Hochschule Merseburg

University of Applied Sciences

Fachbereich Ingenieur- und Naturwissenschaften

Studiengang Informatik und Kommunikationssysteme

Masterarbeit

zur Erlangung des Grades Master of Engineering (M. Eng.)

Echtzeitdetektion von Anomaliebereichen in Zeitreihen

vorgelegt bei

Prof. Dr. rer. Nat. habil. Eckhard Liebscher

Zweitprüfer: Dipl.-Wirtsch.-Inf. André Haucke

EXXETA AG

Föpplstraße 3

04347 Leipzig

Verfasser: Marcus Endtmann

Matrikel: MIKS19

Matrikelnummer: 22351

HOME
HOCHSCHULE MERSEBURG(FH)
University of Applied Sciences

FACHBEREICH INGENIEUR- UND NATURWISSEN-SCHAFTEN

Aufgabenstellung
für die Masterarbeit (M. Eng.)
von Herrn Marcus Endtmann
(MIKS Matrikel 22351)

Thema: **Echtzeitdetektion von Anomaliebereichen in Zeitreihen**

Betreuer: Prof. Dr. Eckhard Liebscher
Dipl.-Wirtsch.-Inf. André Haucke (EXXETA GmbH Leipzig)

Aufgabenstellung

Durchführung von Experimenten mit Machine Learning Methoden und Kontrollkartenverfahren zur Detektion von Bereichen in Zeitreihen, in denen ein abnormales Verhalten vorliegt, Implementierung der Methoden in Python, Optimierung der Detektionsmethoden mit Fokus auf Genauigkeit und Rechtzeitigkeit, Analyse der Bewertungsmaße. Die Analysen erfolgen an Zeitreihen, die vom Unternehmen EXXETA GmbH Leipzig zur Verfügung gestellt werden.

Schwerpunkte

1. Analyse der vorliegenden Zeitreihen
2. Numerische Experimente mit Machine-Learning-Algorithmen in Python
3. Numerische Experimente mit Kontrollkarten
4. Bewertungsmaße für die Detektion mit Fokus auf Genauigkeit und Rechtzeitigkeit

abzugebende Exemplare: 2 + PDF-Datei

J. Spi

Prof. Dr. Spillner
Vorsitzender des Prüfungsausschusses

E. Liebscher

Prof. Dr. Liebscher
Themenstellender Hochschullehrer

Inhalt

Abbildungs- und Tabellenverzeichnis

Quelltextverzeichnis

1 Einleitung

Um den Zustand von Systemen eines Unternehmens zu überprüfen werden Monitoringsysteme benutzt. Diese erheben verschiedene Metriken, sodass der Lebenszyklus des Systems überwacht werden kann. Die Anzahl der somit anfallenden Daten kann meist nicht mehr von einem menschlichen Experten beobachtet werden, sodass ein Defizit zwischen den anfallenden und tatsächlich beobachtbaren Daten entsteht. Aus diesem Grund werden automatisierte Computersysteme eingesetzt, um anomales Verhalten innerhalb der beobachteten Daten zu erkennen. Ein naives Vorgehen besteht dabei meist aus dem Setzen einer festen Ober- bzw. Untergrenze, sodass bei deren überschreiten eine Benachrichtigung an einen zuständigen menschlichen Experten geschickt wird. Ein solches Vorgehen wird bislang auch beim Monitoringsystem Amont der EXXETA AG benutzt. Solche festen Schwellen besitzen den Nachteil, dass Anomalien erst festgestellt werden, wenn das System bereits einen Extrembereich erreicht hat. Zu diesem Zeitpunkt kann jedoch die Behebung des Problems im laufenden Betrieb nicht mehr möglich sein, sodass das System heruntergefahren werden muss, bevor schwerere Konsequenzen entstehen. Äquivalent zu der immer größer werdenden Masse an erhobenen Daten ist deshalb auch das Interesse an intelligenteren Anomalieerkennungssystemen, die Anomalien so früh wie möglich erkennen, gestiegen.

Aufgrund der Vielzahl an Anomalieerkennungsalgorithmen werden nicht selten Survey Paper erstellt, die die Anomalieerkennungsalgorithmen zusammentragen und kategorisieren. Zu diesen zählt [[BW20]], welches die Anomaliedetektion in univariaten Zeitreihen sowie die Erkennung von Punkt- und Sequenzanomalien thematisiert. Umfangreichere Betrachtungen wie [[CBK09]] und [[Bl20]] tragen Algorithmen zu verschiedenen Kategorien zusammen, werten diese jedoch nicht anhand eines Leistungsmerkmals aus. Paper wie [[Di13], [Hu18a], [RDH19], [Hu18]] in denen ein Algorithmus für einen Anwendungsfall vorstellen beinhalten meist nur die Ergebnisse des vorgestellten

Algorithmus und vergleichen diese nicht mit anderen Algorithmen. Eine Ausnahme bildet [Hu18]] und [[Mu19]], wobei in [[Mu19]] für die Erkennung von Subsequenzanomalien keine Ergebnisse genannt werden. Zu beachten ist, dass die Auswertung der Algorithmen in jedem Fall aufgrund der Genauigkeit stattfindet. Keine der genannten Paper betrachtet den Zeitpunkt der Erkennung, sodass Anomalien erst erkannt werden, sobald diese im vollen Maß eingetreten sind. Des Weiteren existiert kein Paper, welches für einen konkreten Anwendungsfall die möglichen Anomalieerkennungsalgorithmen zusammenträgt, miteinander vergleicht und auswertet. Diese Masterarbeit soll diese Lücke für die Erkennung von Anomalien in Monitoringdaten schließen, wobei besonderer Fokus auf die die rechtzeitige und unmittelbare Erkennung von Anomalien gelegt wird. Die Daten entstammen dabei dem Monitoringsystem Amont, sodass der Vergleich der Algorithmen an real erhobenen Daten stattfindet.

Um diese Problematik zu bearbeiten werden in Kapitel 2 die nötigen Grundlagen erläutert. Zu diesen gehören auch Bewertungsmaße, die neben der Genauigkeit auch den Zeitpunkt der Erkennung in die Bewertung einfließen lassen. Neben der Bewertungsmethode spielen Eigenschaften wie die Korrelation der Daten eine große Rolle bei der Auswahl und anschließenden Analyse der Algorithmen. Diese Analyse wird in Kapitel 3, der Datenvorstellung, durchgeführt. Aufgrund der Erkenntnisse aus dieser Analyse und den Bedingungen der Bewertungsmaße aus Kapitel 2, wird in Kapitel 4 eine Auswahl an möglichen Anomalieerkennungsalgorithmen erläutert. Um die Algorithmen miteinander zu vergleichen und eine Aussage über ihre Leistungsfähigkeit zu treffen werden die Ergebnisse in Kapitel 5 vorgestellt und ausgewertet. Zum Abschluss werden in Kapitel 6 die Erkenntnisse der Arbeit zusammengefasst und ein Ausblick auf zukünftige wissenschaftliche Untersuchungen gegeben.

2 Theoretische Grundlagen

Im Folgenden Kapitel sollen die theoretischen Grundlagen erläutert werden, die im weiteren Verlauf der Arbeit benutzt werden. Die in diesem Kapitel erklärten theoretischen Grundlagen dienen als Wissensbasis, auf die sich in den nachfolgenden Kapiteln bezogen werden soll.

2.1 Allgemeine Definitionen zu Zeitreihen

Zeitreihe

Wie bereits in 1 erwähnt handelt es sich bei den im Amont anfallenden Daten um Zeitreihen. Nach [[Yu14]] ist eine Zeitreihe eine Sequenz von d-dimensionalen Vektoren, welche aufgrund von Beobachtungen erhoben wurden. Die durch die Zeitreihe dargestellten Beobachtungen entspringen dabei einem stochastischen Prozess. Ein stochastischer Prozess wird definiert als eine Sammlung von Zufallsvariablen $Z(\omega, t)$, wobei ω Teil eines Stichprobenbereichs ist und Z durch die Zeit t indiziert werden kann. Die Beobachtungen dieses stochastischen Prozesses werden dabei meist in einem äquidistanten Intervall t durchgeführt. [[BW20], S. 4]

Somit lässt sich eine Zeitreihe mit Länge m wie in (2.1.1) definieren.

$$X = \{x(t) \mid 1 \leq t \leq m\} \; mit \; x(t) = \big(x_1(t), x_2(t), \dots, x_d(t)\big) \qquad (2.1.1)$$

Diese Zeitreihen werden univariate Zeitreihen genannt, falls d gleich 1 ist und multivariante Zeitreihe, falls d größer oder gleich 2 ist. [[Yu14], S. 2]

Nach [[BW20]] ist der Unterschied zwischen einer Zeitreihe und einem Datensatz anderen Ursprungs, die Tatsache, dass die Beobachtungen nicht nur von den anderen Komponenten d, sondern auch vom Zeitpunkt t, beeinflusst werden. Demnach unterscheiden sich

Methoden der Zeitreihenanalyse von Methoden zur Auswertung von Zufallsvariablen, da die Unabhängigkeit und die konstante Varianz der Variablen nicht gewährleistet werden kann. [[BW20], S. 4]

Sliding Window

Eine Zeitreihe kann als geordnete Liste von aufeinanderfolgenden Datenwerten angesehen werden. Aufgrund der Tatsache, dass zukünftige Daten einer Zeitreihe von vergangenen Daten abhängen, sollten bei der Analyse eines Zeitpunkts der Zeitreihe auch vergangene Werte in die Analyse einfließen. Um eine solche Analyse zu realisieren wird meist ein Sliding Window benutzt. Bei diesem handelt es sich um ein Vorgehen, bei dem mehrere Sublisten aus der ursprünglichen Liste gebildet werden, wobei die Sublisten aus aufeinanderfolgenden Werten bestehen. Die Länge der Sublisten wird dabei über den Parameter l festgelegt. Für eine Zeitreihe $X = \{x(t) \mid 1 \leq t \leq m\}$ und einer Sublistenlänge l entstehen $m - l + 1$ Sublisten der Form $sl(t) = (x(t),\ x(t+1), \dots,\ x(t+l))$.

Residuen

Ein Grundbaustein vieler Anomalieerkennungsalgorithmen ist die Anpassung eines Modells an die Zeitreihe und die anschließende Berechnung der Residuen. Aufgrund dieser Vorgehensweise wird eine Anomalie erkannt, sollten die Residuen für einen Zeitpunkt eine vorher definierte Schwelle überschreiten. Unabhängig von der Anomalieerkennung werden die Residuen außerdem benutzt, um die Güte eines Modells zu bestimmen, indem der vorhergesagten Werte $\hat{y}_t$ mit der tatsächlichen Beobachtung y_t verglichen wird. Der Vergleich der beiden Werte erfolgt durch die Bildung der Differenz, wobei eine niedrige Differenz einer guten Anpassung des Modells entspricht. [[HA18]]

$$e_t = y_t - \hat{y}_t \qquad (2.1.2)$$

2.2 Anomalien und Anomaliearten

Nach 2.1 entsteht eine Zeitreihe durch die Beobachtung eines stochastischen Prozesses zu äquidistanten Zeitintervallen. Da diese stochastischen Prozesse meist einen realen Prozess als Grundlagen besitzen, beinhalten die Zeitreihen einen Zufallsfehler. Sobald dieser Zufallsfehler ein Maß übersteigt, kann von einem Ausreißer oder einer Anomalie gesprochen werden. Bei einem solchen Maß kann es sich zum Beispiel um eine Schwelle ober- oder unterhalb der Erwartungswerts handeln. Der Unterschied in der Bezeichnung entsteht durch das Vorgehen nach der Erkennung eines solchen anomalen Wertes. Sollen die anomalen Werten nach deren Erkennung entfernt oder korrigiert werden, wird meist von einem Ausreißer gesprochen. Wenn ein anomaler Wert erkannt wird und im Anschluss dessen Ursache analysiert werden soll, wird der Begriff Anomalie verwendet. [[Bl20], S. 2]

Aufgrund der Eigenschaften der Anomalie, sowie der Eingabedaten, können Anomalien in drei verschiedene Kategorien eingeteilt werden. Bei einer Punktanomalie handelt es sich um einen Datenwert innerhalb einer Zeitreihe, der im Vergleich zu anderen Punkten in der Zeitreihe „ungewöhnlich“ ist. Dieses Maß der „Ungewöhnlichkeit“ kann dabei im globalen Kontext, das heißt im Vergleich zu allen anderen Punkten in der Zeitreihe, oder im lokalen Kontext, das heißt in Bezug auf benachbarte Punkte in der Zeitreihe, gemessen werden. Bei einer Subsequenzanomalie handelt es sich um aufeinanderfolgende Punkte, die bei einer gemeinsamen Betrachtung „ungewöhnlich“ erscheinen. Dabei ist zu beachten, dass die individuellen Punkte innerhalb der Subsequenzanomalie keine Punktanomalien sein müssen. Die Subsequenzanomalien können wie Punktanomalien auch im globalen oder lokalen Kontext betrachtet werden. [[Bl20], S. 3] Algorithmen die Subsequenzanomalien aufgrund ihres Unterschieds im Vergleich zu normalen Subsequenzen erkennen, bezeichnen diese Anomalien oft als Discords. [[Hu18]], [[Mu19], [Ad06], [Yi07]] Sollten die Eingabedaten aus einer multivariaten Zeitreihe bestehen, kann außerdem eine Variable der multivariaten Zeitreihe als anomal klassifiziert werden. Dadurch gilt eine gesamte Zeitreihe als Anomalie im Vergleich zu den anderen Zeitreihen der Eingabedaten. [[Bl20], S. 4]

Aufgrund der Einteilung von Anomalien in verschiedene Anomaliearten, sollte bei der Suche eines geeigneten Anomalieerkennungs-Algorithmus zunächst die zu erkennende Anomalieart aus der Problemstellung abgeleitet werden. Erst dann kann die Auswahl der Algorithmen aufgrund von anderen Eigenschaften weiter eingeschränkt werden.

2.3 Semi-Supervised und Unsupervised Anomalieerkennung

Obwohl die Funktionsweisen der Anomalieerkennungsalgorithmen zum Teil sehr unterschiedlich sind, können die Algorithmen in zwei Kategorien eingeteilt werden. Diese Kategorien beziehen sich auf die Vorverarbeitung der Daten, wodurch die Algorithmen in Semi-Supervised und Unsupervised Learning unterschieden werden können [[Go16], S. 3]

Algorithmen die Semi-Supervised trainiert werden, versuchen ein binäres Klassifikationsproblem zu lösen, wobei während des Trainings nur eine Klasse verfügbar ist. Im Fall der Anomalie Erkennung handelt es sich bei dieser Klasse um alle normalen Datenpunkte. Mithilfe dieser Daten wird eine Entscheidungsfunktion bestimmt, die die normalen Daten optimal isoliert. Dementsprechend wird der Algorithmus auf die Erkennung von einer Klasse trainiert, wobei die Existenz einer anderen Klasse berücksichtigt wird. Mithilfe der Entscheidungsfunktion kann bestimmt werden, ob ein Punkt zur Klasse der normalen Punkte gehört oder nicht. Da die Anomalieerkennung als binäres Klassifikationsproblem modelliert wird, kann davon ausgegangen werden, dass jeder Wert, der nicht als normal eingestuft wurde, eine Anomalie ist. [[CIL19], S. 8]

Die Algorithmen des Unsupervised Learning funktionieren ähnlich den Semi-Supervised Learning Algorithmen, da der große Teil der normalen Daten von den Anomalien getrennt werden soll. Hierbei wird jedoch ein Datensatz benutzt, der sowohl normale Daten als auch Anomalien enthält. Der Algorithmus versucht Daten zu erkennen die sich stark von anderen Daten im Datensatz unterscheiden. Diese Erkennung beruht dabei meist auf der Tatsache, dass Anomalien selten und isoliert auftreten. [[CIL19], S. 12]

2.4 Leistungsbewertungsmethoden

Um die Anomalieerkennungsalgorithmen bewerten zu können, muss ein geeignetes Gütekriterium gefunden werden, welches die Algorithmen aufgrund ihrer Leistung bewertet. Bei dieser Bewertung ist es wichtig, dass die Kriterien der realen Problemstellung in die Berechnung des jeweiligen Scores einfließen, sodass der resultierende Score als Maß für die Problemlösefähigkeit benutzt werden kann. Aufgrund dieser Tatsache werden im folgenden Gütekriterien näher erläutert, die die Algorithmen aufgrund ihrer Genauigkeit und des Zeitpunkts der Erkennung bewerten.

2.4.1 F-Score

Das Erkennung von Anomalien kann als binäres Klassifikationsproblem formuliert werden, da ein Datenwert als normal oder anomal klassifiziert werden kann. Bei einer solchen binären Klassifikation können aufgrund der tatsächlichen Klasse vier verschiedene Fälle eintreten. Da die Anomalien korrekt erkannt werden sollen, werden diese bei der Betrachtung als Positives bezeichnet. Von einem True Positive (TP) spricht man, falls der Datenwert korrekt als anomal klassifiziert wurde. Von einem True Negative (TN) spricht man, falls der Datenwert korrekt als normal klassifiziert wurde. Im Gegensatz zu den korrekten Klassifikationen gibt es auch den Fall der inkorrekten Klassifikation. Dementsprechend spricht man von einem False Positive (FP), wenn ein normaler Datenwert als anomal klassifiziert wurde und von einem False Negative (FN), falls ein anomaler Datenwert als normal klassifiziert wurde [[Ab18], S. 19] Zur bildlichen Darstellung der Möglichkeiten bei einem binären Klassifikationsproblem kann eine Konfusionsmatrix aufgestellte werden. Diese ist in Abbildung 1 zu sehen.

		Predicted Class	
		Positive	**Negative**
Actual Class	**Positive**	True Positives (TP)	False Negatives (FN)
	Negative	False Positives (FP)	True Negatives (TN)

Abbildung 1: Konfusionsmatrix einer binären Klassifikation [[Sh20], S. 5]

Mithilfe dieser vier möglichen Fälle lassen sich Metriken bilden, die die Güte einer Klassifikation überprüfen. Bei diesen handelt es sich um Precision (Genauigkeit) und Recall (Trefferquote) welche in Kombination genutzt werden, um den F-Score einer Klassifikation berechnen zu können. [[Mu19], S. 1997]

Precision beschreibt dabei das Verhältnis der tatsächlich anomalen Punkte zu der Gesamtzahl der vom System als anomal klassifizierten Punkte. Dementsprechend kann Precision als die Wahrscheinlichkeit betrachtet werden, dass ein als anomal klassifizierter Punkt, tatsächlich anomal ist.

$$Precision = \frac{TP}{TP + FP} \qquad (2.4.1)$$

Recall ist das Verhältnis der tatsächlich anomalen Punkte zu allen Punkten die als anomal klassifiziert wurden. Aus diesem Grund ist Recall die Wahrscheinlichkeit, dass ein tatsächlich anomaler Punkt als anomal klassifiziert wird.

$$Recall = \frac{TP}{TP + FN} \qquad (2.4.2)$$

Der F-Score (F-Maß) in Gleichung (2.4.3) kombiniert Precision und Recall mithilfe des harmonischen Mittels, wodurch der Klassifikator in seiner Gesamtheit bewertet werden kann. [[Fa06], S. 862]

$$F - Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \qquad (2.4.3)$$

2.4.2 NAB-Score

Der NAB Score ist eine Metrik zum Vergleich von Echtzeit-Anomalieerkennungs-algorithmen. Als Besonderheit betrachtet der NAB Score nicht nur die Genauigkeit, sondern auch den frühsten Zeitpunkt der Erkennung. Dementsprechend werden Algorithmen besser bewertet, je eher sie eine Anomalie erkennen konnten. In Abschnitt 2.4.1 ist zu erkennen, dass traditionelle Metriken wie Precision oder Recall den zeitlichen Aspekt einer Anomalieerkennung nicht miteinbeziehen. Diese Eigenschaft spielt jedoch bei einer Echtzeitanomalieerkennung eine große Rolle, da Anomalien so früh wie möglich erkannt werden müssen.

Weitergehend soll die Berechnung des NAB Scores näher erklärt werden, wobei als Grundlage das Paper [[LA15]] benutzt wird. Da in diesem jedoch nicht die volle Funktionsweise erklärt wird, die Angaben fehlerhaft sind bzw. die Angaben im Vergleich zur Implementierung veraltet sind, wird im Folgenden zum Teil auf das offizielle Github Repository [[LA19]] verwiesen.

Um den zeitlichen Aspekt in die Berechnung des NAB Scores einzubeziehen werden anomale Bereiche im Datensatz benötigt. Sollte der Datensatz nur Punktanomalien enthalten, wird ein prozentualer Bereich um die Punktanomalie als anomaler Bereich festgelegt. Aufgrund dieses Bereichs wird eine Scoring Funktion benutzt, die einer erkannten Anomalie aufgrund der relativen Position in dem Bereich einen Score zuweist. Diese Scoring Funktion ist in der folgenden Abbildung zu erkennen.

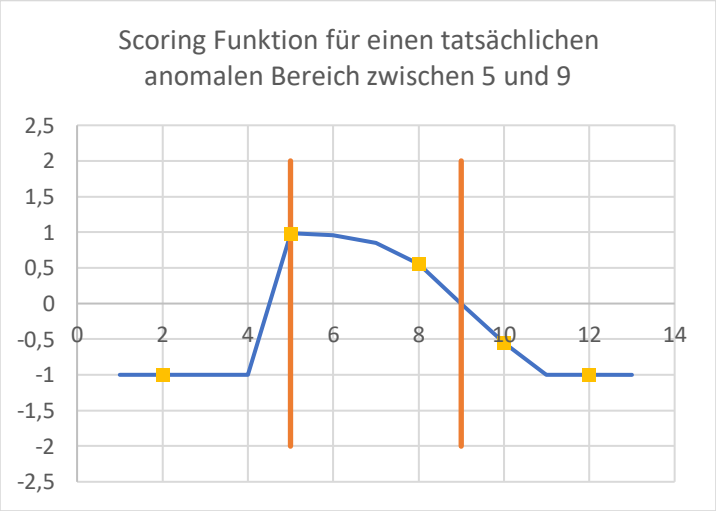

Abbildung 2: Darstellung der Scoringfunktion für eine Anomalie im Bereich zwischen 5 und 9

Der erste Punkt entspricht dabei einem FP, wodurch er dem Score einen Wert von -1 hinzufügt. Die Punkte 2 und 3 liegen beide im anomalen Bereich und entsprechen TP, wobei nur der Punkt genommen wird, dessen relative Position im anomalen Bereich am niedrigsten ist. Es wird demnach der Punkt bewertet, der am frühsten erkannt wurde. Die beiden Punkte, die nach dem anomalen Bereich erkannt wurden, entsprechen auch FP, wobei der Score niedriger wird, je weiter der Punkt vom anomalen Bereich entfernt sind.

Um zu ermöglichen, dass TP, FP und FN unterschiedlich stark in die Berechnung einfließen können, werden sogenannte Application Profiles A angelegt. In diesen werden Gewichte A_{TP}, A_{FP}, A_{FP} definiert, welche den Einfluss der jeweiligen Metriken bei der Berechnung

des Scores festlegen. Die allgemeine Berechnung des NAB Scores besteht aus einer gewichteten Sigmoid Funktion zu sehen in Gleichung (2.4.4).

$$\sigma(y) = 2 \cdot \left(\frac{1}{1 + e^{5y}}\right) - 1 \qquad (2.4.4)$$

Dabei ist y die relative Position des Punktes zum tatsächlich anomalen Bereich. Diese Position ist $0 \leq y \leq -1$ innerhalb des anomalen Bereiches, wobei 0 dem rechten Ende und -1 dem linken Ende entspricht. Für die Berechnung von y lassen sich im Fall eines TP und eines FP zwei unterschiedliche Formeln aufstellen. Diese entstammen dem Github Repository [[LA19], nab/sweeper.py:175-190], da in [[LA15]] nicht darauf eingegangen wird.

$$y_{TP} = \frac{-(rechtes\ Ende\ des\ anom.\ Bereichs - Index\ der\ Anomalie)}{Länge\ des\ Anomalen\ Bereichs} \qquad (2.4.5)$$

$$y_{FP} = \frac{|rechtes\ Ende\ des\ vorherigen\ anom.\ Bereichs - Index\ der\ Anomalie|}{Länge\ des\ Anomalen\ Bereichs} \qquad (2.4.6)$$

Dadurch besitzen FP eine relative Position $y \geq 0$, wobei y in Bezug auf den Bereich berechnet wird, der vor der erkannten Anomalie liegt. Nach der Berechnung der rohen NAB Scores werden die Gewichte aus dem Application Profile benutzt, um den gewichteten NAB Score zu berechnen. Dabei wird jeder anomale Bereich, in dem keine Anomalie erkannt wurde, als FN festgelegt und deren Anzahl in f festgehalten. Für den gewichteten NAB Score ergibt sich dadurch Gleichung (2.4.7).

$$S^A = \sum_{y>0} A_{FP} \cdot \sigma(y) + \sum_{0 \leq y \leq -1} A_{TP} \cdot \sigma(y) + A_{FN} \cdot f \qquad (2.4.7)$$

Dieser Score wird im Anschluss normalisiert, sodass die Performance eines Algorithmus mit der Performance von anderen Algorithmen verglichen werden kann. Der finale NAB Score ergibt sich dadurch als

$$S^A_{NAB} = 100 \cdot \frac{S^A - S^A_{null}}{S^A_{perfect} - S^A_{null}} \qquad (2.4.8)$$

wobei S^A_{null} einem Score entspricht, in dem keine Anomalien erkannt wurden und $S^A_{perfect}$ einem Score entspricht, in dem alle Anomalien perfekt erkannt wurden. Aus dieser

Gleichung ergibt sich, dass ein perfekter Anomalieerkennungsalgorithmus einen normalisierten NAB Score von 100 erhalten würde und ein Algorithmus, der keine Anomalien erkennt, einen Score von 0.

Im Paper [[NC17]] wird auf die Schwächen des NAB Scores eingegangen. Zu diesen gehört die Tatsache, dass die tatsächlichen anomalen Sequenzen alle gleich groß sind, da diese zu Beginn aufgrund von Punktanomalien bestimmt wurden. Außerdem wird die Scoring Funktion kritisiert, da die Berechnung der relativen Position nicht eindeutig ist und eine Konstante benutzt, die nicht näher erklärt wird. Des Weiteren wird darauf hingewiesen, dass die benutzten Beispiele Fehler aufweisen oder komplett von den in den Gleichungen benutzten Vorgehensweisen abweichen. [[NC17], S. 1572] Außerdem bewertet der NAB Score nicht alle möglichen Fälle einer binären Klassifikation gleichmäßig, da Precision stärker in den Score einfließt als Recall [[Ta18], S. 7]. Eine weitere Limitierung entsteht durch die Festlegung, dass frühzeitige Erkennungen in jedem Fall am besten sind. Dadurch können Szenarien, in denen eine Sequenz erst als anomal klassifiziert wird, nachdem ausreichend Informationen gesammelt wurden, nicht realisiert werden.

Aufgrund der Schwächen des NAB Scores wird im Folgenden eine andere Scoring Funktion erläutert, die ihre Grundlage auf dem klassischen Precision und Recall besitzt und diese für anomale Sequenzen anpasst.

2.4.3 Precision and Recall for Time Series

Das klassische Precision und Recall wie in Abschnitt 2.4.1 wurde aufgrund von einzeln auftretenden und unabhängigen Punkten definiert. Bei der Erkennung von anomalen Sequenzen entstehen jedoch einige Besonderheiten, wie zum Beispiel die Überschneidung des tatsächlichen und berechneten anomalen Bereiches. Die klassische Definition von Precision und Recall berücksichtigen Überschneidungen und deren Eigenschaften, wie Position und Größe nicht. Aus diesem Grund ist es notwendig, Precision und Recall für anomale Sequenzen neu zu definieren. Eine solche neue Definition wurde im Paper [[Ta18]] vorgestellt. In diesem werden die Metriken Precision und Recall erweitert, sodass die Größe, Position und Kardinalität der berechneten und tatsächlichen anomalen Sequenzen berücksichtigt wird.

