

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 Hintergrund | 1 |
| 1.2 Problemstellung | 2 |
| 1.3 Verwandte Arbeiten | 2 |
| 2 Verwendete Daten | 4 |
| 2.1 Numenta Zeitreihen Daten | 4 |
| 2.2 Datensätze | 4 |
| 2.2.1 Enron | 4 |
| 3 Statische Algorithmen zur Ausreißer Erkennung | 5 |
| 3.1 Umwandlung Zeitreihe in Netzwerk | 5 |
| 3.2 IsoMap Basierter Algorithmus | 6 |
| 3.2.1 IsoMap | 6 |
| 3.2.2 IsoMap zur Erkennung von Ausreisern | 6 |
| 3.2.3 Implementierung | 7 |
| 3.2.4 Ausreißererkennung in Zeitreihen | 7 |
| 3.3 Perculation basierter Algorithmus | 9 |
| 3.3.1 Implementierung | 9 |
| 3.3.2 Ausreißererkennung in Zeitreihen | 10 |
| 4 Dynamische Algorithmen zur Ausreißer Erkennung | 12 |
| 4.1 Umwandlung der Daten in ein Netzwerk | 12 |
| 4.2 Netsimile | 14 |
| 4.2.1 Grundlagen | 14 |
| 4.2.2 Ausreißer-Erkennung in Graphen | 15 |
| 4.2.3 Erweiterung des Algorithmus | 16 |
| 4.2.4 Optimierte Implementierung des Algorithmus | 23 |
| 4.3 MIDAS | 29 |
| 4.3.1 Grundlagen | 29 |
| 4.3.2 Ausreißer-Erkennung in Graphen | 30 |
| 4.3.3 Ausreißer-Erkennung in Zeitreihen | 30 |

| | | |
|---|--|----|
| A | Netsimile | 36 |
| | A.1 Eindimensionales Signal | 36 |
| | A.2 Zweidimensionales Signal | 37 |
| B | Midas | 41 |
| C | Isomap | 43 |
| | C.1 Eindimensionales Signal | 43 |
| D | Perculation | 45 |
| | D.1 Sliding Window | 45 |
| | Literaturverzeichnis | 48 |

Abbildungsverzeichnis

| | | |
|------|---|----|
| 3.1 | Berechnung der Distanz zwischen zwei Punkten nach Anwendung des IsoMap Algoritmus | 6 |
| 3.2 | Problem Übergänge | 8 |
| 3.3 | Ablauf Perculation basierter Algoritmus | 9 |
| 3.4 | Vergleich Perculation Algoritmus mit Sliding Window Verfahren und ohne Sliding Window Verfahren | 10 |
| 4.1 | Umwandlung einer Zeitreihe in Netzwerk. Bei dem Netzwerk handelt es sich hier um ein Symbolbild. | 12 |
| 4.2 | Datensatz Midas. 1Spalte: Ursprungsknoten, 2Spalte: Zielknoten, 3Spalte: Abschnitt | 13 |
| 4.3 | todo: Platzhalter: Graph with 11 vertices | 16 |
| 4.4 | todo: Platzhalter: Heatmap 1 | 18 |
| 4.5 | todo: Platzhalter: Anomaly 1 | 19 |
| 4.6 | todo: Platzhalter: Anomaly 2 | 19 |
| 4.7 | todo: Platzhalter: Anomaly 3 | 20 |
| 4.8 | todo: Platzhalter: heatmap 3 | 21 |
| 4.9 | todo: Platzhalter: Anomaly 4 | 22 |
| 4.10 | todo: Platzhalter: Dauer 4 | 22 |
| 4.11 | todo: Platzhalter: Sequence Change | 23 |
| 4.12 | todo: Platzhalter: Sequence Change | 23 |
| 4.13 | todo: Platzhalter: Peaks | 24 |
| 4.14 | todo: Platzhalter: Peaks | 24 |
| 4.15 | todo: Platzhalter: No Jump | 25 |
| 4.16 | todo: Platzhalter: No Jump | 25 |
| 4.17 | todo: Platzhalter: Increase Noise | 26 |
| 4.18 | todo: Platzhalter: Increase Noise | 26 |
| 4.19 | todo: Platzhalter: Flatmiddle | 27 |
| 4.20 | todo: Platzhalter: Flatmiddle | 27 |
| 4.21 | todo: Platzhalter: Enron | 31 |
| 4.22 | todo: Platzhalter: Enron | 32 |
| 4.23 | todo: Platzhalter: Darpa | 33 |
| 4.24 | MIDAS Algoritmus angewandt auf Zeitreihe mit einer erhöten Amplitude. | 34 |
| 4.25 | Ausreißer Erkennung in Zeitreihen MIDAS Algoritmus | 34 |
| 4.26 | Ausreißer Erkennung Zeitreihen MIDAS Algoritmus Fenstergröße 110 | 35 |
| 4.27 | Ausreißer Erkennung Zeitreihen MIDAS-R | 35 |

