

Forschungsprojekt
**Ausreißer-Erkennung in Zeitreihen
mittels Graphen-basierter Algorithmen**

im Studiengang Angewandte Informatik
der Fakultät Informationstechnik
Wintersemester 2020/2021

Bahar Uzun

764647

Jeremy Kielman

764097

Marcus Erz

762294

Abgabedatum: 28. Februar 2021

Prüferin: Prof. Dr. rer. nat. Gabriele Gühring

Kurzfassung

todo: Kurzfassung erstellen

Schlagwörter: Anomalie-Erkennung, Ausreißer-Erkennung, NetSimile, MIDAS, Perculation, Iso-Map Graphen-basierte Algorithmen, Zeitreihen

Inhaltsverzeichnis

Abbildungsverzeichnis	v
Tabellenverzeichnis	vi
Listings	vii
1 Einleitung	1
1.1 Problemstellung	1
1.2 Verwandte Arbeiten	2
2 Datensätze	4
2.1 Numenta Zeitreihen Daten	4
2.2 Netzwerk-Datensätze	4
3 Statische Algorithmen zur Ausreißer Erkennung	6
3.1 Transformation der Daten	6
3.2 IsoMap Basierter Algorithmus	7
3.2.1 Implementierung	8
3.2.2 Ausreißer-Erkennung in Zeitreihen	8
3.3 Perculation basierter Algorithmus	10
3.3.1 Implementierung	10
3.3.2 Ausreißerererkennung in Zeitreihen	12
4 Dynamische Algorithmen zur Ausreißer Erkennung	13
4.1 Umwandlung der Daten in ein Netzwerk	13
4.2 NetSimile	15
4.2.1 Grundlagen	15
4.2.2 Anwendung des Algorithmus auf Netzwerkdaten	16
4.2.3 Anwendung des Algorithmus auf Zeitreihen	20
4.3 MIDAS	25
4.3.1 Grundlagen	25
4.4 Ausreißer-Erkennung in Graphen	28
4.5 Ausreißer-Erkennung in Zeitreihen	30
5 Fazit und Ausblick	34
5.1 Fazit	34
5.2 Ausblick	35
A NetSimile	36

B Midas	44
C Isomap	46
D Percolation	48
Literaturverzeichnis	51

Abbildungsverzeichnis

2.1	Beispielzeitreihe Numenta [1]	4
3.1	Berechnung der Distanz zwischen zwei Punkten nach Anwendung des IsoMap Algorithmus [11]	7
3.2	Problem Übergänge	9
3.3	Ablauf Perculation basierter Algorithmus [3]	10
3.4	Vergleich Perculation Algorithmus mit Sliding Window Verfahren und ohne Sliding Window Verfahren	11
4.1	Umwandlung einer Zeitreihe in ein Netzwerk. Bei dem Netzwerk handelt es sich hierbei um ein Symbolbild.	13
4.2	Datensatz Midas. 1Spalte: Ursprungsknoten, 2Spalte: Zielknoten, 3Spalte: Abschnitt	14
4.3	Ausreißer Score Enron Datensatz	18
4.4	<i>todo: Sedanspot Caption</i>	18
4.5	Darstellung der Ausreißer in Heatmaps	19
4.6	Ausreißer Score DARPA	20
4.7	Vollständiger Graph mit 11 Knoten	21
4.8	Ausreißer Score der vollständigen Graphen mit gewichteten Kanten	22
4.9	Der Ausreißer-Score über die Zeit beim ENRON-Datensatz	28
4.10	Die Ausreißer des SEDANSPOT-Algorithmus	29
4.11	Der Ausreißer-Score über die Zeit beim DARPA-Datensatz	30
4.12	MIDAS Algorithmus angewandt auf Zeitreihe mit einer erhöhten Amplitude.	31
4.13	Ausreißer Erkennung in Zeitreihen MIDAS Algorithmus	32
4.14	Ausreißer Erkennung Zeitreihen MIDAS Algorithmus Fenstergröße 110	32
4.15	Ausreißer Erkennung Zeitreihen MIDAS-R	33
A.-2	Signaturvektoren der Zeitreihe mit vollständigen Graphen	42
A.-1	Ausreißer Score der vollständigen Graphen	43

Tabellenverzeichnis

3.1	IsoMap Performance	8
3.2	Perculation Time Series Performance	11
4.1	Netsimile Time Series Perfomance	23
4.2	Parameter NetSimile Zeitreihen	24
4.3	Übersicht über historische Ereignisse, die den Ausreißern zuzuordnen sind	29
5.1	Vergleich der Algorithmen	34

Listings

4.1 Gewichtung als neues Feature	16
1	

1 Einleitung

Im Rahmen der Forschungsprojekt werden verschiedene Algorithmen zur Ausreißer-Erkennung in Graphen erforscht und getestet. Nachfolgend soll die Motivation hinter dieser Thematik erläutert werden.

1.1 Problemstellung

Eine leistungsfähige Methode Zeitreihendaten abzubilden, um Korrelationen zwischen voneinander abhängigen Datenobjekten zu erfassen, ist die Abbildung von Zeitreihendaten als Graphen. Durch diese Datenrepräsentation können Ausreißer bzw. Anomalien, vor allem in Netzwerken, schneller erkannt werden.

In der Forschung gibt es bereits viele Algorithmen zur Ausreißererkennung, die auf statische, sowie dynamische Graphen anwendbar sind. Die Erkennung der Ausreißer erfolgt mit verschiedenen Ansätzen. So gibt es Algorithmen, die die Struktur der Graphen näher betrachten, aber ebenso Algorithmen, die sich auf die Anzahl der Verbindungen zwischen zwei Kanten bzw. Objekten fokussieren. Nimmt man dynamische Graphen zur Untersuchung, so spielen die strukturellen Veränderungen über die Zeit, sowie die plötzlich massiv zunehmende Aktivitäten zwischen Knoten- und Kantenpaaren eine große Rolle. Darüber hinaus gibt es Ansätze zur Ausreißererkennung in sequenziellen Daten.

Allerdings gibt bisher keine Forschung von graphen-basierten Algorithmen auf traditionellen Zeitreihendaten, wie Sensordaten, die über die Zeit gesammelt werden, sondern nur über sich über die Zeit ändernden Netzwerkstrukturen.

Im Rahmen des Forschungsprojekts werden daher, die graphen-basierten Algorithmen zur Erkennung von Ausreißern in Zeitreihendaten herangezogen, um erste Erkenntnisse über die Aussagefähigkeit der Ergebnisse treffen zu können. Bei einem erfolgreichen Einsatz der Algorithmen können die Anwendungsfälle auf die Bereiche Internet of Things, Autonomes Fahren, sowie die Erkennung von Krankheiten, wie Krebs erweitert werden. Dies macht das Thema der Ausreißererkennung mittels graphen-basierter Algorithmen zu einem außerordentlich aktuellen und wichtigen Forschungsgebiet, in dem die Vorteile einer graphen-basierten Struktur auf die Zeitreihen übertragen werden.

Das Ziel der vorliegenden Arbeit setzt sich aus den folgenden Teilzielen zusammen:

1. Die Ermittlung einer Möglichkeit zur Transformation einer Zeitreihe in einen Graphen.
2. Die Anwendung der graphen-basierten Algorithmen auf Zeitreihendaten.
3. Der Vergleich verschiedener Algorithmen und deren Eignung Ausreißer zu erkennen.

1.2 Verwandte Arbeiten

In diesem Abschnitt werden Ansätze zur Erkennung von Ausreißern in statischen und dynamischen Graphen vorgestellt, die im Rahmen des Forschungsprojekts zur Erreichung der Forschungsziele herangezogen werden.

Die **Ausreißer-Erkennung in statischen Graphen** kann hinsichtlich ihrer Ausreißerkategorien unterteilt werden. So ergibt sich die nachfolgende Taxonomie. *todo: In Paper von Iso und Percolation sind zwei Algorithmen. Es wird aber so geschrieben wenn nur ein Algorithmus in dem Paper vorgestellt wird. todo: Vorherige Related work auch einfügen. Macht auf jeden Fall Sinn*

Struktur-basierter Ansatz: Mithilfe der Darstellung von Knoten und Kanten in einem Ego-Netzwerk werden in [2] die zugehörigen Eigenschaften, wie die Anzahl der Knoten und Kanten, extrahiert. Im Anschluss identifiziert dieser Algorithmus diejenigen Knoten und Kanten, die sich strukturell stark vom restlichen Ego-Netzwerk unterscheiden.

Clustering-basierter Ansatz: In [13] werden zwei Knoten und die Überschneidung ihrer Nachbarknoten gegenübergestellt. So wird die Annahme getroffen, dass Knoten, die sehr wenige Nachbarn, im Vergleich zum Rest der Knoten in Ihrer Umgebung haben, Ausreißer sind.

IsoMap-basierter Ansatz: Durch die Dimensionsreduktion mithilfe des Algorithmus in [3] gehen Informationen über Ausreißer verloren. Bei dem Versuch der Rekonstruktion können diese Informationen nicht wiederhergestellt werden. Durch einen anschließenden Vergleich der extrahierten Informationen werden Ausreißer sichtbar.

Kanten-basierter Ansatz: In [3] werden aus einem Graphen schrittweise die Kanten mit den höchsten Gewichten entfernt. Dadurch werden Ausreißer vom Rest des Netzwerks separiert. Die Annahme ist, dass Ausreißer-knoten höhere Kantengewichte zu ihren Nachbarn haben. Der Algorithmus in [7] iteriert zufällig über das Netzwerk. Dabei wird festgehalten wie oft ein Knoten besucht wurde. Ausreißer-Knoten werden dabei besonders selten besucht und können somit identifiziert werden.

Die **Ausreißer-Erkennung in dynamischen Graphen** kann hinsichtlich ihres Inputs unterteilt werden. So ergibt sich die nachfolgende Taxonomie.

Erkennung auf Momentaufnahmen des Graphen in zeitlichen Abständen: Der Algorithmus in [4] vergleicht verschiedene strukturelle Merkmale zweier Momentaufnahmen eines Graphen miteinander, um die Ähnlichkeit zu bewerten. Der Ausreißer wird bei einer starken Veränderung des Graphen deklariert.

Erkennung mithilfe eines Datenflusses: Der Algorithmus in [5] vergleicht jede ankommende Kante, zum aktuellen Zeitpunkt, mit der Anzahl am bisherigen Vorkommen dieser Kante. Hierbei werden Mikrocluster entdeckt die anomale sind.

Um die graphen-basierten Algorithmen zur Erkennung von Ausreißern in Zeitreihendaten nutzen zu können, müssen diese Daten in Graphen umgewandelt werden. Hierbei werden Distanzmaße verwendet, um die Qualität der Ausreißer-Erkennung zu verbessern. Im Rahmen des Forschungsprojekt wird somit die **Nutzung von Distanzmaßen** essenziell.

Überblick an existierenden Distanzmaßen: Ein Vergleich der verschiedenen Distanzmaße ist in [6] zu finden. Diese werden für die Anwendung auf Wahrscheinlichkeitsdichtefunktionen evaluiert und gruppiert.

2 Datensätze

In dieser Ausarbeitung wurden mehrere Datensätze genutzt, um verschiedene Algorithmen auf ihre Leistungsfähigkeit zu testen. Nachfolgend werden, in diesem Kapitel, die dabei verwendeten Datensätze vorgestellt. In [Kap. 2.1](#) werden Zeitreihen-Datensätze und in [Kap. 2.2](#) werden Netzwerk-Datensätze erläutert.

2.1 Numenta Zeitreihen Daten

Der Numenta Datensatz besteht aus einer Reihe an künstlich erzeugten Zeitreihen, die unterschiedliche Arten von Ausreißern simulieren. Durch die Daten wird es möglich eine qualitative Aussage über die Fähigkeiten der Algorithmen zu treffen. Für die Tests auf multivariaten Zeitreihen wurden neue Zeitreihen erzeugt. Dabei wurde für die erste Dimension eine Zeitreihe der Numenta Gruppe verwendet. Für höhere Dimensionen wurde auf eine Zeitreihe ohne Ausreißer zurück gegriffen [\[1\]](#).

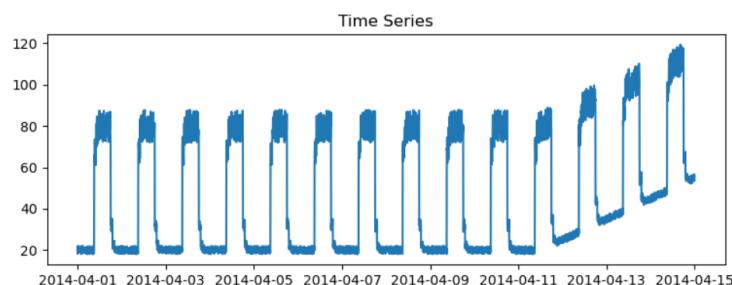


Abb. 2.1: Beispielzeitreihe Numenta [\[1\]](#)

2.2 Netzwerk-Datensätze

Zwischen den Forschungsgruppen, welche sich mit der Ausreißer-Erkennung in Netzwerken beschäftigen, herrscht große Konkurrenz. Aus diesem Grund werden gelabelte Datensätze häufig zurück gehalten. Dadurch wollen die Forscher verhindern, dass sie ihren Wettbewerbsvorteil gegenüber anderen verlieren [\[?\]](#). **todo: Paper finden und Quelle einfügen zu dieser Problematik** Aus diesem Grund ist es schwer geeignete Datensätze zu finden. Mit dem Enron und Darpa Datensatz konnten dennoch zwei passende Datensätze identifiziert werden. Diese Datensätze werden im Folgenden vorgestellt.

Enron

todo: Quelle einfügen? todo: Durchgelesen ob ich das sinngemäß wieder gegeben habe. Der Enron Datensatz enthält die intern versendeten E-Mail Daten von rund 150 Mitarbeitern der Firma Enron. Die Daten wurden von der Federal Energy Regulatory Commission offengelegt. Enthalten sind ca. 50.000 E-Mail-Nachrichten. Für den Algorithmus wird lediglich der Zeitpunkt, an dem eine E-Mail versendet wird, sowie die Sender und Empfänger festgehalten.

Für den Enron Datensatz stehen keine Labels zur Verfügung. Jedoch war es möglich Informationen über potenzielle Ausreißer aus einer anderen Quellen zu entnehmen. Dazu kann einerseits auf die Ausreißer zurück gegriffen werden, welche der SEDANSPOT-Algorithmus erkannt hat. Diese wurden in einem Schaubild dargestellt [?] todo: Quelle eingeben. Außerdem wird im SEDANSPOT-Paper eine ENRON Timeline dargestellt, diese enthält ebenfalls Informationen über mögliche Ausreißer [?] todo: Quelle einfügen.

DARPA

Der DARPA-Datensatz [9] beinhaltet 4.5 Millionen IP zu IP Kommunikationen zwischen 9.4 Tausend Quell-IP's und 23.3 Tausend Ziel-IP's über einen Zeitraum von 87.7 Tausend Minuten. Jede Kommunikation ist eine gerichtete Kante von der Quell-IP zur Ziel-IP in einem Zeitpunkt. Eine vierte Spalte des Datensatzes ist verfügbar, in der ein *label* enthalten bzw. ein Angriff gekennzeichnet ist. Der DARPA-Datensatz besteht zu über 60% aus Ausreißern. todo: Quelle zum Datensatz einfügen

3 Statische Algorithmen zur Ausreißer Erkennung

Die untersuchten Algorithmen können in statische und dynamische Algorithmen untergliedert werden. Für statische Algorithmen ist kennzeichnend, dass die Algorithmen keine Entwicklung im Verlauf der Zeit aufweisen. Gleiches gilt auch für die zugrunde liegenden Daten, diese bleiben während des Algorithmus unverändert. In diesem Kapitel werden zunächst zwei statische Algorithmen zur Ausreißer-Erkennung auf unterschiedlichen Datentypen (z.B. Videos, Bilder, Netzwerke) vorgestellt. Anschließend werden die Ergebnisse verschiedener Experimente mit den Algorithmen erläutert. Bei den Algorithmen handelt es sich um ein auf Percolation basierender Algorithmus (vgl. Kap. 3.3) und ein auf IsoMap basierender Algorithmus (vgl. Kap. 3.2). In Abschnitt Kap. 3.1 wird erklärt wie verschiedene Datentypen in ein Netzwerk umgewandelt werden können. Dieser Schritt ist als Vorverarbeitungsschritt der Daten erforderlich.

3.1 Transformation der Daten

Beim IsoMap- und Percolation-basierten Algorithmus handelt es sich um graphen-basierte Algorithmen. Das heißt sie können nur auf Netzwerkdaten angewandt werden. Aus diesem Grund müssen Daten zunächst in ein Netzwerk umgewandelt werden bevor die Algorithmen darauf angewandt werden können. Eine Voraussetzung für die Transformation ist, dass Distanzen zwischen einzelnen Elementen der Daten berechnet werden können [vgl. 3, S. 2]. Da während des Forschungsprojektes hauptsächlich mit Zeitreihen gearbeitet wurde, wird nachfolgend exemplarisch erläutert wie die Transformation einer Zeitreihe in ein Netzwerk funktioniert.

Für die Transformation einer Zeitreihe muss zunächst die Distanz zwischen den einzelnen Elementen (Zeitpunkten) der Zeitreihe berechnet werden. Hierzu wird das Distanzmaß aus Gl. 3.1 genutzt.

$$D_{ij} = \left(\sum_k |v_k^i - v_k^j|^p \right)^{1/p} \quad (3.1)$$

Insofern in die Gleichung für $p = 2$ eingesetzt wird, handelt es sich hierbei um die euklidische Distanz. Die mit Gl. 3.1 berechneten Distanzen bilden die Kantengewichte in dem neu erstellten Netzwerk. Dabei handelt es sich um ein vollständiges Netzwerk. Das heißt, jeder Knoten ist mit allen anderen Knoten über eine Kante verknüpft. Die Knoten des Netzwerks repräsentieren die einzelnen Elemente (Zeitpunkte) der Zeitreihe. [vgl. 3, S. 2]. Es können mit dieser Vorgehensweise ebenso multivariate Zeitreihen in ein Netzwerk transformiert werden.

3.2 IsoMap Basierter Algorithmus

Der Algorithmus verfolgt den Ansatz Informationen über Ausreißer durch die Reduzierung der Dimensionalität zu eliminieren. Beim anschließenden Versuch diese Informationen zu rekonstruieren, können durch den Vergleich mit der ursprünglichen Matrix Abweichungen bei den Ausreißer Elementen festgestellt werden. [vgl. 3, S. 3]. In Kap. 3.2 wird zunächst erklärt wie der IsoMap Algorithmus eine Reduzierung der Dimensionalität durchführt. Anschließend werden in Kap. 3.2 die zusätzlichen Schritte erläutert, welche notwendig sind um Ausreißer mithilfe des IsoMap Algorithmus zu erkennen.

IsoMap

Beim IsoMap handelt es sich um einen Algorithmus zur nichtlinearen Dimensionsreduktion. Zunächst werden beim IsoMap Algorithmus die Nachbarn eines jeden Punktes(Knoten) über den Ball Tree Algorithmus oder den KD Tree Algorithmus bestimmt. Anschließend wird jeder Punkt mit den gefundenen Nachbarn verknüpft, wodurch ein neuer Körper(Netzwerk) entsteht. Daraufhin wird eine neue Distanzmatrix auf dem entstandenen Körper berechnet, indem die kürzeste Distanz zwischen allen Punkten auf dem Körper berechnet wird. Diese Matrix kann ebenso als geodätische Distanzmatrix D_G bezeichnet werden. Die eigentliche Dimensionsreduktion wird anschließend über die Eigenvektoren und Eigenwerte der Matrix D_G durchgeführt. Das Ergebnis der Dimensionsreduktion ist eine neue Menge an Features für jedes Element $V^i = v_1^i \dots v_r^i$ des ursprünglichen Datensatzes. Durch das Erzeugen der Matrix D_G wird erreicht, das nichtlineare Zusammenhänge bei der Dimensionsreduktion erhalten bleiben. [vgl. 11, S. 3-4].