Recall ist eine Metrik, die erfolgreich erkannte Anomalien (TP) belohnt und nicht erkannte Anomalien (FN) bestraft. Für einen Satz tatsächlicher anomaler Bereiche $R = \{R_1, \dots, R_{N_r}\}$ und einem Satz berechneter anomaler Bereiche $P = \{P_1, \dots, P_{N_p}\}$ wird der Recall Score für jeden tatsächliche anomalen Bereich $R_i \in R$ berechnet, um anschließend einen gesamten Recall Score zu berechnen. Der gesamte Recall Score wird im Anschluss durch Anzahl aller tatsächlichen anomalen Bereiche geteilt, um den durchschnittlichen Recall Score zu erhalten.

$$Recall_T(R, P) = \frac{\sum_{i=1}^{N_r} Recall_T(R_i, P)}{N_r} \quad (2.4.9)$$

Bei der Berechnung eines einzelnen Recall Scores $Recall_T(R_i, P)$ werden dabei vier Einzelheiten der Erkennung betrachtet. Bei diesen handelt es sich um die Existenz, Größe, Position und Kardinalität. Die Existenz beschreibt dabei die Erkennung des anomalen Bereiches, selbst wenn es nur einem Punkt in R_i ist. Die Größe beschreibt den Anteil von R_i der vom Algorithmus erkannt wurde. Die Position beschreibt die Position des berechneten Bereiches in Bezug auf R_i. Die Kardinaliät beschreibt den Zusammenhang der berechneten Bereiche $P_j \in P$, sodass ein einzelner Bereich P_j der R_i abdeckt besser ist als mehrere Bereiche in P. Die Berechnung von $Recall_T(R_i, P)$ erfolgt aufgrund einer gewichteten Summe zwischen zwei Komponenten und ist in Gleichung (2.4.10) zu sehen. Die erste Komponente ist die $Existenz$ und wird mit α gewichtet. Die aus der Überscheidung entstehende Größe, Position sowie Kardinalität werden in der zweiten Komponente $Überschneidung$ vereint, welche mit $(1 - \alpha)$ gewichtet wird. Dabei ist α auf $0 \leq \alpha \leq 1$ begrenzt, wobei α beschreibt, wie stark der die beiden Komponenten in $Recall_T(R_i, P)$ einfließen.

$$Recall_T(R_i, P) = \alpha \cdot Existenz(R_i, P) + (1 - \alpha) \cdot Überschneidung(R_i, P) \quad (2.4.10)$$

Die Existenz wird mit 1 bewertet, sollten die Elemente R_i in den Bereichen $P_j \in P$ mindestens einmal vorhanden sein. Ansonsten wird die Existenz mit 0 bewertet.

$$Existenz(R_i, P) = \begin{cases} 1 & falls \sum_{j=1}^{N_p} |R_i \cap P_j| \geq 1 \\ 0 & sonst \end{cases} \quad (2.4.11)$$

Für die Berechnung von $Überschneidung(R_i, P)$ werden drei Funktionen $0 \leq \gamma() \leq 1$, $0 \leq \omega() \leq 1$ und $\delta() \geq 1$ definiert. Diese beschreiben die oben erklärten Eigenschaften Kardinalität $\gamma()$, Größe $\omega()$ und Position $\delta()$, welche sich in $Überschneidung(R_i, P)$ verbinden. Dabei dient die Kardinalität als Gewicht für die Belohnung, die aus der Position und Größe der Überschneidung resultieren.

$$Überschneidung(R_i, P) = Kardinalität(R_i, P) \cdot \sum_{j=1}^{N_p} \omega\left(R_i, R_i \cap P_j, \delta\right) \qquad (2.4.12)$$

Die Kardinalität ist am größten, wenn nur ein $P_j \in P$ mit R_i überschneidet. [[Ta18], S. 4]

$$Kardinalität(R_i, P) = \begin{cases} 1, & wenn\ nur\ ein\ P_j \in P\ mit\ R_i \text{ überschneidet} \\ \gamma(R_i, P), & sonst \end{cases} \qquad (2.4.13)$$

Die Funktion $\gamma(R_i, P)$ kann auf mehrere Weisen definiert werden, wobei das Reziproke einer rationalen Funktion empfohlen wird. Als Beispiel wird $\gamma(R_i, P) = \frac{1}{x}$ genannt, wobei x der Anzahl der unterschiedlichen Überschneidungen entspricht

Für die Definition der Funktionen $\omega()$ und $\delta()$ dienen die beiden Pseudocodeausschnitte in Abbildung 3.

```
function ω(AnomalyRange, OverlapSet, δ)
    MyValue ← 0
    MaxValue ← 0
    AnomalyLength ← length(AnomalyRange)
    for i ← 1, AnomalyLength do
        Bias ← δ(i, AnomalyLength)
        MaxValue ← MaxValue + Bias
        if AnomalyRange[i] in OverlapSet then
            MyValue ← MyValue + Bias
    return MyValue/MaxValue
```

(a) Overlap size

```
function δ(i, AnomalyLength)          ▷ Flat bias
    return 1
function δ(i, AnomalyLength) ▷ Front-end bias
    return AnomalyLength - i + 1
function δ(i, AnomalyLength) ▷ Back-end bias
    return i
function δ(i, AnomalyLength)      ▷ Middle bias
    if i ≤ AnomalyLength/2 then
        return i
    else
        return AnomalyLength - i + 1
```

(b) Positional bias

Abbildung 3: Pseudocode für die Funktionen Größe $\omega()$ und Position $\delta()$ [[Ta18], S. 4]

Die Funktion $\omega()$ berechnet die Größe der Überschneidung im Verhältnis der Position. Die Position $\delta()$ kann dabei durch verschiedene Funktionen implementiert werden. Dabei entspricht der Flat bias einer Implementierung, bei der die Position immer gleich wichtig ist, der Front-end bias priorisiert frühere Überschneidungen über spätere Überschneidungen und der Middle bias Überschneidungen in der Mitte des anomalen Bereiches.

Precision ist eine Metrik die fälschlicherweise erkannte Anomalien (FP) bestraft anstatt nicht erkannter Anomalien (FN). Um die Precision $Precision_T(R, P)$ für anomale Sequenzen zu berechnen wird der Precision Score $Precision_T(R, P_i)$ für jede berechnete Anomalie $P_i \in P$ berechnet, um im Anschluss die Einzelwerte zu einem Gesamtscore zusammenzufassen. Um $Precision_T(R, P)$ zu erhalten wird dieser Gesamtscore durch die Anzahl der berechneten anomalen Bereiche N_p geteilt.

$$Precision_T(R, P) = \frac{\sum_{i=1}^{N_p} Precision(R, P_i)}{N_p} \qquad (2.4.14)$$

Für die Berechnung von $Precision_T(R, P_i)$ wird ähnlich vorgegangen, wie bei der Berechnung von $Recall_T(R_i, P)$, wobei die Komponente der Existenz nicht mit einfließt. Dies ist in Gleichung (2.4.15) zu sehen.

$$Precision_T(R, P_i) = Kardinalität(\mathrm{P}_i, R) \cdot \sum_{j=1}^{N_r} \omega\left(P_i, P_i \cap R_j, \delta\right) \qquad (2.4.15)$$

Bei der Berechnung von $Precision_T(R, P_i)$ können andere Funktionen für Kardinalität $\gamma()$, Größe $\omega()$ und Position $\delta()$ benutzt werden, als bei der Berechnung von $Recall_T(R_i, P)$. Im Paper wird empfohlen bei der Berechnung von $Recall_T(R_i, P)$ die Kardinalität $\gamma()$ und Größe $\omega()$ genauso zu implementieren, wie bei der Berechnung von $Precision_T(R, P_i)$. Nur für die Berechnung der Position $\delta()$ sollte der Flat bias benutzt werden, da die Position eines FP meistens keine Bedeutung besitzt.

2.5 Machine Learning und Deep Learning

Das Forschungsgebiet des Machine Learning kann als die Verbindung der Informatik und der Stochastik angesehen werden. Die Informatik als Forschungsbereich beschäftigt sich mit der Lösbarkeit von Problemen und deren Umsetzung mithilfe eines Computersystems. Die Stochastik befasst sich im Allgemeinen mit der Extraktion von Informationen aus Daten, sowie der Erzeugung von Modellen, welche diese Daten repräsentiert. Dementsprechend befasst sich das Machine Learning mit der Lösung von Problemen mithilfe von Modellen, die automatisch aufgrund von Daten erzeugt wurden. [[Mi06], S. 1] Machine Learning kann durch diese Vorgehensweise Probleme lösen, die mit traditionellen Methoden der Informatik nicht gelöst werden können. Zu diesen gehören zum Beispiel Probleme, bei denen ein Mensch den Lösungsvorgang nur schwer erklären kann, wie die bei der Spracherkennung. Ein anderes Beispiel sind Probleme, die über eine so diverse Eingabegröße verfügen, dass ein Mensch nicht für jede mögliche Kombination ein Vorgehen festlegen kann. [[Si16], S. 22]

Eine Spezialform des Machine Learnings stellt das Deep Learning dar. Das Deep Learning befasst mit den künstlichen neuronalen Netzen, einem Modell, welches schichtweise Daten verarbeitet. Da jede Schicht die Daten auswertet und das Ergebnis an die nächste Schicht weitergibt, führt das Hinzufügen von Schichten zu einer erhöhten Problemlösefähigkeit des Deep Learning Algorithmus. Um die Vorgehensweise des Machine Learning mit dem Deep Learning zu vergleichen, kann das Beispiel einer Gesichtserkennung betrachtet werden. Der Machine Learning Algorithmus erkennt das Gesicht, indem er das Gesicht auf das Vorhandensein von Augen, der Nase oder den Mund überprüft. Der Deep Learning Algorithmus lernt aufgrund der schichtweisen Verarbeitung auch die Beziehung zwischen den einzelnen Bestandteilen des Gesichts, sodass auch die korrekten Proportionen gelernt werden. [[Si16], S. 23] Obwohl sich die Deep Learning Algorithmen im Laufe der Zeit stetig weiterentwickelt haben, hat ist die grundlegende Funktionsweise gleichgeblieben. Diese Funktionsweise, sowie der Aufbau eines traditionellen künstlichen neuronalen Netzes wird im folgenden Abschnitt erläutert.

2.6 Künstliche neuronale Netze

Das künstliche neuronale Netz ist ein mathematisches Modell, welches dem menschlichen Gehirn nachempfunden wurde. Die kleinste Komponente innerhalb der künstlichen neuronalen Netze sind die namensgebenden Neuronen. Die Aufgabe der Neuronen ist die Verarbeitung von Signalen und die Weitergabe an das nächste Neuron. Ein solches Neuron ist in der Abbildung 4 zu sehen.

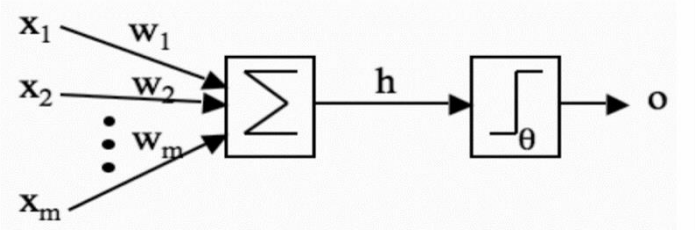

Abbildung 4: Darstellung eines künstlichen Neurons [[Ma14], S. 41]

Die Eingangssignale $x_1, \dots, x_m$ werden mit den gewichten $w_1, \dots, w_m$ multipliziert und anschließend die jeweiligen Ergebnisse summiert. Das Ergebnis ist die Propagierfunktion h mit

$$h = \sum_{j=1}^{m} x_j w_j \qquad (2.6.1)$$

wobei das Ergebnis im Anschluss in eine Aktivierungsfunktion eingesetzt wird um o zu erhalten. [[Ma14], S. 41] Durch die Wahl einer nichtlinearen Aktivierungsfunktion wird das bisher lineare Modell nichtlinear [[Sa94], S. 4].

Ein Beispiel für eine solche nichtlineare Aktivierungsfunktion ist die Sigmoid-Funktion in Gleichung (2.6.2).

$$Sigmoid(h) = \frac{1}{1 + e^{-h}} \qquad (2.6.2)$$

Durch die Sigmoid-Funktion wird die Summe h in einen Bereich zwischen 0 und 1 skaliert. [[Ma14], S. 75]

Ein künstliches neuronales Netz besteht aus mehreren solcher Neuronen, die in Schichten angeordnet sind. Eine Schicht kann als vertikaler Stapel von n Neuronen angesehen werden,

sodass eine Neuronenschicht n Ausgaben besitzt. Jedes Neuron berechnet dabei die Ausgabe o unabhängig von den anderen Neuronen in der gleichen Schicht, wobei jedes Neuron die gleichen Eingangssignale $x_1, \dots, x_m$ erhält. Ein künstliches neuronales Netz entsteht, indem die Neuronenschichten horizontal gestapelt werden. Dieser Aufbau ist in der folgenden Abbildung zu sehen. [[LC01], S. 183]

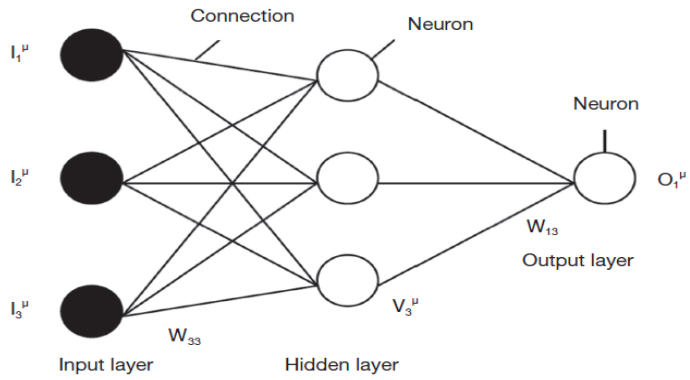

Abbildung 5: Darstellung künstliches neuronales Netz [[Pa15b], S. 954]

In der Abbildung ist zu erkennen, dass die Ausgaben der Eingabeschicht (engl. Input layer) als Eingangssignale $x_1, \dots, x_n$ der verborgenen Schicht (engl. Hidden layer) dienen. Dieser Vorgang wird für die jeweils nächste Schicht wiederholt, bis die Ausgabeschicht (engl. Output layer) des Netzes erreicht ist. [[Ko07], S. 255]

Die Ausgaben der Ausgabeschicht stellen das Ergebnis $\hat{y}$ des künstlichen neuronalen Netzes dar. Um das Netz für einen Datensatz zu trainieren, werden die Gewichte w für jedes Neuron angepasst, sodass das Ergebnis $\hat{y}$ so nah wie möglich am Ergebnis y liegt. Der Trainingsalgorithmus für künstliche neuronale Netze nennt sich Backpropagation-Algorithmus und versucht den entstandenen Fehler durch die Anpassung der Gewichte so weit wie möglich zu reduzieren. Dafür wird im ersten Schritt der Fehler zwischen $\hat{y}$ und y durch eine Fehlerfunktion E berechnet. Im Fall der Regression kann der Mean Squared Error in Gleichung (2.6.3) zur Berechnung des Fehlers benutzt werden. [[Ma14], S. 102]

$$Means\ Squared\ Error = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \qquad (2.6.3)$$

Da der Fehler minimiert werden soll, wird der Gradient der Fehlerfunktion berechnet. Der Gradient wird dabei mithilfe einer partiellen Ableitung der Fehlerfunktion nach den

Gewichten berechnet. Die Fehlerfunktion hängt jedoch nur direkt von der Ausgabe $\hat{y}$ der Ausgabeschicht ab, wobei die Ausgabe aufgrund der Aktivierungsfunktion entsteht. Die Aktivierungsfunktion wird aufgrund der Propagierfunktion h berechnet, welche direkt die Gewichte w zur Berechnung von h benutzt. Aus diesem Grund wird zur Berechnung des Gradienten der Fehlerfunktion E die Kettenregel benutzt, welche im speziellen Beispiel die Form $\frac{\partial E}{\partial w} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial h} \cdot \frac{\partial h}{\partial w}$ besitzt. Der Gradient gibt an in welche Richtung und wie stark die Gewichte angepasst werden müssen. [[Kr07], S. 89ff.] Dieses Vorgehen ist in Abbildung 6 veranschaulicht.

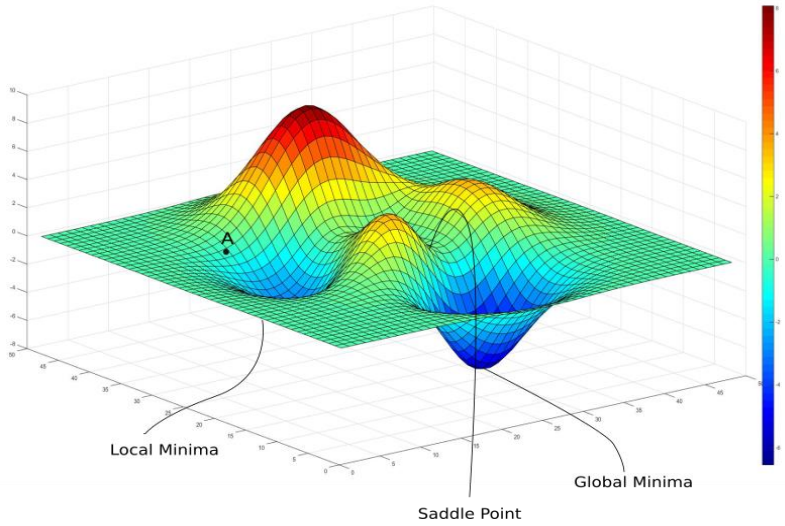

Abbildung 6: Potentielle Fläche der Fehlerfunktion [[bl18]]

Aufgrund des Gradienten können die Gewichte angepasst werden. Dies erfolgt durch Gleichung (2.6.4) wobei w^t die derzeitigen Gewichte sind und α die Lernrate, die den Gradienten skaliert. [[Ma14], S. 103]

$$w^{t+1} = w^t - \alpha \frac{\partial E}{\partial w} \qquad (2.6.4)$$

2.7 Python

Für die praktische Umsetzung wird die Programmiersprache Python benutzt. Python gehört dabei zu den am meist benutzen Programmiersprachen im Umfeld der Datenauswertung und des Machine Learning. Grund dafür ist unter anderem die einfache Lesbarkeit der Programmiersprache, durch die sowohl Informatiker als auch Statistiker oder Data Scientist ihre Projekte in kürzester Zeit umsetzen und fremden Code verstehen können. Außerdem besitzt Python eine hohe Flexibilität, wodurch Programme sowohl ausschließlich objektorientiert, funktional, prozedural oder durch eine Mischung der Paradigmen umgesetzt werden können. Dadurch verschwindet Python als Hürde bei der Lösung von Problemen, sodass sich der Programmierer komplett auf die komplexe Problemstellung konzentrieren kann. Außerdem ermöglichen Jupyter Notebook die Erstellung von interaktiven Umgebungen in denen Code, graphische Darstellungen und Texte miteinander verbunden werden, sodass Ergebnisse gleichzeitig ausprobiert, dargestellt und erklärt werden können. Diese Vorgehensweise wird weitergehend durch das umfangreiche Ökosystem an Bibliotheken unterstützt, durch die für fast jede Thematik eine bereits gepflegte Implementierung existiert. Zu diesen gehört zum Beispiel NumPy für wissenschaftliche Berechnungen auf mehrdimensionalen Arrays, Matplotlib oder Seaborn zur graphischen Darstellung von Daten, Scikit-learn zur Benutzung von Machine Learning Algorithmen und TensorFlow zur Implementierung von Deep Learning Algorithmen. Viele Bibliotheken übernehmen nicht selten Funktionsweisen von Implementierungen anderer Programmiersprachen oder haben diese direkt als Vorbild. Dies ist zum Beispiel bei Matplotlib der Fall, die sich bei der Erzeugung von mathematischen Plots an MATLAB orientiert. Ein potenzieller Nachteil bei der Entwicklung rechenaufwendigen Projekten ist die Langsamkeit von Python. Viele Machine Learning Bibliotheken benutzen deshalb C-Bibliotheken für rechenaufwendige Operationen im Hintergrund. Dadurch bleiben die einfache Lesbarkeit und Flexibilität von Python erhalten und werden mit der Schnelligkeit von C verbunden.

3 Datenvorstellung

In diesem Kapitel sollen die Daten des Monitoringsystems Amont vorgestellt und analysiert. Die aus der Analyse entstehenden Erkenntnisse bestimmen die Auswahl der ausgewählten Algorithmen in Kapitel 4.

3.1 Allgemein

Das Monitoringsystem Amont besteht aus einer Vielzahl an Systemen, deren Komponenten durch Amont überwacht werden. Da im Rahmen dieser Arbeit nicht alle Systeme vorgestellt und analysiert werden können, wurde exemplarisch das System „MTS-ETL“ für die Vorstellung der Daten ausgewählt. Dieses System besteht aus 17 Metriken, wobei nur 8 aus Zeitreihen bestehen, die im Rahmen einer Anomalieerkennung untersucht werden können. Die in dieser Arbeit benutzten Daten bestehen aus einem Auszug der Daten aus Amont, welche sich auf den Zeitbereich vom 16.06.2020 bis 16.07.2020 beziehen. In der Abbildung 7 ist das System „MTS-ETL“ mit seinen 8 Metriken zu sehen.

Eine Messung der Metrik für eine Komponente findet dabei in einem Intervall von 15 Minuten statt. In der Abbildung sind außerdem die bisher benutzten manuellen Schwellwerte zu erkennen. Diese manuellen Schwellen entsprechen einer konstanten Funktion, sodass jeder Messwert, der diese Schwelle übertritt, als Anomalie klassifiziert wird. Als Folge wird festgelegt, dass sich das Gesamtsystem „MTS-ETL“ in einem anomalen Zustand befindet. Die Metriken des hier betrachteten Systems bestehen dabei aus der Verfügbarkeit von Softwarekomponenten, die in Form der Antwortzeit gemessen wird. Bei diesen Softwarekomponenten handelt es sich um die RabbitMQ zur asynchronen Nachrichtenkommunikation, der Webservices MTS.nom, MTS.cm und MTS.etl, sowie der Erreichbarkeit der Datenbank für MTS.nom, MTS.cm sowie MTS.etl. Außerdem ist zu erkennen, dass die Metrik „Erreichbarkeit von RabbitMQ auf TGDEDTM01SV0016 über

Port 5672 mit TCP (ms)“ die manuellen Grenzen nicht überschreitet, obwohl zwei überdurchschnittliche Werte am 26.06 und 18.07 auftreten.

Abbildung 7: Parallele Darstellung der Komponenten des Systems "MTS-ETL"

3.2 Analyse der Korrelation

Um festlegen zu können, ob die Analyse der Anomalien univariat oder multivariat stattfinden soll, muss zuerst überprüft werden, ob die Zeitreihen des Systems miteinander korrelieren. Dazu wurde eine Korrelationsmatrix aufgestellt, welche in Abbildung 8 zu sehen ist. In der Korrelationsmatrix ist zu erkennen, dass die Zeitreihen 5 „Webservice von MTS.cm TSO für Entgelte (ms)“ und 6 „Webservice von MTS.cm TSO für Zeitreihen (ms)“ einen Korrelationskoeffizient ρ von 0.715 und die Zeitreihen 7 „DB Verbindungen von MTS.nom TSO (ms)“ und 8 „DB Verbindungen von MTS.cm TSO (ms)“ einen Korrelationskoeffizient ρ von 0.732 besitzen. Dementsprechend besteht eine lineare Korrelation innerhalb dieser beiden Paare. Diese Korrelation kann danach in einem Scatterplot graphisch dargestellt werden. Dies ist in der Abbildung 9 zu sehen.

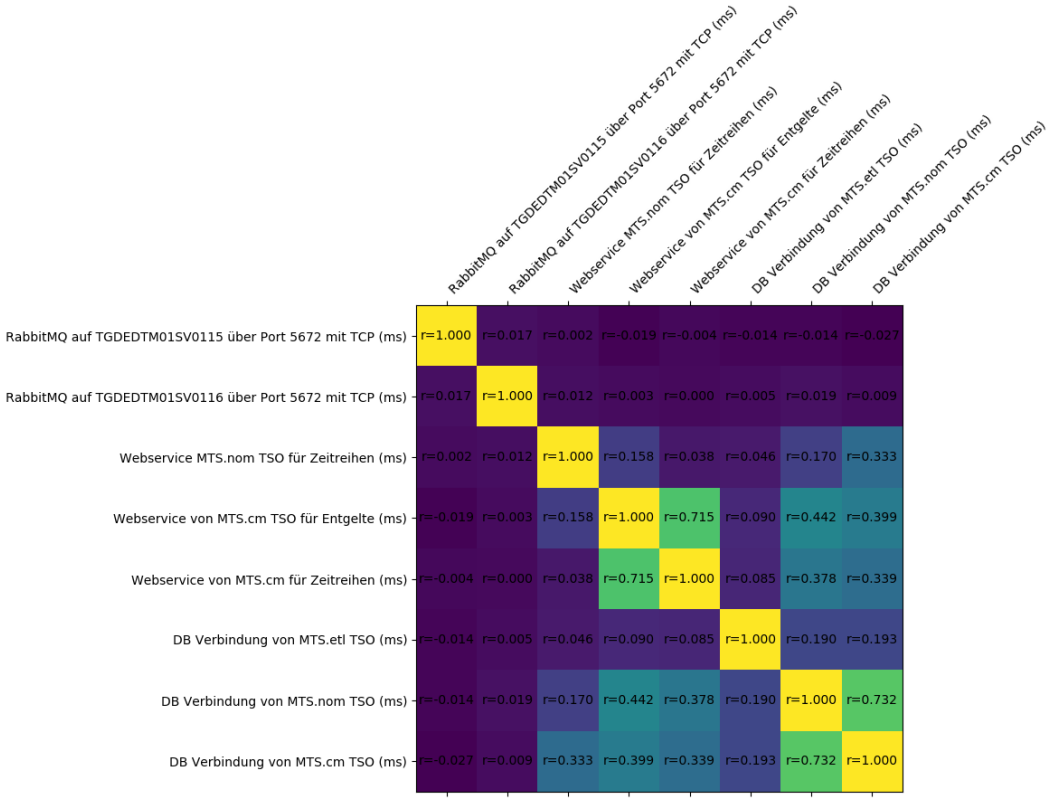

Abbildung 8: Korrelationmatrix des Systems „MTS-ETL“

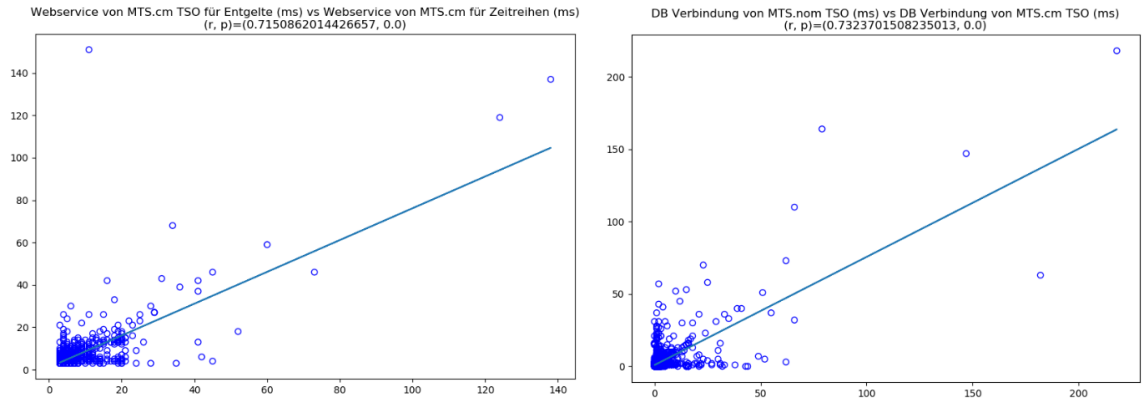

Abbildung 9: Lineare Korrelation der Zeitreihen 5,6 und 7,8

Da Anomalien in den Daten die Korrelation nach Pearson beeinflussen kann, wurde außerdem die Korrelationmatrix ohne die anomalen Bereiche aufgestellt. Das Ergebnis ist in der Folgenden Abbildung zu sehen.

