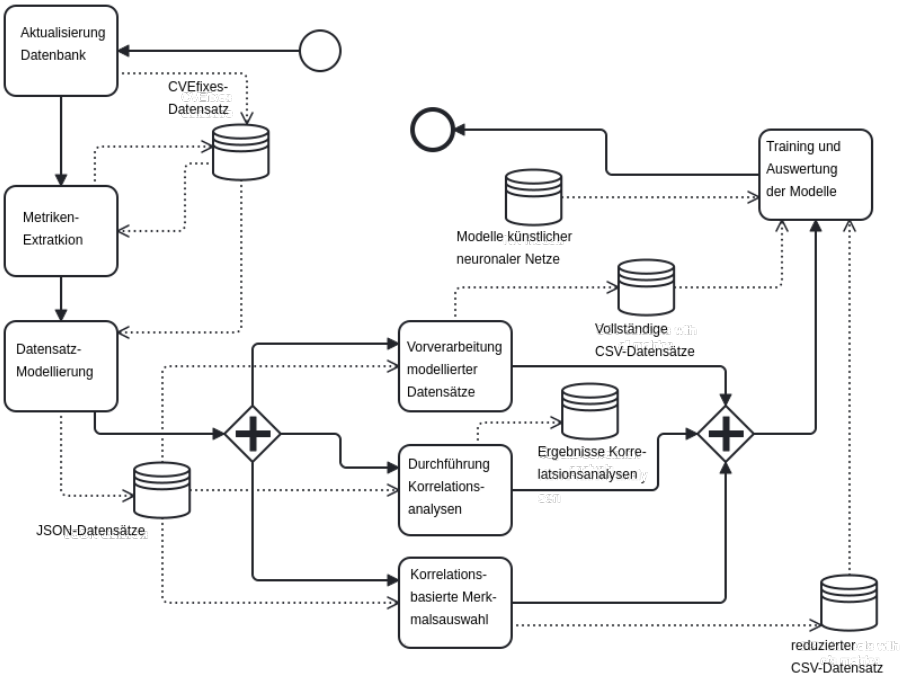


Abbildung 9: Schematischer Aufbau der Experimente

Der erste Prozess der Experimente wird durch die Bereitstellung einer geeigneten Datenbasis zur Erstellung des benötigten Datensatzes initiiert. Dazu wurde der *CVEfixes*-Datensatz verwendet, welcher in Kapitel 4.3 genauer beschrieben

wird. Innerhalb dieses Datensatzes werden für alle darin enthaltenen CVE-Einträge Quellcode bereitgestellt, welcher die entsprechende Schwachstelle abbildet. Darüber hinaus ist für jeden dieser Einträge der Quellcode des gleichen Code-Abschnitts nach Behebung der jeweiligen Schwachstelle enthalten. Der Prozes zur Erstellung dieser Datenbank wurde automatisiert, sodass die Datenbank zur Durchführung der Experimente aktualisiert werden kann.

Nach der Aktualisierung und Bereitstellung des CVEfixes-Datensatzes sollen verschiedene Software-Produktmetriken für den in der Datenbank enthaltenen Quellcode ermittelt werden. Dazu sollen die in Kapitel 2.3.2 vorgestellten Metriken-Extraktionstools verwendet werden, um den referenzierten Quellcode jedes CVE-Eintrags zu analysieren. Die extrahierten Metriken sollen anschließend in das Modell des CVEfixes-Datensatzes für die weiteren Bearbeitungsschritte integriert werden. Eine konkrete Beschreibung des Prozesses der Metriken-Extraktion wird in Kapitel 4.4 detailliert beschrieben.

Auf Grundlage der extrahierten Metriken sollen anschließend verschiedene Roh-Datensätze definiert werden. Diese Datensätze sollen grundsätzlich alle unverarbeiteten Werte der Software-Produktmetriken enthalten, welche durch die Metriken-Extraktionstools ermittelt werden konnten. Durch die Anwendung der Methode des überwachten Lernens müssen für jeden dieser Einträge entsprechende Zielgrößen bestimmt werden, welche letztendlich aus dem CVEfixes-Datensatz extrahiert werden können. Basierend auf diesen Roh-Datensätzen können wiederum Korrelationsanalysen durchgeführt werden, um Korrelationen zwischen einzelnen Merkmalen und den jeweiligen Zielgrößen zu untersuchen. Darüber hinaus müssen diese Datensätze für den Einsatz in künstlichen neuronalen Netzen durch verschiedene Techniken vorverarbeitet werden. Zur Reduzierung der Dimensionalität der Merkmalsmenge soll außerdem eine korrelationsbasierte Merkmalsauswahl durchgeführt werden, welche im Rahmen der Experimente als zusätzliche, separate Datensätze verwendet werden. Diese Prozesse werden in Kapitel 4.5 konkret erläutert.

Im Rahmen der Vorverarbeitung sollen alle erstellten Datensätze in Trainings-, Test- und Validierungsdaten aufgeteilt werden. Mithilfe dieser Aufteilung sollen die Daten letztendlich für das Training verschiedener künstlicher neuronaler Netze sowie zur Auswertung resultierender Ergebnisse verwendet werden. Grundlegend sollen für die Erkennung von Schwachstellen und der Bewertung ihrer Schwere unterschiedliche künstliche neuronale Netze aufgestellt werden, welche sich allerdings nur in der Auswahl der Aktivierungsfunktion innerhalb der Ausgabeschicht unterscheiden. Darüber hinaus sollen für beide Szenarien verschiedene Architekturen künstlicher neuronaler Netze definiert werden, welche sich topologisch anhand ihrer Anzahl verwende-

ter Neuronen und Schichten unterscheiden. Dazu wird der Aufbau dieser Modelle in Kapitel 4.7 charakterisiert. Die Konfiguration des Trainingsprozesses sowie die Möglichkeiten zur Auswertung resultierender Ergebnisse wird wiederum in Kapitel 4.8 beschrieben.

4.3 CVEfixes-Datensatz

Der CVEfixes-Datensatz ist eine umfangreiche Sammlung von Quellcode, welcher Schwachstellen sowie deren Anpassungen zur Behebung dieser Schwachstellen enthält. Dieser Datensatz wurde vollständig automatisch auf Grundlage von Einträgen der *Common Vulnerabilities and Exposures* (*CVE*) generiert und in einer relationalen Datenbank gespeichert. Der konkrete Prozess zur Erzeugung des CVEfixes-Datensatzes wird in Kapitel 4.3.1 vorgestellt. Die Struktur der relationalen Datenbank wird in Kapitel 4.3.2 beschrieben.

4.3.1 Prozess zur Erzeugung des CVEfixes-Datensatzes

Zur Erzeugung des Datensatzes werden alle verfügbaren CVE-Einträge im JSON-Format vom NVD-Server extrahiert. Zusätzlich werden von diesem Server alle Einträge der Common Weakness Enumeration zur Kategorisierung aller CVE-Einträge ermittelt und im XML-Format bereitgestellt. Alle extrahierten CVE-Einträge werden chronologisch verarbeitet, während alle Einträge übersprungen werden, welche keine Referenzierung bezüglich der Behebung der entsprechenden Schwachstelle aufweisen. Während dieses Verarbeitungsschrittes werden für jeden relevanten Eintrag Informationen bezüglich referenzierter Repositories und Commit-Hashes identifiziert [2].

Mithilfe dieser Informationen werden benötigte Repositories durch den Einsatz der GitHub-API lokal geklont, um Informationen auf Quellcode-Ebene zu extrahieren. Zur Ermittlung entsprechender Änderungen wird wiederum der jeweilige Commit-Hash verwendet, um das geklonte Repository in dem Zustand vor und nach dem Commit zu analysieren. Dabei werden die durch den Commit veränderten Dateien und Methoden extrahiert und durch den Einsatz zusätzlicher Werkzeuge einige Quellcode-Metriken und weitere Metaden wie beispielsweise verwendete Programmiersprachen ermittelt. Zur Speicherung aller Daten in der relationalen Datenbank werden redundante Informationen bezüglich der CVE- und CWE-Einträge aggregiert und gefiltert [2].

4.3.2 Struktur des CVEfixes-Datensatzes

Der CVEfixes-Datensatz enthält hauptsächlich Informationen bezüglich der CVE- und CWE-Einträge sowie der untersuchten Repositories inklusive ihrer Commits zur Behebung der Schwachstellen und den darin enthaltenen Dateien und Methoden. Eine Überblick über die Struktur der Datenbank wurde in Abbildung 10 als Entity-Relationship-Diagramm visualisiert.

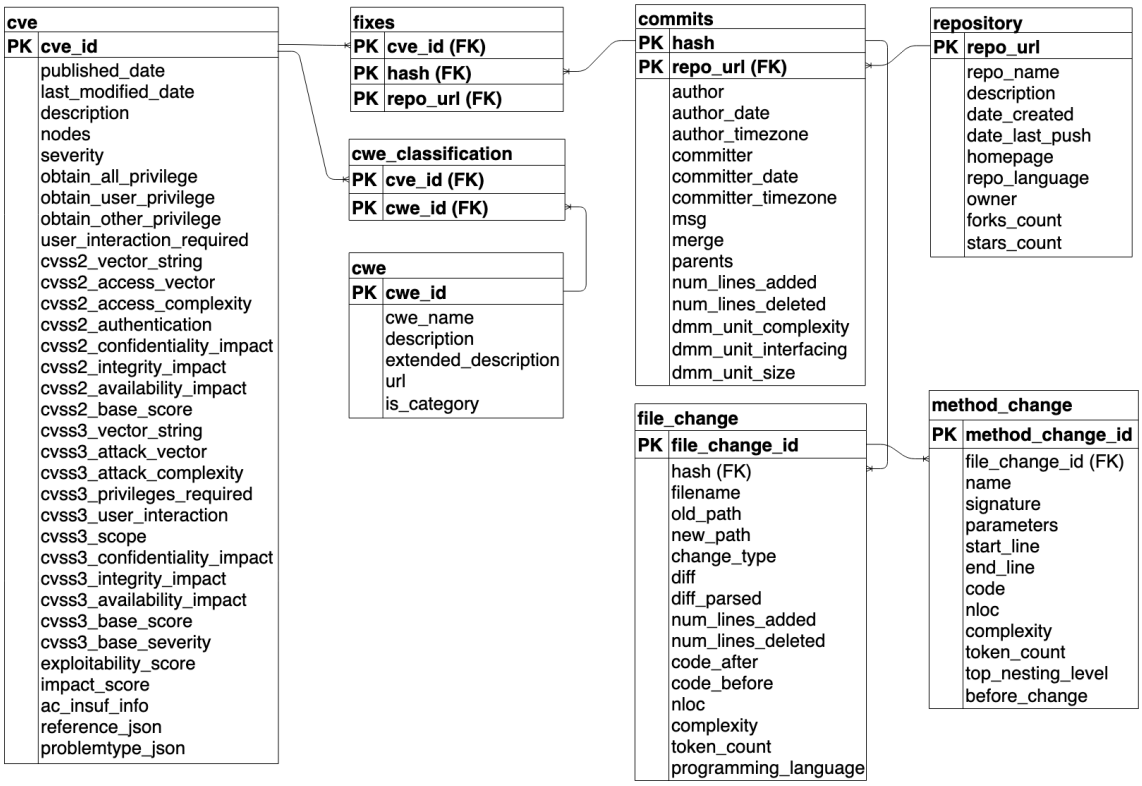


Abbildung 10: ER-Diagramm der relationalen Datenbank des CVEfixes-Datensatzes [2]

Die Tabelle *cve* enthält verschiedene Metadaten der verarbeiteten CVE-Einträge, welche beispilesweise eine eindeutige CVE-ID, das Veröffentlichungsdatum sowie das Datum der letzten Änderung sind. Weiterhin enthalten diese Metriken verschiedene Bewertungen durch das *Common Vulnerability Scoring System* (*CVSS*). Für dieses Wertungssystem existieren zwei verschiedene Versionen namens $CVSS_2$ und $CVSS_3$, welche beide aktiv verwendet werden können. Die Version $CVSS_3$ enthält im Vergleich zur $CVSS_2$-Version einige Änderungen, sodass Schwachstellen präziser bewertet werden können und enthält zusätzliche Informationen zur Unterscheidung zwischen verschiedenen Arten von Schwachstellen [2].

Alle Einträge der cve-Tabelle werden durch die Tabelle *cwe_classification* zu ihren jeweiligen CWE-Kategorien abgebildet, welche in der Tabelle *cwe* enthalten

sind. Diese Tabelle enthält neben einer eindeutigen ID detaillierte Beschreibungen der einzelnen CWE-Kategorien. Für die Erzeugung des Datensatzes wurde nur eine Teilmenge aller CWE-Einträge verwendet. Daher wurde eine zusätzlich CWE-Kategorie namens *NVD-CWE-other* in der cwe-Tabelle integriert. Zur Erhaltung der Konsistenz wurde außerdem eine weitere CWE-Kategorie namens *NVD-CWE-noinfo* erstellt, welche alle CVE-Einträge referenziert, deren CWE-Kategorie nicht bestimmt wurde.

Alle Meta-Informationen der untersuchten Repositories, welche über die GitHub-API extrahiert wurden, sind in der Tabelle *repository* abgebildet. Diese Informationen umfassen den Namen, eine Beschreibung und das Datum des letzten Pushes, die URL, die verwendete Programmiersprache sowie die Anzahl an Stars und Forks jedes Repositories. Mittels der Repository-URL wird jeder betrachtete Commit referenziert, welche in der Tabelle *commits* erfasst worden. Diese Tabelle beinhaltet commit-spezifische Metadaten wie beispielsweise den Author, die Zeit und das Datum sowie die Nachricht des Commits und die Anzahl der durch den Commit hinzugefügten und gelöschten Code-Zeilen. Über die Repository-URL und der Tabelle *fixes* werden alle Commits ihren jeweiligen CVE-Einträgen zugewiesen. Dabei kann ein Commit mehrere CVE-Einträge referenzieren.

Der Quellcode der durch Commits veränderten Dateien werden in der Tabelle *file_change* integriert und mittels Commit-Hash zu ihren entsprechenden Commits der commits-Tabelle referenziert. Jeder Eintrag der file_change-Tabelle enthält den Quellcode vor und nach Ausführung des Commits sowie den Unterschied zwischen beiden Versionen. Darüber hinaus werden Metadaten wie die Art der Veränderung, Dateinamen sowie neue und alte Pfadbezeichnungen gespeichert. Die verwendete Programmiersprache sowie die Anzahl der Code-Zeilen und die zyklomatische Komplexität der Datei werden ebenfalls abgebildet.

Der Quellcode der durch Commits veränderten Methoden und Funkionen werden wiederum in der Tabelle *method_change* erfasst, welche über das Attribut *file_change_id* zu entsprechenden Einträgen der file_change-Tabelle referenziert werden. Dabei wird jede Version einer Methode als separater Eintrag gespeichet, welcher zusätzliche Metadaten wie beispielsweise den Methodennamen, ihre Signatur, eine Parameterliste sowie die Code-Zeilen für den Beginn und das Ende innerhalb der ursprünglichen Datei integriert. Analog zur Datei-Extraktion werden ebenfalls verschiedene Metriken erfasst und archiviert.

Der Datensatz wurde initial am 09.06.2021 erzeugt und verfügt über 5.495 Commits aus 1.754 verschiedenen Open-Source-Projekten. Diese Commits referenzieren 5.365 CVE-Einträge, welche 180 unterschiedlichen CWE-Kategorien zugewiesen wer-

den können. Darüber hinaus enthält dieser Datensatz Versionen des Quellcodes von 18.249 Dateien und 50.322 Funktionen vor und nach der Behebung entsprechender Schwachstellen. Eine Auswertung der Aktualisierung des Datensatzes wird in Kapitel 6.1 vollzogen.

4.4 Integration von Werkzeugen der Metriken-Extraktion

Die Versionen des Quellcodes vor und nach der Behebung einzelner Schwachstellen wird durch den CVEfixes-Datensatz auf Datei- und Methoden-Ebene bereitgestellt. Software-Produktmetriken können dementsprechend auf verschiedenen Granularitätsebenen extrahiert werden, welche über unterschiedliche Vor- und Nachteile verfügen. Die Analyse des Quellcodes auf Methoden-Ebene ermöglicht beispielsweise viel präzisere Aussagen über die Lokalität potenziell erkannter Schwachstellen. Allerdings können nicht alle Software-Produktmetriken Charakteristiken auf Methoden-Ebene erfassen, wodurch die Menge verwendbarer Software-Produktmetriken stark beeinträchtigt wird. Kohäsionsmetriken erfassen beispielsweise den Verwandtschaftsgrad von Interaktionen zwischen verschiedenen Entitäten einzelner Komponenten. Methoden stellen in diesem Kontext einzelne Entitäten dar, wodurch der Verwandtschaftsgrad innerhalb einer solchen Komponente nicht messbar ist.

Für die Extraktion von Metriken auf Datei-Ebene können hingegen deutlich mehr Metriken bereitgestellt werden. In vielen Fällen repräsentieren Dateien einzelne Klassen der objektorientierten Programmierung, sodass alle verfügbaren Software-Produktmetriken der Klassen-Ebene verwendet werden können. Durch die Anwendung verschiedener Aggregationen können außerdem die Charakteristiken der in einer Klasse enthaltenen Methoden ebenfalls repräsentiert werden. So können beispielsweise Quantifizierungs- und Komplexitätsmetriken der Methoden-Ebene entsprechend aggregiert werden, um zusätzliche Informationen bereitzustellen. Darüber hinaus können Kohäsionsmetriken extrahiert werden, welche den Verwandtschaftsgrad zwischen den Interatkionen von Methoden und Attributen erfassen.

Dementsprechend kann im Vergleich zur Metriken-Extraktion auf Methoden-Ebene eine deutlich höhere Anzahl an Software-Produktmetriken auf Datei-Ebene bereitgestellt werden. Ein Nachteil dieses Ansatzes ist jedoch die sinkende Genauigkeit bezüglich der Lokalität potenzieller Schwachstellen, da diese nur auf Datei-Ebene erkannt werden können. Durch die Anwendung von Aggregationen der Charakteristiken auf Methoden-Ebene besteht außerdem das Risiko, subtile Schwachstellen nicht identifizieren zu können. Bezieht sich der Einfluss einer Schwachstelle beispielsweise nur auf einen sehr geringen Abschnitt des Quellcodes, können durch Aggregationen entscheidende Informationen zur Erkennung und Bewertung solcher

Schwachstellen missachtet werden.

Neben der Extraktion von Software-Produktmetriken durch die Betrachtung der einzelnen Entitäten auf Methoden- und Datei-Ebene können diese Komponenten ebenfalls auf Grundlage der entsprechenden Versionen auf Commit- und Projekt-Ebene erfasst werden. Zur Betrachtung der Versionen aller zusammenhängenden Dateien und Methoden auf Commit-Ebene kann der entsprechende Quellcode aus dem CVEfixes-Datensatz entnommen werden. Für die Untersuchung der entsprechenden Dateien und Methoden auf Projekt-Ebene wird jedoch explizit die jeweilige Projekt-Version vor und nach dem entsprechenden Commit benötigt, welcher die vorhandene Schwachstelle behebt. Daher benötigt der Ansatz zur Auswertung der entsprechenden Entitäten auf Projekt-Ebene zusätzlichen Aufwand zur Bereitstellung der jeweiligen Projekt-Versionen.

Die Analyse der Entitäten im Rahmen der vollständigen Projekt-Version verfügt allerdings über die Möglichkeit der Extraktion weiterer Software-Produktmetriken. Durch die Analyse vollständiger Projekt-Versionen können zum Beispiel Vererbungs-, Kopplungs- und Netzwerkmetriken ermittelt werden, welche durch die Untersuchung einzelner Entitäten nicht bestimmt werden können. Eine Berechnung dieser Metriken eignet sich hingegen nicht für die Betrachtung der jeweiligen Entitäten auf Commit-Ebene, da essenzielle Zusammenhänge, die innerhalb eines Commits nicht behandelt wurden, nicht berücksichtigt werden können. Daher kann die Extraktion solcher Metriken auf Commit-Ebene zu Ungenauigkeiten führen, welche die Vorhersageergebnisse künstlicher neuronaler Netze beeinflussen können.

Mithilfe des Analizo-Tools können Software-Produktmetriken auf Projekt- und Modul-Ebene extrahiert werden. Dateien können in diesem Kontext ebenfalls als Module angesehen werden, wodurch die Metriken-Extraktion mittels Analizo auf Datei-Ebene ermöglicht werden kann. Zur Extraktion von Metriken auf Methoden-Ebene kann dieses Werkzeug der Metriken-Extraktion hingegen nicht verwendet werden. Das Understand-Tool stellt wiederum eine Vielzahl von Software-Produktmetriken auf Methoden- und Datei-Ebene bereit. Mittels Understand kann jedoch im Vergleich zur Methoden-Ebene eine deutlich höhere Anzahl an Metriken auf Datei-Ebene extrahiert werden. Basierend auf der Verwendung der Werkzeuge Analizo und Understand eignet sich dementsprechend die Berechnung von Software-Produktmetriken auf Datei-Ebene.

Zur Berechnung der Software-Produktmetriken auf Datei-Ebene können die entsprechenden Projekt-Versionen durch die Verwendung verschiedener Werkzeuge und der Auswertung der Informationen des CVEfixes-Datensatzes betrachtet werden. Aufgrund der Vielzahl zu untersuchender Dateien werden jedoch entsprechend viele

Projekt-Versionen unterschiedlicher Software-Produkte benötigt. Der rechnerische Aufwand zur Replikation dieser Projekt hängt dabei stark von der jeweiligen Projektgröße ab. Darüber hinaus müssen beide Metriken-Extraktionstools auf Grundlage der vollständigen Projekt-Versionen ausgeführt werden, wodurch sich der rechnerische Aufwand zusätzlich erhöht.

Die Identifikation der Metriken-Werte der entsprechenden Entitäten stellt in diesem Zusammenhang außerdem eine kritische Herausforderung dar. Auf Grundlage der vollzogenen Metriken-Extraktion werden die Metriken-Werte mittels entsprechender Bezeichnungen durch die Werkzeuge der Metriken-Extraktion referenziert. Basierend auf der Bezeichnung der jeweiligen Entitäten innerhalb des CVEfixes-Datensatzes können durch die verwendeten Werkzeuge abweichende Bezeichnungen innerhalb der Metriken-Referenzierung entstehen. Darüber hinaus können innerhalb einer Projekt-Version die Bezeichnungen von Dateien mehrfach vergeben werden, sodass die Identifikation korrekter Datei-Entitäten keine triviale Aufgabe darstellt. Dementsprechend stellt die Betrachtung einzelner Entitäten aufgrund des hohen rechnerischen Aufwands und der beschriebenen Problematik einen sicheren und effizienteren Ansatz dar. Eine Betrachtung mehrerer Dateien auf Commit-Ebene soll dabei basierend auf dem Potenzial ähnlicher Herausforderungen ebenfalls nicht durchgeführt werden.

Auf Grundlage der Extraktion von Software-Produktmetriken durch die Werkzeuge Analizo und Understand auf Datei-Ebene, sollen alle ermittelten Metriken in das Schema des CVEfixes-Datensatz für alle weiteren Verarbeitungsschritte integriert werden. Für die extrahierten Metriken jedes Extraktionstools soll das relationale Datenbankschema dieses Datensatzes um eine separate Datenbanktabelle werden, in der für jede untersuchte Datei-Version einzelne Einträge generiert werden. Die Einträge dieser beiden Datenbanktabellen sollen wiederum zu den entsprechenden Einträgen der *file_change*-Tabelle durch Verwendung des Attributs *file_change_id* referenziert werden. Darüber hinaus müssen diese Einträge um ein weiteres bool'sches Attribut erweitert werden, welches bestimmt, ob jeweiligen Metriken für eine Datei-Version vor oder nach Behebung einer Schwachstelle extrahiert wurden. Eine konkrete Implementierung der Metriken-Extraktion auf Grundlage des CVEfixes-Datensatzes unter Verwendung der Werkzeuge Understand und Analizo wird in Kapitel 5.1 beschrieben.

4.5 Modellierung des Datensatzes

Durch die Extraktion und Bereitstellung der Software-Produktmetriken auf Datei-Ebene können Datensätze definiert werden, welche für den Einsatz in künstlichen

neuronalen Netzen verwendet werden können. Dazu müssen die Informationen des erweiterten CVEfixes-Datensatzes erneut extrahiert werden, um Roh-Datensätze zu erstellen, welche in Kapitel 4.5.1 charakterisiert werden. Nach der Erzeugung dieser Datensätze müssen jedoch weiterer Vorverarbeitungsschritte durchgeführt werden, um tatsächlich in künstlichen neuronalen Netzen eingesetzt zu werden. Dieser Prozess wird in Kapitel 4.5.2 thematisiert.

4.5.1 Definition des Roh-Datensatzes

Auf Grundlage der Extraktion von Software-Produktmetriken durch die Werkzeuge Analizo und Understand können drei verschiedene Datensätze definiert werden. Mittels der bereitgestellten Metriken beider Extraktionstools können separate Roh-Datensätze entworfen werden, welche im folgenden als *Understand-* und *Analizo-Datensätze* bezeichnet werden. Basierend auf der Kombination dieser beiden Datensätze kann darüber hinaus ein weiterer Datensatz erzeugt werden, welcher im folgenden als *kombinierter Datensatz* bezeichnet wird. Für jeden Datensatz müssen die entsprechenden Werte der Software-Produktmetriken aus dem erweiterten Modell des CVEfixes-Datensatzes extrahiert werden. Neben diesen Werten soll außerdem das Attribut *programming_language* der file_change-Tabelle als zusätzliches Merkmal verwendet werden. Darüber hinaus soll im Rahmen der Vorverarbeitung der Datensätze das Attribut *repo_url* der *commits*-Tabelle integriert werden.

Zur Erstellung des kombinierten Datensatzes sollen die extrahierten Einträge von Metriken-Werten beider Werkzeuge auf Grundlage identischer Referenzierungen zur file_change-Tabelle zusammengeführt werden. Dabei muss jedoch die entsprechende Datei-Version, welche in den Einträgen der Metriken-Werte als bool'scher Wert hinterlegt ist, beachtet werden, um die Konsistenz dieser Daten sicherzustellen. Dieser bool'sche Wert kann außerdem als Zielgröße zur Erkennung des Auftretens von Schwachstellen für künstliche neuronale Netze verwendet werden. In Abhängigkeit der konkreten Definition dieses Wertes bestimmt dieses Attribut die Existenz einer Schwachstelle innerhalb der jeweiligen analysierten Datei-Version.

Als Zielgröße zur Vorhersage der Schwere einer Schwachstelle soll außerdem die CVSS-Wertung der entsprechenden Schwachstelle verwendet werden. Diese Wertung bezieht sich auf den jeweiligen CVE-Eintrag einer Schwachstelle und verfügt innerhalb des CVEfixes-Datensatzes über eine indirekte Verknüpfung zu den entsprechenden Dateien. Da eine Schwachstelle im Rahmen des CVEfixes-Datensatzes durch die jeweiligen Dateien einer Commit-Version repräsentiert werden, verfügt jede Datei des entsprechenden Commits über die gleiche Wertung. Datei-Versionen nach Behebung einer Schwachstellen sollen jedoch eine Wertung von 0 erhalten, da

die Schwachstelle in dem Zustand dieser analysierten Dateien nicht mehr vorhanden ist. Im Rahmen dieser Experimente wird die CVSS3-Wertung verwendet, da diese Version die Schwere von Schwachstellen präziser bewerten soll als die CVSS2-Wertung.

Die drei resultierenden Roh-Datensätze sollen letztendlich als separate Dateien in einem kompakten Datenformat lesbarer Textform zur Verarbeitung in allen weiteren Prozessen gespeichert werden. Die Implementierung zur Erzeugung dieser Roh-Datensätze wird wiederum in Kapitel 5.2 erläutert.

4.5.2 Vorverarbeitung des Roh-Datensatzes

Zur Verwendung der erzeugten Roh-Datensätze innerhalb künstlicher neuronaler Netze müssen einige Vorverarbeitungsschritte durchgeführt werden. Aufgrund der Untersuchung von Dateien verschiedener Programmiersprachen ist beispielsweise die Extraktion von Metriken-Werten durch die Werkzeuge Analizo und Understand inkonsistent. Jede Programmiersprache verfügt über spezifische Techniken und Prinzipien, sodass grundlegende Unterschiede in den Charakteristiken der jeweiligen Programmiersprache bestehen. Die Programmiersprache C unterliegt beispielsweise einem prozeduralen Programmierparadigma, wodurch Konzepte der objektorientierten Programmierung in diesem Konzept nicht vorhanden sind. Dadurch können beispielsweise keine Charakteristiken von Vererbungseigenschaften für Dateien dieser Programmiersprache abgebildet werden.

Darüber hinaus existieren Software-Produktmetriken, welche für spezifische Programmiersprachen entworfen wurden. Die Metrik *LOC_PHP* quantifiziert beispielsweise die Anzahl aller in PHP formulierten Zeilen des Quellcodes, welche in vielen untersuchten Dateien gar nicht auftreten können. Außerdem kann das Metriken-Extraktionstool Analizo nur die Metriken für Dateien ermitteln, welche in den Programmiersprachen C, C++ und Java formuliert wurden. Understand verfügt hingegen über eine viel größere Unterstützung von Programmiersprachen, sodass für dieses Werkzeug eine deutlich größere Datenbasis erzeugt wird. Für die Zusammensetzung der Metriken-Werte beider Extraktionstools innerhalb des kombinierten Datensatzes resultiert zwangsweise eine inkonsistente Datenbasis aufgrund fehlender Metriken-Werte bei vielen Beispieldaten. Dementsprechend soll im Rahmen der Vorverarbeitung jeder fehlende Metriken-Wert durch den Wert 0 ersetzt werden. Software-Produktmetriken für die grundsätzlich gar keine Metriken-Werte berechnet werden konnten oder in allen Beispielen einen konstanten Wert abbilden, werden hingegen aus allen Datensätzen entfernt.

Weiterhin müssen die Eigenschaften von Werten der Merkmalsmengen unter-

sucht werden. Unverarbeitete nominale Werte können beispielsweise nicht durch künstliche neuronale Netzwerke mit einem gewünschten Ergebnis verarbeitet werden, wodurch solche Werte im Rahmen der Vorverarbeitung transformiert werden müssen. Die Abbildung nominaler Werte durch ordinale Zahlenwerte eignet sich aufgrund der Interpretation durch festgelegte Reihenfolgen in diesem Kontext nicht. Für jeden nominalen Wert eines Merkmals sollte deswegen ein zusätzliches bool'sches Merkmal erstellt werden.

Innerhalb der Roh-Datensätze ist das Attribut der Programmiersprache das einzige nominale Merkmal. Daher soll die Merkmalsmenge jedes Datensatzes erweitert werden, sodass für jede analysierte Programmiersprachen ein zusätzlicher bool'scher Wert als Merkmal integriert wird. Dabei repräsentiert der Wert 1, das dem jeweiligen Beispiel die entsprechende Programmiersprache zugrunde liegt. Somit darf für jedes Beispiel aller Datensätze stets nur eines dieser transformierten Merkmale über den Wert 1 verfügen.

Aufgrund der Verwendung bool'scher Werte innerhalb der Merkmalsmenge sollen außerdem alle Merkmale kontinuierlicher Werte skaliert werden. Da jedes Merkmal über unterschiedliche Wertebereiche verfügt, eignet sich außerdem die Anpassung der Wertebereiche, um den Einfluss unterschiedlicher Software-Produktmetriken auszugleichen. Da alle extrahierten Software-Produktmetriken kontinuierliche Werte repräsentieren, soll durch die Anwendung der Skalierung die Wertebereiche aller Software-Produktmetriken auf das Intervall $[0, 1]$ beschränkt werden.

Nach der Durchführung dieser Vorverarbeitungsschritte müssen diese Datensätze jeweils in Trainings-, Test- und Validierungsdaten aufgeteilt werden. Diese Aufteilung soll auf Grundlage des zuvor erwähnten Attributs *repo_url* stattfinden. Dadurch sollen sich letztendlich alle Beispiele des gleichen Software-Produkts in der gleichen Teilmenge des jeweiligen Datensatzes befinden. Mithilfe dieser Trennung findet das Training künstlicher neuronaler Netze auf anderen Software-Produkten statt als die Validierung und Auswertung mittels Validierungs- und Testdaten. Somit verfügen die Experimente über Eigenschaften der projekübergreifenden Fehlervorhersage.

Die resultierenden Datensätze aller Trainings-, Test- und Validierungsdaten sollen letztendlich als separate Dateien in einem einfach strukturierten Datenformat für den Einsatz in künstlichen neuronalen Netzen gespeichert werden. Die Implementierung Methoden zur Vorverabeitung der Roh-Datensätze wird wiederum in Kapitel 5.2 erläutert.

4.6 Methoden der Korrelationsanalyse

Der Begriff der Korrelation definiert ein grundlegendes Konzept der Statistik zur Beschreibung der Beziehungen zwischen veschiedenen Variablen. Diese Zusammenhänge werden durch Korrelationskoeffizienten beschrieben, welche durch die Anwendung des statistischen Verfahrens der Korrelationsanalyse berechnet werden.