Tabellenverzeichnis

| | | |
|-----|---|----|
| 3.1 | IsoMap Performance | 7 |
| 3.2 | Perculation Time Series Performance | 10 |
| 4.1 | Parameter Netismile Zeitreihen | 26 |
| 4.2 | Netsimile Time Series Perfomance | 28 |

Listings

| | |
|---------------------------------|----|
| 4.1 todo: Platzhalter | 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.

todo: Den dynamischen Aspekt nicht vergessen

1.1 Hintergrund

todo: formulieren

1.2 Problemstellung

todo: Ziele definieren Das Ziel dieser Forschungsprojekt ist es verschiedene Algorithmen anzuwenden und erste Erkenntnisse aus ihnen zu gewinnen. Dieses Hauptziel, im Zuge des ersten Semesters des Forschungsprojekts, kann wie folgt in drei Teilziele unterteilt werden:

1. Verschaffen eines Überblicks über die existierenden Algorithmen zur Erkennung von Ausreißern in Graphen
2. Die Entwicklung eines Ausreißer-Scores für die zugrundeliegenden Algorithmen
3. Erste Anwendung der verwendeten Graphen-basierten Algorithmen auf Zeitreihen

1.3 Verwandte Arbeiten

todo: related work einfügen

Die **Anomalieerkennung in Edge Streams** verwendet als Eingabe einen Fluss von Kanten über die Zeit. Sie werden nach der Art der erkannten Anomalie kategorisiert:

Erkennung anomaler Knoten: Mithilfe eines Edge Streams erkennt (Yu et al. 2013) Knoten, deren Egonetze sich plötzlich und signifikant ändern.

Erkennung anomaler Subgraphen : Mithilfe eines Edge Streams identifiziert (Shin et al. 2017) dichte Teilgraphen, die innerhalb einer kurzen Zeit entstehen.

Anomale Kantenerkennung: (Ranshous et al. 2016) konzentriert sich auf spärlich verbundene Teile eines Graphen, während (Eswaran und Faloutsos 2018) Kantenanomalien basierend auf dem Auftreten von Kanten, bevorzugter Anhaftung und gegenseitigen Nachbarn identifiziert.

Die **Ausreißer Erkennung in Zeitreihen und Sequenziellen Daten** wurde bereits in vielen Literaturquellen diskutiert.

Netzwerk basierter Ansatz zur Erkennung von Ausreißern in Sequenziellen Daten[8]: Der genannte Algorithmus wandelt Sequenziellen Daten in ein Netzwerk um. Dabei wird die Euklidischen Distanz genutzt um die Kantengewichte zu berechnen. Anschließend werden die Knoten mithilfe des Minimum Spanning Tree Algorithmus geclustered. Um hierraus Ausreißer zu abzuleiten wird ein Voting Scheme verwendet. Der vorgestellte Algorithmus wurde genutzt um Ausreißer in Wetter Daten sowie Aktienkursen zu identifizieren.

Ein robuster graphbasierter Algorithmus zur Erkennung und Charakterisierung von Anomalien in verrauschten Multivariaten Zeitreihen: In diesem Paper [5] wird ein Algorithmus vorgestellt, der dazu in der Lage ist Ausreißer in Multivariaten Zeitreihen zu erkennen. Die multivariate Zeitreihe wird dabei über ein Distanzmaß in ein Netzwerk umgewandelt. Auf dem Netzwerk wird anschließend ein Random Walk Algorithmus ausgeführt. Daraufhin werden Knoten die besonders selten besucht wurden als Ausreißer markiert.