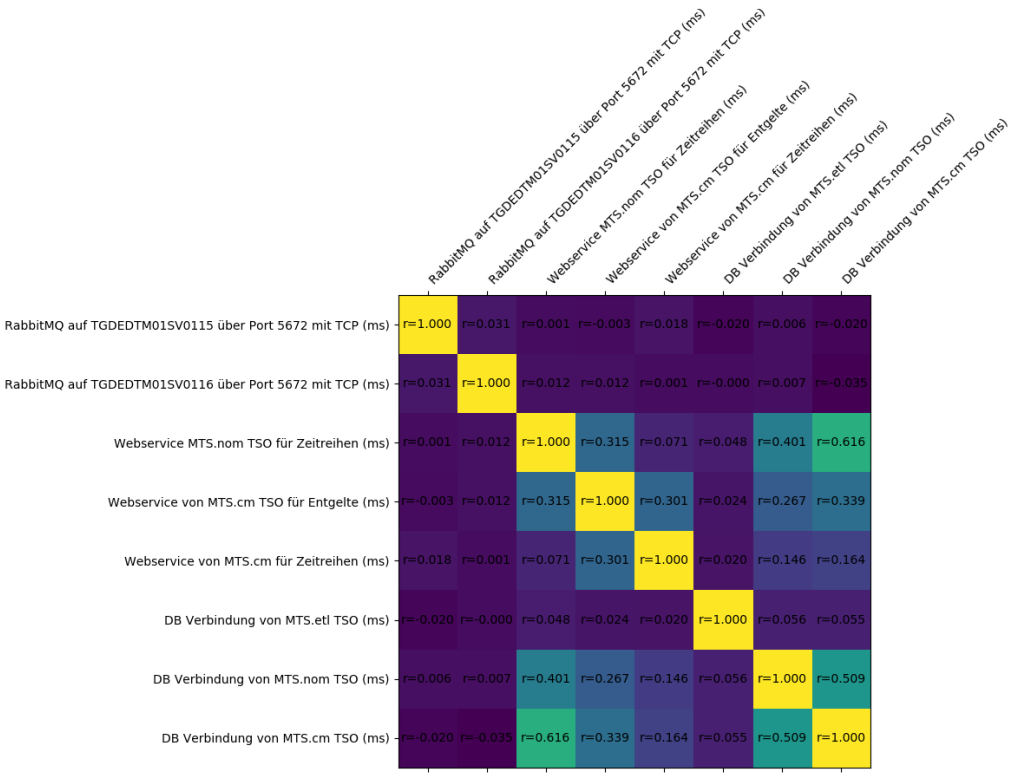

Abbildung 10: Korrelationmatrix des Systems nach entfernen der anomalen Bereiche

Nach dem Entfernen der Anomalien erhöht sich die Korrelation zwischen manchen Zeitreihen, wobei sich jedoch die Korrelation von Zeitreihe 4 und 5 von 0.715 auf 0.301 und die Korrelation zwischen 7 und 8 von 0.732 auf 0.509 sinkt. Außerdem besitzt dieses System eine hohe Korrelation im Vergleich zu den anderen Systemen im Amont. Dies ist in der Folgenden Abbildung zu sehen.

Abbildung 11: Korrelationmatrix des Systems „MTS-WEB-CS-TE“

Aufgrund der Tatsache, dass die meisten Systeme unkorrelierte Zeitreihen aufweisen, wird weitergehend eine multivariate Untersuchung ausgeschlossen.

3.3 Analyse der Häufigkeitsverteilung

Für die genauere Analyse einer einzelnen Zeitreihe wird die Metrik „Erreichbarkeit von Webservice von MTS.cm TSO für Entgelte (ms)“ benutzt. Der Verlauf dieser Zeitreihe ist in Abbildung 12 zu sehen. Zu erkennen ist das anomale Verhalten im Bereich vom 06.07.2020 bis 07.07.2020 und dem 13.07.2020 bis 14.07.2020 indem mehrere Anomalien hintereinander auftreten. Des Weiteren befinden sich zwischen 27.06.2020 und 03.07.2020 3 einzeln auftretende Anomalien. Für den Mittelwert μ ergibt sich 5.195 und für die Standardabweichung σ 5.264, wobei diese durch Anomalien in den bereits genannten Bereichen stark beeinflusst werden.

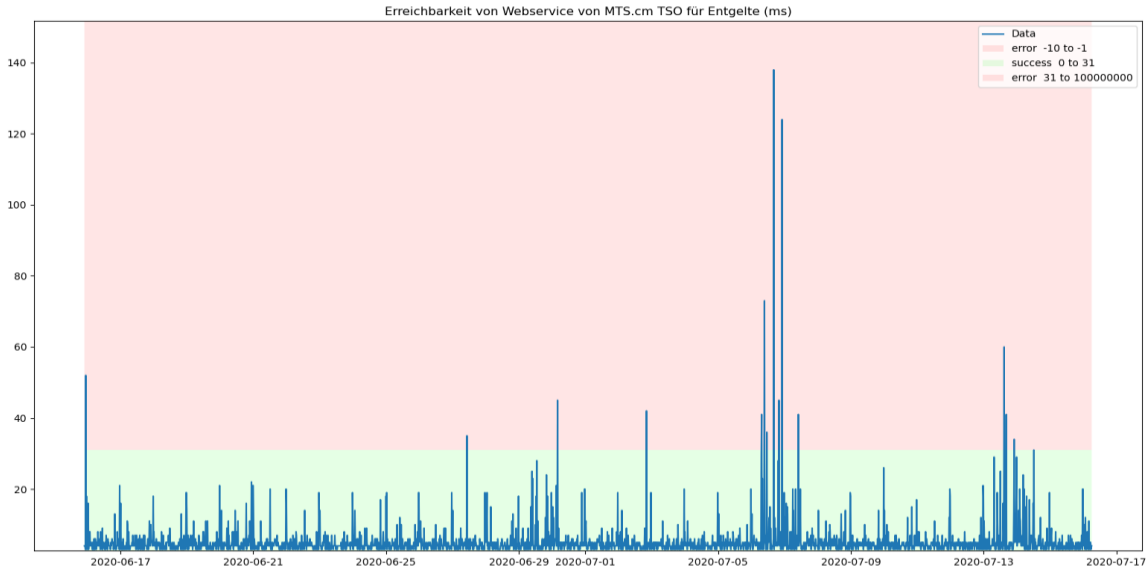

Abbildung 12: Plot der Metrik „Erreichbarkeit von Webservice von MTS.cm TSO für Entgelte (ms)"

Die große Anzahl der Anomalien ist auch im Boxplot und Violinplot zu erkennen, welche in der folgenden Abbildung zu sehen sind.

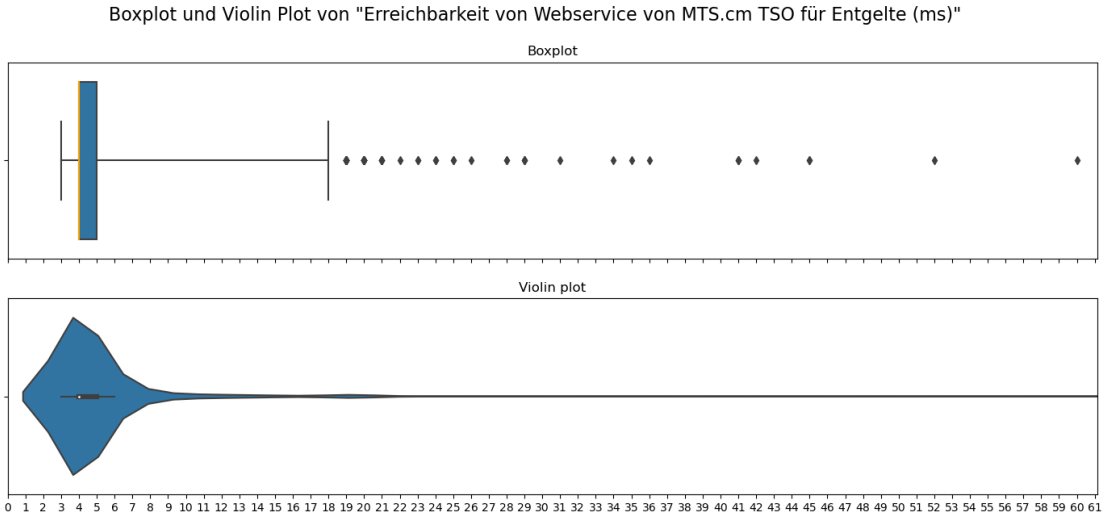

Abbildung 13: Boxplot und Violinplot für „Erreichbarkeit von Webservice von MTS.cm TSO für Entgelte (ms)"

Für die bessere Lesbarkeit wurde die x-Achse der beiden Diagramme auf den Bereich des 99.9 Prozent Quantils beschränkt, da die höchste Antwortzeit bei 138 ms liegt. Im Boxplot ist zu erkennen, dass das obere Quantil bei 5 und das untere Quantil bei 4 liegt, wodurch der obere Whisker (97.5% Quantil) bei 18 und der untere Whisker (2.5% Quantil) bei 3 liegt. Damit befinden sich 50% der Werte zwischen 4 und 5 und 95% der Werte zwischen 3 und 18ms. Im Violin Plot wird veranschaulicht, dass die Werte von 3 bis 6 im 1.5-Fachen

Interquartilsabstands liegen. Außerdem ist die Dichteverteilung zu erkennen, wobei am häufigsten Werte um 4 ms auftreten. Die Dichteverteilung kann im Detail im folgenden Histogramm und Density Plot betrachtet werden.

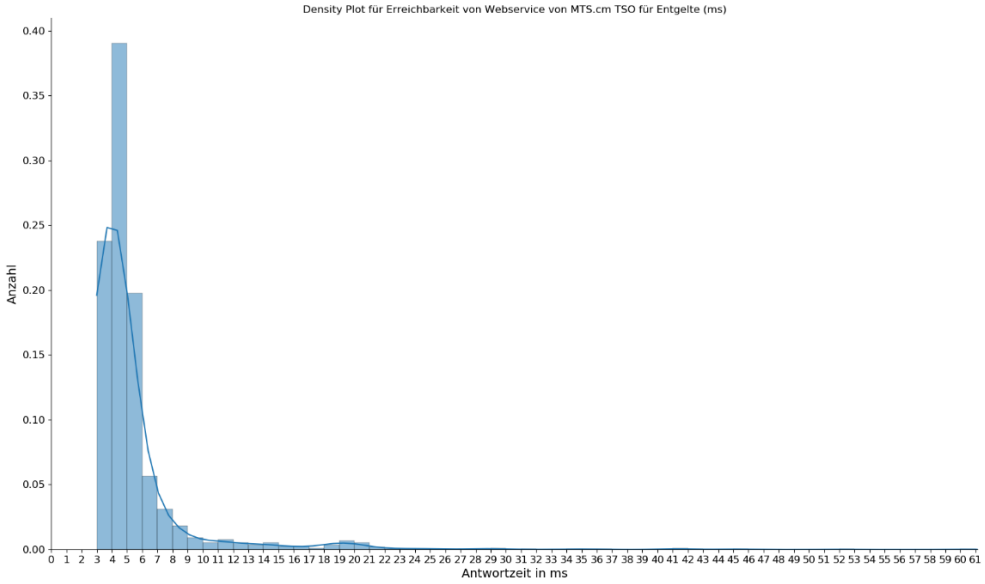

Abbildung 14: Densityplot für „Erreichbarkeit von Webservice von MTS.cm TSO für Entgelte (ms)"

Auch das Histogramm wurde auf zur Erhöhung der Lesbarkeit auf den Bereich des 99.9 Prozent Quantils beschränkt. Das Histogramm zeigt auch ein verstärktes Auftreten von Werten zwischen 3 und 6, ähnlich des 1.5-Fachen Interquartilsabstands. Das Density Plot zeigt außerdem die Rechtsschiefe der Verteilung, die dadurch verstärkt wird, dass in der Stichprobe kein Wert unter 3 liegt. Des Weiteren muss beachtet werden, dass im Kontext der Antwortzeit eines Systems die Werte keinen Wert unter 0 annehmen können.

3.4 Analyse temporaler Besonderheiten

Um möglicherweise temporal auftretende Besonderheiten zu untersuchen, wurden außerdem ein Histogramm für den Vergleich von Messwerten vor 12 Uhr und nach 12 Uhr angefertigt. Dieses ist in Abbildung 15 zu sehen. Diese Untersuchung wurde angefertigt, um Besonderheiten in den zwei Tageshälften zu erkennen, da für viele Systeme ein Großteil der Prozesse um 14 Uhr gestartet werden. Die Einteilung der Antwortzeiten vor 12 Uhr und nach 12 Uhr ergab keine Besonderheiten. In beiden Tageszeiten befinden sich die meisten Werte zwischen 4 und 6 ms. Ein ähnliches Ergebnis ergab die Betrachtung der Antwortzeiten in Verbindung mit den Wochentagen. Dies ist in Abbildung 16 zu sehen.

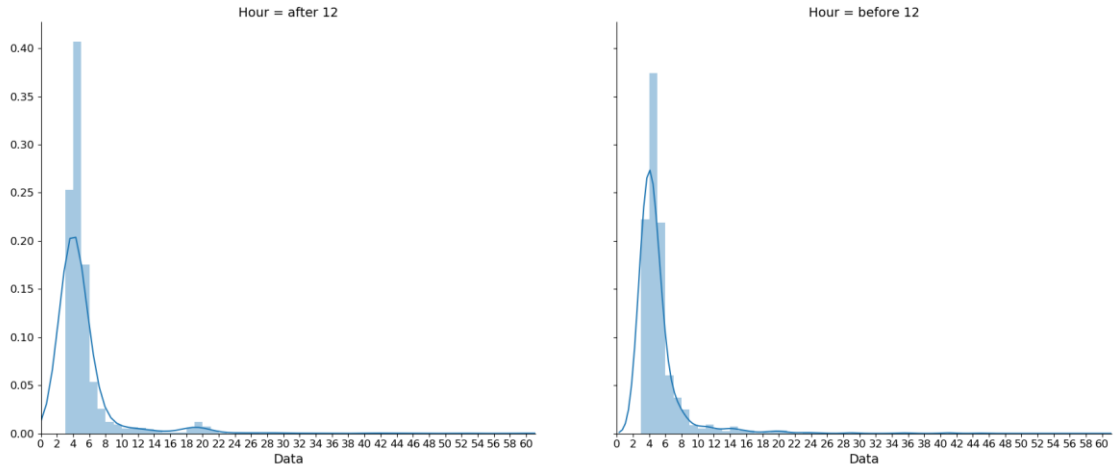

Abbildung 15: Density Plot für die Werte vor 12 Uhr und nach 12 Uhr

Auch bei dieser Betrachtung befinden sich die meisten Werte zwischen 4 und 6 ms. Die einzige Abweichung tritt im Densityplot des Montags. Diese ist flacher als die Kurven der anderen Tage, da die Werte im Bereich von 2-8 gleichmäßiger verteilt sind als an den anderen Tagen

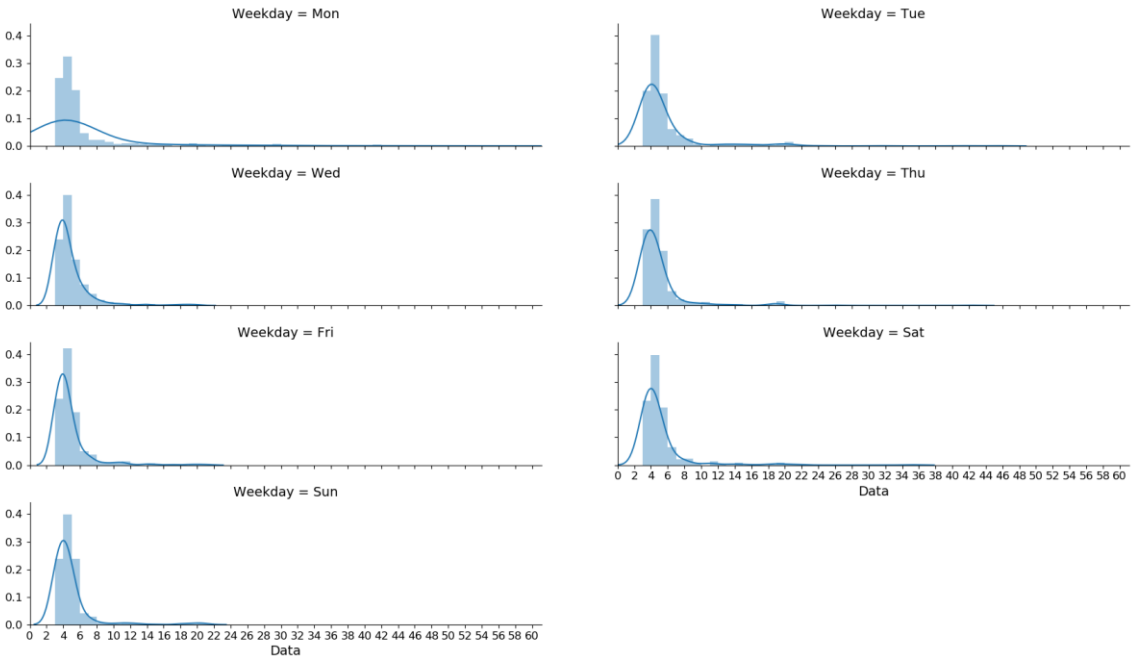

Abbildung 16: Density Plot der Metrik gefiltert nach Wochentag

.

4 Algorithmen zur Erkennung von Anomalien

Um einen geeigneten Anomalieerkennungsalgorithmus zu finden, muss zunächst eine Auswahl an für das Problem geeigneten Algorithmen zusammengetragen werden. Die konkrete Problemstellung gibt Kriterien vor, die die Auswahl an potenziellen Algorithmen einschränken. Da bestimmt werden soll, ob das System sich in einem anomalen Zustand befindet und diese Zustände meist über einen Anfangs- und Endpunkt verfügen, müssen Teilbereiche innerhalb einer Zeitreihe als normal oder anomal klassifiziert werden können. Aus diesem Grund werden Algorithmen näher betrachtet, die fehlerhafte Teilbereiche (Discords) innerhalb einer Zeitreihe erkennen können. Diese Discords können außerdem mithilfe der in 2.4.3 vorgestellten Metriken aufgrund ihrer Rechtzeitigkeit überprüft werden. Des Weiteren handelt es sich bei den Monitoringdaten um Echtzeitdaten, die in einem gleichmäßigen Intervall erhoben werden. Die Auswertung des derzeitigen Zustands des Systems muss demnach nach jedem neu eintreffenden Datensatz stattfinden. Aus diesem Grund müssen die Algorithmen in der Lage sein Echtzeitdaten auszuwerten. Während der Datenanalyse wurde außerdem festgestellt, dass keine Korrelation zwischen den Zeitreihen des Datensatzes herrscht. Demnach genügen Algorithmen, die eine univariate Analyse vornehmen. Um die Auswahl der geeigneten Algorithmen einzuschränken, wurden Algorithmen gewählt, bei denen eine offizielle Referenzimplementierung bereits vorhanden war. Algorithmen wie [[SB14], [AA12], [CDA18], [Fu06], [Wa16], [Pa15a]] konnten dementsprechend in die Auswahl nicht aufgenommen werden, da aufgrund der fehlenden Implementierung die Leistungsfähigkeit der Algorithmen nicht überprüft werden konnte. Des Weiteren wurden keine weiteren Informationen zu diesen Algorithmen außerhalb des jeweiligen vorstellenden Papers gefunden, sodass der Aufwand einer Eigenimplementierung den Rahmen dieser Arbeit überstiegen hätte. Eine Ausnahme bilden CUSUM, COF und

VAE. Diese Algorithmen wurden ausgewählt, da sie aufgrund der umfangreichen Dokumentation mit überschaubarem Aufwand selbstständig implementiert werden konnten.

Während sich diese Arbeit aufgrund des begrenzten Bearbeitungszeitraums auf eine Auswahl an Algorithmen beschränken muss, kann die Abwesenheit von Referenzimplementierungen im allgemeinen wissenschaftlichen Kontext als großes Problem angesehen werden. Wissenschaftliche Arbeiten versuchen aufgrund der Dokumentation des Vorgehens und der Ergebnisse ihrer Untersuchung so nachvollziehbar und reproduzierbar wie möglich zu machen. Durch das Fehlen einer Referenzimplementierung sinkt sowohl die Nachvollziehbarkeit als auch die Reproduzierbarkeit. Während durch Text und Formeln immer Raum für Missinterpretation entsteht, liefert eine Implementierung eine eindeutige Abbildung des in der wissenschaftlichen Arbeit dokumentierten Vorgehens. Des Weiteren können nur mithilfe einer Referenzimplementierung die Ergebnisse einer wissenschaftlichen Arbeit gewissenhaft reproduziert werden, da nur so jedes Detail und ggf. die Fehler reproduziert werden. Da für die Erstellung der Ergebnisse der wissenschaftlichen Arbeit bereits eine Implementierung erzeugt wurde, sorgt die Bereitstellung dieser Implementierung außerdem keinen zusätzlichen Aufwand.

Um die Auswahl der Algorithmen zu strukturieren werden diese aufgrund ihrer Herkunft wie in [[BW20]] eingeteilt. Im Paper werden die Algorithmen deshalb in drei Kategorien eingeteilt. Bei diesen handelt es sich um statistische Ansätze, Machine Learning Ansätze und Deep Learning Ansätze. Im Paper wird die Abgrenzung von Machine Learning und Deep Learning Ansätzen dadurch vorgenommen, dass Deep Learning Ansätze neuronale Netze als Grundlage besitzen. Dabei wird darauf eingegangen, dass in anderen Papern wie [[MSA18]] Algorithmen auf Basis von neuronalen Netzen auch unter Machine Learning zusammengefasst werden. [[PTK19]] beschreibt die Machine Learning als Algorithmen bei denen die Datenerstellung als Black Box angesehen wird und der Algorithmus nur aufgrund der Daten lernt. Dieser Verallgemeinerung wird im Folgenden nicht vorgenommen, da die besonderen Netzformen wie Long-short Term Memory Netze und Convolutional Neural Networks für bestimmte Datenformen erstellt wurden und somit den Datenerstellungsprozess zum Teil mit modellieren. Dementsprechend sind die Algorithmen im Folgenden in die Kategorien Statistik, Machine Learning und Deep Learning eingeteilt, wobei CUSUM der einzige Algorithmus aus der Statistik ist.

4.1 CUSUM

Die meisten Kontrollkarten betrachten eine statische Größe als Grundlage für die Entscheidungsfindung, wodurch nur kurzfristige starke Veränderungen erkannt werden. Solche Kontrollkarten ohne Gedächtnis sind nicht dafür geeignet langsame Veränderungen zu entdecken. Um solche Veränderungen möglichst zeitnah zu erkennen werden deshalb Kontrollkarten mit Gedächtnis benutzt. Bei einer dieser Kontrollkarten mit Gedächtnis handelt es sich um die CUSUM-Kontrollkarte, die die gesamte Menge an Werten zur Entscheidungsfindung benutzt. Die CUSUM-Kontrollkarte kann dabei mithilfe der tatsächlichen Einzelwerte X_j der Metrik oder mit einem Mittelwert von Untergruppen $\bar{X}_j$ benutzt werden. Die Untergruppen besitzen dabei eine einheitliche Länge n. Im Folgenden wird auf die Betrachtung von Einzelwerten näher eingegangen.

Dazu wird zunächst der Sollwert μ_0 und die Standardabweichung σ der Metrik festgelegt. Außerdem müssen zwei Konstanten $d, h > 0$ gewählt werden, mit denen die Referenzwerte k und H wie in Gleichung (4.1.1) erzeugt werden.

$$k = d\sigma, \quad H = B\sigma \tag{4.1.1}$$

Die Referenzwerte dienen als Kontrollgrenzen, wobei k die erste Kontrollgrenze darstellt und H die zweite Kontrollgrenze. Erst nach überschreiten von H wird der Prozess als fehlerhaft angesehen. Die Referenzwerte werden dabei als obere, als auch untere Grenze benutzt. Für die Konstante d wird standardmäßig 0.5 benutzt, wodurch eine Veränderung der Metrik von 1σ optimal erkannt werden kann. Die Konstante B wird meist auf Werte zwischen 3 und 5 gesetzt, wobei ein kleinerer Wert zu einem schnelleren Überschreiten der Kontrollgrenze führt. Dies kann jedoch auch zu einem blinden Alarm führen. Für die Überprüfung eines Wertes X_j einer Metrik zum Zeitschritt j wird zunächst der Sollwert μ_0 abgezogen, sodass die Abweichung vom Sollwert festgestellt wird. Das Ergebnis wird auf das Überschreiten der oberen Kontrollgrenze k überprüft. Sollte dieser Wert größer als 0 sein, wird dies als ein Indiz für die langsame Veränderung des Prozesses angesehen. Die Überschreitung der ersten Grenze wird in S_j festgehalten, wobei S_j^+das Maß der Überschreitung der oberen Grenze k ist und S_j^- die Überschreitung der unteren Grenze. Für diese beiden Maße lassen sich die Folgenden Rekursionsformeln aufstellen.

$$S_j^+ = \max\left(S_{j-1}^+ + \left(X_j - \mu_0 - k\right), 0\right) \quad (4.1.2)$$

$$S_j^- = \min\left(S_{j-1}^- + \left(X_j - \mu_0 + k\right), 0\right) \quad (4.1.3)$$

Nach der Berechnung von S_j^+ und S_j^- zu Zeitschritt j werden die Summen mit der zweiten oberen und unteren Grenze H bzw. $-H$ verglichen. Sollte S_j^+ oder S_j^- die Grenze H bzw. $-H$ überschreiten wird der Prozess als fehlerhaft angesehen. Um den potenziellen Anstieg der Summe pro Zeitschritt zu begrenzen, wurde der CUSUM Algorithmus modifiziert. Dadurch kann S_j^+ bzw. S_j^- nur um einen Bruchteil von H ansteigen. Der Anteil von H um den S_j^+ bzw. S_j^- ansteigen kann wird dabei durch die Sequenzlänge sl angegeben. Dementsprechend bedeutet $sl = 2$, dass zwei Werte k überschreiten müssen, damit h überschritten werden kann. Eine einzelne Anomalie kann somit nicht dafür sorgen, dass S_j^+ bzw. S_j^- die Grenze H bzw. $-H$ überschreiten. Dadurch ergeben sich für die Gleichungen 4.1.2 und 4.1.3 folgende modifizierte Gleichungen 4.1.4 und 4.1.5.

$$S_j^+ = \max\left(\min\left(S_{j-1}^+ + \left(X_j - \mu_0 - k\right),\ S_{j-1}^+ + \frac{H}{sl}\right), 0\right), \forall sl > 0 \quad (4.1.4)$$

$$S_j^- = \min\left(\max\left(S_{j-1}^+ + \left(X_j - \mu_0 + k\right),\ S_{j-1}^+ - \frac{H}{sl}\right), 0\right), \forall sl > 0 \quad (4.1.5)$$

Im folgenden Quelltext ist eine Implementierung von CUSUM zu sehen, wobei nur Übertretungen der oberen Grenze überprüft wird. In Zeile 1-4 werden zunächst Variablen initialisiert, die im Laufe des Algorithmus gebraucht werden. In Zeile 5-21 wird jeder Zeitschritt in der Zeitreihe einzeln ausgewertet. Dafür wird in Zeile 6-9 je nach sl die Summe traditionell berechnet oder die Modifikation aus Gleichung 4.1.4 benutzt. Die entstehende Summe wird in „plot_sum“ für das spätere Plotten der Ergebnisse gespeichert. In Zeile 11-12 wird überprüft, ob S_j^+ 0 ist, sodass der potenzielle Startpunkt einer Anomalie „anomaly_start“ auf „None“ gesetzt werden kann. Sollte S_j^+ größer als 0 und „anomaly_start“ auf „None“ gesetzt sein, wird in Zeilen 14-15 der derzeitige Zeitschritt als Beginn der Anomalie angenommen. In den Zeilen 17-21 wird die Summe auf die zweite Kontrollgrenze H überprüft. Sollte die Summe H überschreiten, wird den Anomalien der Bereich von „anomaly_start“ bis zum derzeitigen Zeitschritt hinzugefügt. In Zeile 18 wird für das Plotten außerdem überprüft, ob im selben Zeitschritt k und H überschritten wurden,

wodurch der anomale Bereich den gleichen Start und Endpunkt haben würde. Um die Anomalie trotzdem plotten zu können wird dem Endpunkt ein Zeitschritt angehangen.

```
1.  sum = 0
2.  anomalies = []
3.  anomaly_start = None
4.  plot_sum = []
5.  for i, date in enumerate(data):
6.      if sl is None:
7.          sum = max(sum + (date - mean - k), 0)
8.      else:
9.          sum = max(min(sum + (date - mean - k), sum + (h/sl)), 0)
10.     plot_sum.append(sum)
11.     if sum == 0 and anomaly_start is not None:
12.         anomaly_start = None
13.
14.     if sum > 0 and anomaly_start is None:
15.         anomaly_start = i
16.
17.     if sum >= h:
18.         if anomaly_start == i:
19.             anomalies.append((anomaly_start, i + 1))
20.         else:
21.             anomalies.append((anomaly_start, i))
```

Quelltext 1: CUSUM-Algorithmus in Python

Das Vorgehen und Ergebnis des modifizierten CUSUM ist in Abbildung 17 dargestellt. Zu erkennen sind dabei die beiden Grenzen k in Grün und H in Rot, sowie die kumulative Summe S_j^+ in Orange.