Zur Berechnung der Koeffizienten werden für die Untersuchung zweier Variablen die Prinzipien des Kreuzprodukts und der Kovarianz verwendet. Mithilfe der Kovarianz werden die Abweichungen von Mittelwerten der einen Variable mit den Abweichungen der Mittelwerte der anderen Variablen untersucht. Diese Abweichungen werden jeweils miteinander multipliziert und aufsummiert, um ein Kreuzprodukt zu erhalten. Da die Größe des Kreuzprodukts abhängig von der Anzahl der verwendeten Variablen-Werte ist, wird das berechnete Kreuzprodukt anschließend durch die Anzahl aller Variablen-Werte dividiert. Dadurch werden die Ergebnisse der Kreuzprodukte relativiert, wodurch letztendlich die Kovarianz berechnet wird [18]. Eine zusammenfassende Formel zur Berechnung der Kovarianz für zwei Variablen namens x und y ist in Abbildung 11 enthalten.

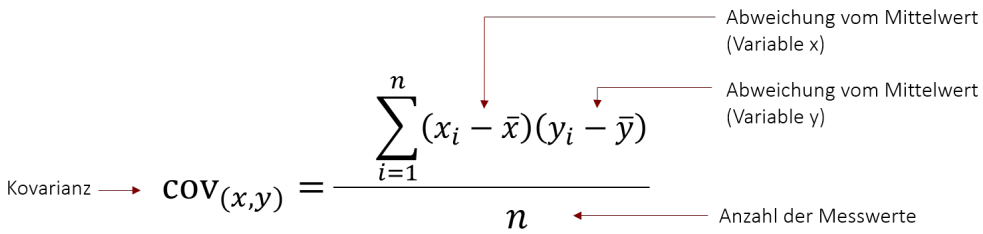


Abbildung 11: Formel zur Berechnung der Kovarianz für zwei Variablen [18]

Für die endgültige Berechnung der Korrelationskoeffizienten werden Korrelationsmaße zur Standardisierung der Kovarianz benötigt. Dadurch werden die untersuchten Variablen in den Wertebereich einer einheitlichen Skala umgerechnet. Durch die Anwendung der Standardisierung werden alle berechneten Koeffizienten in einem Wertebereich zwischen -1 und 1 abgebildet. Diese Werte repräsentieren die Stärke sowie die Richtung ermittelter Korrelationen.

Die Richtung einer Korrelation wird durch das Vorzeichen des Korrelationskoeffizienten repräsentiert. Bei einer positiven Korrelation zwischen zwei Variablen erhöht sich der Wert der einen Variable, sobald sich der Wert der anderen Variable ebenfalls erhöht. Sinkt der Wert einer Variable, sinkt der Wert der anderen Variable ebenfalls. Liegt eine negative Korrelation zwischen zwei Variablen vor, verhalten sich die Werte beider Variablen gegensätzlich. Sobald der Wert der einen Variable erhöht wird, sinkt der Wert der anderen Variable [18].

Der Betrag eines Korrelationskoeffizienten repräsentiert die Stärke einer Korrelation. Anhand dieses Wertes wird der Einfluss durch die Veränderung einer Variable hinsichtlich einer anderen Variable bestimmt. Je höher dieser Wert für die Korrelation zwischen zwei Variablen ist, desto höher ist der Einfluss einer Variable auf die andere Variable. Zur Interpretation dieser Werte existiert wiederum ein Ansatz von Jacob Cohen aus dem jahr 1988, welcher alle Beträge von Korrelationskoeffizienten mit einem Wert kleiner als 0.3 als schwache Korrelationen betrachtet. Eine Korrelationsstärke zwischen 0.3 und 0.5 wird als moderat kategorisiert und eine Korrelationsstärke größer als 0.5 als starke Korrelation interpretiert [19].

In Kapitel 4.6.1, Kapitel 4.6.2 und Kapitel 4.6.3 werden drei verschiedene Korrelationsmaße vorgestellt. Darüber hinaus wird zur Erzeugung eines reduzierten Datensatzes auf Grundlage der Korrelationsmaße die Technik der korrelationsbaiserten Merkmalsauswahl in Kapitel 4.6.4 beschrieben. Auf Grundlage dieser Korrelationsmaße und den Methoden der Vorverarbeitung sollen wiederum Korrelationsanalysen auf den Datensätzen durchegeführt werden, deren Prozess in Kapitel 4.6.5 beschrieben wird. Der Prozess zur Anwendung der korrelationsbasierten Merkmalsauswahl wird wiederum 4.6.6 erläutert.

4.6.1 Pearson-Korrelation

Die Pearson-Korrelation wurde nach dem britischen Statistiker Karl Pearson benannt und wird auch als Produkt-Moment-Korrelation bezeichnet. Zur Standardisierung der Kovarianz werden für dieses Korrelationsmaß die Standardabweichungen s_x und s_y der beiden Variablen x und y benötigt. Durch die Multiplikation dieser beiden Standardabweichungen entsteht der maximal mögliche Wert der Kovarianz. Dementsprechend muss die Kovarianz durch das Produkt der Standardabweichungen dividiert werden, um Korrelationskoeffizienten in einem Wertebereich zwischen -1 und 1 zu erhalten [18]. Eine vollständige Formel zur Berechnung der Pearson-Korrelation wird in Abbildung 12 dargestellt.

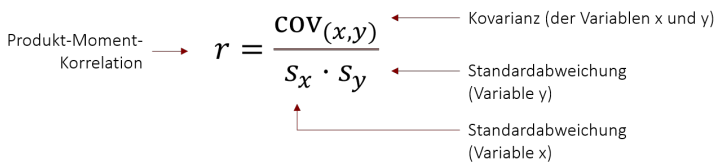


Abbildung 12: Formel zur Berechnung der Pearson-Korrelation für zwei Variablen [18]

Mithilfe dieser Korrelation können die Stärke und die Richtung linearer Beziehungen gemessen werden. Nicht-lineare Zusammenhänge können über diesem Ansatz nicht korrekt erfasst werden und resultieren in ungenauen Ergebnissen. Möglicherweise entstehen in diesem Fall sogar Ergebnisse, in denen der Korrelationskoeffizient gegen null tendiert. Für diese Korrelationsmethode wird außerdem vorausgesetzt, dass die verwendeten Daten normalverteilt und unabhängig voneinander sind [20]. Werden diese beiden Voraussetzungen nicht erfüllt, kann die Genauigkeit und Gültigkeit der Korrelationsergebnisse ebenfalls beeinträchtigt werden. Ausreißer in den Werten der untersuchten Variablen können die Ergebnisse dieser Korrelationsanalyse ebenfalls verzerren. Des Weiteren eignet sich die Pearson-Korrelation nur für metrisch skalierte Variablen. Für nominal- und ordinalskalierte Daten werden daher andere Korrelationsmethoden empfohlen [18].

Auf Grundlage der Pearson-Korrelation existiert außerdem eine besondere Ausprägung des Ansatzes, welche als punktbiseriale Korrelation bezeichnet wird. Dabei eignet sich die punktbiseriale Korrelation vor allem zur Untersuchung von zwei metrischen Variablen, in der eine der beiden Variablen als dichotom gilt. Demnach darf eine der beiden Variablen nur über zwei verschiedene Werte verfügen, welche in der Regel mit 0 und 1 kodiert werden [20].

4.6.2 Spearman-Rangkorrelation

Zur Berechnung von Zusammenhängen zwischen zwei Variablen, in denen mindestens ein ordinales Skalenniveau verwendet wird, eignen sich Methoden der Rangkorrelation. Im Gegensatz zur Pearson-Korrelation berücksichtigen diese Methoden nicht die ursprünglichen Werte zur Berechnung der Korrelationskoeffizienten. Stattdessen werden alle Werte in Ränge transformiert, auf deren Grundlage letztendlich die Korrelation der Variablen untersucht wird. Dadurch eignet sich diese Methode auch zur Auswertung nicht-linearer Beziehungen und es wird keine Normalverteilung der Variablenwerte vorausgesetzt [20].

Die Spearman-Rangkorrelation wurde von Charles Edward Spearman im Jahr 1904 entwickelt und stellt ein nichtparametrisches Äquivalent zur Korrelationsanalyse nach Pearson dar. Zur Transformation aller Variablenwerte wird jeder Wert durch ihre durchschnittliche Position auf dem ordinalen Skalenniveau abgeildet. Die Positionen der einzelnen Ränge werden dabei in aufsteigender Reihenfolge ermittelt. Anschließend kann die Korrelation auf Grundlage der transformierten Werte unter Verwendung der Formel zur Berechnung der Pearson-Korrelation ermittelt werden [21]. Eine Vereinfachung dieser Formel wird in Abbildung 13 dargestellt.

Obwohl die Transformation der Variablenwerte keine Normalverteilung der Da-

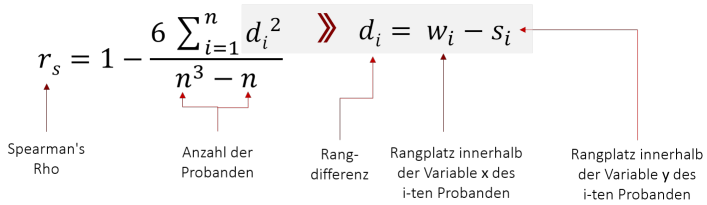


Abbildung 13: Formel zur Berechnung der Spearman-Korrelation für zwei Variablen [18]

ten voraussetzt, müssen die Werte dennoch eine monotone Beziehung zueinander aufweisen. Andernfalls kann die Gültigkeit und Genauigkeit der resultierenden Ergebnisse beeinflusst werden. Darüber hinaus ist die Unabhängigkeit der zu untersuchenden Variablen ebenfalls erforderlich. Außerdem gilt dieser Ansatz nicht für nominale Skalenniveaus.

4.6.3 Kendall'sche Rangkorrelation

Neben dem Ansatz von Spearman existiert ein weiteres Korrelationsmaß zur Berechnung von Rangkorrelationen, welches als Kendall'sche Rangkorrelation bezeichnet wird und im Jahr 1938 von Maruice Kendall entwickelt wurde. Die Transformation der Variablen-Werte beruht hierbei auf dem gleichen Prinzip, welcher für die Spearman-Rangkorrelation angewendet wird. Die Berechnung der Korrelationskoeffizienten wird jedoch auf der Überprüfung der Übereinstimmungen von Rängen aller Wert-Paare ermittelt. Zwei Wert-Paare gelten in diesem Kontext, als übereinstimmend, sobald der Rang beider Variablen-Werte eines Elements entweder größer oder kleiner als die Ränge der beiden Variablen-Werte des anderen Elements sind. Ist der Rang eines Variablen-Werts des einen Elements jedoch größer als der Rang dieses Variablen-Werts des anderen Elements, während der Rang des anderen Variablen-Werts für das eine Element geringer als der Rang des anderen Elements ist, gilt ein Wert-Paar als nicht übereinstimmend. Sind die Ränge beider Variablen-Werte für zwei Elemente identisch, ist dieses Wert-Paar weder übereinstimmend noch nicht übereinstimmend [21].

Zur Berechnung von Korrelationskoeffizienten muss letztendlich die Anzahl der übereinstimmenden und der nicht übereinstimmenden Wert-Paare für zu untersuchende Variablen ermittelt Werten. Anschließend wird die Anzahl aller nicht übereinstimmenden Wert-Paare von der Anzahl übereinstimmender Wert-Paare subtrahiert und durch die Anzahl maximal möglicher Wert-Paare dividiert. In Abbildung 14 wird eine Formel zur Berechnung der Kendall'schen Rangkorrelation darge-

stellt, welche die maximale Anzahl an Wert-Paaren als Summe übereinstimmender und nicht übereinstimmender Wert-Paare sowie der Anzahl an Bindungen in beiden Variablen ermittelt [21]. Diese Bindungen stellen bei der Untersuchung der Übereinstimmung alle Wert-Paare dar, welche über den gleichen Rang verfügen.

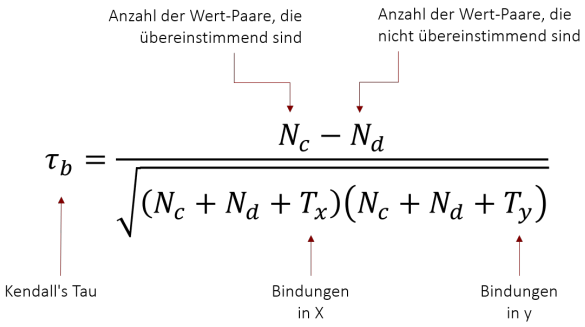


Abbildung 14: Formel zur Berechnung der Kendall-Korrelation für zwei Variablen [18]

4.6.4 Korrelationsbasierte Merkmalsauswahl

Die Merkmalsauswahl ist ein Prozess zur Identifikation redundanter und irrelevanter Informationen zur Reduzierung der Dimensionalität für Aufgaben des maschinellen Lernens. Durch diese Reduzierung können Algorithmen schneller und effektiver ausgeführt werden. Darüber hinaus kann die Genauigkeit von Klassifizierungen erhöht und die Interpetation ihrer Ergebnisse vereinfacht werden.

Die Korrelationsbasierte Merkmalsauswahl ist ein Ansatz der Filter-Methoden. Mittels Filter-Methoden werden untersuchte Merkmale anhand von Heuristiken ausgewählt, welche auf allgemeinen Charakteristiken der Daten basieren. In diesem Fall der Merkmalsauswahl wird die Korrelation als grundlegende Heuristik verwendet, wodurch eine Teilmenge aller Merkmale gefunden werden soll, welche eine hohe Korrelation zur Zielgröße verfügt. Die gefundene Teilmenge soll dabei geringe Korrelationen zwischen ihren einzelnen Merkmalen aufweisen, um mögliche Redundanzen zu vermeiden [22].

Zur Ermittlung dieser Teilmenge wird ein Gütemaß verwendet, deren Berechnungsgrundlage in Abbildung 15 enthalten ist. Diese Formel wird verwendet, um das Gütemaß einer beliebigen Teilmenge s zu berechnen, welche k Merkmale enthält. Die Variable $\overline{r_{cf}}$ repräsentiert in dieser Gleichung die durchschnittliche Korrelation aller Merkmale hinsichtlich der Zielgröße. $\overline{r_{ff}}$ stellt wiederum die durchschnittliche, intrinsische Korrelation zwischen allen Merkmalen der Teilmenge dar [22].

$$Merit_s = \frac{k\overline{r_{cf}}}{\sqrt{k+k(k-1)\overline{r_{ff}}}}$$

Abbildung 15: Formel zur Berechnung des Gütemaßes korrelationsbasierter Merkmalsauswahl [23]

Dieses Gütemaß ist eine Interpetation der Pearson-Korrelation, in der alle enthaltenen Variablen standardisiert worden. Der Zähler der Gleichung gilt als Indikator bezüglich der Vorhersagekraft der jeweiligen Teilmenge gegenüber ihrer Zielgröße. Der Nenner der Gleichung repräsentiert wiederum die Redundanzen zwischen den einzelnen Merkmalen der Teilmenge [22].

Für die Suche der besten Teilmenge existieren für n Merkmale 2^n mögliche Kombinationen. Daher wird zur Erhöhung der Performanz des Algorithmus der Ansatz der *Bestensuche* (*Best-First-Search*) verwendet. Der Algorithmus startet initial mit einer leeren Merkmalsmenge und erzeugt anschließend alle Teilmengen, welche ein einzelnes Merkmal enthalten. Anschließend wird das Gütemaß für jede Teilmenge berechnet und die Teilmenge mit der höchsten Korrelation für alle folgenden Prozesse ausgewählt. Zur Ermittlung dieser Teilmenge kann die Formel zur Berechnung des Gütemaßes vereinfacht werden, da bei der Untersuchung einzelner Merkmale keine Korrelationen innerhalb der Teilmengen existieren können. Die Vereinfachung der Formel wird in Abbildung 16 visualisiert [22].

$$\frac{k\overline{r_{cf}}}{\sqrt{k+k(k-1)\overline{r_{ff}}}} = \frac{1\overline{r_{cf}}}{\sqrt{1+1(1-1)\overline{r_{ff}}}} = \frac{\overline{r_{cf}}}{\sqrt{1}} = \overline{r_{cf}}$$

Abbildung 16: Formel zur Berechnung des Gütemaßes für Teilmengen einzelner Merkmale [23]

Im nächsten Schritt wird die ausgewählte Teilmenge durch jedes weitere einzelne Merkmal erweitert und deren Gütemaße berechnet. Jede Teilmenge, welche ein schlechteres Gütemaß aufweist, wird dabei verworfen. Dieser Schritt wird solange wiederholt, bis der gesamte Merkmalsraum durchsucht wurde und keine weitere Teilmenge gebildet werden kann. Letztendlich wird die Merkmalsteilmenge mit dem höchsten Gütemaß ausgegeben, welche das Endergebnis der korrelationsbasierten Merkmalsauswahl darstellt.

4.6.5 Anwendung der Korrelationsanalyse

Durch die Anwendung der Techniken der Vorverarbeitung können Korrelationsanalysen auf den jeweiligen Datensätzen durchgeführt werden. Dazu werden die Da-

tensätze vor ihrer Aufteilung in Trainings-, Test- und Validierungsdaten verwendet. Ausschlaggebend für diese Untersuchungen sind die Korrelationen zwischen den einzelnen Merkmalen und den beiden Zielgrößen.

Grundlegend sollen die in Kapitel 4.6 vorgestellten Korrelationsmaße nach Pearson, Spearman und Kendall verwendet werden. Mithilfe der unterschiedlichen Interpretationen des Korrelationsmaßes sollen die Beziehungen zwischen den Merkmalen und den Zielgrößen umfassend analysiert werden. So können neben der Untersuchung linearer Beziehungen durch die Pearson-Korrelation ebenfalls potenzielle nichtlineare Rangkorellationen erfasst werden. Darüber hinaus sollen die Ergebnisse zur Vereinfachung ihrer Auswertung visuell dargestellt werden.

Die konkrete Implementierung der Korrelationsanalyse wird in Kapitel 5.3 behandelt.

4.6.6 Anwendung korrelationsbasierter Merkmalsauswahl

Mittels der korrelationsbasierten Merkmalsauswahl sollen die Dimensionalitäten von Merkmalsmengen basierend auf den Korrelationseigenschaften zwischen verschiedenen Merkmalen sowie den beiden Zielgrößen reduziert werden. Aufgrund der Definition unterschiedlicher Modelle künstlicher neuronaler Netze muss diese Technik für jede Zielgröße eines Datensatzes separat durchgeführt werden. Dabei wird die korrelationsbasierte Merkmalsauswahl ebenfalls wie die anderen Methoden der Korrelationsanalyse nach Anwendung der Techniken der Vorverarbeitung und vor der Aufteilung der Datensätze in Trainings-, Test- und Validierungsdaten angewandt.

Die Zielgröße zur Erkennung des Auftretens einer Schwachstelle wird durch eine dichotome Variable dargestellt. Dadurch eignet sich für diesen Anwendungsfall die Methode der punktbiserialen Korrelation, welche in Kapitel 4.6.1 vorgestellt wurde. Für die Zielgröße zur Bewertung der Schwere von Schwachstellen wird jedoch die Pearson-Korrelation verwendet, da diese Zielgröße in einem kontinuierlichen Wertebereich repräsentiert wird. Dementsprechend können sich die Ergebnisse beider Anwendungsfälle signifikant voneinander unterscheiden, sodass sechs weitere Datensätze entstehen. Auf Grundlage der Ergebnisse der Korrelationsanalysen können außerdem die Ergebnisse der korrelationsbasierten Merkmalsauswahl besser nachvollzogen werden.

Eine konkrete Implementierung dieser Methodik wird in Kapitel 5.4 vorgestellt.

4.7 Definition der Modelle neuronaler Netze

Im Rahmen der Experimente müssen für die Aufgaben der Erkennung des Auftretens von Schwachstellen und der Bewertung der Schwere von Schwachstellen verschiedene Modelle künstlicher neuronaler Netze erzeugt werden. Für beide Anwendungsfälle existieren sechs unterschiedliche Datensätze. Jeweils zwei dieser Datensätze enthalten alle Metriken, welche entweder durch Understand oder durch Analizo extrahiert werden konnten und jeweils einen Datensatz, welcher alle extrahierten Metriken beider Werkzeuge umfasst. Auf Grundlage dieser drei Arten von Datensätzen wurden wiederum unter Anwendung der korrelationsbasierten Merkmalsauswahl drei zusätzliche Datensätze für jeden Anwendungsfall erstellt.

Für jeden dieser Datensätze werden drei Architekturen künstlicher neuronaler Netze entworfen, die sich in ihrer Netzwerkgröße unterscheiden, da die optimale Anzahl verwendeter Neuronen und Netzwerkschichten von der jeweiligen Problemstellung abhängt und nicht deterministisch ermittelt werden kann. Diese Architekturen werden für jeden Datensatz aufgrund ihrer unterschiedlichen Netzwerkgröße und Komplexität als S-, M- und L-Architektur bezeichnet. Die auf Grundlage einer Architektur erzeugten Modelle verfügen für jeden Datensatz über den gleichen charakteristischen Aufbau innerhalb ihrer verborgenen Schichten. Die konkrete Struktur der verborgenen Schichten wird für Modelle der S-Architektur in Kapitel 4.7.1, für die Modelle der M-Architektur in Kapitel 4.7.2 sowie für Modelle der L-Architektur in Kapitel 4.7.3 beschrieben.

Im Wesentlichen unterscheiden sich die konkreten Ausprägungen der Architekturen in der Anzahl der verwendeten Neuronen innerhalb von Eingabeschichten sowie in der Auswahl der Aktivierungsfunktionen von Ausgabeschichten. Die Anzahl der der Neuronen entspricht der Anzahl der durch einen Datensatz bereitgestellten Merkmale und hängt nicht von der letztendlichen Architektur des jeweiligen Modells ab. Da jedes konkrete Modell nur zur Vorhersage einer Zielgröße verwendet wird, verfügt jedes Modell unabhängig von seinem Anwendungsfall und seiner Netzwerkgröße nur über ein einzelnes Neuron innerhalb der Ausgabeschicht. Je nach Anwendungsfall unterscheiden sich jedoch die ausgewählten Aktivierungsfunktionen der konkreten Modelle.

Die Erkennung des Auftretens von Schwachstellen stellt eine Aufgabe der binären Klassifikation dar, sodass sich die Verwendung der Sigmoid-Aktivierungsfunktion in diesem Anwendungsfall eignet. Mittels dieser Aktivierungsfunktion können Ausgaben im Intervall $[0, 1]$ erzeugt werden, sodass Ergebnisse mithilfe eines vordefinierten Schwellenwerts als Schwachstellen identifiziert werden können. Die Bewertung der Schwere von Schwachstellen stellt hingegen eine Regressionsaufgabe dar, da die

Vorhersagen dieser Problemstellung in einem kontinuierlichen Wertebereich dargestellt werden sollen. Auf Grundlage der $CVSS_3$-Wertungen wird dieser Wertebereich durch das Intervall $[0, 10]$ definiert, sodass sich die Verwendung einer linearen Aktivierungsfunktion für die Modelle dieser Anwendungsfälle eignet.

4.7.1 Aufbau der Modelle der S-Architektur

Die Modelle künstlicher neuronaler Netze der S-Architektur verfügen über nur eine verborgene Schicht. Diese verborgene Schicht enthält zehn Neuronen, die eine ReLU-Aktivierungsfunktion verwenden und mit allen Neuronen der Eingabeschicht sowie dem Neuron der Ausgabeschicht verknüpft sind. In allen Neuronen dieser Schicht wird die gewichtete Summe basierend auf den Ausgabewerten der Eingangsneuronen und den entsprechenden Verbindungsgewichten berechnet. Anschließend wird basierend auf der Aktivierungsfunktion stets dieser berechnete Wert als Eingabewert des Neurons der Ausgabeschicht übergeben, solange der jeweilige Wert positiv ist. Andernfalls wird der Wert 0 an das Neuron der Ausgabeschicht weitergeleitet.

4.7.2 Aufbau der Modelle der M-Architektur

Die Modelle künstlicher neuronaler Netze der M-Architektur verfügen über fünf verborgene Schichten. Ausgehend von der Eingabeschicht enthalten die ersten beiden verborgenen Schichten jeweils 200 Neuronen. Die dritte verborgene Schicht besteht aus 150 Neuronen, während die vierte verborgene Schicht 120 Neuronen enthält und die fünfte verborgene Schicht über 70 Neuronen verfügt. In allen verborgenen Schichten dieser Architektur wird analog zur S-Architektur die ReLU-Aktivierungsfunktion verwendet. Aufgrund der Definition dieser Architekturen als Feedforward-Netzwerke besteht eine strukturelle Verbindung zwischen allen Schichten, sodass jedes Neuron einer verborgenen Schicht mit jedem Neuron ihrer benachbarten Schichten verknüpft ist. Darüber hinaus werden zwischen diesen verborgenen Schichten die Techniken des Dropouts und der Normalisierung angwendet, um eine Reduzierung des Verallgemeinerungsfehlers zu erzielen.

4.7.3 Aufbau der Modelle der L-Architektur

Die Modelle künstlicher neuronaler Netze der L-Architektur verfügen über zehn verborgene Schichten. Ausgehend von der Eingabeschicht enthält die erste verborgene Schicht 500 Neuronen. Die zweite verborgene Schicht dieser Architektur verfügt wiederum über 450 Neuronen. Die dritte verborgene Schicht besteht wiederum aus 400 Neuronen und die vierte verborgene Schicht beinhaltet 350 Neuronen. In der fünften,

sechsten und siebten verborgenen Schicht sind 300, 250 und 200 Neuronen enthalten.

Die letzten drei verborgenen Schichten dieser Architektur sind identisch zu den letzten drei Schichten der M-Architektur strukturiert. Dementsprechend enthalten diese Schichten 150, 120 und 70 Neuronen. Weiterhin verwendet jede verborgene Schicht ebenfalls die ReLU-Aktivierungsfunktion. Darüber hinaus werden zwischen allen verborgenen Schichten ebenfalls die Techniken des Dropouts und der Normalisierung angewandt. Aufgrund der Definition dieser Architektur als Feedforward-Netzwerke besteht außerdem gleichermaßen eine strukturelle Verknüpfung zwischen allen Schichten.

4.8 Konfiguration der Ausführung und Auswertung

Auf Grundlage der bereitgestellten Datensätze können die konkreten Modelle der verschiedenen Architekturen künstlicher neuronaler Netze trainiert und deren resultierenden Ergebnisse ausgewertet werden. Da für beide Anwendungsfälle sechs verschiedene Datensätze definiert wurden, die in den drei verschiedenen Architekturen künstlicher neuronaler Netze eingesetzt werden sollen, können 36 Experimente definiert werden. Dementsprechend existiert für jedes dieser Experimente ein separates Modell eines künstlichen neuronalen Netzes. Die Konfiguration der Ausführung und Auswertung dieser Experimente unterscheidet sich grundlegend anhand der zugrunde liegenden Problemstellung, sodass die Ausführung und Auswertung der Experimente der gleichen Problemstellung identisch verlaufen. Eine Beschreibung der Konfiguration der Experimente binärer Klassifikation wird in Kapitel 4.8.1 beschrieben, während die Konfiguration der Experimente der Regressionsaufgaben in Kapitel 4.8.2 erläutert wird.

4.8.1 Konfiguration der Experimente binärer Klassifikation

Für das Training der Modelle künstlicher neuronaler Netze zur Erkennung des Auftretens von Schwachstellen wird als Optimierungsfunktion der stochastische Gradientenabstieg in Kombination mit der binären Kreuzentropie als Verlustfunktion verwendet. Dabei soll das Training in 20 Epochen ohne konkrete Festlegung der Batch-Größe einzelner Epochen stattfinden. Nach der Verarbeitung jeder einzelnen Epoche soll das Zwischenergebnis des Trainings durch Validierungsdaten überprüft werden, ehe die Performanz des Modells nach Abschluss des Trainings durch die Verwendung der Testdaten analysiert wird. Die Klassifikation von Vorhersagen dieses Modells sollen wiederum unter Verwendung des Schwellenwerts 0.5 bestimmt werden. Untersuchte Beispiele, für die ein Ergebnis größer als der Schwellenwert

vorhergesagt wird, sollen in diesem Kontext als Beispiele klassifziert werden, in denen eine Schwachstelle auftritt. Andernfalls werden solche Beispiele als Exemplare ohne Auftreten einer Schwachstelle klassifiziert.

Zur Auswertung der Leistung der Modelle sollen die Performanzmetriken der *binären Genauigkeit*, *Recall* und *AUC* verwendet werden. Die Metrik der binären Genauigkeit repräsentiert das Verhältnis hinsichtlich der Anzahl korrekt vorhergesagter Beispiele im Verhältnis zur Anzahl aller getätigten Vorhersagen. Mithilfe der Performanzmetrik des Recalls wird wiederum das Verhältnis zwischen der Anzahl aller korrekt vorhergesagten Beispiele, bei denen eine Schwachstelle auftritt, in Relation zu allen Beispielen dargestellt, in denen tatsächlich eine Schwachstelle auftritt.

Die AUC-Metrik basiert hingegen auf den Metriken der Falsch-Positiv-Rate und der Sensitivität. Dabei wird die Falsch-Positiv-Rate als Proportion zwischen der Anzahl falsch klassifizierten Beispiele, für die das Auftreten einer Schwachstelle vorhergesagt wird und der Anzahl aller Beispiele definiert, in denen tatsächlich keine Schwachstelle auftritt. Mittels der Sensitivität wird wiederum ein Maß zwischen der Anzahl korrekt klassifizierter Beispiele mit Auftreten von Schwachstellen im Verhältnis zur Anzahl aller Beispiele beschrieben, in denen tatsächlich Schwachstellen auftreten. Auf Grundlage dieser Metriken kann durch Variation des Schwellenwerts eine Funktion im Intervall $[0, 1]$ ermittelt werden, dessen Integral als AUC-Metrik definiert wird.

Nach der Durchführung aller Trainingsschritte und der Auswertung des Testdatensatzes sollen die Endergebnisse der Metriken der binären Genauigkeit, des Recalls sowie die AUC-Metrik für Trainings-, Test- und Validierungsdatensätze ausgegeben werden. Darüber hinaus soll für jeden dieser Datensätze die Metrik der *Präzision* ermittelt werden, welche das Verhältnis zwischen korrekt klassifizierten Beispieln, in denen eine Schwachstelle auftritt und allen Beispielen, für die das Auftreten einer Schwachstelle vorhergesagt wird, wiederspiegelt. Für den Testdatensatz soll außerdem die Metrik des *F1-Scores* berechnet werden, welche das harmonische Mittel zwischen Präzision und Recall repräsentiert. Außerdem sollen die Zwischenergebnisse von Metriken der binären Genaugikeit, des Recalls und der AUC-Metrik sowie des Fehlers der Verlustfunktion nach Verarbeitung jeder Epoche in Diagrammen visualisiert werden.

4.8.2 Konfiguration der Experimente der Regressionsaufgaben

Das Training der Modelle künstlicher neuronaler Netze zur Bewertung der Schwere von Schwachstellen basiert auf der Verwendung der Optimierungsfunktion *Adam* in Kombination mit dem mittleren quardratischen Fehler als Verlustfunktion. Die

Adam-Optimierungsfunktion wurde auf Grundlage des stochastischen Gradientenabstiegs und weiteren Optimierungsalgorithmen entworfen. Mithilfe quardratischer Gradienten und der Verwendung des Momentums verfügt Adam über eine adaptive Lernrate für unterschiedliche Parameter.

Der Trainingsprozess basiert wiederum auf der Definition von 100 Epochen und einer Batch-Größe von 10. Auf dieser Grundlage soll eine Kreuzvalidierung stattfinden, in der die Trainingsdatensätze in zehn verschiedene Teilmengen aufgeteilt wird, um zehn unabhängige Modelle jeder Architektur zu trainieren. Dabei wird jeweils eine dieser Teilmengen zur Validierung verwendet, während die restlichen Teilmengen für das eigentliche Training eingesetzt werden. Das resultierende Modell mit der besten Performanz wird letztendlich im Rahmen der Auswertungen betrachtet.

Zur Auswertung dieser Modelle soll die Performanzmetrik der Genauigkeit verwendet werden, deren Prinzip der Metrik der binären Genauigkeit ähnelt. Nach Durchlauf aller Epochen des Trainingsprozesses sollen die Zwischenergebnisse dieser Metrik sowie der berechnete Fehler der Verlustfunktion für alle Epochen der Trainings- und Validierungsdaten durch Diagramme visualisiert werden. Darüber hinaus soll der Mittelwert, das Maximum und das Minimum des mittleren quadratischen Fehlers inklusive des mittleren abosluten Fehlers aller Vorhersagen ausgegeben werden.

5 Implementierung der Experimente

Auf Grundlage der wissenschaftlichen Methodik zur Beantwortung der Forschungsfragen müssen verschiedene Implementierungsentscheidungen getroffen werden, welche im Rahmen dieses Kapitels begründet werden. Grundlegend wird zur Erstellung aller Prozesse die Programmiersprache *Python* verwendet, da der Syntax einfach strukturiert und leicht verständlich ist. Darüber hinaus verfügt diese Programmiersprache über eine große Auswahl an Bibliotheken, die speziell für die Herausforderungen und Aufgaben des maschinellen Lernens entwickelt wurden. Außerdem ist Python plattformunabhängig, sodass der Quellcode auf verschiedenen Umgebungen ausgeführt werden kann. Der Quellcode kann außerdem problemlos in anderen Programmiersprachen wie C++ oder Java integriert werden, sodass die Funktionalitäten von Python ebenfalls durch andere Programmiersprachen verwendbar sind.

