Hochschule für Telekommunikation Leipzig
University of Applied Sciences

Hochschule für Telekommunikation Leipzig (FH)
Institut für Telekommunikationsinformatik

Abschlussarbeit zur Erlangung des akademischen Grades Bachelor of Science

Thema: Generische Verfügbarkeitsanalyse anhand von Logdatei-Anomalieerkennung

Vorgelegt von: Tim Pütz
Geboren am: 30.05.1997
Geboren in: Bonn

Vorgelegt am: 4. November 2020

Erstprüfer: Prof. Dr. Thomas Möbert

Hochschule für Telekommunikation Leipzig
Gustav-Freytag-Straße 43-45
04277 Leipzig

Zweitprüfer: Prof. Dr. techn. habil. Slavisa Aleksic

Hochschule für Telekommunikation Leipzig
Gustav-Freytag-Straße 43-45
04277 Leipzig

Vorwort

Die hier vorliegende Bachelorarbeit *Generische Verfügbarkeitsanalyse anhand von Logdatei-Anomalieerkennung* habe ich als Abschluss meines Studiums der angewandten Informatik an der der Hochschule für Telekommunikation Leipzig verfasst und während der Praxisphase meines dualen Studiums bei der Deutschen Telekom geschrieben. Das Team, in welchem ich eingesetzt bin, arbeitet für die Deutsche Telekom IT im Bereich Monitoring. Das Ziel war es, unser eigens entwickeltes Tool LightBSM zu verbessern, denn zum Zeitpunkt der Verfassung gab es immer wieder Ausfälle dieses Tools. Da diese Ausfälle in verschiedenen Logdateien erfassbar sind, brachte mich mein Business-Experte und guter Freund Silvio Kresse auf die Idee, die Verfügbarkeit einer Anwendung anhand von zusammenhängenden Logeinträgen zu ermitteln. Da in unserem Bereich mehrere Logdatei-Typen vorhanden sind, soll dieses Verfahren so generisch sein wie möglich.

Bei der Suche nach geeigneten Logdateien zum Testen half mir besonders Silvio Kresse. Daher möchte ich mich bei ihm für seine fachtechnische Unterstützung bedanken. Ich möchte ebenfalls meinen Prüfern Prof. Dr. Thomas Möbert und Prof. Dr. techn. habil. Slavisa Aleksic danken. Prof. Dr. techn. habil. Slavisa Aleksic sagte mir kurzfristig als Zweitprüfer zu, obwohl er schon viele andere Studenten zu betreuen hatte. Vielen Dank für Ihre Bereitschaft! Bei Prof. Dr. Thomas Möbert möchte ich mich für die Unterstützung bei allen Fragen rund um das Thema Bachelorarbeit bedanken. Egal zu welchen Fragen, er hatte immer eine passende Antwort.

Aufgrund meiner Probleme mit Rechtschreibung möchte ich mich ebenfalls bei meiner Familie und meinen Freunden für die Korrekturlesungen und die seelische Unterstützung bedanken! Ohne sie alle wäre ich nicht so weit gekommen, wie ich es heute bin.

Ich wünsche Ihnen viel Freude beim Lesen dieser Bachelorarbeit.

Inhaltsverzeichnis

Abkürzungsverzeichnis

Abb Abbildung 4, 5, 18, 22

bzw Beziehungsweise 8, 9, 11, 17, 35, 39

IDC International Data Corporation 9

IT Informationstechnik 1

MCC Matthews Correlation Coefficient 14, 15, 31, 34, 35

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Problemstellung und Motivation

57 Adapter sammeln, aggregieren und senden alle fünf Minuten Metrik-Daten über 235 Services der Deutschen Telekom IT an eine Applikation. Die Applikation basiert auf JHipster und trägt den Namen LightBSM. Aus diesen Daten entsteht eine Baumstruktur, welche mehr als 19000 Knoten beinhaltet. Die Blattanzahl beträgt dabei ca. 15000. Dieser Baum zeigt den Status der Services und deren Transaktionen an. Die Status entstehen aus täglich über 15000 Events, welche wiederum aus Schwellwertüberschreitungen entstehen. Dafür werden knapp 5,8 Millionen Metrikpunkte auf diese Schwellwertüberschreitungen geprüft. Die Schwellwerte werden dafür in einem Regelwerk mit aktuell 168 Regeln festgelegt. Neben den Daten zu Anwendungen und Services kommen noch Daten zur Infrastruktur und Hardware, wie Server, Switche, Firewalls, Loadbalancer etc. dazu. [1]

All diese Daten werden unter anderem in Dashboards angezeigt. Die Dashboards zeigen die Relationen einer oder mehrerer Anwendungen, Services und Hardware in sich und zueinander. Unterschieden wird z.B in Prozessketten-, Helikopter- und Business-Value-Dashboards. Das Resultat ist eine Anwendung, welche einen Großteil der Informationstechnik (IT) der Deutschen Telekom überwacht. Diese Anwendung und alle ihre Mikroservices erstellen Logdaten, um den Programmablauf zu protokollieren.

Leider kommt es aktuell zu unregelmäßigen Ausfällen der Hauptanwendung LightBSM, womit die Überwachung der IT nicht möglich ist. Dies ist besonders in Fehlerfällen, welche die Kunden der Deutschen Telekom betreffen, kritisch.

1.2 Ziele und Grenzen

Die Ursachen der Störungen sind unterschiedlich und nicht selten externen Einflüssen geschuldet. Ein Beispiel ist der Ausfall des zentralen Anmeldeservers. Dadurch läuft der Prozess von LightBSM noch, doch es werden zu viele Sitzungen beim Anmeldeversuch geöffnet, sodass die Kunden das Webportal von LightBSM nicht mehr erreichen können. Ob LightBSM ausfällt, wird bei genauerem Betrachten der Logdatei deutlich. Es erscheint eine beliebige Fehlermeldung und danach ist die Anwendung für Nutzer nicht mehr erreichbar, obwohl sie weiter Logeinträge erstellt.

[1] Die Daten stammen aus einer Bestandsaufnahme vom 01.10.2020

Um eine Frühwarnung vor einem solchen Ausfall der Verfügbarkeit zu ermöglichen, soll eine Methode entwickelt werden, die die Logdatei auf Anzeichen eines Ausfalls analysiert. Um zukünftig auch Logdateien anderer Anwendungen in der Deutschen Telekom IT auf einen Ausfall der Verfügbarkeit hin zu analysieren, soll die Methode so anwendungsunabhängig sein wie möglich. Dadurch ergibt sich eine *generische Verfügbarkeitsanalyse*.

Aufgrund dieser Flexibilität wird in dieser Arbeit eine Methode vorgestellt und getestet, die weder den Kontext der Anwendung noch das Format der Logdatei benötigt. Diese Methode soll Anomalien finden, welche den Ausfall einer abhängigen Anwendung zufolge haben. Das Ziel dieser Arbeit ist es, diese Methode auf Realisierbarkeit und Sinnhaftigkeit zu untersuchen und dabei die Architektur so leichtgewichtig wie möglich zu halten. Aus diesem Grund wird die Methode keine Verfahren des Maschinellen Lernens oder der künstlichen Intelligenz nutzen. Sie soll durch wenig Konfiguration einen möglichst hohen Nutzen erzielen. Genau dies stellt die Herausforderung in dieser Arbeit dar.

Das vorgestellte Verfahren soll in dem getesteten Umfeld über 90% aller Ausfälle erkennen. Über 80% aller vorhergesagten Ausfälle sollen auch wirklich Ausfälle gewesen sein. Somit ergibt sich eine maximale Fehlerrate von 20%. Bei einer Fehlerrate von über 20% würden die verantwortlichen Mitarbeiter die gemeldeten Ausfälle nicht ernst nehmen.

Das Ziel ist es nicht, aktuelle Methoden der Anomalie-Erkennung miteinander und mit der Methode dieser Arbeit zu vergleichen. Ein Vergleich mit anderen Methoden der Anomalie-Erkennung wäre in einer weiteren Forschungsarbeit denkbar. Außerdem ist es nicht das Ziel, ein voll funktionsfähiges Programm zur Analyse der Logdateien zu implementieren. Es werden lediglich Prototypen zum Testen implementiert.

1.3 Aufbau

Zuerst werden die Grundbegriffe und Grundlagen der Anomalie-Erkennung erläutert. Dazu zählen kurze Beschreibungen aktueller Verfahren zur Anomalie-Erkennung. Nachdem ein Bild über das Thema und die benötigten Fakten geschaffen wurde, wird die vorgestellte Methode untersucht. Dazu wird das Konzept vorgestellt, welches die Idee und die Umsetzung des Verfahrens aufzeigt. Anschließend werden die Testumgebung sowie die Testziele, der Testaufbau, die Testdaten und die Testhindernisse beschrieben. Nach der Durchführung des Tests wird das Ergebnis ausgewertet und evaluiert. Abschließend wird in der Schlussbetrachtung der erforschte Inhalt zusammengefasst. Ein Fazit über die untersuchte Realisierbarkeit und Sinnhaftigkeit der Methode wird gezogen und der Ausblick gibt Auskunft über zukünftig mögliche Untersuchungen.

2 Grundlagen der Anomalie- und Ausreißer-Erkennung

In diesem Kapitel werden die notwendigen Grundlagen, die für das Verständnis der gesamten Arbeit benötigt werden, erläutert.

2.1 Anomalien und Ausreißer

Seit über 50 Jahren wird eine einheitliche Definition für Ausreißer (eng. outlier) gesucht. 1969 schrieb Grubbs: „an outlier is one that appears to deviate markedly from the other members of the sample in which it occurs“ [1]. Seitdem definieren viele Autoren Ausreißer als eine Abweichung vom Normverhalten. Die am meisten verbreiteten Definitionen sind die von Hawkings aus 1980: „an outlier is an observation, which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism“ [2] und die von Barnett und Lewis aus 1994: „an outlier is an observation (or subset of observations) which appears to be inconsistent with the remainder of that set of data“ [3].

Auch für Anomalien gibt es keine einheitliche Definition. Die vorhandenen Definitionen verschiedener Autoren deuten darauf hin, dass es keinen nennenswerten Unterschied zwischen Ausreißern und Anomalien gibt. [1, 4, 5]

Deshalb wird für die Begriffe Ausreißer und Anomalie in dieser Arbeit nur noch der Begriff Anomalie genutzt. Für diese Arbeit wird folglich definiert:

> Eine Anomalie ist ein Objekt, welches bei genauerer Beobachtung wesentliche Unterschiede zu den anderen Objekten in seiner Umgebung aufweist.

Dabei kann das Objekt (die Anomalie), wie in [6] beschrieben, vieles sein. Unter anderem ein Wert, eine Zeichenkette, eine Bedingung, ein Event, ein Verhalten oder eine Funktion.

Anomalien werden generell unterteilt in punktuelle Anomalien, kontextuelle Anomalien und kollektive Anomalien. Diese Kategorisierung ist wichtig, um später eine passende Methode zur Erkennung der Anomalien anwenden zu können. So ist ein Algorithmus, der punktuelle Anomalien erkennen soll, nicht in der Lage, kontextuelle oder kollektive Anomalien zu erkennen.

Abbildung 2.1: Beispielhafte Darstellung punktueller Anomalien

Punktuelle Anomalien

Punktuelle Anomalien, wie die in Abbildung (Abb). 2.1 gezeigten Anomalien 1 und 2, distanzieren sich von den anderen Objekten der Gruppe deutlich. Sie sind daran zu erkennen, dass sie sich in keiner Objekt-Wolke befinden. Eine Objekt-Wolke kann dabei jegliche Form haben, solange die Dichte der Punkte nicht zu stark abfällt. Die Aufgabe eines Algorithmus könnte es sein, diese Wolken zu erkennen und alle Punkte, die nicht in einer Wolke liegen, als Anomalie zu markieren. [4]

Eine einfache beispielhafte Anwendung ist die Auswertung eines Thermometers, bei dem der Normalbereich zwischen 35°C und 42°C liegt. Ein Wert von 45°C wäre in diesem Fall eine punktuelle Anomalie. Der Normalbereich wäre die Objekt-Wolke. [7]

Punktuelle Anomalien sind ein Kernpunkt dieser Arbeit. Alle vorgeschlagenen Methoden haben aufgrund ihrer leichten Architektur nur das Ziel, punktuelle Anomalien zu erkennen.

Kontextuelle Anomalien

Kontextuelle Anomalien, wie die in Abb. 2.2 gezeigte Anomalie t_2, sind nur dann Anomalien, wenn sie in Betracht des Kontextes abweichen. Diese Art der Anomalien wird in den meisten Fällen für Zeitverläufe genutzt. Wird der Kontext der Daten entfernt, so kann

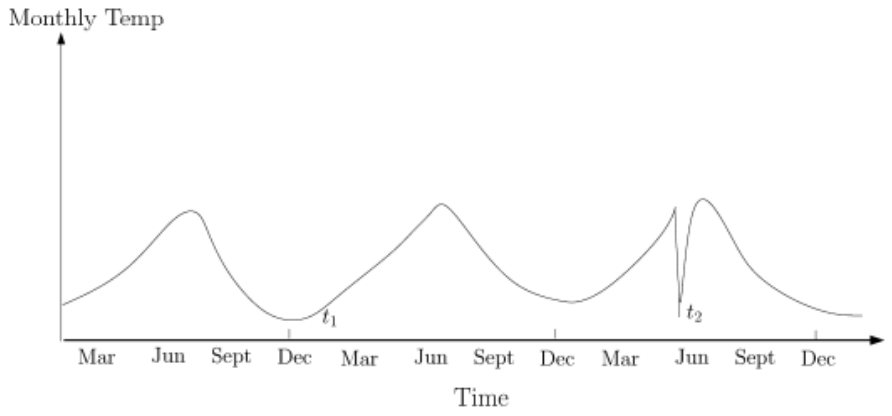

Abbildung 2.2: Beispielhafte Darstellung kontextueller Anomalien [4]

keine Schlussfolgerung auf die Anomalien gezogen werden. Abb. 2.2 zeigt beispielhafte Temperaturdaten über einen Zeitraum x. Es ist anzunehmen, dass die Temperaturen an t_1 und t_2 aus der Abb. 2.2 etwa gleich sind. Ohne Kontext sind es nur zwei gleiche Punkte. Doch mit dem Kontext, dass t_1 im Winter erfasst wurde und t_2 im Sommer, und es bekanntlich im Sommer warm ist, muss t_2 eine Anomalie sein. Die Anomalie ist dementsprechend dem Kontext bedingt. [4]

Kontextuelle Anomalien werden in dieser Arbeit nicht weiter behandelt, da die vorgelegten Methoden den Kontext der Daten nicht kennen.

Kollektive Anomalien

Kollektive Anomalien treten geschlossen als Gruppe in einem Datensatz auf. Die in Abb. 2.3 rot dargestellte Anomalie ist eine kollektive Anomalie, da sie von dem vorgegebenen Intervall des gesamten Zeitraumes abweicht. Sie tritt in diesem Beispiel etwa zwischen 1100 und 1400 auf der x-Achse auf, womit alle Datenpunkte in diesem Bereich zur kollektiven Anomalie gehören. Die Gruppe der kollektiven Anomalie muss in sich nicht anormal sein. Erst im Blick auf die gesamte Menge der Daten wird die Gruppe als Anomalie eingestuft. Der in Abb. 2.3 rot markierte Bereich mit dem Wert um -5.7 ist normal und trat auch zuvor schon auf. Erst durch seine abnormale Länge wird der Bereich anormal. [4]

Kollektive Anomalien werden in dieser Arbeit nicht weiter behandelt, da die präsentierten Methoden beim Auftritt einer Anomalie sofort die Analyse abbrechen und einen Alarm erstellen. Das Erfassen einer Gruppe von Fehlern ist dementsprechend nicht möglich.

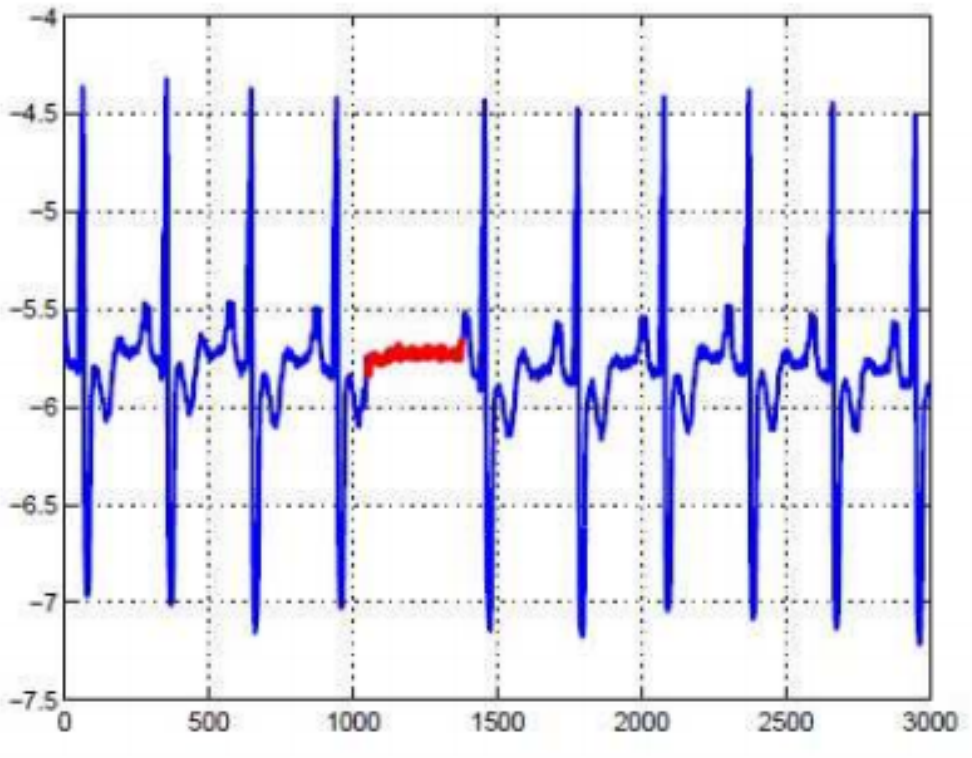

Abbildung 2.3: Beispielhafte Darstellung kollektiver Anomalien [4]

2.2 Methoden zur Erkennung von Anomalien und Ausreißern

Das Erkennen dieser Anomalien spielt besonders in der Informatik eine wichtige Rolle. Durch die stetig steigende Menge an Daten [8] wird es immer wichtiger, die essenziellen Daten herauszufiltern und Fehler darin zu erkennen. Aus diesem Grund wird die Erkennung von Anomalien immer wichtiger. Folglich wurden viele verschiedene Methoden entwickelt und miteinander verglichen. Das Ziel dieser Arbeit ist es nicht, diese Methoden miteinander zu vergleichen. Dies wurde bereits in mehreren wissenschaftlichen Artikeln untersucht (siehe [9, 10]).