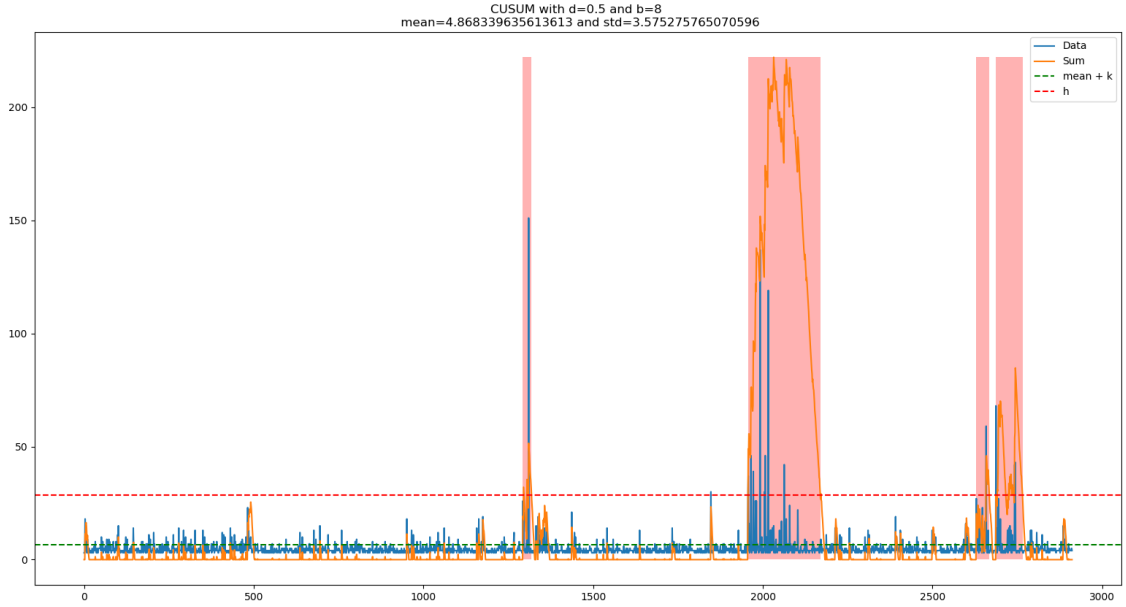

Abbildung 17: Darstellung des Vorgehens von CUSUM

4.2 Machine Learning

In den folgenden Kapiteln werden Algorithmen vorgestellt, die in zur Kategorie der Machine Learning Algorithmen gehören. Diese Algorithmen analysieren die gegeben Daten selbstständig, sodass Anomalien automatisch erkannt werden. Jeder Algorithmus benötigt jedoch bestimmte Variablen, die zu Beginn vom Benutzer gesetzt werden müssen und maßgeblich die Genauigkeit des Algorithmus beeinflussen.

4.2.1 Isolation Forest

Isolation Forest ist ein Machine Learning Ansatz der in [[FKZ08]] zuerst vorgestellt wurde, sodass dieses Paper als Grundlage für die weiteren Erläuterungen dienen soll. Der Isolation Forest Algorithmus erkennt Anomalien in einem Datensatz aufgrund der Isolation des Datenpunktes von anderen Datenpunkten im Datensatz. Ein Isolation Forest bestehen aus einem Ensemble von Isolation Trees. Diese Isolation Trees sind binäre Bäume mit deren Hilfe Datenpunkte isoliert werden können. Innerhalb der Isolation Trees wird angenommen, dass Anomalien stärker isoliert sind von den anderen Punkten im Datensatz. Diese stärkere Isolation sorgt dafür, dass sich die Anomalie näher an der Wurzel des Isolation Trees befindet, da die Isolation der Anomalie in weniger Schritten erreicht werden kann [[LTZ12], S. 4]. In der nachfolgenden Abbildung wird die Erkennung von normalen Datenpunkten und Anomalien graphisch dargestellt.

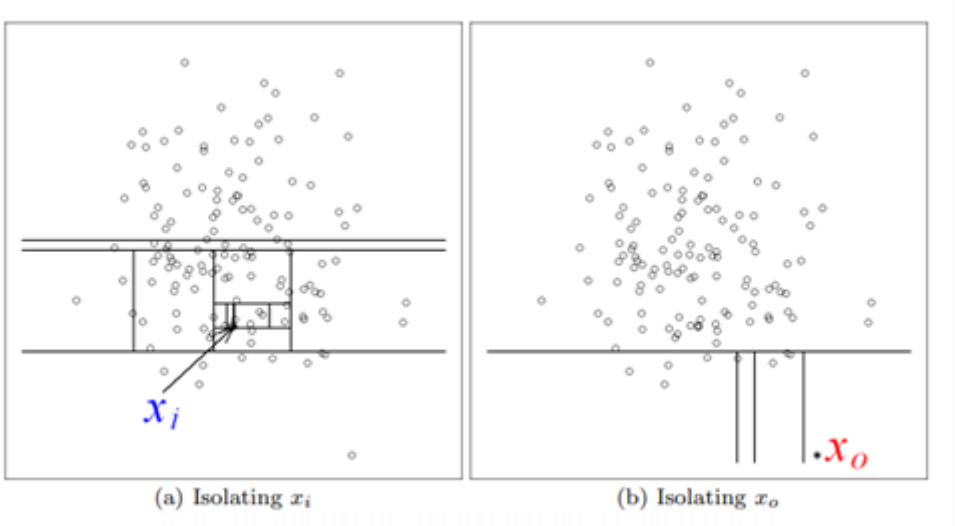

Abbildung 18: Darstellung der benötigten Schritte zur Isolierung eines normalen Datenpunktes und einer Anomalie [[FKZ08], S. 415]

Dabei ist zu erkennen, dass es zur eindeutigen Isolation von Datenpunkt x_i mehr Einteilungen braucht, als für den Datenpunkt x_o, da es sich bei Datenpunkt x_o um eine Anomalie handelt. Da diese Einteilungen in einem binären Baum über die Entscheidungsknoten realisiert werden, ist zu erkennen, dass zur Isolation von x_o weniger Entscheidungsknoten durchlaufen werden müssen. Demnach liegen Anomalien aufgrund ihrer Isoliertheit näher an der Wurzel der Bäume.

Zur Anomalieerkennung wird ein Ensemble mithilfe von Bagging benutzt. Das Ensemble aus T Teilbäumen wird mit Bagging erzeugt, indem der Datensatz X in T Teilstichprobe $X_t' \subset X$ geteilt wird. Dabei können sich Elemente aus X mehrmals oder kein einziges Mal in den Teilmengen X_t' befinden. Dieser Vorgang wird wiederholt bis die Anzahl der X_t' T entspricht, sodass T Isolation Trees erzeugt werden können. Die Bestimmung der Anomalien erfolgt über die Kombination der Entscheidungen der Bäume. Die durch Bagging entstehenden Isolation Trees können aufgrund von mehrmals vorkommenden Werten einen höheren Testfehler erzeugen als ein Isolation Tree mit dem gesamten Datensatz. Wenn diese jedoch kombiniert werden, reduziert sich der Fehler im Vergleich zu einem Isolation Tree der mit dem gesamten Datensatz erstellt wurde. [[RD99], S. 172]

Für die Erzeugung eines Isolation Trees wird ein X_i' aus X_t' benutzt. Die Isolation Trees werden erzeugt, indem ein Attribut $q \in Q$ zufällig aus der Teilstichprobe X_i' ausgewählt wird. Für dieses Attribut wird dann ein zufälliger Spaltwert p gewählt, der zwischen dem Maximum und Minimum von q in X_i' liegt. Der Vergleich von q mit p sorgt dann für die Zuweisung zu einem der beiden Tochterknoten. Im Tochterknoten erfolgt dann derselbe Ablauf mit der Teilmenge von X_i', die die jeweilige Bedingung ($q < p$, $q \geq p$) erfüllt. Dieser Vorgang wird für den Isolation Tree solange wiederholt, bis die Teilmenge von X_i' nicht weiter durch einen Spaltwert aufgeilt werden kann. Dieser Fall trifft ein, falls die Teilmenge von X_i' nur noch aus einem Datum besteht oder die Attribute der Daten jeweils den gleichen Wert besitzen. Diese Konstruktion wird für jedes X_i' wiederholt, bis T Isolation Trees erzeugt wurden.

Damit ein Datenpunkt als Anomalie gekennzeichnet werden kann, muss zuerst die Tiefe des von ihm durchlaufenen Pfades bestimmt werden. Dafür wird Länge des durchlaufenen Pfades e für jeden passierten Entscheidungsknoten um eins erhöht. Durch e lässt sich $h(x)$ wie in Gleichung (4.2.1) bestimmen.

$$h(x) = e + c(Anzahl\ Punkte\ in\ X_i') \qquad (4.2.1)$$

Im Anschluss wird $c(n)$ berechnet. Dieser Wert entspricht der durchschnittlichen Pfadlänge eines binären Suchbaums, der mithilfe von n Datenpunkten konstruiert wurde. Dieser Wert wird durch

$$c(n) = \begin{cases} 2H(n-1) - \dfrac{2(n-1)}{n} & f\ddot{u}r\ n > 2 \\ 1 & f\ddot{u}r\ n = 2 \\ 0 & sonst \end{cases} \qquad (4.2.2)$$

berechnet, wobei $H(i)$ der harmonischen Zahl $H(i) \approx \ln(i) + \gamma$ und γ der Euler-Mascheroni-Konstante entspricht. Damit die Pfadlänge der verschiedenen Isolation Trees miteinander verglichen werden können, muss $h(x)$ normalisiert werden, um die unterschiedlichen Baumgrößen zu kompensieren. Dadurch kann ein Anomalie Score s berechnet mithilfe von Gleichung (4.2.3) berechnet werden.

$$\mathrm{s(x, n)} = 2^{-\frac{E(h(x))}{c(n)}} \qquad (4.2.3)$$

Dabei entspricht $E(h(x))$ der durchschnittliche Pfadlänge der Isolation Trees. Mithilfe von s kann dann entschieden werden, ob es sich bei dem Datenpunkt um eine Anomalie handelt. Dabei werden Werte von nahe 1 als Anomalien klassifiziert. Werte die kleiner sind als 0.5 werden als normale Datenpunkte eingestuft. Sollten für alle Datenpunkte in X ein Wert um 0.5 berechnet werden, wird davon ausgegangen, dass die Stichprobe keine Anomalien enthält. Da der Isolation Forest den gesamten Datensatz benutzt, um normale von anomalen Daten zu trennen, handelt es ich um eine unsupervised Anomalieerkennungsalgorithmus.

In [[Di13]] wird der Algorithmus erweitert, sodass ein Concept Drift erkannt werden kann. Der Concept Drift beschreibt die Änderung der Streamingdaten, sodass eine erneute Anpassung des Isolation Forests nötig ist. Dafür wird der Datensatz in Zeitblöcke der Größe eingeteilt sodass sich der erste Zeitblock Z_1 ergibt als $Z_1 = (t^1, t^2, \dots, t^M)$. Die Zeitschritte $t^1, \dots, t^m$ entsprechen Streamingdaten, die zu jedem Zeitschritt neu erfasst werden. Nachdem ein Zeitblock vom Isolation Forest analysiert wurde, wird die Anzahl der Anomalien im Zeitblock bestimmt. Sollte die Anzahl über einer vorher festgelegten

Schwelle u liegen, wird davon ausgegangen, dass ein Concept Drift vorliegt und der Isolation Forest mit Berücksichtigung des Zeitblocks neu trainiert werden muss. Zur Festlegung der Schwelle u wird eine geschätzte Anomalierate benötigt. [[Di13], S. 14f.]

Für die Benutzung der Isolation Forest wird die Implementierung von scikit-learn [[Pe11]] benutzt. In dieser ist der oben beschriebene Algorithmus in der Klasse IsolationForest implementiert. Im folgenden Quelltext wird die Benutzung der „IsolationForest“ Klasse gezeigt.

```
1. rng = np.random.RandomState(42)
2. isolation_forest = IsolationForest(max_samples=100, ran-
   dom_state=rng, contamination=0.01)
3. isolation_forest.fit(y_train)
```

Quelltext 2: Implementierung des Isolation Forest Algorithmus mit scikit-learn in Python

Im Konstruktor der „IsolationForest“ Klasse, zu sehen in Zeile 2, können mehrere Parameter festgelegt werden, die das Verhalten des Isolation Forests ändern. Der Parameter „max_samples“ gibt die Größe des Datensatzes an, mit dem jeder Isolation Tree trainiert werden soll. Die Anzahl der Isolation Trees ist dabei standardmäßig auf 100 eingestellt und kann über den Parameter „n_estimators“ geändert werden. Da für die Erzeugung des Isolation Forest der Datensatz zufällig aufgeteilt wird, kann zur Reproduzierbarkeit ein eigener Pseudozufallszahlengenerators über den Parameter „random_state“ übergeben werden. Dieser Zufallsgenerator wird in Zeile 1 mit dem Startzustand 42 initialisiert. Über den Parameter „contamination“ kann der prozentuale Anteil der Anomalien im Datensatz festgelegt werden. In Zeile 3 wird der Isolation Forest anhand des Datensatzes trainiert, sodass dieser in Zeile 4 über die „predict“-Funktion Datenpunkte als normal oder anomal klassifizieren kann. Das Ergebnis, welches in „y_pred_test“ gespeichert wird ist ein Array, wobei jeder Eintrag für das Ergebnis eines Datenpunktes aus „y_test“ steht. Sollte es sich um einen normalen Punkt handeln, besitzt der Eintrag den Wert 1. Sollte es sich um eine Anomalie handeln -1.

4.2.2 DBSCAN

Bei DBSCAN (Density-Based Spatial Clustering of Applications with Noise) handelt es sich um einen dichtebasierten Clustering Algorithmus. In der Folgenden Erklärung wird sich auf das Paper [[Es96]] bezogen, indem DBSCAN das erste Mal vorgestellt wurde. DBSCAN benötigt zur Erkennung dieser Anomalien zwei Parameter, die zu Beginn des Algorithmus vom Benutzer festgelegt werden müssen. Bei diesen handelt es sich um den Abstand zu benachbarten Punkten $epsilon$ und die minimale Anzahl von benachbarten Punkten $minpts$.

Für einen gegebenen Punkt ist jeder andere Punkt, dessen Abstand kleiner als $epsilon$ ist, ein Nachbar. Sollte für einen Punkt die Anzahl der dadurch entstehenden Nachbarn größer sein als die durch $minpts$ angegebene Anzahl an Nachbarn, handelt es sich bei dieser Gruppe an Punkten um einen Cluster. Für die Erkennung von Anomalien sorgt die Einteilung der Punkte in Core Points (Kernpunkte), Border Points (Randpunkte) und Outlier (Anomalien). Bei den Core Points handelt es sich um die Punkte, die mindestens $minpts$ Nachbarn besitzen und damit den Cluster aufbauen. Bei den Border Points handelt es sich um Nachbarn von Core Points, die jedoch selbst keine Core Points sind. Sie sind demnach weniger als $epsilon$ von einem Core Point entfern, besitzen jedoch nicht mindestens $minpts$ Nachbarn. Jeder Punkt, der kein Core Point oder Border Point ist, zählt als Anomalie.

Da Nachbarn durch die Nähe zu Core Points entstehen, können sich Cluster aufgrund gemeinsamer Nachbarn miteinander verbinden. Um die übermäßige Verbindung von Clustern zu vermeiden, sollte der Parameter $epsilon$ nicht zu groß gewählt werden, da sich sonst Cluster mit großen Abständen einfacher zusammenschließen. Aus dem gleichen Grund sollte nicht zu gering gewählt werden, da sonst viele kleine Cluster entstehen können, die sich aufgrund ihrer Nachbarn verbinden. Außerdem sorgt eine zu hohe Anzahl an $minpts$ dazu, dass normale Datenpunkte aufgrund ihrer wenigen Nachbarn als Anomalien klassifiziert werden. [[CDD11], S. 92] Eine Veranschaulichung dieses Problem ist in Abbildung 19 zu sehen. Bei einer Anzahl $minpts$ von 4 würden die Punkte in der Gruppe 3 und 5 als Anomalien erkannt werden, obwohl Gruppe 3 relativ nah an den anderen Clustern ist. Sollte nun $epsilon$ größer als in der Abbildung gewählt werden, könnten sich die Cluster miteinander verbinden, was auch zur Folge haben würde, dass Gruppe 3 nun zu einem

anderen Cluster dazugehört. Da der DBSCAN Algorithmus selbständig normale von anomalen Daten trennt, handelt es sich um eine unsupervised Anomalieerkennung.

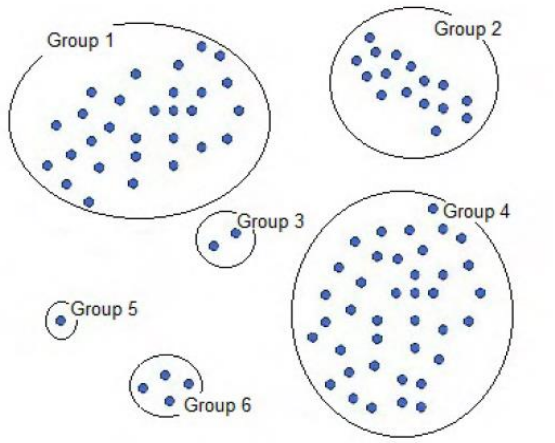

Abbildung 19: Beispiel eines Clustering mit DBSCAN [[CDD11], S. 92]

Außerdem wurde in [[Es98]] der Algorithmus erweitert, sodass er für Streamingdaten angepasst wurde. Durch diese Anpassung können Punkte dem Datensatz hinzugefügt werden, woraufhin sich das Clustering ändern kann. Um die Performance für Streamingdaten anzupassen werden beim Einfügen von Daten nur die Punkte in der Umgebung betrachtet, sodass nicht der gesamte Algorithmus wiederholt werden muss.

Für die Benutzung des DBSCAN Algorithmus für Streamingdaten wurde die Implementierung von [[Cl15]] benutzt. Die Benutzung dieser Implementierung ist im folgenden Quelltext zu sehen.

```
1.  scan = ddbscan.DDBSCAN(epsilon, min_pts)
2.  start = 10
3.  for point in y_sequenced[:start]:
4.    scan.add_point(point=point.tolist(), count=1, desc="")
5.  scan.compute()
6.  for i, stream_point in enumerate(y_sequenced[start:]):
7.    scan.add_point(point=stream_point.tolist(), count=1, desc="")
8.    scan.compute()
9.    for j, point in enumerate(y_sequenced[:start+i]):
10.     scan_point_index = scan.points.index(point.tolist())
11.     point_data = scan.points_data[scan_point_index]
12.       if point_data.cluster == -1:
13.         anomal_sequence = (j, j + sequence_length)
14.         anomalies.append(anomal_sequence)
15. uniq = [list(x) for x in set(tuple(x) for x in anomalies)]
```

Quelltext 3: Implementierung des DBSCAN Algorithmus für Streamingdaten in Python

In Zeile 1 wird dabei das „DBSCAN“ Objekt mit "$epsilon$" und "min_pts" erstellt, sodass in Zeile 2-4 die Daten an das „DBSCAN“ Objekt übergeben werden können und in 5 der Algorithmus gestartet wird. Dabei werden zunächst 10 Punkte hinzugefügt, sodass ein grundlegender Datenbestand besteht. In Zeile 6-14 wird schrittweise ein neuer Punkt hinzugefügt und die Cluster berechnet. Dafür wird in Zeile 9-12 für jeden bisher hinzugefügten Punkt die Zugehörigkeit zu einem Cluster bestimmt. Sollte einer der Punkte zu keinem Cluster gehören, besitzt er liefert „point_data.cluster“ -1 zurück. Wenn dies der Fall ist, wird in Zeile 13-14 der Punkt der Liste von Anomalien „anomalies“ hinzugefügt. In Zeile 15 werden zum Abschluss alle doppelten anomalen Bereiche aussortiert.

4.2.3 COF

Der Connectivity-Based Outlier Factor (COF) aus [[Ji02]] ist eine Erweiterung des Lokal Outlier Factor (LOF). Diese Erweiterung betrifft die getrennte Betrachtung der Isoliertheit eines Punktes und der Dichte der umliegenden Punkte. Eine geringe Dichte resultiert aus der Abwesenheit von Punkte in der unmittelbaren Nähe. Isoliertheit beschreibt die Zugehörigkeit eines Punktes zu anderen, sodass eine Struktur erkennbar ist. Punkte, die isoliert sind, treten deshalb meist in Gebieten mit geringer Dichte auf, jedoch weisen Gebiete mit geringer Dichte nicht auch eine große Anzahl an isolierten Punkten auf. COF versucht beide dieser Kriterien bei der Erkennung von Anomalien einzubeziehen.

Der Connectivity-Based Outlier Factor wird für jeden Punkt im Datensatz berechnet. Für einen Punkt p_1 werden k Punkte bestimmt, deren Abstand zu p_1 minimal ist. (engl. K-Nearest Neighbor) Der Parameter k muss dabei zu Beginn durch den Benutzer festgelegt werden. Die sich daraus ergebende Satz von Punkten (engl. Set) wird als $N_k(p)$ bezeichnet.

Danach wird der $set\ based\ nearest\ (SBN)\ path\ s_x$ bestimmt. Dieser besteht aus der Verbindung der Punkte im Satz $N_k(p_1)$, sodass für jeden Punkt im Satz der geringste Pfad zu einem anderen Punkt im Satz gefunden wird. Dabei wird mit dem Punkt p_1 begonnen und dessen nähester Punkt in $N_k(p_1)$ gefunden. Mit dem dadurch gefunden Punkt wird im Anschluss erneut der Punkt mit dem geringsten Abstand in $N_k(p_1)$ gefunden, wobei Punkte nur einmal miteinander verbunden werden können. s_1 besteht dann aus den nacheinander gefunden Punkten $s_1 = \langle p_1, p_2, p_3, p_4 \dots, p_k \rangle$. [[Ji02], S. 541] Daraus kann der $SBN\ trail\ tr_1$ bestimmt werden. Dieser besteht aus den Kanten zwischen den Punkten in s_1.

Der $SBN\ trail\ tr_1$ besitzt dann die Form $tr_1 = \langle (p_1, p_2), (p_2, p_3), \dots, (p_{k-1}, p_k) \rangle$. Da die Paare in tr_1 für Kanten stehen können für die Paare die Kosten der Kanten berechnet werden. Diese ergeben sich aus dem Abstand zwischen den Punkten und werden durch $C_1 = \langle c_1, c_2, \dots, c_{k-1} \rangle$ beschrieben.

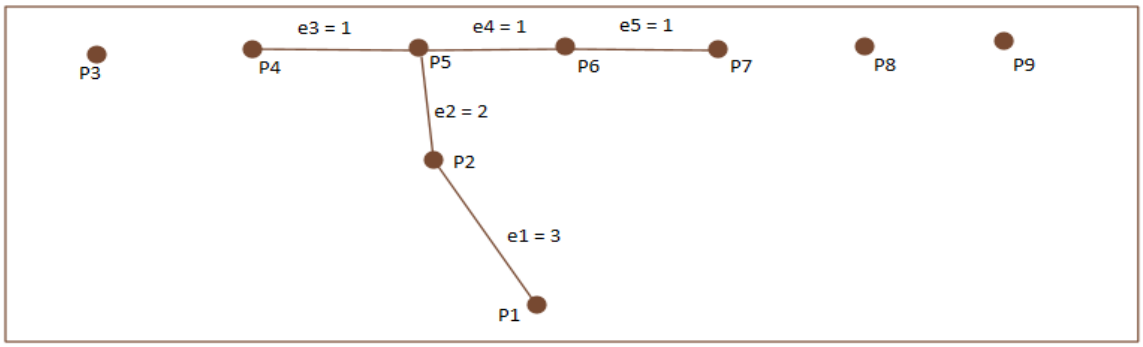

Abbildung 20: Beispiel für die Bildung eines SBN trails für den Punkt p1 [[Ma18]]

In Abbildung 20 ist ein Beispiel für den oben genannten Ablauf zu sehen. Im Beispiel befinden sich die Punkte $p_1, \dots, p_9$ wobei $N_k(p_1)$, s_1 und C_1 berechnet werden sollen für k=5. Dazu werden zuerst die Punkte mit der niedrigsten Distanz zu p_1 bestimmt durch $N_k(p_1) = \{p_2, p_5, p_4, p_6, p_7\}$. Mithilfe von $N_k(p_1)$ kann dann $s_1 = \langle p_2, p_5, p_4, p_6, p_7 \rangle$ und $tr_1 = \langle (p_1, p_2), (p_2, p_5), (p_5, p_4), (p_5, p_6), (p_6, p_7) \rangle$ bestimmt werden. Durch tr_1 können im Anschluss die Kosten für die in tr_1 gefunden Kanten berechnet werden als $C_1 = \{3, 2, 1, 1, 1\}$. Mithilfe der Kosten C_1 kann die $average\ chaining\ distance\ ac-dist_{N_k(p)}(p)$ für p_1 berechnet werden.

$$ac - dist_{N_k(p_1)}(p_1) = \sum_{i=1}^{k} \frac{2(k+1-i)}{k(k+1)} \cdot c_i \qquad (4.2.4)$$

Dabei dient der Bruch nach dem Summenzeichen als Gewichtung der Distanzen, die dazu führt, dass Kanten die Nahe an p_1 liegen einen größeren Einfluss auf die Gesamtsumme besitzen. Dadurch erhalten Punkte eine höhere $ac - dist$, die nah an miteinander verbundenen Strukturen liegen.

Durch die $ac - dist$ kann dann der Connectivity-Based Outlier Factor $COF_k(p)$ berechnet werden.

$$COF_k(p) = \frac{|N_k(p_1)| \cdot ac - dist_{N_k(p_1)}(p_1)}{\sum_{o \in N_k(p_1)} ac - dist_{N_k(o)}(o)} \quad (4.2.5)$$

Dabei ist $|N_k(p_1)|$ die Anzahl der Punkte im Set $N_k(p_1)$ und o die anderen Punkte im Set $N_k(p_1)$. Dadurch kann $COF_k(p)$ als Verhältnis zwischen der $average\ chaining\ distance$ von p_1 zur durchschnittlichen $average\ chaining\ distance$ der anderen Punkte o im Set $N_k(p_1)$ betrachtet werden. Da COF selbständig normale Werte von Anomalien trennt, handelt es sich bei diesem um eine unsupervised Anomalieerkennung.

Im Paper [[Dr08]] wurde der Algorithmus erweitert, sodass der Algorithmus für Streamingdaten benutzt werden kann. Demnach müssen beim Hinzufügen eines Datenpunktes p der $COF_k(o)$ von o nur geändert werden a) falls sich $ac - dist_{N_k(o)}(o)$ ändert b) der Datenpunkt p sich unter den k nähesten Punkten $N_k(o)$ befindet c) sich $ac - dist_{N_k(N_k(o))}(N_k(o))$ von einem der k nähesten Punkten $N_k(o)$ von o ändert.

Aufgrund der Tastsache, dass keine offizielle Implementierung des Algorithmus in [[Dr08]] vorhanden ist, wird für die Benutzung des Connectivity Outlier Factors eine Eigenimplementierung benutzt. Im Folgenden Quelltext wird beispielhaft die Bestimmung des $SBN\ trails$ für einen Punkt p gezeigt. Der Funktion „__get_sbn_trail“ wird ein Punkt vom Typ „COFPoint“ übergeben. Bei dieser Klasse handelt es sich um eine Datenstruktur, die neben den Koordinaten des Punktes auch die Indizes der K-Nearest Neighbors sowie deren Distanz zum Punkt speichert. Da der $SBN\ trail$ aufgrund der K-Nearest Neighbor erstellt wird, werden diese in Zeile 2 mithilfe der Indizes vom Punkt bestimmt. Im Anschluss wird ein k-d-Baum mithilfe der Punkte erzeugt, sodass die Distanzen zwischen den K-Nearest Neighbors schneller bestimmt werden kann.

```
1.  def __get_sbn_trail(self, point: COFPoint):
2.    neighbor_points = self.__get_nearest_neighbor(point)
3.    local_tree = cKDTree([one_point.coordinate for one_point
      in neighbor_points])
4.    local_tree_data = local_tree.data
5.
6.    path_indices, path_distance = [], []
7.
8.    neighbor_distances, neighbor_indices =
      local_tree.query(point.coordinate, k=self.k)
9.    connected_points = list(neighbor_indices[:1])
10.   not_connected_points = neighbor_indices[1:]
11.   for not_connected_point in not_connected_points:
12.     distances_to_connnected_points = []
13.     for connected_point in connected_points:
14.       distance = np.linalg.norm(local_tree_data[not_connected_point]
          - local_tree_data[connected_point])
15.       distances_to_connnected_points.append(distance)
16.     shortest_distance_index =
        np.argsort(distances_to_connnected_points)[0]
17.     path_indices.append((connected_points[shortest_distance_index],
        not_connected_point))
18.     path_distance.append(
        distances_to_connnected_points[shortest_distance_index])
19.     connected_points.append(not_connected_point)
20.
21.   return path_indices, path_distance, local_tree_data
```

Quelltext 4: Implementierung der Bestimmung des SBN trails für einen Punkt in Python

In Zeile 8 werden die Distanzen und Indizes der K-Nearest Neighbors bestimmt, wobei die Indizes von der Speicherung innerhalb des k-d-Baums abhängen. In Zeile 9 und 10 werden zwei Listen erstellt, die die bereits verbundenen Punkte „connected_points“ und die noch nicht verbundenen Punkte „not_connected_points“ speichern. In Zeile 11-16 wird für jeden Punkt in „not_connected_points“ ein Punkt aus „connected_points“ bestimmt, der am nächsten liegt. Dadurch kann in Zeile 17-19 schrittweise der *SBN trail* bestimmt werden, indem nach jeder Verbindung der verbundene Punkt den „connected_points“ hinzugefügt wird. In Zeile 21 werden die Indizes der verbundenen Punkte, deren Distanzen und der k-d-Baum zurückgegeben. Die Indizes, sowie der k-d-Baum werden nur für die graphische Darstellung des *SBN trails* zurückgegeben, da für die Berechnung der *average chaining distance* nur die Distanzen des *SBN trails* benötigt werden.