Mithilfe dieser Programmiersprache wurden die einzelnen Prozesse des gesamten Experiments implementiert. Die Prozesse zur Aktualisierung des CVEfixes-Datensatzes und zur Extraktion der Software-Produktmetriken unter Verwendung der Metriken-Extraktionstools Understand und Analizo werden in Kapitel 5.1 thematisiert. Die Bereitstelung des Roh-Datensatzes in einem einfachen Datenformat lesbarer Textform sowie die Implementierung der Methoden zur Vorverarbeitung des Datensatzes werden wiederum in Kapitel 5.2 beschrieben. In Kapitel 5.3 werden die verwendeten Methoden zur Durchführung der Korrelationsanalyse vorgestellt und in Kapitel 5.4 der Algorithmus zur Implementierung der korrelationsbasierten Merkmalsauswahl behandelt. Die Implementierung der Modelle künstlicher neuronaler Netze sowie die Prozesse des Trainings und der Auswertung dieser Modelle werden in Kapitel 5.5 erläutert. Darüber hinaus werden alle Implementierungen im Rahmen eines GitLab-Projekts[1] bereitgestellt.

5.1 Implementierung der Metirken-Extraktion

Der automatisierte Prozess zur Aktualisierung des CVEfixes-Datensatzes wurde ebenfalls in der Porgrammiersprache Python implementiert und verfügt über eine Integration innerhalb eines Bash-Skriptes. Dementsprechend kann dieser Datensatz durch die entsprechenden Funktionalitäten der Python-Programme oder durch Ausführung auf Kommandozeilenebene durchgeführt werden. Die Auswertung der Aktualisierung des CVEfixes-Datensatz wird in Kapitel 6.1 behandelt.

Auf Grundlage des aktualisierten CVEfixes-Datensatz findet im Rahmen dieses Projekts die Extraktion von Software-Produktmetriken durch die Werkzeuge

[1] `https://git.uni-jena.de/we38tad/softwarequalitymetrics`

Analizo und Understand zur Erzeugung des Roh-Datensatzes statt. Dazu wird die allgemeine Implementierung des Prozesses zur Metriken-Extraktion in Kapitel 5.1.1 thematisiert. Die Integration des Metriken-Extraktionstools Understand wird in Kapitel 5.1.2 und die Integration des Metriken-Extraktionstools Analizo in Kapitel 5.1.3 behandelt.

5.1.1 Allgemeine Implementierung der Metriken-Extraktion

Zur Extraktion von Software-Produktmetriken unter Verwendung der Programmiersprache Python werden die Bibliotheken *os*, *shutil*, *sqlite3* und *pandas* benötigt. Mithilfe der Bibliothek os werden Interaktionen mit dem Betriebssystem ermöglicht, um beispielsweise Datei- und Verzeichnisoperationen ausführen zu können. Im Rahmen dieser Implementierung wird diese Bibliothek benötigt, um beispielsweise die Existenz bestimmter Verzeichnisse und Dateien zu überprüfen oder zu erstellen. Durch die shutil-Bibliothek werden diese Funktionalitäten der Datei- und Verzeichnisoperationen erweitert, sodass diese Entitäten kopiert, verschoben, gelöscht oder umbenannt werden können.

Die Bibliothek sqlite3 wird hingegen zur Interaktion mit SQLite-Datenbanken verwendet, auf deren Grundlage beispielsweise das relationale Datenbankschema des CVEfixes-Datensatzes realisiert wurde. Mithilfe dieser Bibliothek wird die Verbindung zur Datenbank des CVEfixes-Datensatz hergestellt und die Ausführung diverser SQL-Abfragen gewährleistet. Ergebnisse dieser SQL-Abfragen können wiederum durch die Verwendung der pandas-Bibliothek verarbeitet werden. Diese Bibliothek umfasst leistungsstarke Methoden zur Analyse, Verarbeitung und Visualisierung eindimensionaler und zweidimensionaler Datenstrukturen. Dadurch können Daten sowohl gefiltert, sortiert und gruppiert als auch aggregiert und transformiert werden. Darüber hinaus können Daten mithilfe dieser Bibliothek statistisch analysiert und durch die Integration weiterer Bibliotheken beispielsweise in Diagrammen oder Histogrammen visualisiert werden.

Algorithmus 1 beschreibt den schematischen Ablauf zur Extraktion von Software-Produktmetriken durch beide Werkzeuge der Metriken-Extraktion in Form von Pseudo-Code. Auf Grundlage des aktualisierten CVEfixes-Datensatzes existiert eine relationale Datenbank, deren Inhalte in einer Datei gespeichert werden. Mithilfe der Funktionalitäten der sqlite3-Bibliothek wird innerhalb der Funktion *createDb-Connection* eine Verbindung zur relationalen Datenbank des CVEfixes-Datensatzes hergestellt. Dabei wird dieser Funktion der entsprechende Dateipfad übergeben, der auf die entsprechende Datei referenziert, welche die Inhalte der relationalen Datenbank repräsentiert.

Algorithm 1 Verallgemeinerter Algorithmus zur Metriken-Extraktion

```
1: conn ← createDbConnection(db_file)
2: cursor ← getFileInformations(conn)
3: df_sql ← getProcessedMetrics(conn)
4: cursor ← skipProcessedMetrics(df_sql)
5: while True do
6:     result ← cursor.fetchOne()
7:     if result is empty then
8:         break
9:     end if
10:    file_id ← result[0]
11:    filename ← result[1]
12:    code_after ← result[2]
13:    code_before ← result[3]
14:    if code_after is None" then
15:        metrics ← extractMetrics(file_id, filename, code_after)
16:        insertMetricsToDb(conn, metrics)
17:    end if
18:    if code_before is None" then
19:        metrics ← extractMetrics(file_id, filename, code_before)
20:        insertMetricsToDb(conn, metrics)
21:    end if
22: end while
```

Basierend auf der hergestellten Datenbankverbindung werden durch die Funktion *getFileInformations* alle benötigten Informationen zur Extraktion von Software-Produktmetriken bereitgestellt. Für jede in diesem Datensatz referenzierte Datei soll die jeweilige ID, den entsprechenden Dateinamen und den Quellcode vor und nach Behebung einer Schwachstelle ermittelt werden. Dazu wird die folgende SQL-Abfrage unter erneuter Verwendung der sqlite3-Bibliothek ausgeführt:

```
SELECT file_change_id, filename, code_after, code_before
FROM file_change
```

Für diese Abfrage wird wiederum ein Zeiger bereitgestellt, der auf das Ergebnis dieser Abfrage referenziert und durch die Funktion *fetchOne* jeweils ein Tupel dieser Ergebnismenge ausgibt. Um die Idempotenz des Prozesses der Metriken-Extraktion zu gewährleisten, sollen nur für die Tupel der Ergebnismenge Software-Produktmetriken extrahiert werden, für die noch keine Metriken-Werte innerhalb der relationalen Datenbank des CVEfixes-Daten bereitgestellt wurden. Daher werden die Funktionen *getProcessedMetrics* und *skipProcessedMetrics* verwendet, um alle Tupel der Ergebnismenge herauszufiltern, für die bereits eine Metriken-Extraktion stattgefunden hat. Die Funktion getProcessedMetrics ermittelt daher alle Tupel extrahierter Metriken durch die Ausführung der folgenden verallgemeinerten SQL-Abfrage:

```
SELECT * FROM file_metrics_table
```

Das Ergebnis dieser Abfrage wird wiederum in einer zweidimensionalen Datenstruktur der pandas-Bibliothek hinterlegt. Auf Grundlage dieses Abfrageergebnisses soll durch die Funktion skipProcessedMetrics der Zeiger der ersten SQL-Abfrage auf die korrekte Position verschoben werden. Unter der Annahme einer iterativen Metriken-Extraktion und einer unveränderten Reihenfolge aller Tupel der ersten Abfrage wird lediglich der Wert der *file_change_id* des letzten Eintrags der pandas-Datenstruktur benötigt. Nach Ermittlung dieses Werts wird innerhalb der skipProcessedMetrics-Funktion die fetchOne-Funktion solange wiederholt ausgeführt, bis der Wert des Attributs file_change_id des aktuellen Tupels dem ermittelten Wert entspricht.

Basierend auf diesen Schritten werden anschließend im Rahmen einer while-Schleife ohne Abbruchbedingung alle Software-Produktmetriken iterativ extrahiert. Unter Verwendung der fetchOne-Funktion auf dem Zeiger des Abfrageergebnisses wird innerhalb eines jeden Schleifendurchlaufs jeweils ein Tupel bereitgestellt, dessen Informationen in den Zeilen 10 bis 13 in separaten Variablen repliziert wird. Sobald innerhalb eines Schleifendurchlaufs durch Anwendung der fetchOne-Funktion auf

dem Zeiger ein leeres Tupel übergeben wird, soll die in den Zeilen 7 bis 9 definierte If-Anweisung den Algorithmus beenden, da in diesem Fall bereits alle bereitgestellten Tupel verarbeitet wurden.

Auf Grundlage der Bereitstellung aller Tupel-Informationen in separaten Variablen sollen jeweils für den Quellcode vor und nach Behebung einer Schwachstelle die entsprechenden Software-Produktmetriken extrahiert werden. Der entsprechende Quellcode ist in den Variablen *code_before* und *code_after* enthalten, welche in bestimmten Sonderfällen die Zeichenkette *None* enthalten. Diese Sonderfälle treten ein, sofern der Quellcode für ein Beispiel aufgrund des Hinzufügens oder Entfernens entsprechender Dateien zur Behebung von Schwachstellen in dem jeweiligen Szenario nicht existieren.

Andernfalls werden durch die spezifischen Implementierungen der Funktion *extractMetrics* die jeweiligen Software-Produktmetriken durch die Werkzeuge Analizo und Understand auf Grundlage des entsprechenden Quellcodes extrahiert und in einer zweidimensionalen pandas-Datenstruktur bereitgestellt. Diese Datenstruktur wird außerdem zur Kennzeichnung der extrahierten Daten bezüglich der Existenz einer Schwachstelle um ein zusätzliches Attribut namens *before_change* erweitert. Dementsprechend wird das Attribut before_change für alle extrahierten Metriken der Variable code_before mit dem Wert 0 initialisiert und für alle extrahierten Metriken der Variable code_after mit dem Wert 1 initialisiert.

Die letztendliche Integration der Software-Produktmetriken zur Erweiterung des Schemas der relationalen Datenbank des CVEfixes-Datensatzes findet durch die Funktion *insertMetricsToDb* statt. Innerhalb dieser Funktion werden unter Verwendung der Funktionalitäten der pandas- und sqlite3-Bibliotheken die entsprechenden Daten in separaten Datenbanktabellen integriert.

5.1.2 Integration des Understand-Tools

Die Extraktion von Software-Produktmetriken aus einem zu untersuchenden Quellcode findet auf Grundlage der Erzeugung einer Datei mit dem entsprechenden Dateinamen in einem separaten Verzeichnis unter Verwendung der Bibliotheken os und shutil statt. Für jeden Extraktionsprozess wird dieses Verzeichnis zuvor bereinigt, um entsprechende Konflikte zwischen verschiedenen Dateien während der Metriken-Extraktion zu vermeiden. Zur Metriken-Extraktion durch das Werkzeug Understand wird außerdem die Bibliothek *subprocess* benötigt, welche das Starten externer Prozesse sowie deren Steuerung und weitere Interaktionen ermöglicht.

Mithilfe dieser Bibliothek wird der folgende Understand-Befehl auf Kommandozeilenebene ausgeführt, um eine Understand-Datei zu erzeugen, auf dessen Grund-

lage die erzeugte Datei analysiert wird und anschließend Software-Produktmetriken ermittelt werden:

```
und create -db udb_file create -languages all add
    file_path analyze metrics
```

Die Parameter *udb_file* und *file_path* repräsentieren in diesem Zusammenhang den Namen der Understand-Datei sowie den Verzeichnispfad, in dem die zu untersuchende Quellcode-Datei erzeugt wurde. Durch die Ausführung dieses Befehls wird im gleichen Verzeichnispfad eine CSV-Datei erzeugt, welche die Software-Produktmetriken aller Entitäten der Quellcode-Datei enthält. Mittels der pandas-Bibliothek wird wiederum der Inhalt dieser CSV-Datei als zweidimensionales Array initialisiert, in dem die Metriken der untersuchten Datei auf Grundlage ihres Dateinamens lokalisiert werden können. Dementsprechend werden die Metriken dieser Entiät aus der pandas-Datenstruktur extrahiert. Anschließend werden diese Daten in der Datenbanktabelle *file_metrics_understand* integriert, welche eine Erweiterung des relationalen Datenbankschemas des CVEfixes-Datensatzes darstellt.

5.1.3 Integration des Analizo-Tools

Die Integration des Analizo-Tools findet ebenfalls auf der Erzeugung von Dateien für den zu untersuchenden Quellcode auf Grundlage der Python-Bibliotheken os und shutil sowie der Verwendung der subprocess-Bibliothek statt. Zur Berechnung von Software-Produktmetriken durch das Werkzeug Analizo wird der folgende Befehl unter Verwendung der Bibliothek subprocess ausgeführt:

```
analizo metrics --output yaml_file file_path
```

Dabei repräsentiert der Parameter file_path ebenfalls den Verzeichnispfad, in dem die zu untersuchende Quellcode-Datei erzeugt wurde. Der Parameter *yaml_file* definiert hingegen den Namen der YAML-Datei, in der die berechneten Software-Produktmetriken bereitgestellt werden. Zur Auswertung von YAML-Dateien durch die Programmiersprache Python wird zusätzlich die *yaml*-Bibliothek benötigt, welche die Verarbeitung solcher Dateien ermöglicht. Mithilfe dieser Bibliothek werden die Inhalte der erzeugten YAML-Datei eingelesen und unter Verwendung der pandas-Bibliothek in eine zweidimensionale Datenstruktur übertragen. Anschließend werden die entsprechenden Daten in der Datenbanktabelle *file_metrics_analizo* integriert, welche eine Erweiterung des relationalen Datenbankschemas des CVEfixes-Datensatzes darstellt.

5.2 Methoden zur Erstellung der Datensätze

Auf Grundlage der implementierten Prozesse zur Extraktion und Bereitstellung von Software-Produktmetriken werden drei verschiedene Roh-Datensätze definiert, auf denen verschiedene Methoden der Vorverarbeitung angewandt werden. Die Implementierung zur Erstellung dieser Roh-Datensätze wird in Kapitel 5.2.1 behandelt, während die Implementierung von Methoden der Vorverarbeitung in Kapitel 5.2.2 thematisiert wird.

5.2.1 Bereitstellung der Roh-Datensätze

Die Erstellung der drei benötigten Roh-Datensätze basiert auf der Verwendung der Python-Bibliotheken sqlite3 und pandas. Unter Verwendung der sqlite3-Bibliothek wird eine Verbindung zur relationalen Datenbank des CVEfixes-Datensatzes hergestellt. Auf Grundlage dieser Verbindung werden verschiedene SQL-Abfragen ausgeführt, um alle relevanten Informationen zur Erzeugung der Roh-Datensätze zu erhalten. Für alle drei verschiedenen Datensätze wird die folgende SQL-Abfrage verwendet, um alle benötigten Datei-Informationen bereitzustellen:

```
SELECT file_change_id, hash, programming_language
FROM file_change
```

Zur Erstellung des Datensatzes werden zusätzliche Informationen wie beispielsweise die CVSS3-Wertungen des entsprechenden CVE-Eintrags benötigt. Darüber hinaus wird das Attribut *repo_url* zur Aufteilung der Datensätze in Trainings-, Test- und Validierungsdaten benötigt. Dementsprechend wird ebenfalls die folgende SQL-Abfrage ausgeführt:

```
SELECT fixes.repo_url, fixes.hash, cve.cvss3_base_score
FROM fixes, cve
WHERE fixes.cve_id = cve.cve_id
```

Für die Extraktion der bereitgestellten Metriken des Understand-Tools wird außerdem die folgende SQL-Abfrage ausgeführt:

```
SELECT * FROM file_metrics_understand
```

Zur Extraktion der Software-Produktmetriken des Analizo-Tools wird wiederum die folgende SQL-Abfrage verwendet:

```
SELECT * FROM file_metrics_analizo
```

Die Ergebnisse aller SQL-Abfragen werden in zweidimensionalen Datenstrukturen der pandas-Bibliothek bereitgestellt. Zur Erzeugung der Roh-Datensätze werden diese Ergebnisse unter Verwendung der *merge*-Funktion der pandas-Bibliothek zusammengeführt. Dabei müssen für die Erstellung des kombinierten Roh-Datensatzes zuerst die Abfrageergebnisse der beiden Werkzeuge zur Metriken-Extraktion miteinander verschmolzen werden. Diese Zusammenführung basiert auf den Attributen *file_change_id* und *before_change*, welche in beiden Abfrageergebnissen enthalten sind. Für die Erstellung der anderen beiden Roh-Datensätze werden lediglich die entsprechenden Abfrageergebnisse der extrahierten Metriken-Werte verwendet.

Diese Metriken-Daten werden anschließend mit allen weiteren Informationen zusammengeführt. Dabei findet die Zusammenführung des ersten Abfrageergebnis mit den Metriken-Daten auf Grundlage des Attributs *file_change_id* statt. Die Integration der Informationen aus dem entsprechenden CVE-Eintrag basiert wiederum auf dem Attribut *hash*, welches in den ersten beiden Abfrageergebnissen enthalten ist.

Auf Grundlage dieser Zusammenführungen entsteht das Grundgerüst der drei Roh-Datensätze, welches jedoch weitere Verarbeitungsschritte benötigt. Einerseits werden durch die *drop*-Funktion der pandas-Bibliothek Metriken-Attribute entfernt, für die keine Daten durch die Werkzeuge der Metriken-Extraktion bereitgestellt werden. Andererseits müssen die Reihenfolgen aller Attribute der zweidimensionalen Datenstrukturen restrukturiert werden, um alle Merkmale der Datensätze von ihren entsprechenden Zielgrößen abzugrenzen. Diese Restrukturierung basiert auf der Verwendung einfacher Operationen, die durch den Einsatz der pandas-Bibliothek ermöglicht werden.

Alle Einträge dieses Datensatzes, für die keine $CVSS_3$-Wertung existiert, müssen außerdem herausgefiltert werden, um eine konsistente Datenbasis zu gewährleisten. Des Weiteren müssen die Werte aller Zielgrößen transformiert werden, da das Attribut *before_change* aufgrund der Bereitstellung durch SQL-Abfragen lediglich die Werte *True* und *False* bereitstellt und die CVSS3-Wertung für Einträge mit behobenen Schwachstellen nicht dem Wert 0 entspricht. Daher müssen durch die Ausführung der folgenden Python-Befehle die Werte der Zielgröße before_change durch die binären Werte 0 und 1 abgebildet sowie die Werte der $CVSS_3$-Wertungen entsprechend angepasst werden:

```
data.loc[data['before_change'] == 'False', 'cvss3_base_score'] =  0
data.loc[data['before_change'] == 'False', 'before_change'] =  0
data.loc[data['before_change'] == 'True', 'before_change'] =  1
```

Auf Grundlage dieser Anpassungen werden die Roh-Datensätze auf Grundlage der *to_json*-Funktion der pandas-Bibliothek letztendlich im JSON-Format gespei-

chert. Die Darstellung der Roh-Datensätze im JSON-Format eignet sich vor allem aufgrund der einfachen Lesbarkeit und der Plattformunabhängigkeit. Darüber hinaus ist dieses textbasierte Datenformat leichtgewicht sowie effizient und bietet eine flexible Verwendung der darin enthaltenen Daten.

5.2.2 Implementierung von Methoden der Vorverarbeitung

Zur Implementierung von Vorverarbeitungsmethoden wird die Python-Bibliothek *sklearn* verwendet. Diese Bibliothek stellt Algorithmen und Werkzeuge des maschinellen Lernens und der Datenanalyse bereit und verfügt über eine Vielzahl von Funktionen zur Vorverarbeitung von Daten.

Die Klassen *ColumnTransformer* und *OneHotEncoder* sind beispielsweise Bestandteile dieser Bibliothek und werden zur Transformation der kategorischen Werte des Merkmals *programming_language* verwendet. Mittels der Klasse ColumnTransformer können Transformationen auf bestimmten Spalten des Datensatzes angewendet werden, um beispielsweise kategorische Daten zu kodieren. Die Klasse OneHotEncoder dient wiederum zur Umwandlung kategorischer Werte in numerische Merkmale mit binärer Darstellung. Durch die Kombination dieser beiden Klassen der sklearn-Bibliothek wird jeder kategorische Wert des Merkmals programming_language durch ein binäres Merkmal repräsentiert und entsprechend im Datensatz integriert.

Zur Ersetzung fehlender Werte wird die Klasse *SimpleImputer* der Bibliothek sklearn verwendet, die zur Behandlung entsprechender fehlender Werte dient. Mithilfe dieser Klasse werden durch die folgenden Code-Zeilen die fehlenden Werte aller Datensätze konstant durch den Wert 0 ersetzt:

```
imputer_data = SimpleImputer(missing_values=np.nan,
                             strategy='constant', fill_value=0)
imputer_data = imputer_data.fit(data)
data = pd.DataFrame(imputer_data.transform(data),
                    columns=data.columns)
```

Die Aufteilung der Datensätze in Trainings-, Test- und Validierungsdaten wird auf Grundlage des Attributs *repo_url* realisiert, wodurch alle Einträge der Datensätze ihren entsprechenden Software-Produkten zugeordnet werden können. Durch die Verwendung der Klasse *GroupShuffleSplit* der sklearn-Bibliothek kann eine solche zufällige Aufteilung unter Berücksichtigung entsprechender Gruppenzugehörigkeiten gewährleistet werden. Eine einmalige Anwendung der Funktionalitäten dieser Klasse teilt einen Datensatz jedoch nur in zwei verschiedene Mengen auf. Daher muss unter Verwendung dieser Klasse die Aufteilung zweimal durchgeführt werden, um die

entsprechenden Datensätze zu generieren. Im Rahmen dieser Aufteilung sollen zirka 70% der Daten als Traininsdaten, 20% der Daten als Validierungsdaten und 10% der Daten als Testdaten bereitgestellt werden. Aufgrund der Aufteilung anhand der Zugehörigkeit zu einem bestimmten Software-Produkt kann eine exakte Verteilung basierend auf diesem Verhältnis nicht vollständig garantiert werden.

Nach der Aufteilung der Datensätze in Trainings-, Test- und Validierungsdaten müssen die Merkmale und Zielgrößen dieser Teilmengen voneinander separiert werden, um sie für das Training und die Auswertung künstlicher neuronaler Netze bereitzustellen. Mithilfe der Restrukturierung aus Kapitel 5.2.1 kann die Trennung von Zielgrößen und Merkmalen durch die Verwendung der *iloc*-Funktion der pandas-Bibliothek durch die folgenden Code-Zeilen realisiert werden:

```
x_data = data.iloc[:, :-2]
y_data = data.iloc[:, -2:]
```

Anschließend müssen die Werte aller Merkmale in den Wertebereich $[0, 1]$ normalisiert werden, um die Robustheit der Experimente gegenüber Ausreißern sicherzustellen und die Interpretierbarkeit durch die Modelle künstlicher neuronaler Netze zu erhöhen. Dazu wird die Klasse *MinMaxScaler* der sklearn-Bibliothek verwendet, auf dessen Grundlage Daten in diesem Wertebereich skaliert werden können. Diese Skalierung wird durch die Ausführung der folgenden Code-Zeilen realisiert:

```
mms_X = MinMaxScaler()
x_data_columns = x_data.columns
x_data = pd.DataFrame(mms_X.fit_transform(x_data),
                      columns=x_data_columns)
```

Auf Grundlage dieser Methoden der Vorverarbeitung werden letztendlich die Teilmengen von Zielgrößen und Merkmalen der Trainings-, Test- und Validierungsdaten im CSV-Format bereitgestellt. Dieses Format wird aufgrund seiner einfachen Struktur, einem geringen Speicherbedarf sowie den Eigenschaften der Plattformunabhängigkeit und der Kompatibilität mit verschiedenen Werkzeugen der Datenverarbeitung und -analyse verwendet.

5.3 Implementierung der Korrelationsanalyse

Für die Durchführung von Methoden der Korrelationsanalyse werden die Prozesse der Vorverarbeitung zur Ersetzung fehlender Daten sowie zur Transformation des kategorischen Merkmals programming_language benötigt. Diese Schritte müssen

dementsprechend auf dem Roh-Datensatz ausgeführt werden, um zuverlässige Korrelationskoeffizienten zu ermitteln. Eine Aufteilung der Daten in Trainings-, Test- und Validierungsdatensätze sowie die Trennung von Merkmalen und Zielgrößen sowie die Skalierung der Merkmalswerte dürfen wiederum vor der Durchführung von Korrelationsanalysen auf den Datensätzen nicht angewendet werden.

Die Berechnung von Korrelationskoeffizienten basiert auf der Funktion *corr* der pandas-Bibliothek. Mithilfe dieser Funktion wird eine Korrelationsmatrix zwischen allen Spalten einer pandas-Datenstruktur berechnet. Dieses Korrelationsmatrix ist quadratisch und enthält die Korrelationskoeffizienten zwischen allen Spalten der untersuchten Datenstruktur. Alle Koeffizienten sind in einem Wertebereich zwischen -1 und 1 definiert und können auf Grundlage der in Kapitel 4.6 beschriebenen theoretischen Grundlagen interpretiert werden.

Zur Bestimmung des verwendeten Korrelationsmaßes existiert der Parameter *method* innerhalb der corr-Funktion. Standardmäßig wird bei fehlender Übergabe dieses Parameters stets das Pearson-Korrelationsmaß zur Untersuchung linearer Zusammenhänge verwendet. Durch die Übergabe der Zeichenketten *spearman* und *kendall* können hingegen die Korrelationskoeffizienten der Korrelationsmaße Spearman und Kendall berechnet werden.

Auf Grundlage dieser Funktion werden die Korrelationskoeffizienten zwischen allen Merkmalen und Zielgrößen der Datensätze für alle drei Korrelationsmaße berechnet. Unter Verwendung der Klasse *heatmap* der Python-Bibliothek *seaborn* werden die resultierenden quadratischen Korrelationsmatrizen visualisiert. Generell dient die seaborn-Bibliothek zur Visualisierung statistischer Daten und verfügt über die Möglichkeit einer direkten Integration auf pandas-Datenstrukturen ohne zusätzlichen Aufwand. Die heatmap-Klasse eignet sich wiederum zur visuellen Darstellung von Daten in Form einer Matrix, in der verschiedene Farbtöne oder Schattierungen für unterschiedliche Intensitäten der Werte von Daten verwendet werden. Diese Visualisierungen dienen letztendlich zur optimierten Veranschaulichung und Auswertung der Ergebnisse der Korrelationsanalyse und werden in Form von skalaren Vektorgrafiken für weitere Analysen gespeichert.

5.4 Algorithmen korrelationsbasierter Merkmalsauswahl

Für die Durchführung der korrelationsbasierten Merkmalsauswahl müssen ebenfalls die Prozesse zur Ersetzung fehlender Daten und der Transformation kategorischer Merkmale durchgeführt werden. Darüber hinaus wird ein Algorithmus zur Berechnung des Gütemaßes benötigt, der auf Grundlage der Korrelationen zwischen einzelnen Merkmalen sowie der Korrelation zwischen den Merkmalen und der jeweiligen

Zielgröße basiert.

Eine Ausprägung dieses Algorithmus wird in Kapitel 5.4.1 präsentiert. Zur Ermittlung der optimalen Teilmenge der korrelationsbasierten Merkmalsauswahl auf Grundlage dieses Gütemaßes wird außerdem die Datenstruktur einer Vorrangwarteschlange verwendet, die in Kapitel 5.4.2 beschrieben wird. Auf Grundlage dieser Komponenten wird wiederum in Kapitel 5.4.3 das grundlegende Konzept zur Durchführung der korrelationsbasierten Merkmalsauswahl erläutert.

5.4.1 Algorithmus zur Berechnung des Gütemaßes

Algorithmus 2 beschreibt eine solche Implementierung für die Berechnung eines Gütemaßes der korrelationsbasierten Merkmalsauswahl in Form von Pseudo-Code für die Zielgröße der binären Klassifikation. Zur Berechnung des Gütemaßes der korrelationsbasierten Merkmalsauswahl der Regressionsaufgabe besteht der einzige Unterschied in Zeile 4 bezüglich der Verwendung der entsprechenden Funktion zur Berechnung der Korrelationskoeffizienten. Dementsprechend wird zur Berechnung des Gütemaßes der binären Klassifikation die punktbiseriale Korrelations verwendet und für Regressionsaufgaben die Pearson-Korrelation.

Algorithm 2 Algorithmus zur Berechnung des Gütemaßes korrelationsbasierter Merkmalsauswahl

Input: data, subset, label

1: $k \leftarrow len(subset)$
2: $rcf\_all \leftarrow []$
3: **for** feature in subset **do**
4: $\quad coeff \leftarrow pointbiserialr(data[label], data[feature])$
5: $\quad rcf_all.append(abs(coeff.correlation))$
6: **end for**
7: $rcf = mean(rcf\_all)$
8: $corr \leftarrow abs(data[subset].corr())$
9: $rff \leftarrow mean(corr)$
10: **return** $(k * rcf)/sqrt(k + k * (k - 1) * rff)$

Zur Berechnung des Gütemaßes benötigt dieser Algorithmus den jeweiligen Datensatz sowie eine Liste von Namen der aktuell betrachteten Merkmale sowie den Namen der zu untersuchenden Zielgröße als Eingaben. Auf Grundlage dieser Eingaben werden die Werte aller Variablen ermittelt, welche für die Berechnung dieses Gütemaßes verwendet werden. In Zeile 1 wird die Variable k basierend auf der Anzahl der betrachteten Merkmale berechnet. Zeile 2 initialisiert eine leere Liste *rcf_all*, welche die Korrelationen zwischen allen betrachteten Merkmalen und der jeweiligen Zielgröße enthalten soll. Dazu wird innerhalb einer Schleife in Zeile 4

für jedes Merkmal die Korrelation zu der entsprechenden Zielgröße ermittelt und in Zeile 5 der absolute Wert des berechneten Koeffizienten der Liste rcf_all hinzugefügt. Auf Grundlage dieser Liste wird in Zeile 7 durch die Funktion *mean* die durchschnittliche Korrelation *rcf* zwischen allen Merkmalen und der ausgewählten Zielgröße bestimmt. Anschließend werden in Zeile 8 die absoluten Werte von Korrelationskoeffizienten zwischen allen betrachteten Merkmalen berechnet und in Zeile 9 die durchschnittliche Korrelation *rff* zwischen diesen Merkmalen ermittelt. Letztendlich wird auf Grundlage der Formel aus Abbildung 15 das Gütemaß der korrelationsbasierten Merkmalsauswahl der betrachteten Merkmalsteilmenge basierend auf der untersuchten Zielgröße ausgegeben.

5.4.2 Implementierung einer Vorrangwarteschlange

Für die Durchführung der korrelationsbasierten Merkmalsauswahl wird die Datenstruktur einer Vorrangwarteschlange benötigt, um die beste Merkmalsteilmenge zu ermitteln. Innerhalb dieser Datenstruktur werden alle untersuchten Teilmengen inklusive ihres berechneten Gütemaßes als Priorität integriert. Diese Vorrangwarteschlange wird als eigenständige Klasse definiert, welche über eine Liste namens *queue* als einziges Attribut verfügt. Durch die Initialisierung eines Objekts dieser Klasse enthält das darin enthaltene Listen-Attribut keine Elemente.

Darüber hinaus verfügt die Klasse über drei verschiedene Methoden. Die Methode *isEmpty* überprüft, ob das Attribut *queue* Elemente enthält und gibt in diesem Fall den Wahrheitswert *False* zurück. Andernfalls wird der bool'sche Wert *True* zurückgegeben.

Durch die Methode *push* soll diese Liste um eine zusätzliche Merkmalsteilmenge als Element erweitert werden. Dabei wird überprüft, ob diese Teilmenge bereits in der Vorrangwarteschlange enthalten ist. Tritt dieser Fall ein, wird die Priorität des enthaltenen Elements mit der Priorität der hinzuzufügenden Merkmalsteilmenge verglichen. Sofern das hinzuzufügende Element über eine höheres Gütemaß verfügt, wird die Priorität des bereits enthaltenen Elements entsprechend aktualisiert. Andernfalls finden keine weiteren Operationen statt.

Die Methode *pop* gibt hingegen die Merkmalsteilmenge mit der höchsten Priorität zurück. Dazu wird innerhalb einer Schleife über alle Elemente des *queue*-Attributs iteriert und der Index des Elements mit dem höchsten Gütemaß markiert. Anschließend wird dieses Element bestehend aus der jeweiligen Merkmalsteilmenge und des entsprechenden Gütemaßes aus der Liste entfernt und durch die Methode zurückgegeben.

5.4.3 Implementierung korrelationsbasierter Merkmalsauswahl

Auf Grundlage der Beschreibung des Algorithmus zur Berechnung des Gütemaßes und der Beschreibung der Datenstruktur einer Vorrangwarteschlange kann der Algorithmus zur Durchführung der korrelationsbasierten Merkmalsauswahl implementiert werden. Zur Initialisierung dieses Prozesses wird eine Bestensuche ausgeführt, um das Merkmal mit der höchsten Korrelation zur ausgewählten Zielgröße zu ermitteln. Dazu werden die absoluten Korrelationskoeffizienten der punktbiserialen Korrelation beziehungsweise der Pearson-Korrelation für jedes einzelne Merkmal hinsichtlich der jeweiligen Zielgröße berechnet. Anschließend wird das Merkmal mit dem höchsten absoluten Korrelationskoeffizienten als erste Merkmalsteilmenge einem initialisierten Objekt der Vorrangwarteschlange mittels der push-Methode hinzugefügt.