Überblicksartikel über die Ausreißer Erkennung in Diskreten Sequenzen : In [4] werden verschiedene Methoden vorgestellt, wie Ausreißer in Sequenzen erkannt werden können. Es wird dabei, auch auf die Ausreißer Erkennung in Zeitreihen eingegangen. Die vorgestellten Algorithmen werden in drei Kategorien untergliedert. 1: Erkennung abnormaler Sequenzen in Bezug auf eine Datenbank normaler Sequenzen 2: Erkennung einer abnormalen Untersequenz innerhalb einer langen Sequenz. 3: Erkennung eines Musters in einer Sequenz deren Auftrittshäufigkeit anomal ist.

Neuronale Netze zur Ausreißer Erkennung : Die Verwendung von Neuronalen Netzen zur Erkennung von Ausreißern wird immer beliebter. Beispielsweise wurde in [6] ein Replicator Neuronales Netz, einerseits genutzt um Störungen in einem Netzwerk zu erkennen. Des Weiteren wurde das Neuronale Netz verwendet um Ausreißer in einem Brustkrebs Datensatz zu identifizieren. Neuronale Netze wurden ebenso dazu eingesetzt um Ausreißer in Zeitreihen zu finden [7]. Ein Vorteil dieses Ansatzes ist, dass Ausreißer online entdeckt werden können. Das Neuronale Netz wird hierbei dazu genutzt den nächsten Wert einer Zeitreihe zu schätzen. Die Differenz zwischen der Vorhersage und dem tatsächlich auftretenden Wert wird als Ausreißer Score verwendet.

2 Verwendete Daten

2.1 Numenta Zeitreihen Daten

Bei diesem Datensatz handelt es sich um künstlich erzeugte Zeitreihen der Numenta Gruppe. Diese Zeitreihen enthalten unterschiedliche Arten von Ausreißern. Dadurch kann untersucht werden für welche Ausreißer Typen die Algorithmen gut geeignet sind. Für die Tests auf multivariaten Zeitreihen wurden neue Zeitreihen erzeugt. Dabei wurde für die erste Dimension eine Zeitreihe der Numenta Gruppe verwendet. Für weitere Dimensionen wurde auf eine Zeitreihe ohne Ausreißer zurück gegriffen.

todo: vielleicht noch Bild von einer Zeitreihe einfügen. todo: Vielleicht Numenta verlinken

2.2 Datensätze

2.2.1 Enron

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.

3 Statische Algorithmen zur Ausreißer Erkennung

In diesem Kapitel werden zunächst zwei statische Algorithmen zur Ausreißer Erkennung auf unterschiedlichen Datentypen (z.B. Videos, Bilder, Netzwerke) vorgestellt. Hierbei handelt es sich um ein auf Percolation basierender Algorithmus und ein auf IsoMap basierender Algorithmus. Statische Algorithmen kennzeichnen, dass sie nicht mit Daten umgehen können, welche kontinuierlich an sie übergeben werden. Damit die Algorithmen anwendbar sind müssen die Daten vollständig und abgeschlossen vorliegen. Eines unserer Hauptziele des Forschungsprojektes war die Ausreißer Erkennung in Zeitreihen, aus diesem Grund untersuchten wir die Algorithmen auf ihre Fähigkeit Ausreißer in Zeitreihen zu finden. In [Kap. 3.2](#) wird der Iso Map basierte Algorithmus vorgestellt in [Kap. 3.3](#) wird der Percolation basierte Algorithmus vorgestellt.

Beide Algorithmen liefern lediglich einen Ausreißer Score zurück. Um zu bestimmen inwiefern ein Element konkret ein Ausreißer ist, wird zunächst den Mittelwert und die Standardabweichung des Outlier Scores berechnet. Falls ein Element in Abhängigkeit von der Standardabweichung sehr stark vom Mittelwert abweicht wird das Element als Ausreißer klassifiziert.

3.1 Umwandlung Zeitreihe in Netzwerk

Damit sowohl der auf Percolation basierende Algorithmus wie auch der auf Iso Map basierende Algorithmus angewandt werden können, müssen die Daten zunächst in ein einheitliches Format überführt werden. Dazu müssen die unterschiedlichen Daten in ein Netzwerk umgewandelt werden. Hierzu ist erforderlich, dass eine Distanz zwischen unterschiedlichen Elementen des Datensatzes berechnet werden kann [vgl. [1](#), S. 2]. Der Algorithmus kann auf allen Daten angewandt werden, welche diese Voraussetzung erfüllen. Nachfolgend wird exemplarisch beschrieben wie die Transformation für eine Zeitreihe funktionieren kann.