4.2.4 Support Vector Machine

Support Vector Machines ist ein Machine Learning Algorithmus der in der Regression, Klassifikation und auch Anomalieerkennung benutzt werden kann. Die für die Erkennung von Anomalien benutzten Support Vector Machines werden dabei als $\nu - Support\ Vector\ Machines$ bezeichnet. Um die Anomalieerkennung mit Hilfe der Support Vector Machines zu erklären, soll zuerst erklärt werden, wie diese zur Klassifikation funktionieren.

Im Allgemeinen erzeugen Support Vector eine lineare Grenzfunktion, wie zum Beispiel $\theta^T x = 0$, die die Eingabedaten optimal voneinander trennt. Im Fall der Klassifikation werden die Klassen im Datensatz mithilfe der Grenzfunktion voneinander getrennt, sodass die Entscheidung der Zugehörigkeit aufgrund der Position des Punktes zu Grenzfunktion getroffen wird. Dabei findet die Einteilung in $\theta^T x > 0$ oder $\theta^T x < 0$ statt, sodass der Punkt auf einer der beiden Seiten der Grenzfunktion liegt. Je weiter der Punkt von der Grenzfunktion entfernt ist, desto klarer ist die Klassifikation der Support Vector Machine. [[Sc99], S. 583]

Um eine solche lineare Grenzfunktion zu finden, wird eine Hyperebene gesucht, deren Abstand zu den Punkten im Datensatz maximal ist. Dazu werden die Funktionen $\hat{\gamma}^{(i)} = y^{(i)}(w^T x^{(i)} + b)$ bestimmt, die orthogonal zur Grenzfunktion liegen, wobei $y^{(i)}$ dem Abstand von $x^{(i)}$ zur Grenzfunktion entspricht. [[An19], S. 12] Eine Grenzfunktion mit orthogonalem Vektor w und Abstand $y^{(i)}$ ist in der folgenden Abbildung zu sehen.

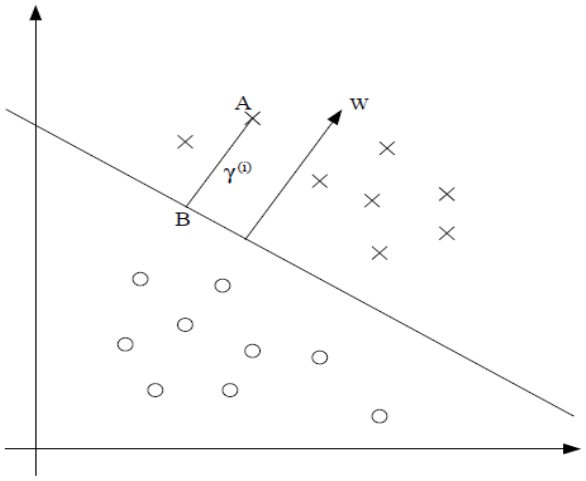

Abbildung 21: Darstellung einer Grenzfunktion mit orthogonalem Vektor w [[An19], S. 14]

Der Vektor w beschreibt dabei einen Vektor der orthogonal zur Grenzfunktion steht. Um $\gamma^{(i)}$ zu bestimmen wird der Vektor w benutzt in

$$\gamma^{(i)} = y^{(i)}\left(\left(\frac{w}{||w||}\right)^T x^{(i)} + \frac{b}{||w||}\right) \quad (4.2.6)$$

wobei $y^{(i)}$ der Abstand des Datenpunktes ist, $\frac{w}{||w||}$ der Einheitsvektor von w und $\frac{b}{||w||}$ der normalisierte lineare Faktor der Funktion. Für alle entstehenden $\gamma^{(i)}$ aus $x^{(i)}$ und $y^{(i)}$ kann dann die Funktion $\gamma^{(i)}$ mit dem kleinsten Abstand von der Grenzfunktion gefunden werden als $\gamma = \min_{i=1,\dots,m} \gamma^{(i)}$[[An19], S. 15]

Dieser kleinste Abstand aus allen $\gamma^{(i)}$ soll maximiert werden, sodass das Problem als

$$max_{\hat{\gamma},w,b} \frac{\gamma}{||w||} \; mit \; y^{(i)}\left(w^T x^{(i)} + b\right) \geq \gamma \quad (4.2.7)$$

formuliert werden. Dies kann in ein Minimierungsproblem umgewandelt werden, indem der Abstand $\gamma = 1$ gesetzt wird und angenommen wird, dass das Maximieren von $\frac{1}{\|w\|}$ das gleiche ist wie das Minimieren von $\|w\|^2$. [[An19], S. 16f.]

$$min_{w,b} \frac{1}{2}||w||^2 \; mit \; y^{(i)}(w^T x + b) \geq 1 \quad (4.2.8)$$

Um der Support Vector Machine außerdem die Möglichkeit der fehlerhaften Klassifikation zu geben, wird dem Problem die Slack Variable ξ hinzugefügt. Dadurch kann das Problem als

$$min_{w,b} \frac{1}{2}||w||^2 + C\sum_{i=1}^{l} \xi_i \; mit \; y^{(i)}(w^T x + b) \geq 1 - \xi_i \quad (4.2.9)$$

beschrieben werden kann. Dabei dient C als Parameter um den Einfluss der Missklassifikation auf den Fehler einzustellen. Wenn dieses Problem mithilfe der Lagrange Mulitplikatoren gelöst wird, ergeben sich die Werte für w. Durch ihre entscheidende Rolle bei der Klassifikation von Datenpunkten handelt es sich bei diesen um die namengebenden Support Vektoren. [[An19], S. 25]

Zur Erkennung von Anomalien werden $v - Support\ Vector\ Machines$ benutzt die das oben genannte Vorgehen leicht abändern. Dieses Vorgehen wurde zuerst in [[Sc99]] dokumentiert, sodass sich die folgenden Erklärungen auf die Ergebnisse dieses Papers beziehen. Die $v - Support\ Vector\ Machines$ trennen während des Trainings nicht zwei Klassen voneinander werden, sondern nur eine Klasse vom Koordinatenursprung. Die Trennung der Daten vom Koordinatenursprung erfolgt mithilfe einer Grenzfunktion, sodass alle Eingabedaten innerhalb der Grenze als normal und jeder Datenpunkt, der außerhalb der Grenzfunktion liegt, als anomal angenommen werden. In der Folgenden Abbildung ist dieser Fall veranschaulicht.

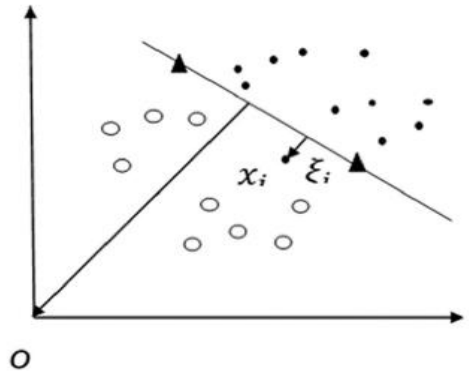

Abbildung 22: Trennung der normalen Daten vom Kooridinatenursprung O [[VSD19], S. 5]

Dadurch verändert sich das Problem zu

$$min_{w,b}\frac{1}{2}||w||^2 + \frac{1}{vl}\sum_{i=1}^{l}\xi_i - \rho\ \ mit\ y^{(i)}(w^T x + b) \geq p - \xi_i \qquad (4.2.10)$$

wobei ρ für den Abstand vom Koordinatenursprung steht und v erneut den Einfluss von ξ reguliert. Zu erkennen ist, dass nun der Abstand vom Koordinatenursprung ρ anstatt des Abstands von den Punkten γ von der Grenzfunktion benutzt wird. Da im Fall der Anomalieerkennung jeder Datenpunkt, der außerhalb der Grenzfunktion liegt, als Anomalie angenommen wird, bestimmt ξ die Anzahl der Anomalien, die in den Trainingsdaten vorkommen.

Nicht jeder Datensatz ist jedoch linear-separierbar, sodass die Support Vector Machines Kernel Funktionen ϕ benutzen, um die Eingabedaten in eine höhere Dimension zu transformieren. Ein Beispiel für eine solche Kernel Funktion ist der gaußscher Kernel.

$$k(x,y) = e^{\frac{-||x-y||^2}{c}} \qquad (4.2.11)$$

In dieser höheren Dimension wird, wie oben beschrieben, eine Hyperebene gefunden, die die Daten voneinander separiert. Dadurch können Daten, die in ihrer ursprünglichen Dimension nicht linear separierbar sind, in einer höheren Dimension linear separiert werden. Da nur normale Daten zum Training der ν-SVM benutzt werden, handelt es sich um eine semi-supervised Anomalieerkennung.

Die ν-SVM sind auch in scikit-learn [[Pe11]] implementiert. Der Folgende Quelltext zeigt die Benutzung der „OneClassSVM“ Klasse zu Erstellung einer ν-SVM.

```
1. svm = OneClassSVM(nu=0.01, kernel="rbf", gamma=0.01)
2. svm.fit(y_train)
3. y_pred_test = svm.predict(y_test)
```

Quelltext 5: Implementierung der ν-SVM mit scikit-learn in Python

In Zeile 1 wird ein neues „OneClassSVM“ Objekt erstellt. Dieses kann mithilfe der Parameter des Konstruktors angepasst werden. Der Parameter „nu“ legt wie oben beschrieben das Verhältnis zwischen den Trainingsfehlern und den Support Vectors fest. Durch den Parameter „kernel“ kann die Kernel Funktion festgelegt werden. Im Quelltext wurde die radiale Basisfunktion als Kernel festgelegt. Der radiale Basisfunktion Kernel hat die Form $K(x,x') = e^{-\gamma\|x-x'\|^2}$, sodass der Parameter γ der Kernel Funktion manuell festgelegt werden kann. Dieser Parameter legt fest, wie stark ein einzelner Punkt in die Erzeugung der Grenzfunktion einfließt. In Zeile 2 wird das „OneClassSVM“ Objekt am Datensatz trainiert, sodass in Zeile 3 ein Testdatensatz ausgewertet werden kann. Das Ergebnis dieser Auswertung ist erneut ein Array mit den Ergebnissen, wobei eine 1 für einen normalen Datenpunkt und -1 für eine Anomalie steht.

4.3 Deep Learning

In den folgenden Kapiteln werden Algorithmen vorgestellt, die in zur Kategorie der Deep Learning Algorithmen gehören. Diese Algorithmen besitzen die künstlichen neuronalen Netze als Grundlage, wobei jeder Algorithmus die in 2.4 erläuterten Grundlagen erweitert. Bei den Deep Learning Algorithmen wird zu Beginn ein Datensatz benötigt, an dem der

Algorithmus trainiert werden kann. Aus diesem Grund gehören die Deep Learning Algorithmen zu den semi-supervised Anomalieerkennungsalgorithmen.

4.3.1 DeepAnT

Die Convolutional Neural Networks erweitern die ursprünglichen künstlichen neuronalen Netze um zwei weitere Arten von Schichten. Bei diesen handelt es sich um die Convolutional-Schicht und die Pooling-Schicht. Diese beiden Schichten sind in der Lage dreidimensionale Eingaben zu verarbeiten. Aus diesem Grund können Convolutional Neural Networks Bilder in ihrer ursprünglichen Form (Höhe, Breite, Tiefe) verarbeiten. [[ON15], S. 4]

Die Convolutional-Schicht bildet das Skalarprodukt zwischen der Eingabe und einem Filter. Die Eingabe besteht aus ein oder mehreren Matrizen, wobei im Fall eines Bildes jeder Farbkanal eine Eingabematrix besitzt. Im Fall eines Bilds in Graustufen ist dementsprechend nur eine Matrix vorhanden. Der Filter besitzt ebenfalls die Form einer Matrix, wobei dessen Höhe und Breite meist kleiner ist als die der Eingabematrizen. Da der Filter eine niedrigere Höhe und Breite als die Eingabematrizen besitzt, wird das Skalarprodukt pro Schritt nur über einen Teil der Eingabematrizen berechnet, jedoch für jede Eingabematrix im selben Bereich. Dieses Vorgehen ist ähnlich der Propagierfunktion einer normalen Neuronenschicht, wobei der Filter den Gewichten entspricht und jede Position in der Eingabematrix einem Neuron. Als Aktivierungsfunktion wird danach meist die Rectified-Linear Funktion $ReLU(x) = \max(0, x)$ benutzt. Dieser Vorgang wird wiederholt, indem die Position des Filters schrittweise verändert wird. Die Veränderung der Position wird über den Stride des Filters bestimmt. Der Stride gibt an, um wie viele Schritte sich die Position des Filters in eine Richtung (x oder y) ändern soll. Falls der Filter an das Ende der Eingabematrix erreicht, wird die jeweils andere Koordinate erhöht. Dieser Vorgang wird wiederholt, bis der Filter das rechte untere Ende der Eingabematrix erreicht hat. [[LKX20]] Dieses Vorgehen ist für zwei Schritte in Abbildung 23 zu sehen. Dabei ist zu beachten, dass in der Abbildung ein Stride von zwei gewählt wurde, sodass sich die x-Position des Filters in jedem Schritt um zwei erhöht. Außerdem wird Skalarprodukt zwischen der Eingabematrix und des Filters noch ein Bias von 1 hinzugefügt.

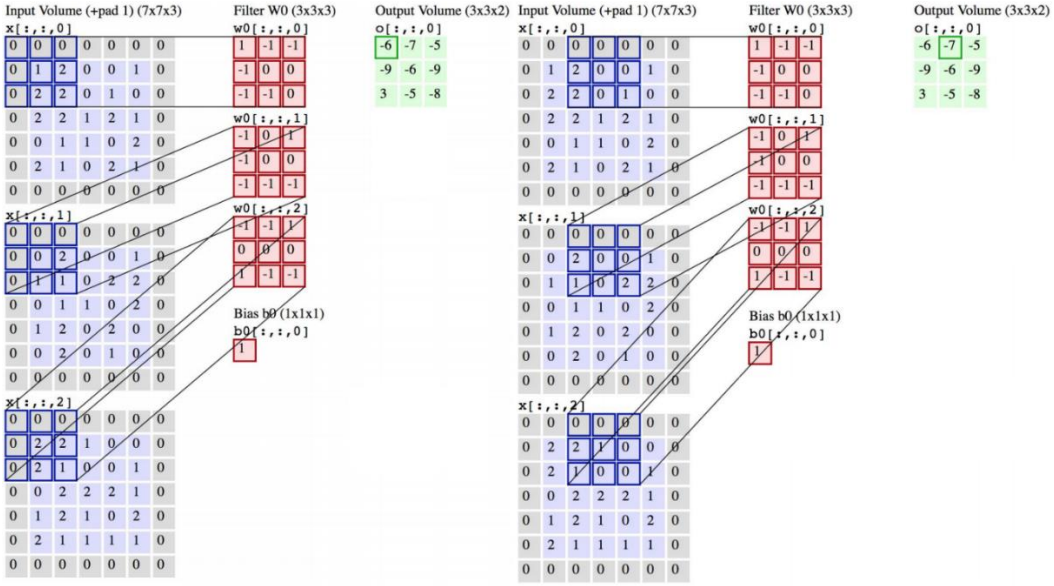

Abbildung 23: Beispiel für die Verarbeitung innerhalb einer Convoultional-Schicht. Bearbeitet entnommen aus [[Ga18], S. 14f.]

Für jeden Filter in einer Convolutional-Schicht wird eine Ausgabematrix berechnet. Da bei einer hohen Anzahl an Filtern pro Convolutional-Schicht die Dimensionalität stark ansteigen kann, werden die Pooling-Schicht benutzt, um die Dimensionalität der Daten zu verringern. Das Vorgehen einer Pooling-Schicht ist ähnlich der Convolutional-Schicht. Dabei wird jedoch nicht das Skalarprodukt zwischen den Eingabematrizen und den Filtern berechnet, sondern eine Funktion auf die Elemente in den Eingabematrizen angewandt. Die zwei am häufigsten benutzten Funktionen sind die Max-Funktion und die Durchschnitts-Funktion, wodurch die Max-Pooling-Schicht und die Average-Pooling-Schicht entsteht. Die Pooling-Schicht besitzt ebenfalls eine Höhe und Breite, sodass für jeden Schritt ein Teil der Elemente in den Eingabematrizen zusammengefasst werden. Die Anzahl der Eingabematrizen ändert sich jedoch bei der Benutzung einer Pooling-Schicht nicht, da diese auf jede Eingabematrix einzeln angewandt wird. [[ON15], S. 8] Ein Beispiel für die zwei genannten Pooling-Schichten mit einem Stride von zwei ist Abbildung 24 zu sehen.

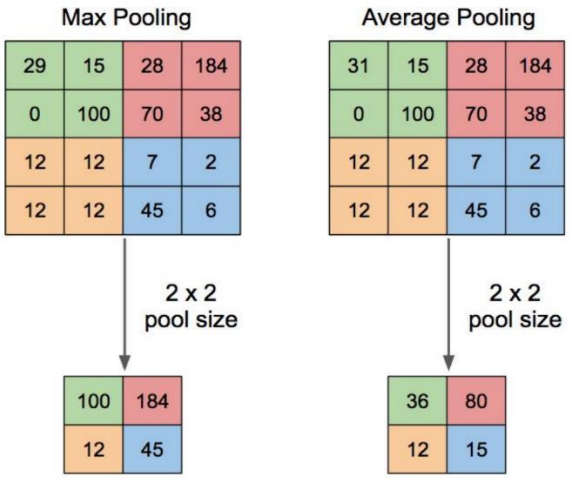

Abbildung 24: Beispiel für eine Max-Pooling-Schicht und eine Average-Pooling-Schicht [[Ga18], S. 19]

Um am Ende des Convolutional Neural Networks eine normale Neuronenschicht benutzen zu können, müssen die Ausgabematrizen der vorherigen Schicht zu einem Vektor umgewandelt werden. Dies geschieht, indem die Matrizen vektorisiert und an einen finalen Vektor angehangen werden.

DeepAnT [[Mu19]] ist ein Deep Learning Anomalieerkennungsalgorithmus durch den Punktanomalien, kontextuale Anomalien und Discords zu erkennen. Um eine Anomalie zu erkennen benutzt DeepAnT ein Zeitreihen-vorhersage-Modul und ein Anomlieerkennungs-Modul. Das Zeitreihen-vorhersage-Modul besteht aus einem Convolutional Neural Network um den nächsten bzw. die nächsten Zeitschritte vorherzusagen. Nach der Vorhersage sorgt das Anomalieerkennungs-Modul dafür, dass Anomalien im Datensatz markiert werden. Für die Erklärung des Algorithmus wird im Folgenden [[Mu19]] benutzt.

Eine Zeitreihe kann wird mithilfe einer eindimensionalen Convolutional-Schicht ausgewertet werden. Die Zeitreihe kann dabei als ein Bild mit einer Höhe von eins angesehen werden, wobei die Breite des Bildes für die Zeit steht. Demensprechend bewegt sich der Filter der eindimensionalen Convolutional-Schicht über mehrere Zeitschritte gleichzeitig, wie in der Folgenden Abbildung zu sehen ist.

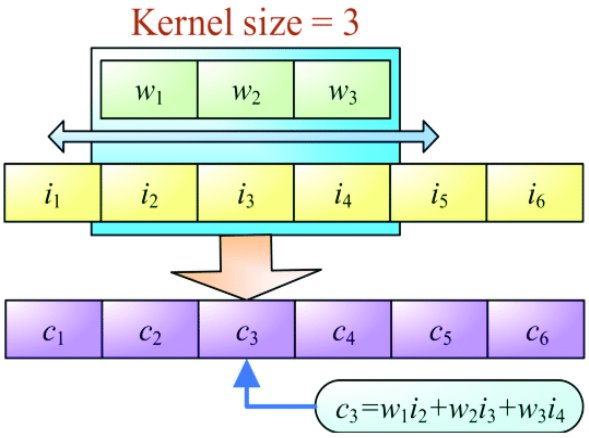

Abbildung 25: Berechnung einer eindimensionalen Convolution mit einer Filtergröße von 3 [[KH18], S. 824]

Bei der Benutzung von DeepAnT wird zuerst die Anzahl der zu benutzenden historischen Werte w festgelegt. Diese gibt an, wie viele Werte aus der unmittelbaren Vergangenheit benutzt werden sollen um den Wert $\hat{y}$ vorherzusagen. Dieses Zeitfenster wird mithilfe eines Sliding Windows aus der Zeitreihe extrahiert. Als zweites wird die Vorhersagelänge p_w festgelegt, die angibt, wie viele Zeitschritte in der Zukunft bestimmt werden sollen. Soll eine Punktanomalie erkannt werden, wird p_w auf 1 gesetzt, sodass ein Wert $\hat{y}$ für $t+1$ bestimmt wird. [[Mu19], S. 1995] Die Eingabedaten x_{in} werden aus überlappenden Subsequenzen der Zeitreihe mit Größe w zusammengestellt, wobei $\hat{y}$ den nächsten p_w Werten entspricht. Dementsprechend ergibt sich für eine Zeitreihe $\{x^{(1)}, x^{(2)}, \dots, x^{(t-1)}, x^{(t)}, x^{(t+1)}, \dots\}$ eine Vorhersage $\hat{y}$ welche in Gleichung (4.3.1) beschrieben wird.

$$x_{in} = \{x^{(t-w)}, x^{(t-w+1)}, \dots, x^{(t-1)}, x^{(t)}\} \rightarrow \hat{y} = \{x^{(t+1)}, x^{(t+2)}, \dots, x^{(t+p_w)}\} \quad (4.3.1)$$

Das in DeepAnT benutzte Convolutional Neural Network besteht aus zwei Convolutional-Schichten mit 32 Filtern und einer Rectified-Linear Aktiverungsfunktion, denen jeweils eine Max-Pooling-Schicht folgt. Danach folgt eine normale Neuronenschicht, bei der alle Ausgaben der vorherigen Schicht mit allen Neuronen in der derzeitigen Schicht verbunden sind. Die Ausgabeschicht besteht aus p_w Neuronen, sodass jedes Neuron für einen vorhergesagten Zeitschritt steht. Um während des Trainings die Abweichung zwischen y und $\hat{y}$ zu berechnen wird der Mean Absolute Error als Fehlerfunktion benutzt.

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j| \quad (4.3.2)$$

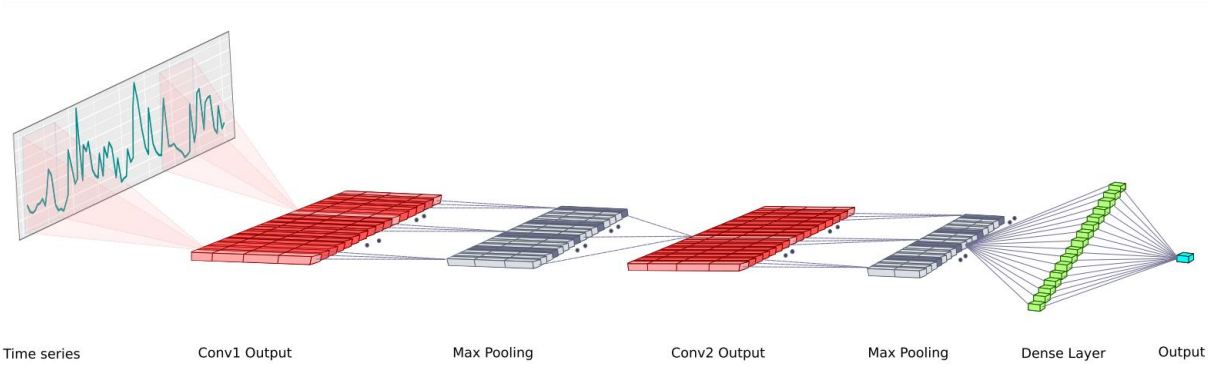

Abbildung 26: Darstellung der DeepAnT Architektur [[Mu19], S. 1996]

Das Anomalieerkennungs-Modul berechnet einen Anomalie Score aufgrund des euklidsichen Abstands zwischen y und $\hat{y}$.

$$Euklidischer\ Abstand(y, \hat{y}) = \sqrt{\sum_{j=1}^{n} \left(y_j - \hat{y}_j\right)^2} \qquad (4.3.3)$$

Sollte eine Vorhersagelänge $p_w = 1$ gewählt werden und der Datensatz univariat sein, reduziert sich die Berechnung des Anomalie Scores auf die Bildung der Differenz zwischen y und $\hat{y}$. Um aufgrund des Anomalie Scores festzustellen, ob ein Punkt als Anomalie gekennzeichnet werden muss, werden zwei Arten der Bestimmung eines Schwellwertes vorgeschlagen. Diese Ansätze stammen vom Paper [[LAF15]] indem der Anomalieerkennungsalgorithmus von Yahoo beschrieben wird. Der erste Ansatz benutzt die k-fache Standardabweichung vom Mittelwert, um eine Schwelle zu definieren. Dabei wird davon ausgegangen, dass die Daten des Anomalie Scores normalverteilt sind. Durch die Wahl von k kann die Empfindlichkeit der Erkennung angepasst werden, wobei für $k = 3$ 99.73% der Anomalie Scores unterhalb der Schwelle liegen. Der zweite Ansatz benutzt den Local Outlier Factor Algorithmus, um aufgrund der Anomalie Scores die Anomalien zu erkennen. Der Local Outlier Factor benutzt die k-nearest neighbor Distanz, um die lokale Dichte des Punktes zu bestimmen. Diese lokale Dichte wird mit der lokalen Dichte der Nachbarpunkte verglichen, sodass Punkte mit geringerer lokaler Dichte als Anomalien gekennzeichnet werden. [[LAF15], S. 1943]

Wenn DeepAnT für die Erkennnung von Discords benutzt wird, wird der Anomalie Score für jeden Punkt in der Subsequenz bestimmt und die Scores anschließend summiert. Dieser summierte Anomalie Score steht für die gesamte Subsequenz und kann im Anschluss wie

oben beschrieben aufgrund eines Schwellwertes als normal oder anomal klassifiziert werden. Im Quelltext 6 wird gezeigt, wie mit der Keras Bibliothek ein Convolutional Neural Network erzeugt werden kann. Dafür werden einem „Sequential“ Objekt mehrere Schichten hinzugefügt, sodass das Netz den gleichen Aufbau wie in Abbildung 26 erhält.

```
1.  model = tf.keras.models.Sequential()
2.  model.add(Conv1D(32, kernel_size=2, input_shape=(n_timesteps,
    n_features), activation='relu'))
3.  model.add(Dropout(0.3))
4.  model.add(MaxPooling1D(pool_size=2))
5.  model.add(Conv1D(32, kernel_size=2))
6.  model.add(Dropout(0.3))
7.  model.add(MaxPooling1D(pool_size=2))
8.  model.add(Flatten())
9.  model.add(Dense(16))
10. model.add(Dense(n_outputs))
11. model.compile(loss='mean_squared_error', optimizer='adam')
```

Quelltext 6: Erstellung des Convoultional Neural Networks von DeepAnT mit Keras in Python

4.3.2 Telemanom

Traditionelle Multilayer Perzeptron werten jedes Datum unabhängig vom vorherigen aus. In vielen Problemstellungen basiert jedoch die derzeitige Entscheidung auf bereits vorher bekannten Informationen. Die Convolutional Neural Networks benutzen Filter, um unmittelbar benachbarte Punkte in die Auswertung einzubeziehen. Eine andere Möglichkeit ist das Hinzufügen von Rückkopplungen, sodass vergangene Werte in die derzeitige Berechnung einfließen können. Die Recurrent Neural Networks (RNN) besitzen diese Rückkopplung, indem die Neuronen eine Verbindung zu sich selbst besitzen. Dadurch kann das Ergebnis des Zeitschritts $t - 1$ für die Berechnung im Zeitschritt t benutzt werden.