Leider gibt es jedoch mehrere Faktoren, die das Finden von Anomalien erschweren. Wie in [4] beschrieben, ist, die einfachste Lösung (alles, was nicht normal ist als Anomalie zu erkennen) schwierig, da die Definition von *normal* nicht immer einfach ist. Dies hat folgende Gründe:

Enger Grad Das Einfangen von allen normalen Einträgen in einem Datensatz kann sich als äußerst schwierig erweisen. Je enger der Grad zwischen normalem und anormalem Verhalten ist, desto schwieriger ist es, dieses als solches zu erkennen.

Täuschung Ein menschlichen Angreifer, welcher vorhat, in ein System einzudringen, versucht immer, seine Spuren zu verwischen und die hinterlassenen Daten nicht anormal erscheinen zu lassen.

Weiterentwicklung Oftmals entwickelt sich die Umgebung, in der die Daten gesammelt werden, weiter. Dadurch ändert sich die Norm der Daten, sodass die gegenwärtige Norm in der Zukunft eine Anomalie sein kann und umgekehrt.

Kontext Abweichungen in einem System sind nicht generell übertragbar auf andere Systeme. So kann im medizinischen Bereich die kleinste Veränderung eines Wertes lebenskritisch sein, während an der Börse Schwankungen ganz normal sind. Eine Methode, die für kontextuelle Anomalien eines Systems entwickelt wurde, kann dementsprechend nur dort genutzt werden.

Unmarkierte Daten Häufig sind die Daten, die zum Testen und Validieren des Algorithmus genutzt werden, nicht als normal oder anormal markiert. Somit sinkt die Qualität der Anomalie-Erkennung, da der Algorithmus nicht lernen kann, ob seine Vorhersagen wahr oder falsch sind.

Rauschen Viele Datensätze besitzen ein Grundrauschen oder ein Störrauschen. Ähnelt dieses den Werten einer Anomalie, ist es schwierig, eine Anomalie von diesem Rauschen zu unterscheiden.

Als Eingabe erhalten diese Methoden meistens markierte oder unmarkierte Daten, mit oder auch ohne Kontext. Dabei wird in parametrisierte und nichtparametrisierte Methoden unterschieden. Nichtparametrisierte Methoden lernen eigenständig und passen sich an, wobei parametrisierte Methoden vom Anwender konfiguriert und angepasst werden müssen [11]. Diese Arbeit nutzt parametrisierte Methoden, da für nichtparametrisierte Methoden Maschinelles Lernen benötigt wird. Nach der Analyse dieser Eingangsdaten markiert der Algorithmus die Daten. Dabei kann es sich um eine Markierung wie „normal“, „anormal“ und/oder eine Punktebewertung handeln. Die Punkte geben die Anomalität (Abweichung von der Norm) des analysierten Datenobjekts an, sodass der Anwender einen Schwellwert zur Festlegung von Anomalien bestimmen kann. [4]

Die meisten der in diesem Kapitel vorgestellten Algorithmen werden nicht weiter in dieser Arbeit behandelt und wurden deshalb nur kurz angeschnitten. Sie dienen dazu, sich einen groben Überblick über die heutigen Möglichkeiten zu verschaffen. Weitere Informationen befinden sich in den Literaturangaben.

Classification based

Für diese Herangehensweise wird ein Klassifikator (eng. classifier) benötigt. Der erste Schritt ist es, ihn mit beaufsichtigten (eng. supervised) bzw. markierten Datensätzen oder unbeaufsichtigten (eng. unsupervised) bzw. nicht markierten Datensätzen zu trainieren. Somit soll er in normale und nicht normale Datensätze unterscheiden können. Dabei hat die Methode mit beaufsichtigen Daten wesentliche Vorteile, sodass die unbeaufsichtigte Methode in der meisten Literatur nicht aufzufinden ist. Außerdem wird unterschieden in eine klassenlose (eng. one-class) und eine klassenvolle (eng. multi-class) Technik. [4, 9, 12]

Diese Methodik bietet sehr viel Potenzial, benötigt jedoch Machine-Learning und wird deshalb in dieser Arbeit nicht weiter erläutert.

Distance und Density Based

Das Erkennen von Anomalien mittels Entfernung (eng. Distance) und/oder Dichte (eng. Density) basiert auf statistischen Methoden. Die Algorithmen sind nicht in allen Fachliteraturen gleich definiert, deshalb werden sie in dieser Arbeit folgend unterschieden:

- Algorithmen, welche die Entfernung zum nahestehendsten Nachbarn messen und damit eine Anomalie erkennen, basieren auf Entfernung [13].

- Algorithmen, welche in einem Radius r um den Punkt x prüfen, ob eine Anzahl p an Punkten vorhanden ist [14–16], basieren auf Dichte. In [17] wird dieser Algorithmus jedoch als „distance based“ bezeichnet.

Diese Verfahren eignen sich nicht für die Methoden dieser Arbeit, da sie Messpunkte benötigen, welche in ein Koordinatensystem eingefügt werden können. Die Methoden dieser Arbeit nutzen jedoch weder Zahlenwerte aus Daten noch wandeln sie Daten in Zahlen um. Eine Anwendung ist deshalb nicht möglich.

Window based

Bei dieser Methode wird ein Fenster (eng. window) der Größe x über die Datenmenge d gelegt. Dieses Fenster hat einen Start s und ein Ende e, wobei $e - s = x$. Der Start und das Ende werden dabei kontinuierlich verändert. So kann das Fenster in alle Richtungen verschoben werden. Dementsprechend werden nur die Daten ds bis de betrachtet. Objekte bzw. Werte können Anomalien sein, doch der Algorithmus würde diese nicht erkennen, wenn das Fenster nicht drüber liegt. Ein Wert kann somit ohne das Fenster als Anormal bezeichnet werden, doch im Fenster gilt er als Normal, da er, in Relation zu den anderen

Werten im Fenster, normal ist. Dies ist zugleich der Vorteil und der Nachteil dieses Verfahrens. Dieses Verfahren ergibt dementsprechend nur Sinn, wenn Daten nicht absolut, sondern relativ betrachtet werden sollen. Am besten eignet es sich deshalb für historische Daten. [5, 7, 10]

Dieses Verfahren wird später für die Lösungsvorschläge genutzt.

2.3 Big Data

Ein Begriff, der besonders in der Informatik viele Fragen und Interessen weckt. Die freie deutsche Übersetzung ist „eine große Anzahl an Daten“, welches auch gut widerspiegelt, warum es so gut zur Informatik passt. Denn eine große Anzahl an Daten befindet sich im Internet. Laut International Data Corporation (IDC) nutzten 2018 täglich mehr als 5 Milliarden Menschen das Internet, 2025 sollen es dann 75% der Weltbevölkerung, nämlich 6 Milliarden Menschen sein [8]. Und diese Menschen erzeugen ununterbrochen Daten. Selbst beim Lesen von Artikeln und Schauen von Videos wird das Nutzerverhalten erfasst und gespeichert, um später analysiert werden zu können. So lag die weltweite Datenmenge laut Schätzung der IDC in 2018 bei 33 Zettabytes und soll bis 2025 auf 175 Zettabytes ansteigen [8]. Ein Zettabyte sind 10^{21} Bytes, also eine Milliarde Terabytes. Diese Daten werden aufgeteilt in strukturierte, semistrukturierte und unstrukturierte Daten.

Strukturierte Daten sind markierte Daten und werden meist in relationalen Datenbanken gespeichert. Diese Daten beschreiben sich selbst. Die Informationen, welche Aussage ein Wert oder ein Wort hat, sind den Daten beigefügt. Ein Beispiel ist eine Textdatei mit Schlüsselwerten. [18]

Semistrukturierte Daten sind unter anderem Logdateien, Ortsinformationen, PDF-Dateien und/oder E-Mails. Sie sind nur teilweise markiert, sodass ein großer Teil von den Daten keine Beschreibung bzw. Meta-Informationen enthält. [18]

Unstrukturierte Daten sind Text, Bilder oder Ton, deren Inhalt in keinem Zusammenhang steht. [18]

Die analysierten Daten stammen in dieser Arbeit aus Logdateien und gehören laut der ober genannten Definition deshalb zu den semistrukturierten Daten.

Eine einheitliche Definition von Big Data gibt es zum Zeitpunkt der Verfassung dieser Arbeit nicht. Verschiedene existierende Definitionen sind dabei in die jeweiligen Interessengebiete unterteilt. So reichen die Definitionen von wirtschaftlichen und technischen über gesellschaftliche und rechtliche bis hin zu politischen Perspektiven [18]. Viele Autoren

[18–23] definieren im technologischen Kontext Big Data anhand von den drei bis vier V's. Diese sind Volume, Velocity (dt. Geschwindigkeit), Variety (dt. Vielfalt) und Veracity (dt. Richtigkeit). Andere Autoren nutzen bis zu zehn V's und wiederum andere wollen die V's durch die F's fast, flexibel, focused ersetzten. Folgend werden nur die vier V's erläutert.

Volume

Der Umfang (eng. volume) beschreibt die Masse oder auch die Größe der zu erfassenden, zu speichernden und zu analysierenden Daten. Es stellt sich die Frage, ob deshalb eine bestimmte Größe überschritten werden muss, damit ein Datensatz der Kategorie Big Data, zugeordnet werden kann und wie „groß" dieser Schwellwert ist. Durch die sich ändernde Definition von „groß" kann es einen solchen Schwellwert nicht geben. Heute sind vielleicht zehn Petabyte „groß" und in Zukunft gelten Datensätze erst ab zehn Zettabyte als „groß". Angenommen, es gibt eine statische Definition für „groß", so folgert aus einer Überschreitung nicht direkt das Fazit, dass der Datensatz zu Big Data gehört. Denn Big Data beschäftigt sich hauptsächlich mit semistrukturierten und unstrukturierten Daten. Eine aktuell zehn Petabyte große relationale Datenbank würde laut Definition nicht zu Big Data gehören. [18]

Zusammenfassend schreiben deshalb die Autoren in dem „Handbuch für die industrielle Praxis" [21]:

> „Volume (Menge) bezeichnet die schiere Masse an Daten, die mit herkömmlichen Methoden der Datenverarbeitung nicht mehr gespeichert, verarbeitet oder gar analysiert werden können."

Im Bezug auf Logdateien ist hinzuzufügen, dass die Größe von dem Logtypen abhängt. Außerdem ist die Größe der Informationen bei Logdateien schwer zu definieren, da diese oftmals analysiert und danach gelöscht werden. So bleiben nur die Ergebnisse der Analyse zurück, deren Größe schwer messbar ist.

Velocity

> „Velocity (Geschwindigkeit) steht für den Umstand, Daten in kleinsten Zeitabständen zu verarbeiten und auswertbar zu machen." [21]

Die Geschwindigkeit (eng. velocity) beschreibt die benötigte Zeit zum Lesen, Schreiben und/oder Analysieren von Daten. Welcher Aspekt genau gemeint ist, wird, wie [18] aufzeigt, von verschiedenen Autoren unterschiedlich wahrgenommen. Geschwindigkeitsprobleme beim Schreiben sind bei Nutzerinteraktionen wie z. B. Twitter-Tweets oder

Youtube-Uploads aufzufinden, beim Lesen spielt hauptsächlich das oben genannte Volume eine Rolle und wie gestreut die Daten abgespeichert sind Beziehungsweise (bzw) in welcher Struktur sie gespeichert werden. Die Geschwindigkeit der Analysen hängt dabei von der ausgewählten Methode zur Analyse ab.

In dieser Arbeit wird die Geschwindigkeit nicht ausführlich behandelt, da der Fokus auf der Methodik der Verfahren liegt.

Variety

Haben Daten verschiedene Formate, so wird von der Vielfalt (eng. variety) gesprochen. Dieser Faktor beschreibt die Variation der Strukturierungsformen der vorgelegten Daten [19]. Außerdem zählt zu dieser Charakteristik die Vielfalt der Quellen [20]. Bieten Daten eine zu hohe Auswahl an Quellen und Formaten, lassen sie sich in Big Data einordnen. Eine genaue Definition von „zu hohe Auswahl“ gibt es, wie bei den anderen Eigenschaften von Big Data, auch nicht.

Aufgrund dieser Beschaffenheit entsteht „die Anforderung, wenig oder gänzlich unstrukturierte Daten durch Algorithmen in eine auswertbare Struktur umzuformen, in der Zusammenhänge erkannt und mit bekannten Informationen verknüpft werden können“ [21].

Dies spielt in dieser Arbeit eine große Rolle, da die vorgestellten Verfahren keine festen Formate benötigen. Das Ziel ist eine generische Lösung, unabhängig von dem Format oder der Struktur der Logdatei.

Veracity

Veracity (dt. Richtigkeit) beschreibt die Qualität der Daten [19, 23]. Dies spielt eine besonders große Rolle, wenn Daten von dritten Quellen kommen. Es sollte infrage gestellt werden, ob die Daten vertrauenswürdig sind oder nicht. Ein Vergleich außerhalb der Informatik sind Nachrichten. So sollte nicht immer alles geglaubt werden, was man selber teilweise gesehen hat oder was von dritten, vielleicht unseriösen Quellen kommt.

In dieser Arbeit können wir diesen Aspekt vernachlässigen, da wir nur eigene Quellen analysieren und davon ausgehen, dass diese korrekt sind.

2.4 Data Mining und Machine Learning

Data Mining (dt. Datenmustererkennung [24]) und Machine Learning (dt. Maschinelles Lernen) unterscheiden sich und ergänzen sich gegenseitig. In der Literatur stehen diese Begriffe meist zusammen. Der Grund ist die Verknüpfung und Überschneidung der Begriffe. Die Datenmustererkennung beschreibt „die Extraktion implizit vorhandenen, nicht trivialen und nützlichen Wissens aus großen, dynamischen, relativ komplex strukturierten Datenbeständen“ [24]. Das Maschinelle Lernen beschreibt die „Techniken, mit denen man strukturierte Muster in Daten erkennen und beschreiben kann, mit dem Ziel, die Daten zu erklären und Voraussagen zu treffen“ [21]. Das Maschinelle Lernen wird gerne verknüpft mit Künstlicher Intelligenz und der Datenmustererkennung. Zusammenfassend lässt sich sagen, dass zur Datenmustererkennung unter anderem Verfahren des Maschinellen Lernens eingesetzt werden. Das Maschinelle Lernen beruht gleichzeitig jedoch auf anderen Verfahren der Datenmustererkennung. Beispielhafte Verfahren sind Entscheidungsbäume, Regelbäume, Assoziationsregeln, Clusteranalysen, Support-Vector-Machines und Neuronale Netze [21].

Viele der bis hierhin aufgezeigten Verfahren nutzen Maschinelles Lernen zur Umsetzung, weshalb sie für den späteren Verlauf dieser Arbeit nicht mehr relevant sind. Deshalb wird auch hier das Maschinelle Lernen nicht weiter erläutert. Weitere Informationen zu den Themen Datenmustererkennung und Maschinelles Lernen bieten die Quellen [21, 24–26].

Genauer wird jedoch die Konfusionsmatrix betrachtet. Konfusionsmatrizen sagen aus, wie gut oder schlecht ein Klassifikator klassifiziert. Deshalb ist zu wissen:

> „Klassifikation ist ein überwachtes Lernverfahren, das markierte Daten verwendet, um Objekte zu Klassen zuzuordnen. Es werden falsch positive und falsch negative Fehler unterschieden und auf dieser Basis zahlreiche Klassifikationskriterien definiert. Oft werden Paare solcher Kriterien zur Bewertung von Klassifikatoren verwendet.“ [27]

Die Konfusionsmatrix besteht dabei aus vier Feldern. Unterschieden wird in:

1. richtig positiv oder *true positive* (TP)
2. richtig negativ oder *true negative* (TN)
3. falsch positiv oder *false positive* (FP)
4. falsch negativ oder *false negative* (FN)

Wie erkennbar, bestehen die Namen der Felder aus jeweils 2 Wörtern. Das erst Wort ist *richtig* oder *falsch*, gefolgt von *positiv* oder *negativ*. Bei Feldern, die mit dem Wort *richtig*

anfangen, hat der Klassifikator einen Wert richtigerweise als positiv oder negativ angenommen. Felder, die mit dem Wort *falsch* anfangen, haben dementsprechend eine falsche Annahme getroffen. Die Wörter *positiv* und *negativ* sagen aus, wie der Wert betrachtet wurde. Somit sind:

1. richtig positiv, alle Fälle, in denen richtigerweise angenommen wurde, dass ein Wert positiv war

2. richtig negativ, alle Fälle, in denen richtigerweise angenommen wurde, dass ein Wert negativ war

3. falsch positiv, alle Fälle, in denen fälschlicherweise angenommen wurde, dass ein Wert positiv war

4. falsch negativ, alle Fälle, in denen fälschlicherweise angenommen wurde, dass ein Wert negativ war

In dieser Arbeit werden Konfusionsmatrizen (wie folgt) in Tabellen dargestellt.