Für die nächsten Schritte wird eine leere Liste namens *visited* initialisiert, welche alle Merkmalsteilmengen enthalten soll, welche im Rahmen der korrelationsbasierten Merkmalsauswahl bereits betrachtet wurden. Danach wird eine Schleife initiiert, deren Abbruchbedingung erfüllt wird, sobald das Listen-Attribut der Vorrangwarteschlange keine Elemente enthält. Innerhalb eines jeden Schleifendurchlaufs wird die Merkmalsteilmenge mit höchster Priorität aus dem Objekt der Vorrangwarteschlange entnommen.

Basierend auf dieser Teilmenge werden weitere Merkmalsteilmengen durch die Erweiterung um einzelne Merkmale innerhalb einer weiteren Schleife erzeugt. Die erweiterten Merkmalsteilmengen werden anschließend dahingehend überprüft, ob sie bereits untersucht wurden und dementsprechend in der visited-Liste enthalten sind. Falls diese Teilmengen noch nicht untersucht wurden, wird diese Teilmenge als Element der visited-Liste hinzugefügt. Darüber hinaus werden für diese Teilmengen die jeweiligen Gütemaße durch Anwendung des in Kapitel 5.4.1 vorgestellten Algorithmus berechnet. Anschließend wird dieses Element dem Objekt der Vorrangwarteschlange durch Ausführung der push-Funktion hinzugefügt.

Innerhalb eines jeden Schleifendurchlaufs wird die Priorität der aktuell entnommenen Merkmalsteilmenge mit der zuvor höchsten Priorität verglichen. Dabei wird stets das Element mit der höchsten Priorität markiert und nach Beendigung des Schleifendurchlaufs zurückgegeben. Auf Grundlage der resultierenden Merkmalsteilmenge können anschließend unter Verwendung von Operationen der pandas-Bibliothek die reduzierten Datensätze erzeugt werden. Diese Datensätze müssen ebenfalls durch die Methoden der Vorverarbeitung in Trainings-, Test- und Validierungsdaten aufgeteilt werden. Eine Trennung von Merkmalen und Zielgrößen sowie die Skalierung von Merkmalswerten findet ebenfalls statt, sodass diese Datensätze

als CSV-Dateien für die Anwendung in künstlichen neuronalen Netzen bereitgestellt werden.

5.5 Implementierung künstlicher neuronaler Netze

Zur Implementierung künstlicher neuronaler Netze mittels der Programmiersprache Python wird die Bibliothek *TensorFlow* verwendet. Diese Bibliothek stellt eine umfangreiche Sammlung von Werkzeugen, APIs und Ressourcen zur Durchführung komplexer mathematischer Berechnungen und zur Erstellung von Modellen des maschinellen Lernens bereit. Auf Grundlage der Definition von Schichten, Aktivierungsfunktionen, Optimierern und anderen Komponenten können komplexe Modelle neuronaler Netze und anderer maschineller Lernalgorithmen entworfen, trainiert und eingesetzt werden. Darüber hinaus stellt TensorFlow eine flexible und effiziente Plattform zur Verarbeitung großer Datenmengen.

Für die Erleichterung der Entwicklung von Modellen des maschinellen Lernens existiert wiederum die Python-Bibliothek *Keras*. Diese Bibliothek stellt eine auf TensorFlow basierende High-Level-API dar, die über eine intuitive Schnittstelle zur Definition von Modellen neuronaler Netzwerke und deren Trainingsprozesse verfügt. Zur Erstellung komplexer Modelle können verschiedene Schichten basierend auf einer Vielzahl bereitgestellter Aktivierungsfunktionen und weiteren Komponenten entworfen werden. Trainingsprozesse können wiederum mithilfe von Funktionen durch die Auswahl von Optimierern, Verlustfunktionen und Metriken konfiguriert werden. Zur Bewertung der Leistung solcher Modelle können diese Metriken auf Grundlage von Validierungsdaten berechnet werden, wodurch aufgrund der Speicherung von Modellen während des Trainings Fortschritte jederzeit überwacht werden können. Nach Abschluss des Trainings können ebenfalls durch Funktionen Vorhersagen für neue Daten generiert und deren Ergebnisse interpretiert werden.

In Kapitel 5.5.1 wird die Implementierung zur Erstellung der Modelle künstlicher neuronaler Netze der verschiedenen Architekturen beschrieben. Kapitel 5.5.2 behandelt die Konfiguration des Trainingsprozesses und Kapitel 5.5.3 thematisiert die Bereitstellung interpretierbarer Ergebnisse.

5.5.1 Erzeugung der Architekturen künstlicher neuronaler Netze

Die konkrete Implementierung der Architekturen künstlicher neuronaler Netze basiert auf der Verwendung der Keras-Klassen *Sequential* und *layers*. Mithilfe der Sequential-Klasse können sequentielle Modelle des maschinellen Lernens auf Grundlage der linearen Stapelung unterschiedlicher Arten von Schichten realisiert werden.

Diese Schichten können unter Verwendung vorgefertigter Schichttypen der layers-Klasse konfiguriert und nacheinander dem sequentiellen Modell hinzugefügt werden. Die Anzahl der Neuronen sowie die Auswahl einer Aktivierungsfunktion findet dabei für jede einzelne Schicht separat statt.

Für die Implementierung der in Kapitel 4.7 beschriebenen Architekturen von Modellen künstlicher neuronaler Netze werden die vorgefertigen Schichttypen *InputLayer*, *Dense*, *LayerNormalization* und *Dropout* benötigt. Der Schichttyp InputLayer wird zur Definition des Eingabevektors von Modellen definiert und repräsentiert in der Regel die Eingabeschicht eines künstlichen neuronalen Netzes. Innerhalb einer solchen Schicht finden keine Berechnungen statt, wodurch lediglich Eingabedaten bereitgestellt werden. Da die Anzahl der Eingabevariablen von der Auswahl des jeweiligen Datensatzes abhängt, werden die Eingabeschichten aller Modelle wie folgt definiert:

```
tf.keras.layers.InputLayer(input_shape=(input,))
```

In diesem Zusammenhang wird die Dimensionalität der Eingabedaten durch den Parameter *input* bestimmt.

Unter Verwendung des Dense-Schichttyps werden alle verborgenen Schichten sowie die Ausgabeschichten aller Modelle definiert. Die Konfiguration dieser Schichten basiert auf der Festlegung einer Anzahl an Neuronen und der Auswahl ihrer zugrunde liegenden Aktivierungsfunktion. Eine besondere Eigenschaft von Schichten dieser Typen ist die vollständige Verbundenheit zwischen den Neuronen dieser Schicht und der Neuronen der vorherigen Schicht, wodurch die Realisierung von Feedforward-Netzwerken erleichtert wird. Die Festlegung der Anzahl von Neuronen der verborgenen Schichten basiert auf der Beschreibung der Architekturen aus Kapitel 4.7. Für jede dieser Schichten wird außerdem die ReLU-Aktivierungsfunktion integriert. Zur Konfiguration aller Ausgabeschichten beträgt die Anzahl der Neuronen hingegen eins, während die Auswahl der Aktivierungsfunktion durch den jeweiligen Anwendungsfall bedingt ist.

Dementsprechend können die Modelle der binären Klassifikation unter Verwendung der S-Architektur wie folgt definiert werden:

```
model = tf.keras.Sequential([
    tf.keras.layers.InputLayer(input_shape=(input,)),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Die Modelle der Regressionsaufgaben unter Verwendung der S-Architektur werden hingegen wie folgt implementiert:

```
model = tf.keras.Sequential([
    tf.keras.layers.InputLayer(input_shape=(input,)),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(1, activation='linear')
])
```

Zur Realisierung von Modellen der Architekturen M und L werden die Schichttypen Dropout und LayerNormalization zusätzlich verwendet. Mittels des Schichttyps Dropout können die Funktionalitäten der Regularisierungsstrategie des Dropouts realisiert werden. Durch die Verwendung dieses Schichttyps werden während des Trainings einige Neuronen der vorherigen Dense-Schicht deaktiviert. Mithilfe des Schichttyps der LayerNormalization werden wiederum die Ausgaben der vorherigen Dense-Schicht normalisiert.

Auf Grundlage dieser Komponenten werden Modelle der M-Architektur beispielsweise wie folgt implementiert:

```
model = tf.keras.Sequential([
    tf.keras.layers.InputLayer(input_shape=(input,)),
    tf.keras.layers.Dense(200, activation='relu'),
    tf.keras.layers.LayerNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(200, activation='relu'),
    tf.keras.layers.LayerNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(150, activation='relu'),
    tf.keras.layers.LayerNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(120, activation='relu'),
    tf.keras.layers.LayerNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(70, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Die Modelle der L-Architektur können beispielsweise wie folgt realisiert werden:

```
model = tf.keras.Sequential([
    tf.keras.layers.InputLayer(input_shape=(input,)),
    tf.keras.layers.Dense(500, activation='relu'),
    tf.keras.layers.LayerNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(450, activation='relu'),
    tf.keras.layers.LayerNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(400, activation='relu'),
    tf.keras.layers.LayerNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(350, activation='relu'),
    tf.keras.layers.LayerNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(300, activation='relu'),
    tf.keras.layers.LayerNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(250, activation='relu'),
    tf.keras.layers.LayerNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(200, activation='relu'),
    tf.keras.layers.LayerNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(150, activation='relu'),
    tf.keras.layers.LayerNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(120, activation='relu'),
    tf.keras.layers.LayerNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(70, activation='relu'),
    tf.keras.layers.Dense(1, activation='linear')
])
```

Die auf Grundlage dieser Implementierungen erstellten Modelle werden letztendlich für die Integration in Trainingsprozesse unter Verwendung der Funktion *save* als persistente Dateien bereitgestellt.

5.5.2 Implementierung des Trainingsprozesses

Für die Implementierung des Trainingsprozesses wird der jeweilige Datensatz als zweidimensionale pandas-Datenstruktur aus den bereitgestellten CSV-Dateien initialisiert. Die Initialisierung des entsprechenden Modells eines künstlichen neuronalen Netzes aus seiner jeweiligen Datei wird ebenfalls benötigt, um den auszuführenden Trainingsprozess zu definieren.

Die Konfiguration der Trainingsprozesse basiert auf der Verwendung der Klassen *optimizers*, *losses* und *metrics* der Keras-Bibliothek. Mittels der optimizers-Klasse werden verschiedene Optimierungsalgorithmen wie besipielsweise der stochastische Gradientenabstieg oder Adam bereitgestellt. Auf Grundlage der losses-Klasse können Verlustfunktionen wie beispielsweise der mittlere quadratische Fehlder oder die binäre Kreuzentropie für den Trainingsprozess der Modelle künstlicher neuronaler Netze definiert werden. Über die Klasse metrics steht eine Vielzahl von Performanzmetriken wie zum Beispiel die Genauigkeit oder der Recall zur Verfügung, die während des Trainings und zur Auswertung der Modelle berechnet werden können.

Für das Training binärer Klassifikationen wird der Optimierungsalgorithmus des stochastischen Gradientenabstiegs, die Verlustfunktion der binären Kreuzentropie sowie die Performanzmetriken der binären Genauigkeit, des Recalls und AUC verwendet. Auf Grundlage dieser Konfiguration werden diese Trainingsprozesse wie folgt implementiert:

```
model.compile(
    optimizer=tf.keras.optimizers.SGD(),
    loss=tf.keras.losses.BinaryCrossentropy(),
    metrics=[tf.keras.metrics.BinaryAccuracy(),
             tf.keras.metrics.AUC(),
             tf.keras.metrics.Recall()]
)
```

Die Ausführung des Trainings binärer Klassifikationen wird wiederum durch die *fit*-Funktion der Sequential-Klasse realisiert. Dieser Funktion werden die Merkmale und Zielgrößen der Trainings- und Validierungsdaten sowie die gewünschte Anzahl verwendeter Epochen übergeben. Auf Grundlage der Bezeichnung der Merkmale und Zielgrößen von Trainingsdaten als *x_train* und *y_train* sowie der Bezeichnung der Merkmale und Zielgrößen von Validierungsdaten als *x_val* und *y_val* wird die Ausführung des Trainingsprozesses wie folgt implementiert:

```
initial_history = model.fit(
```

```
    x_train,
    y_train,
    epochs=20,
    validation_data = (x_val, y_val)
)
```

Für die Modelle der Regressionsaufgaben wird der Optimierungsalgorithmus Adam, die Verlustfunktion des mittleren quadratischen Fehlers sowie die Performanzmetrik der Genauigkeit benötigt. Die Konfiguration dieses Trainingsprozesses basiert auf der folgenden Implementierung:

```
model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss=tf.keras.losses.MeanSquaredError(),
    metrics=['accuracy']
)
```

Im Rahmen dieses Projekts soll eine optimale Leistung der Regressionsmodelle mittels k-facher Kreuzvalidierung sichergestellt werden. Mithilfe der Klasse *KerasRegressor* werden die durch Keras definierten Modelle als Schätzer in Funktionen der sklearn-Bibliothek verwendet. Dazu wird ein Objekt der *KFold*-Klasse initialisiert, welcher den zugrunde liegenden Trainingsdatensatz in zehn zufällige Teilmengen aufteilt. Für jede dieser Teilmengen werden alle anderen Teilmengen unter Verwendung der Funktion *cross_val_score* als Testdatensätze eingesetzt. Die Implementierung dieses Schrittes wird wie folgt realisiert:

```
estimator = KerasRegressor(build_fn=model, epochs=100,
                           batch_size=10, verbose=0)
kfold = KFold(n_splits=10)
results = cross_val_score(estimator, x_train, y_train, cv=kfold)
```

Auf Grundlage der Verwendung von 100 Epochen mit einer Batch-Größe von 10 werden die Ausführungen von Trainingsprozesen dieser Modelle wie folgt implementiert:

```
history = estimator.fit(x_train, y_train,
    validation_data=(x_val, y_val),
    epochs=100, batch_size=10,
    verbose=1
).history_
```

5.5.3 Bereitstellung von Ergebnissen

Zur Bereitstellung von Ergebnissen des Trainingsprozesses und der Auswertung des Testdatensatzes werden die Keras-Funktionen *predict* und *evaluate* verwendet. Mithilfe der predict-Funktion können auf Grundlage trainierter Modelle Vorhersagen für ausgewählte Daten getroffen werden. Durch die evaluate-Funktion werden wiederum die zuvor definierten Performanzmetriken auf Grundlage der Merkmale und Zielgrößen dieser Daten berechnet.

Für die trainierten Modelle der binären Klassifikation können die Vorhersagen und Performanzmetriken für Trainings-, Test- und Validierungsdaten wie folgt implementiert werden:

```
y_train_pred = model.predict(x_train)
y_val_pred = model.predict(x_val)
y_test_pred = model.predict(x_test)

train_err, train_acc, train_auc, train_recall =
    model.evaluate(x_train, y_train, verbose=0)
val_err, val_acc, val_auc, val_recall =
    model.evaluate(x_val, y_val, verbose=0)
test_err, test_acc, test_auc, test_recall =
    model.evaluate(x_test, y_test, verbose=0)
```

Die Berechnungen der Performanzmetriken des F1-Scores und der Präzision können mithilfe der Funktionen *precision_score* und *f1_score* unter Verwendung des Schwellenwerts 0.5 wie folgt implementiert werden:

```
y_train_pred = (y_train_pred>0.5)
y_val_pred = (y_val_pred>0.5)
y_test_pred = (y_test_pred>0.5)

train_prec = precision_score(y_train, y_train_pred)
train_f1 = f1_score(y_train, y_train_pred)
val_prec = precision_score(y_val, y_val_pred)
val_f1 = f1_score(y_val, y_val_pred)
test_prec = precision_score(y_test, y_test_pred)
test_f1 = f1_score(y_test, y_test_pred)
```

Die Visualisierung der Metriken basiert wiederum auf der Verwendung der Python-Bibliothek *matplotlib*, auf deren Grundlage die berechneten Metriken der Regressi-

onsmodelle ebenfalls in Diagrammen dargestellt werden. Mithilfe dieser Bibliothek werden diese Diagramme letztendlich als skalare Vektorgrafiken gespeichert.

6 Auswertung der Ergebnisse

Die in Kapitel 4.2 vorgestellten Prozesse wurden zur Durchführung von Experimenten basierend auf den in Kapitel 5 beschriebenen Implementierungen realisiert. Alle Experimente wurden auf Grundlage eines Laptops unter Verwendung des Linux-Betriebssystems Ubuntu 20.04.4 LTS durchgeführt. Dieser Laptop enthält einen Intel Core i7-8665U-Prozessor der achten Generation, der über 8 Kerne mit einer Leistung von jeweils 1.90 GHz verfügt. Darüber hinaus ist dieser Computer mit 16 GB RAM und 500 GB Festplattenspeicher ausgestattet.

Zur Beantwortung der in Kapitel 4.1 definierten Forschungsfragen sollen die Ergebnisse der Experimente ausgewertet werden. Diese Auswertung basiert auf der Aktualisierung des CVEfixes-Datensatzes und der Extraktion von Metriken sowie den Ergebnissen der Korrelationsanalysen und der Vorhersagen trainierter Modelle künstlicher neuronaler Netze.

Kapitel 6.1 beschreibt den aktualisierten CVEfixes-Datensatz. In Kapitel 6.2 werden die Ergebnisse der Metriken-Extraktion durch die Werkzeuge Understand und Analizo präsentiert und in Kapitel 6.3 die Resultate der Korrelationsanalysen analysiert. Auf Grundlage der ausgewerteten Korrelationsanaylse werden die Ergebnisse der korrelationsbasierten Merkmalsauswahl in Kapitel 6.4 untersucht. Kapitel 6.5 beantwortet die Forschungsfrage RQ1 bezüglich des Zusammenhangs zwischen Software-Produktmetriken und dem Auftreten von Schwachstellen durch die Auswertung der Ergebnisse binärer Klassifikationen. Die Forschungsfrage RQ2 bezüglich der Bewertung der Schwere von Schwachstellen wird wiederum durch die Auswertung der Regressionsaufgaben in Kapitel 6.6 behandelt. Durch Kapitel 6.7 soll die Forschungsfrage RQ3 hinsichtlich der Eignung spezifischer Software-Produktmetriken zur Bewertung der Kriterien und Aspekte der Software-Qualität ausgewertet werden.

6.1 Aktualisierung des Datensatzes

Die Aktualisierung des CVEfixes-Datensatzes wurde am 27.03.2023 durchgeführt und benötigte mehrere Tage. Auf dieser Grundlage verfügt der aktualisierte Datensatz über 9.229 CVE-Einträge, die zu 239 CWE-Kategorien zugeordnet werden können. Durch diese Einträge werden 9.365 verschiedene Commits aus 3.249 unterschiedlichen Open-Source-Projekte referenziert. Diese Commits bestehen aus den Versionen von 42.282 verschiedenen Dateien und 187.203 Methoden, welche durch 32 verschiedene Programmiersprachen implementiert wurden.

Mithilfe des automatisierten Prozesses zur Aktualisierung der Datenbank konn-

ten dementsprechend 3.864 neue CVE-Einträge und 59 weiteren CWE-Kategorien bereitgestellt werden. Dieser Zuwachs basiert auf der Verarbeitung von 3.871 weiteren Commits, welche die Versionen von 24.033 neuen Dateien und 136.881 neue Funktonen enthalten und aus 1.495 zusätzlichen Open-Source-Projekten extrahiert wurden. Basierend auf diesen Werten konnte eine Steigerung von 72% hinsichtlich der Anzahl verfügbarer CVE-Einträge und eine Steigerung von fast 33% referenzierter CWE-Kategorien erzielt werden. Im Rahmen dieser Aktualisierung stieg die Anzahl analysierter Open-Source-Projekte um zirka 85% und die Anzahl bereitgestellter Commits um knapp über 70%. Die Anzahl verfügbarer Quellcode-Dateien konnte wiederum um etwas mehr als 131% auf mehr als doppelt so viele Einträge erhöht werden. Die Anzahl des extrahierten Quellcodes auf Methoden-Ebene wurde durch eine Steigerung um 270% sogar fast verdreifacht.

Auf Grundlage dieser Erhöhungen verfügt eine Durchführung der Experimente auf Grundlage der Extraktion von Software-Produktmetriken auf Methoden-Ebene eine deutlich größere Datenbasis. Daher wurde die Qualität dieser Datenbasis hinsichtlich der korrekten und vollständigen Bereitstellung des Quellcodes dieser Methoden überprüft. Im Rahmen dieser Überprüfung wurden über 60% der im aktualisierten CVEfixes-Datensatz enthaltenen Methoden als unvollständig deklariert. Dementsprechend eignet sich der Quellcode von 114.062 Methoden nicht zur Extraktion von Software-Produktmetriken. Für die restlichen 73.141 vollständigen Methoden besteht jedoch das Risiko mangelnder Qualität hinsichtlich weiterer Aspekte. Die Datenqualität des bereitgestellten Quellcodes auf Datei-Ebene verfügt hingegen nicht über solche Qualitätsprobleme, wodurch die Extraktion von Software-Produktmetriken den geeigneteren Ansatz repräsentiert. Darüber hinaus existieren für die 42.282 bereitgestellten Dateien bis zu zwei verschiedene Versionen des jeweiligen Quellcodes, wodurch die Größe der zugrunde liegenden Datenbasis ähnlich hoch ist.

6.2 Metriken-Extraktion

Unter Verwendung des aktualisierten CVEfixes-Datensatzes konnten aus dem Quellcode von 52.279 verschiedenen Datei-Versionen Software-Produktmetriken durch das Werkzeug Understand extrahiert werden. Insgesamt 24.724 dieser Beispiele enthalten eine Schwachstelle, während die resltichen 27.555 Beispiele Datei-Versionen mit behobenen Schwachstellen repräsentieren. Dabei existieren für 23.312 verschiedene Dateien Beispiele mit und ohne Schwachstellen und für 1.412 Dateien ausschließlich Beispiele mit Schwachstellen sowie für 4.243 Dateien ausschließlich Beispiele behobener Schwachstellen. Von den durch das Understand-Tool extrahierbaren Metriken

konnten für die folgenden 37 Software-Produktmetriken aufgrund der Betrachtung einzelner Dateien keine Werte ermittelt werden:

NOPRA, NIIV, CDCMF, NMI, PLCM, NF, CDCIVPT, ASMEVG, SMEVG,
SSMEVG, FANOUT, PLC, CBOM, NOPA, DIT, CDCIVPTI, CDCMI, SP,
KNOTS, CDCMC, FANUP, CDCMPT, NOSPRM, PACK, FANIN, MAXEK, NCF, AIP, MMEVG, NHF, PROP, CBO, NM, MINEK, NOSPM, NSC, ALCOMMI

Stattdessen konnten für die folgenden 74 Software-Produktmetriken Werte ermittelt werden, welche sowohl in den Understand-Rohdatensatz als auch in den kombinierten Roh-Datensatz integriert wurden:

EU, AVG, MMVG, DST_JS, CS, EST, BLOC_JS, BLOC, MEVG, LCOMMI,
NST, DST, MAXVG, DST_PHP, AEVG, RCC, NCLASS, LCOMM_JS, NPATHL, MSMVG, PU, NPATH, LOC, LCOMM_HTML, ALCOMM, CLP, SUMMVG, LOC_PHP, CLCD, SUMVG, NOPRM, ASLOCI, XST, SMVG, SVG,
XST_PHP, LOC_JS, SLOC_PHP, CDCMPTI, BLOC_HTML, VG, SLOC_JS,
BLOCI, SEVG, LCOMM_PHP, LCOMM, ASVG, MSVG, NIV, WMC, ABLOCI,
NOPM, SSMVG, ASLOC, SSVG, MVG, ASMVG, LOC_HTML, LOC, MN, NOM, ABLOC, AMVG, BLOC_PHP, XST_JS, NOF, ALCOMM, EVG, CDCMD,
NIM, ALOC, CLI, NOA, ELOC

Zur Untersuchung der Unterschiede zwischen der durch das Werkzeug Understand extrahierten Werte von Software-Produktmetriken hinsichtlich der verschiedenen Versionen gleicher Dateien werden alle relevanten Informationen in Tabelle 11.4.1 bereitgestellt. Innerhalb dieser Tabelle werden für diese Wert-Paare von Software-Produktmetriken die Anzahl fehlender Metriken-Werte sowie die Anzahl identischer Wert-Paare dargestellt. Darüber hinaus enthält die Tabelle die mittlere Abweichung einzelner Wert-Paare jeder Software-Produktmetrik.

Basierend auf diesen Ergebnissen müssen im Rahmen der Vorverarbeitung für mindestens die Hälfte aller Einträge fehlende Werte für die folgenden 20 Software-Metriken ersetzt werden:

CS, EST, LCOMMI, PU, ALOCI, CLP, CLCD, NOPRM, ASLOCI, CDCMPTI, BLOCI, NIV, WMC, ABLOCI, NOPM, NOM, CDCMD, NIM, CLI, NOA, ELOC

Für die folgenden 22 weiteren Software-Produktmetriken Metriken müssen die fehlende Werte von mehr als einem Drittel der Einträge ebenfalls ersetzt werden:

EU, BLOC_JS, DST_PHP, LCOMM_JS, NPATHL, NPATH, LCOMM_HTML, LOC_PHP, SMVG, SVG, XST_PHP, LOC_JS, SLOC_PHP, BLOC_HTML, VG, SLOC_JS, LCOMM_PHP, MVG, LOC_HTML, BLOC_PHP, XST_JS, EVG

Auf Grundlage dieser Erkenntnisse wird die Qualität des Understand-Datensatzes sowie des kombinierten Datensatzes stark beeinträchtigt. Eine Überprüfung aller vorhandenen Wert-Paare hinsichtlich ihrer Unterschiede verdeutlicht dieses Qualitätsproblem zusätzlich. Dementsprechend verfügen nur die fünf Software-Metriken BLOC, NST, RCC, XST und SSMVG über mehr als 30% von unterschiedlichen Wert-Paaren, anhand deren eine geeignete Klassifikation möglich ist. Die Metrik *RCC* enthält in diesem Zusammenhang mit einem Verhältnis von über 57% die meisten verschiedenen Wert-Paare. Lediglich die folgenden 22 Software-Produktmetriken beinhalten über mindestens 10% an Einträgen unterschiedlicher Wert-Paare:

EU, MMVG, CS, DST, MAXVG, MSMVG, LOC, SUMMVG, LOC_PHP, CLCD, SUMVG, ASLOCI, XST_PHP, SLOC_PHP, BLOCI, SEVG, LCOMM, MSVG, LOC, NOF, ALOC, ELOC

Mithilfe des aktualisierten CVEfixes-Datensatzes konnten durch das Werkzeug Analizo Software-Produktmetriken für lediglich 16.298 verschiedene Beispiele aufgrund der exklusiven Unterstützung der Programmiersprachen C, C++ und Java extrahiert werden. Insgesamt 8.065 dieser Beispiele enthalten eine Schwachstelle, während die resltichen 8.233 Beispiele Datei-Versionen mit behobenen Schwachstellen repräsentieren. Dabei existieren für 7.994 verschiedene Dateien Beispiele mit und ohne Schwachstellen und für 71 Dateien ausschließlich Beispiele mit Schwachstellen sowie für 239 Dateien ausschließlich Beispiele behobener Schwachstellen. Von den durch das Analizo-Tool extrahierbaren Metriken auf Modul-Ebene konnten nur für die beiden Software-Produktmetriken CBO und NSC keine Werte ermittelt werden. Stattdessen konnten für die folgenden 12 Software-Produktmetriken Werte ermittelt werden, welche sowohl in den Analizo-Rohdatensatz als auch in den kombinierten Roh-Datensatz integriert wurden:

AVG, AMLOC, ANPAR, DIT, LCOM1, LOC, MLOC, NOA, NOM, NOPA, NOPM, RFC

Die durch das Werkzeug Analizo bereitgestellte Datenbasis ist im Vergleich zur Datenbasis des Understand-Tools aufgrund der Anzahl verarbeiteter Beispiele und der Anzahl extrahierter Software-Produktmetriken deutlich geringer. Aufgrund dieses Ungleichgewichts können Ergebnisse trainierter künstlicher neuronaler Netze unter Verwendung des kombinierten Roh-Datensatzes stärker durch die Datenbasis des Understand-Tools beeinflusst werden.

Zur Untersuchung der Unterschiede zwischen den durch das Werkzeug Analizo extrahierten Werten von Software-Produktmetriken hinsichtlich der verschiedenen Versionen gleicher Dateien werden alle relevanten Informationen in Anhang 11.4.2 bereitgestellt. Im Rahmen der Verwendung dieses Metriken-Extraktionstools werden keine Einträge mit fehlenden Werten erzeugt. Daher repräsentiert diese Tabelle nur die Anzahl identischer Wert-Paare und die mittlere Abweichung unterschiedlicher Wert-Paare für alle extrahierten Software-Produktmetriken.

Basierend auf diesen Ergebnissen verfügen ausschließlich die Metriken *LOC* und *AMLOC* in mehr als 50% der Einträge über unterschiedliche Wert-Paare. Eine Klassifikation des Auftretens von Schwachstellen auf Grundlage dieser Metriken kann jedoch kritische Ergebnisse erzeugen, da diese Metriken lediglich die Anzahl verwendeter Code-Zeilen quantifizieren. Für die Software-Produktmetriken ANPAR, DIT, LCOM1, NOA, NOM, NOPA und NOPM sind jedoch die extrahierten Metriken-Werte für die verschiedenen Versionen der gleichen Datei in über 90% der Fälle identisch.

Diese Charakteristiken der extrahierten Werte für Software-Produktmetriken durch die beiden Werkzeuge Understand und Analizo lassen sich allerdings auf granulare Unterschiede zwischen den unterschiedlichen Versionen der gleichen Dateien zurückführen. In vielen Fällen werden lediglich kleine Änderungen durchgeführt, welche nur innerhalb einzelner Zeilen enthalten sind und weder die Komplexität oder weitere grundlegende Eigenschaften des Quellcodes beeinflussen. Der Großteil der extrahierten Metriken können solche Änderungen nicht erfassen, weshalb sich die Werte der extrahierten Metriken im Vergleich zwischen verschiedenen Datei-Versionen in den meisten Fällen nicht verändern.

6.3 Auswertung der Korrelationsanalyse

Im Rahmen der Korrelationsanalyse wurden die Beziehungen zwischen den einzelnen Merkmalen des Understand- und Analizo-Datensatzes und ihren Zielgrößen untersucht. Anstatt der Visualisierung in Form von Grafiken werden aufgrund der hohen Anzahl an Merkmalen die Ergebnisse der Korrelationsanalyse in Form von Tabellen repräsentiert.

Anhang 11.5.1 enthält eine Tabelle der berechneten Korrelationskoeffizienten zwischen allen Merkmalen und Zielgrößen des Understand-Datensatzes der Korrelationsmaße Pearson, Spearman und Kendall. Der erste Wert jeder Spalte bildet den berechneten Korrelationskoeffizienten für die Zielgröße der binären Klassifikation ab. Der zweite Wert jeder Spalte repräsentiert wiederum den Koeffizienten für die Zielgröße der Regressionsaufgabe.

Auf Grundlage der berechneten Koeffizienten besteht ausschließlich für das Merkmal der Programmiersprache *unknown* eine schwache Korrelation zu beiden Zielgrößen. Für die Zielgröße der Regressionsaufgabe bestehen zusätzlich schwache Korrelationen zu den Merkmalen der Programmiersprache *PU* und der Metrik *PU*. Alle weiteren Merkmale verfügen laut Definition über keine signifikante Korrelation zu einer der beiden Zielgrößen. Dabei bestehen unter Anwendung der verschiedenen Korrelationsmaße nur geringe Unterschiede hinsichtlich der berechneten Koeffizienten. Daher konnten keine konkreten Zusammenhänge zwischen diesen Merkmalen und den beiden Zielgrößen identifiziert werden. Die hohe Anzahl fehlender Werte und die geringen Unterschiede zwischen den Werten extrahierter Metriken verschiedener Datei-Versionen können als Ursache dieser Ergebnisse angenommen werden.

Anhang 11.5.2 repräsentiert eine Tabelle der berechneten Korrelationskoeffizienten zwischen den Merkmalen und Zielgrößen des Analizo-Datensatzes aller Korrelationsmaße in der gleichen Struktur wie die Tabelle aus Anhang 11.5.1. Laut Definition verfügt im Rahmen der Korrelationsanalyse keines dieser Merkmale über eine Korrelation zu einer der beiden Zielgrößen, wodurch keine konkreten Zusammenhänge ermittelt werden konnten. In diesem Kontext verfügt das Merkmal der Programmiersprache *unknown* ebenfalls über die höchste Korrelation bezüglich beider Zielgrößen. Da der Analizo-Datensatz über keine fehlenden Werte verfügt, können die geringen Unterschiede zwischen den Werten extrahierter Metriken verschiedener Datei-Versionen ausschlaggebend für diese Ergebnisse sein.