Der Trainingsalgorithmus für rekurrente Netze nennt sich Back-Propagation Through Time und verteilt den Fehler von Zeitschritten aus der Zukunft auf die Gewichte des Netzes. Durch dieses Vorgehen wird das Netz „ausgerollt“, sodass für jeden Zeitschritt eine Schicht entsteht. Innerhalb des Backpropagtion Algorithmus wird die Kettenregel benutzt, wodurch sich der Fehler der derzeitigen Schicht aufgrund eines Produktes berechnet. Im Fall eines RNN werden dabei nicht nur die einzelnen Schichten durch ein Produkt miteinander verbunden, sondern auch die Neuronen der zukünftigen Zeitschritte. Das bedeutet, dass mit jedem Zeitschritt das Produkt weiterwächst. Dieses Produkt wird umso größer, je näher es an der Eingabe des Netzes liegt, wobei alle zukünftigen Zeitschritte miteinbezogen werden

müssen. Dies kann dazu führen, dass der Fehler „explodiert“ oder „verschwindet“, falls die Gewichte der jeweiligen Schicht sehr klein oder sehr groß sein sollten. Dieses Phänomen nennt sich „Vanishing Gradient“. Dabei nimmt die Chance eines „Vanishing Gradients“ mit zunehmender Schichtanzahl zu, wodurch auch die Chance steigt, falls eine hohe Anzahl an Zeitschritten in einem rekurrenten Netz benutzt wird. Dementsprechend führt der „Vanishing Gradient“ dazu, dass Langzeitabhängigkeiten nur schwer trainiert werden können. [[Jo91], S. 21] Um dieses Problem lösen wurden die Long Short-Term Memory Netze entwickelt.

Long Short Term Memory Networks (LSTM Networks) sind dafür gemacht Langzeitabhängigkeiten zu lernen. Traditionelle RNN besitzen kettenähnliche Verbindungen, welche sich wiederholen. Diese könnten zum Beispiel einer einfachen Aktivierungsfunktion in Form einer Tanh-Funktion entsprechen. Anstatt einer einfachen Tanh-Funktion besitzen LSTM Networks ein komplexeres sich wiederholendes Modul mit dem Namen „Memory Cell“. Dieses sieht folgendermaßen aus.

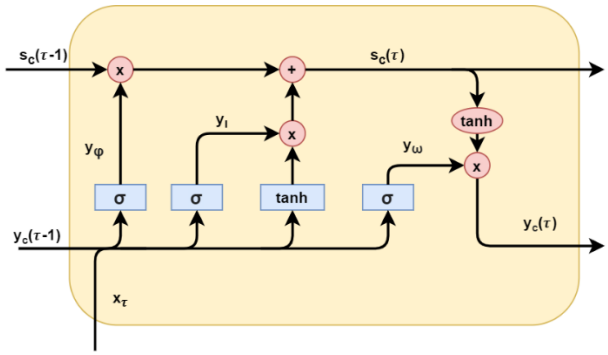

Abbildung 27: Darstellung der Memory Cell eines LSTM

Die Memory Cell besitzt sogenannte Gates, die aus einer Sigmoid Funktion und einem punktweisen Produkt bestehen. Das LSTM Netzwerk besitzt zum Speichern von Langzeitinformationen den Cellstate $S_c(\tau)$. Dabei dienen die Gates dazu, Informationen aus dem Cellstate zu löschen oder hinzuzufügen. [[Jo91], S. 6f.]

Zur Erklärung der Funktionsweise wird das originale Paper [[HS97]] mit der Notation von [[GS05]] benutzt, da diese übersichtlicher ist. Zu beachten ist, dass die Erweiterung aus [[GS05]], wodurch $S_c(\tau)$ mit in die Gates einfließt, nicht übernommen werden.

Zu Beginn wird festgestellt, welche Informationen im Cellstate "vergessen" werden sollen. Das dazugehörige Gate nennt sich "Forget Gate" y_φ.

$$y_\varphi = Sigmoid\left(\sum_{j \in N} w_{\varphi j} y_j(\tau - 1)\right) \quad (4.3.4)$$

Dabei ist $y_j(\tau - 1)$ die Ausgabe des vorherigen Zeitschritts $y_c(\tau - 1)$ konkateniert mit der Eingabe des jetzigen Schrittes $x_j(t - 1)$. Das „Input Gate“ y_ι sorgt dafür, dass neue Informationen in Cellstate gespeichert werden. Dieses besteht aus der Anwendung der Sigmoid-Funktion auf $y_j(\tau - 1)$.

$$y_\iota = Sigmoid\left(\sum_{j \in N} w_{\iota j} y_j(\tau - 1)\right) \quad (4.3.5)$$

Durch das „Input Gate“ kann im Anschluss der Cell State für den derzeitigen Zeitschritt τ aktualisiert werden.

$$x_c = \sum_{j \in N} w_{cj} y_j(\tau - 1) \quad (4.3.6)$$

$$s_c(\tau) = y_\varphi s_c(\tau - 1) + y_\iota \tanh(x_c) \quad (4.3.7)$$

Mit dem neuen Cell State $s_c(\tau)$ kann die Ausgabe y_c der Memory Cell berechnet werden. Dies ist in Gleichung (4.3.8) und (4.3.9) zu sehen.

$$y_w = Sigmoid\left(\sum\nolimits_{j \in N} w_{cj} y_j(\tau - 1)\right) \quad (4.3.8)$$

$$y_c = y_w * \tanh\big(s_c(\tau)\big) \quad (4.3.9)$$

Für die Verarbeitung in der nächsten Cell wird die Ausgabe y_c und der Cellstate $s_c(\tau)$ weitergegeben, sodass diese in der nächsten Cell als $y_c(\tau - 1)$ und $s_c(\tau - 1)$ benutzt werden können.

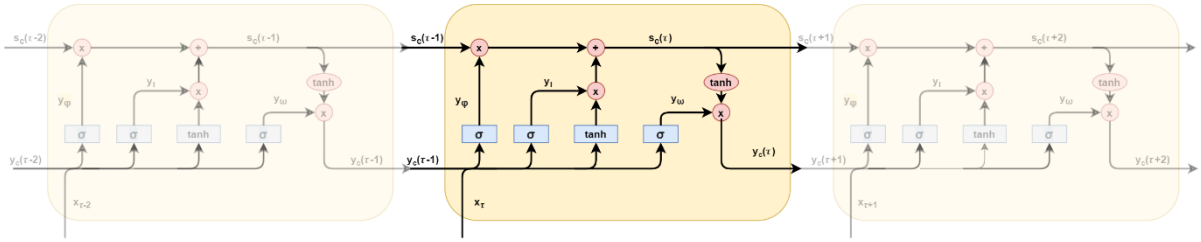

Abbildung 28: Mehrere Memory Cells aneinandergereiht

Der im Paper [[Hu18a]] vorgestellte Algorithmus Telemanom besteht aus der Kombination eines LSTM zur Vorhersage zukünftiger Werte und der Erzeugung einer Fehlergrenze aufgrund von historischen Daten. Der Algorithmus lässt sich in vier Schritte einteilen. Bei diesen handelt es sich um das Training des LSTM, die Bestimmung einer Fehlergrenze ε durch das Lösen eines Optimierungsproblems, der Berechnung des Anomalie Scores s für Bereiche deren Fehler über ε liegen und der Aussortierung von Anomalien, die aufgrund von Rauschen entstanden sind. Als Ergebnis liefert der Algorithmus Bereiche die aufgrund ihrer Abweichung von ε als anormal klassifiziert wurden. Im Allgemeinen wird sich im Folgenden die Vorgehensweise aus [[Hu18a]] erläutert. Da jedoch auch in [[Hu18a]] nicht alle Informationen für eine Umsetzung enthalten sind, wird zum Teil auf die offizielle Implementierung [[Hu18b]] verwiesen.

Das Training des LSTM erfolgt aufgrund von Daten, die keine Anomalien enthalten, um das normale Verhalten anzutrainieren. Da das Netz nur das normale Verhalten kennt, entsteht in einem Bereich mit Anomalien ein größerer Fehler als in einem normalen Bereich. Beim Training des LSTM muss die Sequenzlänge der Eingabe l_s festgelegt werden. Diese gibt an, wie viele Werte aus der unmittelbaren Vergangenheit benutzt werden sollen um den Wert $\hat{y}$ vorherzusagen. Diese Daten werden mithilfe eines Sliding Windows der Größe l_s erzeugt. Außerdem muss die Vorhersagelänge l_p festgelegt werden, da ein LSTM mehrere Zeitschritte gleichzeitig vorhersagen kann. Im Originalen Paper wird l_p auf 1 gesetzt, sodass nur der Zeitschritt $t+1$ vorhergesagt wird. Aufgrund von l_s und l_p ergibt sich die Eingabe x_{in} zur Berechnung von $\hat{y}^{(t+1)}$ als $x_{in} = \{x^{(t-l_s)}, x^{(t-l_s+1)}, \dots, x^{(t-1)}, x^{(t)}\}$. Die Werte von x_{in} können dabei Vektoren mit m-Dimensionen sein in der Form $x^{(t)} = \{x_1^{(t)}, x_2^{(t)}, \dots, x_m^{(t)}\}$. Außerdem muss die Dimension d der Ausgabe festgelegt werden, da $\hat{y}^{(t+1)}$ aus d-Dimensionen bestehen kann. Im originalen Paper wird dabei $d = 1$ gesetzt, sodass $\hat{y}^{(t+1)}$

ein skalarer Wert ist. Es kann jedoch auch $d > 1$ gewählt werden, sodass Werte für mehrere parallele Zeitreihen gleichzeitig bestimmt werden.

Nach dem Training des LSTM wird die Fehlergrenze ε ermittelt. Dafür wird zunächst festgelegt wie viele Werte aus der Vergangenheit für die Berechnung von ε benutzt werden sollen. Diese Anzahl wird durch h festgelegt und beschreibt den Bereich, der für die Bestimmung von ε benutzt werden soll. Für jeden Wert im Zeitfenster $t - h$ wird eine Voraussage $\hat{y}^{(t)}$ durch das LSTM berechnet und der berechnete Wert dem beobachteten Wert $y^{(t)}$ verglichen. Danach kann für den jeweiligen Zeitschritt t der Fehler berechnet werden wie in Gleichung (2.1.2). Dabei wird davon ausgegangen wird, dass $\hat{y}^{(t)}$ und $y^{(t)}$ Skalare sind. Sollte $d > 1$ gewählt wurden sein, kann der Fehler durch den Mittelwert der einzelnen Fehler berechnet werden, sodass er durch

$$e^{(t)} = \sum_{i=1}^{d} \frac{\left| y_i^{(t)} - \hat{y}_i^{(t)} \right|}{d} \qquad (4.3.10)$$

berechnet wird. Dieser Vorgang wird für jeden Zeitschritt t in h wiederholt und der resultierende Fehler e^t an einen eindimensionalen Fehlervektor e angehangen.

$$e = \left[e^{(t-h)}, \dots, e^{(t-l_s)}, \dots, e^{(t-1)}, e^{(t)} \right] \qquad (4.3.11)$$

Danach wird empfohlen den Fehlervektor e zu glätten, da das LSTM starke Veränderungen in y nicht gut abbilden kann und dadurch Fehlerspitzen entstehen. Für die Glättung wird der exponentielle geglättete Durchschnitt (exponential weighted moving average) gebildet, sodass sich für e ein geglätteter Vektor e_s ergibt mit $e_s = \left[e_s^{(t-h)}, \dots, e_s^{(t-l_s)}, \dots, e_s^{(t-1)}, e_s^{(t)} \right]$.

Mithilfe der ermittelten Fehler pro Datenpunkt kann die Fehlergrenze ε ermittelt werden. Die Fehlergrenze besitzt die Form

$$\varepsilon = \mu(e_s) + z\sigma(e_s) \qquad (4.3.12)$$

wobei z durch das Lösen eines Optimierungsproblems gefunden wird. Es entsteht demnach eine Fehlergrenze, die der z-fachen Standardabweichung über dem Durchschnitt des Fehlers entspricht. Dabei wird ein Wert für z gefunden, der ε maximiert. Das Optimierungsproblem wird formuliert als

$$argmax(\varepsilon) = \frac{\frac{\Delta\mu(e_s)}{\mu(e_s)} + \frac{\Delta\sigma(e_s)}{\sigma(e_s)}}{|e_a| + |E_{seq}|^2} \quad (4.3.13)$$

mit

$$\Delta\mu(e_s) = \mu(e_s) - (\{pe_s \in e_s \mid pe_s < \varepsilon\}) \quad (4.3.14)$$

$$\Delta\sigma(e_s) = \sigma(e_s) - \sigma(\{pe_s \in e_s \mid pe_s < \varepsilon\}) \quad (4.3.15)$$

$$e_a = \{pe_s \in e_s | pe_s > \varepsilon\} \quad (4.3.16)$$

$$E_{seq} = Anomlien\ e_a\ die\ zu\ Sequenzen\ erweitert\ wurden \quad (4.3.17)$$

Es wird demnach versucht ε zu maximieren, bis eine Grenze gefunden wird, durch die bei der Entfernung aller Werte über der Schwelle die stärkste Veränderung in μ und σ eintritt. Dabei werden sowohl einzeln auftretende Anomalien in Form von e_a sowie aneinander angrenzende Anomalien durch E_{seq} in das Problem mit einbezogen. [[Hu18a], S. 4] Im Paper wird auf die Erzeugung von E_{seq} nicht näher eingegangen, weshalb zur Erklärung das offizielle Github Repository [[Hu18b]] benutzt wird. Die Fehlersequenzen E_{seq} werden demnach erzeugt, indem den Anomalien in e_a ein Bereich von $t \pm b$ Zeitpunkten hinzugefügt wird. Der Parameter b kann dabei frei gewählt werden und beeinflusst über welchen Abstand Anomalien miteinander verbunden werden. [[Hu18b], errors.py:221-229] Das Optimierungsproblem wird gelöst, indem Werte für z iterativ festgelegt werden und der z Wert gespeichert wird, der $argmax(\varepsilon)$ maximiert. Im originalen Paper nimmt z Werte zwischen 2 und 10 an, wobei Schritte von 0.5 festgelegt werden. Im nachfolgenden Quelltext ist das Finden einer optimalen Fehlergrenze abgebildet. Dafür werden in Zeile 6 Werte für z von 2.5 bis 30 in 0.5er Schritten ausprobiert. In Zeile 15-19 werden aufgrund der Fehlergrenze und b die anomalen Fehlersequenzen E_{seq} erstellt. Durch die Fehlersequenzen E_{seq} kann von Zeile 20-27 $argmax(\varepsilon)$ bestimmt werden sodass in 28-31 er maximale $argmax(\varepsilon)$ Wert und z Wert gespeichert werden können.

```
1.  def calc_max_epsilon(errors: list, anomaly_sequence_size: int):
2.    max_eps = 0
3.    max_s = 0
4.    max_threshold = 0
5.    optimal_z = 2.5
6.    for z in np.arange(2.5, 30, 0.5):
7.      epsilon_threshold = np.mean(errors) + (z * np.std(errors))
8.
9.      pruned_es = []
10.     anomaly_indices = []
11.     for i, es in enumerate(errors):
12.         if es < epsilon_threshold:
13.             pruned_es.append(es)
14.         else:
15.             for j in range(anomaly_sequence_size):
16.                 if i + j not in anomaly_indices and i + j < len(errors):
17.                     anomaly_indices.append(i + j)
18.                 if i - j not in anomaly_indices and not i - j < 0:
19.                     anomaly_indices.append(i - j)
20.     if len(anomaly_indices) > 0:
21.         anomaly_indices = sorted(anomaly_indices)
22.         groups = [list(group) for group in
                 mit.consecutive_groups(anomaly_indices)]
23.         e_seq = [(g[0], g[-1]) for g in groups if not g[0] == g[-1]]
24.
25.         delta_mean = np.mean(errors) - np.mean(pruned_es)
26.         delta_std = np.std(errors) - np.std(pruned_es)
27.         epsilon = (delta_mean + delta_std) / (len(e_seq) ** 2 +
            len(anomaly_indices))
28.         if epsilon >= max_eps and len(e_seq) <= 5 and
            len(anomaly_indices) <(len(errors) * 0.5):
29.             optimal_z = z
30.             max_s = epsilon
31.             max_threshold = epsilon_threshold
```

Quelltext 7: Bestimmung des optimalen Epsilons in Python

Der im zweiten Schritt berechnete Wert z wird im dritten Schritt für die Bestimmung der Anomalie Scores benutzt. Der Anomalie Score ergibt sich dabei aus

$$s^i = \frac{\max\left(e_{seq}^{(i)}\right) - argmax(\varepsilon)}{\mu(e_s) + \sigma(e_s)} \qquad (4.3.18)$$

sodass die Differenz zwischen ε und dem maximalen Fehler innerhalb einer Fehlersequenz ermittelt und durch den Mittelwert und der Standardabweichung normalisiert wird.

Im vierten Schritt können gefundene Anomalien entfernt werden, wodurch die Anzahl der False Positives reduziert werden soll. Dafür werden zwei verschiedene Methoden vorgeschlagen. Die erste Vorgehensweise benutzt dabei bereits annotierte Anomalien, um eine Mindestgrenze s_{min} festzulegen. Dadurch kann der Anomalie Score für bekannte

Anomalien bestimmt werden und s_{min} als der niedrigste Score dieser Anomalien festgelegt, sodass Anomalien mit niedrigeren Scores als normal klassifiziert werden. Die zweite Vorgehensweise benutzt den prozentualen Unterschied zwischen den absteigend sortierten Sequenzen in max (e_{seq}). Dazu wird zunächst e_{max} bestimmt, indem das Maximum des Fehlers in den Sequenzen e_{seq} bestimmt wird. Im Anschluss wird e_{max} absteigend sortiert und der größte, der nicht anormal ist, an e_{max} angehangen. Für jeden Wert in e_{max} wird der prozentuale Unterschied d zwischen den Elementen in e_{max} bestimmt durch

$$d^{(i)} = \frac{\left(e_{max}^{(i-1)} - e_{max}^{(i)}\right)}{e_{max}^{(i-1)}} \; f\ddot{u}r \; i \in \left\{1, 2, \dots, \left(\left|E_{seq}\right| + 1\right)\right\} \qquad (4.3.19)$$

Des Weiteren wird ein Schwellwert p festgelegt, der zum Vergleich der prozentualen Differenz dient. Sollte für einen Wert $d^{(i)} < p$ zutreffen, wird die jeweilige Anomalie für $d^{(i)}$ und alle darauffolgenden Anomalien $d^{(i)}, d^{(i+1)}, \dots, d^{(i+|E_{seq}|+1)}$ als normal klassifiziert.

4.3.3 VAE

Ein Autoencoder ist ein künstliches neuronales Netz, welches mithilfe von unsupervised Learning trainiert werden kann. Die Aufgabe eines Autoencoders ist die Rekonstruktion der originalen Eingabe, nachdem diese in eine niedrigere Dimension transformiert wurde. Diese Reduktion wird erzeugt, indem die Anzahl der Neuronen in der verborgenen Schicht kleiner ist als die Anzahl der Neuronen in der Ein- bzw. Ausgabeschicht. Der Autonencoder kann deshalb in zwei Teile eingeteilt werden. Der erste Teil wird Encoder genannt und besteht aus den Schichten zwischen der Eingabeschicht und der verborgenen Schicht mit der geringsten Anzahl an Neuronen. Der Encoder sorgt für die Reduktion der Eingabe und erzeugt dadurch eine Codierung ähnlich einer Reduktion mithilfe der PCA [[WYZ15], S. 2]. Der zweite Teil des Autoencoders ist der Decoder und besteht aus den Schichten zwischen der verborgenen Schicht und der Ausgabeschicht. Der Decoder benutzt die vom Encoder reduzierte Repräsentation der originalen Eingabe, um diese Eingabe wiederherzustellen.

Autoencoder können zur Erkennung von Anomalien benutzt werden, indem der Autoencoder trainiert wird normale Daten zu rekonstruieren. Während des Trainings wird der Rekonstruktionsfehler zwischen der Eingabe und der rekonstruierten Eingabe berechnet

und die Gewichte aufgrund dieses Fehlers reduziert. Da der Autoencoder anhand von normalen Daten trainiert wurde, werden normale Daten mit nur einem geringen Rekonstruktionsfehler wiederhergestellt, während anomale Daten einen größeren Rekonstruktionsfehler aufweisen. Mithilfe einer vorher definierten Schwelle α wird jede Eingabe als anomal klassifiziert, sollte dessen Rekonstruktionsfehler über der Schwelle α liegen. [[AC15], S. 4f.]

Traditionelle Autoencoder benutzen nur den Rekonstruktionsfehler, um die Gewichte des Netzes anzupassen, wodurch die aus dem Encoder entstehende Codierung ohne Bedingungen erzeugt wird. Ein variationalen Autoencoder benutzt den Encoder $f(x)$, um eine Reduktion der Eingabe zu erzeugen und im Anschluss eine Normalverteilung $q_\varphi(z|x)$ mit μ_φ und σ_φ zu erzeugen. Mithilfe von $q_\varphi(z|x)$ kann danach eine Stichprobe z erzeugt werden, die im Decoder $g(z)$ benutzt wird um x zu rekonstruieren und somit $\hat{x}$ zu erhalten. Die Stichprobe z entspricht der Codierung im variationalen Autoencoder. Der Decoder erzeugt dafür eine weitere Normalverteilung $p_\theta(x|z)$ mit μ_θ und σ_θ sodass daraus $\hat{x}$ erzeugt werden kann. [[Gu18], S. 100] Der eben beschriebene Aufbau ist in Abbildung 29 zu sehen.

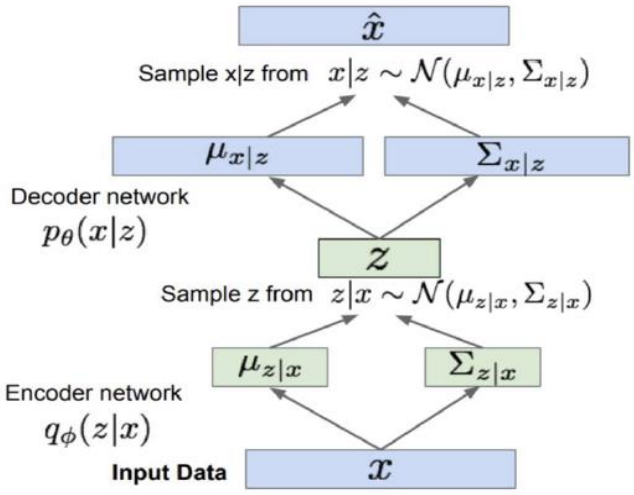

Abbildung 29: Aufbau eines variationalen Autoencoders [[La18], S. 2]

Die Fehlerfunktion des VAE ist die variationale untere Grenze der marginalen Likelihood der gesamten Daten (variational lowerbound of the marginal likelihood of data), wobei sie sich aus der Summe der variationale untere Grenze der marginalen Likelihood der Datenpunkt ergibt $\log p_\theta(x^1, \dots, x^n) = \sum_{i=1}^{N} \log p_\theta(x^i)$. Diese Likelihood $\log p_\theta(x^i)$

besteht aus der Kullback-Leibler Divergenz zwischen $q_\varphi(z|x)$ und einer A-priori-Verteilung $p_\alpha(z)$ mit $\mu_\alpha = 0$ und $\sigma_\alpha = 1$ sowie des Rekonstruktionsfehlers zwischen x und $\hat{x}$ in Form von $E_{q_\varphi(z|x)}[\log \mathrm{p}_\theta(x|z)]$. [[DM14], S. 13]

$$\log p_\theta(x^i) = -D_{KL}\left(q_\varphi(z|x^{(i)})||p_\alpha(z)\right) + E_{q_\varphi(z|x)}[\log \mathrm{p}_\theta(x|z)] \qquad (4.3.20)$$

Wie in 2.4 beschrieben wird der Backpropagation Algorithmus benutzt, um künstliche neuronale Netze zu trainieren. Da $E_{q_\varphi(z|x)}[\log \mathrm{p}_\theta(x|z)]$ mithilfe von Monte Carlo Methoden berechnet wird, müssen während des Backpropagation Monte Carlo Gradient Methoden benutzt werden. Um die Entstehung einer hohen Varianz bei der der Benutzung von Monte Carlo Gradient Methoden zu umgehen, wird der sogenannte „Reparametrisierungstrick" benutzt. Dieser Trick fügt der Variable z$\sim q_\varphi(z|x)$ eine Zufallsvariable $\varepsilon \sim N(0,1)$ hinzu. Dadurch wird z durch eine Transformation $h_\varphi(z|x)$ reparametrisiert, sodass z nun die Form $z = \mu_\theta + \sigma_\theta \cdot \varepsilon$ besitzt. Durch das Hinzufügen von ε kann während des Backpropagation eine Stichprobe von ε genommen werden, anstatt von der tatsächlichen Verteilung. [[AC15], S. 7]

Im Folgenden Quelltext ist die Erstellung eines VAE mit der Keras Functional API abgebildet. Mithilfe der Functional API können Modelle mit parallelen Schichten oder mehreren Ausgaben pro Ebene erstellt werden. Von Zeile 1-13 wird der Encoder des VAE aufgebaut. Dabei werden in Zeile 6 und 7 zwei parallele Schichten angelegt, die für die Erzeugung von μ_φ und σ_φ zuständig sind. Der „KLDivergenceLayer" ist eine benutzerdefinierte Schicht, die Eingaben nicht verändert, jedoch die Kullback-Leibler Divergenz der Fehlerfunktion hinzufügt. In Zeile 15-24 wird der Decoder erstellt der in Zeile 18 und 19 die zweite Normalverteilung μ_θ und σ_θ erzeugt. In 26-31 wird das Gesamtmodell erzeugt, indem zuerst der Encoder und Decoder erstellt wird und im Anschluss der Encoder in den Decoder eingesetzt wird. Die Funktion „log_loss" fügt dem Netz den Rekonstruktionsfehler $E_{q_\varphi(z|x)}[\log \mathrm{p}_\theta(x|z)]$ hinzu.

```
1.  x = Input(shape=(original_dim,))
2.  eps = Input(tensor=K.random_normal(stddev=1.0, shape=(K.shape(x)[0],
    latent_dim)))
3.
4.  h = Dense(intermediate_dim, activation='relu')(x)
5.
6.  z_mu = Dense(latent_dim)(h)
7.  z_log_var = Dense(latent_dim)(h)
8.
9.  z_mu, z_log_var = KLDivergenceLayer()([z_mu, z_log_var])
10. z_sigma = Lambda(lambda t: K.exp(.5 * t))(z_log_var)
11.
12. z_eps = Multiply()([z_sigma, eps])
13. z_internal = Add()([z_mu, z_eps])
14.
15. z = Input(shape=(latent_dim,))
16. z_expanded = Dense(intermediate_dim, activation='relu')(z)
17.
18. reconstr_mu = Dense(original_dim, activation='tanh')(z_expanded)
19. reconstr_log_var = Dense(original_dim)(z_expanded)
20.
21. reconstr_sigma = Lambda(lambda t: K.exp(.5 * t))(reconstr_log_var)
22. x_hat_reconstructed = Add()([reconstr_mu, reconstr_sigma])
23.
24. x_hat_reconstructed_bound = Dense(original_dim,
    activation='sigmoid')(x_hat_reconstructed)
25.
26. vae_encoder = Model(inputs=[x, eps], outputs=z_internal)
27. vae_decoder = Model(inputs=[z], outputs=[x_hat_reconstructed_bound,
    reconstr_mu, reconstr_sigma])
28.
29. outputs = vae_decoder(vae_encoder([x, eps]))[0]
30. vae = Model(inputs=[x, eps], outputs=outputs)
31. vae.compile(optimizer='adam', loss=log_loss)
```

Quelltext 8: Erstellung eines VAE mit der Keras Functional API in Python

Um mithilfe eines VAE Anomalien zu erkennen wird die *reconstruction probability* berechnet. Dieses Vorgehen wurde erstmals in [[AC15]] dokumentiert. Die *reconstruction probability* entsteht, indem x in den Encoder eingesetzt wird um μ_φ und σ_φ zu erzeugen. Anschließend werden mit μ_φ und σ_φ und der Zufallsvariable ε die Codierung z erzeugt. Dieser Vorgang wird aufgrund der Zufallsvariable ε l mal wiederholt, sodass l Codierungen entstehen. Für jedes z wird im Decoder eine Normalverteilung $p_\theta(x|z)$ mit $\mu_\theta(l)$ und $\sigma_\theta(l)$ erzeugt, um im Anschluss die Likelihood zu bestimmen, dass x mit $\mu_\theta(l)$ und $\sigma_\theta(l)$ erzeugt wird $N(x \mid \mu_\theta, \sigma_\theta)$. Bei dieser Liklihood handelt es sich um die *reconstruction probability*. Aus den l *reconstruction probabilities* wird im Anschluss der Durchschnitt gebildet, um die finale *reconstruction probability* zu erhalten. Sollte diese unter einer Schwelle α liegen, wird x als anomal klassifiziert.