Für die Transformation der 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. Das neue Netzwerk ist hierbei Fully Connected. Das heißt jeder Knoten ist mit allen

anderen Knoten über eine Kante verknüpft. Die Knoten des Netzwerks repräsentieren die einzelnen Elemente(Zeitpunkte) der zeit reihe. [vgl. 1, S. 2]. Es können mit dieser Vorgehensweise auch multivariate Zeitreihen in ein Netzwerk transformiert werden.

3.2 IsoMap Basierter Algorithmus

Der Ansatz dieses Algorithmus ist, dass Informationen über Ausreißer bei der Reduzierung der Dimensionalität mit dem IsoMap verloren gehen. Insofern versucht wird, die Informationen zu rekonstruieren und mit der ursprünglichen Matrix vergleicht, können große Abweichungen bei Ausreißer Elementen festgestellt werden [vgl. 1, S. 3]. In Kap. 3.2.1 wird zunächst erklärt wie der Iso Map Algorithmus eine Reduzierung der Dimensionalität durchführt. Anschließend werden in Kap. 3.2.2 die zusätzlichen Schritte erläutert, welche notwendig sind um Ausreißer mithilfe des IsoMap Algorithmus zu erkennen.

3.2.1 IsoMap

Beim IsoMap handelt es sich um einen Algorithmus zur nichtlinearen Dimensionsreduktion. Zunächst werden beim IsoMap Algorithmus die Nachbarn eines jeden Punktes(Knoten) über K-Nearest Neighbor 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 auch 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. 9, S. 3-4].

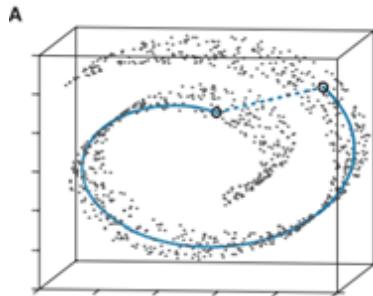


Abb. 3.1: Berechnung der Distanz zwischen zwei Punkten nach Anwendung des IsoMap Algorithmus

3.2.2 IsoMap zur Erkennung von Ausreisern

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 Featurn die ursprüngliche Distanzmatrix zu

rekonstruieren. Dazu wird aus den Featuren 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. 1, S. 3].

3.2.3 Implementierung

Für den Iso Map Algorithmus stellt SciKit Learn eine sehr gute Implementierung zur Verfügung. Diese Implementierung konnten 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 Python/Numpy zurückgegriffen.

3.2.4 Ausreißererkennung in Zeitreihen

todo: Erklären welche Zeitreihen verwendet wurden. Hab ich so ein bisschen gemacht vielleicht das noch in extra Kapitel machen.

Für die durchgeföhrten Tests wurden die Zeitreihen aus ... verwendet. Um zu bewerten wie gut der Algorithmus funktioniert, wurde ein Punktesystem eingeföhrt. In dem Punktesystem konnten maximal vier Sterne erreicht werden. Das bedeutet Ausreißer sehr gut erkannt. Null Sterne hingegen bedeuten Ausreißer überhaupt nicht erkannt. Der IsoMap Algorithmus liefert eher schwache Ergebnisse bei der Erkennung von Ausreißern in Zeitreihen. Hauptproblem hierbei ist, dass starke Anstiege, bei welchen es sich nicht um Ausreißer handelt, fälschlicherweise zu einem starken Anstieg des Ausreißer Scores führen (vgl. Abb. 3.2 mit Pfeil markierte stellen). Dies kann, je nach Threshold, zu einer hohen Quote an falsch positiven Klassifizierungen führen.

Tab. 3.1: IsoMap Performance

| Ausreißer Typ | Datei Name | 1D |
|---------------------------------------|-----------------------------------|----|
| 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 | - |

Aus diesem Grund können die tatsächlichen Ausreißer nicht eindeutig identifiziert werden. Eine Ähnliche Problematik trat in [10] 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 abrupte Übergänge in der Zeitreihe abgemildert und deshalb nicht mehr als Ausreißer erkannt [vgl. 10, S. 31,36]. Dies könnte ein möglicher Ansatz sein um zukünftig bessere Ergebnisse zu erzielen.