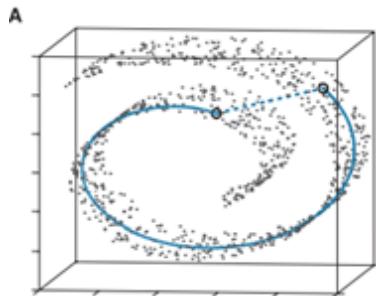


Abb. 3.1: Berechnung der Distanz zwischen zwei Punkten nach Anwendung des IsoMap Algorithmus [11]

IsoMap Algorithmus zur Erkennung von Ausreißern

Mithilfe des IsoMap Algorithmus wurden für jedes Element neue Features ($V^i = v_1^i \dots v_r^i$) berechnet. Im nächsten Schritt wird versucht aus diesen Eigenschaften die ursprüngliche Distanzmatrix zu rekonstruieren. Dazu wird aus den Eigenschaften V^i unter Verwendung von Gl. 3.1 eine neue Distanzmatrix \hat{D} berechnet. Nun können die Matrizen D_G und \hat{D} miteinander verglichen werden. Hierzu muss die Pearson Korrelation zwischen den jeweiligen Spalten der Matrizen berechnet werden. Für Ausreißer wird erwartet, dass die Korrelation sehr niedrig ist, da Informationen über sie bei der Reduktion verloren gehen [vgl. 3, S. 3]. Die Korrelation kann also als Ausreißer Score

genutzt werden. Um zu klassifizieren, ob es sich bei einem konkreten Element um einen Ausreißer handelt, wird zunächst der Mittelwert und die Standardabweichung des Ausreißer Score berechnet. Falls ein Element um einen bestimmten Threshold vom Mittelwert abweicht, wird das Element als Ausreißer klassifiziert.

3.2.1 Implementierung

Für den IsoMap Algorithmus stellt die Python Bibliothek 'scikit-learn' eine sehr gute Implementierung zur Verfügung [10]. Diese Implementierung könnten wir gut in unseren Algorithmus integrieren. Es musste lediglich geändert werden, dass auf die Matrix D_G zugegriffen werden kann. Dies ist standardmäßig nicht der Fall. Für die Implementierung der weiteren Funktionalität wurde auf die Python Bibliothek 'NumPy' zurückgegriffen.

3.2.2 Ausreißer-Erkennung in Zeitreihen

Für die durchgeföhrten Tests wurden die Zeitreihen aus Kap. 2.1 verwendet. Um zu bewerten wie gut der Algorithmus funktioniert, wurde ein Punktesystem eingeföhrt. In dem Punktesystem konnten maximal vier Sterne erreicht werden, was bedeutet, dass Ausreißer sehr gut erkannt worden sind. Null Sterne hingegen bedeuten, dass Ausreißer überhaupt nicht erkannt wurden. Der IsoMap Algorithmus liefert eher schwache Ergebnisse bei der Erkennung von Ausreißern in Zeitreihen. Das Hauptproblem hierbei ist, dass starke Anstiege, bei welchen es sich nicht um Ausreißer handelt, fälschlicherweise zu einem starken Anstieg des Ausreißer Score führen (vgl. Abb. 3.2 mit Pfeil markierte stellen). Dies kann, je nach Threshold, zu einer hohen Quote an falsch positiven Klassifizierungen führen. Aus diesem Grund können die tatsächlichen Ausreißer nicht eindeutig identifiziert werden. Eine Ähnliche Problematik trat in [12] bei der Verwendung des Random Walk Algorithmus auf. Das Problem konnte hierbei gelöst werden, indem vor der Anwendung des Algorithmus, eine Glättung der Zeitreihe durchgeführt wurde. Dadurch werden

Ausreißer Typ	Datei Name	1D
Einzelne Peaks (vgl. C.1c)	anomaly-art-daily-peaks	*
Zunahme an Rauschen (vgl. C.1b)	anomaly-art-daily-increase-noise	**
Signal Drift (vgl. C.1a)	anomaly-art-daily-drift	**
Kontinuierliche Zunahme der Amplitude (vgl. C.0e)	art-daily-amp-rise	**
Zyklus mit höherer Amplitude (vgl. C.0h)	art-daily-jumpsup	*
Zyklus mit geringerer Amplitude (vgl. C.0g)	art-daily-jumpsdown	**
Zyklus-Aussetzer (vgl. C.0f)	art-daily-flatmiddle	*
Signal-Aussetzer (vgl. C.0i)	art-daily-nojump	-
Frequenzänderung (vgl. C.1d)	anomaly-art-daily-sequence-change	-

Tab. 3.1: IsoMap Performance

abrupte Übergänge in der Zeitreihe abgemildert und deshalb nicht mehr als Ausreißer erkannt [vgl. 12, S. 31,36]. Dies könnte ein möglicher Ansatz sein um zukünftig bessere Ergebnisse erzielen zu können. Des Weiteren ist zu erkennen, dass der Algorithmus für einige Ausreißer Typen nicht geeignet ist, hierzu gehören Signal Aussetzer und Frequenzänderungen. Bei diesen Ausreißer Typen treten keinerlei unüblichen Werte auf, sondern lediglich Änderungen in der Saisonalität.

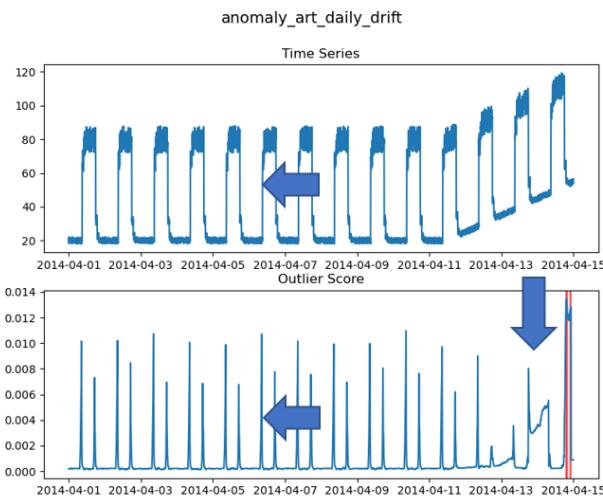


Abb. 3.2: Problem Übergänge

3.3 Perculation basierter Algorithmus

Bei diesem Algorithmus werden schrittweise die Kanten mit den höchsten Gewichten aus der mit Gl. 3.1 erzeugten Distanzmatrix D_{ij} entfernt. Ziel dieses Prozesses ist es Ausreißer vom restlichen Teil des Netzwerks zu trennen. Dabei kann davon ausgegangen werden, dass Ausreißer höhere Kantengewichte zu ihren Nachbarn aufweisen und deshalb schneller separiert werden. Sobald ein Knoten komplett separiert ist, wird diesem ein Ausreißer Score zugeordnet. Der Wert des Ausreißer Scores wird über die zuletzt entfernte Kante des Knoten definiert. Dadurch erhalten früher separierte Knoten höhere Ausreißer Scores als später separierte Knoten [vgl. 3, S. 3].

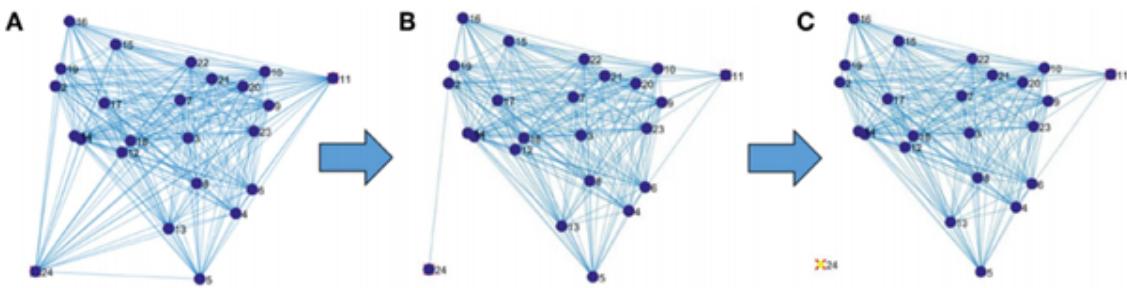


Abb. 3.3: Ablauf Perculation basierter Algorithmus [3]

3.3.1 Implementierung

Für die Implementierung des Perculation-basierten Algorithmus wurde genauso wie in Kap. 3.2.1, Python/Numpy verwendet. Bei der Implementierung eines Prototypen des Algorithmus konnte festgestellt werden, dass die Laufzeit des Algorithmus sehr langsam ist. Aus diesem Grund wurden einige Veränderungen an dem Algorithmus vorgenommen um die Performance zu verbessern. Dazu gehörte, dass nicht einzelne Kanten, sondern Gruppen an Kanten aus dem Netzwerk entfernt werden. Der Vorteil dieser Modifikation ist, dass seltener überprüft werden muss ob ein Knoten weiterhin mit den Rest des Netzwerks verbunden ist. Eine weitere Verbesserung, die eingeführt wurde, ist die Verankerung eines Abbruchkriteriums. Dabei wird der Algorithmus angehalten sobald eine bestimmte Menge an Kanten aus dem Netzwerk entfernt wurde. Da der Algorithmus nicht alle Berechnungen ausführen muss, kann damit eine Optimierung der Laufzeit erreicht werden. Weiterhin konnte festgestellt werden, dass diese Veränderung keinen Einfluss auf die Qualität der Ausreißer Erkennung hat, da Ausreißer lediglich zu Beginn des Algorithmus gefunden werden. Ein weiteres Problem des Ursprünglichen Algorithmus war, das bei aufeinanderfolgenden Elementen der Zeitreihe teilweise starke Schwankungen im Ausreißer Score auftraten (vgl. Abb. 3.4). Aus diesem Grund konnten Ausreißer, welche sich über mehrere Zeitpunkte hinweg erstrecken nicht komplett erkannt werden. Um die Schwankungen im Ausreißer Score abzumildern, wurde dieser geglättet. Dazu wurde der gleitende Mittelwert des Ausreißer Score berechnet. In Gl. 3.2 ist exemplarisch die Formel für einen gleitenden Mittelwert der Ordnung drei dargestellt. In Abb. 3.4 ist zu sehen wie sich der Ausreißer Score durch das Glätten verändert.

$$m_{\text{MA}}^{(3)}(t) = \frac{1}{3} (x(t-1) + x(t) + x(t+1)) \quad (3.2)$$

todo: Das Verfahren zur Glättung das hier eingesetzt wird heißt gleitender Mittelwert. Vielleicht Formel davon einfügen und dan noch passende Quelle finden todo: Parameter noch erklären

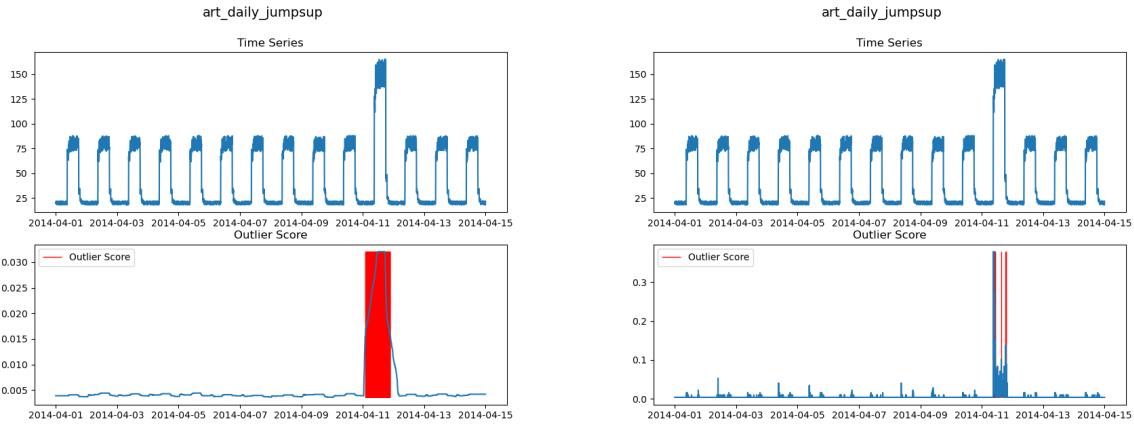


Abb. 3.4: Vergleich Perculation Algorithmus mit Sliding Window Verfahren und ohne Sliding Window Verfahren

Um zu klassifizieren, ob es sich bei einem konkreten Element um einen Ausreißer handelt, wird zunächst der Mittelwert und die Standardabweichung des Ausreißer Score berechnet. Falls ein Element um einen bestimmten Threshold vom Mittelwert abweicht, wird das Element als Ausreißer klassifiziert.

Ausreißer Typ	Datei Name	1D
Einzelne Peaks (vgl. D.1c)	anomaly-art-daily-peaks	*
Zunahme an Rauschen (vgl. D.1b)	anomaly-art-daily-increase-noise	****
Signal Drift (vgl. D.1a)	anomaly-art-daily-drift	***
Kontinuierliche Zunahme der Amplitude (vgl. D.0e)	art-daily-amp-rise	***
Zyklus mit höherer Amplitude (vgl. D.0h)	art-daily-jumpsup	****
Zyklus mit geringerer Amplitude (vgl. D.0g)	art-daily-jumpsdown	****
Zyklus-Aussetzer (vgl. D.0f)	art-daily-flatmiddle	****
Signal-Aussetzer (vgl. D.0i)	art-daily-nojump	-
Frequenzänderung (vgl. D.1d)	anomaly-art-daily-sequence-change	-

Tab. 3.2: Perculation Time Series Performance

3.3.2 Ausreißererkennung in Zeitreihen

Es konnte festgestellt werden, dass der Perculation basierte Algorithmus, viele Ausreißer Typen sehr gut erkennt. Ob Ausreißer in einer Zeitreihe mit einzelnen Peaks gefunden werden, hängt davon ab, ob der Ausreißer Score geglättet wird. Insofern keine Glättung des Ausreißer Scores durchgeführt wird, können einzelne Peaks gefunden werden. Denn durch die Glättung der Zeitreihe verschwinden die Ausschläge im Ausreißer Score. Es sollte somit in Abhängigkeit des Anwendungsfalles entschieden werden ob der Ausreißer Score geglättet wird. Dabei wäre ebenfalls denkbar, das beide Varianten zur Erkennung von Ausreißern verwendet werden. Der Perculation basierte Algorithmus ist genauso wie der Iso Map basierte Algorithmus (vgl. Kap. 3.2.2) nicht dazu im Stande Ausreißer in Zeitreihen mit Signal Aussetzer und Frequenzänderung zu erkennen.

4 Dynamische Algorithmen zur Ausreißer Erkennung

In diesem Kapitel werden zwei Algorithmen zur dynamischen Erkennung von Ausreißern vorgestellt. Hierbei handelt es sich um den NetSimile (vgl. Kap. 4.2) und den MIDAS (vgl. Kap. 4.3) Algorithmus. Dynamische Algorithmen können im Gegensatz zu statischen Algorithmen, Ausreißer in Echtzeitdaten finden. Dies kann in der Praxis sehr wichtig sein, da Ausreißer möglichst schnell gefunden werden müssen um finanzielle Schäden abzuwenden. Die dynamischen Algorithmen wurden genauso wie die statischen Algorithmen, von uns so gestaltet, das sie mit unterschiedlichen Daten Typen umgehen können. In unseren Experimenten wurden die Algorithmen auf Netzwerk- und Zeitreihen Daten angewandt.

4.1 Umwandlung der Daten in ein Netzwerk

Dieser Schritt muss durchgeführt werden bevor die Algorithmen angewandt werden können. Dabei funktioniert die Umwandlung der Daten genauso wie für statische Algorithmen (vgl. ??). Einziger Unterschied hierbei ist, das jeweils kleine Abschnitte der Daten in Netzwerke umgewandelt werden. Um dies zu veranschaulichen ein kurzes Beispiel: Ein Temperatur Sensor liefert jede Sekunde einen Wert. Sobald 100 Werte des Sensors eingegangen sind erfolgt die Umwandlung dieser Daten in ein Netzwerk unter Verwendung von Gl. 3.1. Dieser Vorgang wiederholt sich anschließend immer wieder. Der Wert für die Länge der Abschnitte ist hierbei frei wählbar und kann als Parameter übergeben werden. Insofern die Zeitreihe eine Saisonalität besitzt, bietet es sich an diese für die Länge der Abschnitte zu verwenden. In einem letzten Schritt werden anschließend die Netzwerkdaten in eine Datei geschrieben. Dieser Schritt ist aufgrund der Art und Weise, wie die Algorithmen implementiert sind notwendig. In ?? ist graphisch dargestellt wie die Umwandlung der Daten in ein Netzwerk funktioniert.

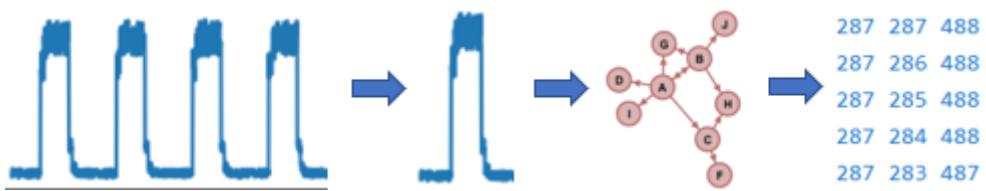


Abb. 4.1: Umwandlung einer Zeitreihe in ein Netzwerk. Bei dem Netzwerk handelt es sich hierbei um ein Symbolbild.

Die verschiedenen Algorithmen erfordern unterschiedliche Übergabeformate. Aus diesem Grund werden anschließend kurz die Besonderheiten erklärt, auf welche dabei geachtet werden muss.

NetSimile: Das Übergabeformat für den NetSimile Algorithmus ist in Abb. 4.1 ganz rechts dargestellt. Jede Zeile stellt hierbei eine Kante des Netzwerks dar. Bei der ersten Spalte handelt

es sich um den Ursprungsknoten der Kante, bei der zweiten Spalte um den Zielknoten und bei der letzten Spalte um die Gewichtung.

MIDAS: Beim MIDAS Algorithmus ist es nicht möglich die Gewichtung der Kanten direkt an den Algorithmus zu übergeben. Es ist jedoch möglich die Gewichtung der Kanten indirekt an den Algorithmus zu übergeben. Dazu wird die gleiche Kante mehrmals in Abhängigkeit der Gewichtung an den Algorithmus übergeben. In Abb. 4.2 ist ein kleiner Ausschnitt einer Datei für den MIDAS dargestellt.

```
248 259 7  
248 259 7  
248 259 7  
248 259 7
```

Abb. 4.2: Datensatz Midas. 1Spalte: Ursprungsknoten, 2Spalte: Zielknoten, 3Spalte: Abschnitt

MIDAS-R: Die Berechnungen für den MIDAS-R Algorithmus sind im Verhältnis zum MIDAS Algorithmus umfangreicher. Insofern für den MIDAS-R Algorithmus die gleichen Daten verwendet werden wie für den MIDAS Algorithmus, benötigen die Berechnungen sehr lange. Aus diesem Grund wurde eine Hauptkomponenten Zerlegung durchgeführt, um die Größe der Adjazenzmatrix zu verringert. Es entsteht ein kleineres Netzwerk, welches an den MIDAS-R Algorithmus übergeben werden kann.

todo: Midas R liefert eigentlich mehrere Ausreißer Scores es wäre vielleicht interessant diese einzeln zu betrachten und nicht zusammenaddiert.