5 Evaluation der Algorithmen

Im Folgenden wird die Evaluation der Algorithmen durchgeführt. Dazu wird zuerst der Versuchsaufbau, die Parameterwahl und die Vorgehensweise für jeden Algorithmus erläutert. Im Anschluss werden die Ergebnisse der Untersuchung vorgestellt und begründet. Abschließend werden zusätzliche Erkenntnisse, die während der Untersuchung entstandenen sind, näher erläutert.

5.1 Versuchsaufbau

Die Algorithmen des Machine und Deep Learning werden meist als Blackboxen definiert. Dementsprechend können domänenspezifische Besonderheiten nur über die eingegebenen Daten und die gewählten Parameter in die Algorithmen einfließen. Im Folgenden wird zunächst auf die Datenrepräsentationen eingegangen, die in der Literatur benutzt wurden, um Discords innerhalb von Zeitreihen zu erkennen.

Im Paper [[Di13]] wurden Isolation Forest benutzt um Discords zu erkennen, indem die Daten mithilfe eines Sliding Windows extrahiert wurden. Dabei wird ein Sliding Window mit Größe N benutzt, sodass N Features entstehen, wobei jedes Feature einem Zeitschritt entspricht. In [[RDH19]] wurde der DBSCAN Algorithmus für die Erkennung von Discords benutzt. In diesem wurden der Datensatz in Subsequenzen aufgeteilt wurden und im Anschluss aufgrund verschiedener Transformationen in sogenannte „Meta Features“ umgewandelt. Bei der Untersuchung wurde der höchste F-Score mit den Features Mittelwert, Maximum, Minimum, Interquartilsabstand, Standardabweichung, Anzahl der Ausschläge, Median, Wölbung, Länge der Subsequenz, Schräge und linearen Regression erzielt. Ein ähnliches Vorgehen wurde in [[Hu18]] beschrieben, wobei in diesem die SVM für die Erkennung von Discords benutzt wurden. Dementsprechend werden die Features aus [[RDH19]], sowie den Koeffizient der Variation und die Variation des Trends aus [[Hu18]] benutzt. Die Oszillation, sowie das die Erkennung eines Rechtecksignals wird nicht als

Feature aus [[RDH19]] benutzt, da diese nur bei industriellen Prozessen relevant sind. Mithilfe dieses Vorgehens können theoretisch auch Subsequenzen verschiedener Länge miteinander verglichen werden, da die der Subsequenz extrahierten Anzahl der Features immer gleich ist. Um einen optimalen Algorithmus zu finden, werden bei der experimentellen Untersuchung die Subsequenzen sowohl als Sliding Window, als auch in Form ihrer Meta Features übergeben.

Bei der Untersuchung der Deep Learning Algorithmen wird, die für jeden Algorithmus definierte Datenrepräsentation, benutzt. Die Daten werden dabei normalisiert in die Netze eingegeben, um das Training zu beschleunigen. [[Le98], S. 17] Dazu wird die Min-Max Normalisierung genommen.

$$x_{norm} = \frac{x - \min(x)}{\max(x) - \min(x)} \qquad (5.1.1)$$

Aufgrund der Tatsache, dass manche Punktanomalien um ein Vielfaches größer sind als die normalen Datenwerte und dadurch die Min-Max Normalisierung sehr kleine Werte liefert, wird als Maximalwert das 99% Quantil benutzt.

Da eine Echtzeiterkennung simuliert werden soll, werden die Datenwerte der Zeitreihe in chronologischer Reihenfolge übergeben. Für CUSUM wird für die Bestimmung der Standardabweichung ein getrimmter Datensatz benutzt, sodass nur Werte unter dem 99% Quantil benutzt werden. Bei der Bestimmung der optimalen Parameter wird außerdem die traditionelle Vorgehensweise untersucht (sl = None), sowie die in 4.1 erklärte modifizierte Funktionsweise von CUSUM. Bei der Untersuchung des Isolation Forest wird der Datensatz zunächst für den Aufbau der Isolation Trees benutzt. Im Anschluss werden die Daten chronologisch überprüft und jeder Datenwert als normal oder anomal eingeordnet. Für DBSCAN wird der Algorithmus zunächst mit 10 Datenwerten initialisiert. Danach werden sequenziell neue Datenwerte hinzugefügt und die Zuordnung von jedem bisher hinzugefügten Datenwert überprüft. Jeder einzigartige Anomaliebereich der dadurch erkannt wurde fließt in die Berechnung des F-Scores mit ein. Ein ähnliches Vorgehen wird bei der Betrachtung von COF benutzt. Auch COF wird zunächst mit 10 Datenwerten initialisiert, sodass im Anschluss chronologisch neue Daten hinzugefügt und klassifiziert werden können. Für SVM, DeepAnT, Teleanom und VAE wird zunächst ein Trainingsbereich festgelegt, der nur normale Daten enthält. Im Anschluss werden alle Daten sequenziell an

die trainierten Algorithmen übergeben, sodass jeder Datenwert als normal oder anomal klassifiziert werden kann. Für Teleanom wird außerdem ein zweiter Bereich h im Datensatz bestimmt, der für die dynamische Bestimmung der Fehlergrenze benutzt wird.

Neben der Datenrepräsentation beeinflusst die Wahl der Parameter die Genauigkeit der Anomalieerkennung. Diese Parameter werden mithilfe eines Grid Search bestimmt, wobei der F-Score als Leistungskriterium benutzt wird.

Algorithmus	**Parameter**	**Belegung**
CUSUM	d	0.5-4 mit Schrittgröße 0.5
	B	5-30 mit Schrittgröße 1
	sl	None, 1, 2
Isolation Forest	Sequenzlänge	4-32 mit Schrittgröße 4
	Kontaminierung	0.01-0.2 mit Schrittgröße 0.01
DBSCAN	Sequenzlänge	4-16 mit Schrittgröße 4
	ε	1-7 mit Schrittgröße 0.25
COF	Sequenzlänge	4-16 mit Schrittgröße 4
	k	20-100 mit Schrittgröße 20
	Kontaminierung	0.01-0.05 mit Schrittgröße 0.01
SVM	Trainingsbereich	manuell bestimmt
	Sequenzlänge	4-16 mit Schrittgröße 4
	γ	0.00005-0.0015 mit Schrittgröße 0.00005
	ν	0.005-0.15 mit Schrittgröße 0.005
DeepAnT	Trainingsbereich	manuell bestimmt
	Eingabelänge	8, 12, 16, 25, 60, 75, 100
	Ausgabelänge	4-16 mit Schrittgröße 4
	Vielfaches der Standardabweichung	1-10 mit Schrittgröße 1
Teleanom	Trainingsbereich	manuell bestimmt
	Eingabelänge	8, 12, 16, 25, 60, 75, 100
	b	2, 4, 8, 12, 16, 20
	h	manuell bestimmt
VAE	Trainingsbereich	manuell bestimmt
	Eingabelänge	4, 8, 12, 16
	Wahrscheinlichkeitsquantil	0.01, 0.025, 0.05, 0.075
	l	20, 80

Tabelle 1: Parameterbelegung der Algorithmen

Die für den F-Score benötigten Kriterien Recall und Precision werden wie in 2.4.3 berechnet. Die für die Berechnung dieser Metriken benötigten korrekten Anomaliebereiche wurden

durch Experten der EXXETA AG bestimmt, sodass die Erkennung der Algorithmen mit einer menschlichen Erkennung verglichen wird. Dieser optimale F-Score wird für sieben Zeitreihen bestimmt, wobei der optimale F-Score sowohl für die Daten in Form eines Sliding Windows als auch in Form von Meta Features bestimmt wird. Dabei wird für den Vergleich zwischen den Algorithmen die Datenrepräsentation pro Datensatz benutzt, die den höchsten F-Score erhält. Die sieben verschiedenen Zeitreihen dienen dazu auszuschließen, dass ein Algorithmus nur für diese spezielle Zeitreihe einen hohen F-Score liefert. Auf die Ergebnisse der Untersuchung wird im folgenden Kapitel näher eingegangen.

5.2 Ergebnisse der Algorithmen

Die Ergebnisse der Untersuchung sind in Abbildung 30 zu sehen. Bei diesen handelt es sich um die durchschnittlichen Recall, Precision und F-Scores aus den sieben Datensätzen. Die komplette Auflistung der Ergebnisse mit den dazugehörigen Parametern sind im Anhang zu sehen.

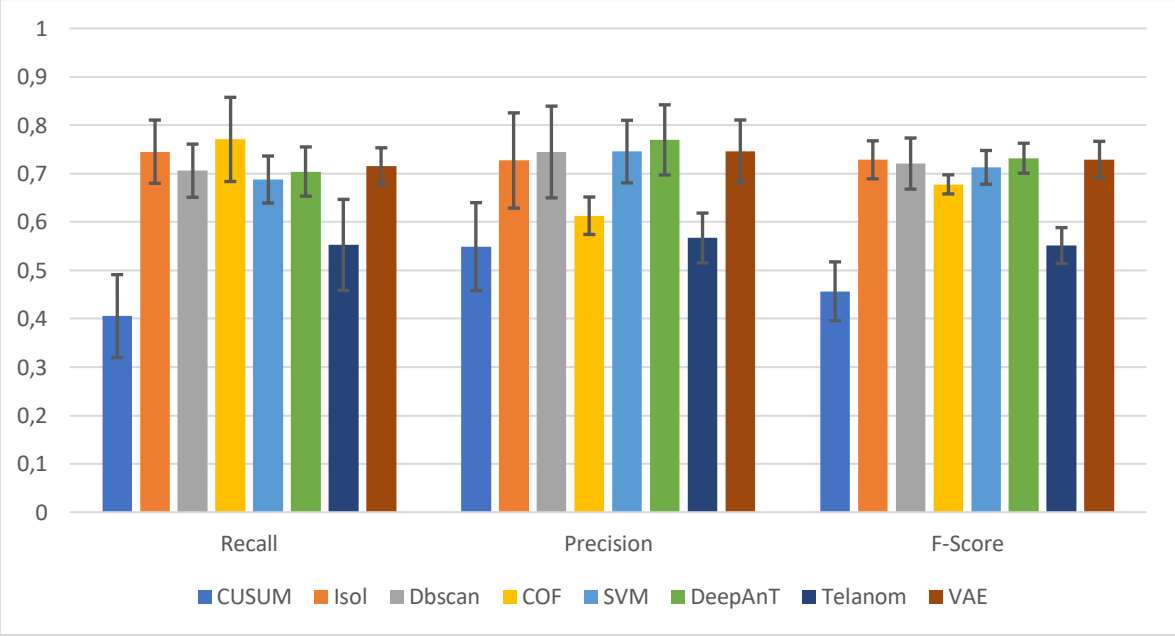

Abbildung 30: Graphische Darstellung der Scores der Algorithmen

Trotz der genannten Anpassung in 4.1 handelt es sich bei CUSUM um den simpelsten Algorithmus aus den Untersuchten. Aus diesem Grund sollte CUSUM als niedrigster Richtwert für die anderen Algorithmen dienen. CUSUM ist dazu konzipiert eine Anomalie zu erkennen und den laufenden Prozess zu stoppen, um weitere Schäden zu verhindern.

Dementsprechend sollen Anomalie spät erkannt werden, um unnötiges Stoppen des Prozesses zu vermeiden. Dieses Konzept spiegelt sich auch im Recall und Precision wider, wodurch der Precision Score 0.1 Punkt über dem Recall liegt. Da CUSUM in der Untersuchung den niedrigsten F-Score erhielt, wurde die Rolle von CUSUM als niedrigster Richtwert bestätigt.

Teleanom erhält in der Untersuchung einen niedrigen F-Score im Vergleich zu den anderen Machine und Deep Learning Algorithmen. Ein Grund für das schlechte Ergebnis ist die Art und Weise wie anomale Bereiche erkannt werden. Zum einen wird mithilfe eines Referenzbereichs h eine Fehlergrenze erzeugt. Mithilfe des Referenzbereichs wird eine minimale Fehlergrenze bestimmt, die für die Bestimmung aller zukünftigen Anomalien benutzt wird. Da einzelne Datenwerte stark vom Mittelwert abweichen, wird eine Grenze gefunden, die diese Punktanomalien ausgrenzt. Dadurch entsteht eine sehr hohe Fehlergrenze in Bezug auf den Referenzbereich. Dadurch wurde während der Untersuchung die Wahl des Referenzbereichs h als ein wichtigeres Problem als das optimale Training des LSTM eingestuft. Die durch den Referenzbereich festgelegten hohen Fehlergrenzen in Kombination mit dem Hinzufügen von festen Anomaliebereichen zu Punktanomalien mithilfe von b sorgen für eine späte, jedoch genaue Erkennung von Anomalien. Die dadurch entstehenden hohen Precision und niedrigen Recall Scores sind jedoch nicht in den Ergebnissen der Algorithmen zu erkennen, da diese aufgrund der besten F-Scores ausgewählt wurden.

Ohne die bereits erwähnten Ausnahmen CUSUM und Teleanom erhalten die verbleibenden Algorithmen vergleichbare F-Scores. Dementsprechend hängt der Erfolg der Algorithmen vom jeweiligen Datensatz ab. Um dies zu veranschaulichen dient die folgende Abbildung.

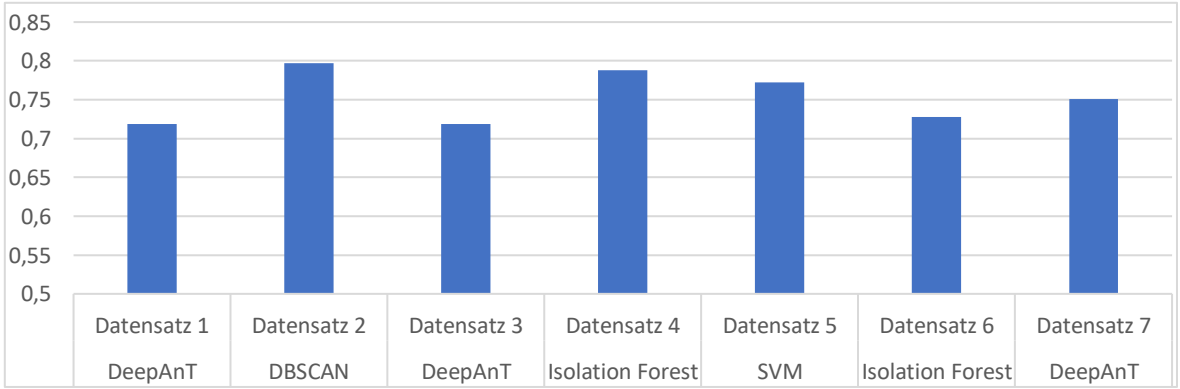

Abbildung 31: Bester F-Score pro Datensatz

In der Graphik ist zu erkennen, dass die höchsten F-Scores durch Isolation Forest, DBSCAN, SVM und DeepAnT erzeugt wurden. Dementsprechend können sowohl Machine Learning in Form einer unsupervised Anomalieerkennung und semi-supervised Anomalieerkennung, als auch Deep Learning Algorithmen eine optimale Erkennung erzielen.

Bei der Betrachtung der durch die Algorithmen erkannten Discords konnte festgestellt werden, dass die markierten Gebiete sich um einen starken Ausreißer gruppieren. Während dieses Verhalten erwartet ist, besitzt der frühste Discord aus einem solchen markierten Gebiet meist einen starken Ausreißer am Ende seines Bereichs. Dieses Verhalten soll mithilfe eines einfachen Beispiels in Abbildung 32 dargestellt werden.

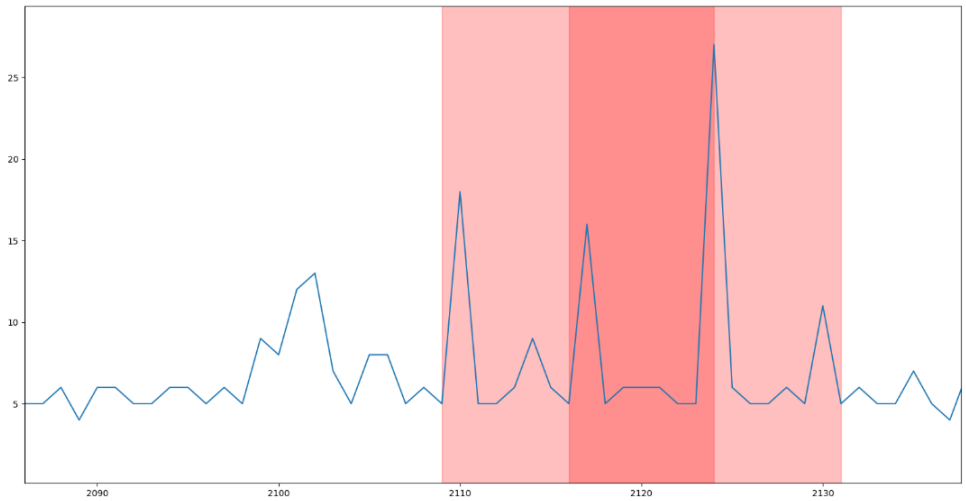

Abbildung 32: Detektion eines Discords durch VAE

In der Abbildung ist zu sehen, dass sich der Ausreißer an Position 2125 aufgrund von drei vorangegangenen Ausschlägen ankündigt. Der erste Discord der jedoch in diesem Gebiet als Anomalie erkannt wurde ist der Bereich, in dem der Ausreißer an Position 2125 das erste Mal aufgenommen wird. Dementsprechend wird die Anomalie erst erkannt, wenn der starke Ausreißer bereits eingetreten ist.

Dieses Verhalten kann auf mehrere Ursachen zurückgeführt werden. Eine dieser Ursachen kann die Art und Weise sein, wie die Daten annotiert wurden. Die Annotationen umfassen Gebiete, die ein Mensch als anomal klassifizieren würde. Da eine frühzeitige Erkennung nur dann belohnt wird, wenn sie sich auch innerhalb eines annotierten Bereichs befindet, sollten die annotierten Bereiche auch einige Punkte vor dem tatsächlich anomalen Bereich enthalten. Bei den in der Untersuchung benutzen Annotationen ist dies jedoch nicht immer

der Fall. Dadurch könnten Parameterbelegungen, die eine frühzeitigere Erkennung erreicht haben, einen schlechteren F-Score erhalten haben, da diese frühzeitigen Erkennungen in keinem annotierten Beriech lagen. Eine zweite Ursache für diese späten Erkennungen kann ihren Ursprung in den Algorithmen haben. Jeder der in der Untersuchung benutzen Algorithmen wurde mit dem Fokus auf die Genauigkeit der Erkennung entwickelt. Aus diesem Grund wird die Leistungsfähigkeit der Algorithmen meist aufgrund einer traditionellen Genauigkeitsbewertung wie in 2.4.1 bestimmt und mit anderen Algorithmen verglichen. Aufgrund der Fixierung auf die Genauigkeit, fehlen den Algorithmen Mechanismen, die eine frühzeitige Erkennung ermöglichen. Die dritte Ursache könnten die Daten sein. Die Entscheidungen der Algorithmen beruhen in jedem Fall auf den untersuchten Daten, wobei sich Komplexität der Anomalieerkennung pro Datensatz nur schwer definieren lässt. Aus diesem Grund kann es nicht in allen Fällen sein, die Anomalie vor dem tatsächlichen Eintreten zu erkennen. Des Weiteren wurden die Daten nur univariat untersucht, da keine Korrelation zwischen den Komponenten erkannt wurde. Dementsprechend ist es möglich, dass Informationen, die zu einer frühzeitigeren Erkennung führen könnten, bislang noch nicht erfasst werden.

5.3 Erkenntnisse aus der Evaluation

Während die F-Scores der Machine Learning und Deep Learning Algorithmen relativ nah beieinander liegen, sind während der Erstellung der Scores einige allgemeine Vor- und Nachteile festgestellt wurden. Diese beruhen auf der Einteilung der Algorithmen in Semi-Supervised und Unsupervised Anomalierkennung. Die Semi-Supervised Anomalie-erkennungsalgorithmen teilen ihre Erkennung in zwei Teile ein, sodass das Modell zuerst trainiert wird und mit diesem trainierten Modell jeder Punkt als normal oder anomal eingeordnet werden kann. Da das Training getrennt von der Klassifizierung stattfindet, wird während der Klassifikation der Daten das Modell nicht verändert. Daraus entsteht der Vorteil, dass die Rechenleistung für die Klassifikation eines Datenwertes konstant ist und durch eine Trennfunktion beschrieben werden kann. Aus dieser Eigenschaft entsteht jedoch auch der Nachteil, dass das Modell keine Möglichkeit hat, sich aufgrund von neuen Daten anzupassen. Sollte ein solcher Concept Drift eintreten, muss das Modell neu trainiert werden. Dieses Problem besitzen die unsupervised Anomalieerkennungsalgorithmen nicht, da sich bei diesen Algorithmen die Klassifikation eines Datenwertes mit jedem neu hinzugefügten Datenwert ändern kann. Diese Eigenschaft sorgt jedoch auch für mehrere Nachteile. Aufgrund der Funktionsweise von DBSCAN und COF steigt die Rechenleistung potenziell mit der Gesamtzahl an bereits hinzugefügten Datenwerten. Außerdem besteht die Gefahr, dass mit steigender Anzahl an Anomalien die Isoliertheit der Anomalien sinkt, da im Datensatz bereits ähnlich Datenwerte vorhanden sind. Um den konstanten Anstieg zu verhindern, kann mit jedem hinzugefügten Datenwert ein alter Datenwert entfernt werden, sodass die Anzahl der Datenwerte im Datensatz gleichbleibt. Dadurch entsteht jedoch ein neues Problem. Das Entfernen eines Datenwertes ist ähnlich aufwendig ist wie das Hinzufügen eines Datenwertes, da sich auch beim Entfernen eines Datenwertes die Zuordnung ändern kann.

Im Versuchsaufbau wurde erklärt, dass die Daten als Sliding Window und in Form von Meta Features an die Machine Learning Algorithmen übergeben werden. Bei der Untersuchung wurde festgestellt, dass vor allem DBSCAN von der Umwandlung der Daten in Meta Features profitierte. Das genaue Gegenteil entstand bei COF. COF erhielt mit Meta Features nur einen sehr geringen F-Score, wobei dieser durch die Erhöhung von k zunahm. Da die Zeit für die Berechnung des Connectivity-Based Outlier Factor mit k steigt, wurde k auf 100

begrenzt. Trotz dieser hohen k Werte erhielt COF mit den Meta Features nur die Hälfte des F-Scores, im Vergleich mit den Daten als Sliding Window. Bei der SVM und dem Isolation Forest führen beide Datenformen zu ähnlichen F-Scores, sodass die Wahl der Datenform vom Datensatz abhängt.

Während der Auswertung der Ergebnisse wurde bereits darauf eingegangen, dass manche Algorithmen, wie Teleanom, hohe Precision Scores erzielen kann, dies jedoch nicht durch die Ergebnisse widergespiegelt wird. Die Ursache liegt in der Berechnung des F-Scores, da dieser aus dem harmonischen Mittel von Precision und Recall berechnet wurde. Dadurch tendieren Precision und Recall dazu, nahe beieinander zu liegen. Die Erweiterung von Precision und Recall in 2.4.3 führt dazu, dass sich die Funktion der Scores stärker unterscheidet im Vergleich zu der Definition aus 2.4.1. Aus diesem Grund sollte bei zukünftigen Auswertungen darauf geachtet werden, ob eine der beiden Komponenten für die Untersuchung wichtiger ist, sodass die Berechnung des F-Scores entsprechend angepasst werden kann.

Ein VAE erzeugt seine Codierung mithilfe einer Normalverteilung und einer Zufallsvariable ε. Um die Beeinflussung dieser Zufallsvariable zu reduzieren wird die Codierung und $reconstruction\ probabilities\ l$ mal erzeugt. Dementsprechend sollte bei der Benutzung eines VAE darauf geachtet werden, dass die Erkennung für niedrige l nicht reproduzierbar ist, sodass mehrere Analysen einer Zeitreihe zu anderen Ergebnissen führen kann.

6 Zusammenfassung und Ausblick

In dieser Arbeit wurde untersucht, welche Algorithmen sich zur Echtzeiterkennung von Anomalien in Monitoringdaten eignen. Dazu wurde zuerst mithilfe des Precision and Recall for Timeseries eine Metrik vorgestellt, die eine Anomalieerkennung sowohl aufgrund ihrer Genauigkeit, als auch ihrer Rechtzeitigkeit bewertet. Um die Rechtzeitigkeit zu bewerten, müssen die Anomalien in Form von Subsequenzanomalien vorliegen. Während der Untersuchung der Daten wurde festgestellt, dass zwischen den Zeitreihen keine Korrelation vorliegt. Aufgrund der somit erkannten Bedingungen wurden Algorithmen zusammengetragen, die univariate Streamingdaten analysieren und Subsequenzanomalien in diesen erkennen. Zur Untersuchung der Algorithmen wurden sieben real erhobenen Zeitreihen untersucht, um die Abhängigkeit des Erfolgs aufgrund einer speziellen Zeitreihe auszuschließen. Die Anomalien innerhalb der Zeitreihen wurden dabei mithilfe von Experten annotiert. Von den acht untersuchten Algorithmen erhielten sechs vergleichbare mittlere F-Scores, sodass keine klare Empfehlung für einen Algorithmus ausgesprochen werden kann. Weitergehend wurde festgestellt, dass der Erfolg der einzelnen Algorithmen stark von der untersuchten Zeitreihe abhängt, sodass für jede Zeitreihe ein passender Algorithmus gesucht werden muss.

Dieses Ergebnis kann zum einen auf die Tatsache zurückgeführt werden, dass die untersuchten Algorithmen ihren Fokus auf die Genauigkeit und nicht auf eine frühzeitige Erkennung legen. Dementsprechend sollte bei der Erstellung zukünftiger Anomalieerkennungsalgorithmen darauf geachtete werden, dass ähnliche Kriterien berücksichtigt werden, wie bei der Erstellung der Metrik Precision and Recall for Timeseries.

Des Weiteren konnten im Rahmen dieser Arbeit nicht alle Algorithmen untersucht werden, die die oben genannten Bedingungen erfüllen. Im Fall von VAE wurde sich auf die Grundform beschränkt, da Erweiterungen wie [[Xu18], [Su19]] und deren verbesserte

Funktionsfähigkeit nur aufgrund der traditionellen Genauigkeitsmaße überprüft wurde. Neben der erhöhten Genauigkeit besitzt Donut [[Xu18]] außerdem die Besonderheit, dass lückenhafte Daten benutzt und Annotationen in den Trainingsprozess einfließen können. OmniAnomaly [[Su19]] erweitert das traditionelle VAE, sodass auch multivariate Zeitreihen ausgewertet werden können. Diese Besonderheiten waren bei den in dieser Arbeit untersuchten Daten nicht nötig, könnten jedoch bei zukünftig untersuchten Datensätzen relevant sein.

Eines der größten Probleme, während er Erstellung dieser Arbeit, war das Finden von Algorithmen, die univariate Subsequenzanomalien in Streamingdaten erkennen können. Um unter diesen Bedingungen passende Algorithmen zu finden, wurden deshalb zunächst Survey Paper benutzt. Dabei wurde festgestellt, dass sich Survey Paper im Forschungsgebiet der Anomalieerkennung in Zeitreihen meist auf ein bis zwei Kategorien beschränken. [[BW20]] betrachtet univariate Anomalieerkennungsalgorithmen zur Erkennung von Punkt- und Sequenzanomalien. [[Bl20]] untersucht sowohl Punkt- als auch Sequenzanomalien in univariaten und multivariaten Zeitreihen. In der Praxis sorgt der Kontext meist dafür, dass die Algorithmen mehr als zwei Bedingungen erfüllen müssen. Demensprechend hilft ein anwendungsfallorientiertes Vorgehen wie in [[CBK09]] dabei, einen vergleichbaren Anwendungsfall und die möglichen Algorithmen zu erkennen. [[CBK09]] ist jedoch nicht mehr aktuell und betrachtet nur eine Auswahl an Anwendungsfällen. Dementsprechend könnte ein zukünftiges Survey Paper Algorithmen zusammentragen und untersuchen, welche Algorithmen für welche Kategoriekombinationen geeignet sind. Diese Kategorien sollten den Datentyp (univariate/multivariate), den Lerntyp (Semi-Supervised/Unsupervised), die Anomalieart (Punktanomalie/Sequenzanomalie) und die Datenbereitstellung (statisch/streaming) betrachten.