Tabelle 2.1: Vorlage einer Konfusionsmatrix

		Angenommen		
		Ausfall	**Nicht-Ausfall**	**Summe**
Echt	**Ausfall**	TP	FN	
	Nicht-Ausfall	FP	TN	
	Summe			

Aus den Fehlern erster (FP) und zweiter (FN) Art sowie den anderen Variabeln (TP und TN) werden zur Beurteilung verschiedene Verhältnisse und Werte errechnet. Runkler beschreibt diese wie folgt (abgewandelt von gesunden und kranken Patienten auf Ausfall- und Nicht-Ausfall-Zeiträume):

- Anzahl der Klassifikationen $n = TP + TN + FP + FN$

- richtige Klassifikationen (engl. True Classifications) $T = TP + TN$ (Anzahl der richtig klassifizierten Zeiträume)

- falsche Klassifikationen (engl. False Classifications) $F = FP + FN$ (Anzahl der falsch klassifizierten Zeiträume)

- relevante Klassifikationen $R = TP + FN$ (Anzahl der Ausfälle)

- irrelevante Klassifikationen $I = FP + TN$ (Anzahl der Nicht-Ausfälle)

- positive Klassifikationen $P = TP + FP$ (Anzahl der als Ausfall klassifizierten Zeiträume)
- negative Klassifikationen $N = TN + FN$ (Anzahl der als Nicht-Ausfall klassifizierten Zeiträume)
- Korrektklassifikationsrate oder Vertrauenswahrscheinlichkeit $\frac{T}{n}$ (Wahrscheinlichkeit, dass ein Zeitraum korrekt klassifiziert wird)
- Falschklassifikationsrate $\frac{F}{n}$ (Wahrscheinlichkeit, dass ein Zeitraum falsch klassifiziert wird)
- Richtig-Positiv-Rate (engl. True Positive Rate) oder Sensitivität oder Empfindlichkeit oder Trefferquote (engl. Recall) $TPR = \frac{TP}{R}$ (Wahrscheinlichkeit, dass ein Zeitraum mit Ausfall als Ausfall klassifiziert wird)
- Richtig-Negativ-Rate (engl. True Negative Rate) oder Spezifität $TNR = \frac{TN}{I}$ (Wahrscheinlichkeit, dass ein Zeitraum ohne Ausfall als Nicht-Ausfall klassifiziert wird)
- Falsch-Positiv-Rate (engl. False Positive Rate) oder Ausfallrate $FPR = \frac{FP}{I}$ (Wahrscheinlichkeit, dass ein Zeitraum ohne Ausfall als krank klassifiziert wird)
- Falsch-Negativ-Rate (engl. False Negative Rate) $FNR = \frac{FN}{R}$ (Wahrscheinlichkeit, dass ein Zeitraum mit Ausfall als Nicht-Ausfall klassifiziert wird)
- positiver Vorhersagewert oder positiver prädiktiver Wert oder Relevanz oder Wirksamkeit oder Genauigkeit (engl. Precision) $\frac{TP}{P}$ (Wahrscheinlichkeit, dass ein als Ausfall klassifizierter Zeitraum ein Ausfall ist)
- negativer Vorhersagewert oder negativer prädiktiver Wert oder Segreganz oder Trennfähigkeit $\frac{TN}{N}$ (Wahrscheinlichkeit, dass ein als Nicht-Ausfall klassifizierter Zeitraum ein Nicht-Ausfall ist)
- negative Falschklassifikationsrate $\frac{FN}{N}$ (Wahrscheinlichkeit, dass ein als Nicht-Ausfall klassifizierter Zeitraum ein Ausfall ist)
- positive Falschklassifikationsrate $\frac{FP}{P}$ (Wahrscheinlichkeit, dass ein als Ausfall klassifizierter Zeitraum ein Nicht-Ausfall ist)
- F-Maß F = $2 \times \frac{TP}{R+P}$ (harmonisches Mittel aus positivem Vorhersagewert und Richtig-Positiv-Rate)

Zusätzlich wird in dieser Arbeit noch der Matthews Correlation Coefficient (MCC) (dt. Matthews Korrelationskoeffizient) genutzt.

$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$ [28]

Der Matthews Correlation Coefficient (MCC) ähnelt dem F-Maß, doch er bezieht sich dabei nicht nur auf die positiven Werte TP und TN. Beide Maße versuchen dabei den gewählten Klassifikator in seiner Funktionsweise zu beurteilen. Weitere Erläuterungen zu dem Unterschied zwischen dem F-Maß und dem MCC sind bei den Ergebnissen der empirischen Analyse (Kapitel 3.4) zu finden. Genauer untersuchten Davide Chicco und Gisueppe Jurman die Vorteile des MCC gegenüber des F-Maß in Ihrem Artikel „The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation“ [28].

2.5 Logdatei-Analyse

Logdatei

Das Bundesamt zur Sicherheit in der Informationstechnik schrieb 2013:

> „Unter Logdaten werden (...) in erster Linie Daten verstanden, die ein System oder eine Anwendung ereignisbezogen generiert, und die entweder auf dem System selbst oder auf einem gesonderten System gespeichert werden. Wohlbekannte Beispiele für Ereignisse, zu denen Logdaten erzeugt werden, sind die An- oder Abmeldung von Benutzern, gescheiterte Anmeldeversuche und Statusmeldungen von Diensten, wie sie beispielsweise unter Unix-Betriebssystemen in der syslog-Datei oder unter Windows im Ereignisprotokoll festgehalten werden. Weitere Beispiele sind Anfragen an Web- oder Proxysever, sowie Informations- oder Warnmeldungen von Intrusion Detection Systemen, die diese Anwendungen meist in eigenen Logdateien oder -datenbanken speichern.“ [29]

Diese Definition von Logdaten und Logdateien wird folglich als Standard in dieser Arbeit bestimmt. Der in der Studie [29] genutzte Begriff „Monitoringdaten“ wird ergänzend zu dem Begriff Logdaten genutzt und beschreibt folglich Daten, welche Auslastungsinformationen der loggenden Anwendung enthalten.

Um diese Logeinträge zu verstehen, wurden zahlreiche Tools entwickelt. Die meisten dieser Tools beziehen sich dabei jedoch auf die Logdatei-Analyse von Servern (z. B. Proxy-, Mail-, Firewall- oder Webservern). Die Formate dieser Server-Logdateien sind nicht immer gleich, denn einen Standard für das Format von Logdateien gibt es nicht. Der Grund für die hohe Varianz der Formate ist die Menge der Anwendungen selbst. Da die Logdaten dazu dienen, Fehler oder Störungen einer Anwendung zu finden, sind diese der Anwendung angepasst. Die Entwickler der Anwendung entscheiden über das Format der Logdatei. Aus diesem Grund unterscheiden sich Logdateien in vielen Aspekten. Jedes Mal, wenn eine

neue Anwendung erstellt wird und die Entwickler sich Gedanken zur Logdatei machen, sollten sie folgendes bedenken:

Binär- oder Klartext **Soll die Datei für einen Menschen lesbar sein oder soll sie nur von einer Maschine analysiert werden?** Binärformate sind äußerst effektiv im Bezug auf die Datenmenge, wobei Klartext die Datei aufbläht. Durch ein Klartextformat kann die Datei auch von Menschen analysiert werden, doch nicht nur die Größe wird dadurch beeinflusst, auch das automatisierte Analysieren dieser Dateien wird dadurch schwieriger. [29]

Zeichensätze und Sprachen **In welcher Sprache wird der Text gespeichert und werden besondere Zeichen benötigt?** Soll die Logdatei deutschen Text enthalten, so sollte der Zeichensatz Umlaute unterstützen. Sind es Logdateien über nicht zustellbare Nutzernachrichten, müssen eventuell Emojis auch erfasst werden können. Auch das Festlegen des Zeichensatzes ändert die Größe der Datei. Außerdem muss das analysierende Programm diesen Zeichensatz unterstützen. [29]

Einzeilig oder mehrzeilig **Wie viele Zeilen soll ein Logeintrag haben?** Geht eine Logmeldung über mehrere Zeilen, muss ein Anfang und ein Ende definiert werden, sonst kann das Analyseprogramm nicht wissen, welche Nachrichten zusammengehören. Für diese Art wird dementsprechend ein Kontext benötigt. Deshalb sind einzeilige Logeinträge für Maschinelle Verarbeitung zu bevorzugen. Diese können dadurch jedoch unleserlicher für den Menschen werden. [29]

Trennzeichen **Welches Trennzeichen soll zum Unterscheiden der Felder genutzt werden?** Enthält ein Logeintrag beispielsweise nur Zahlen, so kann er durch Leerzeichen getrennt werden. Enthält er aber Sätze, so wäre ein Leerzeichen als Trennzeichen ungeeignet, denn die einzelnen Wörter würden als eigenes Feld interpretiert werden. In diesem Fall wäre ein senkrechter Strich vorteilhafter. [29]

Felder-Tagging **Können Felder in einem Logeintrag wegfallen?** Logeinträge müssen markiert werden, wenn sie nicht bei jedem Eintrag das gleiche Format aufweisen. Manche Anwendungen tragen in den Logeintrag keinen Wert ein, der nicht existiert. Dadurch verändert sich Zeilen-

weise das Format. Somit kann ein Programm, welches solch einen Logeintrag analysiert, nicht wissen, welches Feld welche Informationen enthält. Dadurch müssen die Felder markiert bzw. getaggt werden (z. B. „Quell-IP-Adresse=10.0.0.1“). [29]

Datum und Zeit

Wie soll die zeitliche Abhängigkeit eines Eintrages dargestellt werden? Um später zu wissen, wann eine Störung aufgetreten ist und wie lange sie gedauert hat, werden die Logeinträge zeitlich markiert. Auch hier sind die Möglichkeiten nahezu unendlich. Eine Möglichkeit ist, dass jeder Eintrag in Millisekunden den Verzug zu dem vorherigen Eintrag angibt. Dies wäre eine relative Darstellung. Eine absolute Darstellung ist es, den Zeitstempel in Millisekunden seit dem 01. Januar 1970 zu nutzen. Diese Zahl kann jedoch ein Mensch normalerweise nicht interpretieren. Deshalb wird die Zeit in einem Kalenderformat darstellt (z. B. 23.09.2020 13:21:55.978). Durch die verschiedenen Länder und Kulturen kommen auch dort verschiedene Formate zum Einsatz (Monat vor Tag oder ein 12 Stunden Format mit am und pm Angaben). Zu dieser bereits großen Vielfalt an Möglichkeiten kommt noch das Problem der Zeitzone dazu. Es muss definiert werden, in welcher Zeitzone der Logeintrag geschrieben wird. Bei europäischen Zeitzonen muss zusätzlich auf Sommer- und Winterzeit geachtet werden. [29]

Es wird deutlich, dass bereits die Formatierung einer Logdatei viele Schwierigkeiten bereitet und die Vielfalt an Logdateien und deren Formaten enorm ist. Um genau dieses Problem zu lösen, bietet diese Arbeit einen Algorithmus, welcher von dem Format der Logdatei unabhängig ist. Der Grundgedanke ist eine Analyse von Logdateien ohne die Angabe des Formates. Solch ein Ansatz wurde, im Rahmen dieser Arbeit, in keiner bisherigen Literatur gefunden.

Aktuelle Programme (zur Analyse von Logdateien) sind entweder für ein spezielles Gebiet entwickelt (z. B. Webserver) oder bieten einen eigenen Parser, mit dem es möglich ist, durch Mustererkennung verschiedene Parameter aus Logdaten auszulesen, abzuspeichern und zu analysieren. Der Konfigurationsaufwand hängt von der zu analysierenden Logdatei-Struktur ab. In allen Fällen muss das Logformat aus einer Reihe von Standards ausgesucht oder selbst konfiguriert werden. Jedes der vorher beschriebenen Aspekte erschwert dies. Deshalb sind die vorhandenen Tools oftmals beschränkt in ihren Möglichkeiten.

Ein beispielhaftes System zur Erfassung, Verarbeitung und Analyse von Logdateien ist der ELK-Stack. Der ELK-Stack besteht aus den drei Komponenten Elasticsearch, Logstash und Kibana. Dabei ist Elasticsearch die Suchmaschine, die Datenbank und das Analyse-Tool

zugleich. Die in der Datenbank von Elasticsearch gespeicherten Daten können durch eine Query-Sprache (Query Domain Specific Language) abgerufen werden. Logstash sammelt die Logdateien und ermöglicht es, diese zu parsen und zu verändern. Dadurch können die Informationen aus den einzelnen Spalten eines Logeintrages ergänzt oder verworfen werden. Außerdem bietet Logstash die Möglichkeit, die Logdateien an andere Services weiter zu senden. Die verarbeiteten Daten werden dann zum Speichern und Analysieren an Elasticsearch weitergeleitet. Um diese Daten darzustellen, wird Kibana genutzt, es dient zur Visualisierung. Die vorgegebene Abfragesprache ermöglicht es, dynamisch Diagramme zu erstellen. Unter den Diagramm-Arten befinden sich unter anderem Charts, Gauges, Karten und Tabellen. [30]

Weitere Systeme zur Analyse von Logdateien sind unter anderem:

- Graylog [31]
- Nagios [32]
- LOGalyze [33]
- Fluentd [34]
- DataDog [35]
- Security Event Manager [36]
- Papertrail [37]
- Loggly [38]
- Sematext [39]
- XpoLog [40]

Die in dieser Arbeit genutzten Logdateien enthalten Logdaten und Monitoringdaten einer Java-Anwendung (siehe Abb. 2.4), welche auf der Plattform JHipster (Java Hipster) [41] basiert. Die gezeigten Einträge enthalten keine oder geschwärzte Nutzerdaten, somit liegt keine Geheimhaltung oder Verschwiegenheit vor. Das Grundgerüst besteht dabei aus Spring Boot als Backend und Angular als Frontend. Die Logeinträge werden nur Einträge aus dem Backend sein, da dieses Informationen zu Verfügbarkeit der Anwendung gibt und Einträge aus dem Frontend nicht gespeichert werden.

```
2020-09-29 08:33:44.263 [] [HikariPool-1 housekeeper] DEBUG com.zaxxer.hikari.pool.HikariPool.logPoolState - HikariPool-1 - Before cleanup stats (total=13, active=0, idle=13, waiting=0)
2020-09-29 08:33:44.263 [] [HikariPool-1 housekeeper] DEBUG com.zaxxer.hikari.pool.HikariPool.logPoolState - HikariPool-1 - After cleanup  stats (total=13, active=0, idle=13, waiting=0)
2020-09-29 08:33:51.953 [] [XNIO-2 task-30] WARN  o.z.p.spring.web.advice.AdviceTrait.log - Unauthorized: Full authentication is required to access this resource
2020-09-29 08:33:52.957 [] [XNIO-2 task-32] WARN  o.z.p.spring.web.advice.AdviceTrait.log - Unauthorized: Full authentication is required to access this resource
2020-09-29 08:33:54.987 [] [XNIO-2 task-5] WARN  o.z.p.spring.web.advice.AdviceTrait.log - Unauthorized: Full authentication is required to access this resource
2020-09-29 08:33:55.990 [] [XNIO-2 task-2] WARN  o.z.p.spring.web.advice.AdviceTrait.log - Unauthorized: Full authentication is required to access this resource
2020-09-29 08:33:58.015 [] [XNIO-2 task-12] WARN  o.z.p.spring.web.advice.AdviceTrait.log - Unauthorized: Full authentication is required to access this resource
2020-09-29 08:33:59.023 [] [XNIO-2 task-18] WARN  o.z.p.spring.web.advice.AdviceTrait.log - Unauthorized: Full authentication is required to access this resource
2020-09-29 08:34:03.141 [] [XNIO-2 task-11] WARN  o.z.p.spring.web.advice.AdviceTrait.log - Unauthorized: Full authentication is required to access this resource
2020-09-29 08:34:04.150 [] [XNIO-2 task-13] WARN  o.z.p.spring.web.advice.AdviceTrait.log - Unauthorized: Full authentication is required to access this resource
2020-09-29 08:34:06.155 [] [XNIO-2 task-9] WARN  o.z.p.spring.web.advice.AdviceTrait.log - Unauthorized: Full authentication is required to access this resource
2020-09-29 08:34:06.246 [] [XNIO-2 task-24] ERROR o.z.p.spring.web.advice.AdviceTrait.log - Internal Server Errorjava.io.IOException: Connection reset by peer
  at sun.nio.ch.FileDispatcherImpl.write0(Native Method)
  at sun.nio.ch.SocketDispatcher.write(SocketDispatcher.java:47)
  at sun.nio.ch.IOUtil.writeFromNativeBuffer(IOUtil.java:93)
  at sun.nio.ch.IOUtil.write(IOUtil.java:51)
  at sun.nio.ch.SocketChannelImpl.write(SocketChannelImpl.java:471)
  at org.xnio.nio.NioSocketConduit.write(NioSocketConduit.java:164)
  at io.undertow.protocols.ssl.SslConduit.doWrap(SslConduit.java:892)
  at io.undertow.protocols.ssl.SslConduit.write(SslConduit.java:379)
  at io.undertow.server.protocol.http.HttpResponseConduit.write(HttpResponseConduit.java:646)
  at io.undertow.conduits.ChunkedStreamSinkConduit.doWrite(ChunkedStreamSinkConduit.java:163)
  at io.undertow.conduits.ChunkedStreamSinkConduit.write(ChunkedStreamSinkConduit.java:127)
  at io.undertow.conduits.ChunkedStreamSinkConduit.write(ChunkedStreamSinkConduit.java:216)
  at io.undertow.conduits.DeflatingStreamSinkConduit.performFlushIfRequired(DeflatingStreamSinkConduit.java:419)
  at io.undertow.conduits.DeflatingStreamSinkConduit.deflateData(DeflatingStreamSinkConduit.java:487)
```

Abbildung 2.4: Screenshot mehrerer Logeinträge aus LightBSM

3 Empirische Analyse

3.1 Konzept

Das hier vorgestellte Verfahren soll anhand einer Anomalie-Erkennung die Verfügbarkeit einer Anwendung analysieren. Um unabhängig von dem eingesetzten Bereich und dem Logformat zu sein, werden Textmuster, die auf einen Ausfall hindeuten, mithilfe von regulären Ausdrücken definiert. Der Algorithmus zur Umsetzung des Verfahrens überwacht dabei eine oder mehrere Logdateien. Sobald diese Logdateien einen neuen Eintrag erhalten, wird dieser mit dem definierten regulären Ausdruck verglichen. Findet eine Übereinstimmung statt, so wird angenommen, dass die Verfügbarkeit der überwachten Anwendung gestört ist. Daraus lässt sich ein Alarm erstellen, welcher die verantwortlichen Personen kontaktiert oder eine automatisierte Routine durchführt. Sollte keine Übereinstimmung stattfinden, so wird nichts getan. Ein beispielhafter regulärer Ausdruck wäre .*ERROR.*, dieser würde jedes Mal eine Übereinstimmung finden, wenn der neue Logeintrag den genauen Wortlaut „ERROR“ enthält.