6.4 Ergebnisse korrelationsbasierter Merkmalsauswahl

Mithilfe der korrelationsbasierten Merkmalsauswahl soll die Dimensionalität der verschiedenen Datensätze reduziert werden. Dazu werden in Tabelle 1 die im Rahmen dieses Prozesses selektierten Merkmale für binäre Klassifikationen sowie für Regressionsaufgaben aufgelistet.

In jedem Anwendungsfall der korrelationsbasierten Merkmalsauswahl wird die Programmiersprache *unknown* aufgrund ihrer hohen Korrelation zu beiden Zielgrößen selektiert. Dieses Merkmal verfügt in allen Datensätzen über die höchste Korrelation zu den Zielgrößen und wird durch Anwendung der Bestensuche als er-

ste Merkmalsteilmenge verwendet, auf der alle weiteren untersuchten Teilmengen erzeugt werden. Die Teilmenge mit dem höchsten Gütemaß besteht in den meisten Fällen ausschließlich aus diesem einzelnen Merkmal. Aufgrund hoher Korrelationen zwischen den einzelnen Merkmalen und niedrigen Korrelationen zwischen Merkmalen und Zielgrößen verfügen die meisten erweiterten Teilmengen über ein niedrigeres Gütemaß. Lediglich die Programmiersprache *PHP* kann für die Zielgröße der Regressionsaufgaben des Understand-Datensatzes sowie des kombinierten Datensatzes das Gütemaß dieser Teilmenge erhöhen. Dieses Verhalten beruht auf der schwachen Korrelation zwischen diesem Merkmal und der Zielgröße der Regressionsaufgaben.

Datensatz	**Binäre Klassifikation**	**Regressionsaufgabe**
Understand	Programmiersprache unknown	Programmiersprache unknown, Programmiersprache PHP
Analizo	Programmiersprache unknown	Programmiersprache unknown
Kombiniert	Programmiersprache unknown	Programmiersprache unknown, Programmiersprache PHP

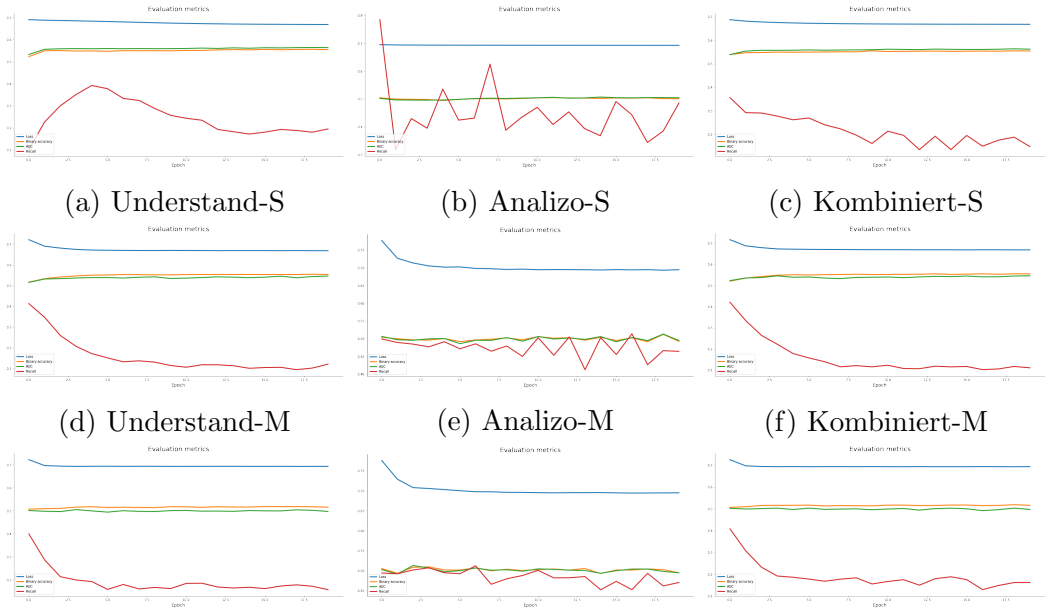
Tabelle 1: Ergebnisse korrelationsbasierter Merkmalsauswahl

Die Qualität dieser reduzierten Datensätze sollte jedoch kritisch betrachtet werden. Durch die Verwendung dieser Datensätze in künstlichen neuronalen Netzen kann das Auftreten sowie die Bewertung von Schwachstellen lediglich anhand dieser Merkmale vollzogen werden. Dementsprechend sollen diese Entscheidungen anhand der Verwendung unbekannter Programmiersprachen beziehungsweise anahnd der Verwendung der Programmiersprache *PHP* getroffen werden. Allerdings sollte die reine Information bezüglich der Verwendung einer Programmiersprache nicht ausschlaggebend bezüglich des Auftretens von Schwachstellen oder ihrer Schwere sein, da der Quellcode jeder Programmiersprache Schwachstellen potenzielle Schwachstellen unterschiedlicher Schwere enthalten kann. Der Quellcode einer Programmiersprache muss jedoch nicht zwingend eine Schwachstelle enthalten, wodurch die Informationsgrundlage dieser Experimente stark limitiert ist. Darüber hinaus können die reduzierten Datensätze auf Grundlage dieser binären Merkmale über viele widersprüchliche Beispiele verfügen.

6.5 Binäre Klassifikation neuronaler Netze zur Erkennung von Schwachstellen

Im Rahmen dieses Kapitels werden die Ergebnisse binärer Klassifikationen zur Beantwortung der Forschungsfrage RQ1 ausgewertet.

Abbildung 17 visualisiert Ergebnisse der Trainingsprozesse aller vollständigen



(a) Understand-S (b) Analizo-S (c) Kombiniert-S

(d) Understand-M (e) Analizo-M (f) Kombiniert-M

(g) Understand-L (h) Analizo-L (i) Kombiniert-L

Abbildung 17: Visualisierung der Performanz-Metriken vollständiger Trainingsdatensätze im Verlauf des Lernprozesses

Datensätze unter Verwendung der verschiedenen Modell-Architekturen. Alle Diagramme beschreiben den Verlauf des berechneten Fehlers (blau) sowie der Performanzmetriken der binären Genauigkeit (gelb), des Recalls (rot) sowie der AUC-Metrik (grün) nach Abschluss jeder Epoche. Der berechnete Fehler der Verlustfunktion ist in jedem Experiment ähnlich hoch und konnte nur in den ersten Epochen leicht minimiert werden. Die AUC-Metrik sowie die binäre Genauigkeit verfügen in der Regel über fast konstante Werte, welche für jedes Experiment ähnlich sind und im Verlauf des Trainings nicht sginifikant optimiert werden konnten.

Für die Experimente des Analizo-Datensatzes variieren die Recall-Werte im Verlauf des Trainingsprozesses auf einem deutlich höheren Skalenniveau als in den Experimenten der anderen Datensätze. Dieser Wert kann im Verlauf des Trainingsprozesses in den Experimenten der anderen Datensätze mit Ausnahme des Experiments des Understand-Datensatzes unter Verwendung der S-Architektur stetig reduziert werden. In den Experimenten des kombinierten Datensatzes sind jedoch leichte Variationen der Recall-Werte bei stetiger Reduzierung zu erkennen. Das Verhältnis bereitgestellter Understand- und Analizo-Metriken ist innerhalb dieses Datensatzes sehr unausgeglichen. Daher verfügt dieser Datensatz im Vergleich über deutlich mehr Beispiele des Understand-Datensatzes, worauf sich die stetige Reduzierung des Recalls zurückführen lässt. Die Variationen der Werte dieser Performanzmetrik im Verlauf des Trainingsprozesses können wiederum auf dem Einfluss der integrierten

Analizo-Daten beruhen.

Auf Grundlage dieser Diagramme werden die Ergebnisse der Performanzmetriken unter Verwendung der unterschiedlichen Architekturen in Kombination mit den einzelnen Datensätze ausgewertet. Tabelle 2 stellt für die Experimente des vollständigen Understand-Datensatzes die resultierenden Ergebnisse des berechneten Fehlers und der Performanzmetriken der binären Genauigkeit, des Recalls sowie den zusätzlich berechneten Metriken des F1-Scores und der Präzision bereit. Darüber hinaus enthält diese Tabelle für jeden Testdatensatz ebenfalls den berechneten Wert der Performanzmetrik AUC.

Datensatz		Genauigkeit	Verlust	Recall	Präzision	F1-Score	AUC
	Training	0.555713	0.66827	0.175076	0.61112	0.272177	
S	Test	0.532448	0.683911	0.276294	0.50616	0.357462	0.549744
	Validierung	0.509422	0.693163	0.133773	0.517447	0.212587	
	Training	0.556177	0.668079	0.064691	0.999313	0.121516	
M	Test	0.536669	0.684982	0.015692	1.0	0.030899	0.54665
	Validierung	0.511273	0.690928	0.012981	0.983333	0.025624	
	Training	0.525502	0.691962	0.0	0.0	0.0	
L	Test	0.529282	0.691477	0.0	0.0	0.0	0.468927
	Validierung	0.504956	0.694664	0.0	0.0	0.0	

Tabelle 2: Performanzmetriken vollständiger Understand-Datensätze binärer Klassifikationen

Die Ergebnisse der binären Genauigkeit sowie des berechneten Fehlers sind für alle Trainings-, Test- und Validierungsdatensätze der verschiedenen Architekturen ähnlich. Anhand der Komplexität der zugrunde liegenden Architektur werden die Ergebnisse der Recall-Werte bei steigender Komplexität reduziert und die Ergebnisse der Präzisionswerte erhöht. Für das Experiment der L-Architektur ergibt sich ein Recall-Wert von 0, auf dessen Grundlage keine weiteren Ergebnisse bezüglich der Performanz-Metriken der Präzision und des F1-Scores in diesem Fall ermittelt werden konnten. Aufgrund der starken der Reduzierung des Wertes der AUC-Metrik für den Testdatensatz konnte eine Verschlechterung der Leistungsfähigkeit dieses Modells ermittelt werden. Basierend auf der hohen Wert der Präzision und dem niedrigen Wert des Recalls des Experiments der M-Architektur besteht die Annahme, dass durch das Experiment der L-Architektur dieses Verhalten verstärkt wird und alle Beispiele mit dem Auftreten keiner Schwachstellen klassifiziert werden.

Datensatz		Genauigkeit	Verlust	Recall	Präzision	F1-Score	AUC
	Training	0.503653	0.692761	0.459431	0.501834	0.479697	
S	Test	0.505319	0.693023	0.489614	0.494012	0.491803	0.50599
	Validierung	0.509522	0.69247	0.335124	0.510747	0.404704	
	Training	0.500484	0.693098	0.310235	0.49759	0.382186	
M	Test	0.493230	0.693274	0.34817	0.475034	0.401826	0.492166
	Validierung	0.499499	0.693069	0.530557	0.497168	0.51332	
	Training	0.500836	0.694006	0.78257	0.499267	0.609612	
L	Test	0.501934	0.693916	0.744807	0.49377	0.593849	0.508936
	Validierung	0.50284	0.693742	0.630625	0.500266	0.557932	

Tabelle 3: Performanzmetriken vollständiger Analizo-Datensätze binärer Klassifikationen

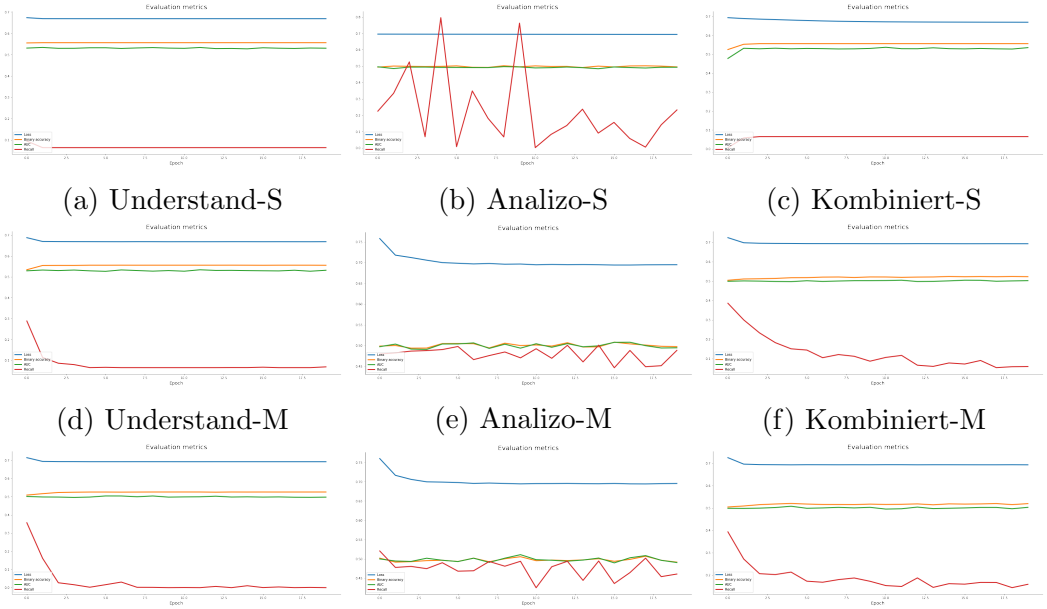
Die Ergebnisse der Experimente unter Verwendung des vollständigen Analizo-Datensatzes werden in Tabelle 3 dargestellt. Die Experimente verfügen über ähnliche Werte hinsichtlich des berechneten Fehlers sowie der Performanzmetriken der binären Genauigkeit und der Präzision. Im Vergleich zu den Ergebnissen des Understand-Datensatzes ergibt sich jedoch ein gering erhöhter Fehler bei einer niedrigeren Genauigkeit und einer deutlich geringeren Präzision. Gleichzeitig sind die Recall-Werte der Trainings-, Test- und Validierungsdatensätze für den Analizo-Datensatz deutlich höher. Dabei kann jedoch kein Zusammenhang hinsichtlich der Modellkomplexität identifiziert werden. Während das Experiment unter Verwendung der M-Architektur den geringsten Recall-Wert des Analizo-Datensatzes repräsentiert, verfügt das Modell unter Einsatz der L-Architektur den höchsten Recall-Wert. Das Experiment der M-Architektur verfügt über den geringsten F1-Score und das Experiment der L-Architektur über den höchsten F1-Score.

Datensatz		Genauigkeit	Verlust	Recall	Präzision	F1-Score	AUC
S	Training	0.555037	0.667816	0.090052	0.764062	0.161114	0.547249
	Test	0.535438	0.682757	0.048384	0.578125	0.089295	
	Validierung	0.509422	0.695565	0.043784	0.557423	0.081191	
M	Training	0.556177	0.668057	0.064691	0.999313	0.121516	0.542622
	Test	0.536669	0.685115	0.015692	1.0	0.030899	
	Validierung	0.511273	0.691037	0.012981	0.983333	0.025624	
L	Training	0.525502	0.69191	0.0	0.0	0.0	0.51331
	Test	0.529282	0.691375	0.0	0.0	0.0	
	Validierung	0.504956	0.694223	0.0	0.0	0.0	

Tabelle 4: Performanzmetriken der vollständigen kombinierten Datensätze binärer Klassifikationen

Die Ergebnisse der Experimente des vollständigen kombinierten Datensatezs werden durch Tabelle 4 repräsentiert. Die ermittelten Werte der Verlustfunktion sowie der Performanzmetriken ähneln den Ergebnissen des Understand-Datensatzes. Für die Performanzmetriken der binären Genauigkeit sowie des berechneten Fehlers sind die Werte aller Modell-Architekturen ähnlich. Bei steigender Modellkomplexität wird der Recall-Wert sowie der Wert des F1-Scores stetig reduziert, während sich die Präzisionswerte stetig erhöhen. Im Vergleich zu den Ergebnissen des Understand-Datensatzes sind alle Werte des Recalls und des F1-Scores jedoch sginifikant geringer und alle Präzisionwerte signifikant höher. Gleichzeitig konnten ebenfalls für die Performanzmetriken des Recalls, der Präzision und des F1-Scores keine Werte für die L-Architektur berechnet werden. Die Werte der AUC-Metrik des Testdatensatzes sind im Vergleich zur Verwendung der anderen Architekturen ebenfalls geringer. Für Experimente der L-Architektur gilt ebenfalls die Annahme, dass alle Beispiele mit dem Auftreten keiner Schwachstelle klassifiziert wurden.

Abbildung 18 visualisiert Ergebnisse der Trainingsprozesse aller durch die Anwendung der korrelationsbasierten Merkmalsauswahl reduzierten Datensätze un-



(a) Understand-S (b) Analizo-S (c) Kombiniert-S

(d) Understand-M (e) Analizo-M (f) Kombiniert-M

(g) Understand-L (h) Analizo-L (i) Kombiniert-L

Abbildung 18: Visualisierung der Performanz-Metriken reduzierter Trainingsdatensätze im Verlauf des Lernprozesses

ter Verwendung der verschiedenen Modell-Architekturen. Alle Diagramme beschreiben den Verlauf des berechneten Fehlers (blau) sowie der Performanzmetriken der binären Genauigkeit (gelb), des Recalls (rot) sowie der AUC-Metrik (grün) nach Abschluss jeder Epoche. Diese Ergebnisse ähneln den Ergebnissen der Experimente unter Verwendung vollständiger Datensätze. Der berechnete Fehler der Verlustfunktion ist ebenfalls in jedem Experiment ähnlich hoch und konnte nur in den ersten Epochen leicht minimiert werden. Die AUC-Metrik sowie die binäre Genauigkeit verfügen in der Regel im gleichen Maße über fast konstante Werte, welche für jedes Experiment ähnlich sind und im Verlauf des Trainings nicht sginifikant optimiert werden konnten.

Für die Experimente des Analizo-Datensatzes variierten ebenso die Werte des Recalls ohne signifikante Optimierungen stetig bei einem deutlich höheren Skalenniveau als in den Experimenten der anderen Datensätze. Mit Ausnahme des Experiments des kombinierten Datensatzes unter Verwendung der S-Architektur konnte dieser Wert im Verlauf des Trainingsprozesses stetig reduziert werden. Für alle weiteren Experimente dieser Datensätze ist der Recall-Wert der ersten Epoche bei steigender Modellkomplexität signifikant höher. In den Experimenten des kombinierten Datensatzes variierte der Recall-Werte bei stetiger Reduzierung minimal. Diese Variationen können ebenfalls auf der unausgeglichenen Zusammensetzung des kombinierten Datensatzes beruhen.

Auf Grundlage dieser Diagramme werden die Ergebnisse der Performanzmetriken unter Verwendung der unterschiedlichen Architekturen in Kombination mit den einzelnen reduzierten Datensätze ausgewertet. Tabelle 5 stellt für die Experimente des reduzierten Understand-Datensatzes die resultierenden Ergebnisse bereit. Im Vergleich zum vollständigen Understand-Datensatz verfügt der binäre Datensatz über eine ähnliche binäre Genauigkeit und über einen ähnlichen Fehler. Der Recall-Wert ist für alle Architekturen sehr gering und tendiert in Kombination mit der L-Architektur gegen 0, sodass ebenfalls keine Werte für die Performanzmetriken der Präzision und des F1-Scores berechnet werden konnten. Für die anderen Architekturen ist die Präzision jedoch bereits für das Modell der S-Architektur sehr hoch und tendiert gegen 1. Daher besteht ebenfalls die Annahme, dass die Modelle aller drei Architekturen den Großteil aller Beispiele mit dem Auftreten keiner Schwachstelle klassifizieren. Darüber hinaus sind die Werte des F1-Scores sehr gering und die Werte der AUC-Metrik im Vergleich zu den Experimenten des vollständigen Understand-Datensatzes signifikant geringer.

Datensatz		Genauigkeit	Verlust	Recall	Präzision	F1-Score	AUC
S	Training	0.556177	0.668592	0.064691	0.999313	0.121516	0.507846
	Test	0.536669	0.68595	0.015692	1.0	0.030899	
	Validierung	0.511273	0.691447	0.012981	0.983333	0.025624	
M	Training	0.556177	0.668079	0.064691	0.999313	0.121516	0.507846
	Test	0.536669	0.686032	0.015692	1.0	0.030899	
	Validierung	0.511273	0.692459	0.012981	0.983333	0.025624	
L	Training	0.525502	0.691857	0.0	0.0	0.0	0.5
	Test	0.529282	0.691508	0.0	0.0	0.0	
	Validierung	0.504956	0.693757	0.0	0.0	0.0	

Tabelle 5: Performanzmetriken reduzierter Understand-Datensätze binärer Klassifikationen

Die Ergebnisse der Experimente unter Verwendung des reduzierten Analizo-Datensatzes werden in Tabelle 6 dargestellt. Die Experimente verfügen über ähnliche Werte bezüglich des berechneten Fehlers sowie der Performanzmetriken AUC und der binären Genauigkeit im Vergleich zu den Experimenten des vollständigen Analizo-Datensatzes. Für die Experimente der Architekturen S und L wurde ein Recall-Wert von 0 ermittelt, auf dessen Grundlage keine Werte für der Performanzmetriken der Präzision und des F1-Scores bereitgestellt werden konnten. Daher besteht ebenfalls die Annahme, dass diese Modelle alle Beispiele mit dem Auftreten keiner Schwachstelle klassifizieren. Das Experiment der M-Architektur verfügt hingegen über einen Recall-Wert von 1 und einen hohen F1-Score für alle Teildatensätze. Dabei repräsentieren die Performanzmetriken der Präzision und die Performanzmetrike der binären Genauigkeit einen identischen Wert. In diesem Fall besteht die Annahme, dass in diesem Experiment alle Beispiele mit dem Auftreten einer Schwachstelle klassifiziert wurden.

Datensatz		Genauigkeit	Verlust	Recall	Präzision	F1-Score	AUC
	Training	0.501981	0.693449	0.0	0.0	0.0	
S	Test	0.511122	0.693095	0.0	0.0	0.0	0.499011
	Validierung	0.502506	0.693676	0.0	0.0	0.0	
	Training	0.498019	0.693255	1.0	0.498019	0.664904	
M	Test	0.488878	0.693440	1.0	0.488878	0.656707	0.5
	Validierung	0.497494	0.693272	1.0	0.497494	0.664436	
	Training	0.501981	0.693325	0.0	0.0	0.0	
L	Test	0.511122	0.6929	0.0	0.0	0.0	0.5
	Validierung	0.502506	0.693298	0.0	0.0	0.0	

Tabelle 6: Performanzmetriken reduzierter Analizo-Datensätze binärer Klassifikationen

Die Ergebnisse der Experimente des reduzierten kombinierten Datensatezs werden durch Tabelle 7 repräsentiert. Die ermittelten Werte der Verlustfunktion sowie der Performanzmetriken ähneln den Ergebnissen des vollständigen kombinierten Datensatzes. Die AUC-Werte der Testdatensätze sind im Vergleich den Ergebnissen des vollständigen kombinierten Datensatzes hingegen signifikant geringer. Für die Performanzmetriken der binären Genauigkeit, des Recalls, der Präzision sowie des F1-Scores und der AUC-Metrik sind die Werte der Modell-Architekturen M und L identisch. In diesem Zusammenhang konnten für die Metriken des Recalls, der Präzision und des F1-Scores keine Werte für die Experimente dieser beiden Architekturen berechnet werden. Dementsprechend gilt für diese beiden Experimente ebenfalls die Annahme, dass alle Beispiele mit dem Auftreten keiner Schwachstelle klassifiziert wurden. Für das Experiment der S-Architektur sind die Werte des Recalls und des F1-Scores ebenfalls sehr gering, während der Präzisionswert sehr hoch ist.

Datensatz		Genauigkeit	Verlust	Recall	Präzision	F1-Score	AUC
	Training	0.556177	0.66937	0.064691	0.999313	0.121516	
S	Test	0.536669	0.686246	0.015692	1.0	0.030899	0.507846
	Validierung	0.511273	0.691506	0.012981	0.983333	0.081191	
	Training	0.525502	0.691891	0.0	0.0	0.0	
M	Test	0.529282	0.691435	0.0	0.0	0.0	0.5
	Validierung	0.504956	0.694472	0.0	0.0	0.0	
	Training	0.525502	0.692136	0.0	0.0	0.0	
L	Test	0.529282	0.691544	0.0	0.0	0.0	0.0
	Validierung	0.504956	0.695077	0.0	0.0	0.0	

Tabelle 7: Performanzmetriken der vollständigen reduzierten Datensätze binärer Klassifikationen

Die Ergebnisse aller Modelle weisen eine binäre Genauigkeit von unter 60% auf, während der Wert des Recalls dieser Modelle sehr gering ist. Daher werden viele Beispiele ohne das Auftreten von Schwachstellen klassifiziert, obwohl in vielen Fällen eine Schwachstelle vorhanden ist. In einigen Fällen konnte jedoch eine hohe Präzision erzielt werden, sodass tatsächliche Schwachstellen mit einer hohen Wahrscheinlichkeit erkannt werden. Die Anwendung zufälliger Klassifikationen zur Erkennung des Auftretens von Schwachstellen resultiert in diesem Kontext jedoch in einer ähnlichen

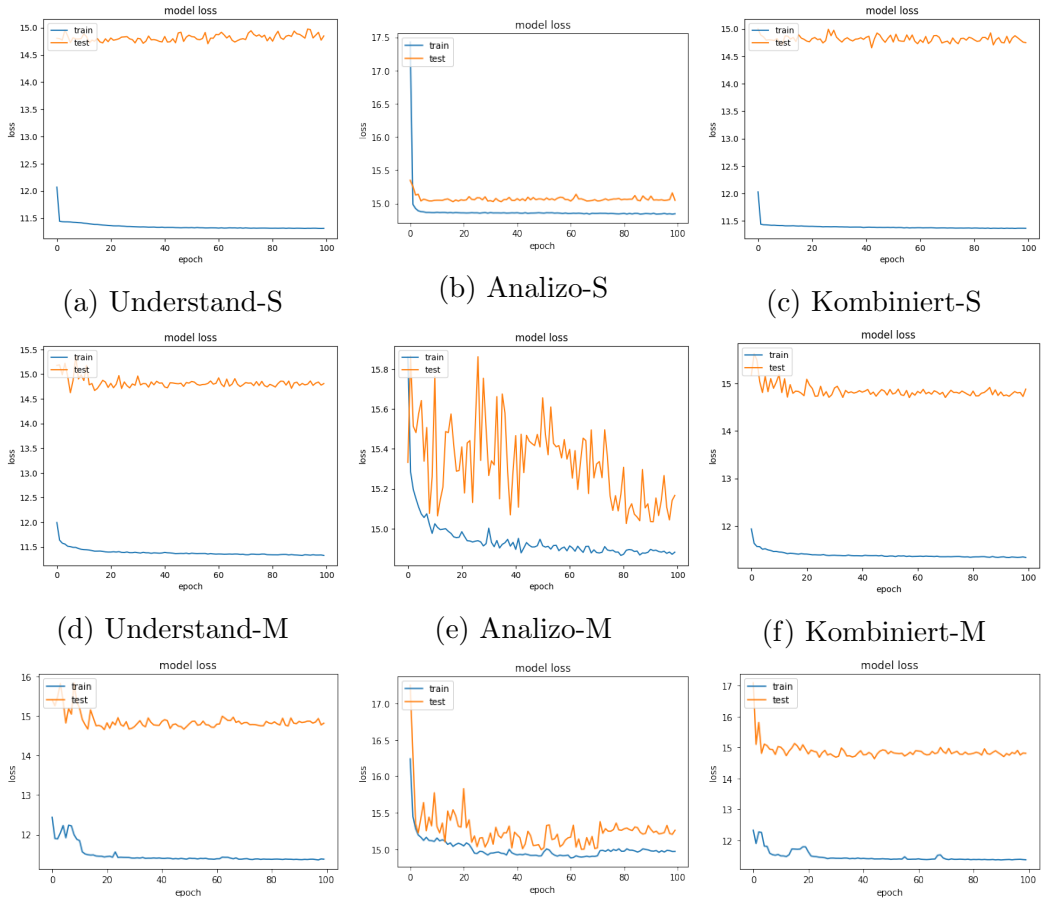
Leistung. Aufgrund der hohen Anzahl fehlerhafter Klassifikationen eignet sich deshalb dieser Ansatz der Schwachstellenerkennung nicht.

Die schlechte Leistung dieser Modelle beruht auf vor allem auf der Qualität der bereitgestellten Daten. In den meisten Fällen existieren Beispiele einer Datei-Version mit und ohne Schwachstellen. Die auf Grundlage dieser Datei-Versionen extrahierten Software-Produktmetriken unterscheiden sich aufgrund feingranularer Änderungen des Quellcodes gering. Dadurch konnte bereits im Rahmen der Korrelationsanalyse kein konkreter Zusammenhang zwischen dem Auftreten von Schwachstellen und den bereitgestellten Software-Produktmetriken identifiziert werden. Auf Grundlage dieser Eigenschaften wird die Leistung der Modelle binärer Klassifikationen stark beeinträchtigt. Gleichzeitig konnte durch die Anwendung der korrelationsbasierten Merkmalsauswahl aufgrund starker Korrelationen zwischen einzelnen Merkmalen kein reduzierter Datensatz ermittelt werden, auf dessen Grundlage gute Ergebnisse entstehen können. Dementsprechend verfügen die Modelle reduzierter Datensätze ebenfalls über keine gute Performanz, sodass sich die Extraktion von Software-Produktmetriken zur Erkennung des Auftretens von Schwachstellen im Rahmen dieser Experimente nicht eignet.

6.6 Bewertung von Schwachstellen durch Regression

Zur Beantwortung der Forschungsfrage RQ2 werden im Rahmen dieses Kapitels die Ergebnisse der Modelle von Regressionsaufgaben ausgewertet. Abbildung 19 beschreibt die Entwicklung der Verlustfunktion des mittleren quadratischen Fehlers der auf den vollständigen Datensätzen basierenden Experimente. In diesen Diagrammen werden die ermittelten Fehler auf Grundlage der Trainings- (blau) und Validierungsdatensätze (gelb) für jede Epoche eines Experiments visualisiert.

Im Vergleich der Entwicklung des mittleren quadratischen Fehlers verfügen die Experimente des vollständigen Understand-Datensatzes und des vollständigen kombinierten Datensatzes über die meisten Ähnlichkeiten. Der Unterschied des Fehlers basierend auf dem Trainingsdatensatz ist innerhalb dieser Experimente deutlich geringer als der Fehler des Validierungsdatensatzes. Auf Grundlage des Trainingsdatensatzes konnte der mittlere quadratische Fehler vor allem in den ersten Trainingsepochen signifikant reduziert werden, während der Fehler des Validierungsdatensatzes lediglich mit steigender Modellkomplexität signifikant reduziert werden konnte. Dementsprechend könnte eine leichte Erhöhung der Verallgemeinerungsfähigkeit dieser Modelle bei steigender Modellkomplexität bestehen. Da der Fehler der Validierungsdaten dennoch deutlich höher ist, könnte ebenfalls eine Überanpassung der Modelle basierend auf den Trainingsdatensätzen existieren, welche bei steigender



(a) Understand-S (b) Analizo-S (c) Kombiniert-S

(d) Understand-M (e) Analizo-M (f) Kombiniert-M

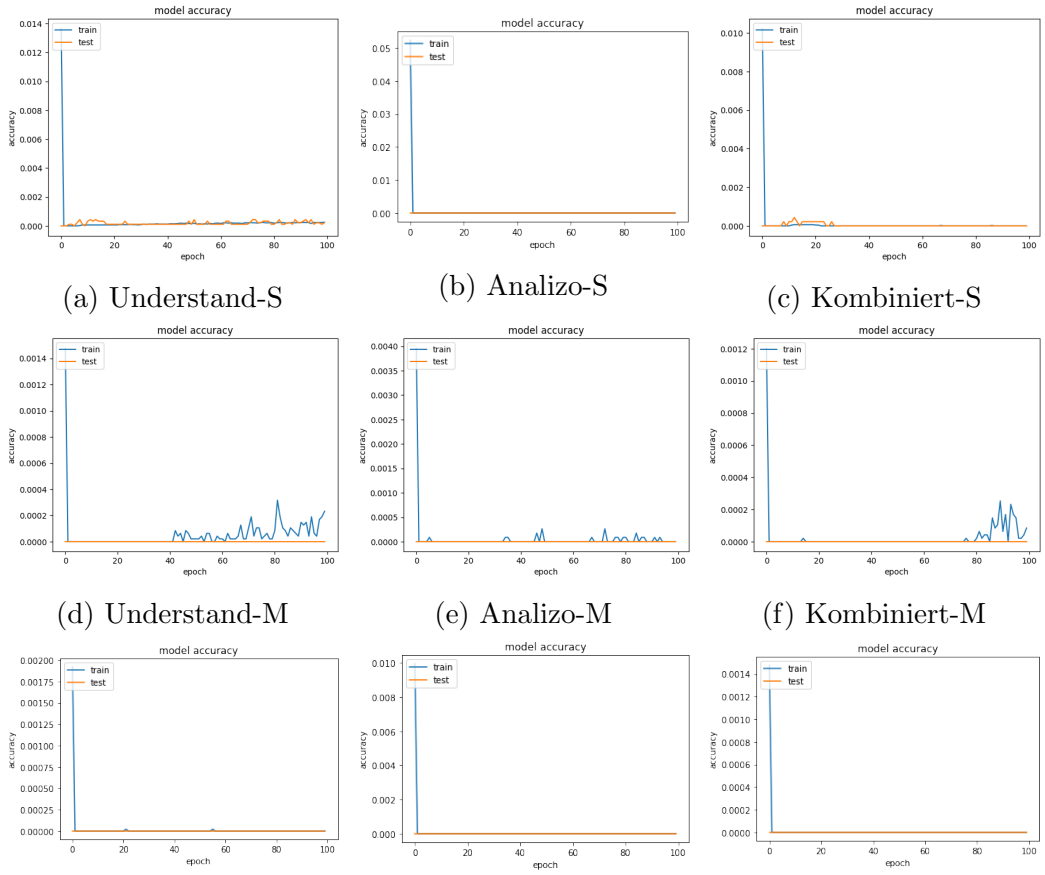
(g) Understand-L (h) Analizo-L (i) Kombiniert-L

Abbildung 19: Visualisierung des berechneten Fehlers vollständiger Trainingsdatensätze im Verlauf des Lernprozesses

Modellkomplexität lediglich leicht minimiert wird.