4.2 NetSimile

4.2.1 Grundlagen

NetSimile ist ein skalierbarer Algorithmus zur Erkennung von Ähnlichkeiten, sowie Anomalien in Netzwerken unterschiedlicher Größen. Hierfür wird der Datensatz in gleich große Zeitintervalle unterteilt, um die daraus resultierenden Graphen auf unterschiedliche Merkmale zu untersuchen. Die Merkmale sind hierbei strukturelle Eigenschaften der einzelnen Knoten wie bspw. die Dichte eines Knotens oder die Anzahl an Nachbarn in einem Ego-Netzwerk. Die Signatur ergibt sich aus den einzelnen Aggregationen der Knoten wie bspw. der Median aus der Dichte der jeweiligen Knoten. So entsteht bspw. aus sieben Merkmalen und fünf Aggregationen ein Signaturvektor mit 35 verschiedenen Signaturen. So ermöglicht der Signaturvektor die Beschreibung sowie den Vergleich der einzelnen Graphen. Für den Vergleich wird die Canberra Distanz aus den beiden Signaturvektoren zweier zeitlich nebeneinander liegenden Graphen berechnet. [vgl. 4, S. 1]

Als Input für diesen Algorithmus wird eine Menge von k -anonymisierten Netzwerken mit beliebig unterschiedlichen Größen, die keine überlappenden Knoten oder Kanten besitzen sollten, herangezogen werden. Das Resultat sind Werte für die strukturelle Ähnlichkeit oder Abstands eines jeden Paares der gegebenen Netzwerke bzw. ein Merkmalsvektor für jedes Netzwerk. [vgl. 4, S. 1]

NetSimile durchläuft drei Schritte, die im Folgenden erläutert werden.

Extrahierung von Merkmalen

Für jeden Knoten i werden, basierend auf ihren Ego-Netzwerken, die folgenden Merkmale generiert:

$$\bar{d}_i = |N(i)|$$

Die Anzahl der Nachbarn (d.h. Grad) von Knoten i , wobei $N(i)$ die Nachbarn von Knoten i beschreibt.

$$\bar{c}_i$$

Der Clustering-Koeffizient von Knoten i , der als die Anzahl von Dreiecken, die mit Knoten i verbunden sind, über die Anzahl von verbundenen Dreiecken, die auf Knoten i zentriert sind, definiert ist.

$$\bar{d}_{N(i)}$$

Die durchschnittliche Anzahl der Nachbarn von Knoten i , die zwei Schritte entfernt sind. Dieser wird berechnet als **todo: Paper Seite 2 unten Formel einfügen**

$$\bar{c}_{N(i)}$$

Der durchschnittliche Clustering-Koeffizient von $N(i)$, der als **todo: Paper Seite 2 unten Formel einfügen** berechnet wird.

$$|E_{ego(i)}|$$

Die Anzahl der Kanten im Ego-Netzwerk vom Knoten i , wobei $ego(i)$ das Ego-Netzwerk von i zurückgibt.

$|E_{ego(i)}^{\circ}|$ Die Anzahl der von $ego(i)$ ausgehenden Kanten. $|N(ego(i))|$ Die Anzahl von Nachbarn von $ego(i)$.

Aggregierung von Merkmalen

Im nächsten Schritt wird für jeden Graphen G_j eine $Knoten \times Merkmal$ -Matrix F_{G_j} zusammengefasst. Dieser besteht aus den Merkmalsvektoren aus Schritt 1. Da der Vergleich von k -ten F_{G_j} sehr aufwändig ist, wird für jede F_{G_j} ein Signaturvektor \vec{s}_{G_j} ausgegeben. Dieser aggregiert den Median, den Mittelwert, die Standardabweichung, die Schiefe, sowie die Kurtosis der Merkmale aus der Matrix.

Vergleich der Signaturvektoren

Für die Ausreißererkennung werden die letzten drei Graphen anhand der Canberra-Distanz-Funktion, die als Ähnlichkeitsmaß dient, herangezogen. Steigt die Canberra Distanz zwischen zwei Graphen oberhalb des Thresholds so wird dies im Algorithmus festgehalten. Falls der darauf folgende Graph ebenfalls oberhalb des Thresholds liegt, so wird dieser als Ausreißer definiert. Dadurch wird die Anzahl der Ausreißer reduziert, damit nur diejenigen identifiziert werden, bei denen ein Trend hin zu einem abnormalen Verhalten erkennbar ist.

Der Algorithmus arbeitet dabei dynamisch, da die Signaturen der Graphen in einzelne Teil-Berechnungen aufgesplittet und zwischengespeichert werden können, ohne das eine Neuberechnung notwendig ist. Der Threshold wird aus dem Median und dem Mean berechnet, welche ebenfalls Zwischengespeichert werden können und nach Bedarf um weitere Graphen ergänzt werden können.

4.2.2 Anwendung des Algorithmus auf Netzwerksdaten

Beim ersten Versuch den Algorithmus auf Netzwerksdaten anzuwenden, wurde folgende Probleme festgestellt:

Der Algorithmus verwendet eine Bibliothek *igraph*, welche Kanten zwischen zwei Knoten nur einmalig hinzufügen kann. Beim Eliminieren der Duplikate würde aber ein Drittel des Datensatzes nicht berücksichtigt werden, wodurch wertvolle Informationen bei der Ausreißererkennung verloren gehen würden. Aus diesem Grund wurden die Netzwerksdaten soweit angepasst, dass Mehrfachverbindungen zwischen zwei Knoten aufsummiert werden und als Gewichtung dieser Kante hinzugefügt wird.

```

1  for i in range(len(e_list)):
2      g.add_edge(e_list[i][0], e_list[i][1], weight=e_list[i][2])

```

List. 4.1: Gewichtung als neues Feature

Dadurch kann der Datensatz zum einen vollständig analysiert werden und zum anderen konnte dadurch ein weiteres Feature hinzugefügt werden, dass durch die fünf verschiedenen Aggregationen den Signaturvektor um diese fünf Werte erweitert.

Dadurch das der Datensatz zuerst eingelesen und in einen Graphen transformiert wird und anschließend aus dem Graphen die jeweiligen Features extrahiert werden, verliert der Algorithmus extrem an Performanz. Des Weiteren wird im ersten Schritt der maximale Knoten-Wert als Größe des Graphen übergeben. Wird bspw. für jeden Mitarbeiter eine eigene ID übergeben und die inkrementell erhöht, so kann es sein, dass aus einem Netzwerk mit 20 verschiedenen Knoten ein Graph erzeugt wird, der 1000 Knoten erzeugt, weil eine ID mit dem Wert 1000 vorhanden ist. Dadurch büßt die Performanz an Geschwindigkeit ein, da Iterationen nicht über die 20 Knoten durchgeführt werden, sondern über 1000. Hierbei muss entweder der Datensatz vorab angepasst werden, indem die IDs neu vergeben werden oder der Algorithmus muss grundlegend neu aufgebaut werden.

Da der Fokus auf der Anwendung von Zeitreihen liegt, wird die Optimierung erst in diesem Abschnitt erläutert.

Das Problem hierbei ist, dass jeder Knoten eines Graphen die gleichen Features beinhaltet würde. Dadurch würden die Aggregationen überflüssig werden und der Signaturvektor auf sieben Features schrumpfen. Die Bildung von Cluster-Features wäre demnach nur noch bedingt möglich und die Betrachtung an Nachbarn, unabhängig ob im Ego-Netzwerk oder im gesamten Netzwerk würde sich die Gesamtzahl an Knoten annähern. Im Folgenden wird das Verhältnis der Features zum Durchschnitt dargestellt.

Anwendung auf ENRON-Datensatz

Da der Enron Datensatz ebenfalls von einem anderen Paper analysiert und veröffentlicht wurde, können die dort erkannten Ausreißer zum Vergleich in Form eines gelabelten Datensatz verwendet werden. Betrachtet man in diesem Kontext den Ausreißerscore, ist gut zu erkennen, dass der Ausreißer Ende 2001 als alleiniger herausstechender Ausreißer ebenso im Ergebnis wiederzufinden ist. Grundlegend ist ebenso zu erkennen, dass die Ausreißer sich nur sehr wenig voneinander unterscheiden, wodurch sich eine Klassifizierung innerhalb des Ausreißerscores als schwierig erweist. Die Extrahierung weiterer Features könnte hierbei eventuell aushelfen, wobei dies nicht im Rahmen dieses Forschungsprojektes behandelt werden soll, da der Fokus auf Zeitreihen liegt. Zwecks Performanz konnte der Datensatz innerhalb von zwei Minuten analysiert werden.

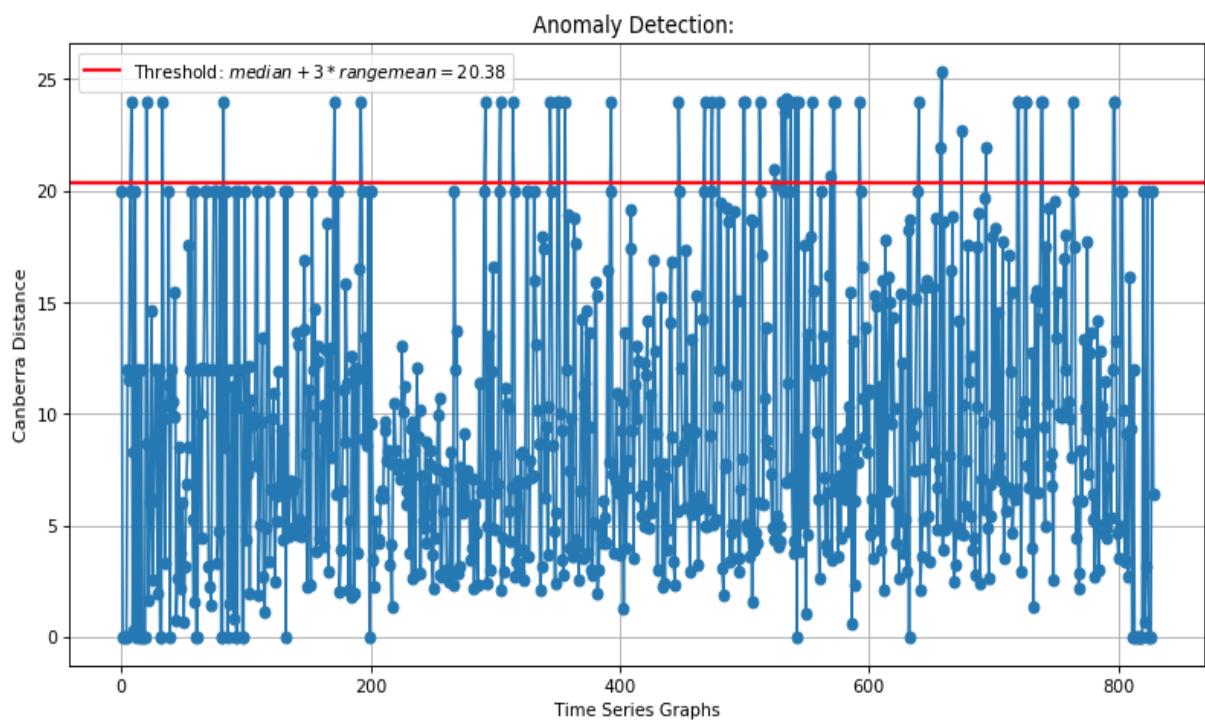


Abb. 4.3: Ausreißer Score Enron Datensatz

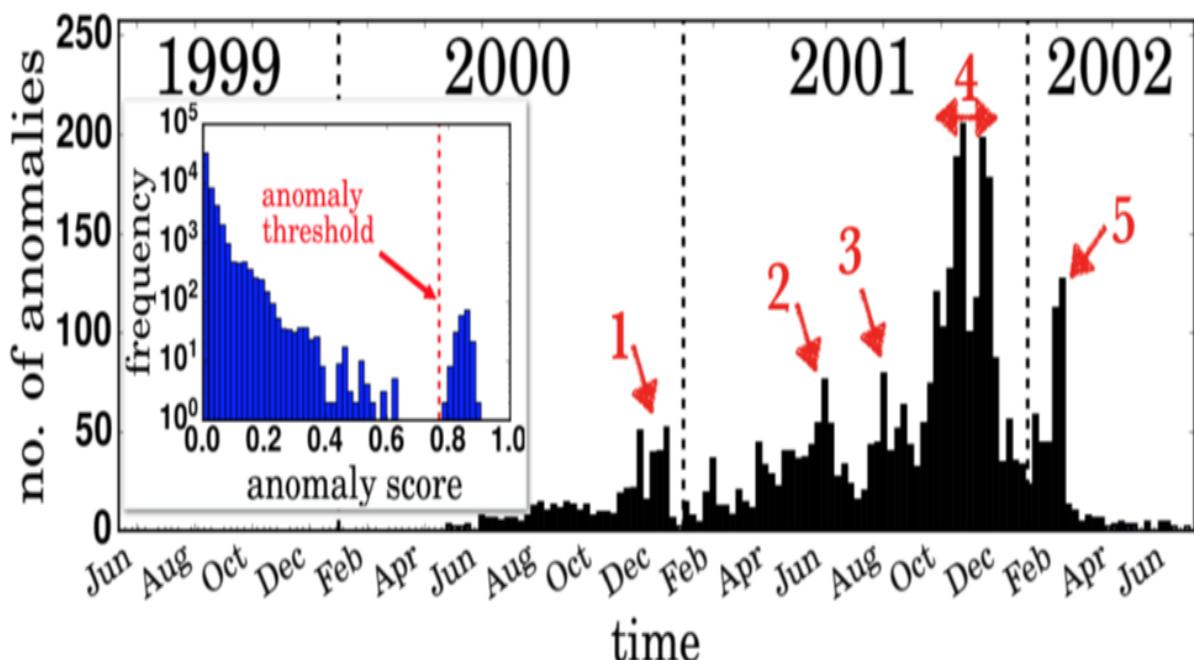


Abb. 4.4: todo: Sedanspot Caption

Betrachtet man die Differenz aus dem Durchschnitt der Signaturvektoren und den der Ausreißergraphen in einer Headmap kann man erkennen, dass die Ausreißer vorwiegend durch besonders große Ego-Netzwerke und einer hohen Anzahl an E-Mails verschuldet ist.

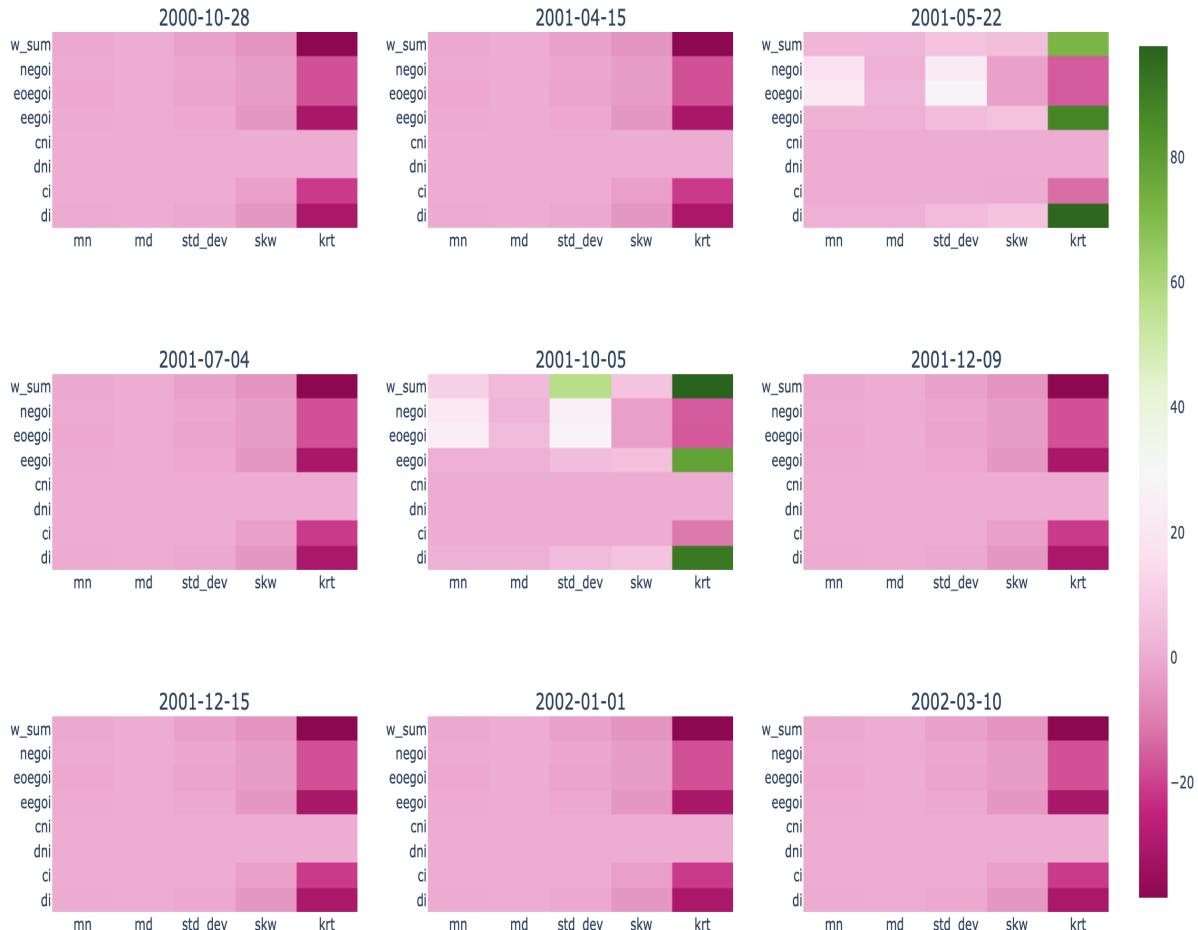


Abb. 4.5: Darstellung der Ausreißer in Heatmaps

Anwendung auf DARPA-Datensatz

Beim Darpa Datensatz können die Ausreißer besser klassifiziert werden. Die Gründe können hierbei auf die Größe und Vielfalt des Datensatzes zurückgeführt werden. Der Enron Datensatz hat eine Größe von 1MB und rund 50.000 Kanten. Der Darpa Datensatz hingegen hat eine Größe von 50 MB mit 4.5 Mio Kanten. Die Berechnung hat dabei eine Länge von 3h. Haben wir bei der Dateigröße den Faktor 50 und bei der Kantenanzahl den Faktor 90, so ist bei der Berechnungszeit der Faktor 90 wiederzufinden. Betrachtet man die Laufzeit, so kann eine lineare Abhängigkeit zwischen Kantenanzahl und der benötigten Berechnungszeit festgestellt werden.

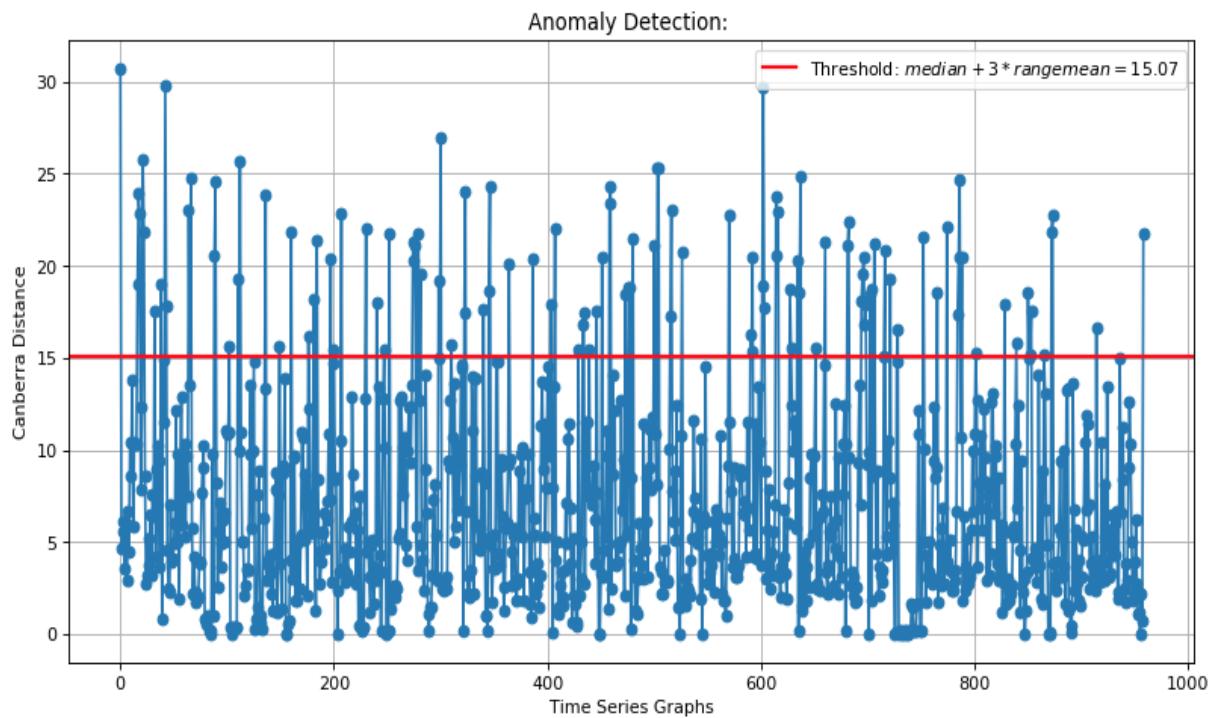


Abb. 4.6: Ausreißer Score DARPA

4.2.3 Anwendung des Algorithmus auf Zeitreihen

Wird der Algorithmus auf Zeitreihen anwendet, entsteht folgendes Problem. Bei der Transformation der Daten entstehen vollständige Graphen, wodurch die strukturellen Eigenschaften identisch werden, sowie die daraus resultierenden Merkmale, wie in der Abbildung verdeutlicht werden soll.