Die überwachten Logdateien müssen dabei nicht zwangsläufig die Logdateien der überwachten Anwendung sein. In einigen Fällen enthalten auch abhängige Programme und Prozesse Informationen über die Verfügbarkeit der überwachten Anwendung. Dies ist besonders sinnvoll, wenn die Logdateien der zu überwachenden Anwendung nicht verfügbar sind. In dem in dieser Arbeit durchgeführten Test wird dafür die Logdatei eines Datenzulieferers überwacht.

Durch die automatisierte Überwachung der Logdateien müssen Anwender nicht mehr periodisch in diese hineinschauen, um Fehler frühzeitig zu erkennen. Deshalb wird gehofft, dass mit einer geringen Menge an Fehlalarmen eine hohe Menge an Störungen erkannt werden kann. Diese Alarme sollen dabei so schnell wie möglich erstellt werden, um die Dauer der Störung zu minimieren.

3.2 Test

3.2.1 Umgebung

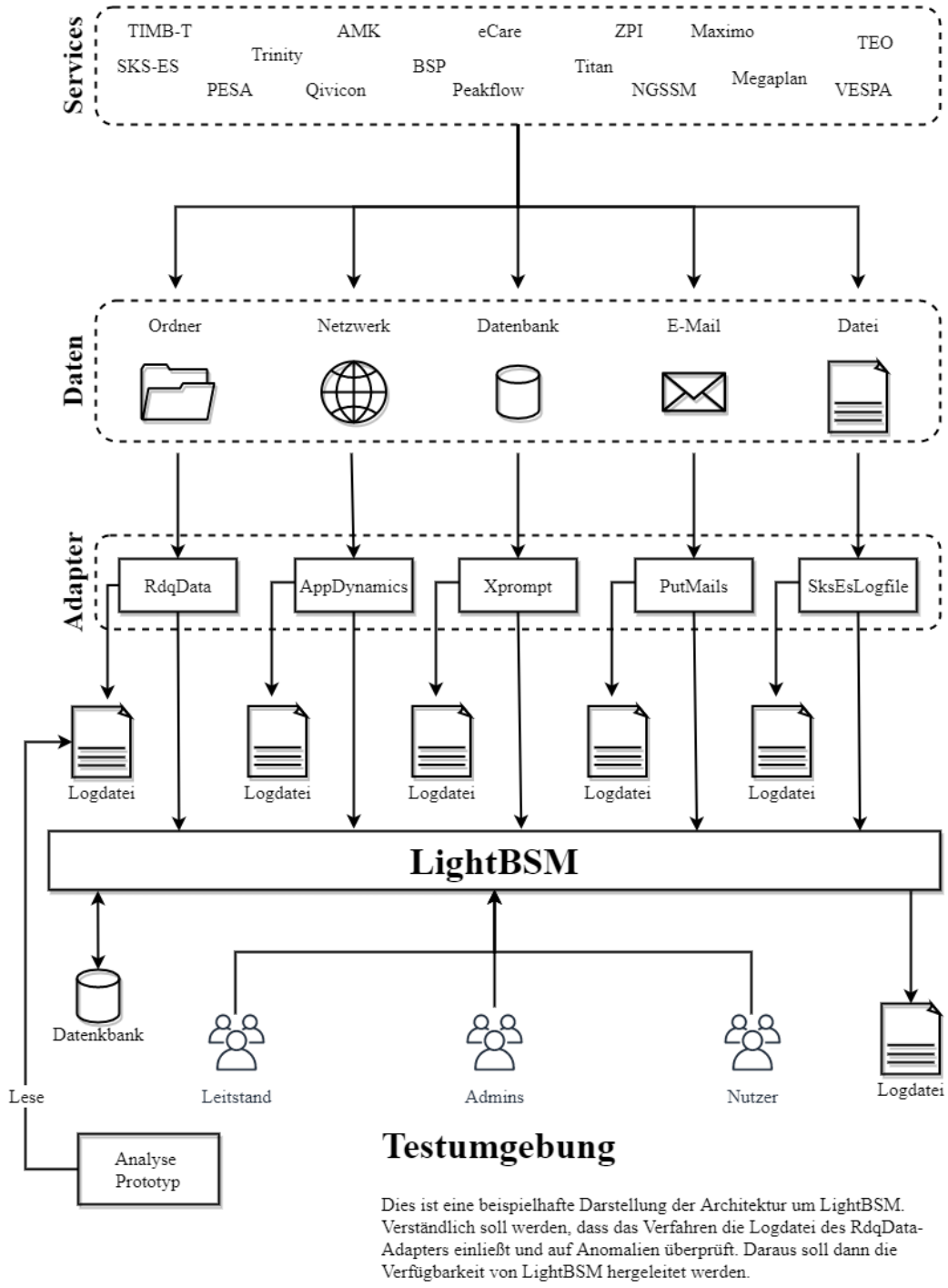

Abbildung 3.1: Vereinfachte Darstellung der LightBSM Architektur und Testumgebung

Für den hier durchgeführten Test ist die in Abb. 3.1 gezeigte Architektur anzunehmen. Sie entspricht leicht verändert der Architektur um LightBSM. Der Unterschied zur Realität ist, dass es wesentlich mehr Adapter, Services, LightBSM-Instanzen, Datenbank-Instanzen und Nutzergruppen gibt. Mit dieser vereinfachten Darstellung wird jedoch der Datenfluss deutlich. Die Services senden und/oder speichern ihre Daten auf verschiedene Art und Weise. Die Adapter lesen oder empfangen die Daten, verarbeiten diese und senden die entstandenen Metriken an LightBSM. Dabei schreiben sie Ihren Ablauf in eine jeweilige Logdatei. LightBSM speichert und liest alle Daten in und aus einer Datenbank, außerdem speichert es (wie die Adapter) seine Abläufe in einer Logdatei. Die Nutzer können sich dann diese Daten in LightBSM anzeigen lassen. Das hier getestete Verfahren nutzt die Logeinträge aus den Logdateien des RdqData-Adapters. Daran wird deutlich, dass diese Daten aus zweiter Quelle stammen. Die Daten aus erster Quelle kommen direkt aus der Datenbank oder der Logdatei der überwachten Anwendung, hier LightBSM. Warum diese Daten aus zweiter Quelle verwendet werden und warum ausgerechnet Logeinträge von dem RdqData-Adapter genutzt werden, wird in den nächsten Kapiteln beschrieben.

3.2.2 Ziele, Aufbau und Daten

Mit diesem Test sollen folgende Fragen beantwortet werden:

1. Wie viele Ausfälle wurden erkannt?

2. Wie viele vorhergesagte Ausfälle sind korrekt?

3. Wie viele Alarme entstanden in den Zeiträumen der Störungen?

Um diese Fragen zu beantworten und diesen Test durchzuführen, werden die bereits erstellten Logdateien eines Adapter-Programms rückblickend analysiert. Dieser Adapter wird alle fünf Minuten gestartet und sendet drei Metriken an LightBSM. Die Anwendung LightBSM ist in diesem Fall die überwachte Anwendung. Aus der Analyse der Adapter-Logdateien sollen dementsprechend Schlüsse über die Verfügbarkeit der Anwendung gezogen werden können. Bei einem Durchlauf erzeugt der Adapter verschiedene Logeinträge. Diese Logeinträge geben kurz an, welche Aufgaben der Adapter abarbeitet. Nach dem Auslesen der Rohdaten verarbeitet der Adapter diese und erzeugt daraus Metrikdaten. Diese muss der Adapter im letzten Schritt an LightBSM senden. Der letzte Logeintrag eines Durchlaufes enthält somit die Information, ob das Senden an LightBSM erfolgreich war. Dabei wird angegeben, wie viele Metriken erfolgreich zugestellt werden konnten und wie viele fehlgeschlagen sind. Sollten eine oder mehr Metriken erfolgreich zugestellt werden, so kann in diesem Fall davon ausgegangen werden, dass LightBSM verfügbar ist. Sollten alle drei Metriken nicht zugestellt werden können, so wird davon ausgegangen, dass LightBSM nicht verfügbar ist. Der Screenshot in Abb. 3.2 zeigt einen erfolgreichen Durchlauf, während der Screenshot in Abb. 3.3 einen nicht erfolgreichen Durchlauf zeigt. Dieser nicht erfolgreiche Durchlauf erfolgte aufgrund einer Störung in LightBSM.

```
2020-10-07 09:53:04,435 INFO  [main] de.telekom.lightBsm.core.Adaptor: {apiProtocol=http, apiUser=vsmsync, apiPort=8090, archivePath=/
2020-10-07 09:53:04,436 INFO  [main] de.telekom.lightBsm.core.Adaptor: Starting FILEAdaptor [RdqData] with id [16020571828182197]
2020-10-07 09:53:07,248 INFO  [Thread-0] de.telekom.lightBsm.core.Adaptor: start: ThreadSendMsgs_0
2020-10-07 09:53:07,248 INFO  [Thread-0] de.telekom.lightBsm.core.Adaptor: finish: ThreadSendMsgs_0 with 0 ok-msgs and 0 error-msgs
2020-10-07 09:53:07,539 INFO  [Thread-1] de.telekom.lightBsm.core.Adaptor: start: ThreadSendMsgs_0
2020-10-07 09:53:07,617 INFO  [Thread-1] de.telekom.lightBsm.core.Adaptor: finish: ThreadSendMsgs_0 with 3 ok-msgs and 0 error-msgs
```

Abbildung 3.2: Logeinträge nach einem erfolgreichen Durchlauf des RdqData-Adapters

```
2020-09-17 15:53:03,955 INFO  [main] de.telekom.lightBsm.core.Adaptor: {apiProtocol=http, apiUser=vsmsync, apiPort=8090, archivePath=/
2020-09-17 15:53:03,956 INFO  [main] de.telekom.lightBsm.core.Adaptor: Starting FILEAdaptor [RdqData] with id [16003507828132005 7]
2020-09-17 15:53:04,224 ERROR [main] de.telekom.lightBsm.core.Adaptor: error response for message [j_username=vsmsync&j_], sessionId [
2020-09-17 15:53:04,277 INFO  [Thread-0] de.telekom.lightBsm.core.Adaptor: start: ThreadSendMsgs_0
2020-09-17 15:53:04,277 INFO  [Thread-0] de.telekom.lightBsm.core.Adaptor: finish: ThreadSendMsgs_0 with 0 ok-msgs and 0 error-msgs
2020-09-17 15:53:04,335 ERROR [main] de.telekom.lightBsm.core.Adaptor: error response for message [j_username=vsmsync&j_], sessionId [
2020-09-17 15:53:04,386 INFO  [Thread-1] de.telekom.lightBsm.core.Adaptor: start: ThreadSendMsgs_0
2020-09-17 15:53:04,397 ERROR [Thread-1] de.telekom.lightBsm.core.Adaptor: error response for message [appName=VSM&transName=RdqData&p
2020-09-17 15:53:04,421 ERROR [Thread-1] de.telekom.lightBsm.core.Adaptor: error response for message [appName=VSM&transName=RdqData&p
2020-09-17 15:53:04,427 ERROR [Thread-1] de.telekom.lightBsm.core.Adaptor: error response for message [appName=VSM&transName=RdqData&p
2020-09-17 15:53:04,427 INFO  [Thread-1] de.telekom.lightBsm.core.Adaptor: finish: ThreadSendMsgs_0 with 0 ok-msgs and 3 error-msgs
```

Abbildung 3.3: Logeinträge nach einem fehlgeschlagenen Durchlauf des RdqData-Adapters

In Rahmen dieses Testverfahrens wird dementsprechend der reguläre Ausdruck *.*3 error-msgs.** zum Erfassen von Anomalien festgelegt. Der betrachtete Zeitraum startet am 07.03.2020 um 16:08 Uhr und endet am 07.10.2020 um 09:53 Uhr. In diesem Zeitraum wurden 652551 Zeilen in die Logdateien geschrieben. Diese werden folgend zeilenweise eingelesen und mit dem regulären Ausdruck verglichen. Bei einer Übereinstimmung wird eine Anomalie und somit eine Störung erkannt. Dies wird als Alarm notiert. Aus den erstellten Alarmen lassen sich dadurch rückblickend Ausfälle ableiten.

Um die angenommenen Ausfälle zu überprüfen, stehen Daten der Ausfälle in diesem Zeitraum zur Verfügung. Diese Daten wurden aus E-Mails und aggregierten Datenbankeinträgen erfasst. Die E-Mails stammen von Entwicklern nach dem Neustart einer neuen Version und dem Neustart der Applikation bei bemerkten Ausfällen durch Menschen oder Maschine. Um die erfassten Zeitpunkte aus diesen E-Mails zu bestätigen, wurden die Ausführungsmetriken aller Adapter (die LightBSM beliefern) für den auswählten Zeitraum manuell überprüft. Jedes Mal, wenn ein Adapter Daten an LightBSM sendet, speichert LightBSM den Zeitstempel und die Laufzeit. Dabei erhöht er außerdem die Anzahl der Ausführungen dieses Adapters um eins. Durch eine Gruppierung dieser Ausführungsmetriken auf Stundenbasis ließ sich eine Abweichung vom Normalfall erkennen. Aus diesen Abweichungen lässt sich schließen, dass und wie oft die Adapter ihre Daten nicht zustellen konnten. Um die Zeitpunkte zu bestätigen und zu ergänzen, musste dementsprechend eine Abweichung bei allen Adaptern vorliegen. In der Tabelle A.1 sind alle Neustarts, die aus E-Mails hervorgehen, dokumentiert.

Wie in der Tabelle A.1 zu erkennen ist, dauern manche Ausfälle länger und benötigen mehr als einen Neustart. Aus diesen Grund werden ab hier sowohl die Alarmierungen als auch

die Ausfälle in Zeitspannen zusammengefasst. Die Tabelle A.2 zeigt alle aufgetretenen Störungen (nicht nur aus E-Mails) zwischen März und Oktober 2020.

3.2.3 Hindernisse

Der hier genutzte Datensatz aus den Logeinträgen des Adapters ist nicht perfekt. Versucht man etwa, die Anzahl der eingelesenen Zeilen rechnerisch zu belegen, so wird schnell erkennbar, dass dies nicht stimmen kann. Der Zeitraum entspricht 307725 Minuten. Wenn der Adapter alle fünf Minuten einmal ausgeführt wird und dabei sechs Logeinträge erstellt, wären das $307725 \times \frac{6}{5} = 369270$ Einträge. Die Differenz beträge dementsprechend $652551 - 369270 = 283281$ Zeilen. Der Grund sind nicht, wie vielleicht anzunehmen, die Fehlermeldungen, welche fünf zusätzliche Logeinträge erzeugen. Am 28.04.2020 um 10:48 Uhr wurde das Loglevel von TRACE auf INFO gestellt. Bis zu diesem Zeitpunkt loggte der Adapter dadurch wesentlich mehr Einträge. Genau genommen erstellte er pro Durchgang 25 Logeinträge. Somit muss neu gerechnet werden. Vom Start bis zum Ändern des Loglevels waren es 74500 Minuten, also 14900 Durchläufe, also 372500 (14900×25) Logeinträge. Von da an waren es noch 233225 Minuten, was in 46645 Durchläufen und somit 279870 (46645×5) Zeilen endet. Zusammengerechnet sind das 652370 Zeilen, nur 181 weniger als in Wirklichkeit. Diese 181 zusätzlichen Zeilen sind hauptsächlich durch Fehler-Logeinträge entstanden. Bei genauerem Blick wird jedoch klar, dass immer noch etwas nicht stimmt. Denn wenn fünf Logeinträge bei einem erfolglosen Durchlauf entstehen, müssten es $\frac{181}{5} = 36,2$ fehlerhafte Durchläufe gewesen sein. Diese Zahl entspricht aus folgenden Gründen nicht der Realität:

1. Der erste Durchlauf ist in der Logdatei abgeschnitten.

2. Der Adapter ist zwischenzeitig ausgefallen und erzeugte zu diesen Zeitpunkten keine Logeinträge.

3. Es ist unbekannt, wann und wie oft das Loglevel vor dem 28.04.2020 um 10:48 Uhr gewechselt wurde.

Ein anderes Hindernis für diesen Test war der Ausfall des Adapters. Wie bereits vermutet, hat dieses Verfahren einen großen Nachteil: Sollten keine Logeinträge erstellt werden, so kann auch nichts analysiert werden. Wenn nichts analysiert werden kann, kann keine Anomalie erkannt werden, weswegen keine Störung erfasst werden kann. Und genau dies ist hier zweimal aufgetreten. Der Adapter hat zu den in Tabelle 3.1 gezeigten Zeitpunkten keine Logeinträge erstellt. Zu diesen Zeiten muss es entweder ein Problem beim Starten, der Ausführung des Adapters oder im Betriebssystem gegeben haben. An beiden Tagen der Ausfälle gab es auch eine Störung in LightBSM.