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 es kommt zu Änderungen in der Saisonalität der Zeitreihe.

todo: Wie genau bezieht man sich auf seine eigene Ausarbeitung

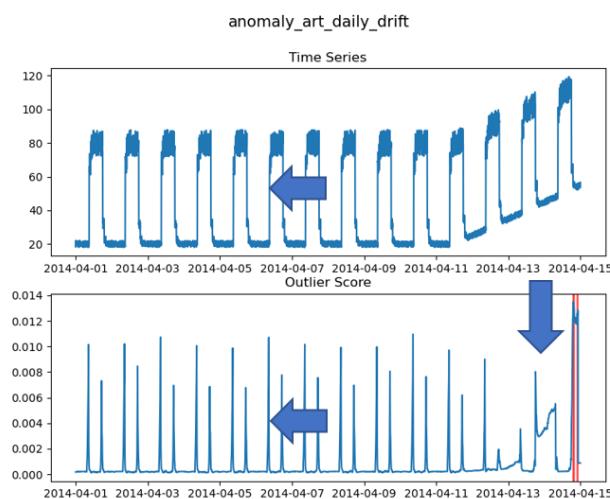


Abb. 3.2: Problem Übergänge

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 ihm 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. 1, S. 3].

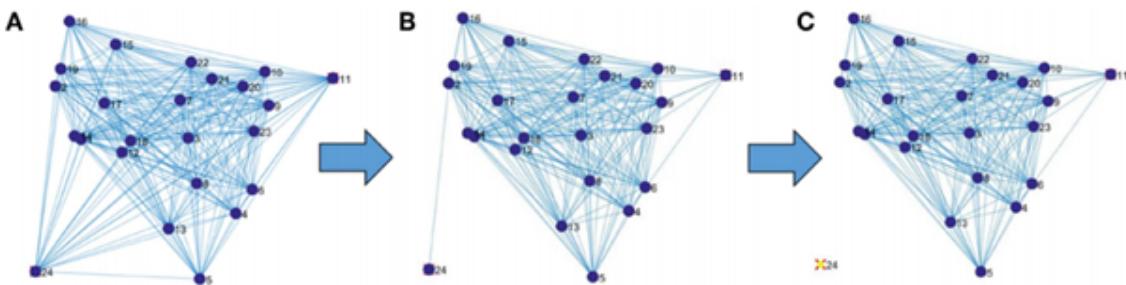


Abb. 3.3: Ablauf Perculation basierter Algorithmus

3.3.1 Implementierung

Für die Implementierung des Perculation-basierten Algorithmus wurde genauso wie in Kap. 3.2.3, 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, dass 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 Scores berechnet. In ?? 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

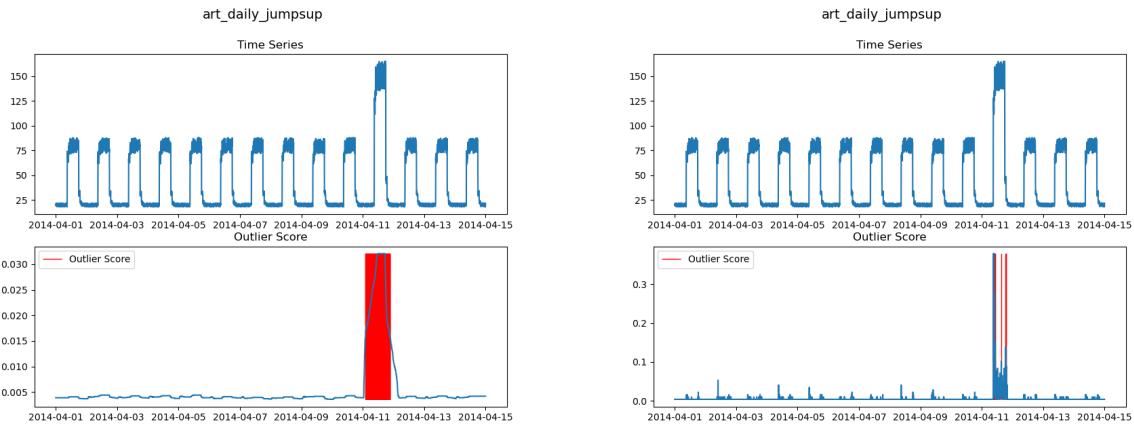


Abb. 3.4: Vergleich Perculation Algorithmus mit Sliding Window Verfahren und ohne Sliding Window Verfahren