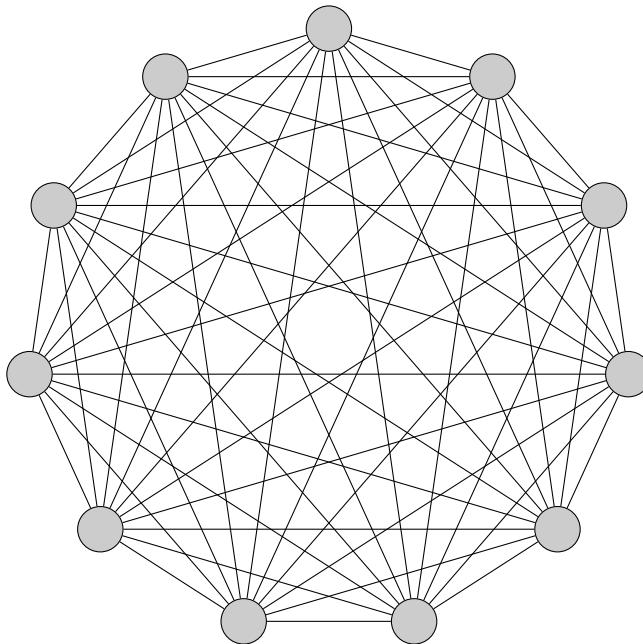


Abb. 4.7: Vollständiger Graph mit 11 Knoten

So hat bspw. das Feature $|E_{ego(i)}^o|$ keine Aussagekraft in einem vollständigen Graphen, da jeder Knoten die gleiche Anzahl Kanten in seinem Ego-Netzwerk aufweist. Subtrahiert man also vom durchschnittlichen Signaturvektor aller Graphen die einzelnen Signaturvektoren, so erkennt man den Wert 0 in allen Headmaps.

Somit müssen hierbei für diese Art von Graphen andere Features extrahiert werden. Außerdem ist die Laufzeit in großen Datensätzen, wie bspw. dem Darpa-Datensatz mit 3h Berechnungszeit nicht gerade performant.

Aus diesem Grund werden aus dem NetSimile lediglich die Ansätze der Feature Extrahierung übernommen, die Distanzbildung zweier Signaturvektoren, sowie der Threshold für die Ausreißeridentifizierung.

Das heißt die Netzwerke der Zeitreihe werden nicht in ein Graphen Objekt umgewandelt, sondern als Adjazenzmatrix gespeichert. Dadurch können die Features deutlich effizienter berechnet werden. Zudem werden lediglich Features verwendet, die für vollständige Graphen geeignet sind. Dabei werden folgende Features neu eingeführt:

todo: n-te Wurzel bei Formel für Geometrischen Mittelwert

$\sum_{i=1}^n x_i$
Summe der Kantengewichte eines Knoten.

$\frac{1}{n} \sum_{i=1}^n x_i$
Arithmetisches Mittel der Kantengewichte eines Knoten.

$\sqrt[n]{\prod_{i=1}^n x_i}$
Geometrisches Mittel der Kantengewichte eines Knoten.

$$x(p) = \begin{cases} \frac{1}{2}x(np) + & \text{if } n \text{ is even} \\ -(n+1)/2 & \text{if } n \text{ is odd} \end{cases}$$

Geometrisches Mittel der Kanten mit den 10% höchsten Kantengewichten.

$$\frac{1}{n} \sum_{i=1}^n x_i$$

Geometrisches Mittel der Kanten mit den 20% höchsten Kantengewichten.

Von diesen Features wurde dann auch den Median, den Mittelwert, die Standardabweichung, die Schiefe, sowie die Kurtosis berechnet. Dadurch konnten erste Ausreißer in der Zeitreihe gefunden werden (vgl. Abb. 4.8).

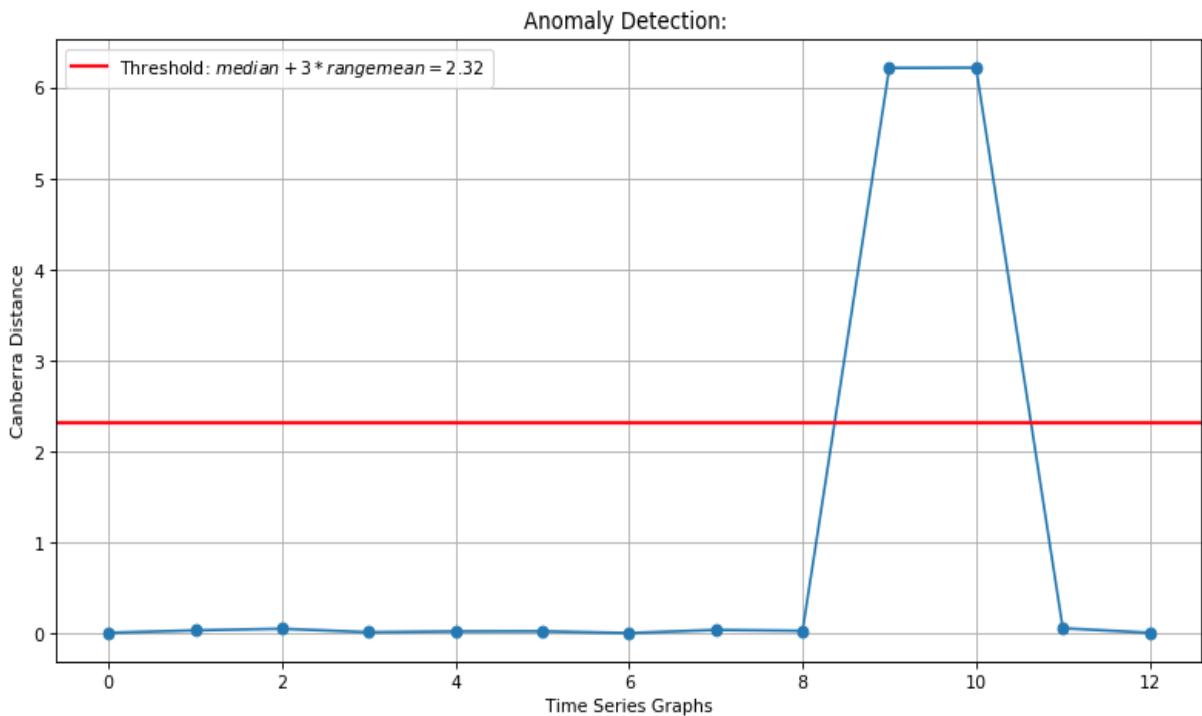


Abb. 4.8: Ausreißer Score der vollständigen Graphen mit gewichteten Kanten

todo: Bin mir nicht sicher zu welchen Elementen die Canbarra Distanz berechnet wird.. Des Weiteren wurde ein neuer Parameter eingeführt. Über diesen kann gesteuert werden zu wie vielen vorgänger Abschnitten die Distanz berechnet werden soll. Dadurch kann gesteuert werden wie schnell ein Algorithmus vergisst. Eine Auflistung der Parameter des Algorithmus ist in Tab. 4.2 zu sehen.

Um zu untersuchen, wie gut der Algorithmus funktioniert, wurde er auf Zeitreihen getestet. Als Testdaten wurden, ein und zweidimensionale Zeitreihen der Numenta Gruppe verwendet. Diese Zeitreihen enthalten verschiedene Ausreißer Typen, auf deren Erkennung der Algorithmus getestet wurde. Die Qualität der Ausreißerkennung wurde mithilfe eines Punktesystem bewertet. Dabei bedeuteten 0 Punkte, Ausreißer nicht erkannt und 4 Punkte bedeuteten Ausreißer sehr gut erkannt. Die Parameter, welche für die Tests gewählt werden mussten, werden in Tab. 4.2 beschrieben.

Ausreißer Typ	Datei Name	1D	2D
Einzelne Peaks	anomaly-art-daily-peaks	**	-
Zunahme an Rauschen	anomaly-art-daily-increase-noise	****	***
Signal Drift	anomaly-art-daily-drift	**	-
Kontinuierliche Zunahme der Amplitude	art-daily-amp-rise	****	***
Zyklus mit höherer Amplitude	art-daily-jumpsup	****	*
Zyklus mit geringerer Amplitude	art-daily-jumpsdown	****	-
Zyklus-Aussetzer	art-daily-flatmiddle	****	***
Signal-Aussetzer	art-daily-nojump	****	***
Frequenzänderung	anomaly-art-daily-sequence-change	****	***

Tab. 4.1: Netsimile Time Series Perfomance

Tab. 4.1 zeigt die Ergebnisse der Tests. Es ist zu erkennen, dass die Qualität der Ausreißer-Erkennung im eindimensionalen Fall sehr gut ist. Lediglich einzelne Peaks können durch den Algorithmus nicht als Ausreißer identifiziert werden. Außerdem wird bei Signal Drifts und der kontinuierlichen Zunahme der Amplitude lediglich der Anfang des Ausreißers detektiert. Aus diesem Grund wurde eine Bewertung mit drei Sternen vergeben. Bei der Betrachtung der Grapheken in ?? und Kap. A ist zu erkennen, dass das sechste oder siebte Intervall der Zeitreihe häufig als Ausreißer markiert wird. Der Grund hierfür ist, dass bei einer Fenstergröße von fünf für die ersten fünf Abschnitte kein Ausreißer Score berechnet wird. Dadurch ist die Standardabweichung zu Beginn sehr niedrig wodurch Abschnitte schnell als Ausreißer gekennzeichnet werden. Dieser Umstand wurde bei der Bewertung in Tab. 4.1 nicht berücksichtigt. Im zweidimensionalen Fall ist die Qualität der Ausreißer-Erkennung etwas durchwachsener. Auffallend ist, dass Zyklen mit höherer und niedriger Amplitude nicht als Ausreißer erkannt werden. Insbesondere ist dies auffällig, da diese Ausreißer Typen üblicherweise zuverlässig erkannt werden (vgl. ??). Außerdem ist der Algorithmus im zweidimensionalen Fall nicht mehr dazu in der Lage Signal Drifts zu erkennen. Andere Ausreißer Typen können durch den Algorithmus weiterhin erkannt werden, jedoch oftmals nicht mit der selben Qualität.

Parameter	Beschreibung
Periodizität	Wie in ?? todo: Referenz sollte glaube ich autoref -> sec:trsnsNeti sein erläutert muss die Zeitreihe in kleinere Intervalle aufgegliedert werden. Über diesen Parameter wird die Größe der Intervalle gesteuert. Für die Tests wurde der Parameter auf 288 gesetzt, da es sich hierbei um die Saisonalität der Zeitreihen handelt.
Fenstergröße	Wie in ?? erklärt, bestimmt dieser Parameter die Anzahl der vorangegangenen Abschnitte zu welchen die Canberra Distanz berechnet wird. Dieser Parameter wurde für die Tests auf 5 gesetzt.
Abweichung	Legt fest ab wann es sich bei einem Abschnitt um einen Ausreißer handelt. Der Parameter wurde für die Tests auf 3 gesetzt. Bedeutet wenn der Ausreißer Score um das dreifache der Standardabweichung vom Durchschnitt abweicht, wird der Abschnitt als Ausreißer gekennzeichnet.

Tab. 4.2: Parameter NetSimile Zeitreihen

todo: Die beiden Tabellen zusammenführen

todo: Die Bilder noch einordnen bspw. im Anhang. Im Text darauf verweisen.

Dadurch ist die Ausreißererkennung von Zeitreihen in Graphen nicht möglich. Fügt man die Gewichtung als weiteres Feature hinzu, wird hier eine erste Betrachtung der Ausreißer möglich. Der Graph 10 wird hier wie erhofft als Ausreißer identifiziert.

todo: bis hierhin gilt der letzte Kommentar

4.3 MIDAS

todo: In diesem Kapitel werden grundlegende Themen behandelt, die im Rahmen des Forschungsprojekts zum Verständnis der Ausreißer-Erkennung in Graphen gedient haben.

Erst erklären wie der MIDAS funktioniert. Und zum Laufen gebracht mit Graphen über die Zeit ENRON & DARPA. Im Anschluss auf Zeitreihendaten angewendet.

4.3.1 Grundlagen

todo: Einführung in den Algorithmus, NodeHash- sowie EdgeHash-Funktionen beschreiben

MIDAS, Eng. *Microcluster-Based Detector of Anomalies in Edge Streams*, steht für einen Algorithmus, der plötzlich auftretende Ausbrüche von Aktivitäten in einem Netzwerk bzw. Graphen erkennt. Dieses vermehrte Auftreten von Aktivitäten zeigt sich durch viele sich wiederholende Knoten- und Kantenpaare in einem sich zeitlich entwickelnden Graphen, die Mikrocluster bezeichnet werden. Mikrocluster bestehen demnach aus einem vermehrten Vorkommen eines einzigen Quell- und Zielpaars bzw. einer Kante (u,v) todo: Folgender Absatz kann vor der Beschreibung des Algorithmus eingefügt werden, wie im Paper auch Dies geschieht in Echtzeit, wobei jede Kante in konstanter Zeit und Speicher verarbeitet wird. In der Theorie garantiert er eine False-positive-Wahrscheinlichkeit und ist durch einen 162 bis 644 mal schnelleren Ansatz, sowie einer 42% bis 48% höhere Genauigkeit, im Hinblick auf die AUC, sehr effektiv. [vgl. 5, S. 1]

Anwendungsfälle für MIDAS sind die Erkennung von Anomalien in Computer-Netzwerken, wie SPAM oder DoS-Angriffe oder Anomalien in Kreditkartentransaktionen.

Count-Min-Sketch

Damit die relevanten Informationen für den Algorithmus mit einem konstanten Speicher verarbeitet werden, wird Count-Min-Sketch genutzt, dass eine Streaming-Datenstruktur mithilfe der Nutzung von Hash-Funktionen entspricht. Count-Min-Sketch zählt somit die Frequenz einer Aktivität bei Streaming-Daten. Diese Datenstruktur hat ebenfalls den Vorteil, dass man zu Beginn keine Kenntnis über die Anzahl an Quell- und Zielpaaren haben muss. [8]

MIDAS verwendet zwei Arten von CMS. Die erste Variante s_{uv} wird als die Anzahl an Kanten von u zu v bis zum aktuellen Zeitpunkt t definiert. Durch die CMS-Datenstruktur werden alle Zählungen von s_{uv} approximiert, sodass jederzeit eine annähernde Abfrage \hat{s}_{uv} erhalten werden kann. Die zweite Variante a_{uv} wird als die Anzahl an Kanten von u zu v im aktuellen Zeitpunkt t definiert. Dieser CMS ist identisch zu s_{uv} , wobei bei jedem Übergang zum nächsten Zeitpunkt die Datenstruktur zurückgesetzt wird. Dadurch resultiert aus dem CMS für den aktuellen Zeitpunkt die annähernde Abfrage \hat{a}_{uv} . [vgl. 5, S. 3]

Erkennung von Mikrocluster

Mithilfe der Näherungswerte \hat{s}_{uv} und \hat{a}_{uv} ist das Detektieren von Mikroclustern möglich. Hierzu wird der mittlere Pegel todo: andere übersetzung für mean level? (d.h. die durchschnittliche Rate mit der Kanten erscheinen) betrachtet. Es wird hierbei angenommen, dass dieser für den aktuellen Zeitpunkt (z.B. $t = 10$) äquivalent ist zu dem vor dem aktuellen Zeitpunkt ($t < 10$). Dadurch wird die Annahmen vermieden, dass die Daten auf einer bestimmten zugrundeliegenden Verteilung basieren oder Stationarität über die Zeit aufweisen.

Durch die genannte Annahme lassen sich vergangene Kanten in zwei Klassen einteilen. Eine für den aktuellen Zeitpunkt $t = 10$ und eine für alle vergangenen Zeitpunkte $t < 10$. Hierbei beträgt die Anzahl der Ereignisse zum Zeitpunkt $t = 10$ a_{uv} und die Anzahl der Kanten in vergangenen Zeitpunkten $t < 10$ ist $s_{uv} - a_{uv}$.

Die Auswertung der Daten kann mithilfe des chi-squared goodness-of-fit test erfolgen. Hierbei wird die Summe der Klassen $t = 10$ und $t < 10$ für $\frac{(\text{beobachtet} - \text{erwartet})^2}{\text{erwartet}}$ bestimmt. Bei einer Gesamtanzahl von s_{uv} Kanten ergibt sich, auf Basis eines mittleren Pegels todo: wie oben andere bezeichnung?, für $t = 10$ eine erwartete Anzahl von $\frac{s_{uv}}{t}$ Kanten todo: oder ereignisse?. Analog hierzu ergibt sich für $t < 10$ eine erwartete Anzahl an $\frac{t-1}{t}s_{uv}$ vergangenen Kanten. Daraus ergibt sich für die chi-squared Statistik [vgl. 5, S. 3]:

$$\begin{aligned}\chi^2 &= \frac{\left(\text{beobachtet}_{(t=10)} - \text{erwartet}_{(t=10)}\right)^2}{\text{erwartet}_{(t=10)}} \\ &+ \frac{\left(\text{beobachtet}_{(t<10)} - \text{erwartet}_{(t<10)}\right)^2}{\text{erwartet}_{(t<10)}} \\ &= \frac{\left(a_{uv} - \frac{s_{uv}}{t}\right)^2}{\frac{s_{uv}}{t}} + \frac{\left((s_{uv} - a_{uv}) - \frac{t-1}{t}s_{uv}\right)^2}{\frac{t-1}{t}s_{uv}} \\ &= \frac{\left(a_{uv} - \frac{s_{uv}}{t}\right)^2}{\frac{s_{uv}}{t}} + \frac{\left(a_{uv} - \frac{s_{uv}}{t}\right)^2}{\frac{t-1}{t}s_{uv}} \\ &= \left(a_{uv} - \frac{s_{uv}}{t}\right)^2 \frac{t^2}{s_{uv}(t-1)}\end{aligned}\tag{4.1}$$

Die Größen a_{uv} und s_{uv} können, mithilfe der CMS-Datenstruktur, approximiert werden. Daraus ergibt sich, unter Verwendung der approximierten Größen \hat{a}_{uv} und \hat{s}_{uv} , der folgende Anomaly Score [vgl. 5, S. 4]:

$$score((u,v,t)) = \left(\hat{a}_{uv} - \frac{\hat{s}_{uv}}{t}\right)^2 \frac{t^2}{\hat{s}_{uv}(t-1)}\tag{4.2}$$

Mithilfe des in Gl. 4.2 angegeben Anomaly Score lässt sich eine neue Kante (u,v) zum Zeitpunkt

t bewerten. Dieser wird in einem binären Entscheidungsverfahren verwendet, um zu bestimmen, ob es sich bei einer neuen Kante um Anomalie handelt oder nicht. Die Wahrscheinlichkeit von false positive Ergebnissen soll hierbei nicht einen benutzerdefinierten Schwellenwert ϵ übersteigen. CMS-Datenstrukturen mit einer angemessenen Größe besitzen die Eigenschaft, dass die Approximationen \hat{a}_{uv} , für beliebige ϵ und ν , folgende Vorschrift mit einer Wahrscheinlichkeit von mindestens $1 - \frac{\epsilon}{2}$ erfüllen:

$$\hat{a}_{uv} \leq a_{uv} + \nu \cdot N_t \quad (4.3)$$

N_t beschreibt hierbei die Anzahl an Kanten zum Zeitpunkt t . Eine weitere Eigenschaft der CMS-Datenstrukturen ist, dass diese die tatsächlichen Anzahl an Kanten nur überbewerten können:

$$s_{uv} \leq \hat{s}_{uv} \quad (4.4)$$

Der in Gl. 4.2 gegebene Score kann wie folgt angepasst werden:

$$\tilde{a}_{uv} = \hat{a}_{uv} - \nu N_t \quad (4.5)$$

Daraus lässt sich die in Gl. 4.1 gegebene Statistik anpassen:

$$\tilde{\chi}^2 = \left(\tilde{a}_{uv} - \frac{s_{uv}}{t} \right)^2 \frac{t^2}{s_{uv}(t-1)} \quad (4.6)$$

Bei Verwendung der Teststatistik in Gl. 4.6 und eines Schwellenwertes von $\chi^2_{1-\frac{\epsilon}{2}}(1)$ ergibt sich eine Wahrscheinlichkeit für ein false positive Ergebnis von höchstens ϵ :

$$P \left(\tilde{\chi}^2 > \chi^2_{1-\frac{\epsilon}{2}}(1) \right) < \epsilon \quad (4.7)$$

Der Term $\chi^2_{1-\frac{\epsilon}{2}}(1)$ beschreibt hierbei das $1 - \frac{\epsilon}{2}$ -Quantil.