Unter Verwendung des vollständigen Analizo-Datensatzes resultieren hingegen sowohl für den Trainings- als auch für den Validierungsdatensatz mittlere quadratische Fehler, die in der Regel höher als der Fehler der Validierungsdaten der anderen Datensätze sind. Allerdings ist der Unterschied des Fehlers zwischen Trainings- und Validierungsdatensatz deutlich geringer, während bei steigender Modellkomplexität der Fehler des Validierungsdatensatzes im Verlauf der Epochen stark variiert. Dadurch besteht die Annahme, dass eine Überanpassung bezüglich des Trainignsdatensatzes nicht vollzogen wurde. Darüber hinaus wird mittlere quadratische Fehler der Modelle des vollständigen Analizo-Datensatzes ebenfalls in den ersten Epochen des Lernprozesses signifikant reduziert.

Die Entwicklung der Performanzmetrik der Genauigkeit wird im Verlauf aller Trainingsepochen unter Verwendung der vollständigen Datensätze in Abbildung 20 für Trainings- und Validierungsdatensätze dargestellt. Hinsichtlich der Untersuchung



(a) Understand-S (b) Analizo-S (c) Kombiniert-S

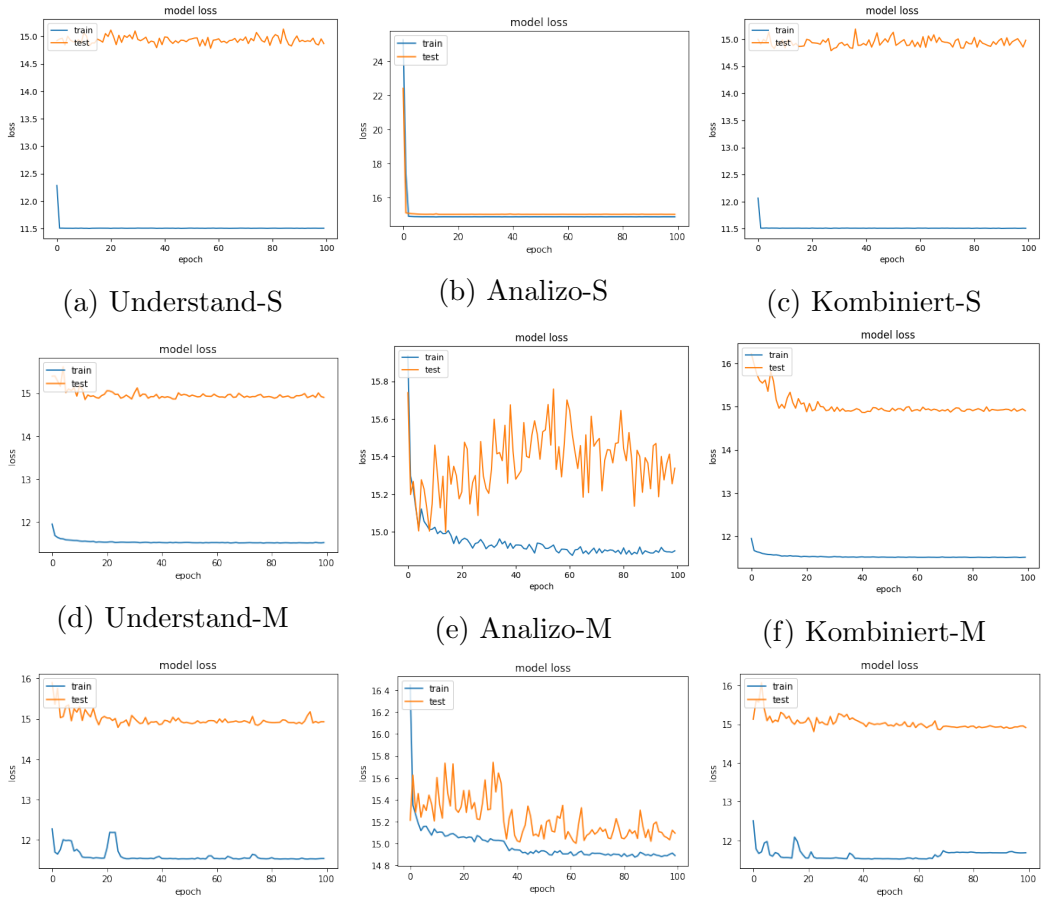
(d) Understand-M (e) Analizo-M (f) Kombiniert-M

(g) Understand-L (h) Analizo-L (i) Kombiniert-L

Abbildung 20: Visualisierung der Performanz-Metrik der Genauigkeit vollständiger Trainingsdatensätze im Verlauf des Lernprozesses

der Genauigkeit dieser Modelle verfügen die Experimente der gleichen Modellarchitekturen der verschiedenen vollständigen Datensätze über viele Ähnlichkeiten. Die Genauigkeit der Vorhersagen basierend auf den Validierungsdatensätzen ist im Verlauf aller Epochen mit wenigen Ausnahmen stetig 0. Auf Grundlage des Trainingsdatensatzes ist die Genauigkeit innerhalb dieser Experimente in der ersten Epochen leicht erhöht und wird in den folgenden Epochen in der Regel auf den Wert 0 reduziert. Unter Verwendung der M-Architekturen konnten während der Trainingsprozesse dennoch leichte Erhöhungen der Genauigkeit auf einen Wert unter 0,5% ermittelt wewrden. Zu Beginn des Trainings wird jedoch stets eine Genauigkeit von maximal 5% berechnet, welche im Rahmen der Auswertung bereits eine schlechte Performanz repräsentiert.

In Abbildung 21 wird die Entwicklung des mittleren quadratischen Fehlers im Verlauf aller Epochen für Trainings- und Validierungsdaten der reduzierten Datensätze visualisiert. Die Entwicklung dieser Fehler verfügt in der Regel über starke Ähnlichkeiten zu den Ergebnissen der jeweiligen Experimente vollständiger Da-



(a) Understand-S (b) Analizo-S (c) Kombiniert-S

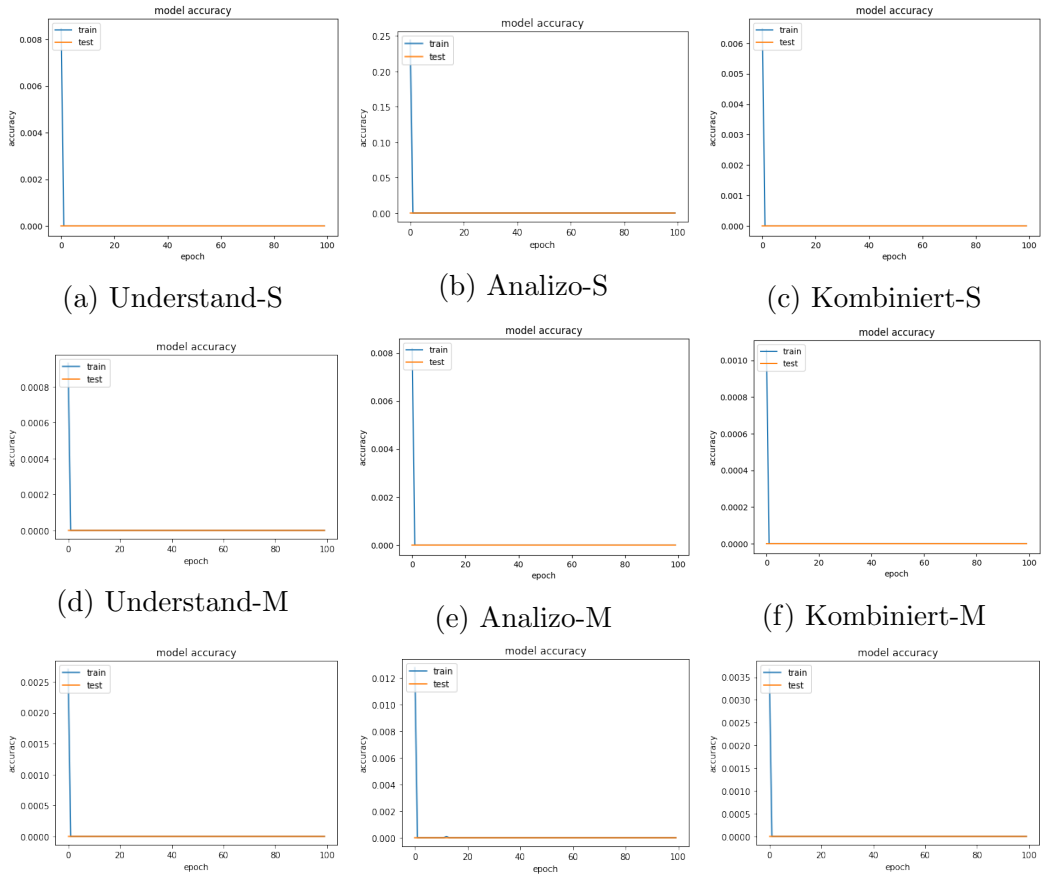
(d) Understand-M (e) Analizo-M (f) Kombiniert-M

(g) Understand-L (h) Analizo-L (i) Kombiniert-L

Abbildung 21: Visualisierung des berechneten Fehlers reduzierter Trainingsdatensätze im Verlauf des Lernprozesses

tensätze. Grundsätzlich werden in diesen Experimenten ähnliche Fehler der Verlustfunktion berechnet, sodass der Fehler der Trainingsdatensätze im Vergleich zum Fehler der Validierungsdatensätze unter Verwendung des reduzierten Understand-Datensatzes und des reduzierten kombinierten Datensatzes geringer ist. Mithilfe des reduzierten Analizo-Datensatzes werden ebenfalls höhere Fehler ermittelt, welche für den Trainings- und Validierungsdatensatz keine signifikanten Unterschiede aufweisen. Der mittlere quardratische Fehler dieses Datensatezs variiert ebenfalls bei steigender Modellkomplexität im Verlauf des Trainingsprozeses und wird allerdings unter Verwendung der M-Architektur letztendlich leicht erhöht.

Die Entwicklung der Genauigkeit während des Trainingsprozesses der Experimente wird unter Verwendung der reduzierten Datensätze in Abbildung 21 bereitgestellt. Der Verlauf dieser Diagramme ist für alle dieser Experimente ähnlich. Zu Beginn der Trainignsprozesse verfügt das Modell innerhalb der ersten Epoche unter Betrachtung des Trainingsdtensatzes eine Genauigkeit von maximal 1%. Im Verlauf



(a) Understand-S (b) Analizo-S (c) Kombiniert-S

(d) Understand-M (e) Analizo-M (f) Kombiniert-M

(g) Understand-L (h) Analizo-L (i) Kombiniert-L

Abbildung 22: Visualisierung der Performanz-Metrik der Genauigkeit reduzierter Trainingsdatensätze im Verlauf des Lernprozesses

des Trainings wird diese Genauigkeit in den folgenden Epochen ausnahmelos auf 0 reduziert. Dabei bestehen keine Ausreißer korrekter Bewertungen, sodass diese Modelle grundsätzlich eine schlechte Performanz aufweisen.

Alle untersuchten Experimente weisen hohe mittlere Fehler in Zusammenhang mit sehr geringen Genauigkeiten auf. Dementsprechend verfügen die resultierenden Modelle dieser Experimente auf Grundlage der Ergebnisse über keine gute Performanz zur Bewertung der Schwere von Schwachstellen.

Regressionsaufgaben stellen im Vergleich zur binären Klassifikation eine deutlich komplexere Herausforderung dar. Basierend auf der schlechten Performanz der Modelle binärer Klassifikationen kann die Performanz dieser Modelle auf die gleichen Ursachen zurückgeführt werden. Durch geringe Unterschiede extrahierter Software-Produktmetriken verschiedener Versionen gleicher Dateien in Zusammenhang mit einer hohen Anzahl fehlender Metriken-Werte wird die Qualität der Datensätze stark reduziert. Aufgrund der schlechten Datenqualität und schwachen Korrelationen zwischen der Zielgröße und Merkmalen wird die Performanz dieser Modelle stark

beeinträchtigt, wodurch letztendlich keine korrekten Vorhersagen resultieren. Innerhalb reduzierter Datensätze existieren zusätzlich widersprüchliche Daten, auf deren Grundlage keine korrekten Vorhersagen bezüglich der Schwere von Schwachstellen getroffen werden können. Dementsprechend eignet sich die Extraktion von Software-Produktmetriken im Rahmen der Experimente nicht zur Bewertung der Schwere von Schwachstellen.

6.7 Software-Qualitätsmodelle auf Grundlage der Verwendung von Software-Produktmetriken

Basierend auf den Ergebnissen zur Erkennung des Auftretens von Schwachstellen und der Bewertung ihrer Schwere sollen alle verwendeten Software-Produktmetriken hinsichtlich ihrer Verwendung innerhalb von Software-Qualitätsmodellen beurteilt werden. Auf Grundlage der verschiedenen Hauptqualitätskriterien und den jeweiligen Aspekten weden die Möglichkeiten zur Verwendung verschiedener Software-Produktmetriken in den entsprechenden Unterkapiteln thematisiert.

6.7.1 Bewertung der Funktionalität

Die Bewertung der Funktionalität basiert auf der Auswertung aufgestellter Anforderungen, welche im Kontext eines Software-Produkts spezifischen Gegebenheiten unterliegen. Die Erfüllung funktionaler Anforderungen basiert auf der Vollständigkeit, Korrektheit und Angemessenheit realisierter Implementierungen und wird in der Regel durch die Ausführung verschiedener Testszenarien sichergestellt. Grundsätzlich lassen sich die Aspekte der Funktionalität nur durch die Ausführung entsprechender Code-Abschnitte bewerten. Software-Produktmetriken repräsentieren jedoch nur eine statische Sichtweise eines Software-Produkts, wodurch das Kriterium der Funktionalität sowie deren Teilaspekte der Vollständigkeit, Korrektheit und Angemessenheit nicht durch solche Metriken bewertet werden können. Vor allem die Beurteilung der Angemessenheit bereitgestellter Funktionalitäten kann auf Grundlage der Charakteristiken des Quellcodes nicht durchgeführt werden.

6.7.2 Bewertung der Benutzbarkeit

Die Bewertung der Benutzbarkeit eines Software-Produkts basiert auf der Verwendung solcher Systeme auf Grundlage der Ausführung des Quellcodes. Eine Beurteilung findet in diesem Kontext durch spezifische Anforderungen statt, welche die Bedürfnisse der Benutzer dieser Software-Produkte charakterisieren. Diese Anforderungen beschreiben die Erkenn-, Erlern- und Bedienbarkeit von Software-Produkten

für eine optimale Verwendung. Darüber hinaus bestehen weitere Anforderungen bezüglich des Schutzes vor Fehlbedienungen sowie der Ästhetik und Zugänglichkeit eines Software-Produkts Mithilfe der statischen Repräsentation von Eigenschaften des Quellcodes können die Charakteristiken dieser Bedürfnisse nicht wiedergespiegelt werden. Dadurch eignen sich solche Metriken nicht zur Bewertung der Benutzbarkeit eines Software-Produkts.

6.7.3 Bewertung der Zuverläsigkeit

Die Bewertung der Zuverläsigkeit eines Software-Produkts basiert auf Ausführungen bestimmter Funktionalitäten unter spezifizierten Bedingungen innerhalb einer bestimmten Zeitperiode. Diese Charakteristiken durch die Aspekte der Reife und Zuverlässigkeit beschrieben. Darüber hinaus beschreiben die Aspekte der Fehlertoleranz und der Wiederherstellbarkeit das erwartete Fehlverhalten eines Software-Produkts. Diese Aspekte thematisieren ebenfalls die Ausführung eines Systems unter spezifischen Anforderungen, wodurch sich die Bewertung durch statische Eigenschaften von Software-Produktmetriken nicht eignet.

6.7.4 Bewertung der Wartbarkeit

Die Bewertung der Wartbarkeit eines Software-Produkts basiert auf dessen Zusammensetzung hinsichtlich der Effiktivität und Effizienz von Änderungen und weiteren Anpassungen von Systemen. Unter dem Kriterium der Wartbarkeit werden Aspekte zusammengefasst, die sowohl den modularen Aufbau des Produkts und die Wiederverwendbarkeit von Komponenten als auch die Analysier- und Modifizierbarkeit dieser Systeme charakterisieren. Darüber hinaus enthält dieses Kriterium ein Aspekt bezüglich der Möglichkeiten zur Definition von Testkriterien, welche im Rahmen der Wartbarkeit einen wichtigen Faktor darstellen. Dieser Aspekt basiert jedoch auf der Ausführung der Funktionalitäten eines Produkts, wodurch sich die Verwendung von Software-Produktmetriken zur Bewertung umfangreicher Testoptionen nicht eignet.

Die Zusammensetzung eines Software-Produkts kann wiederum durch die statischen Eigenschaften von Software-Produktmetriken repräsentiert werden, da diese auf der Auswertung des implementierten Quellcodes basieren. Dementsprechend lassen sich die Eigenschaften eines modularen Aufbaus sowie der Wiederverwendbarkeit von Komponenent durch die Extraktion von Software-Produktmetriken erfassen. Für die Beschreibung eines modularen Aufbaus eignen sich beispielsweise Kohäsions- und Kopplungsmetriken, welche die Interkationen innerhalb einzelner Komponenten sowie die Interaktionen zwischen verschiedenen Komponenten charakterisieren.

Für die Beschreibung der Wiederverwendbarkeit von Komponenten eignen sich darüber hinaus Komplexitäts- und Vererbungsmetriken. Mittels der Anwendung von Vererbungsprinzipien kann beispielsweise die Wiederverwendbarkeit einzelner Komponenten erhöht werden. Durch eine steigende Komplexität innerhalb von Komponenten wird die Wiederverwendbarkeit von Komponenten bei notwendigen Anpassungen beeinträchtigt.

Die Bewertung von Aspekten der Analysier- und Modifizierbarkeit kann ebenfalls auf Grundlage der aufgelisteten Metriken stattfinden. Beide Aspekte werden durch die Komplexität des Quellcodes bedingt und durch die Eigenschaften der Kohäsion, Kopplung und Vererbung verstärkt oder reduziert. Neben den bereits erwähnten Kategorien Software-Produktmetriken können sich Netzwerkmetriken ebenfalls zur Bewertung dieser Aspekte eignen.

Diese Metriken wurden allerdings nicht im Rahmen dieses Projekts verwendet, da ausschließlich die Metriken unter Betrachtung einzelner Entitäten und Komponenten extrahiert wurden. Daher sollten Metriken dieser Kategorie sowie aller weiteren erwähnten Metriken-Kategorien für die Bewertung dieser Aspekte sorgfältig überprüft werden.

6.7.5 Bewertung der Effizienz

Die Bewertung der Effizienz eines Software-Produkts basiert auf seiner Leistung bezüglich benötigter Verarbeitungs- und Antwortzeiten sowie auf der effektiven Nutzung von Ressourcen und der Schonung von Kapazitäten. Diese Aspekte werden durch spezifische Anforderungen im Rahmen einer tatsächlichen Verwendung dieser Produkte durch die Ausführung des entsprechenden Quellcodes definiert. Auf Grundlage der statischen Repräsentation von Eigenschaften des Quellcodes werden jedoch die Charakteristiken von Prozessen nicht erfasst. Stattdessen wird eine dynamische Repräsentation der Quellcode-Eigenschaften benötigt, wodurch Software-Produktmetriken nicht für die Bewertung der Effizienz eines Software-Produkts geeignet sind.

6.7.6 Bewertung der Kompatibilität

Die Bewertung der Kompatibilität eines Software-Produkts basiert auf der Kommunikation mit anderen Produkten sowie des Teilens benötigter Ressourcen unter bestimmten Bedingungen. Diese Aspekte werden durch spezifische Anforderungen beschrieben, die auf Grundlage der Verwendung bestimmter Umgebungen oder im Zusammenhang mit anderen Software-Produktmetriken definiert werden. Dementsprechend kann die Kompatibilität eines Software-Produkts nur durch des-

sen Ausführung unter den gegebenen Bedingungen beurteilt werden. Auf Grundlage der Repräsentation der statischen Charakteristiken des Quellcodes mittels Software-Produktmetriken kann die Erfüllung solcher Anforderungen nicht erfüllt werden. Stattdessen verfügt die Erfassung dynamischer Eigenschaften von Prozessmetriken über ein höheres Potenzial zur Bewertung der Kompatibilitätseigenschaften.

6.7.7 Bewertung der Portierbarkeit

Die Bewertung der Portierbarkeit eines Software-Produkts basiert auf den verschiedenen Aspekten der Adaptivität, Installierbarkeit und Austauschbarkeit. Der Aspekt der Installierbarkeit beruht auf Grundlage der Ausführung spezifischer Prozesse zur Initialisierung und Bereitstellung des jeweiligen Produkts durch die Definition spezifischer Anforderungen. Statische Eigenschaften von Software-Produktmetriken können das Verhalten dieser Prozesse nicht beurteilen, wodurch sie sich für eine Bewertung der Installierbarkeit eines Software-Produkts nicht eignen.

Die Aspekte der Adaptivität und der Austauschbarkeit basieren hingegen besipielsweise auf der Wartbarkeit eines Software-Produkts. Gut wartbare Systeme können beispielsweise auf Grundlage eines modularen Aufbaus, wiederverwendbarer Komponenten, leichter Analysier- und Modifizierbarkeit die Adaptivität eines Systems erhöhen. Durch umfangreiche Testoptionen kann die Anpassung eines Systems für anderen Umgebungen ebenfalls erleichtert werden. Mithilfe einer guten Analysierbarkeit der Funktionalitäten eines Software-Produkts kann zusätzlich die Austauschbarkeit eines Systems besser beurteilt werden.

Dementsprechend basiert die Bewertung dieser beiden Aspekte einerseits auf der Zusammensetzung eines Software-Produkts. Daher eignet sich die Verwendung von Komplexitäts-, Kohäsions-, Kopplungs- und Vererbungsmetriken zur Bewertung der Adaptivität und Austauschbarkeit von Software-Produktmetriken.

Andererseits können im Rahmen dieser Aspekte weitere Anforderungen definiert werden, auf dessen Grundlage sich eine Bewertung durch Software-Produktmetriken eignet. Die Austauschbarkeit von Software-Produkten kann besipielsweise auch sehr durch das Verhalten bereitgestellter Funktionalitäten und Prozessen bedingt sein. Daher sollten die aufgelisteten Software-Produktmetriken zur Bewertung dieser Qualitätsaspekte sorgfältig überprüft werden.

6.7.8 Bewertung der Sicherheit

Die Bewertung der Sicherheit eines Software-Produkts im Rahmen eines Software-Qualitätsmodells basiert auf Sicherstellung des Datenschutzes, Vermeidung unautorisierten Zugriffs sowie der Nachweisbarkeit von Handlungen und Identitäten. Durch

die fehlerhafte Realisierung dieser Aspekte können Schwachstellen innerhalb eines Software-Produkts entstehen, durch deren Ausnutzung die Sicherheit dieser Anwendungen beeinträchtigt wird.

Im Rahmen dieses Projektes wurde ein Ansatz zur Erkennung und Bewertung der Schwere von Schwachstellen unter Verwendung von Software-Produktmetriken entworfen. Basierend auf den Ergebnissen der Experimente konnten nur wenige, schwache Korrelationen zwischen diesen Metriken und den untersuchten Schwachstellen identifiziert werden. Anhand des Trainings von Modellen künstlicher neuronaler Netze wurden in diesem Zusammenhang schlechte Leistungen bezüglich der Erkennung und Bewertung von Schwachstellen erzielt werden. Daher eignen sich die verwendeten Metriken nur bedingt zur Bewertung der Software-Sicherheit.

Auf Grundlage der Experimente konnten jedoch allerdings nicht alle Arten von Schwachstellen analysiert werden, während verwendete extrahierte Datensätze ebenfalls keine hohe Qualität aufweisen konnten. Lösungen zur Erkennung des Auftretens von Schwachstellen unter Verwendung von Software-Produktmetriken existieren bereits und verfügen zum Teil über eine hohe binäre Genauigkeit. In der Regel werden innerhalb dieser Lösungsansätze verschiedene Komplexitäts-, Quantifizierungs- und Netzwerkmetriken eingesetzt. Für die Bewertung der Schwere existieren allerdings laut aktuellem Stand der Forschung keine Ansätze zur Bewertung der Schwere von Schwachstellen auf Grundlage von Software-Produktmetriken. Daher werden weitere, umfangreiche Experimente höherer Qualität benötigt, um die Bewertung von Sicherheitsaspekten durch Software-Produktmetriken zu beurteilen.

7 Fazit

Im Rahmen dieses Kapitels werden alle relevanten Prozesse zur Durchführung der Experimente im Zusammenhang mit ihren resultierenden Ergebnissen hervorgehoben. Diese Informationen werden in Kapitel 7.1 zusammengefasst. Basierend auf dieser Zusammenfassung werden alle wesentlichen Erkenntnisse dieses Projektes in Kapitel 7.2 beschrieben und die Möglichkeiten zur Erweiterung und Optimierung der Experimente in Kapitel 7.3 thematisiert.

7.1 Zusammenfassung

Der Zusammenhang zwischen Software-Produktmetriken und dem Auftreten von Schwachstellen innerhalb des Quellcodes von Software-Projekten wurde im Rahmen dieses Projektes untersucht. Unter Verwendung künstlicher neuronaler Netzwerke sollten dazu verschiedene Modelle zur Erkennung des Auftretens von Schwachstellen und der Bewertung ihrer Schwere erstellt werden. Die Erkennung des Auftretens von Schwachstellen sollte auf Grundlage binärer Klassifikationen und die Bewertung ihrer Schwere basierend auf Regressionen realisiert werden. Auf Grundlage der Ergebnisse dieser Modelle sollten anschließend die verwendeten Software-Produktmetriken hinsichtlich ihrer Möglichkeiten zur Verwendung innerhalb von Software-Qualitätsmodellen untersucht werden.

Zur Erstellung der Datensätze von Software-Produktmetriken für den Einsatz innerhalb der Modelle künstlicher neuronaler Netze diente der CVEfixes-Datensatz als Grundlage. Dieser Datensatz enthält CVE-Einträge historischer Schwachstellen und stellt die veränderten Quellcode-Abschnitte von Open-Source-Projekten bereit, die zur Behebung dieser Schwachstellen benötigt werden. Dabei wird neben weiteren Informationen der Quellcode auf Datei- und Methoden-Ebene vor und nach Behebung der jeweiligen Schwachstelle zur Verfügung gestellt. Darüber hinaus basiert dieser Datensatz auf einem automatisierten Prozess, der am 27.03.2023 zur Aktualisierung dieses Datensatzes ausgeführt wurde. Mittels dieser Aktualisierung konnte die Anzahl enthaltener CVE-Einträge nahezu verdoppelt werden.

Basierend auf diesem aktualisierten Datensatz wurden unter Betrachtung des Quellcodes einzelner Dateien Software-Produktmetriken durch den Einsatz der Werkzeuge Understand und Analizo extrahiert. Die Integration dieser Werkzeuge zur Extraktion von Software-Produktmetriken basierte auf der Programmiersprache Python. Im Rahmen dieses Prozesses konnten über 70 verschiedene Metriken für über 40.000 verschiedene Versionen von Dateien durch das Werkzeug Understand extrahiert werden. Unter Verwendung des Werkzeugs Analizo konnten wiederum 12 ver-

schiedene Software-Produktmetriken für über 16.000 verschiedene Datei-Versionen bereitgestellt werden.

Auf Grundlage dieser Metriken-Extraktion wurden drei verschiedene Datensätze definiert, die jeweils auf den extrahierten Metriken der einzelnen Werkzeuge sowie auf einer Kombination der Metriken beider Werkzeuge basieren. Innerhalb dieser Datensätze wurde zusätzlich die verwendete Programmiersprache als kategorisches Merkmal integriert, welches im Rahmen der Vorverarbeitung der Datensätze durch mehrere binäre Merkmale abgebildet wird. Darüber hinaus wurden für die beiden Aufgaben der künstlichen neuronale Netze zwei verschiedene Zielgrößen definiert. Für die Erkennung des Auftretens von Schwachstellen wurde eine binäre Zielgröße definiert, welche bestimmt jeweilige Version einer Datei, ihren Zustand vor oder nach Behebung einer Schwachstelle repräsentiert. Zur Bewertung der Schwere wurde wiederum eine Zielgröße basierend auf den durch den CVEfixes-Datensatz bereitgestellten Informationen von $CVSS_3$-Wertungen definiert.

Aufgrund einer hohen Anzahl fehlender Werte extrahierter Metriken wird die Qualität dieser Datensätz jedoch beeinträchtigt. Diese fehlenden Werte wurden durch die Anwendung der Strategie einer konstanten Ersetzung durch den Wert 0 im Rahmen der Vorverarbeitung angepasst. Anschließend wurden unter Verwendung der Korrelationsmaße Pearson, Spearmn, und Kendall Korrelationsanalysen zur Untersuchung potenzieller Zusammenhänge zwischen den Merkmalen und den beiden Zielgrößen zu identifizieren. Basierend auf den Ergebnissen dieser Korrelationsanalysen konnten allerdings nur wenige schwache Korrelationen binär abgebildeter Merkmale der verwendeten Programmiersprachen ermittelt werden. Diese Ergebnisse können teilweise auf die Qualität des Datensatzes aufgrund einer hohen Anzahl fehlender Werte zurückgeführt werden. Im Vergleich der extrahierten Werte von Software-Produktmetriken verschiedener Versionen gleicher Dateien existieren jedoch innerhalb vieler Beispiele keine Unterschiede signifikanten Unterschiede, wodurch die Ergebnisse der Korrelationsanalysen zusätzlich beeinflusst werden.

Zusätzlich wurden für beide Zielgrößen die Technik der korrelationsbasierten Merkmalsauswahl angewendet, um die Dimensionalität der Datensätze zu reduzieren. Neben den schwachen Korrelationen zwischen den Merkmalen und den beiden Zielgrößen bestehen hingegen viele mittlere und starke Korrelationen zwischen den einzelnen Merkmalen. Aufgrund dieser Eigenschaften wurden die Datensätze auf bis zu maximal zwei binäre Merkmale von Programmiersprachen reduziert, welche über die höchsten Korrelationen zu den jeweiligen Zielgrößen verfügen. Dementsprechend verfügen diese Datensätze nur über Informationen bezüglich verwendeter Programmiersprachen, wodurch die Aussagekraft dieser Datensätze kritisch zu betrachten

ist. Darüber hinaus verfügen diese Datensätze über viele widersprüchliche Beispiele, wodurch die Qualität dieser Datensätze stark beeinträchtigt wird.

Im Rahmen der Vorverarbeitung aller definierten Datensätzen wurden die Werte aller Merkmale in einem Intervall von $[0, 1]$ skaliert und in Trainings-, Validierungs- und Testdaten mit einem Verhältnis von 70/20/10 aufgeteilt. Die Aufteilung aller Beispiele basiert auf ihrer Zugehörigkeit zu dem entsprechenden Open-Source-Projekt, sodass eine Form der projektübergreifenden Schwachstellenerkennung simuliert werden konnte.

Auf Grundlage der vorverarbeiteten Datensätze wurden drei verschiedene Architekturen künstlicher neuronaler Netze unterschiedlicher Komplexitäten definiert, welche sich in der Anzahl verwendeter Schichten und Neuronen unterscheiden. Diese Architekturen wurden als S, M und L bezeichnet und basieren auf dem Prinzip von Feedforward-Netzwerken. Zwischen den verborgenen Schichten der Architekturen M und L wurden außerdem die Strategien der Normalisierung und des Dropouts integriert. Durch die Ergebnisse trainierter Modelle binärer Klassifikationen zur Erkennung des Auftretens von Schwachstellen verfügen jedoch über eine schlechte Leistung aufgrund einer binären Genauigkeit von unter 60% und einem hohen Fehler ihrer Verlustfunktion. Die trainierten Modelle zur Bewertung der Schwere von Schwachstellen resultieren wiederum in einer noch schlechteren Leistung. Diese Modelle verfügen ebenfalls über einen hohen Fehler der Verlustfunktion und eine Genauigkeit von 0%.

7.2 Schlussfolgerungen

Die Aktualisierung des CVEfixes-Datensatzes 18 Monate nach Initialisierung des ursprünglichen Datensatzes resultierte annähernd in einer Verdopplung bereitgestellter CVE-Einträge und Quellcode-Abschnitten. In diesem Zusammenhang ist ein Trend der Zunahme an Open-Source-Projekten mit Veröffentlichung identifizierter Schwachstellen erkennbar. Auf Grundlage dieses Wachstums stellt der Prozess zur Aktualisierung CVEfixes-Datensatz eine interessante Methode dar, um im Laufe der Zeit eine deutlich größere Datenbasis zu erzeugen.