Tabelle 3.1: Ausfälle des RdqData-Adapters zwischen März und Oktober 2020

Ausfall	Zeitraum
1	24.07.2020 06:23 - 24.07.2020 06:37
2	27.07.2020 08:08 - 27.07.2020 08:37

3.3 Durchführung und Prototypen

Zur Durchführung des Tests wurden zwei Prototypen entworfen. Dazu wurde die in Abb. 3.4 gezeigte Ordnerstruktur aufgebaut.

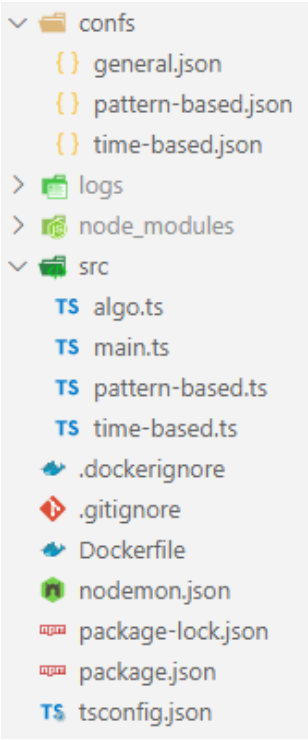

Abbildung 3.4: Struktur des Projekts der Prototypen

Die Prototypen basieren auf *Node.js*® [42] in der Version 12.18.4. Als Programmiersprache wurde bei der Entwicklung auf *TypeScript* [43] gesetzt. Dadurch war es möglich, ein typisiertes und objektorientiertes Klassenmodell zu nutzen. Die genutzten Klassen werden in dem Klassendiagramm in Abb. 3.5 gezeigt.

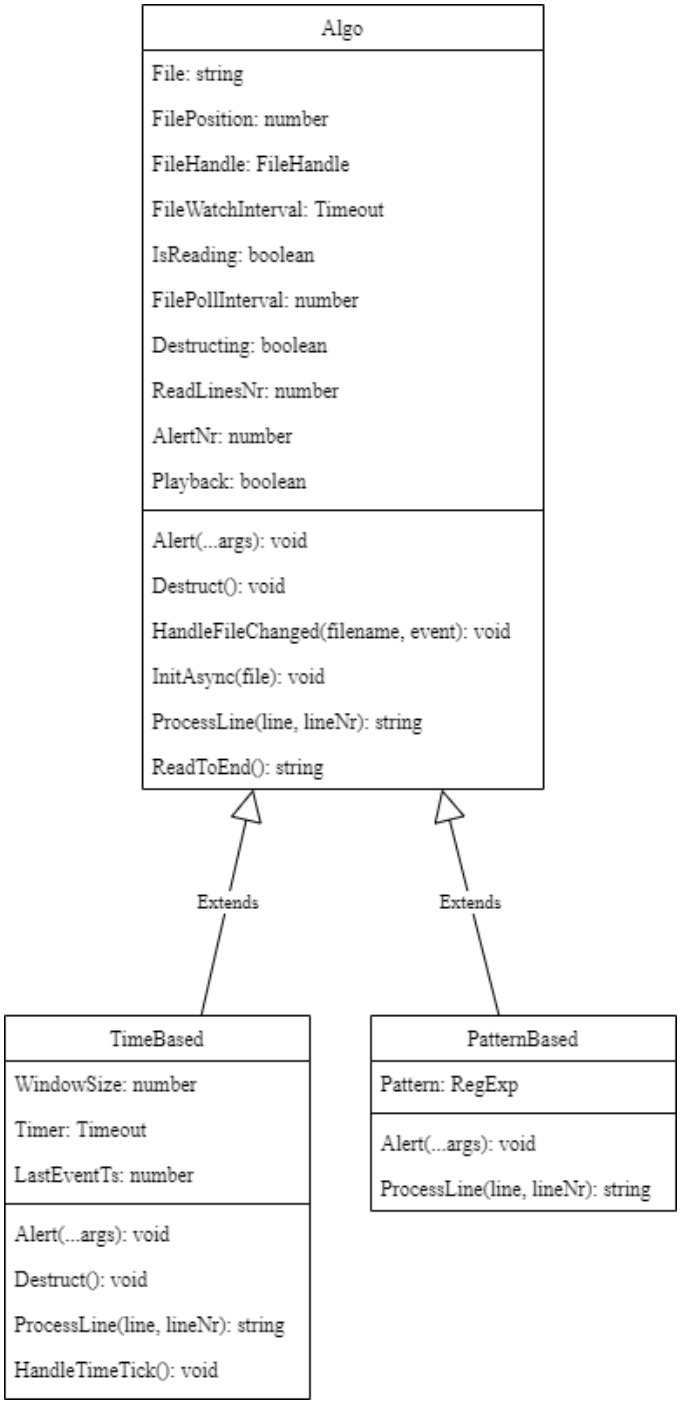

Abbildung 3.5: Klassendiagramm der Prototypen

Die Hauptarbeit ist in der Basisklasse *Algo* definiert. Diese Klasse enthält das asynchrone Verarbeiten der gegebenen Logdatei. Durch den Parameter *Playback* wird bestimmt, ob die Logdatei vollständig eingelesen werden soll. Sollte dieser Parameter nicht aktiviert sein, so wird auf neue Dateiinhalte gewartet. Sobald neue Dateiinhalte gefunden wurden, werden diese zeilenweise eingelesen. Dazu wird die Methode *ReadToEnd* aufgerufen. Diese

teilt den neuen Dateiinhalt in Zeilen auf und ruft für jede Zeile die Methode „ProcessLine“ auf. Die beiden Unterklassen *PatternBased* und *TimeBased* fügen eigene Logik hinzu und überschreiben die *ProcessLine* Methode. Das in diesem Test untersuchte Verfahren basiert folgendermaßen auf der Methode *ProcessLine* der Klasse *PatternBased*.

```
1   import { Algo } from "./algo";
    You, a few seconds ago | 1 author (You)
2   export class PatternBased extends Algo {
3     protected _Pattern: RegExp;
4     public get Pattern() { return this._Pattern; }
5     public set Pattern(v: RegExp) { this._Pattern = v; }
6
7     constructor(file: string, playback: boolean, filePollInterval: number, pattern: RegExp) {
8       super(file, playback, filePollInterval);
9       this._Pattern = pattern;
10    }
11
12    ProcessLine(line: string, lineNr: number) {
13      line = super.ProcessLine(line, lineNr);
14      if(line.match(this.Pattern)) {
15        this.Alert(lineNr, line);
16      }
17      return line;
18    }
19
20    Alert(...args: any[]) {
21      super.Alert(...args);
22      console.warn("\x1b[0m", `[${(new Date()).toISOString()}] Pattern based ALERT! Pattern ${this.Pattern}
        found in line `,  args[0], "\x1b[34m", args[1], "\x1b[0m");
23    }
24  }
```

Abbildung 3.6: Quellcode des Prototypen zur Mustererkennung

Wie in Abb. 3.6 sichtbar, wird lediglich die untersuchte Zeile auf das Muster hin untersucht, welches als Anomalie konfiguriert wurde. Die Konfiguration ist in drei Dateien ausgelagert. Dadurch lassen sich allgemeine und algorithmusspezifische Konfigurationen einstellen.

Die Abb. 3.7 zeigt die drei genutzten Konfigurationen zur Durchführung des Tests. Diese sind wie folgt zu interpretieren:

Abbildung 3.7: Konfiguration der Prototypen

Die *general.json* enthält die allgemeinen Einstellungen. Die wichtigsten sind *pollInterval* und *playback*. Durch ein *pollInterval* von 250 wird die Logdatei alle 250 Millisekunden auf Änderungen untersucht. Die Aktivierung von *playback* simuliert das vergangene Schreiben der Logdatei. Dadurch werden die bereits geschrieben Zeilen neu eingelesen, womit rückschließend betrachtet werden kann, wie der Algorithmus entschieden hätte.

In der *pattern-based.json* sind die Einstellungen des Algorithmus gespeichert, welcher eine Anomalie-Erkennung basierend auf einer Mustererkennung durchführt. Einstellbar ist, ob dieses Verfahren genutzt werden soll und auf welchen regulären Ausdruck die Logeinträge untersucht werden sollen. Der Parameter *flags* kann benutzt werden, um z.B. die Groß- und Kleinschreibung nicht zu beachten.

Zuletzt enthält auch die *time-based.json* Einstellungen. Diese konfigurieren den Algorithmus, welcher prüft, ob noch regelmäßig Logeinträge erstellt werden. Normalerweise macht der Algorithmus dies, indem er bei jedem Aufruf von *ProcessLine* den aktuellen Zeitstempel speichert. Alle *pollInterval* Millisekunden wird geprüft, ob der gespeicherte Zeitstempel mehr als *windowSize* Millisekunden von dem aktuellen Zeitstempel entfernt ist. Dadurch, dass *playback* in den allgemeinen Einstellungen aktiviert wurde, ist dies jedoch nicht mehr der Fall. Durch die Aktivierung dieses Parameters ändert der Algorithmus seine Methodik. Anstatt alle *pollInterval* Millisekunden zu prüfen, ob noch alles stimmt, tut er dies bei jeder neu eingelesenen Zeile. Dazu speichert der Algorithmus anstatt dem aktuellen Zeitstempel den Zeitstempel, der in dem Logeintrag vorhanden ist (siehe Abb. 2.4, 3.2 oder 3.3), und vergleicht ihn mit dem letzten eingelesenen Zeitstempel.

Abb. 3.8 zeigt einen kleinen Teil der Ausgabe, welcher von den Prototypen erstellt wurde. Sie zeigt Alarme basierend auf einer Anomalie-Erkennung per Muster und per Zeitfenster. Alle gezeigten Alarme lassen sich in den Tabellen 3.1 und A.3 wiederfinden.

```
 Pattern based ALERT! Pattern /.*3 error-msgs*/ found in line  46275  2020-07-09 13:48:13,323 INFO  [Thread-1] de.telekom.lightBs
 Pattern based ALERT! Pattern /.*3 error-msgs*/ found in line  46295  2020-07-09 14:03:13,263 INFO  [Thread-1] de.telekom.lightBs
 Pattern based ALERT! Pattern /.*3 error-msgs*/ found in line  70136  2020-07-23 09:08:05,219 INFO  [Thread-1] de.telekom.lightBs
Done reading D:\Dokumente\Web-Projekte\Typescript\Bachelor\logs\RdqData.log.2 with 72112 nr of lines, created 24 alerts!
Time based ALERT! Last entry was 9.97 minutes ago, that is 3.97 minutes too late. Window size is 6.00 minutes.
Done reading D:\Dokumente\Web-Projekte\Typescript\Bachelor\logs\RdqData.log.2 with 72112 nr of lines, created 1 alerts!
 Pattern based ALERT! Pattern /.*3 error-msgs*/ found in line  17683  2020-05-12 08:43:03,787 INFO  [Thread-1] de.telekom.lightBs
 Pattern based ALERT! Pattern /.*3 error-msgs*/ found in line  23763  2020-05-15 21:08:13,188 INFO  [Thread-1] de.telekom.lightBs
 Pattern based ALERT! Pattern /.*3 error-msgs*/ found in line  23771  2020-05-15 21:13:14,131 INFO  [Thread-1] de.telekom.lightBs
```

Abbildung 3.8: Teilausgabe der Prototypen

Die Durchführung des Tests beträgt ca. 25 Sekunden auf einem nicht speziell konfigurierten Desktop-Computer. Die Prototypen wurden dafür in einem Docker-Container ausgeführt. Dieser Docker-Container basiert auf dem offiziellen Node-Docker-Image [44]. Das Verschieben auf ein leistungsstärkeres System stellt in Zukunft somit kein Problem mehr dar.

3.4 Ergebnisse

Aus Tabelle A.4 wird deutlich, dass 12 der 13 Störungen erkannt wurden. Die Störung 8 vom 24.07.2020 08:29 - 24.07.2020 08:35 wurde nicht entdeckt. Der Grund ist, wie in Tabelle 3.1 gezeigt, dass der Adapter am 23.07.2020 zu den Zeiten 08:13, 08:18 und 08:23 ausgefallen ist. Dies wird erkenntlich daran, dass zu dieser Zeit keine Logeinträge des Adapters erstellt wurden. Wäre der Adapter nicht ausgefallen, hätte er diese Ausfälle erkannt, was einer Abfangrate von 100% entsprechen würde. Als Abfangrate wird in dieser Arbeit die Rate definiert, welche die Aussage über echte Ausfälle und davon auch erkannte Ausfälle trifft. Würde man eine Konfusionsmatrix nutzen, wäre dies die Richtig-Positiv-Rate.

Bei der Auswertung der Ergebnisse wird keine Konfusionsmatrix genutzt, die auf den analysierten Zeilenwerten basiert. Stelle man sich eine solche Konfusionsmatrix vor, so müsste man die Anzahl der aufgetretenen Alarme in die „Positiv“-Spalte schreiben. Diese würde die Werte 116 „True-Positive“ und 14 „False-Positive“ enthalten. Diese Spalte zeigt somit nur Alarmierungen an, genauer gesagt einzelne Zeitpunkte. Denn ein Ausfall existiert immer über einen bestimmten Zeitraum; der Alarm ist also nur eine Momentaufnahme. Über die Aussagekraft dieser Spalte lässt sich noch streiten.

Die „Negativ“-Spalte wäre jedoch immer aussagelos. Denn sie müsste widerspiegeln, wie oft der Algorithmus angegeben hat, keine Anomalie und daraus folgend keinen Ausfall zu erkennen. Doch das ist nicht das, was dieser Algorithmus tun soll: Er soll nicht erkennen, ob es keine Störung gibt oder wie lange diese andauert. Wenn ein Wert für diese Daten erhoben werden müsste, könnte dieser z.B. die Anzahl aller Zeilen angeben, die analysiert wurden, aber keine Anomalie fanden. Doch dadurch wäre der Wert für „True Negative“ 652435 (652551 Zeilen – 116 Anomalie-Erkennungen). Daraus würde folgen, dass der Algorithmus 652435 Mal richtigerweise keinen Ausfall gemeldet hat. Das ist jedoch nicht korrekt, denn es wurden auch die fünf Logeinträge vor der erkannten Anomalie als „Nicht-Ausfall“ bewertet. Diese entstanden im selben Adapter-Durchlauf, in dem die Anomalie erkannt wurde, welche auf einen Ausfall hindeutet. Das bedeutet jedoch nicht, dass diese Logeinträge zu den „False Negative“-Werten gehören. Der Algorithmus hat richtigerweise erkannt, dass das Muster (welches eine Anomalie erkennt) nicht in dem Logeintrag vorhanden war.

Der „False Negative“ Wert wäre im vorher angesprochenen Fall Null. Denn es wurde niemals angenommen, dass es keinen Ausfall gibt, obwohl es einen gab. Doch wie im vorherigen Abschnitt beschrieben, gab es einen Ausfall, der nicht erkannt wurde. Zu diesem Ausfall gab es keine Logeinträge. Folglich gab es keine Zeilen, die analysiert wurden. Somit konnte nicht geschlossen werden, dass eine Anomalie vorhanden ist. Diese Null würde dementsprechend so interpretiert werden, dass der Algorithmus oder das Verfahren jeden Ausfall erkannt hat. Richtig ist jedoch, dass er lediglich jeden für ihn möglich erkennbaren Ausfall erkannt hat.

Eine Möglichkeit zur Nutzung einer Konfusionsmatrix gibt es jedoch. Dafür werden die Zeiträume aus Tabelle A.3 genutzt. Dabei entspricht jeder Zeitraum einer eins. Die Zeiträume werden unterteilt in

1. Zeiträume, in den angenommen wurde, es gibt einen Ausfall

2. Zeiträume, in denen angenommen wurde, es gibt keinen Ausfall

Diese werden dann wiederum aufgeteilt in Zeiträume, in denen es tatsächlich einen Ausfall oder keinen Ausfall gab. Die Zeiträume, von denen angenommen wurde, dass es keinen Ausfall gab, sind alle Zeiträume zwischen den Zeiträumen, in denen durch einen oder mehrere Alarme angenommen wurde, dass es einen Ausfall gibt.

Tabelle 3.2: Konfusionsmatrix ohne Anomalie-Schwellwert

		Angenommen		
		Ausfall	**Nicht-Ausfall**	**Summe**
Echt	**Ausfall**	12	1	13
	Nicht-Ausfall	11	23	34
	Summe	23	24	47

Die gesamte Anzahl von 47 errechnet sich aus den 23 erwarteten Ausfallzeiträumen, den 22 Zeiträumen, die dazwischen liegen, und den zwei Zeiträumen, die vor dem ersten und nach dem letzten erwarteten Ausfallzeitraum liegen.