MIDAS-R

Bei dem MIDAS-R Algorithmus handelt es sich um eine Erweiterung des MIDAS Algorithmus. Das R steht hierbei für den Relationalen Ansatz des MIDAS-R Algorithmus. Dabei wird versucht die räumliche oder zeitliche Verknüpfung zwischen Kanten stärker zu berücksichtigen. Es werden hierzu zwei neue Konzepte eingeführt [vgl. 5, S. 4].

Temporal Relations: Durch diesen Ansatz soll der Algorithmus mehr zeitliche Flexibilität erhalten. Dabei sollen Kanten aus der jüngsten Vergangenheit auch in einem neuen Zeitabschnitt berücksichtigt werden. Allerdings reduziert um eine bestimmte Gewichtung. Anstatt die CMS Datenstruktur nach jedem Zeitabschnitt zu reseten, werden die Gewichte hierbei um einen bestimmten Prozentsatz reduziert [vgl. 5, S. 4].

Spatial Relations: Hierbei werden zwei neue Features eingeführt um verschiedene Ausreißer-Typen identifizieren zu können. Die neuen Features werden hierbei in einer CMS Datenstrukturen gespeichert. Der Algorithmus speichert demzufolge diese drei Features:

Anzahl an Kanten zwischen Knoten u un Knoten v. Dieses Feature wird auch vom MIDAS Algorithmus verwendet.

Gesamtanzahl an Nachbarknoten eines Knoten u.

Aktuelle Anzahl an Nachbarknoten eines Knoten u.

Aus diesen drei Features wird anschließend ein Ausreißer Score abgeleitet. [vgl. 5, S. 5]

4.4 Ausreißer-Erkennung in Graphen

todo: ausformulieren

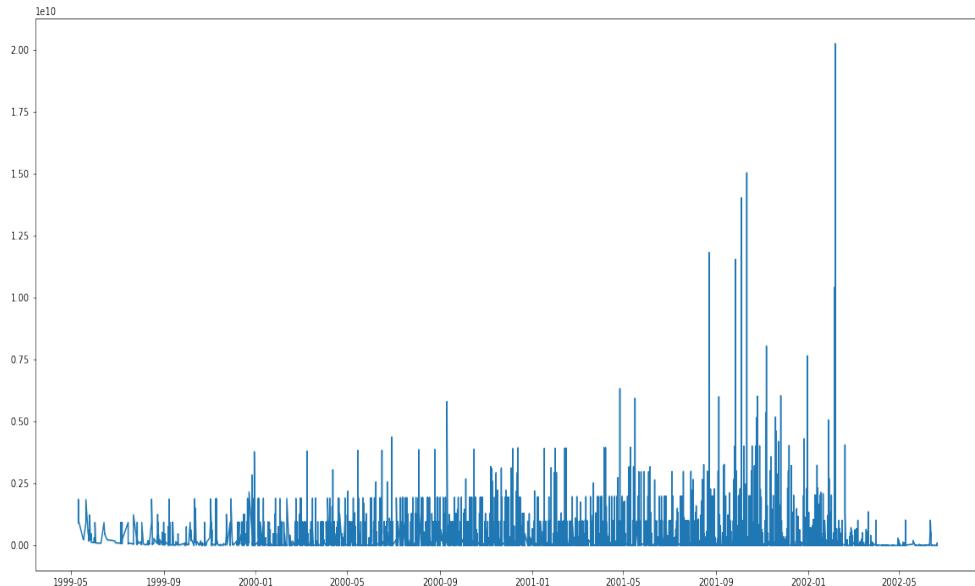


Abb. 4.9: Der Ausreißer-Score über die Zeit beim ENRON-Datensatz

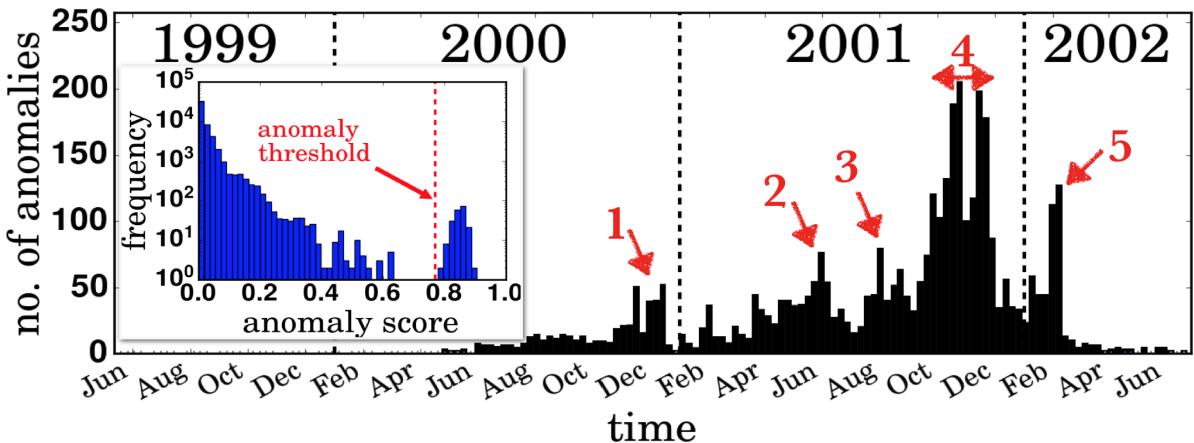


Abb. 4.10: Die Ausreißer des SEDANSPOT-Algorithmus

Die Herausforderung geeignete *labels* für die Datensätze zu finden wird in zwei Schritten eingedämmt.

Zum einen werden Erkenntnisse aus dem Schaubild des SEDANSPOT-Algorithmus gewonnen *todo: Quelle eingeben*. Hierbei kann man sehen, dass beide Algorithmen einen ähnlichen Verlauf vorweisen. Im nächsten Schritt wird die selbe Vorgehensweise wie aus dem SEDANSPOT-Paper gewählt und die ENRON Timeline *todo: Quelle einfügen wie bei Datensatz-Kapitel* zur Erhebung von möglichen Auswirkungen für die Ausreißer hinzugezogen.

Die Tab. 4.3 bietet eine Übersicht der historischen Ereignisse, die die Ausreißer des MIDAS-Algorithmus erklären. Im Vergleich zum SEDANSPOT-Algorithmus werden mehr Ausreißer erkannt.

1.	Aktie erreicht Allzeithoch. Federal Energy Regulatory Commission ordnet Untersuchung an.
2.	<ul style="list-style-type: none"> Vierteljährliche Telefonkonferenz zur Finanzsituation und erste Symptome eines Problems. „Geheimes“ Treffen – Schwarzenegger, Lay, Milken. Angebot zur Rettung der Deregulierung.
3.	<ul style="list-style-type: none"> Skilling (CEO) kündigt. Mitarbeiterin warnt Lay (Gründer) vor Pleite. Skilling verkauft seine Aktien. Enron veröffentlicht 618 Mio. \$ Verlust. Interessenskonflikt wird untersucht und Akten vernichtet.
4.	<ul style="list-style-type: none"> Beginn der Strafermittlung. Lay's Rücktritt Internen Ermittlung verteilt die Schuld auf Führungskräfte und den Vorstand

Tab. 4.3: Übersicht über historische Ereignisse, die den Ausreißern zuzuordnen sind

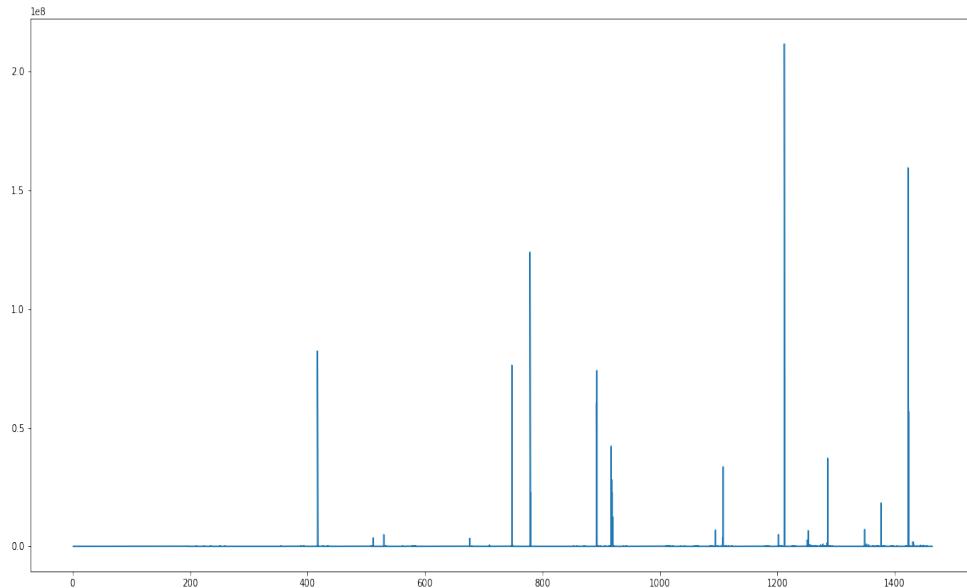


Abb. 4.11: Der Ausreißer-Score über die Zeit beim DARPA-Datensatz

Bei der Anwendung des MIDAS auf dem DARPA-Datensatz sieht man sehr schön einzelne Ausreißer, die entdeckt wurden. Für diesen Datensatz gibt es, speziell für MIDAS entwickelt, einen *ground truth*, der die *labels* für diesen Datensatz zur Verfügung stellt.

Bei der Berechnung der "Area under the Curve" für die ermittelten Ausreißer-Scores wird ein Wert von 0.9172724836793507 berechnet. Das bedeutet, dass der MIDAS-Algorithmus mit einer Wahrscheinlichkeit von ca. 91,73% die Kanten des Datensatzes richtig klassifiziert.

Somit kann festgehalten werden, dass MIDAS ein sehr guter Algorithmus ist bei der Erkennung von Ausreißern in Graphen und eine sehr hohe Genauigkeit erreicht.

todo: Schwierigkeit geeignete Datensätze zu finden, dazu gibt es ein Paper. Wenn man die Anomalyscores als gewichte nimmt, kommen Graphen in Networkx raus in denen man die anomalous nodes identifizieren kann dabei sollten es Edges sein

4.5 Ausreißer-Erkennung in Zeitreihen

todo: Tabelle wie für NetSimile einfügen bzgl. den verschiedenen Numenta-Datensätzen. Bisher nicht dringlich gewesen, da MIDAS schlecht ist und wir das für den abstract nicht benötigen

Um den MIDAS Algorithmus auf Zeitreihen anwenden zu können muss die Zeitreihe, wie in ?? beschrieben, zunächst in verschiedene Netzwerke umgewandelt werden. Bei den Tests konnte festgestellt werden, dass der MIDAS Algorithmus nicht dazu in der Lage ist Ausreißer in Zeitreihen

zu erkennen. Die vollständigen Ergebnisse der Tests können in Kap. B eingesehen werden. Hierbei ist jedoch der Verlauf des Ausreißer Scores schwierig zu interpretieren. Es ist zu erkennen, dass der Ausreißer-Score zu Beginn eines jeden Abschnitts sehr hoch ist, am Ende des Abschnitts ist der Ausreißer Score hingegen relativ niedrig. Grund hierfür ist, dass die Anzahl an Kanten zu Beginn eines Abschnittes im Verhältnis zu der Anzahl an Kanten aus den vorangegangenen Abschnitten deutlich niedriger ist. Im weiteren Verlauf werden weitere Kanten innerhalb des Abschnitts hinzugefügt. Dadurch gleicht sich die Anzahl an Kanten innerhalb der Abschnitte an und der Ausreißer Score sinkt.

Der MIDAS Algorithmus ist lediglich bei einer Zeitreihe dazu in der Lage den Ausreißer zu identifizieren. Hierbei handelt es sich um die Zeitreihe mit erhöhter Amplitude (vgl. Abb. 4.12). Durch den Ausschlag nach oben in der Zeitreihe entsteht ein Netzwerk, mit sehr hohen Gewichten. Die hohen Gewichte führen zu einer erhöhten Anzahl an Kanten, was schlussendlich zu einem Ausschlag des Ausreißer Scores führt. Die erhöhte Anzahl an Kanten führt ebenfalls dazu, dass der Abschnitt mit dem Ausreißer in der Abbildung deutlich breiter ist als die anderen. Bei anderen Ausreißer Typen sind die Differenzen zwischen den verschiedenen Elementen der Zeitreihe nicht so groß. Dadurch ergeben sich keinerlei hohe Kantengewichte und der Ausreißer kann nicht erkannt werden.

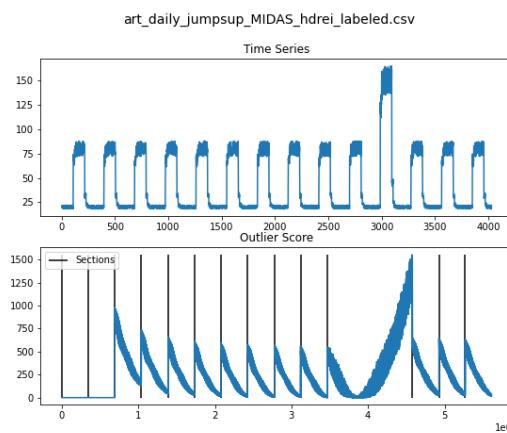


Abb. 4.12: MIDAS Algorithmus angewandt auf Zeitreihe mit einer erhöhten Amplitude.

Teilweise führen die Ausreißer ebenso zu besonders wenigen Kanten (vgl. 4.13a). Bei diesem Ausreißer Typ sind alle Werte auf der selben Ebene. Dadurch gehen die Kantengewichte gegen Null. Dies führt zu einem sehr kurzen Abschnitt in der Abbildung (Der Abschnitt wurde mit einem Pfeil markiert). Des Weiteren ergibt sich durch die Ausreißer eine leicht veränderte Anzahl an Kanten in dem Abschnitt mit dem Ausreißer (vgl. 4.13b). Die Abweichungen sind jedoch so gering, dass es nicht zu einem starken Anstieg des Ausreißer Score führt.

Es wurden außerdem Tests durchgeführt, um zu untersuchen, wie sich der Algorithmus bei veränderter Fenstergröße verhält (vgl. Abb. 4.14). Bei den Untersuchungen in Abb. 4.12 und Abb. 4.13 wurde eine Fenstergröße von 288 genutzt, was der Saisonalität der Zeitreihe entspricht. Für dieses Experiment wurde eine Fenstergröße von 110 verwendet. Es konnte festgestellt werden, dass diese Veränderung keinen zusätzlichen Nutzen bringt. Allerdings ist der Ausschlag nach oben im Ausreißer Score für die Zeitreihe mit erhöhter Amplitude noch deutlicher zu erkennen. Die anderen Ausreißer Typen werden weiterhin nicht erkannt.

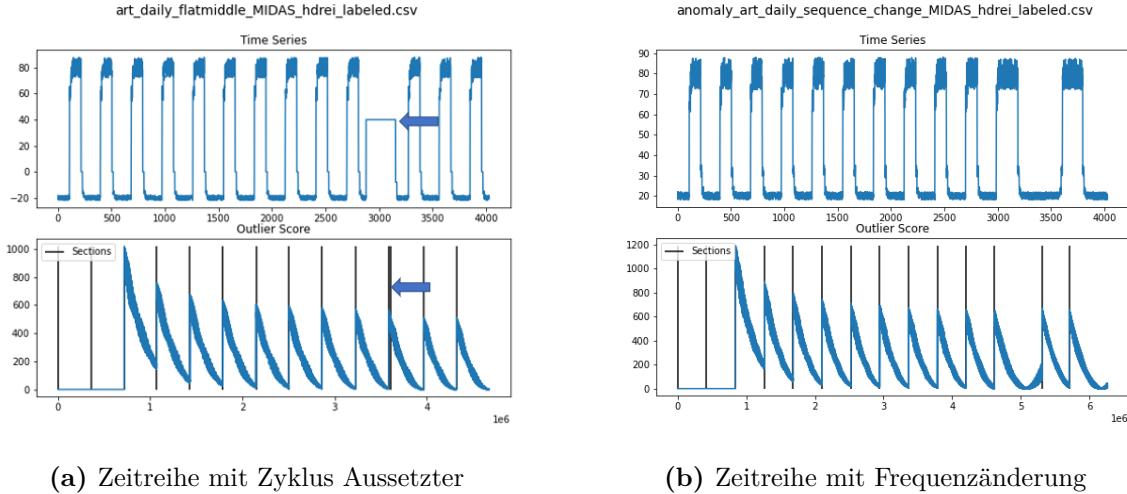


Abb. 4.13: Ausreißer Erkennung in Zeitreihen MIDAS Algorithmus

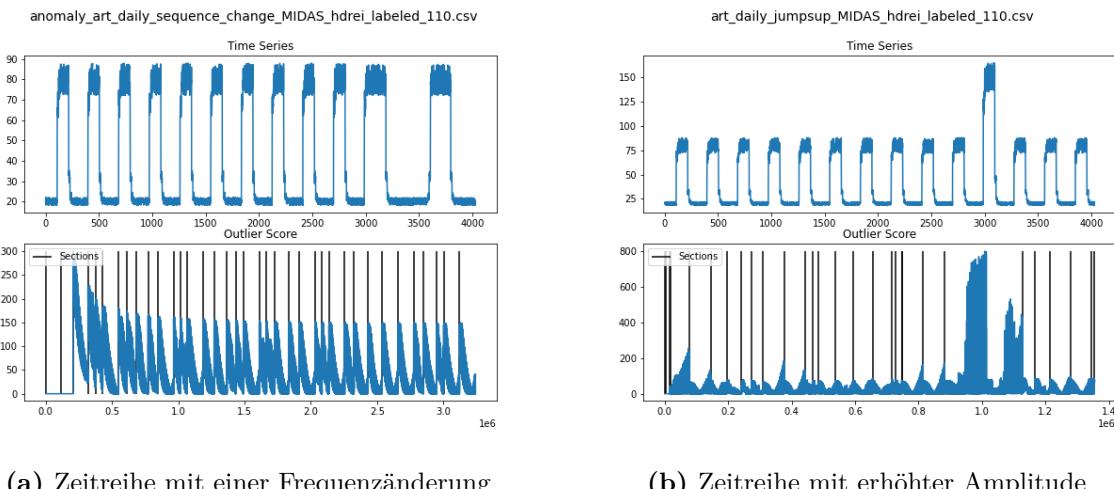


Abb. 4.14: Ausreißer Erkennung Zeitreihen MIDAS Algorithmus Fenstergröße 110

In einem nächsten Schritt wurde untersucht inwiefern der MIDAS-R Algorithmus zu einer Verbesserung bei der Ausreißer Erkennung beitragen kann (vgl. Abb. 4.15). Der MIDAS-R Algorithmus berücksichtigt bei der Berechnung des Ausreißer Scores für den aktuellen Abschnitt ebenso die Daten aus der jüngsten Vergangenheit(vorangegangene Abschnitte). Aus diesem Grund erhofften wir uns durch den Einsatz des MIDAS-R Algorithmus, dass die Ausschläge zu Beginn eines jeden Abschnitts ausbleiben, sodass Ausreißer deutlicher hervortreten. Es konnte festgestellt werden, dass der Ausschlag des Ausreißer Scores zu Beginn der Abschnitte deutlich kleiner ist. Jedoch steigt der Ausreißer Score zum Ende eines jeden Abschnitts wieder an. Es konnte somit keine Signifikante Verbesserung bei der Erkennung von Ausreißern erreicht werden. Insbesondere da der MIDAS-R Algorithmus ebenfalls nur den Ausreißer in der Zeitreihe mit erhöhter Amplitude anzeigt. Somit konnte festgestellt werden, dass die durch den MIDAS-R Algorithmus eingeführten Features ebenso zu keiner Verbesserung der Ergebnisse geführt haben.