Mithilfe des Metriken-Extraktionstools Understand kann eine hohe Anzahl an Metriken unter Betrachtung einer großen Vielfalt verschiedener Programmiersprache auf unterschiedlichen Granularitätsebenen ermittelt werden. Das Analizo-Tool stellt hingegen nur eine deutlich geringere Anzahl an Metriken bereit, die nur auf Grundlage drei verschiedener Programmiersprachen auf Projekt- und Modul-Ebene extrahiert werden können. Aufgrund solcher Beschränkungen wird die Erstellung von aus Software-Produktmetriken bestehenden Datensätzen beschränkt. Das Werkzeug

Analizo stellt jedoch im Vergleich zu den wenigen anderen Metriken-Extraktionstools eine deutlich höhere Anzahl an Funktionalitäten bereit. Solche Werkzeuge sind in der Regel veraltet oder können nur auf kommerzieller Basis erworben werden, sodass nur sehr wenige dieser Tools die Anforderungen des Projekts erfüllen.

Aufgrund fehlender Daten innerhalb der erzeugten Datensätzen und geringen Unterschieden zwischen den Beispielen gleicher Dateien ist eine Verbesserung der Datenqualität zur Erhöhung der Leistung der Modelle künstlicher neuronaler Netze notwendig. Im Zusammenhang mit der Optimierung der Datenqualität können sich ebenfalls die Ergebnisse der Korrelationsanalysen sowie der korrelationsbasierten Merkmalsauswahl verbessern. Die Datenqualität muss jedoch nicht zwangsweise die einzige Ursache bezüglich der Identifikation weniger schwacher Korrelationen sein, weswegen die die Anwendung der korrelationsbasierten Merkmalsauswahl in diesem Kontext nicht den besten Ansatz zur Reduzierung der Dimensionalität von Merkmalsmengen darstellt. Stattdessen besteht ebenfalls die Möglichkeit, dass durch die Optimierung der Datenqualität dennoch ähnliche Ergebnisse erzielt werden.

Fehlende Korrelationen deuten jedoch nicht grundsätzlich auf eine fehlende Kauslität bezüglich des Auftretens von Schwachstellen durch bestimmte charakteristische Eigenschaften des Quellcodes hin. Aufgrund der geringen Qualität der Datensätze ist die schlechte Leistung der trainierten Modelle künstlicher neuronaler Netze dennoch begründbar.

7.3 Ausblick

Auf Grundlage der Verdopllung der Einträge des CVEfixes-Datensatzes nach über 18 Monaten seit seiner initialen Veröffentlichung eignet sich eine Replikation der Experimente zu späteren Zeitpunkten. Durch eine Erhöhung der Anzahl bereitgestellter Beispiele kann die Optimierung der Datenqualität mittels der Anwendung weiterer Techniken erleichtert werden. Mithilfe einer größeren Datenbasis können beispielsweise Einträge herausgefiltert werden, die zu wenige Informationen extrahierter Software-Produktmetriken bereitstellen, ohne einen zu kleinen Datensatz zu erzeugen.

Darüber hinaus können die Experimente auf feineren Granularitätsebenen ausgeführt werden, um die geringen Unterschiede zwischen den verschiedenen Quellcode-Beispielen einer Datei zu abzubilden. Eine Extraktion von Metriken auf Methoden-Ebene kann sich beispielsweise in diesem Zusammenhang eignen. Weiterhin können die Experimente durch die Betrachtung dieser Entitäten in Zusammenhang mit den anderen durch einen Commit bereitgestellten Entitäten oder durch die vollständige Betrachtung der entsprechenden Projekt-Version erweitert werden. Dieser Ansatz

sollte jedoch aufgrund eines hohen Ressourcenbedarfs möglichst effizient und performant implementiert werden.

Die Erzeugung der Modelle künstlicher neuronaler Netze kann im Rahmen dieser Experimente ebenfalls optimiert werden. Einerseits können beispielsweise durch die Anwendung anderer Strategien zur Ersetzung fehlender Daten effektivere Methoden der Vorverarbeitung verwendet werden, um die Leistung der Modelle zu erhöhen. Darüber hinaus können Experimente zur Optimierung von Hyperparametern dieser Modelle durchgeführt sowie weitere Architekturen verschiedener Arten künstlicher neuronaler Netze untersucht werden. Dementsprechend kann ebenfalls die Anwendung weiterer Methoden zur Reduzierung der Dimensionalität der Merkmalsmengen angewandt werden.

8 Literaturverzeichnis

[1] John Viega und Gary McGraw. „Building Secure Software: How to Avoid Security Problems the Right Way“. In: (Jan. 2001).

[2] Guru Bhandari, Amara Naseer und Leon Moonen. „CVEfixes: Automated Collection of Vulnerabilities and Their Fixes from Open-Source Software“. In: *Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering*. PROMISE 2021. Athens, Greece: Association for Computing Machinery, 2021, S. 30–39. ISBN: 9781450386807. DOI: 10.1145/3475960.3475985. URL: https://doi.org/10.1145/3475960.3475985.

[3] Marcus Geiger. *Safety vs. Security: Der Unterschied einfach erklärt (und wie Sie beide Ziele kombinieren können)*. 2020. URL: https://www.sichere-industrie.de/safety-security-unterschied-erklaert-kombination-ziele-industrial-security/ (besucht am 29. 08. 2023).

[4] Gary McGraw. „Software Security: Building Security In“. In: *2006 17th International Symposium on Software Reliability Engineering* (2006), S. 6–6. URL: https://api.semanticscholar.org/CorpusID:25146846.

[5] Das Wirtschaftslexikon. *Qualität*. 2016. URL: http://www.daswirtschaftslexikon.com/e/qualit%C3%A4t/qualit%C3%A4t.htm (besucht am 07. 01. 2023).

[6] INZTITUT - Inspiraton . Innovation . Digitalisierung. *ISO 25010*. 2012. URL: https://inztitut.de/blog/glossar/iso-25010/.

[7] ISO 25000 software und data quality. *ISO 25010*. 2022. URL: https://iso25000.com/en/iso-25000-standards/iso-25010.

[8] Celestino Madera Castro. „Software-Produktmetriken zur Erstellung eines Software-Qualitätsmodells“. In: (2022).

[9] Alberto S. Nuñez-Varela u. a. „Source code metrics: A systematic mapping study“. In: *Journal of Systems and Software* 128 (2017), S. 164–197. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2017.03.044. URL: https://www.sciencedirect.com/science/article/pii/S0164121217300663.

[10] SciTools. *Understand: The Software Developer's Multi-Tool*. 2023. URL: https://scitools.com/ (besucht am 02. 07. 2023).

[11] Analizo. *Analizo — mulit-language source code analysis suite*. 2022. URL: https://www.analizo.org/ (besucht am 02. 07. 2023).

[12] M. Traeger u. a. „Künstliche neuronale Netze“. In: *Der Anaesthesist* 52.11 (Nov. 2003), S. 1055–1061. ISSN: 1432-055X. DOI: 10.1007/s00101-003-0576-x. URL: https://doi.org/10.1007/s00101-003-0576-x.

[13] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[14] Tian Yang und Vikas Agrawal. „Log file anomaly detection“. In: *CS224d Fall 2016* (2016), S. 1–7.

[15] Laurenz Wuttke. *Künstliche Neuronale Netzwerke: Definition, Einführung, Arten und Funktion*. 2023. URL: https://datasolut.com/neuronale-netzwerke-einfuehrung/ (besucht am 27. 07. 2023).

[16] Yann LeCun, Yoshua Bengio und Geoffrey Hinton. „Deep learning“. In: *Nature* 521.7553 (Mai 2015), S. 436–444. ISSN: 1476-4687. DOI: 10.1038/nature14539. URL: https://doi.org/10.1038/nature14539.

[17] Sagar Sharma. *Activation Functions in Neural Networks*. 2017. URL: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6 (besucht am 27. 07. 2023).

[18] Patrick Planing. *Statistik Grundlagen - Das interaktive ebook mit YouTube Erklärvideos*. 2022. URL: https://statistikgrundlagen.de/ebook/ (besucht am 05. 07. 2023).

[19] Miltiadis Siavvas, Dionysios Kehagias und Dimitrios Tzovaras. „A Preliminary Study on the Relationship Among Software Metrics and Specific Vulnerability Types“. In: *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*. 2017, S. 916–921. DOI: 10.1109/CSCI.2017.159.

[20] Austria. DATAtab Team (2023). DATAtab: Online Statistics Calculator. DATAtab e.U. Graz. *Korrelationsanalyse — Einfach erklärt — DATAtab*. 2023. URL: https://datatab.de/tutorial/korrelation (besucht am 05. 07. 2023).

[21] Marie-Therese Puth, Markus Neuhäuser und Graeme D. Ruxton. „Effective use of Spearman's and Kendall's correlation coefficients for association between two measured traits“. In: *Animal Behaviour* 102 (2015), S. 77–84. ISSN: 0003-3472. DOI: https://doi.org/10.1016/j.anbehav.2015.01.010. URL: https://www.sciencedirect.com/science/article/pii/S0003347215000196.

[22] Mark Hall. „Correlation-based Feature Selection for Discrete and Numeric Class Machine Learning“. In: Jan. 2000, S. 359–366.

[23] Johannes S. Fischer. *Correlation-based Feature Selection in Python from Scratch.* 2021. URL: https://johfischer.com/2021/08/06/correlation-based-feature-selection-in-python-from-scratch/ (besucht am 05. 07. 2023).

9 Abbildungsverzeichnis

10 Tabellenverzeichnis

11 Anlagen

11.1 Anlage A - Qualitätsaspekte nach ISO/IEC 25010

11.1.1 Anlage A.1 - Aspekte der Funktionalität

Aspekt	Beschreibung
Vollständigkeit der Software-Funktionen	Dieser Aspekt beschreibt die Abdeckung aller spezifizierten Aufgaben und Ziele von Benutzern durch die Menge aller Funktionen.
Funktionale Korrektheit	Dieser Aspekt beschreibt die Bereitstellung korrekter Ergebnisse im benötigten Präzisionsgrad durch ein System.
Angemessenheit der Funktionalität	Dieser Aspekt beschreibt den Grad der Erleichterung zur Erfüllung spezifizierter Aufgaben und Ziele durch die Funktionen eines Systems.

Tabelle 8: Aspekte funktionaler Software [6] [7]

11.1.2 Anlage A.2 - Aspekte der Benutzbarkeit

Aspekt	Beschreibung
Optimale Erkennbarkeit	Dieser Aspekt beschreibt die Erkennung von Angemessenheit eines Produkts oder Systems durch Benutzer hinsichtlich ihrer Bedürfnisse.
Leichte Erlernbarkeit und Lernfähigkeit	Dieser Aspekt beschreibt die Verwendbarkeit eines Produkts oder Systems durch spezifizierte Benutzer bei Erreichung spezifzierter Lernziele zur Anwendung des Produkts oder Systems mit Effektivität, Effizienz und Risikofreiheit in einem spezifizierten Anwendungskontext.
Gute Bedienbarkeit	Dieser Aspekt behandelt die Existenz von Attributen eines Produkts oder Systems zur Erleichterung seiner Bedienung und Steuerung.
Schutz vor Fehlbedienung durch Nutzer	Dieser Aspekt thematisiert den Schutz von Benutzern gegen die Verursachung von Fehlern.
Ästetische Benutzerschnittstelle	Dieser Aspekt thematisiert die Ermöglichung angenehmer und befriedigender Interaktionen für Benutzer durch Benutzerschnittstellen.
Leichter Zugang	Dieser Aspekt behandelt die Verwendung eines Produkts oder Systems durch Menschen hinsichtlich einer breiten Vielfalt an Eigenschaften und Fähigkeiten zur Erfüllung spezifizierter Ziele in einem spezifizierten Anwendungskontext.

Tabelle 9: Aspekte einfach benutzbarer Software [6] [7]

11.1.3 Anlage A.3 - Aspekte der Zuverlässigkeit

Aspekt	Beschreibung
Ausgereifte Software-Qualität	Dieser Aspekt beschreibt den Grad der Erfüllung von Anforderungen der Zuverlässigkeit eines Produkts, eines Systems oder einer Komponente im Normalbetrieb.
Verfügbarkeit	Dieser Aspekt umfasst die Betriebsbereitschaft und Zugänglichkeit eines Produkts, eines Systems oder einer Komponente bei dessen Benötigung zur Verwendung.
Fehlertoleranz	Dieser Aspekt beschreibt bestimmungsmäßige Funktionalität eines Produkts, eines Systems oder einer Komponente trotz vorhandener Software- oder Hardwarefehler.
Wiederherstellbarkeit	Dieser Aspekt thematisiert die Wiederherstellung eines Produkts, eines Systems oder einer Komponente im Falle einer Unterbrechung oder eines Ausfalls direkt betroffener Daten sowie den gewünschten Systemzustands.

Tabelle 10: Aspekte zuverlässiger Software [6] [7]

11.1.4 Anlage A.4 - Aspekte der Wartbarkeit

Aspekt	Beschreibung
Modularer Aufbau	Dieser Aspekt beschreibt die Zusammensetzung eines Systems oder Computer-Proramms in diskrete Komponenten mit minimalen Einfluss durch die Änderungen einer Komponente auf anderen Komponenten.
Wiederverwendbare Komponenten	Dieser Aspekt thematisiert die Möglichkeiten zur Verwendung einer Komponente in mehr als einem System oder zur Erzeugung weiterer Komponenten.
Gute Analyse-Funktion	Dieser Aspekt behandelt Effektivität und Effizienz hinsichtlich der Möglichkeiten zur Bewertung des Einflusses beabsichtiger Änderungen einzelner oder mehrerer Abschnitte eines Produkts oder Systems sowie zur Diagnositizierung von Fehlern und Mängeln eines Produkts als auch zur Identifikation zu verändernder Abschnitte.
Leichte Modifizierbarkeit	Dieser Aspekt behandelt die Effektivität und Effizienz zur Veränderung eines Systems ohne Integration von Defekten und der Verschlechterung der Produktqualität.
Umfangreiche Testoptionen	Dieser Aspekt behandlet die Effektivität und Effizienz zur Integration von Testkriterien in einem Produkt, einem System oder einer Komponente und der Möglichkeit der Durchführung von Tests zur Überprüfung der Testkriterien hinsichtlich ihrer Erfüllung.

Tabelle 11: Aspekte einfach wartbarer Software [6] [7]

11.1.5 Anlage A.5 - Aspekte der Effizienz

Aspekt	Beschreibung
Gutes Zeitverhalten	Dieser Aspekt thematisiert Antwort- und Verarbeitungszeiten sowie Durchsatzraten eines Systems bei Durchführung seiner Funktionen zur Erfüllung festgelegter Anforderungen.
Effektive Nutzung von Ressourcen	Dieser Aspekt beschreibt den Umfangs und die Art von Ressourcen, welche durch ein System bei Durchführung seiner Funktionen zur Erfüllung der Anforderungen verwendet werden.
Schonung von Kapazitäten	Dieser Aspekt beschreibt maximale Grenzen eines Systems durch Parameter zur Erfüllung der Anforderungen.

Tabelle 12: Aspekte effizienter Software [6] [7]

11.1.6 Anlage A.6 - Aspekte der Kompatibilität

Aspekt	Beschreibung
Optimale Co-Existenz zu weiterer Software	Dieser Aspekt beschreibt die effiziente Durchführung benötigter Funktionen eines Produkts während des Teilens gemeinsamer Umgebungen und Ressourcen mit anderen Produkten ohne schädlichen Einfluss auf andere Produkte.
Interoperabilität	Dieser Aspekt behandelt die Übermittlung von Informationen und der Verwendung übermittelter Informationen zwischen zwei oder mehr Produkten, Systemen oder Komponenten.

Tabelle 13: Aspekte kompatibler Software [6] [7]

11.1.7 Anlage A.7 - Aspekte der Portierbarkeit

Aspekt	Beschreibung
Gute Adaptivität	Dieser Aspekt thematisiert effektive und effiziente Anpassung eines Produkts oder Systems für verschiedene sich entwickelnden Hardware- und Software-Komponenten sowie anderen Betriebs- oder Nutzungsumgebungen.
Leichte Installierbarkeit	Dieser Aspekt behandelt die Effektivität und Effizienz zur erfolgreichen Installtion und Deinstalltion eines Produkts oder Systems in einer spezifizierten Umgebung.
Einfache Austauschbarkeit	Dieser Aspekt beschreibt die Ersetzbarkeit eines Produkts durch ein anderes spezifiziertes Software-Produkt für den gleichen Zweck in der gleichen Umgebung.

Tabelle 14: Aspekte portierbarer Software [6] [7]

11.1.8 Anlage A.8 - Aspekte der Sicherheit

Aspekt	Beschreibung
Datenschutz	Dieser Aspekt beschreibt die Sicherstellung der Zugänglichkeit für ausschließlich Zugriffsberechtigte Benutzer und Systeme durch das Produkt oder System.
Integrität	Dieser Aspekt thematisiert die Vermeidung unautorisierten Zugriffs und Vermeidung von Manipulationen der Computer-Programme und Daten des Produkts oder Systems.
Keine Manipulierbarkeit	Dieser Aspekt behandelt die Nachweisbarkeit von Handlungen und Ereignissen ohne der Möglichkeit einer späteren Widerlegung dieser Handlungen und Ereignisse.
Sichere Administration und geschützte Benutzer-Accounts	Dieser Aspekt thematisiert die eindeutige Zurückverfolgung von Handlungen durch eine Entität sowie die Zurückverfolgung von Handlungen, welcher auf einer Entität ausgeführt worden.
Authentizierbarkeit	Dieser Aspekt beschreibt die Nachweisbarkeit der Identität eines Subjekts oder einer Ressource hinsichtlich konkreter Beanspruchungen.

Tabelle 15: Aspekte sicherer Software [6] [7]

11.2 Anlage B - Kategorien der Software-Produktmetriken

Kategorie	Beschreibung
Quantifizierungs-metriken	Quantifizierungsmetriken umfassen alle Metriken, welche die Anzahl Vorkommnisse bestimmter Entitäten erfassen.
Komplexitätsmetriken	Komplexitätsmetriken betrachten die Komplexität einer Software auf verschiedenen System-Ebenen hinsichtlich verschiedener Aspekte. Aufgrund fehlender theoretischer Grundlagen und mangelnder Verallgemeinerungen bezüglich des Komplexitätsbegriffs existieren unterschiedliche Ansätze zur Messung der Software-Komplexität.
Kohäsionsmetriken	Kohäsionsmetriken repräsentieren den Verwandtschaftsgrad zwischen verschiedenen Entitäten eines Software-Moduls. Kohäsion kann beispielsweise auf Grundlage der Interaktionen zwischen Methoden und Attributen ermittelt werden. Darüber hinaus kann die Kohäsion auch auf Grundlage struktureller Merkmale wie beispielsweise der Referenzierung gleicher Entitäten berechnet werden.
Kopplungsmetriken	Kopplungsmetriken beschreiben die Unabhängigkeit zwischen Entitäten wie beispielsweise Methoden oder Klassen. Diese Unabhängigkeit basiert auf der Stärke von Assoziationen als Verbindungen zwischen verschiedenen Modulen eines Systems. In der Regel werden Methodenaufrufe und die Referenzierung von Attributen als Assoziationen verwendet.
Vererbungsmetriken	Vererbungsmetriken beruhen auf der Auswertung der Entitäten anhand des Konzeptes der Vererbung. Es können sowohl Eigenschaften von Vererbungsbeziehungen als auch die Wiederverwendbarkeit geteilter Attribute und die Auswertung der gesamten Vererbungshierarchie als Berechnungsgrundlage verwendet werden.
Sicherheitsmetriken	Sicherheitsmetriken repräsentieren die Angriffsoberfläche eines Software-Produkts, welche aus der Summe aller Ein- und Ausgangspunkte einzelner Komponenten gebildet werden. Das Prinzip der Kapselung kann dabei ebenfalls berücksichtigt werden.
Netzwerkmetriken	Netzwerkmetriken beschreiben die Abhängigkeiten zwischen einzelnen Entitäten eines Software-Produkts. Diese Metriken werden in der Regel von Abhängigkeitsgraphen abgeleitet. Dieser Graph enthält alle vorhandenen Entitäten als Knoten-Elemente. Die Kanten dieses Graphen repräsentieren wiederum bestimmte Abhängigkeitsbeziehungen.

Tabelle 16: Kategorien der Software-Produktmetriken [8]

11.3 Anlage C - Metriken der Extraktionswerkzeuge

11.3.1 Anlage C.1 - Metriken des Extraktionswerkzeugs Understand

Metrik	Beschreibung
AMLOC	*Durchschnittliche Anzahl Code-Zeilen einer Methode (Average Lines)*: Durchschnittliche Anzahl an Code-Zeilen aller Methode
ABLOC	*Durchschnittliche Anzahl leerer Code-Zeilen (Average Blank Lines)*: Durchschnittliche Anzahl leerer Code-Zeilen aller Methoden
ABLOCI	*Durchschnittliche Anzahl leerer Code-Zeilen (Average Blank Lines includes inactive)*: Durchschnittliche Anzahl leerer Code-Zeilen aller Methoden inklusive inaktiver Code-Zeilen
ASLOC	*Durchschnittliche Anzahl Quellcode-Zeilen (Average Code Lines)*: Durchschnittliche Anzahl an Zeilen des Quellcodes aller Methoden
ASLOCI	*Durchschnittliche Anzahl Quellcode-Zeilen (Average Code Lines includes inactive)*: Durchschnittliche Anzahl an Zeilen des Quellcodes aller Methoden inklusive inaktiver Code-Zeilen
ALCOMM	*Durchschnittliche Anzahl Kommentarzeilen (Average Comment Lines)*: Durchschnittliche Anzahl an Kommentarzeilen aller verschachtelten Funktionen und Methoden
ALCOMMI	*Durchschnittliche Anzahl Kommentarzeilen (Average Comment Lines includes inactive)*: Durchschnittliche Anzahl an Kommentarzeilen aller verschachtelten Funktionen und Methoden inklusive inaktiver Code-Zeilen
AVG	*Durchschnittliche zyklomatische Komplexität (Average Cyclomatic Complexity)*: Durchschnittliche zyklomatische Komplexität aller verschachtelten Funktionen und Methoden
AMVG	*Durchschnittliche modifizierte zyklomatische Komplexität (Average Modified Cylcomatic Complexity)*: Durchschnittliche modifizierte zyklomatische Komplexität aller verschachtelten Funktionen und Methoden
ASVG	*Durchschnittliche strikte zyklomatische Komplexität (Average Strict Cyclomatic Complexity)*: Durchschnittliche strikte zyklomatische Komplexität aller verschachtelten Funktionen und Methoden
ASVG	*Durchschnittliche strikte, modifizierte zyklomatische Komplexität (Average Strict Modified Cyclomatic Complexity)*: Durchschnittliche strikte, modifizierte zyklomatische Komplexität aller verschachtelten Funktionen und Methoden

Tabelle 16 – Fortsetzung Understand-Metriken

Metrik	Beschreibung
AEVG	*Durchschnittliche essentielle zyklomatische Komplexität (Average Essential Complexity)*: Durchschnittliche essentielle zyklomatische Komplexität aller verschachtelten Funktionen und Methoden
SUPC, FANUP	*Anzahl an Superklassen (Base Classes)*: Anzahl der Superklassen einer bestimmten Klasse
CBO	*Kopplung zwischen Objekten (Coupled Classes)*: Anzahl aller mit einer Klasse durch Methodenaufrufe, Feldzugriffe, Vererbung, Argumenten, Rückgabewerten und Ausnahmebehandlungen gekoppelten Klassen
CBOM	*Kopplung zwischen Objekten (Coupled Classes Modified)*: Anzahl aller mit einer Klasse durch Methodenaufrufe, Feldzugriffe, Vererbung, Argumenten, Rückgabewerten und Ausnahmebehandlungen gekoppelten nicht standardisierten Klassen
NSC	*Anzahl an Kindern (Derived Classes)*: Anzahl aller direkten Unterklassen einer Klasse
NCLASS, NC	*Anzahl an Klassen (Classes)*: Anzahl aller Klassen eines Systems oder einer Komponente
NOA, NF	*Anzahl an Attributen (Class Variables)*: Anzahl aller Attribute einer Klasse
NOM	*Anzahl an Methoden (Class Methods)*: Anzahl aller Methoden einer Klasse
EU	*Anzahl ausführbarer Programmeinheiten (Executable Units):* Anzahl aller Programmeinheiten mit ausführbaren Code
NF	*Anzahl an Dateien (Files):* Anzahl aller Dateien
NCF	*Anzahl an Code-Dateien (Code Files):* Anzahl aller Code-Dateien
NHF	*Anzahl an Header-Dateien (Header Files):* Anzahl aller Header-Dateien
NOF	*Anzahl an Funktionen (Functions):* Anzahl aller Funktionen
NIM	*Anzahl an Instanz-Methoden (Instance Methods)*: Anzahl an in einer Klasse implementierten Methoden mit Zugriff auf lokale Daten durch eine Instanz als Objekt der Klasse
NIV, INST	*Anzahl an Instanz-Variablen (Instance Variables)*: Anzahl aller Instanz-Variablen einer Klasse
NIIV	*Anzahl interner Instanz-Variablen (Internal Instance Variables)*: Anzahl interner Instanz-Variablen einer Klasse
NOPA, PUBA, NOAP, NPV	*Anzahl öffentlicher Instanz-Variablen (Public Instance Variables)*: Anzahl aller innerhalb einer Klasse als 'public' deklarierten Instanz-Variablen

Tabelle 16 – Fortsetzung Understand-Metriken

Metrik	Beschreibung
CDCIVPT	*Anzahl deklarierter geschützer Instanz-Variablen (Public Instance Variables)*: Anzahl der als 'protected' deklarierten Instanz-Variablen
CDCIVPTI	*Anzahl deklarierter geschützer, interner Instanz-Variablen (Protected Internal Instance Variables)*: Anzahl der als 'protected' deklarierten internen Instanz-Variablen
NOPRA	*Anzahl privater Instanz-Variablen (Private Instance Variables)*: Anzahl aller innerhalb einer Klasse als 'private' deklarierten Instanz-Variablen
WMC, SUMVG	*Gewichtete Methoden pro Klasse (Methods)*: Anazhl aller verschachtelten Funktionen und Methoden einer Klasse
NMI, NOMI	*Anzahl aller Methoden inklusive geerbter Methoden (All Methods)*: Anzahl aller Methoden einer Klasse inklusive aller geerbten Methoden
CDCMC	*Anzahl deklarierter konstanter Methoden (Const Methods)*: Anzahl der als konstant deklarierten, lokalen Methoden
CDCMD	*Anzahl deklarierter konstanter Methoden (Default Methods)*: Anzahl der deklarierten, lokalen Default-Methoden
CDCMF	*Anzahl deklarierter Freundesmethoden (Friend Methods)*: Anzahl der lokal deklarierten Friend-Methoden
CDCMI	*Anzahl deklarierter Freundesmethoden (Internal Methods)*: Anzahl der lokal deklarierten internen Methoden
NOPM, NPM, PM	*Anzahl öffentlicher Methoden (Public Methods)*: Anzahl aller innerhalb einer Klasse als 'public' deklarierten Methoden
CDCMPTI	*Anzahl deklarierter geschützter Methoden (Protected Methods)*: Anzahl der als 'protected' deklarierten, lokalen Methoden
CDCMPT	*Anzahl deklarierter geschützter internen Methoden (Protected Internal Methods)*: Anzahl der als 'protected' deklarierten, lokalen, internen Methoden
NOPRM	*Anzahl privater Methoden (Private Methods)*: Anzahl aller innerhalb einer Klasse als 'private' deklarierten Methoden
NOSPRM	*Anzahl strikt privater Methoden (Strict Private Methods)*: Anzahl aller innerhalb einer Klasse als strikt 'private' deklarierten Methoden
NOSPM	*Anzahl strikt veröffentlichten Methoden (Strict Published Methods)*: Anzahl aller innerhalb einer Klasse als strikt veröffentlichte deklarierten Methoden
NM	*Anzahl an Modulen (Modules)*: Anzahl aller Module eines Systems
PU	*Anzahl an Programmeinheiten (Program Unit)*: Anzahl aller nicht verschachtelten Modulen, Datenblöcke und Unterprogrammen

Tabelle 16 – Fortsetzung Understand-Metriken

Metrik	Beschreibung
PROP	*Anzahl an Eigenschaften (Properties)*: Anzahl aller Eigenschaften eines Systems
AIP	*Anzahl an automatisch implementierter Eigenschaften (Auto-Implemented Properties)*: Anzahl aller automatisch implementierter Eigenschaften eines Systems
SP	*Anzahl an Unterprogrammen (Subprograms)*: Anzahl aller Unterrogramme eines Systems
FANIN	*Eingehende Kopplung (Inputs)*: Anzahl aller durch Unterprogramme und eingelesenen, globalen Variablen, die eine Methode aufrufen
LOC, NLOC	*Anzahl Code-Zeilen (Lines)*: Anzahl aller physischen Code-Zeilen
LOC_HTML	*Anzahl Code-Zeilen (Lines HTML)*: Anzahl aller physischen HTML-Code-Zeilen
LOC_JS	*Anzahl Code-Zeilen (Lines JavaScript)*: Anzahl aller physischen JavaScript-Code-Zeilen
LOC_PHP	*Anzahl Code-Zeilen (Lines PHP)*: Anzahl aller physischen PHP-Code-Zeilen
BLOC	*Anzahl leerer Code-Zeilen (Blank Lines)*: Anzahl aller leeren Code-Zeilen
BLOC_HTML	*Anzahl leerer Code-Zeilen (Blank Lines HTML)*: Anzahl aller leeren HTML-Code-Zeilen
BLOC_JS	*Anzahl leerer Code-Zeilen (Blank Lines JavaScript)*: Anzahl aller leeren JavaScript-Code-Zeilen
BLOC_PHP	*Anzahl leerer Code-Zeilen (Blank Lines PHP)*: Anzahl aller leeren PHP-Code-Zeilen
BLOCI	*Anzahl leerer Code-Zeilen (Blank Lines includes inavtive)*: Anzahl aller leeren Code-Zeilen inklusive inaktiver Code-Zeilen
SLOC	*Anzahl Quellcode-Zeilen (Code Lines)*: Anzahl aller Zeilen des Quellcodes ausschließlich Kommentare
ELOC	*Anzahl ausführbarer Quellcode-Anweisungen (Executable Code Lines)*: Anzahl aller ausführbaren Zeilen des Quellcodes
CLCD	*Anzahl deklarativer Code-Zeilen (Declarative Code Lines)*: Anzahl aller deklarativen Zeilen des Quellcodes
SLOC_JS	*Anzahl Quellcode-Zeilen (Code Lines JavaScript)*: Anzahl aller Zeilen des JavaScript-Quellcodes ausschließlich Kommentare
SLOC_PHP	*Anzahl Quellcode-Zeilen (Code Lines PHP)*: Anzahl aller Zeilen des PHP-Quellcodes ausschließlich Kommentare
SLOCI	*Anzahl Quellcode-Zeilen (Code Lines includes inactive)*: Anzahl aller Zeilen des Quellcodes inklusive inaktiver Quellcode-Zeilen

Tabelle 16 – Fortsetzung Understand-Metriken

Metrik	Beschreibung
LCOMM, CCML	*Anzahl Kommentarzeilen (Comment Lines)*: Anzahl aller Kommentarzeilen des Quellcodes
LCOMM_HTML	*Anzahl Kommentarzeilen (Comment Lines HTML)*: Anzahl aller HTML-Kommentarzeilen des Quellcodes
LCOMM_JS	*Anzahl Kommentarzeilen (Comment Lines JavaScript)*: Anzahl aller JavaScript-Kommentarzeilen des Quellcodes
LCOMM_PHP	*Anzahl Kommentarzeilen (Comment Lines PHP)*: Anzahl aller PHP-Kommentarzeilen des Quellcodes
LCOMMI	*Anzahl Kommentarzeilen (Comment Lines includes inactive)*: Anzahl aller Kommentarzeilen des Quellcodes inklusive inaktiver Zeilen
CLI	*Anzahl inaktiver Zeilen (Inactive Lines)*: Anzahl inaktiver Code-Zeilen
CLP	*Anzahl Zeilen zur Vorverarbeitung (Preprocessor Lines)*: Anzahl an Code-Zeilen zur Vorverarbeitung des Quellcodes
FANOUT	*Ausgehende Kopplung (Outputs)*: Anzahl aller durch eine Methode aufgerufenen Unterprogramme und verwendeten globalen Variablen
PACK	*Anzahl importierter Pakete (Coupled Packages)*: Anzahl der durch eine Klasse oder das gesamte System importierten Pakete
NPATH	*Anzahl möglicher Pfade (Paths)*: Anzahl möglicher, eindeutiger Entscheidungspfade ohne abnormale Abbrüche oder Sprünge
NPATHL	*Anzahl möglicher Pfade logarithmiert (Paths Log x)*: Anzahl möglicher, eindeutiger Entscheidungspfade ohne abnormale Abbrüche oder Sprünge logarithmiert mit der Basis 10
CS	*Anzahl an Semikolons (Semicolons)*: Anzahl aller Semikolons
NST	*Anzahl an Statements (Statements)*: Anzahl aller Statements
DST	*Anzahl deklarativer Statements (Declarative Statements)*: Anzahl aller deklarativen Statements
DST_JS	*Anzahl deklarativer JavaScript-Statements (Declarative Statements JavaScript)*: Anzahl aller deklarativen JavaScript-Statements
DST_PHP	*Anzahl deklarativer PHP-Statements (Declarative Statements PHP)*: Anzahl aller deklarativen PHP-Statements
EST	*Anzahl leerer Statements (Empty Statements)*: Anzahl aller leeren Statements
XST	*Anzahl ausführbarer Statements (Executable Statements)*: Anzahl aller ausführbaren Statements