Anhang

Anhangsverzeichnis

Anhang 1 Mittlerer Recall, Precision und F-Score für die Algorithmen

Algorithmus	Recall	Precision	F-Score
CUSUM	0.3969±0.0763	0.4898±0.1220	0.4338±0.0899
Isolation Forest	0.7453±0.0654	0.7271±0.0984	0.7285±0.0393
DBSCAN	0.7060±0.0549	0.7447±0.0948	0.7207±0.0529
COF	0.7706±0.0870	0.6129±0.0388	0.6777±0.0198
SVM	0.6713±0.0359	0.6807±0.0021	0.7129±0.0349
DeepAnT	0.7598±0.0337	0.7527±0.0404	0.7318±0.0310
Teleanom	0.5528±0.0941	0.5670±0.0514	0.5512±0.0371
VAE	0.7157±0.0376	0.7463±0.0645	0.7292±0.0376

Anhang 2 Ergebnisse für CUSUM

d: 0.5-4 mit Schrittgröße 0.5

B: 5-30 mit Schrittgröße 1

Sl: None, 1, 2

CUSUM	Parameter	Recall	Precision	F-Score
Datensatz 1	d = 0.5 B = 11 sl = None	0.3371	0.4609	0.3894
Datensatz 2	d = 1 B = 28 sl = 1	0.5164	0.4246	0.4660
Datensatz 3	d = 1 B = 11 sl = 1	0.3260	0.4703	0.3851
Datensatz 4	d = 1 B = 12 sl = 1	0.5138	0.5811	0.5454
Datensatz 5	d = 0.5 B = 25 sl = 1	0.4494	0.5786	0.5059
Datensatz 6	d = 0.5 B = 14 sl = 1	0.4097	0.6635	0.5066
Datensatz 7	d = 0.5 B = 17 sl = 1	0.2849	0.6647	0.3989
Total		0.4053±0.0858	0.5491±0.0901	0.4567±0.0608

Anhang 3 Ergebnisse für Isolation Forest

Sequenzlänge: 4-32 mit Schrittgröße 4

Kontaminierung: 0.01-0.2 mit Schrittgröße 0.01

Isolation Forest	**Datentyp**	**Parameter**	**Recall**	**Precision**	**F-Score**
Datensatz 1	Sliding Window	Sequenzlänge = 20 Kontaminierung = 0.7	0.8265	0.6268	0.7129
Datensatz 2	Meta Feature	Sequenzlänge = 4-20 Kontaminierung = 0.02	0.8116	0.6447	0.7186
Datensatz 3	Sliding Window	Sequenzlänge = 8 Kontaminierung = 0.075	0.7486	0.5871	0.6570
Datensatz 4	Sliding Window	Sequenzlänge = 20 Kontaminierung = 0.05	0.7750	0.8002	0.7882
Datensatz 5	Sliding Window	Sequenzlänge = 8 Kontaminierung = 0.05	0.7418	0.7960	0.7708
Datensatz 6	Meta Feature	Sequenzlänge = 4-16 Kontaminierung = 0.02	0.6917	0.7678	0.7278
Datensatz 7	Meta Feature	Sequenzlänge = 4-16 Kontaminierung = 0.03	0.6222	0.8673	0.7245
Total			0.7453±0.0654	0.7271±0.0984	0.7285±0.0393

Anhang 4 Ergebnisse für DBSCAN

Sequenzlänge: 8-16 mit Schrittgröße 4

Epsilon: 1-7 mit Schrittgröße 0.25

DBSCAN	**Datentyp**	**Parameter**	**Recall**	**Precision**	**F-Score**
Datensatz 1	Meta Feature	Sequenzlänge = 8-12 Epsilon = 3 Minpts = 2	0.6847	0.7521	0.7168
Datensatz 2	Meta Feature	Sequenzlänge = 8-8 Epsilon = 7 Minpts = 2	0.7508	0.8487	0.7968
Datensatz 3	Meta Feature	Sequenzlänge = 8-12 Epsilon = 3 Minpts = 3	0.6809	0.5859	0.6298
Datensatz 4	Meta Feature	Sequenzlänge = 8-12 Epsilon = 4 Minpts = 2	0.6523	0.8907	0.7531
Datensatz 5	Meta Feature	Sequenzlänge = 8-8 Epsilon = 3 Minpts = 3	0.7789	0.7445	0.7613
Datensatz 6	Sliding Window	Sequenzlänge = 12 Epsilon = 7 Minpts = 5	0.7668	0.6773	0.7193
Datensatz 7	Meta Feature	Sequenzlänge = 8-16 Epsilon = 0.75 Minpts = 4	0.6276	0.7136	0.6679
Total			0.7060±0.0549	0.7447±0.0948	0.7207±0.0529

Anhang 5 Ergebnisse für COF

Sequenzlänge: 8-16 mit Schrittgröße 4

k: 20-100 mit Schrittgröße 20

Kontaminierung: 0.01-0.05 mit Schrittgröße 0.01

COF	**Datentyp**	**Parameter**	**Recall**	**Precision**	**F-Score**
Datensatz 1	Sliding Window	Sequenzlänge = 16 k = 60 Kontaminierung = 0.05	0.8339	0.5666	0.6747
Datensatz 2	Sliding Window	Sequenzlänge = 16 k = 80 Kontaminierung = 0.01	0.8825	0.5649	0.6889
Datensatz 3	Sliding Window	Sequenzlänge = 12 k = 20 Kontaminierung = 0.03	0.6998	0.6149	0.6546
Datensatz 4	Sliding Window	Sequenzlänge = 16 k = 80 Kontaminierung = 0.03	0.7321	0.6405	0.6833
Datensatz 5	Sliding Window	Sequenzlänge = 12 k = 60 Kontaminierung = 0.05	0.8292	0.6172	0.7077
Datensatz 6	Sliding Window	Sequenzlänge =16 k = 60 Kontaminierung = 0.05	0.8058	0.6016	0.6889
Datensatz 7	Sliding Window	Sequenzlänge = 8 k = 100 Kontaminierung = 0.02	0.6113	0.6847	0.6459
Total			0.7706±0.0870	0.6129±0.0388	0.6777±0.0198

Anhang 6 Ergebnisse für SVM

Sequenzlänge: 4-16 mit Schrittgröße 4

γ: 0.00005-0.0015 mit Schrittgröße 0.00005

ν: 0.005-0.15 mit Schrittgröße 0.005

SVM	**Datentyp**	**Parameter**	**Recall**	**Precision**	**F-Score**
Datensatz 1	Sliding Window	Trainingsbereich = [10:941] Sequenzlänge = 8 γ = 0.001 $\nu = 0.06$	0.7072	0.6827	0.6947
Datensatz 2	Meta Feature	Trainingsbereich = [0:968] Sequenzlänge = 8-12 γ = 0.0003 $\nu = 0.005$	0.6354	0.6786	0.6562
Datensatz 3	Sliding Window	Trainingsbereich = [22:1090] Sequenzlänge = 8 γ = auto (0.125) $\nu = 0.02$	0.6893	0.6930	0.6912
Datensatz 4	Sliding Window	Trainingsbereich =[0:1281] Sequenzlänge = 8 γ = 0.00015 $\nu = 0.01$	0.64336	0.8071	0.7160
Datensatz 5	Sliding Window	Trainingsbereich = [793:1266] Sequenzlänge = 8 γ = 0.0013 $\nu = 0.05$	0.7885	0.7568	0.7723
Datensatz 6	Sliding Window	Trainingsbereich = [77:1286] Sequenzlänge = 12 γ = 0.000005 $\nu = 0.19$	0.6973	0.7362	0.7162
Datensatz 7	Sliding Window	Trainingsbereich = [1540:2770] Sequenzlänge = 12 γ = 0.000255 $\nu = 0.01$	0.6530	0.8643	0.7439
Total			0.6877±0.0486	0.7455±0.0647	0.7129±0.0349

Anhang 7 Ergebnisse für DeepAnT

Eingabelänge: 8, 12, 16, 25, 60, 75, 100

Ausgabelänge: 4, 8, 12, 16

Vielfaches der Standardabweichung: 1-10 mit Schrittgröße 1

DeepAnT	**Parameter**	**Recall**	**Precision**	**F-Score**
Datensatz 1	Trainingsbereich = [10:941] Eingabelänge = 25 Ausgabelänge = 16 Vielfaches der Standardabweichung = 4	0.7258	0.7123	0.7190
Datensatz 2	Trainingsbereich = [1856:2911] Eingabelänge = 25 Ausgabelänge = 12 Vielfaches der Standardabweichung = 5	0.7932	0.7931	0.7932
Datensatz 3	Trainingsbereich = [22:1090] Eingabelänge = 75 Ausgabelänge = 8 Vielfaches der Standardabweichung = 5	0.6623	0.7848	0.7184
Datensatz 4	Trainingsbereich = [0:1281] Eingabelänge = 50 Ausgabelänge = 8 Vielfaches der Standardabweichung = 7	0.6518	0.8402	0.7341
Datensatz 5	Trainingsbereich = [793:1253] Eingabelänge = 12 Ausgabelänge = 8 Vielfaches der Standardabweichung = 9	0.6931	0.6782	0.6856
Datensatz 6	Trainingsbereich = [77:1286] Eingabelänge = 25 Ausgabelänge = 16 Vielfaches der Standardabweichung = 4	0.7524	0.6928	0.7214
Datensatz 7	Trainingsbereich = [1540:2770] Eingabelänge = 25 Ausgabelänge = 12 Vielfaches der Standardabweichung = 8	0.6513	0.8861	0.7507
Total		0.7043±0.0509	0.7696±0.0725	0.7318±0.0310

Anhang 8 Ergebnisse für Teleanom

Eingabelänge: 8, 12, 16, 25, 60, 75, 100

b: 2, 4, 8, 12, 16, 20

Teleanom	**Parameter**	**Recall**	**Precision**	**F-Score**
Datensatz 1	Trainingsbereich = [10:941] Eingabelänge = 8 b = 8 h = [239, 745]	0.4784	0.5753	0.5224
Datensatz 2	Trainingsbereich = [0:970] Eingabelänge = 12 b = 16 h = [16020, 2845]	0.5785	0.5111	0.5427
Datensatz 3	Trainingsbereich = [22:1090] Eingabelänge = 12 b = 8 h = [56, 1090]	0.4968	0.5719	0.5317
Datensatz 4	Trainingsbereich = [0:1281] Eingabelänge = 8 b = 12 h = [0, 920]	0.6919	0.4827	0.5687
Datensatz 5	Trainingsbereich = [793:1253] Eingabelänge = 75 b = 8 h = [2660, 2900]	0.6539	0.5843	0.6171
Datensatz 6	Trainingsbereich = [77:1286] Eingabelänge = 8 b = 12 h = [100, 690]	0.5673	0.5910	0.5789
Datensatz 7	Trainingsbereich = [1540:2770] Eingabelänge = 8 b = 20 h = [2470, 2910]	0.4012	0.6526	0.4969
Total		0.5528±0.0941	0.5670±0.0514	0.5512±0.0371

Anhang 9 Ergebnisse für VAE

Eingabelänge: 4, 8, 12, 16

Wahrscheinlichkeitsquantil: 0.01, 0.025, 0.05, 0.075

l: 20, 80

VAE	**Parameter**	**Recall**	**Precision**	**F-Score**
Datensatz 1	Trainingsbereich = [10:941] Eingabelänge=8 Wahrscheinlichkeitsquantil= 0.05 l=80	0.7164	0.6603	0.6872
Datensatz 2	Trainingsbereich = [2069:2581] Eingabelänge=8 Wahrscheinlichkeitsquantil= 0.01 l=20	0.7787	0.7941	0.7879
Datensatz 3	Trainingsbereich = [22:1090] Eingabelänge=12 Wahrscheinlichkeitsquantil= 0.035 l=20	0.6772	0.6695	0.6733
Datensatz 4	Trainingsbereich = [0:1281] Eingabelänge=12 Wahrscheinlichkeitsquantil= 0.025 l=20	0.6815	0.8567	0.7591
Datensatz 5	Trainingsbereich = [793:1253] Eingabelänge=8 Wahrscheinlichkeitsquantil= 0.05 l=20	0.7604	0.7448	0.7525
Datensatz 6	Trainingsbereich = [77:1286] Eingabelänge=16 Wahrscheinlichkeitsquantil= 0.025 l=20	0.7164	0.7239	0.7201
Datensatz 7	Trainingsbereich = [1540:2770] Eingabelänge=16 Wahrscheinlichkeitsquantil= 0.025 l=80	0.6796	0.7752	0.7242
Total		0.7157±0.0376	0.7463±0.0645	0.7292±0.0376

Literaturverzeichnis

[[AA12]] Abusafia, A.; Al Aghbari, Z.: Detection of variable length anomalous subsequences in data streams. In International Journal of Intelligent Information and Database Systems, 2012, Ausgabe 6; S. 273–288.

[[Ab18]] Aboode, A.: Anomaly Detection in Time Series Data Based on Holt-Winters Method, Stockholm, Schweden, 2018.

[[AC15]] An, J.; Cho, S.: Variational autoencoder based anomaly detection using reconstruction probability. In Special Lecture on IE, 2015, Ausgabe 2; S. 1–18.

[[Ad06]] Ada Wai-Chee Fu et al.: Finding Time Series Discords Based on Haar Transform. In (Xue Li; Osmar R. Za\"ıane; Zhanhuai Li Hrsg.), In Advanced Data Mining and Applications, Second International Conference, ADMA 2006, Xi'an, China, August 14-16, 2006, Proceedings. Springer, 2006, Ausgabe 4093; S. 31–41.

[[An19]] Andrew Ng: CS229 Lecture notes. Part V: Support Vector Machines, Verfügbar unter http://cs229.stanford.edu/notes2019fall/cs229-notes3.pdf, Zuletzt geprüft 08.11.2020.

[[bl18]] blog.paperspace.com: Bild Gradient Descent, Verfügbar unter https://blog.paperspace.com/content/images/2018/05/challenges-1.png, Zuletzt geprüft 11.11.2020.

[[Bl20]] Bl'azquez-Garc'ia, A. et al.: A review on outlier/anomaly detection in time series data. In arXiv e-prints, 2020, Ausgabe abs/2002.04236; S. 1–32.

[[BW20]] Braei, M.; Wagner, S.: Anomaly Detection in Univariate Time-series: A Survey on the State-of-the-Art. In ArXiv, 2020, Ausgabe abs/2004.00433; S. 1–39.

[[CBK09]] Chandola, V.; Banerjee, A.; Kumar, V.: Anomaly Detection: A Survey. In ACM Comput. Surv., 2009, Ausgabe 41; 15: 1–58.

[[CDA18]] Chau, P. M.; Duc, B. M.; Anh, D. T.: Discord Discovery in Streaming Time Series Based on an Improved HOT SAX Algorithm, In Proceedings of the Ninth International Symposium on Information and Communication Technology. Association for Computing Machinery, New York, NY, USA, 2018; S. 24–30.

[[CDD11]] Celik, M.; Dadaşer-Çelik, F.; Dokuz, A. Ş.: Anomaly detection in temperature data using DBSCAN algorithm. In 2011 International Symposium on Innovations in Intelligent Systems and Applications, 2011; S. 91–95.

[[CIL19]] Carreño, A.; Inza, I.; Lozano, J.: Analyzing rare event, anomaly, novelty and outlier detection terms under the supervised classification framework. In Artificial Intelligence Review, 2019, Ausgabe 53; S. 3575–3594.

[[Cl15]] CloudWalk, Inc.: Discrete DBSCAN, Verfügbar unter https://github.com/cloudwalk/ddbscan.

[[Di13]] Ding, Z.: An Anomaly Detection Approach Based on Isolation Forest Algorithm for Streaming Data Using Sliding Window, In IFAC Proceedings Volumes, 2013, Ausgabe 46; S. 12–17.

[[DM14]] Diederik P. Kingma; M. Welling: Auto-Encoding Variational Bayes. In CoRR, 2014, Ausgabe abs/1312.6114; S. 1–14.

[[Dr08]] Dragoljub Pokrajac et al.: Incremental Connectivity-Based Outlier Factor Algorithm. In (Erol Gelenbe; Samson Abramsky; Vladimiro Sassone Hrsg.), In Visions of Computer Science - BCS International Academic Conference, Imperial College, London, UK, 22-24 September 2008. British Computer Society, 2008; S. 211–224.

[[Es96]] Ester, M. et al.: A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise, In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining. AAAI Press, 1996; S. 226–231.

[[Es98]] Ester, M. et al.: Incremental Clustering for Mining in a Data Warehousing Environment, In VLDB, 1998; S. 323–333.

[[Fa06]] Fawcett, T.: Introduction to ROC analysis. In Pattern Recognition Letters, 2006, Ausgabe 27; S. 861–874.

[[FKZ08]] Fei Tony Liu; Kai Ming Ting; Zhi-hua Zhou: Isolation Forest, In ICDM ’08: Proceedings of the 2008 Eighth IEEE International Conference on Data Mining. IEEE Computer Society, 2008; S. 413–422.

[[Fu06]] Fu, A. W.-C. et al.: Finding Time Series Discords Based on Haar Transform. In (Li, X.; Zaïane, O. R.; Li, Z. Hrsg.), In Advanced Data Mining and Applications. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, Ausgabe 4093; S. 31–41.

[[Ga18]] Gao, J.: Introduction to Convolutional Neural Networks, Verfügbar unter https://web.stanford.edu/class/cs231a/lectures/intro_cnn.pdf, Zuletzt geprüft 22.11.2020.

[[Go16]] Goix, N.: How to Evaluate the Quality of Unsupervised Anomaly Detection Algorithms? In ArXiv, 2016; S. 1–13.

[[GS05]] Graves, A.; Schmidhuber, J.: Framewise phoneme classification with bidirectional LSTM and other neural network architectures. In Neural networks the official journal of the International Neural Network Society, 2005, Ausgabe 18; S. 602–610.

[[Gu18]] Guo, Y. et al.: Multidimensional Time Series Anomaly Detection: A GRU-based Gaussian Mixture Variational Autoencoder Approach. In (Jun Zhu; Ichiro Takeuchi Hrsg.), In Proceedings of The 10th Asian Conference on Machine Learning. PMLR, 2018, Ausgabe 95; S. 97–112.

[[HA18]] Hyndman, R. J.; Athanasopoulos George: Forecasting: Principles and Practice, Verfügbar unter https://otexts.com/fpp2/residuals.html, Zuletzt geprüft 08.11.2020.

[[HS97]] Hochreiter, S.; Schmidhuber, J.: Long Short-term Memory. In Neural computation, 1997, Ausgabe 9; S. 1735–1780.

[[Hu18]] Hu, M. et al.: Detecting Anomalies in Time Series Data Via A Meta-feature Based Approach. In IEEE Access, 2018, Ausgabe 99; S. 1–17.

[[Hu18a]] Hundman, K. et al.: Detecting Spacecraft Anomalies Using LSTMs and Nonparametric Dynamic Thresholding. In CoRR, 2018, Ausgabe abs/1802.04431; S. 1–9.

[[Hu18b]] Hundman, K. et al.: Telemanom, Verfügbar unter https://github.com/PKUZHOU/anomaly_det, Zuletzt geprüft 15.11.2020.

[[Ji02]] Jian Tang et al.: Enhancing Effectiveness of Outlier Detections for Low Density Patterns. In (Ming-Shan Cheng; Philip S. Yu; Bing Liu Hrsg.), In Advances in Knowledge Discovery and Data Mining, 6th Pacific-Asia Conference, PAKDD 2002, Taipei, Taiwan, May 6-8, 2002, Proceedings. Springer, 2002, Ausgabe 2336; S. 535–548.

[[Jo91]] Josef Hochreiter: Untersuchungen zu dynamischen neuronalen Netzen, München, 1991.

[[KH18]] Kuo, P.-H.; Huang, C.-J.: A Green Energy Application in Energy Management Systems by an Artificial Intelligence-Based Solar Radiation Forecasting Model. In Energies, 2018, Ausgabe 11; S. 819–834.

[[Ko07]] Kotsiantis, S.: Supervised Machine Learning: A Review of Classification Techniques. In Informatica (Ljubljana), 2007, Ausgabe 31; S. 249–268.

[[Kr07]] Kriesel, D.: Ein kleiner Überblick über Neuronale Netze, 2007.

[[LA15]] Lavin, A.; Ahmad, S.: Evaluating Real-time Anomaly Detection Algorithms - the Numenta Anomaly Benchmark. In 14th International Conference on Machine, 2015, Ausgabe abs/1510.03336; S. 1–8.

[[La18]] Laptev, N.: AnoGen: Deep Anomaly Generator, In Outlier Detection De-constructed (ODD) Workshop. Available online: https://research. fb. com/publications/anogen-deep-anomaly-generator/(accessed on 20 August 2018), 2018; S. 1–3.

[[LA19]] Lavin, A.; Ahmad, S.: Numeta Benchmark Repository, Verfügbar unter https://github.com/numenta/NAB, Zuletzt geprüft 10.11.2020.

[[LAF15]] Laptev, N.; Amizadeh, S.; Flint, I.: Generic and Scalable Framework for Automated Time-Series Anomaly Detection, In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Association for Computing Machinery, New York, NY, USA, 2015; S. 1939–1947.

[[LC01]] Lämmel, U.; Cleve, J.: Lehr- und Übungsbuch künstliche Intelligenz. Mit 48 Tabellen, 92 Beispielen, 109 Aufgaben, 65 Kontrollfragen, 26 Referatsthemen. Fachbuchverl. Leipzig im Carl-Hanser-Verl., München, 2001.

[[Le98]] LeCun, Y. et al.: Efficient BackProp, In Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop. Springer-Verlag, Berlin, Heidelberg, 1998; S. 9–50.

[[LKX20]] Li, F.-F.; Krishna, R.; Xu, D.: Lecture 5: Convolutional Neural Networks, Verfügbar unter https://cs231n.github.io/convolutional-networks/, Zuletzt geprüft 22.11.2020.

[[LTZ12]] Liu, F. T.; Ting, K.; Zhou, Z.-H.: Isolation-Based Anomaly Detection. In ACM Transactions on Knowledge Discovery From Data - TKDD, 2012, Ausgabe 6; S. 1–39.

[[Ma14]] Marsland, S.: Machine Learning. An Algorithmic Perspective, Second Edition. CRC Press, Hoboken, 2014.

[[Ma18]] MachineLearningStories.com: Connectivity Based Outlier Detection and its implementation in R, Verfügbar unter http://machinelearningstories.blogspot.com/2018/09/connectivity-based-outlier-detection.html, Zuletzt geprüft 20.11.2020.

[[Mi06]] Mitchell, T.: The Discipline of Machine Learning. In CMU ML-06, 2006; S. 1–12.

[[MSA18]] Makridakis, S.; Spiliotis, E.; Assimakopoulos, V.: Statistical and Machine Learning forecasting methods: Concerns and ways forward. In PloS one, 2018, Ausgabe 13; S. 1–26.

[[Mu19]] Munir, M. et al.: DeepAnT: A Deep Learning Approach for Unsupervised Anomaly Detection in Time Series. In IEEE Access, 2019, Ausgabe 7; S. 1991–2005.

[[NC17]] N. Singh; C. Olinsky: Demystifying Numenta anomaly benchmark, In 2017 International Joint Conference on Neural Networks (IJCNN), 2017; S. 1570–1577.

[[ON15]] O'Shea, K.; Nash, R.: An Introduction to Convolutional Neural Networks. In ArXiv, 2015, Ausgabe abs/1511.08458; S. 1–11.

[[Pa15a]] Pavel Senin et al.: Time series anomaly discovery with grammar-based compression, In EDBT, 2015; S. 481–492.

[[Pa15b]] Pasini, A.: Artificial neural networks for small dataset analysis. In Journal of thoracic disease, 2015, Ausgabe 7; S. 953–960.

[[Pe11]] Pedregosa, F. et al.: Scikit-learn: Machine Learning in Python. In Journal of Machine Learning Research, 2011, Ausgabe 12; S. 2825–2830.

[[PTK19]] Papacharalampous, G.; Tyralis, H.; Koutsoyiannis, D.: Comparison of stochastic and machine learning methods for multi-step ahead forecasting of hydrological processes. In Stochastic Environmental Research and Risk Assessment, 2019, Ausgabe 33; S. 481–514.

[[RD99]] R. Maclin; D. Opitz: Popular Ensemble Methods: An Empirical Study. In Journal of Artifical Intelligence Research, 1999, Ausgabe 11; S. 169–198.

[[RDH19]] Rui Varandas.; Duarte Folgado.; Hugo Gamboa.: Evaluation of Spatial-Temporal Anomalies in the Analysis of Human Movement, In Proceedings of the 12th International Joint Conference on Biomedical Engineering Systems and Technologies, 2019, Ausgabe 4: BIOSIGNALS; S. 163–170.

[[Sa94]] Sarle, W.: Neural Networks and Statistical Models, In Proceedings of the Nineteenth Annual SAS Users Group International Conference, 1994; S. 1–13.

[[SB14]] Sanchez, H.; Bustos, B.: Anomaly Detection in Streaming Time Series Based on Bounding Boxes. In (Traina, A. J. M.; Traina, C.; Cordeiro, R. L. F. Hrsg.), In Similarity Search and Applications. Springer International Publishing, Cham, 2014, Ausgabe 8821; S. 201–213.

[[Sc99]] Schölkopf, B. et al.: Support Vector Method for Novelty Detection. In (Solla, S. A.; Leen, T. K.; Müller, K.-R. Hrsg.), In Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]. The MIT Press, 1999; S. 582–588.

[[Sh20]] Sheng, B. et al.: A comparison of different machine learning algorithms, types and placements of activity monitors for physical activity classification. In Measurement, 2020, Ausgabe 154.

[[Si16]] Simon, A. et al.: An Overview of Machine Learning and its Applications. In International Journal of Electrical Sciences & Engineering, 2016; S. 22–24.

[[Su19]] Su, Y. et al.: Robust Anomaly Detection for Multivariate Time Series through Stochastic Recurrent Neural Network, In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. Association for Computing Machinery, New York, NY, USA, 2019; S. 2828–2837.

[[Ta18]] Tatbul, N. et al.: Precision and Recall for Time Series. In CoRR, 2018, Ausgabe abs/1803.03639; S. 1–11.

[[VSD19]] Vasighizaker, A.; Sharma, A.; Dehzangi, A.: A novel one-class classification approach to accurately predict disease-gene association in acute myeloid leukemia cancer. In PloS one, 2019, Ausgabe 14; S. 1–12.

[[Wa16]] Wang, X. et al.: A Self-Learning and Online Algorithm for Time Series Anomaly Detection, with Application in CPU Manufacturing, In Proceedings of the 25th ACM International on Conference on Information and Knowledge Management. Association for Computing Machinery, New York, NY, USA, 2016; S. 1823–1832.

[[WYZ15]] Wang, Y.; Yao, H.; Zhao, S.: Auto-Encoder Based Dimensionality Reduction. In Neurocomputing, 2015, Ausgabe 184.

[[Xu18]] Xu, H. et al.: Unsupervised Anomaly Detection via Variational Auto-Encoder for Seasonal KPIs in Web Applications. In Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW ’18, 2018; S. 1–12.

[[Yi07]] Yingyi Bu et al.: WAT: Finding Top-K Discords in Time Series Database, In Proceedings of the Seventh SIAM International Conference on Data Mining, April 26-28, 2007, Minneapolis, Minnesota, USA. SIAM, 2007; S. 449–454.

[[Yu14]] Yufeng, Y. et al.: Time Series Outlier Detection Based on Sliding Window Prediction. In Mathematical Problems in Engineering, 2014, Ausgabe 2014/2; S. 1–14.

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne unerlaubte Hilfe Dritter verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen stammen, sind als solche kenntlich gemacht. Diese Arbeit lag in gleicher oder ähnlicher Weise noch keiner Prüfungsbehörde vor und wurde bisher noch nicht veröffentlicht.

_____ _____

Ort, Datum Unterschrift