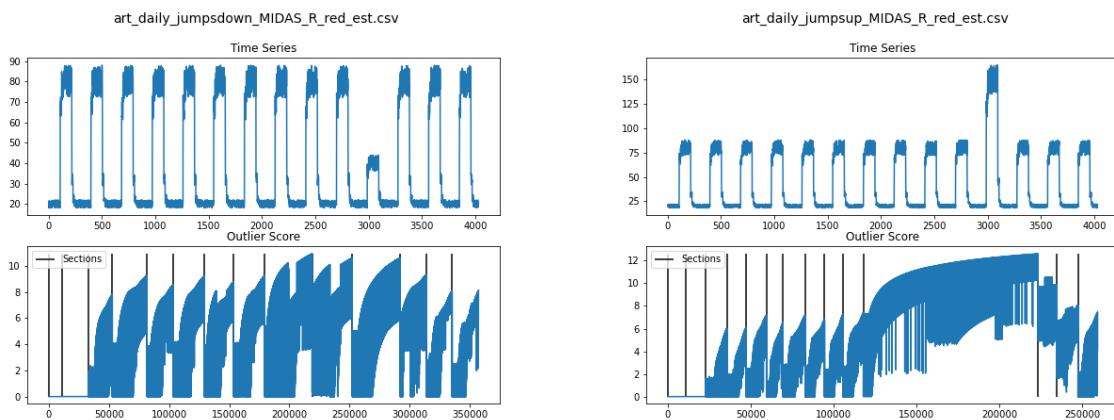


Abb. 4.15: Ausreißer Erkennung Zeitreihen MIDAS-R

5 Fazit und Ausblick

5.1 Fazit

Das dritte Teilziel ist es die verschiedenen graphen-basierten Algorithmen, die wir im Rahmen des fortgeführten Forschungsprojekt analysiert und evaluiert haben miteinander zu vergleichen. (vgl. Kap. 1.1) Durch den Vergleich soll ermittelt werden, wie erfolgreich ein Algorithmus bei der Erkennung von Ausreißern in Daten ist.

	Statisch	Dynamisch	Qualität Ausreißer- Erkennung Netzwerke	Qualität Ausreißer- Erkennung Zeitreihen	Performanz
IsoMap-based	+	-	-	o	+
Perculation-based	+	-	-	+	+
NetSimile	-	+	+	++	+
MIDAS	-	+	++	-	o

Tab. 5.1: Vergleich der Algorithmen

Aus der Tabelle Tab. 5.1 kann man die Kriterien entnehmen, die zum Vergleich der Algorithmen herangezogen wurden. So werden alle Algorithmen zunächst einmal in statisch und dynamisch entsprechend der Taxonomie des Forschungsprojekts unterteilt. Im nächsten Schritt werden die Ergebnisse aller Algorithmen hinsichtlich ihrer Fähigkeit Ausreißer in Graphen, sowie in Zeitreihen qualitativ zu erkennen gegenübergestellt. Zuletzt liegt der Fokus auf ihrer Performance im Vergleich zu den anderen Algorithmen. Nachfolgend werden die Ergebnisse ausgeführt.

todo: die Vergleiche autoref einfügen. Der IsoMap-based und der Perculation-based Algorithmus sind statische Algorithmen, die jeweils nur auf einem Graphen angewendet werden können. Aus diesem Grund eignen sich diese nicht für die Ausreißer-Erkennung in Netzwerken, da hierbei mehrere Graphen übergeben werden. Beide können jedoch erfolgreich auf Zeitreihen angewendet werden, wobei beim IsoMap-based Algorithmus die Ausreißer nur teilweise erkennbar sind, da sich die Ausreißer nur geringfügig von den anderen Werten unterscheiden. Der Perculation-based Algorithmus hingegen zeigt eindeutige Ausreißer, weshalb dieser Algorithmus geeignet ist für eine qualitative Ausreißer-Erkennung. Bei den Ausreißer Typen "Signal-Aussetzer" und "Frequenzänderung" können beide Algorithmen keine Ausreißer identifizieren. Die Performance beider Algorithmen ist gut, da sie die Berechnungen innerhalb weniger Sekunden durchführen.

Der NetSimile- und der MIDAS-Algorithmus können beide auf Netzwerkdatensätzen angewendet werden, da sie dynamisch sind. Der Unterschied hierbei liegt bei dem Ausreißer-Score. Beim NetSimile sind die erkannten Ausreißer nahezu identisch, wodurch eine Klassifizierung unzureichend ist. Beim MIDAS hingegen sind die Ausschläge weitaus größer, wodurch eine Klassifizierung erreicht wird. Bei der Anwendung auf Zeitreihen liefert der NetSimile sehr gute Ergebnisse auf den Ausreißer-Typen die wir verwendet haben (vgl. ??). Da sich bei der Transformation von Zeitreihen zu Graphen die Gewichtung nur geringfügig unterscheidet und der MIDAS-Algorithmus Ausreißer nur bei einer hohen Kantengewichtung erkennt, fallen die Ergebnisse hierbei schlechter aus (vgl. Kap. 4.5). Dementsprechend ist MIDAS für die Erkennung von Ausreißern in Zeitreihen eher ungeeignet. Bei der Performanz hat der NetSimile-Algorithmus ursprünglich bis zu einer Stunde benötigt, da dieser rechenintensive Bibliotheken verwendete. Durch die Verwendung anderer Bibliotheken wurde eine Optimierung erreicht. Die Visualisierung des Graphen ist hierdurch zwar nicht mehr möglich, jedoch wird die Rechenzeit auf wenige Sekunden reduziert. Der MIDAS benötigt hingegen mehrere Minuten, wodurch der NetSimile nach der Optimierung performanter geworden ist.

Der NetSimile-Algorithmus ist demnach ein Algorithmus, der, nach Optimierung, den Anforderungen des Forschungsprojekts gerecht wird. Dieser Algorithmus ist dynamisch, multidimensional, performant und die Erkennung von Ausreißern in Zeitreihen ist sehr gut. Der Schwerpunkt hierbei liegt bei der Wahl der richtigen Features. Strukturelle Merkmale sind eher ungeeignet bei vollständig verknüpften Graphen und sollten ersetzt werden durch Features, wie bspw. die Gewichtung der Kanten.

5.2 Ausblick

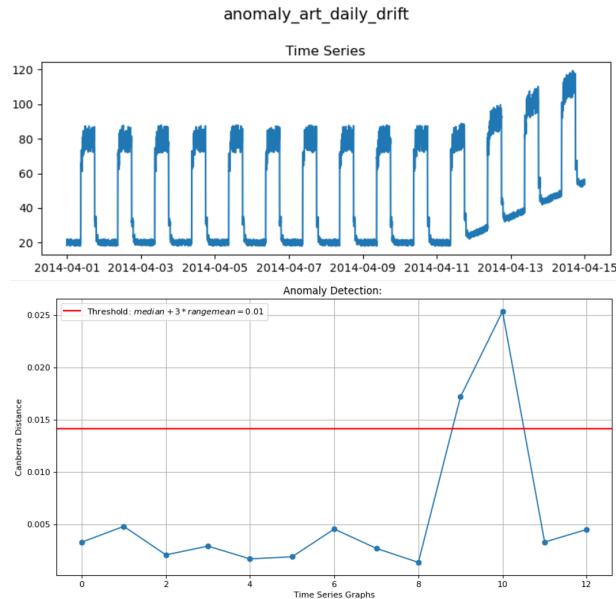
Im Rahmen des Forschungsprojekts wurden drei Thematiken behandelt. Zunächst einmal wurde eine Möglichkeit ermittelt Zeitreihendaten in einen Graphen umzuwandeln. In einem weiteren Schritt wurden graphen-basierte Algorithmen auf die transformierten Zeitreihendaten angewandt. Im Anschluss wurden diese Algorithmen hinsichtlich ihrer Eignung zur Ausreißer-Erkennung verglichen.

Durch die erfolgreiche Transformation von Zeitreihendaten in Graphen kann der Vorgang ebenso für andere Datenkategorien herangezogen werden, unter der Prämisse, dass Distanzen zwischen den einzelnen Elementen des Datensatzes gebildet werden können. So ist es im nächsten Schritt möglich bspw. Ausreißer in Finanzdaten oder Bilddaten zu erkennen.

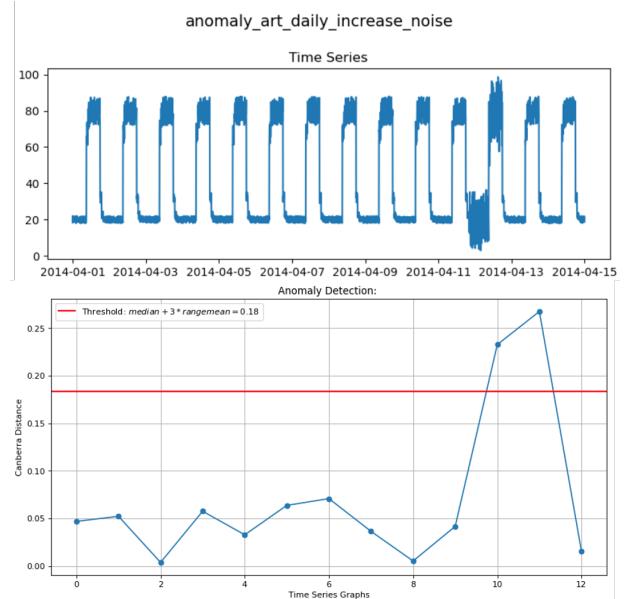
Der NetSimile-Algorithmus kann durch neue Merkmale ergänzt werden. So ist es zukünftig möglich neben den mathematischen Merkmalen auch statistische Algorithmen als Feature einzusetzen um eine erweiterte und optimierte Möglichkeit zu erhalten Aussagen hinsichtlich Ausreißer treffen zu können. Eine heutige Problematik des NetSimile ist zum einen die, dass er nur den Ausreißer-Graphen zurückgibt und zum anderen wird der Graph erst dann berechnet, wenn dieser alle Kanten eines Zeitintervalls beinhaltet. Hierbei können weitere Optimierungen folgen, die bspw. den Ausreißer-Graphen auf Knoten oder Kanten untersucht, die für den Ausreißer-Score am relevantesten sind. Zudem kann eine Methode ermittelt werden, die einen Graphen iterativ vergrößert, damit eventuell schon vor dem vollständigen Berechnen des Graphs bestimmt werden kann, ob es sich um einen Ausreißer handelt.

A NetSimile

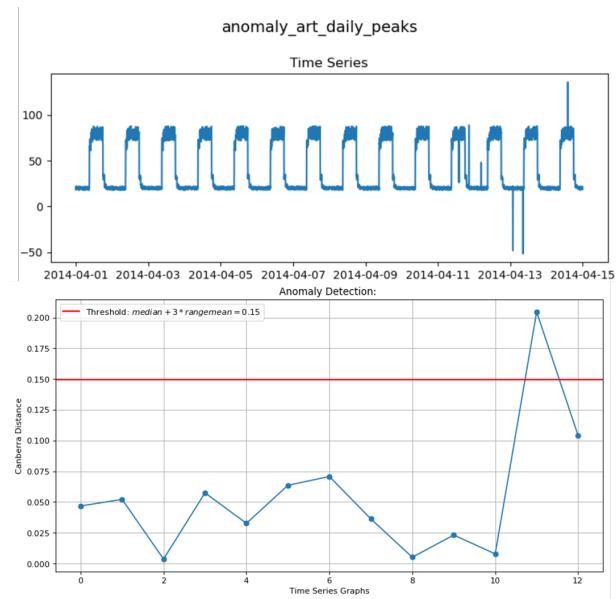
Eindimensionales Signal



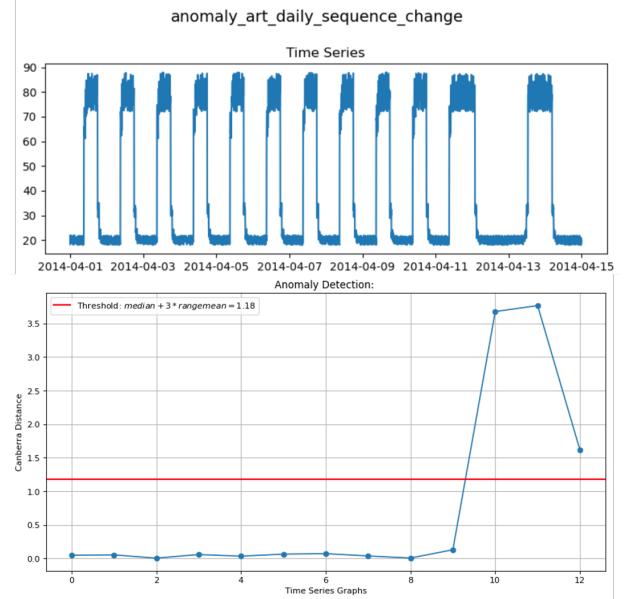
(a) Ausreißertyp Signal Drift



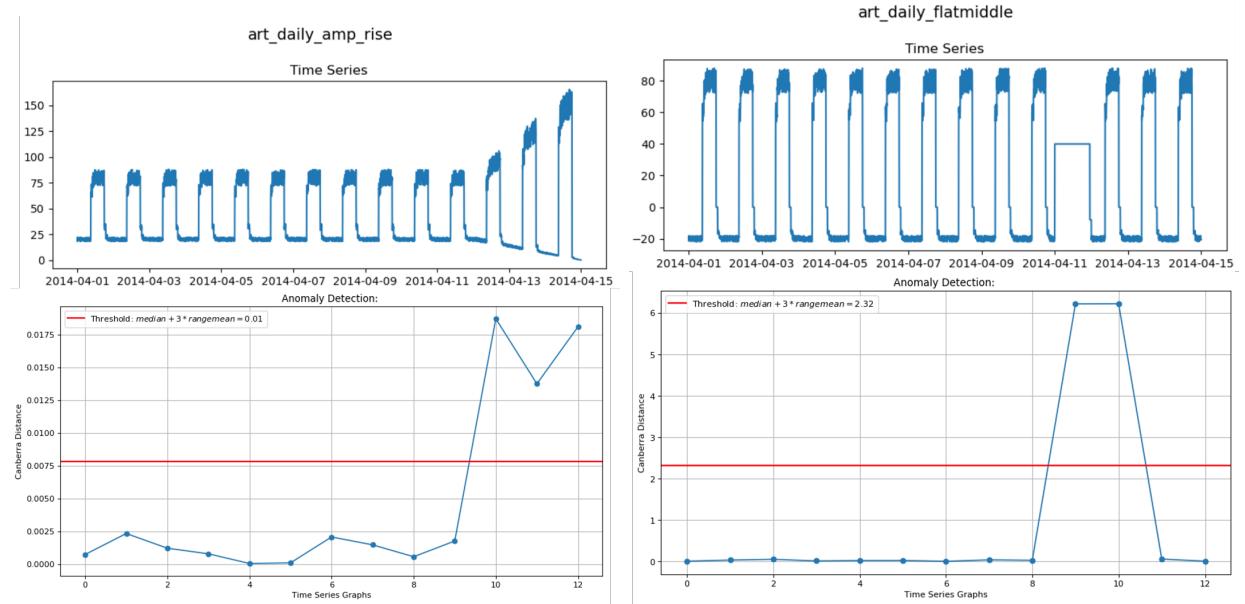
(b) Ausreißertyp Zunahme am Rauschen



(c) Ausreißertyp Einzelne Peaks

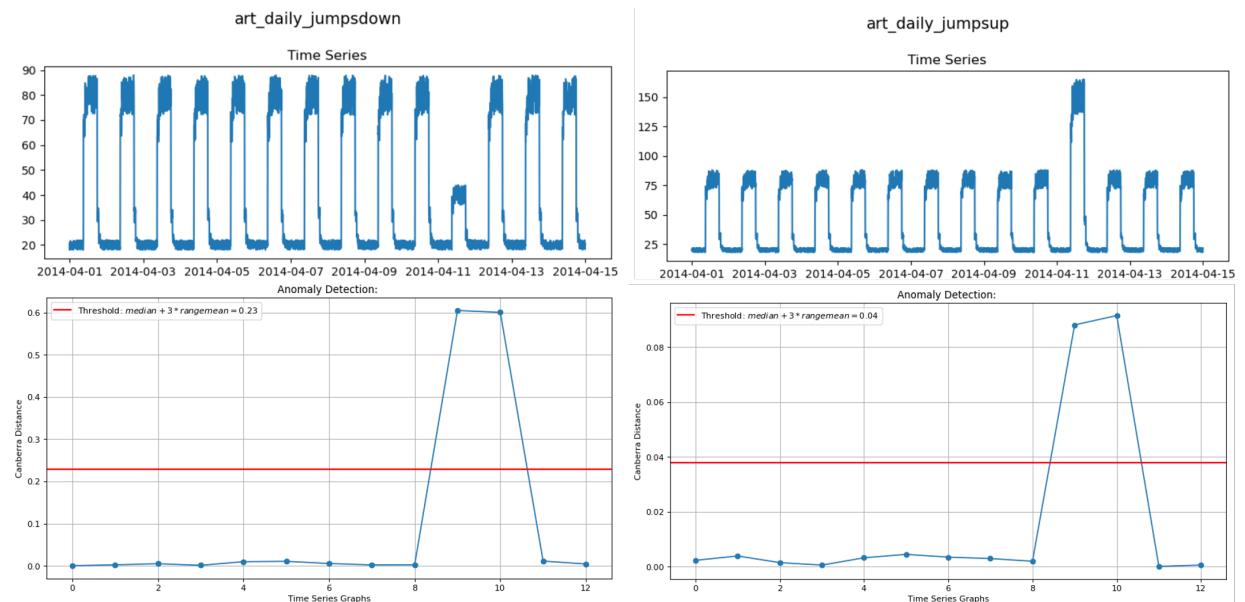


(d) Ausreißertyp Frequenzänderung



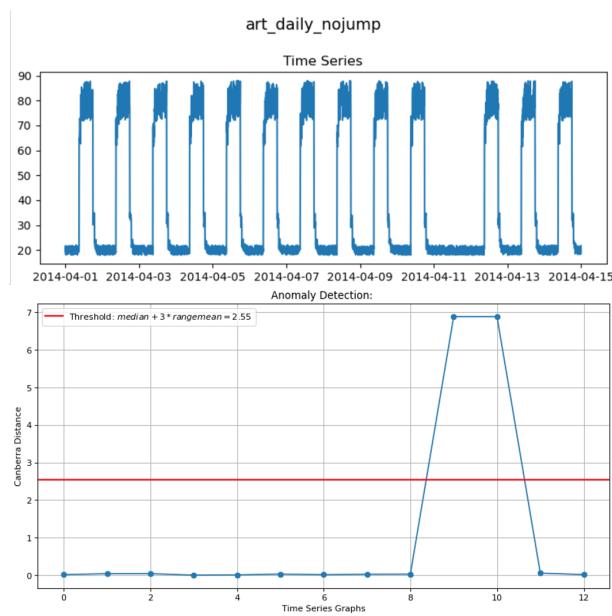
(e) Ausreißertyp Kontinuierliche Zunahme der Amplitude