Tabelle 16 – Fortsetzung Understand-Metriken

Metrik	Beschreibung
XST_JS	*Anzahl ausführbarer JavaScript-Statements (Executable Statements JavaScript)*: Anzahl aller ausführbaren JavaScript-Statements
XST_PHP	*Anzahl ausführbarer PHP-Statements (Executable Statements PHP)*: Anzahl aller ausführbaren PHP-Statements
VG, CYCLO	*McCabe's zyklomatische Komplexität (Cyclomatic Complexity)*: Anzahl der unabhängigen Pfade durch eine Programmeinheit wie beispielsweise die Anzahl aller Entscheidungsanweisungen addiert mit 1
MVG	*Modifizierte zyklomatische Komplexität (Modified Cyclomatic Complexity)*: Identisch zu *VG* mit dem einzigen Unterschied, dass die Case-Blöcke einer Switch-Anweisung nicht einzeln gezählt werden, sondern die gesamte Switch-Anweisung als 1 gezählt wird
SVG, CCS	*Strikte zyklomatische Komplexität (Strict Cyclomatic Complexity)*: Identisch zu *VG* mit dem einzigen Unterschied, dass logische Operatoren wie *AND* und *OR* ebenfalls gezählt werden
SMVG	*Strikte modifizierte zyklomatische Komplexität (Strict Modified Cyclomatic Complexity)*: Berechnung zyklomatischer Komplexität unter den Bedingungen der Metriken *SVG* und *MVG*
EVG, CCE	*Essentielle zyklomatische Komplexität (Essential Complexity)*: Berechnung identisch zu dem Prinzip von *VG*, nachdem alle strukturierten Programmierprimitiven iterativ durch einzelne Anweisungen ersetzt wurden
SMEVG	*Strikte modifizierte essentielle zyklomatische Komplexität (Strict Modified Essential Complexity)*: Berechnung zyklomatischer Komplexität unter den Bedingungen der Metriken *SVG*, *MVG* und *EVG*
KNOTS	*Anzahl an Knoten (Number of Knots)*: Anzahl überlappender Sprünge im Kontrollfluss aufgrund von bedingten Entscheidungen und Schleifenwiederholungen
MAXVG	*Maximale zyklomatische Komplexität (Max Cyclomatic Complexity)*: Maximale zyklomatische Komplexität aller verschachtelten Funktionen und Methoden
MMVG	*Maximale modifizierte zyklomatische Komplexität (Max Modified Cyclomatic Complexity)*: Maximale modifizierte zyklomatische Komplexität aller verschachtelten Funktionen und Methoden
MSVG	*Maximale strikte zyklomatische Komplexität (Max Strict Cyclomatic Complexity)*: Maximale strikte zyklomatische Komplexität aller verschachtelten Funktionen und Methoden

Tabelle 16 – Fortsetzung Understand-Metriken

Metrik	Beschreibung
MSMVG	*Maximale strikte modifizierte zyklomatische Komplexität (Max Strict Modified Cyclomatic Complexity)*: Maximale strikte, modifizierte zyklomatische Komplexität aller verschachtelten Funktionen und Methoden
MEVG	*Maximale strikte zyklomatische Komplexität (Max Essential Complexity)*: Maximale essentielle zyklomatische Komplexität aller verschachtelten Funktionen und Methoden
MAXEK	*Maximale Anzahl essentieller Knoten (Max Essential Knots)*: Maximale Anzahl überlappender Sprünge im Kontrollfluss nach der Beseitigung von Konstrukten struktureller Programmierung
MSMEVG	*Maximale strikte modifizierte essentielle Komplexität (Max Strict Modified Cyclomatic Complexity)*: Maximale strikte, modifizierte, essentielle zyklomatische Komplexität aller verschachtelten Funktionen und Methoden
DIT	*Tiefe des Vererbungsbaums (Max Inheritance Tree)*: Die maximale Länge eines Pfades innerhalb eines Vererbungsbaumes beginnend bei einer Klasse bis hin zur Wurzel dieser Struktur
MN, MAXNEST	*Maximale Verschachtelungsebene (Max Nesting)*: Maximale Verschachtelungstiefe aller Kontrollstrukturen
MINEK	*Minimale Anzahl essentieller Knoten (MinEssentialKnots)*: Minimale Anzahl überlappender Sprünge im Kontrollfluss nach der Beseitigung von Konstrukten struktureller Programmierung
PLC	*Prozentualer Mangel an Kohäsion (Percent Lack of Cohesion)*: 100 Prozent minus die durchschnittliche Kohäsion von Paket-Entitäten
PLCM	*Prozentualer Mangel an Kohäsion (Percent Lack of Cohesion Modified)*: 100 Prozent minus die durchschnittliche Kohäsion von Paket-Entitäten ohne Berücksichtigung geerbter Methoden
RCC	*Verhältnis Kommentare zu Code (Comment To Code Ratio)*: Anzahl an Kommentarzeilen im Verhältnis zur Anzahl an der Quellcode-Zeilen
SUMVG	*Summe zyklomatischer Komplexität (Sum Cyclomatic Complexity)*: Summe der zyklomatischen Komplexität aller verschachtelten Funktionen und Methoden
SMVG	*Summe modifizierter zyklomatischer Komplexitäten (Sum Modified Cyclomatic Complexity)*: Summe der modifizierten zyklomatischen Komplexität aller verschachtelten Funktionen und Methoden

Tabelle 16 – Fortsetzung Understand-Metriken

Metrik	Beschreibung
SSVG	*Summe strikter zyklomatischer Komplexitäten (Sum Strict Cyclomatic Complexity)*: Summe der strikten zyklomatischen Komplexität aller verschachtelten Funktionen und Methoden
SSMVG	*Summe strikter modifizierter zyklomatischer Komplexitäten (Sum Strict Modified Cyclomatic Complexity)*: Summe der strikten, modifizierten zyklomatischen Komplexität aller verschachtelten Funktionen und Methoden
SEVG	*Summe essentieller zyklomatischer Komplexitäten (Sum Essential Complexity)*: Summe der essentiellen zyklomatischen Komplexität aller verschachtelten Funktionen und Methoden
SSMEVG	*Summe strikter modifizierter essentieller zyklomatischer Komplexitäten (Sum Strict Cyclomatic Complexity)*: Summe der strikten, modifizierten, essentiellen zyklomatischen Komplexität aller verschachtelten Funktionen und Methoden

Tabelle 17: Überblick der durch Understand bereitgestellten Software-Produktmetriken [8]

11.3.2 Anlage C.2 - Metriken des Extraktionswerkzeugs Analizo

Metrik	Beschreibung
COF, CF	*Kopplungsfaktor (Coupling Factor)*: Relative Anzahl aktueller Kopplungen im Verhältnis zur maximal möglichen Anzahl an Kopplungen auf Projekt-Ebene
LOC, NLOC	*Anzahl Code-Zeilen (Lines of Code)*: Anzahl aller Code-Zeilen auf Projekt-Ebene
NMAC	*Anzahl Methoden abstrakter Klassen (Number of methods per abstract class)*: Durchschnittliche Anzahl der Methoden pro abstrakter Klasse
NCLASS, NC	*Anzahl an Klassen (Number of Classes)*: Anzahl aller Klassen eines Systems
NMDA	*Anzahl an Modulen mit mindestens einem definierten Attribut (Number of Modules with at least one defined attribute)*: Anzahl an Modulen mit mindestens einem definierten Attribut
NMDM	*Anzahl an Modulen mit mindestens einer definierten Methode (Number of Modules with at least one defined method)*: Anzahl an Modulen mit mindestens einer definierten Methode
NOM, NM, MPC, NCM, TNM	*Anzahl an Klassen-Methoden (Number of Class Methods)*: Anzahl aller in Klassen implementieren Methoden eines Systems
CA, FANIN, FIN	*Eingehende Kopplung (Afferent-Coupling)*: Anzahl aller anderen Klassen mit Methodenaufruf der gegebenen Klasse

Tabelle 17 – Fortsetzung Analizo-Metriken

Metrik	Beschreibung
AVG	*Durchschnittliche zyklomatische Komplexität (Average Cyclomatic Complexity)*: Durchschnittliche zyklomatische Komplexität aller verschachtelten Funktionen und Methoden
AMLOC	*Durchschnittliche Anzahl Code-Zeilen einer Methode (Average Method Lines of Code)*: Durchschnittliche Anzahl an Code-Zeilen aller Methoden
ANPAR, APAR	*Durchschnittliche Anzahl an Parameter (Average Parameters)*: Durchschnittliche Anzahl an Parametern innerhalb der Deklaration aller Methoden
CBO	*Kopplung zwischen Objekten (Coupling Between Objects)*: Anzahl aller mit einer Klasse durch Methodenaufrufe, Feldzugriffe, Vererbung, Argumenten, Rückgabewerten und Ausnahmebehandlungen gekoppelten Klassen
DIT	*Tiefe des Vererbungsbaums (Depth of Inheritance Tree)*: Die maximale Länge eines Pfades innerhalb eines Vererbungsbaumes beginnend bei einer Klasse bis hin zur Wurzel dieser Struktur
LCOM1	*Mangel an Kohäsion in Methoden 1 (Lack of Cohesion in Methods 1)*: Anzahl der Methodenpaare einer Klasse, die sich keine Instanz-Variablen teilen
LOC, NLOC	*Anzahl Code-Zeilen (Lines of Code)*: Anzahl aller Code-Zeilen eines Moduls
NOA, NF	*Anzahl an Attributen (Number of Attributes)*: Anzahl aller Attribute einer Klasse
NSC	*Anzahl an Kindern (Number of Children)*: Anzahl aller direkten Unterklassen einer Klasse
NOM, NM, MPC, NCM, TNM	*Anzahl an Klassen-Methoden (Number of Class Methods)*: Anzahl aller in einer Klasse implementierten Methoden
NOPM, NPM, PM	*Anzahl öffentlicher Methoden (Number of Public Methods)*: Anzahl aller innerhalb einer Klasse als 'public' deklarierten Methoden
RFC	*Antworten für eine Klasse (Response for a class)*Anzahl der durch eine gegebene Klasse unterschiedliche, direkt aufgerufene Methoden

Tabelle 18: Überblick der durch Analizo bereitgestellten Software-Produktmetriken [8]

11.4 Anlage D - Analyse extrahierter Metriken-Werte

11.4.1 Anlage D.1 - Analyse extrahierter Understand-Metriken

Metrik	Fehlende Werte	Identische Werte	Abweichung
EU	7.728 (33,15%)	12.984 (55,69%)	3,2015384615384614
AVG	3 (0,01%)	21.637 (92,81%)	1,9413875598086126
MMVG	3 (0,01%)	20.710 (88,83%)	2,0711812235475184
DST_JS	10.999 (47,18%)	11001 (47,19%)	4,3948170731707314
CS	13.842 (59,38%)	3.862 (16,57%)	5,559379457917261
EST	15.587 (66,86%)	7.498 (32,16%)	1,920704845814978
BLOC_JS	10.999 (47,18%)	11.162 (47,88%)	5,233709817549957
BLOC	0 (0%)	15.839 (67,94%)	3,816138097149739
MEVG	3 (0,01%)	22.324 (95,76%)	2,616243654822335
LCOMMI	15.587 (66,86%)	5803 (24,89%)	4,633194588969823
NST	0 (0%)	13.116 (56,26%)	8,383679874460572
DST	0 (0%)	17.308 (74,25%)	3,756495669553631
MAXVG	3 (0,01%)	20.704 (88,81%)	2,147792706333973
DST_PHP	10.999 (47,18%)	11316 (48,54%)	2,497492477432297
AEVG	51 (0,22%)	22.782 (97,72%)	1,569937369519833
RCC	0 (0%)	9.875 (42,36%)	0,050350876943067654
NCLASS	8 (0,03%)	23.107 (99,12%)	1,5532994923857868
LCOMM_JS	10.999 (47,18%)	11.576 (49,67%)	9,85753052917232
NPATHL	9.526 (40,86%)	13.523 (58,01%)	1,1482889733840305
MSMVG	3 (0,01%)	20.404 (87,53%)	2,532185886402754
PU	23.307 (99,97%)	5 (0,02%)	0
NPATH	9.526 (40,86%)	13.101 (56,19%)	52156539,81751825
LOC	0 (0%)	18.520 (79,44%)	15,095111016225449
LCOMM_HTML	10.999 (47,18%)	12.290 (52,72%)	7,608695652173913
ALOCI	15.587 (66,86%)	6.438 (27,62%)	2,7832167832167833
CLP	15.325 (65,73%)	7.210 (30,93%)	2,7734877734877736
SUMMVG	3 (0,01%)	16.693 (71,61%)	4,024032648125756
LOC_PHP	10.999 (47,18%)	7.887 (33,83%)	20,07749661093538
CLCD	12.313 (52,81%)	7.055 (30,26%)	4,564401622718052
SUMVG	3 (0,01%)	16.668 (71,49%)	4,126486974853185
NOPRM	21.829 (93,64%)	1.352 (5,79%)	1,5954198473282444
ASLOCI	15.587 (66,86%)	2.415 (10,36%)	8,716007532956686
XST	0 (0%)	14.534 (62,35%)	8,02153110047847
SMVG	9.526 (40,86%)	12.871 (55,21%)	2,890710382513661
SVG	9.523 (40,85%)	12.869 (55,2%)	2,860869565217391
XST_PHP	10.999 (47,18%)	9.103 (39,05%)	7,417133956386293
LOC_JS	10.999 (47,18%)	10.221 (43,84%)	22,485181644359464
SLOC_PHP	10.999 (47,18%)	8.414 (36,09%)	17,810464221595282

Tabelle 18 – Fortsetzung der Metriken-Analyse

Metrik	Fehlende Werte	Identische Werte	Abweichung
CDCMPTI	21.829 (93,64%)	1.441 (6,18%)	1,2619047619047619
BLOC_HTML	10.999 (47,18%)	12.260 (52,59%)	3,1320754716981134
VG	9.523 (40,85%)	12.956 (55,58%)	2,146458583433373
SLOC_JS	10.999 (47,18%)	10.305 (44,2%)	18,302290836653388
BLOCI	15.587 (66,86%)	5.294 (22,71%)	2,8663101604278074
SEVG	3 (0,01%)	19.044 (81,69%)	3,628604923798359
LCOMM_PHP	10.999 (47,18%)	10.164 (43,59%)	7,876686831084225
LCOMM	0 (0%)	17.620 (75,58%)	6,791286015460295
ASVG	3 (0,01%)	21.362 (91,64%)	2,310734463276836
MSVG	3 (0,01%)	20.402 (87,52%)	2,564499484004128
NIV	21.829 (93,64%)	1.369 (5,87%)	1,6403508771929824
WMC	21.829 (93,64%)	1.101 (4,72%)	2,392670157068063
ABLOCI	15.587 (66,86%)	7.415 (31,81%)	1,6
NOPM	21.829 (93,64%)	1.243 (5,33%)	2,970833333333333
SSMVG	3 (0,01%)	16.282 (69,84%)	4,749537498221147
ASLOC	3 (0,01%)	18.776 (80,54%)	5,117582175159939
SSVG	3 (0,01%)	16.263 (69,76%)	4,8322452455293785
MVG	9.523 (40,85%)	12.962 (55,6%)	2,083434099153567
ASMVG	3 (0,01%)	21.369 (91,67%)	2,297422680412371
LOC_HTML	10.999 (47,18%)	12.118 (51,98%)	8,615384615384615
LOC	0 (0%)	18.520 (79,44%)	15,095111016225449
MN	3 (0,01%)	22.489 (96,47%)	1,2975609756097561
NOM	21.829 (93,64%)	1.389 (5,96%)	1,8191489361702127
ABLOC	3 (0,01%)	21.886 (93,88%)	2,668306394940267
AMVG	3 (0,01%)	21.670 (92,96%)	1,9017693715680293
BLOC_PHP	10.999 (47,18%)	9.882 (42,39%)	4,106129164952694
XST_JS	10.999 (47,18%)	10.621 (45,56%)	12,958628841607565
NOF	53 (0,23%)	20.452 (87,73%)	3,043462771642323
ALCOMM	3 (0,01%)	22.341 (95,83%)	2,8863636363636362
EVG	9.526 (40,86%)	13.678 (58,67%)	2,7037037037037037
CDCMD	21.829 (93,64%)	1.417 (6,08%)	2,8636363636363638
NIM	21.829 (93,64%)	1.165 (4,99%)	2,393081761006289
ALOC	3 (0,01%)	18.223 (78,17%)	5,697208022021234
CLI	15.277 (65,53%)	7.381 (31,66%)	11,813455657492355
NOA	21.829 (93,64%)	1.343 (5,76%)	1,7428571428571429
ELOC	12.313 (52,81%)	6.726 (28,85%)	12,011233325532412

Tabelle 19: Analyse extrakhierter Understand-Metriken

11.4.2 Anlage D.2 - Analyse extrahierter Analizo-Metriken

Metrik	Identische Werte	Abweichung
AVG	5445 (68,11%)	0,30036277277029005
AMLOC	3923 (49,07%)	1,158921891403617
ANPAR	7213 (90,23%)	0,1587587750617573
DIT	7992 (99.97%)	1,0
LCOM1	7643 (95,61%)	1,5128205128205128
LOC	3906 (48,86%)	11,88013698630137
MLOC	6428 (80,41%)	8,43933588761175
NOA	7571 (94,71%)	1,9858156028368794
NOM	7289 (91,18%)	1,8056737588652483
NOPA	7731 (96,71%)	2,273764258555133
NOPM	7386 (92,39%)	1,9407894736842106
RFC	7026 (87,89%)	4,112603305785124

Tabelle 20: Analyse extrakhierter Analizo-Metriken

11.5 Anlage E - Ergebnisse der Korrelationsanalysen

11.5.1 Anlage E.1 - Korrelationsanalyse der Understand-Metriken

Metrik	Pearson	Spearman	Kendall
Batchfile	-0.0018 / -0.0005	-0.0018 / -0.0003	-0.0018 / -0.0003
C	0.0207 / 0.0302	0.0207 / 0.0432	0.0207 / 0.039
C#	0.0011 / 0.0097	0.0011 / 0.0117	0.0011 / 0.0106
C++	0.0011, / 0.0266	0.0114 / 0.0324	0.0114 / 0.0295
CSS	-0.0016, / -0.0163	-0.0016 / -0.0158	-0.0016 / -0.0143
CofeeScript	-0.0049, / -0.0005	-0.0049 / -0.0007	-0.0049 / -0.0006
Erlang	0.0003 / 0.002	0.0003 / 0.002	0.0003 / 0.0018
Go	-0.0006 / 0.003	-0.0006 / 0.0038	-0.0006 / 0.0035
HTML	-0.0091 / -0.0023	-0.0091 / -0.0002	-0.0091 / -0.0002
Haskell	0.0008 / 0.0025	0.0008 / 0.0034	0.0008 / 0.0031
Java	0.0001 / 0.0312	0.0001 / 0.0309	0.0001 / 0.028
JavaScript	0.0032 / -0.006	0.0032 / -0.0081	0.0032 / -0.0073
Jupyter	0.0004, / 0.0014	0.0004 / 0.0014	0.0004 / 0.0013
Lua	0.0007 / 0.0045	0.0007 / 0.005	0.0007 / 0.0046
Markdown	-0.0025 / -0.0011	-0.0025 / -0.0026	-0.0025 / -0.0023
Matlab	0.0062 / -0.0043	-0.0062 / -0.0032	-0.0062 / -0.0029
None	-0.006 / -0.0056	-0.006 / -0.0056	-0.006 / -0.005
Objective-C	0.0034 / 0.0121	0.0034 / 0.0156	0.0034 / 0.0142
PHP	-0.0672 / -0.1134	-0.0672 / -0.1287	-0.0672 / -0.117
Perl	-0.0036 / 0.0005	-0.0036 / 0.0009	-0.0036 / 0.0008
PowerShell	-0.0017 / -0.0156	-0.0017 / -0.0158	-0.0017 / -0.014
Python	0.0043 / 0.0227	0.0043 / 0.0257	0.0043 / 0.0233
R	0.0004 / 0.0022	0.0004 / 0.003	0.0004 / 0.0027
Ruby	0.0029 / 0.0108	0.0026 / 0.0154	0.0026 / 0.014
Rust	0.0008 / 0.0024	0.0008 / 0.0026	0.0008 / 0.0024
SQL	-0.0032 / 0.0017	-0.0032 / 0.0018	-0.0032 / 0.0016
Scala	0.0013 / 0.0056	0.0013 / 0.0054	0.0013 / 0.0049
Shell	0.0009 / 0.0055	0.0009 / 0.0071	0.0009 / 0.0064
Swift	-0.0007 / 0.002	-0.0007 / 0.0025	-0.0007 / 0.0022
TeX	-0.0003 / 0.0005	-0.0003 / 0.00001	-0.0003 / 0.00001
TypeScript	-0.0025 / 0.0139	-0.0025 / 0.0146	-0.0025 / 0.0133
unknown	0.16 / 0.1701	0.156 / 0.1712	0.156 / 0.1556
EU	0.0054 / 0.0096	0.0228 / 0.0582	0.0193 / 0.0449
AVG	0.0029 / 0.0104	0.001 / 0.0128	0.0009 / 0.0101
MMVG	0.0076 / 0.023	0.0141 / 0.0346	0.0118 / 0.0263
DST_JS	0.0065 / 0.0136	0.0317 / 0.0618	0.0299 / 0.0538
CS	0.0031 / -0.0017	0.0062 / -0.0182	0.005 / -0.0134
EST	0.0004 / -0.0024	0.0012 / -0.0301	0.001 / -0.0243

Tabelle 20 – Fortsetzung der Korrelationsanalyse

Metrik	Pearson	Spearman	Kendall
BLOC_JS	0.0066 / 0.0148	0.0309 / 0.0605	0.0293 / 0.0534
BLOC	0.0119 / 0.0252	0.0213 / 0.0822	0.0175 / 0.0599
MEVG	0.0049 / 0.0145	0.0046 / 0.019	0.0041 / 0.0152
LCOMMI	-0.0006 / 0.0039	-0.0016 / -0.0019	-0.0014 / -0.0016
NST	0.005 / 0.009	0.0248 / 0.0588	0.0203 / 0.0434
DST	0.0091 / 0.0128	0.0205 / 0.0641	0.017 / 0.0471
MAXVG	0.007 / 0.0225	0.0139 / 0.0344	0.0117 / 0.0262
DST_PHP	-0.0018 / 0.0288	-0.0347 / -0.0428	-0.0305 / -0.0366
AEVG	-0.0077 / -0.0035	-0.0106 / -0.0104	-0.01 / -0.0092
RCC	-0.0013 / -0.0114	-0.0126 / -0.0398	-0.0103 / -0.0291
NCLASS	0.0035 / 0.0126	-0.0183 / -0.018	-0.0173 / -0.016
LCOMM_JS	0.0056 / 0.0118	0.0267 / 0.0488	0.0253 / 0.0428
NPATHL	0.0088 / 0.0262	0.0166 / 0.0401	0.0161 / 0.0355
MSMVG	0.0068 / 0.0195	0.0131 / 0.0318	0.011 / 0.0242
PU	0 / -0.1195	0.0 / -0.1925	0.0 / -0.1857
NPATH	0.0039 / 0.0144	0.0437 / 0.077	0.0405 / 0.0655
LOC	0.014 / 0.0325	0.0284 / 0.0707	0.0232 / 0.0517
LCOMM_HTML	-0.0106 / -0.0052	-0.0143 / -0.0026	-0.0142 / -0.0024
ALOCI	-0.001 / 0.0036	-0.003 / -0.00005	-0.0025 / -0.0003
CLP	0.00005 / 0.0156	0.0006 / 0.001	0.0005 / 0.0009
SUMMVG	0.005 / 0.0091	0.0211 / 0.0466	0.0175 / 0.0347
LOC_PHP	0.0092 / 0.0377	-0.0111 / -0.0116	-0.0094 / -0.01
CLCD	0.0017 / 0.0036	0.0057 / -0.0244	0.0046 / -0.018
SUMVG	0.005 / 0.0092	0.021 / 0.0464	0.0173 / 0.0346
NOPRM	0.0098 / -0.0068	0.0131 / -0.0211	0.0118 / -0.0174
ASLOCI	0.00004 / 0.0066	-0.0007 / -0.0059	-0.0006 / -0.0045
XST	0.0043 / 0.0082	0.0201 / 0.0399	0.0165 / 0.0296
SMVG	0.0091 / 0.0233	0.0442 / 0.0789	0.0409 / 0.0669
SVG	0.0092 / 0.023	0.0443 / 0.0786	0.0409 / 0.0667
XST_PHP	0.0096 / 0.0429	-0.009 / -0.0085	-0.0077 / -0.0079
LOC_JS	0.0093 / 0.0242	0.0298 / 0.0631	0.028 / 0.0548
SLOC_PHP	0.0103 / 0.0358	-0.0105 / -0.0116	-0.0089 / -0.0102
CDCMPTI	0.0095 / -0.0003	0.009 / -0.0152	0.0086 / -0.013
BLOC_HTML	0.0002 / 0.0066	-0.0075 / 0.0013	-0.0075 / 0.00123
VG	0.0096 / 0.0245	0.0442 / 0.0785	0.0409 / 0.0667
SLOC_JS	0.0094 / 0.0254	0.0298 / 0.0632	0.028 / 0.0549
BLOCI	0.0001 / -0.0038	-0.0003 / -0.0196	-0.0003 / -0.0145
SEVG	0.0048 / 0.0076	0.0165 / 0.0385	0.0138 / 0.0289
LCOMM_PHP	0.0039 / 0.0307	-0.0122 / -0.022	-0.0104 / -0.0182
LCOMM	0.0097 / 0.0261	0.0197 / 0.0458	0.0162 / 0.0336
ASVG	0.0025 / 0.009	0.00003 / 0.0104	0.00002 / 0.0082
MSVG	0.0065 / 0.0198	0.0131 / 0.0319	0.011 / 0.0242

Tabelle 20 – Fortsetzung der Korrelationsanalyse

Metrik	Pearson	Spearman	Kendall
NIV	0.0104 / -0.0057	0.0064 / -0.0157	0.0056 / -0.0127
WMC	0.0115 / -0.0127	0.0198 / -0.0194	0.0165 / -0.0144
ABLOCI	-0.0003 / 0.0014	-0.0025 / -0.021	-0.0022 / -0.0167
NOPM	0.0109 / -0.0118	0.0218 / -0.0047	0.018 / -0.0036
SSMVG	0.0042 / 0.0074	0.0203 / 0.0448	0.0168 / 0.0334
ASLOC	0.0033 / 0.0044	0.0052 / 0.0196	0.0043 / 0.0144
SSVG	0.0042 / 0.0075	0.0202 / 0.0447	0.0167 / 0.0333
MVG	0.0094 / 0.0249	0.0442 / 0.0787	0.0409 / 0.0669
ASMVG	0.0025 / 0.0089	0.0003 / 0.0107	0.0003 / 0.0084
LOC_HTML	0.0012 / 0.0084	0.0021 / 0.0185	0.002 / 0.017
LOC	0.015 / 0.0351	0.0306 / 0.0777	0.025 / 0.0572
MN	0.013 / 0.0331	0.0144 / 0.0355	0.0127 / 0.0282
NOM	0.004 / -0.0168	0.0013 / -0.0163	0.0012 / -0.0138
ABLOC	0.0024 / 0.0045	0.0084 / 0.0477	0.0076 / 0.0389
AMVG	0.0028 / 0.0103	0.001 / 0.013	0.0009 / 0.0103
BLOC_PHP	0.006 / 0.0389	-0.0225 / -0.0127	-0.019 / -0.0124
XST_JS	0.0051 / 0.0096	0.031 / 0.0628	0.0292 / 0.0544
NOF	0.0042 / 0.0062	0.0175 / 0.0335	0.0147 / 0.0254
ALCOMM	0.0013 / 0.0029	0.0028 / 0.0507	0.0026 / 0.0426
EVG	0.0092 / 0.0176	0.0472 / 0.0781	0.0466 / 0.0709
CDCMD	-0.0029 / -0.0129	-0.0073 / -0.0408	-0.0071 / -0.0355
NIM	0.0111 / -0.0095	0.0174 / -0.0188	0.0145 / -0.0138
ALOC	0.0032 / -0.0048	0.0053 / 0.0257	0.0045 / 0.0189
CLI	0.0006 / 0.0093	0.0029 / 0.0065	0.0025 / 0.0051
NOA	0.0027 / 0.003	-0.006 / -0.0488	-0.0054 / -0.04
ELOC	0.0021 / 0.0009	0.0024 / -0.0209	0.002 / -0.0156

Tabelle 21: Korrelationsanalyse der Understand-Metriken

11.5.2 Anlage E.2 - Korrelationsanalyse der Analizo-Metriken

Metrik	Pearson	Spearman	Kendall
Batchfile	0.00016 / -0.0008	0.00016 / -0.0006	0.00016 / -0.00054
C	0.0025 / -0.0009	0.0025 / -0.0001	0.0025 / -0.0001
C#	-0.002 / -0.0057	-0.002 / -0.0077	-0.002 / -0.007
C++	0.0006, / -0.014	0.0006 / -0.021	0.0006 / -0.0188
Erlang	0.00007 / 0.0019	0.00007 / 0.0026	0.00007 / 0.0023
Go	0.00019 / -0.0014	0.00019 / -0.0014	0.00019 / -0.0013
HTML	0.00007 / -0.0031	0.00007 / -0.0044	0.00007 / -0.004
Haskell	0.00012 / -0.0016	0.00012 / -0.0025	0.00012 / -0.0022
Java	-0.0163 / 0.0065	-0.0163 / 0.0113	-0.0163 / 0.0102
JavaScript	0.00029 / -0.0018	0.00029 / -0.0025	0.00029 / -0.0023

Tabelle 21 – Fortsetzung der Korrelationsanalyse

Metrik	Pearson	Spearman	Kendall
Lua	-0.0017 / -0.0029	-0.0017 / -0.002	-0.0017 / -0.0018
Markdown	-0.0028 / -0.0068	-0.0028 / -0.008	-0.0028 / -0.0072
Matlab	0.0002 / -0.0027	0.0002 / -0.0025	0.0002 / -0.0023
Objective-C	0.0011, / -0.0079	0.0011 / -0.0104	0.0011 / -0.0093
PHP	0.0001 / 0.0004	0.0001 / 0.0002	0.0001 / 0.0001
Perl	-0.0044 / 0.0043	-0.0044 / -0.0044	-0.0044 / -0.004
PowerShell	0.00007 / 0.00003	0.00007 / -0.0003	0.00007 / -0.0003
Python	0.0001 / -0.0029	0.0001 / -0.0042	0.0001 / -0.0038
R	0.0001 / 0.00007	0.0001 / -0.0007	0.0001 / -0.0006
Ruby	0.0004 / 0.0003	0.0004 / 0.0082	0.0004 / 0.0074
Rust	0.0003 / 0.00003	0.0003 / -0.0002	0.0003 / -0.0002
SQL	0.0003 / 0.008	0.0003 / 0.0101	0.0003 / 0.0091
Scala	0.0001 / 0.0014	0.0001 / 0.0013	0.0001 / 0.0012
Shell	-0.0005 / -0.0002	-0.0005 / 0.0013	-0.0005 / 0.0012
Swift	-0.0025 / -0.0042	-0.0025 / -0.0047	-0.0025 / -0.0042
TeX	0.0001 / 0.0006	0.0001 / 0.0021	0.0001 / 0.0019
TypeScript	-0.0028 / -0.0038	-0.0028 / -0.0048	-0.0028 / -0.0043
unknown	0.0695 / 0.08	0.0695 / 0.0804	0.0695 / 0.0724
AVG	0.0005 / -0.0032	0.0017 / -0.0058	0.0014 / -0.0042
AMLOC	0.0004 / 0.0004	0.0004 / -0.0026	0.0003 / -0.0017
ANPAR	0.0052 / 0.0017	0.0057 / -0.0025	0.0048 / -0.0017
DIT	-0.0095 / 0.0087	-0.0095 / 0.0121	-0.0095 / 0.0109
LCOM1	0.0008 / -0.0059	0.0006 / -0.0251	0.0005 / -0.0193
LOC	0.0023 / -0.0117	0.0031 / -0.0171	0.0026 / -0.0126
MLOC	0.0015 / -0.0022	0.0023 / -0.0079	0.0019 / -0.0056
NOA	0.001 / -0.001	0.0021 / 0.002	0.0018 / 0.0015
NOM	0.002 / -0.0158	0.0031 / -0.02	0.0026 / -0.015
NOPA	0.0013 / -0.0011	0.0057 / -0.0002	0.0049 / -0.0001
NOPM	0.0023 / -0.016	0.0044 / -0.0201	0.0036 / -0.0151
RFC	0.0019 / -0.0121	0.0031 / -0.0157	0.0026 / -0.0116

Tabelle 22: Korrelationsanalyse der Analizo-Metriken

Selbständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Seitens des Verfassers bestehen keine Einwände die vorliegende Hausarbeit für die öffentliche Benutzung im Universitätsarchiv zur Verfügung zu stellen.

Jena, den 29. August 2023