Aus dieser Konfusionsmatrix lassen sich nun folgende Werte errechnen:

Tabelle 3.3: Abgeleitete Werte der Konfusionsmatrix ohne Anomalie-Schwellwert

Name	**Wert**	**Formel**
Richtig-Positiv-Rate	0.9231	$TPR = \frac{TP}{TP+FN}$
Richtig-Negativ-Rate	0.6765	$TNR = \frac{TN}{FP+TN}$
Relevanz	0.5217	$PPV = \frac{TP}{TP+FP}$
Segreganz	0.9583	$NPV = \frac{TN}{TN+FN}$
Falsch-Positiv-Rate	0.3235	$FPR = \frac{FP}{FP+TN}$
positive Falschklassifikationsrate	0.4783	$FDR = \frac{FP}{FP+TP}$
Falsch-Negativ-Rate	0.0769	$FNR = \frac{FN}{FN+TP}$
Vertrauenswahrscheinlichkeit	0.7447	$ACC = \frac{TP+TN}{TP+TN+FP+FN}$
F-Maß	0.6667	$F1 = \frac{2TP}{2TP+FP+FN}$
MCC	0.5365	$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$

Der wichtigste Wert ist die Richtig-Positiv-Rate. Sie zeigt das Verhältnis zwischen Zeitfenstern, in denen es wirklich Ausfälle gab und in denen Ausfälle richtig als solche angenommen wurden. In diesem Test wurden dementsprechend 92,31% aller Ausfälle erkannt. Dies entspricht dem geforderten Wert (> 90%) aus Kapitel 1.2. Was nicht den geforderten Werten entspricht, ist Relevanz (siehe Tabelle 3.3). Sie beschreibt, wie oft der Zeitraum

eines angenommenen Ausfalls auch wirklich stimmt. In diesem Fall sind das nur 52,17%, gefordert waren mehr als 80%.

Mit diesen Daten können nun die Fragen aus Kapitel 3.2.2 beantwortet werden.

Wie viele Ausfälle wurden erkannt?

Es wurden 12 Ausfälle von diesem Verfahren erkannt. Ein Ausfall konnte nicht erkannt werden. In dem Zeitpunkt des nicht erkannten Ausfalls schrieb der Adapter keine Logeinträge in die überwachte Logdatei. Aufgrund dessen konnte keine Anomalie erkannt und somit kein Alarm erstellt werden. Der Grund ist wahrscheinlich, dass der Adapter zu dieser Zeit nicht (erfolgreich) gestartet ist.

Wie viele vorhergesagte Ausfälle sind korrekt?

Von den vorhergesagte 23 Ausfällen sind 12 korrekt und 11 nicht korrekt. Somit ergibt sich die errechnete Relevanz von 52,17%. Eine erste Anomalie deutet dementsprechend zu 52,17% auf einen Ausfall hin. Nach der zweiten (ohne Abstände zueinander) folgenden Anomalie beträgt die Relevanz bereits 92,31% und nach der dritten folgenden Anomalie sind es 100%.

Wie viele Alarme entstanden in den Zeiträumen der Störungen?

Die Anzahl der Alarme ist in der Tabelle A.3 dargestellt. Daraus wird schnell erkennbar, dass die meisten (10 aus 11) Fehlalarmen nur eine Anomalie erkannt haben, während die korrekten Alarme mindestens drei Anomalien für diesen Zeitraum detektierten. Wie daher und in der Antwort der vorherigen Frage hervorgeht, müssen erst drei Anomalien aufeinanderfolgen, um mit einer hundertprozentigen Sicherheit auszuschließen, dass es sich um keinen Fehlalarm handelt.

Würde man folglich einen Schwellwert erschaffen, der bestimmt, nach wie vielen aufeinanderfolgenden Anomalien ein Ausfall gemeldet werden soll, so könnte man die Relevanz erhöhen und somit die geforderten Ziele doch erreichen. Deshalb werden folgend zwei neue Konfusionsmatrizen erzeugt sowie die dazugehörigen Werte ausgerechnet. Sie sollen zeigen, was geschieht, wenn ein solcher Schwellwert hinzugezogen wird.

Betrachtet man den neuen Schwellwert und die bisher ausgewerteten Ergebnisse, wird klar, dass der Schwellwert für die aktuellen Ergebnisse eins beträgt. Denn nach einer erkannten Anomalie wurde ein Alarm ausgelöst. Es wurde angenommen, dass die überwachte Anwendung LightBSM ausgefallen ist. Die Konfusionsmatrix mit einem Schwellwert von

zwei aufeinanderfolgenden Anomalien zum Erstellen eines Alarmes sieht wie folgt (in Tabelle 3.4 gezeigt) aus.

Tabelle 3.4: Konfusionsmatrix mit Anomalie-Schwellwert = 2

		Angenommen		
		Ausfall	**Nicht-Ausfall**	**Summe**
Echt	**Ausfall**	12	1	13
	Nicht-Ausfall	1	13	14
	Summe	13	14	27

Um die Konfusionsmatrix (siehe Tabelle 3.4) mit einem Anomalie-Schwellwert von zwei zu erstellen, wurden alle Alarme aus Tabelle A.3 mit einer Anzahl an Anomalien kleiner zwei entfernt. Somit bleiben nur noch der eine falsch angenommene Ausfall 14 (siehe Tabelle A.3) und alle 12 richtig angenommenen Ausfälle. Das ergibt somit 13 angenommene Ausfälle. Außerdem verringert sich die Anzahl der richtig angenommenen Nicht-Ausfälle von 23 (RN) auf 12.

Um die Konfusionsmatrix (siehe Tabelle 3.5) mit einem Anomalie-Schwellwert von drei zu erstellen, wurden alle Alarme aus Tabelle A.3 mit einer Anzahl an Anomalien kleiner drei entfernt. Somit bleiben nur noch alle 12 richtig angenommenen Ausfälle. Außerdem verringert sich die Anzahl der richtig angenommenen Nicht-Ausfälle von 23 (RN) auf 12.

Tabelle 3.5: Konfusionsmatrix mit Anomalie-Schwellwert = 3

		Angenommen		
		Ausfall	**Nicht-Ausfall**	**Summe**
Echt	**Ausfall**	12	1	13
	Nicht-Ausfall	0	12	12
	Summe	12	13	25

Mit diesen Werten lassen sich die neuen Raten ausrechnen. Die folgende Tabelle 3.6 zeigt alle errechneten Werte für die Schwellwerte eins bis drei.

Betrachtet man das Verhalten beim Einsatz von den genannten Anomalie-Schwellwerten (siehe Tabelle 3.6), so wird deutlich, dass sich die Werte stark von eins zu zwei und nur leicht von zwei zu drei verändern. Die Werte, die sich nie ändern, sind die Richtig-Positiv-Rate und die Falsch-Negativ-Rate. Der Grund ist, dass sich die „True Positives“ von 12 zusammen mit den „False Negatives“ von eins nicht ändern. Somit ergibt sich

immer 13 als Summe. Der Wert, welcher sich insgesamt nur kaum verändert hat, ist die Segreganz. Dieser Wert zeigt das Verhältnis zwischen richtig und falsch angenommenen Nicht-Ausfall-Zeitfenstern. Er nimmt leicht ab, da die „False Negatives“ von eins gleich bleiben und die „True Negatives“ immer vergleichsweise hoch bleiben. Durch die Abnahme der „True Negatives“, wegen Aussortierung durch den gesetzten Schwellwert wird auch die Segreganz kleiner. Mit diesen drei gleich oder fast gleich bleibenden Werten lassen sich folgende gemeinsame Aussagen treffen. Unabhängig vom Schwellwert wurden immer 92,31% aller Ausfälle erkannt. Somit wurden 7.69% aller Ausfälle nicht erkannt und alle angenommenen Nicht-Ausfälle wurden zu über 90% korrekt als solche erkannt.

Tabelle 3.6: Abgeleitete Werte der Konfusionsmatrizen mit Anomalie-Schwellwerten 1 - 3 (siehe 3.2,3.4,3.5)

Name	**Anomalie-Schwellwert = 1**	**Anomalie-Schwellwert = 2**	**Anomalie-Schwellwert = 3**
Richtig-Positiv-Rate	0.9231	0.9231	0.9231
Richtig-Negativ-Rate	0.6765	0.9286	1
Relevanz	0.5217	0.9231	1
Segreganz	0.9583	0.9286	0.9231
Falsch-Positiv-Rate	0.3235	0.0714	0
positive Falschklassifikationsrate	0.4783	0.0769	0
Falsch-Negativ-Rate	0.0769	0.0769	0.0769
Vertrauenswahrscheinlichkeit	0.7447	0.9259	0.96
F-Maß	0.6667	0.9231	0.96
MCC	0.5365	0.8516	0.9231

Die übrig bleibenden veränderten Werte sind die Richtig-Negativ-Rate, Relevanz, Falsch-Positiv-Rate, positive Falschklassifikationsrate, Falsch-Negativ-Rate, Vertrauenswahrscheinlichkeit, der F-Maß und der MCC.

Da der F-Maß nur ein gewichteter Durchschnitt zwischen Richtig-Positiv-Rate und Relevanz ist und die Richtig-Positiv-Rate gleich bleibt, während die Relevanz steigt, wächst auch der F-Maß. Neben dem F-Maß bewertet zusätzlich der MCC die Qualität des Klassifizierungsverfahrens beim Maschinellen Lernen. Das Klassifizierungsverfahren ist in diesem Fall das parametrisierte Verfahren zur Erkennung von Anomalien. In dieser Arbeit wurden in Kapitel 3.2.2 Zielwerte für das Verfahren festgelegt. An dem Erfüllen oder Nichterfüllen dieser Werte kann festgestellt werden, wie zuverlässig das Verfahren funktioniert. Doch neben diesen festen Werten und den darauf folgendem „Erfüllt oder nicht Erfüllt“, sagen uns der F-Maß und der MCC, wie gut dieses Verfahren generell funktioniert. Der F-Maß ist

dabei nur auf die positive Klasse, in diesem Fall das Erkennen eines Anomalie-Zeitraumes ausgelegt. Er bewertet zwischen 0 und 1, wie zuverlässig das Verfahren diese Anomalien erkennt. Er bewertet nicht, wie zuverlässig Nicht-Anomalien erkannt werden. Im Vergleich dazu gibt der MCC eine Auskunft über die Qualität der positiven und negativen Klassen. Sollte ein Verfahren besonders gut die positive Klasse erkennen, aber extrem schlecht die negative Klasse, so wäre der F-Maß trotzdem sehr hoch, während der MCC sehr niedrig wäre. Der MCC reicht dabei von -1 bis 1. Wobei -1 angibt, dass das Verfahren stark inkonsistent zwischen angenommener Klasse und tatsächlicher Klasse ist. 0 gibt an, dass das Verfahren willkürlich oder sogar zufällig zwischen den Klassen unterscheidet und 1 angibt, dass das Verfahren perfekt zwischen den Klassen unterscheiden kann. [28]

Mit einem F-Maß von 0.6667 und einem MCC von 0.5365 bei einem Anomalie-Schwellwert von 1 sagen diese Werte aus, dass das Verfahren nicht perfekt ist und besser Anomalien (positive Klasse) erkennt als Nicht-Anomalien (negative Klasse). Durch den starken Anstieg des F-Maß und dem MCC wird deutlich, dass auch das Verfahren viel besser geworden ist bzw. die Klassifizierung besser funktioniert hat. Dies wird durch die Relevanz von 92,31% bestätigt. Sie hat sich um 76,94% verbessert. Dementsprechend sind ab einem Anomalie-Schwellwert von zwei 92,31% aller vorhergesagten Ausfall-Zeitfenster korrekt. Ab einem Anomalie-Schwellwert von drei beträgt die Relevanz sogar 100%, womit jeder angenommene Ausfall-Zeitraum tatsächlich einer war. Was darin endet, dass jeder Alarm, der erstellt worden wäre, einen Ausfall zugrunde liegen hätte. Mit dieser hohen Relevanz gibt es absolut keine Ausrede mehr, die Alarme zu ignorieren oder sie als falsch abzustempeln. Denn der einzige verbleibende Fehler des Verfahrens mit einem Anomalie-Schwellwert von drei ist, dass ein einziger Ausfall nicht erkannt wurde.

An dieser Stelle soll aufgeklärt werden, warum die Werte für Richtig-Positiv-Rate, Richtig-Negativ-Rate, Relevanz und Vertrauenswahrscheinlichkeit nicht generell von schlecht bis perfekt bewertet wurden. Man stelle sich ein Verfahren vor, welches Banküberweisungen auf Betrüger untersuchen soll. Dabei wäre alles unter 100% Empfindlichkeit (Richtig-Positiv-Rate) unakzeptabel. Denn das Verfahren muss so empfindlich sein, dass jeder Betrug erkannt wird. Das muss nicht heißen, dass jeder angenommene Betrug auch wirklich ein Betrug sein muss. Aber jeder wirkliche Betrug muss auch angenommen worden sein. Verändert man das Szenario zu einem Verfahren, welches Börsenkurse als steigend oder nicht steigend vorhersagen soll, so würde eine Empfindlichkeit von über 50% das Verfahren perfekt und den Erfinder reich machen. Denn wenn über 50% der angenommenen steigenden Aktien wirklich gestiegen sind, würde das Verfahren schlussendlich mehr Gewinne als Verluste machen. Dieses Beispiel erstreckt sich über alle vorher vorgestellten Werte. Es kommt deshalb immer auf das Szenario und die Ansprüche des Anwenders an. In diesem Testszenario sind das die Mitarbeiter im Leitstand, welche 24 Stunden am Tag und 7 Tage die Woche Anwendungen in der Deutschen Telekom IT überwachen. Im Rahmen dieser Arbeit wurde nicht abgestimmt, welche Werte nun schlecht, gut oder perfekt wären. Aus diesem Grund wurde darüber nicht geurteilt und lediglich die Ziele und Ergebnisse wurden miteinander verglichen.

3.5 Evaluation

Der Schwellwert von drei Anomalien entspricht in diesem Testszenario zehn Minuten. Denn wenn zum Zeitpunkt x eine Anomalie erkannt wurde, ist es zu 52,174% sicher, dass es einen Ausfall gibt. Wenn nun fünf Minuten später, zum Zeitpunkt $x1$ eine weitere Anomalie erkannt wird, sind es schon 92,308%, und nach der dritten Anomalie (zu $x2$) ist es dann zu 100% ein Ausfall. Dementsprechend wird für diesen Fall ein Schwellwert von zwei Anomalien zum Erstellen eines Alarmes vorgeschlagen. Denn dadurch entsteht eine maximale Störung von zehn Minuten und die Relevanz steigt von 52,17% auf 92,31%. Zehn Minuten deswegen, da hier die überwachte Anwendung LightBSM sofort nach einem erfolgreichen Durchlauf des Adapters ausfallen kann. Der Adapter startet erneut in fünf Minuten, erstellt die erste Anomalie und die zweite folgt das nächste Mal fünf Minuten später. Dadurch ergibt sich die maximale Störzeit von zehn Minuten. Diese kann gegen unendlich wandern, wenn der Adapter keine Logeinträge mehr erstellt. Deshalb sollte ein zweites Verfahren hinzugefügt werden, welches überwacht, ob die überwachte Logdatei des Adapters noch regelmäßig Einträge erhält. Erhält sie keine neuen Einträge in einem vorgegebenen Intervall, so kann die überwachte Anwendung LightBSM gestört sein, muss sie aber nicht. Definitiv gestört ist dann jedoch die Anwendung, deren Logdatei überwacht wird. Deshalb empfiehlt es sich, parallel noch die Logdatei von LightBSM zu überwachen zusätzlich, zu den Logdateien der anderen Adaptern.

Neben der erfassten Prävalenz von 52,174% sollte jedoch die „True Positive Rate“ von 92,308% nicht vergessen werden. Denn dieses Verfahren hat somit fast alle Ausfälle erkannt. Die einzige nicht erkannte Störung konnte dieses Verfahren durch das Fehlen von Logeinträgen nicht erkennen. Sollte dieser Messpunkt entfernt werden, ergäbe sich somit eine „True Positive Rate“ von 100%. Und auch die erfassten Anomalien, hinter denen schlussendlich kein Ausfall der überwachten Anwendung LightBSM stand, deuten darauf hin, dass etwas nicht stimmt. Eine genauere Untersuchung seitens der verantwortlichen Entwickler von LightBSM sollte in der Zukunft durchgeführt werden. Denn selbst wenn LightBSM nicht für alle Adapter gestört war und hauptsächlich verfügbar war, konnten die Daten des analysierten Adapters nicht übermittelt werden. Somit entsteht eine Datenlücke, die nicht hätte sein sollen, da die Anwendung LightBSM verfügbar war. Ob LightBSM durch zu hohe Lasten die Anfrage nicht annehmen konnte oder es ein Problem bei der Verarbeitung der empfangenen Daten gab, ist nicht bekannt. Die Ursachen sollen hier nicht weiter spekuliert werden. Klar ist jedoch, dass zu den 11 Fehlalarmen irgendetwas nicht stimmte und die Fehlalarme außerhalb des Kontextes der generellen Verfügbarkeit vielleicht gar keine Fehlalarme gewesen wären.

Die Sinnhaftigkeit des vorgeschlagenen und getesteten Verfahrens wurde bestätigt. Die Nachteile des Verfahrens entstehen durch das Fehlen von neuen Logeinträgen und dem sofortigen Erstellen eines Alarms bei einer Anomalie. Diese können jedoch durch das Konfigurieren eines Schwellwertes und eine zweite Überwachung der Logdateien eingegrenzt und vermindert, wenn nicht sogar beseitigt werden.

4 Schlussbetrachtung

4.1 Zusammenfassung

Die auf JHipster basierende Anwendung LightBSM empfängt täglich knapp 5,8 Millionen Datenpunkte, verarbeitet diese zu über 15000 Events, verschickt Alarme und stellt alle gesammelten Daten verschieden visuell dar. Es ist eine Anwendung zur großteiligen Überwachung der Deutschen Telekom IT. Diese Daten werden von 57 Adaptern gesammelt, verarbeitet und an LightBSM verschickt. Es gibt jedoch keinen Adapter und keine Anwendung, die LightBSM überwacht. Eine Grundlage für eine solche Überwachung liefern die Logdateien von LightBSM selbst und die verschiedenen Logdateien der Adapter. Aus diesen Logeinträgen lässt sich schließen, ob die Anwendung LightBSM noch verfügbar ist oder nicht. Diese Arbeit kreiert und untersucht ein Verfahren zur Analyse dieser Logdateien auf Anomalien. Das Ziel ist es, aus dem Auftreten der Anomalien die Verfügbarkeit der überwachten Anwendung zu korrelieren. Dabei wird versucht, die Struktur und das Format der Logdatei zu vernachlässigen, da diese beiden Faktoren sich von Logdatei zu Logdatei und von Anwendung zu Anwendung unterschieden.