(f) Ausreißertyp Zyklus Aussetzer



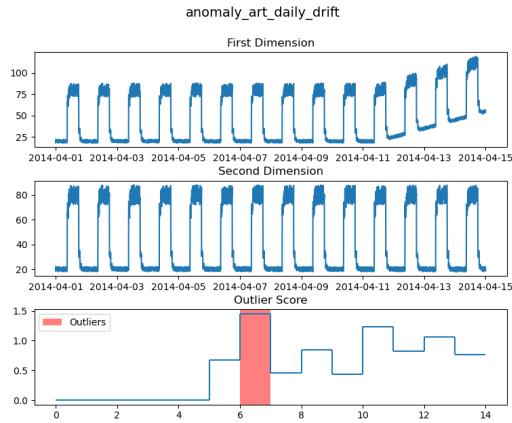
(g) Ausreißertyp Zyklus mit geringerer Amplitude

(h) Ausreißertyp Zyklus mit höherer Amplitude

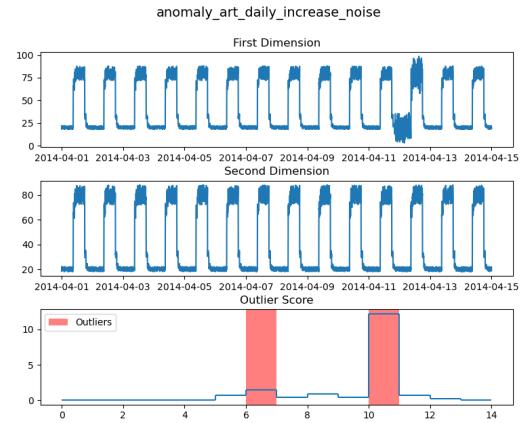


(i) Ausreißertyp Signal-Aussetzer

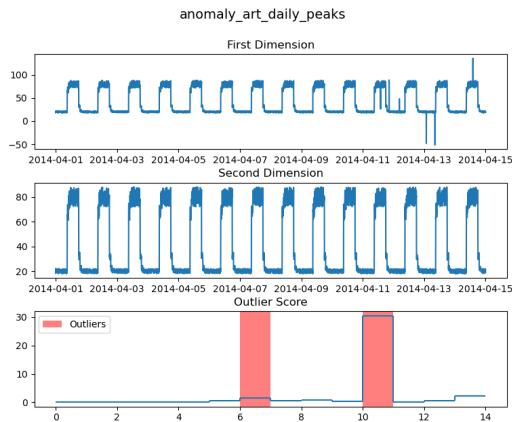
Zweidimensionales Signal



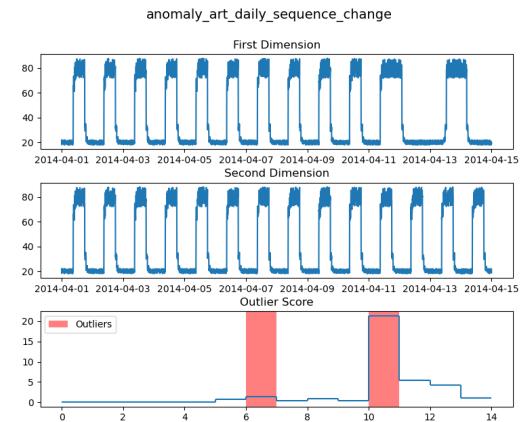
(a) Ausreißer-Typ Signal Drift



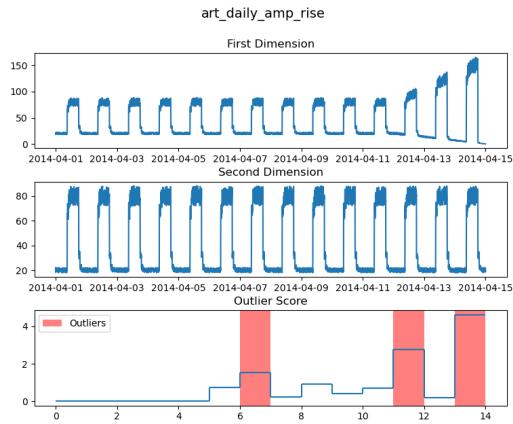
(b) Ausreißer-Typ Zunahme an Rauschen



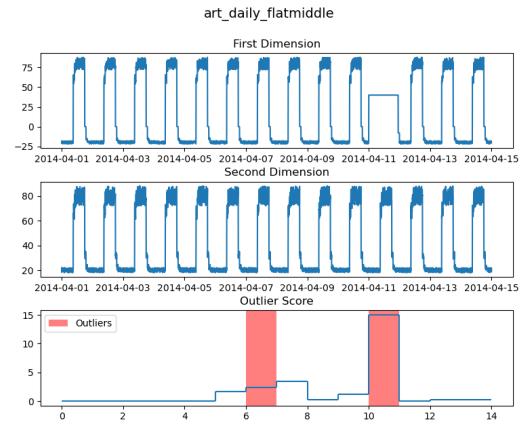
(c) Ausreißer-Typ Einzelne Peaks



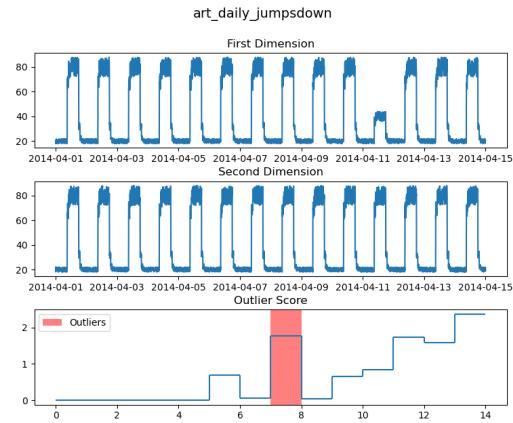
(d) Ausreißer-Typ Frequenzänderung



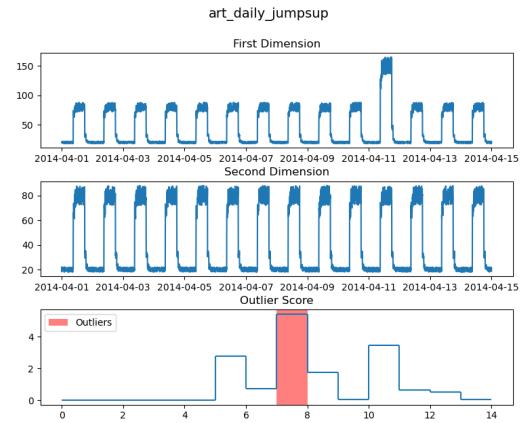
(e) Ausreißer-Typ Kontinuierliche Zunahme der Amplitude



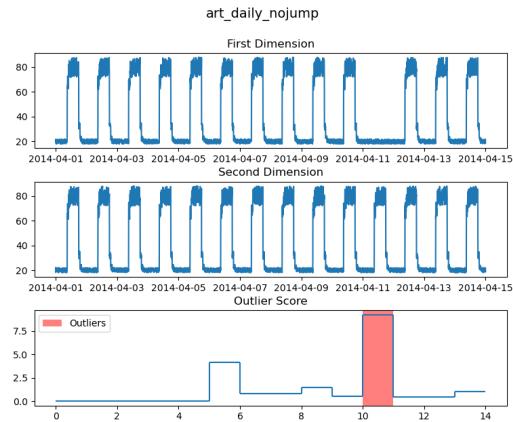
(f) Ausreißer-Typ Zyklus-Aussetzer



(g) Ausreißer-Typ Zyklus mit geringerer Amplitude



(h) Ausreißer-Typ Zyklus mit höherer Amplitude



(i) Ausreißer-Typ Signal-Aussetzer

Ergebnisse vollständiger Graphen

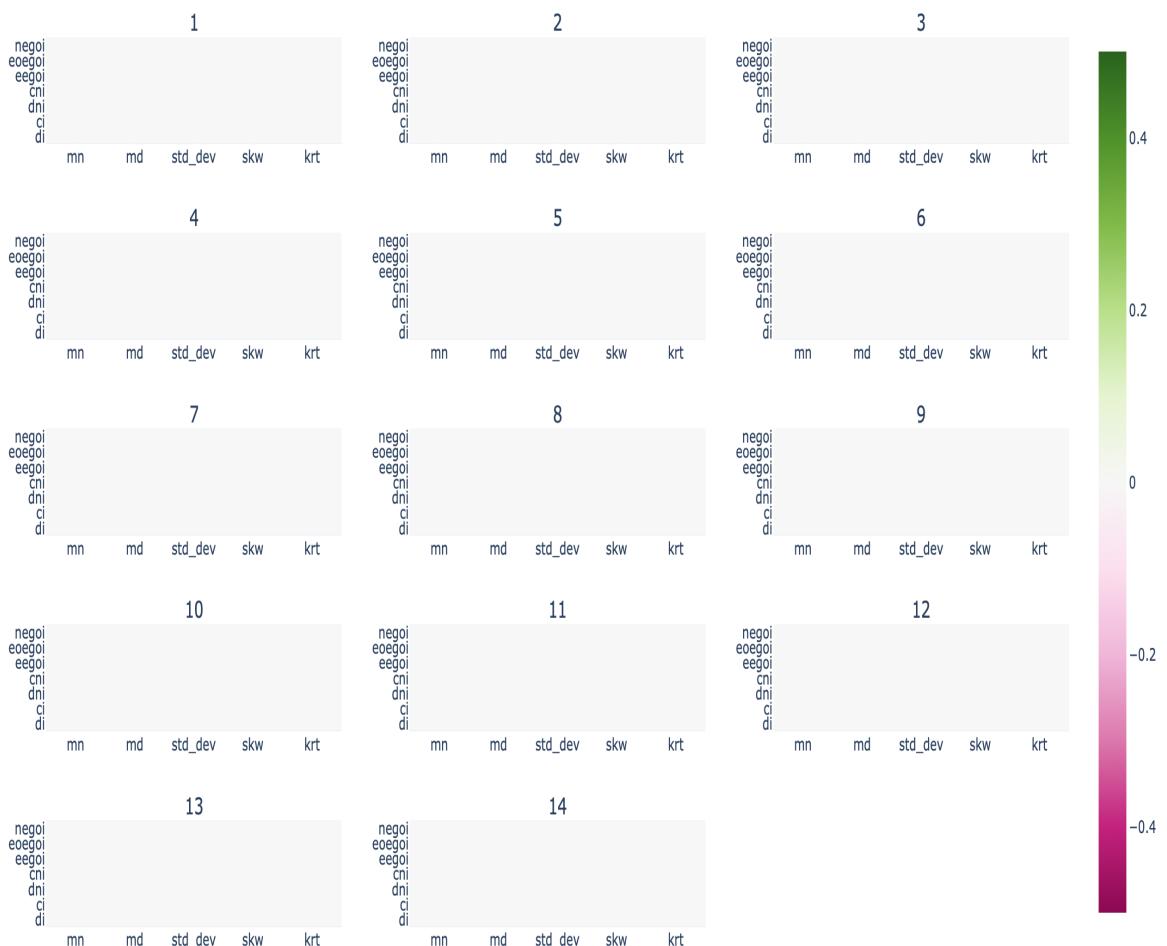


Abb. A.-2: Signaturvektoren der Zeitreihe mit vollständigen Graphen

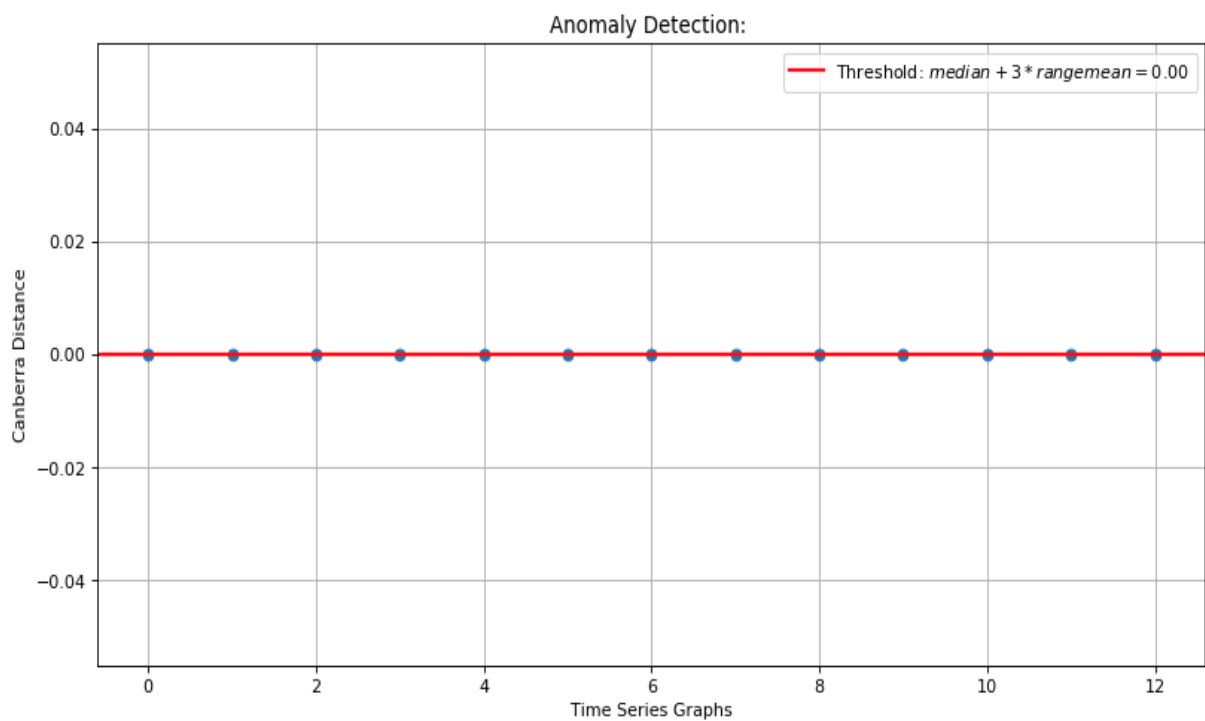
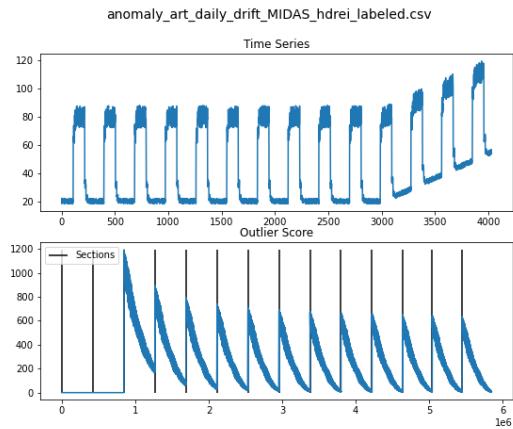


Abb. A.-1: Ausreißer Score der vollständigen Graphen

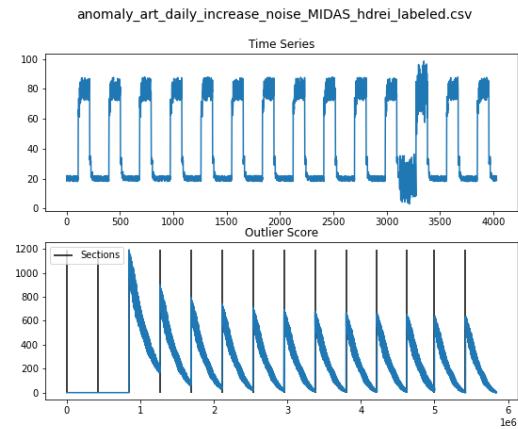
todo: Wrong picture for daily peaks. Change that the sixed element is not always an outlier

B Midas

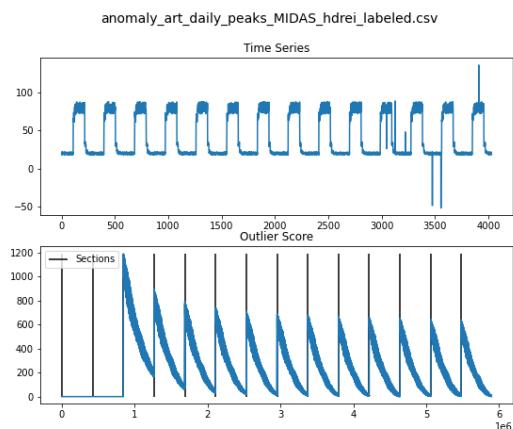
Eindimensionales Signal



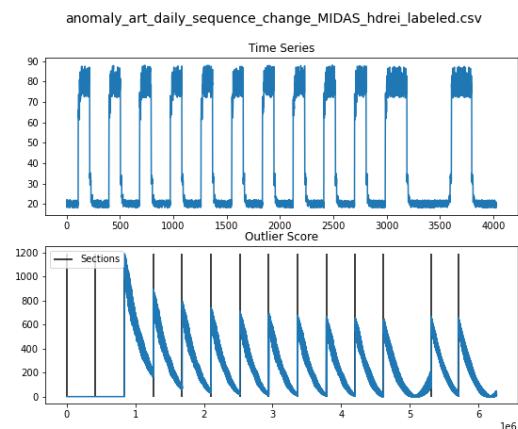
(a) Ausreißer-Typ Signal Drift



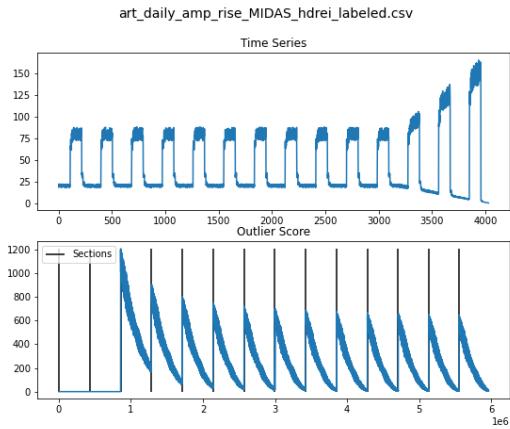
(b) Ausreißer-Typ Zunahme an Rauschen



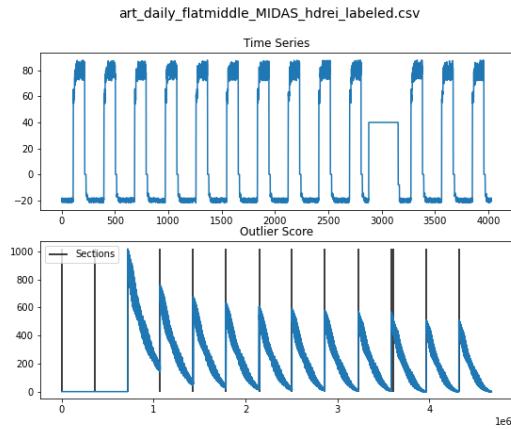
(c) Ausreißer-Typ Einzelne Peaks



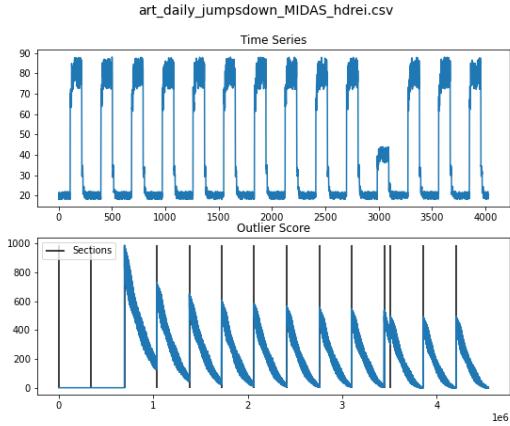
(d) Ausreißer-Typ Frequenzänderung



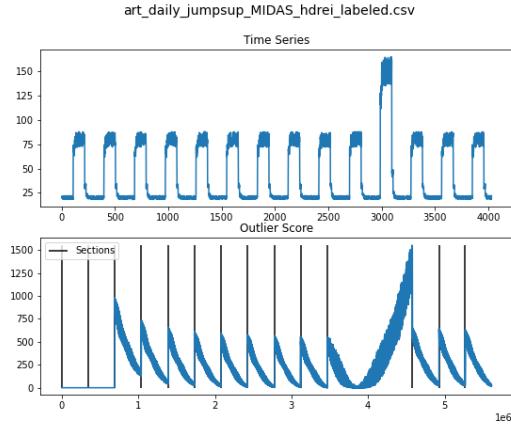
(e) Ausreißer-Typ Kontinuierliche Zunahme der Amplitude



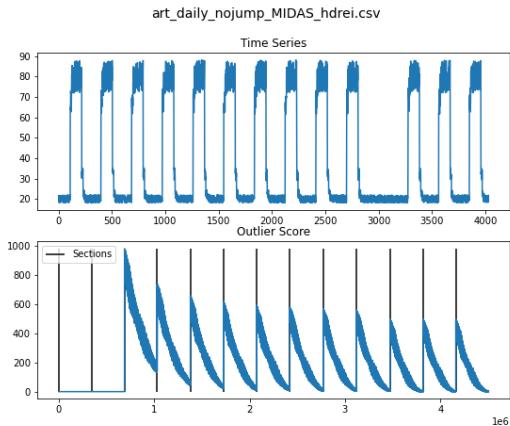
(f) Ausreißer-Typ Zyklus-Aussetzer



(g) Ausreißer-Typ Zyklus mit geringerer Amplitude



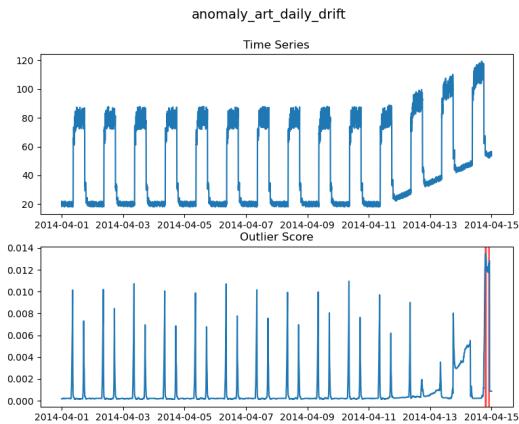
(h) Ausreißer-Typ Zyklus mit höherer Amplitude



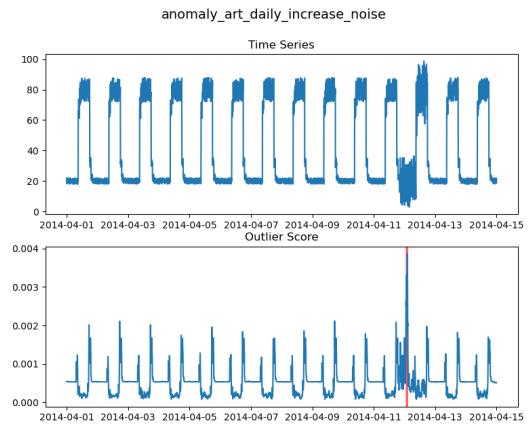
(i) Ausreißer-Typ Signal-Aussetzer

C Isomap

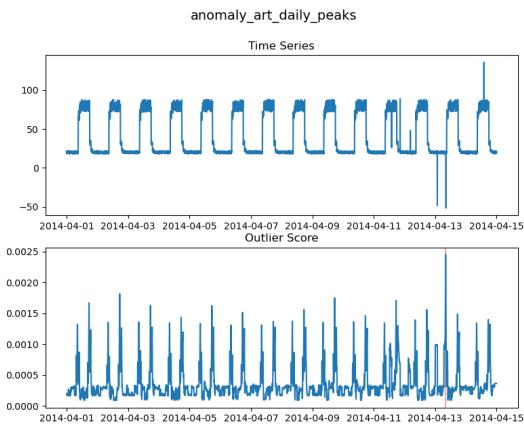
Eindimensionales Signal



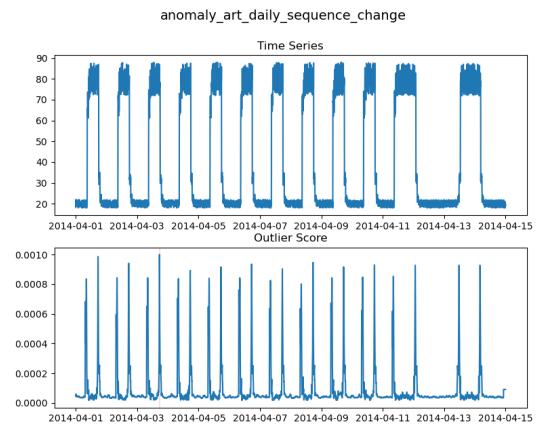
(a) Ausreißer-Typ Signal Drift



(b) Ausreißer-Typ Zunahme an Rauschen

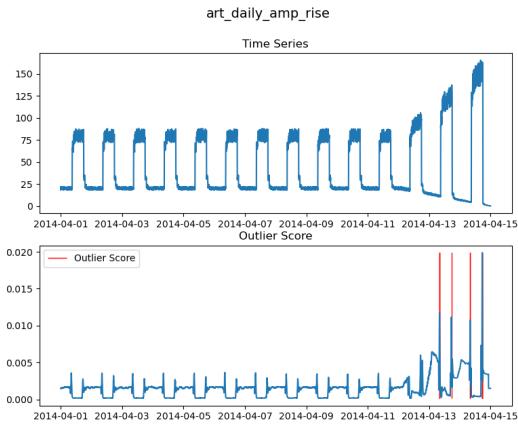


(c) Ausreißer-Typ Einzelne Peaks

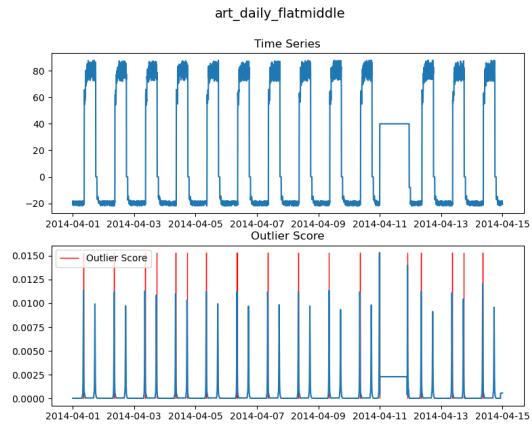


(d) Ausreißer-Typ Frequenzänderung

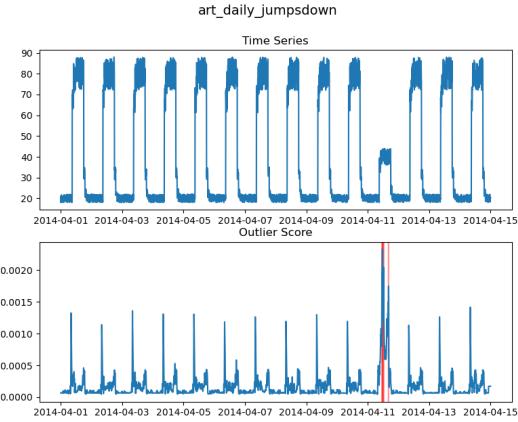
todo: In einigen Bilden fehlt die Legende. Vielleicht noch ein Paar bessere Ergebnisse zu erzielen



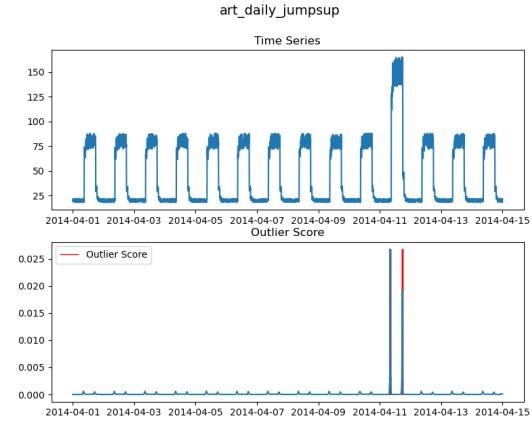
(e) Ausreißer-Typ Kontinuierliche Zunahme der Amplitude



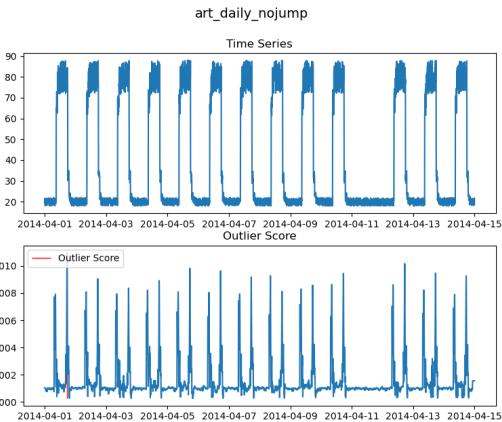
(f) Ausreißer-Typ Zyklus-Aussetzer



(g) Ausreißer-Typ Zyklus mit geringerer Amplitude



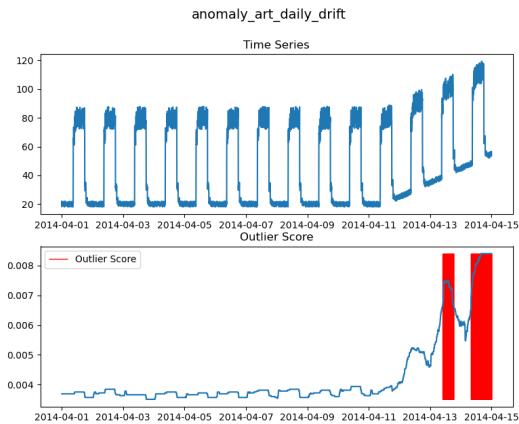
(h) Ausreißer-Typ Zyklus mit höherer Amplitude



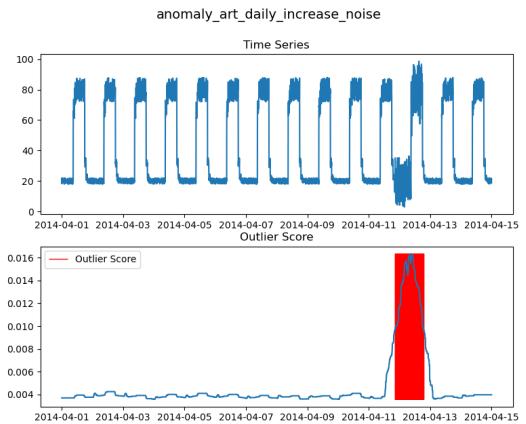
(i) Ausreißer-Typ Signal-Aussetzer

D Percolation

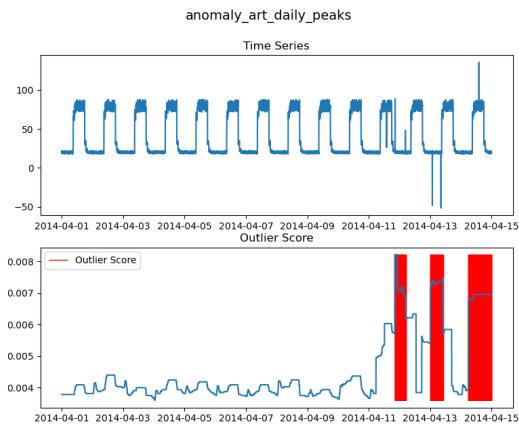
Eindimensionales Signal



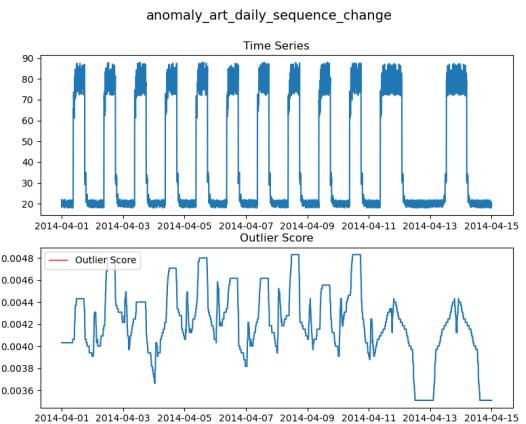
(a) Ausreißer-Typ Signal Drift



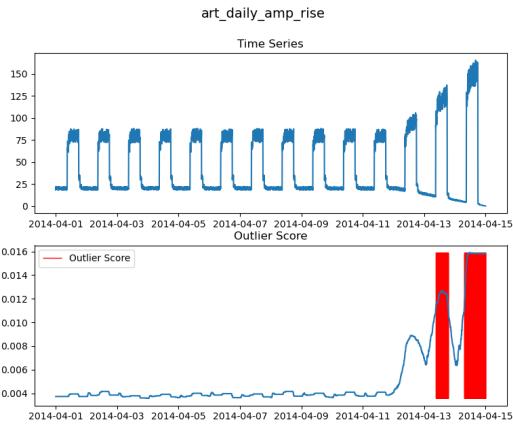
(b) Ausreißer-Typ Zunahme an Rauschen



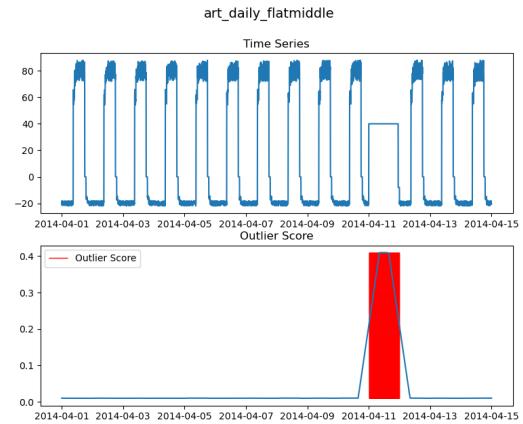
(c) Ausreißer-Typ Einzelne Peaks



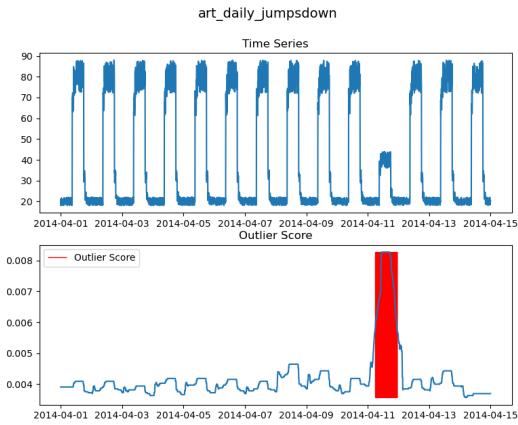
(d) Ausreißer-Typ Frequenzänderung



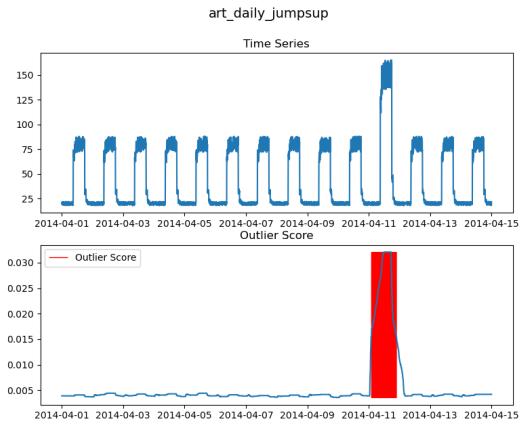
(e) Ausreißer-Typ Kontinuierliche Zunahme der Amplitude



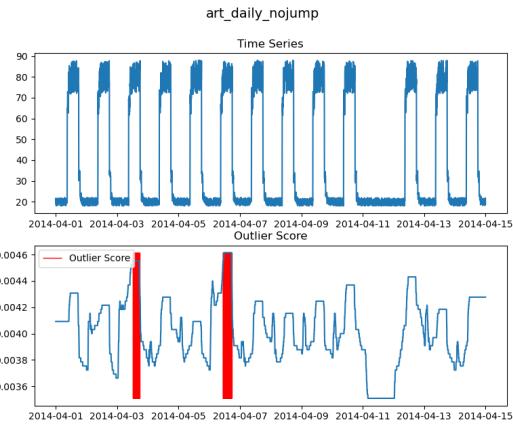
(f) Ausreißer-Typ Zyklus-Aussetzer



(g) Ausreißer-Typ Zyklus mit geringerer Amplitude



(h) Ausreißer-Typ Zyklus mit höherer Amplitude



(i) Ausreißer-Typ Signal-Aussetzer

todo: Nim mir nicht sicher ob ich das ohne sliding window auch noch einfügen soll. Vielleicht kann ich oben ja einmal einen Vergleich mit sliding window und ohne sliding window rein machen

Literaturverzeichnis

- [1] Ahmad, S., Lavin, A., Purdy, S. and Agha, Z. [2017], ‘Unsupervised real-time anomaly detection for streaming data’, *Neurocomputing* **262**, 134–147. Online Real-Time Learning Strategies for Data Streams.
URL: <https://www.sciencedirect.com/science/article/pii/S0925231217309864>
- [2] Akoglu, L., McGlohon, M. and Faloutsos, C. [2010], oddball: Spotting anomalies in weighted graphs, in M. J. Zaki, J. X. Yu, B. Ravindran and V. Pudi, eds, ‘Advances in Knowledge Discovery and Data Mining’, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 410–421.
- [3] Amil, P., Almeira, N. and Masoller, C. [2019], ‘Outlier mining methods based on graph structure analysis’, *Frontiers in Physics* **7**, 194.
URL: <https://www.frontiersin.org/article/10.3389/fphy.2019.00194>
- [4] Berlingerio, M., Koutra, D., Eliassi-Rad, T. and Faloutsos, C. [2012], ‘Netsimile: A scalable approach to size-independent network similarity’, *CoRR abs/1209.2684*.
URL: <http://arxiv.org/abs/1209.2684>
- [5] Bhatia, S., Hooi, B., Yoon, M., Shin, K. and Faloutsos, C. [2020], Midas: Microcluster-based detector of anomalies in edge streams, in ‘AAAI 2020 : The Thirty-Fourth AAAI Conference on Artificial Intelligence’.
- [6] Cha, S.-H. [2007], ‘Comprehensive survey on distance/similarity measures between probability density functions’, *Int. J. Math. Model. Meth. Appl. Sci.* **1**.
- [7] Cheng, H., Tan, P., Potter, C. and Klooster, S. [2008], A robust graph-based algorithm for detection and characterization of anomalies in noisy multivariate time series, in ‘2008 IEEE International Conference on Data Mining Workshops’, pp. 349–358.
- [8] Cormode, G. and Muthukrishnan, S. [2004], An improved data stream summary: The count-min sketch and its applications., pp. 29–38.
- [9] Lippmann, R., Haines, J., Fried, D., Korba, J. and Das, K. [2000], Analysis and results of the 1999 darpa off-line intrusion detection evaluation, pp. 162–182.
- [10] scikit-learn developers [2020], ‘sklearn.manifold.isomap’. Letzter Zugriff : 13.03.2021.
URL: <https://scikit-learn.org/stable/modules/generated/sklearn.manifold.Isomap.html>
- [11] Tenenbaum, J. B., Silva, V. d. and Langford, J. C. [2000], ‘A global geometric framework for nonlinear dimensionality reduction’, *Science* **290**(5500), 2319–2323.
URL: <https://science.sciencemag.org/content/290/5500/2319>
- [12] Uzun, B., Kielman, J. and Erz, M. [2020], ‘Anomalie-Erkennung in Graphen’. nicht publiziert.

- [13] Xu, X., Yuruk, N., Feng, Z. and Schweiger, T. A. J. [2007], Scan: A structural clustering algorithm for networks, *in* ‘Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining’, KDD ’07, Association for Computing Machinery, New York, NY, USA, p. 824?833.

URL: <https://doi.org/10.1145/1281192.1281280>