Derzeit wird die Verfügbarkeit der Anwendung LightBSM halbautomatisiert überwacht. Ein Programm ruft über einen wget-Befehl jede Minute die Webseite von LightBSM auf. Danach wird der Statuscode auf 200 (OK) überprüft. Sollte der Wert fünfmal hintereinander nicht 200 (OK) entsprechen, so wird eine E-Mail versendet und LightBSM wird neu gestartet. Eines der Probleme bei dieser Überwachung ist, dass nicht überprüft wird, ob der Login, die Datenbankverbindungen oder die APIs noch funktionieren. All diese Schritte werden jedoch beim Durchlauf eines Adapters genutzt, sodass dies bei einem Fehler in den Logdateien eingetragen wird. Deswegen gibt es ein Team, welches 24/7 überprüft, ob die Anwendung noch verfügbar ist. Dazu ruft es das Webportal von LightBSM manuell auf und überprüft die Funktionalitäten. Mithilfe des vorgestellten Verfahrens kann dieses Team abgelöst werden und die Überwachung voll automatisiert stattfinden.

Das Verfahren ist dabei sehr leichtgewichtig aufgebaut. Es untersucht neu erstellte Logeinträge auf einen vorher definierten regulären Ausdruck. Sollte eine Übereinstimmung stattfinden, wird dies als Anomalie gedeutet. Der Anwender konfiguriert somit das Verfahren und definiert deutlich, was eine Anomalie ist und was nicht. Eine aufgetretene Anomalie sollte als Ausfall interpretiert werden, durch welchen die Verfügbarkeit der Anwendung gestört ist. Um dies zu testen, wurden die Logdateien eines Adapters rückblickend analysiert. Als größter Schwachpunkt des Verfahrens wurde angenommen, dass es sinnfrei ist, sobald keine neuen Logeinträge entstehen, da es keine Anomalien mehr erkennen kann. Dies bestätigte sich bei den Tests. Aus 13 Ausfällen wurde ein Ausfall nicht

erkannt, da zu dieser Zeit der Adapter ausgefallen war. Um diese Schwäche auszugleichen, wurde vorgeschlagen, ein zweites Verfahren zu entwickeln, welches die Logdatei darauf prüft, ob sie regelmäßig Logeinträge erhält. Aufgrund des Arbeitsumfangs konnten die vorhandenen Logdateien des Adapters untersucht werden, doch eine detaillierte Analyse dieses Verfahrens blieb aus. Die getroffenen 12 von 13 Treffern der Ausfälle und somit 92,31% sind auf den ersten Blick ein Erfolg, doch es stellte sich heraus, dass nur 52,17% der angenommenen Störungen auch wirklich Störungen waren. Bei abgerundet 50% Trefferrate können die alarmierten Mitarbeiter eine Münze werfen, ob es einen Ausfall gibt oder nicht. Und würde die Alarmierung einen automatischen Neustart auslösen, wäre die Anwendung doppelt so häufig durch einen Neustart nicht erreichbar, als sie müsste. Die Anomalien, welche auftraten, obwohl es keinen Ausfall der überwachten Anwendung gab, schlossen darauf, dass die überwachte Anwendung zu diesen Zeitpunkten die Daten nicht empfangen konnte oder der Adapter die Daten nicht versenden konnte. Folglich gab es entweder Störungen in LightBSM, die keinen Ausfall erzeugten oder der Adapter funktionierte nicht richtig. Diese „Fehlanomalien“ konnten leider nicht weiter untersucht werden. Sie sind in Wirklichkeit vielleicht keine Fehler und weisen auf ein unbekanntes anderes Problem hin. Bei genauerer Betrachtung der Ausfallzeiträume und der erfassten Anomalien (in diesen Zeiträumen) wurde sichtbar, dass erst ab drei aufeinanderfolgenden Anomalien ein Ausfall garantiert war. Dadurch entstand somit eine einhundertprozentige Trefferrate. Nach nur zwei aufeinanderfolgenden Anomalien stieg die Trefferrate bereits von den 52,17% auf 93,31%, was eine Steigerung um 76,94% ergibt.

Deswegen wurde empfohlen, dem Verfahren einen Anomalie-Schwellwert hinzuzufügen. Dieser Anomalie-Schwellwert soll dabei angeben, nach wie vielen folgenden Anomalien ein Alarm erstellt und die überwachte Anwendung somit als gestört empfunden werden soll. Da eine Ausführung des Adapters nur alle fünf Minuten stattfindet, empfiehlt sich ein Anomalie-Schwellwert von zwei. Dieser erschafft eine sehr hohe Trefferrate, welche es zulässt, ernst zu nehmende Alarme zu versenden und die Anwendung sogar automatisiert neu zu starten. Des Weiteren empfiehlt es sich, ein zweites Verfahren zu implementieren und zu testen, welches die überwachte Logdatei auf fehlende Einträge untersucht. Bei der kurzen Untersuchung dieses Verfahrens stellte sich schon heraus, dass die Erkennung eines fehlenden Eintrages nicht mit der Verfügbarkeit der überwachten Anwendung korrelierbar war. Denn nur weil in diesem Zeitraum der Adapter ausgefallen ist, muss dies nicht bedeuten, dass die überwachte Anwendung LightBSM auch ausgefallen ist.

4.2 Fazit

Schlussendlich ist es somit sinnvoll, mit dem vorgestellten Verfahren Logdateien auf Anomalien zu untersuchen, um die Verfügbarkeit einer abhängigen Anwendung zu überwachen. Dabei müssen die analysierten Logdateien nicht direkt von der überwachten Anwendung kommen. Zur Erhöhung der Trefferrate sollte ein Anomalie-Schwellwert hinzugefügt und für das Szenario passend eingestellt werden. Je kleiner der Schwellwert desto kleiner die Trefferrate. Doch je größer man den Schwellwert setzt, desto größer

wird der zeitliche Abstand bis zum Alarm und die Zeit der Störung. Es muss bewusst sein, dass das Verfahren nicht funktioniert, wenn keine Logeinträge zum Analysieren erstellt werden. Deshalb sollte ein zweites Verfahren die Logdatei der untersuchten Anwendung auf fehlende Logeinträge untersuchen.

4.3 Ausblick

Nachdem im Rahmen dieser Arbeit das Verfahren zur Analyse von Logdateien zur Erkennung von Ausfällen einer Anwendung untersucht wurde, sind viele weitere Untersuchungen möglich. Unter anderem legt diese Arbeit den Grundbaustein für eine Untersuchung, welche Maschinelles Lernen oder sogar künstliche Intelligenz verwendet. Das in dieser Arbeit vorgestellte und untersuchte Verfahren könnte dabei schon als einfacher Klassifikator dienen. Erweitert könnte damit untersucht werden, wie der vorgeschlagene bestmögliche Anomalie-Schwellwert automatisch generiert wird. Ähnlich oder sogar zusätzlich könnten noch die regulären Ausdrucke bzw. Textmuster untersucht werden, welche am stärksten auf einen Ausfall hindeuten. Ein einfaches anzunehmendes Beispiel dafür wäre das Wort „Error“. Diese Textmuster könnten händisch untersucht, mengenweise getestet oder automatisch verknüpft werden, indem man sich die Logeinträge vor und während eines Ausfalles anschaut und miteinander aggregiert und korreliert. Sollte man nicht in das Thema Maschinelles Lernen oder künstliche Intelligenz einsteigen, aber trotzdem dieses Verfahren weiter untersuchen wollen, so wäre es möglich, das hier gezeigte Verfahren noch einmal mit Logdateien einer zu überwachenden Anwendung durchzuführen. Die Ergebnisse wären mit den Ergebnissen dieser Arbeit vergleichbar, welche die Logdatei der nicht überwachten Anwendung nutzt. Somit ließe sich feststellen, welche Quelle besser geeignet wäre. Dabei kann vielleicht gleichzeitig noch das in dieser Arbeit angeschnittene Verfahren zum Überwachen der Regelmäßigkeit von Logeinträgen getestet werden. Dies war in dieser Arbeit leider nicht machbar, scheint jedoch vor allem in Kombination mit dem Verfahren dieser Arbeit großes Potenzial zu haben.

Literaturverzeichnis

[1] F. E. Grubbs. *Procedures for Detecting Outlying Observations in Samples*. In: *Technometrics* 11.1 (Feb. 1969). ISSN: 0040-1706, 1537-2723. DOI: 10.1080/00401706.1969.10490657 (zitiert auf Seite 3).

[2] D. Hawkins. *Identification of Outliers*. Monographs on Statistics and Applied Probability. Springer Netherlands, 1980. ISBN: 978-94-015-3996-8. DOI: 10.1007/978-94-015-3994-4 (zitiert auf Seite 3).

[3] Barnett und Lewis. *Outliers in Statistical Data 3e*. 3. Aufl. Chichester ; New York: Wiley John + Sons, Feb. 1994. ISBN: 978-0-471-93094-5 (zitiert auf Seite 3).

[4] V. Chandola, A. Banerjee und V. Kumar. *Anomaly Detection: A Survey*. In: *Association for Computing Machinery* 41.3 (Juli 2009). ISSN: 0360-0300. DOI: 10.1145/1541880.1541882 (zitiert auf den Seiten 3–8).

[5] S. Vaishampayan, G. K. Palshikar, M. Apte und V. Z. Attar. *Causality: An Overlooked Aspect in Anomaly Detection*. In: *TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON)*. Okt. 2019. ISBN: 2159-3450. DOI: 10.1109/TENCON.2019.8929238 (zitiert auf den Seiten 3, 9).

[6] *IEEE Standard Classification for Software Anomalies*. In: *IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993)* (Jan. 2010). DOI: 10.1109/IEEESTD.2010.5399061 (zitiert auf Seite 3).

[7] B. Zhao, X. Li, J. Li, J. Zou und Y. Liu. *An Area-Context-Based Credibility Detection for Big Data in IoT*. In: *Hindawi* 2020 (Jan. 2020). Hrsg. von M. Picone. ISSN: 1574-017X. DOI: 10.1155/2020/5068731 (zitiert auf den Seiten 4, 9).

[8] D. Reinsel, J. Gantz und J. Rydning. *The Digitization of the World from Edge to Core*. In: (2018) (zitiert auf den Seiten 6, 9).

[9] C. Isaksson und M. H. Dunham. *A Comparative Study of Outlier Detection Algorithms*. In: *Machine Learning and Data Mining in Pattern Recognition*. Hrsg. von P. Perner. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009. ISBN: 978-3-642-03070-3. DOI: 10.1007/978-3-642-03070-3_33 (zitiert auf den Seiten 6, 8).

[10] Z. Nazari, S.-M. Yu, D. Kang und Y. Kawachi. *Comparative Study of Outlier Detection Algorithms for Machine Learning*. In: *Proceedings of the 2018 2nd International Conference on Deep Learning Technologies*. ICDLT '18. New York, NY, USA: Association for Computing Machinery, Juni 2018. ISBN: 978-1-4503-6473-7. DOI: 10.1145/3234804.3234817 (zitiert auf den Seiten 6, 9).

[11] A. Zimek und P. Filzmoser. *There and back again: Outlier detection between statistical reasoning and data mining algorithms*. In: *WIREs Data Mining and Knowledge Discovery* 8.6 (2018). DOI: 10.1002/widm.1280 (zitiert auf Seite 7).

[12] Y. Zhang, N. Meratnia und P. Havinga. *Outlier Detection Techniques for Wireless Sensor Networks: A Survey*. In: *IEEE Communications Surveys Tutorials* 12.2 (2010). ISSN: 1553-877X. DOI: 10.1109/SURV.2010.021510.00088 (zitiert auf Seite 8).

[13] H. Wang, M. J. Bah und M. Hammad. *Progress in Outlier Detection Techniques: A Survey*. In: *IEEE Access* 7 (2019). ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2932769 (zitiert auf Seite 8).

[14] A. Boukerche, L. Zheng und O. Alfandi. *Outlier Detection: Methods, Models, and Classification*. In: *ACM Computing Surveys* 53.3 (Juni 2020). ISSN: 0360-0300. DOI: 10.1145/3381028 (zitiert auf Seite 8).

[15] K. Leung und C. Leckie. *Unsupervised Anomaly Detection in Network Intrusion Detection Using Clusters*. In: *Proceedings of the Twenty-eighth Australasian Conference on Computer Science*. 2005 (zitiert auf Seite 8).

[16] X. Yang, W. Zhou, N. Shu und H. Zhang. *A Fast and Efficient Local Outlier Detection in Data Streams*. In: *Proceedings of the 2019 International Conference on Image, Video and Signal Processing - IVSP 2019* (2019) (zitiert auf Seite 8).

[17] E. M. Knorr und R. T. Ng. *Algorithms for Mining Distance-Based Outliers in Large Datasets*. In: *Proceedings of the 24rd International Conference on Very Large Data Bases*. VLDB '98. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Aug. 1998. ISBN: 978-1-55860-566-4. (Besucht am 16. 09. 2020) (zitiert auf Seite 8).

[18] J. Dorschel, Hrsg. *Praxishandbuch Big Data: Wirtschaft – Recht – Technik*. Gabler Verlag, 2015. ISBN: 978-3-658-07288-9. DOI: 10.1007/978-3-658-07289-6 (zitiert auf den Seiten 9, 10).

[19] S. King. *Big Data: Potential und Barrieren der Nutzung im Unternehmenskontext*. VS Verlag für Sozialwissenschaften, 2014. ISBN: 978-3-658-06585-0. DOI: 10.1007/978-3-658-06586-7 (zitiert auf den Seiten 10, 11).

[20] S. Müller. *Big Data Analysen: Für den schnellen Einstieg*. De Gruyter Oldenbourg, Sep. 2018. ISBN: 978-3-11-045780-3. URL: https://doi.org/10.1515/9783110457803 (besucht am 17. 09. 2020) (zitiert auf den Seiten 10, 11).

[21] R. Otte, B. Wippermann, S. Schade und V. Otte. *Von Data Mining bis Big Data: Handbuch für die industrielle Praxis*. München: Carl Hanser Verlag GmbH & Co. KG, Juli 2020. ISBN: 978-3-446-45550-4 (zitiert auf den Seiten 10–12).

[22] F. Iafrate. *From Big Data To Smart Data*. London, UK : Hoboken, NJ: John Wiley & Sons, Ltd., März 2015. ISBN: 978-1-84821-755-3 (zitiert auf Seite 10).

[23] B. Engels und H. Goecke. *Big Data in Wirtschaft und Wissenschaft: Eine Bestandsaufnahme*. IW-Analysen, 2019. ISBN: 978-3-602-45624-6. URL: http://hdl.handle.net/10419/201760 (besucht am 17. 09. 2020) (zitiert auf den Seiten 10, 11).

[24] J. Hagedorn, N. Bissantz und P. Mertens. *Data Mining (Datenmustererkennung): Stand der Forschung und Entwicklung*. In: *Wirtschaftsinformatik : WI ; Organ der Fachbereichs Wirtschaftsinformatik der Gesellschaft für Informatik e.V. und der Wissenschaftlichen Kommission Wirtschaftsinformatik im Verband der Hochschullehrer für Betriebswirtschaft e.V*. Wirtschaftsinformatik : WI ; Organ der Fachbereichs Wirtschaftsinformatik der Gesellschaft für Informatik e.V. und der Wissenschaftlichen Kommission Wirtschaftsinformatik im Verband der Hochschullehrer für Betriebswirtschaft e.V. - Wiesbaden : Springer Gabler, ISSN 0937-6429, ZDB-ID 10173626. - Vol. 39.1997, 6, p. 601-612 39.6 (1997) (zitiert auf Seite 12).

[25] W. Ertel. „Maschinelles Lernen und Data Mining“. In: *Grundkurs Künstliche Intelligenz: Eine praxisorientierte Einführung*. Hrsg. von W. Ertel. Computational Intelligence. Wiesbaden: Springer Fachmedien, 2016. ISBN: 978-3-658-13549-2. DOI: 10.1007/978-3-658-13549-2_8 (zitiert auf Seite 12).

[26] P. Ghavami. *Big Data Analytics Methods: Analytics Techniques in Data Mining, Deep Learning and Natural Language Processing*. De Gruyter, Dez. 2019. ISBN: 978-1-5474-0156-7. URL: https://doi.org/10.1515/9781547401567 (besucht am 17. 09. 2020) (zitiert auf Seite 12).

[27] T. A. Runkler. *Data Mining: Modelle und Algorithmen intelligenter Datenanalyse*. 2. Aufl. Computational Intelligence. Springer Vieweg, 2015. ISBN: 978-3-8348-1694-8. DOI: 10.1007/978-3-8348-2171-3 (zitiert auf Seite 12).

[28] D. Chicco und G. Jurman. *The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation*. In: *BMC Genomics* 21 (Dez. 2020). DOI: 10.1186/s12864-019-6413-7 (zitiert auf den Seiten 15, 35).

[29] Bundesamt für Sicherheit in der Informationstechnik. *Studie über die Nutzung von Log- und Monitoringdaten im Rahmen der IT-Frühwarnung und für einen sicheren IT-Betrieb*. 2013. URL: https://www.internet-sicherheit.de/fileadmin/docs/downloads/andere_studien_dokumente/BSI/2013_Logdatenstudie.pdf (zitiert auf den Seiten 15–17).

[30] *Elastic Stack: Elasticsearch, Kibana, Beats & Logstash*. URL: https://www.elastic.co/elastic-stack (besucht am 29. 09. 2020) (zitiert auf Seite 18).

[31] *Industry Leading Log Management | Graylog*. URL: https://www.graylog.org/ (besucht am 30. 09. 2020) (zitiert auf Seite 18).

[32] *Nagios - The Industry Standard In IT Infrastructure Monitoring*. URL: https://www.nagios.org/ (besucht am 30. 09. 2020) (zitiert auf Seite 18).

[33] *LOGalyze - Open Source Log Management Tool, SIEM, Log Analyzer*. URL: http://www.logalyze.com/ (besucht am 30. 09. 2020) (zitiert auf Seite 18).

[34] *Fluentd | Open Source Data Collector | Unified Logging Layer*. URL: https://www.fluentd.org/ (besucht am 30. 09. 2020) (zitiert auf Seite 18).

[35] Datadog. *Cloud Monitoring as a Service | Datadog*. 400. URL: https://www.datadoghq.com/ (besucht am 30. 09. 2020) (zitiert auf Seite 18).

[36] *Security Event Manager – Anzeigen von Ereignisprotokollen im Remote-Modus | SolarWinds*. URL: https://www.solarwinds.com/de/security-event-manager (besucht am 30. 09. 2020) (zitiert auf Seite 18).

[37] *Papertrail - cloud-hosted log management, live in seconds*. URL: https://www.papertrail.com/ (besucht am 30. 09. 2020) (zitiert auf Seite 18).

[38] *Log Analysis | Log Management by Loggly*. URL: https://www.loggly.com/ (besucht am 30. 09. 2020) (zitiert auf Seite 18).

[39] *Sematext: Cloud Monitoring & Management Tools*. URL: https://sematext.com/ (besucht am 30. 09. 2020) (zitiert auf Seite 18).

[40] *Log Management and Analysis Platform, Trusted Worldwide | XPLG*. URL: https://www.xplg.com/home/ (besucht am 30. 09. 2020) (zitiert auf Seite 18).

[41] *JHipster - Full Stack Platform for the Modern Developer!* URL: https://www.jhipster.tech/ (besucht am 28. 09. 2020) (zitiert auf Seite 18).

[42] Node.js. *Node.js*. URL: https://nodejs.org/en/ (besucht am 23. 10. 2020) (zitiert auf Seite 25).

[43] *Typed JavaScript at Any Scale.* URL: https://www.typescriptlang.org/ (besucht am 23. 10. 2020) (zitiert auf Seite 25).

[44] *node - Docker Hub*. URL: https://hub.docker.com/_/node (besucht am 23. 10. 2020) (zitiert auf Seite 29).

Selbstständigkeitserklärung

Hiermit erkläre ich, Tim Pütz, dass die von mir an der *Hochschule für Telekommunikation Leipzig (FH)* eingereichte Abschlussarbeit zum Thema

Generische Verfügbarkeitsanalyse anhand von Logdatei-Anomalieerkennung

selbstständig verfasst wurde und von mir keine anderen als die angegebenen Quellen und Hilfsmittel verwendet wurden.

Leipzig, den 4. November 2020

Tim Pütz

A Anhang

A.1 Große Tabellen

Tabelle A.1: Gemeldete Neustarts von LightBSM zwischen März und Oktober 2020

Neustart	Datum
1	16.03.2020 12:22
2	14.04.2020 21:47
3	15.05.2020 21:12
4	04.06.2020 15:27
5	09.07.2020 11:22
6	09.07.2020 12:12
7	09.07.2020 12:37
8	09.07.2020 13:02
9	09.07.2020 13:37
10	24.07.2020 08:32
11	29.07.2020 12:17
12	26.08.2020 17:37
13	26.08.2020 18:02
14	26.08.2020 18:27
15	26.08.2020 18:52
16	26.08.2020 19:17
17	17.09.2020 15:27

Tabelle A.2: Echte Störungen von LightBSM zwischen März und Oktober 2020

Störung	Zeitraum
1	16.03.2020 12:20 - 16.03.2020 12:22
2	14.04.2020 21:42 - 14.04.2020 21:47
3	15.05.2020 21:09 - 15.05.2020 21:12
4	16.05.2020 23:38 - 16.05.2020 23:48
5	04.06.2020 15:23 - 04.06.2020 15:33
6	09.07.2020 11:14 - 09.07.2020 14:04
7	23.07.2020 09:05 - 23.07.2020 09:12
8	24.07.2020 08:29 - 24.07.2020 08:35
9	27.07.2020 08:40 - 27.07.2020 11:10
10	28.07.2020 11:56 - 27.07.2020 12:02
11	29.07.2020 12:13 - 27.07.2020 12:34
12	26.08.2020 17:32 - 26.08.2020 19:40
13	17.09.2020 15:22 - 17.09.2020 16:03

Tabelle A.3: Musterbasierte Ausfälle zwischen März und Oktober 2020

Ausfall	Ausgefallen	Zeitraum	Anz. Anomalien
1	Falsch	10.03.2020, 13:03	1
2	Richtig	16.03.2020, 12:18	1
3	Falsch	16.03.2020, 13:28	1
4	Falsch	31.03.2020, 10:39	1
5	Falsch	02.04.2020, 17:43	1
6	Richtig	14.04.2020, 21:08 - 14.04.2020, 21:53	10
7	Falsch	16.04.2020, 09:13	1
8	Falsch	12.05.2020, 08:43	1
9	Richtig	15.05.2020, 21:08 - 15.05.2020, 21:18	3
10	Richtig	16.05.2020, 23:38 - 16.05.2020, 23:58	4
11	Richtig	04.06.2020, 15:23 - 04.06.2020, 15:33	3

12	Falsch	08.06.2020, 07:18	1
13	Falsch	09.06.2020, 09:08	1
14	Falsch	04.07.2020, 22:03 - 04.07.2020, 22:08	2
15	Richtig	09.07.2020, 11:18 - 09.07.2020, 14:03	18
16	Richtig	23.07.2020, 09:08	1
17	Richtig	27.07.2020, 08:43 - 27.07.2020, 11:08	26
18	Richtig	28.07.2020, 11:58	1
19	Richtig	29.07.2020, 12:13 - 29.07.2020, 12:33	5
20	Falsch	26.08.2020, 02:03	1
21	Richtig	26.08.2020, 17:33 - 26.08.2020, 19:38	26
22	Falsch	31.08.2020, 02:03	1
23	Richtig	17.09.2020, 15:53 - 17.09.2020, 16:03	3

Tabelle A.4: Geprüfte Störungen von LightBSM zwischen März und Oktober 2020

Störung	**Zeitraum**	**Erkannt**
1	16.03.2020 12:20 - 16.03.2020 12:22	Richtig
2	14.04.2020 21:42 - 14.04.2020 21:47	Richtig
3	15.05.2020 21:09 - 15.05.2020 21:12	Richtig
4	16.05.2020 23:38 - 16.05.2020 23:48	Richtig
5	04.06.2020 15:23 - 04.06.2020 15:33	Richtig
6	09.07.2020 11:14 - 09.07.2020 14:04	Richtig
7	23.07.2020 09:05 - 23.07.2020 09:12	Richtig
8	24.07.2020 08:29 - 24.07.2020 08:35	Falsch
9	27.07.2020 08:40 - 27.07.2020 11:10	Richtig
10	28.07.2020 11:56 - 27.07.2020 12:02	Richtig
11	29.07.2020 12:13 - 27.07.2020 12:34	Richtig
12	26.08.2020 17:32 - 26.08.2020 19:40	Richtig
13	17.09.2020 15:22 - 17.09.2020 16:03	Richtig

A.2 Quellcode des Projekts der Prototypen

```
1   import fs from "fs";
2
3   export class Algo {
4     protected _File!: string;
5     public get File() { return this._File; }
6     public set File(v: string) { this._File = v; }
7
8     protected _FilePosition!: number;
9     public get FilePosition() { return this._FilePosition; }
10    public set FilePosition(v: number) { this._FilePosition = v; }
11
12    protected _FileHandle!: fs.promises.FileHandle;
13    public get FileHandle() { return this._FileHandle; }
14    public set FileHandle(v: fs.promises.FileHandle) { this._FileHandle = v; }
15
16    protected _FileWatchInterval!: NodeJS.Timeout;
17    public get FileWatchInterval() { return this._FileWatchInterval; }
18    public set FileWatchInterval(v: NodeJS.Timeout) { this._FileWatchInterval = v; }
19
20    protected _IsReading!: boolean;
21    public get IsReading() { return this._IsReading; }
22    public set IsReading(v: boolean) { this._IsReading = v; }
23
24    protected _FilePollInterval: number; // tick rate in milliseconds
25    public get FilePollInterval() { return this._FilePollInterval; }
26    public set FilePollInterval(v: number) { this._FilePollInterval = v; }
27
28    protected _Destructing: boolean;
29    public get Destructing() { return this._Destructing; }
30    public set Destructing(v: boolean) { this._Destructing = v; }
31
32    protected _ReadLinesNr: number;
33    public get ReadLinesNr() { return this._ReadLinesNr; }
34    public set ReadLinesNr(v: number) { this._ReadLinesNr = v; }
35
36    protected _AlertNr: number;
37    public get AlertNr() { return this._AlertNr; }
38    public set AlertNr(v: number) { this._AlertNr = v; }
39
40    protected _Playback: boolean;
41    public get Playback() { return this._Playback; }
42    public set Playback(v: boolean) { this._Playback = v; }
43
44    constructor(file: string, playback: boolean, filePollInterval: number) {
45      this._Playback = playback;
46      this._AlertNr = 0;
47      this._Destructing = false;
48      this._ReadLinesNr = 0;
49      this._FilePollInterval = filePollInterval;
50      this._IsReading = false;
51      this._File = file;
52      this._FilePosition = 0;
53      this.InitAsync(file);
54    }
55
```

Abbildung A.1: Quellcode der Algo-Klasse des Prototypen (Teil 1) (src/algo.ts)

```
56  async InitAsync(file: string) {
57    this._FileHandle = await fs.promises.open(this.File, "r");
58    if(!this.Playback) {
59      const stats = await fs.promises.stat(this.File);
60      this._FilePosition = stats.size;
61    }
62
63    this._FileWatchInterval = setInterval( () => {
64      if(!this.IsReading) {
65        this.ReadToEnd();
66      }
67    }, this.FilePollInterval);
68  }
69
70  async ReadToEnd() {
71    if(this.IsReading) {
72      console.warn(`[${(new Date()).toISOString()}] Already reading...file gets faster written than read! Skipping this
      read...`);
73    }
74
75    this.IsReading = true;
76
77    const stats = await fs.promises.stat(this.File);
78    const firstRead = this.FilePosition === 0;
79    const size2Read = stats.size - this.FilePosition;
80    let text = "";
81
82    if(size2Read > 0) {
83
84      let buffer = Buffer.alloc(size2Read, "");
85
86      const r = await this.FileHandle.read(buffer, 0, size2Read, stats.size - size2Read);
87
88      this.FilePosition += r.bytesRead;
89
90      text = r.buffer.toString();
91      if(text && text.length > 0) {
92        this.HandleFileChanged(this.File, "change");
93
94        const splitted = text.split("\n");
95        for (const line of splitted) {
96          this.ReadLinesNr++;
97          this.ProcessLine(line, this.ReadLinesNr);
98
99          if(this.Destructing) { break; }
100       }
101     }
102   }
103
104   if(firstRead) {
105     console.log(`[${(new Date()).toISOString()}] Done reading ${this.File} with ${this.ReadLinesNr} nr of lines,
      created ${this.AlertNr} alerts!`);
106   }
107
108   this.IsReading = false;
109
110   return text;
111 }
```

Abbildung A.2: Quellcode der Algo-Klasse des Prototypen (Teil 2) (src/algo.ts)

```
112
113   ProcessLine(line: string, lineNr: number) {
114     return line;
115   }
116
117   async HandleFileChanged(filename: string, event: string) {
118   }
119
120   Alert(...args: any[]) {
121     this.AlertNr++;
122   }
123
124   async Destruct() {
125     this.Destructing = true;
126     if(this.FileWatchInterval) {
127       clearInterval(this.FileWatchInterval);
128     }
129
130     if(this.FileHandle && this.FileHandle.fd >= 0) {
131       await this.FileHandle.close();
132     }
133   }
134 }
135
```

Abbildung A.3: Quellcode der Algo-Klasse des Prototypen (Teil 3) (src/algo.ts)

```
1  import { Algo } from "./algo";
2
3  export class PatternBased extends Algo {
4    protected _Pattern: RegExp;
5    public get Pattern() { return this._Pattern; }
6    public set Pattern(v: RegExp) { this._Pattern = v; }
7
8    constructor(file: string, playback: boolean, filePollInterval: number, pattern: RegExp) {
9      super(file, playback, filePollInterval);
10     this._Pattern = pattern;
11   }
12
13   ProcessLine(line: string, lineNr: number) {
14     line = super.ProcessLine(line, lineNr);
15     if(line.match(this.Pattern)) {
16       this.Alert(lineNr, line);
17     }
18     return line;
19   }
20
21   Alert(...args: any[]) {
22     super.Alert(...args);
23     console.warn("\x1b[0m", `[${(new Date()).toISOString()}] Pattern based ALERT! Pattern ${this.Pattern} found in line
       `,  args[0], "\x1b[34m", args[1], "\x1b[0m");
24   }
25 }
```

Abbildung A.4: Quellcode der PatternBased-Klasse des Prototypen (src/pattern-based.ts)

```typescript
import { Algo } from "./algo";

export class TimeBased extends Algo {
  public get WindowStart() { return Date.now() - this.WindowSize; }

  protected _WindowSize: number; // window size in milliseconds
  public get WindowSize() { return this._WindowSize; }
  public set WindowSize(v: number) { this._WindowSize = v; }

  protected _Timer: NodeJS.Timeout;
  public get Timer() { return this._Timer; }
  public set Timer(v: NodeJS.Timeout) { this._Timer = v; }

  protected _LastEventTs: number;
  public get LastEventTs() { return this._LastEventTs; }
  public set LastEventTs(v: number) { this._LastEventTs = v; }

  constructor(file: string, playback: boolean, filePollInterval: number, windowSize: number, windowPollInterval: number
  = 1000) {
    super(file, playback, filePollInterval);
    this._LastEventTs = Date.now();
    this._WindowSize = windowSize;

    this._Timer = setInterval( this.HandleTimerTick.bind(this), windowPollInterval);
  }

  HandleTimerTick() {
    if(!this.Playback) {
      if(this.LastEventTs < this.WindowStart) {
        this.Alert( Date.now() - this.LastEventTs, this.WindowStart - this.LastEventTs);
      }
    }
  }

```

Abbildung A.5: Quellcode der TimeBased-Klasse des Prototypen (Teil 1) (src/time-based.ts)

```
34    ProcessLine(line: string, lineNr: number) {
35      line = super.ProcessLine(line, lineNr);
36
37      if(this.Playback) {
38        const now = new Date(line.split(",")[0]);
39        if(!isNaN(now.getTime())) {
40          const diff = now.getTime() - this.LastEventTs;
41          if(diff >= this.WindowSize) {
42            this.Alert( diff, diff - this.WindowSize, now);
43          }
44          this.LastEventTs = now.getTime();
45        }
46      } else {
47        this.LastEventTs = Date.now();
48      }
49      return line;
50    }
51
52    Alert(...args: any[]) {
53      super.Alert(...args);
54      if(this.Playback) {
55        console.warn(`[${args[2].toISOString()}] Time based ALERT! Last entry was ${(args[0] / 1000 / 60).toFixed(2)}
          minutes ago, that is ${(args[1] / 1000 / 60).toFixed(2)} minutes too late. Window size is ${(this.WindowSize /
          1000 / 60).toFixed(2)} minutes.`);
56      } else {
57        console.warn(`[${(new Date()).toISOString()}] Time based ALERT! Last entry was ${(args[0] / 1000).toFixed(2)}
          seconds ago, that is ${(args[1] / 1000).toFixed(2)} seconds too late. Window size is ${(this.WindowSize / 1000).
          toFixed(2)} seconds.`);
58      }
59    }
60
61    async Destruct() {
62      await super.Destruct();
63
64      if(this.Timer) {
65        clearInterval(this.Timer);
66      }
67    }
68  }
```

Abbildung A.6: Quellcode der TimeBased-Klasse des Prototypen (Teil 2) (src/time-based.ts)

```json
{
  "name": "verfuegbarkeitsanalyse",
  "version": "1.0.0",
  "description": "Algorithmen zum analysieren von Logdateien zur ermittlung der Verfuebarkeit der loggenden Anwendung.",
  "main": "src/main.ts",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "tsc": "tsc",
    "start:dev": "nodemon",
    "build": "rimraf ./build && tsc",
    "start": "npm run build && node build/main.js"
  },
  "keywords": [
    "algorithms",
    "log"
  ],
  "author": "Tim Pütz",
  "license": "ISC",
  "dependencies": {
    "typescript": "^4.0.3"
  },
  "devDependencies": {
    "@types/node": "^14.11.2",
    "nodemon": "^2.0.4",
    "rimraf": "^3.0.2",
    "ts-node": "^9.0.0"
  }
}
```

Abbildung A.7: Quellcode der benötigten Abhängigkeiten (package.json)

```json
{
  "watch": ["src"],
  "ext": ".ts,.js",
  "ignore": [],
  "exec": "ts-node ./src/main.ts"
}
```

Abbildung A.8: Quellcode der nodemon-Konfiguration (nodemon.json)

```json
{
  "pollInterval": 250,
  "writeInterval": 6000,
  "writeLine": "nope",
  "playback": true
}
```

Abbildung A.9: Quellcode der generellen Algorithmus-Konfiguration (confs/general.json)

```json
{
  "enabled": true,
  "regexp": {
    "pattern": ".*3 error-msgs*",
    "flags": ""
  }
}
```

Abbildung A.10: Quellcode der PatternBased-Konfiguration (confs/pattern-based.json)

```json
{
  "enabled": true,
  "windowSize": 360000,
  "pollInterval": 1000
}
```

Abbildung A.11: Quellcode der TimeBased-Konfiguration (confs/time-based.json)