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 muss also 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. 3.2.4)

Tab. 3.2: Perculation Time Series Performance

| Ausreißer Typ | Datei Name | 1D |
|---------------------------------------|-----------------------------------|------|
| 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 | - |

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 Netismile (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. Kap. 3.1). 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 Kap. 3.1 ist graphisch dargestellt wie die Umwandlung der Daten in ein Netzwerk funktioniert.

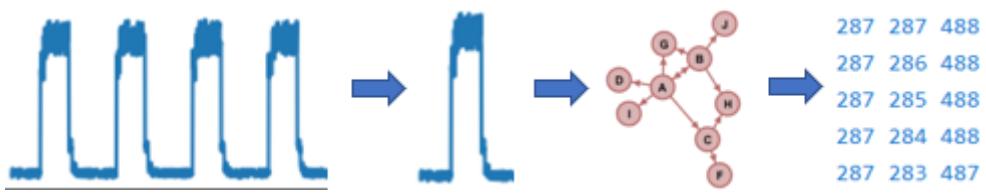


Abb. 4.1: Umwandlung einer Zeitreihe in Netzwerk. Bei dem Netzwerk handelt es sich hier um ein Symbolbild.

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

Netismile: Das Übergabeformat für den Netismile 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

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

4.2.1 Grundlagen

NetSimile ist ein skalierbarer Algorithmus zur Erkennung von Ähnlichkeiten, sowie Anomalien, in Netzwerken unterschiedlicher Größen. Wenn der Datensatz eines Graphs über die Zeit in bestimmte Abstände, wie z.B. in Tage, unterteilt wird, so kann NetSimile die Veränderung des Graphs über die Zeit bewerten. Der Algorithmus extrahiert strukturelle Merkmale aus den Momentaufnahmen des Graphs für jeden Tag. Diese Merkmale bilden den Signaturvektor für jeden Graphen in der sich verändernden Netzwerkumgebung und bestehen aus den Ego-Netzeigenschaften, Knotengrad, Clustering-Koeffizient usw. Um die Ähnlichkeit zwischen zwei Graphen bewerten zu können, wird beim NetSimile der Abstand ihrer entsprechenden Signaturvektoren berechnet. Dieser Abstand wird Canberra Distance genannt. [vgl. 2, 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. 2, 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.

$$d_{N(i)}$$

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

$$c_{N(i)}$$

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.

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

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

Im letzten Schritt wird bei diesem Algorithmus die Canberra-Distance-Funktion als Ähnlichkeitsmaß herangezogen.

todo: Canberra Distance Formel Seite 3 in Paper einfügen oder von Dictionary of Distances Chapter 17 als neue Quelle, es gibt kein Paper mit einer Beschreibung hierfür

todo: Info an Jeremy: Du könntest als Eigenarbeit weitere Distanzmetriken nutzen glaub. Hab ein Paper "Comprehensive Survey on Distance/Similarity Measures between Probability Density Functions" in dem Canberra in eine Kategorie mit Gower, Soergel, Kulczynski d, Lorentzian fällt und auch bei Netsimile wird erwähnt dass Canberra aus einer Vielzahl an Möglichkeiten gewählt wurde.

4.2.2 Ausreißer-Erkennung in Graphen

todo: Ergebnisse der Tests aufzeigen. Vielleicht die Visualisierung mit rein bringen. Vielleicht kurz Datensätze erklären und woher die Implementierung des Algorithmus stammt.

todo: 3.2.1 von Google Docs einfügen bis S.16 Datensätze Überschrift

4.2.3 Erweiterung des Algorithmus

todo: Erklären warum ein neues Feature benötigt wird. Vergleich des Algorithmus mit Feature und ohne neues Feature Damit der Algorithmus auf Netzwerke angewendet werden kann, muss dieser vorab erweitert werden, da in Netzwerken ansonsten nur die Verbindung im Allgemeinen betrachtet wird, nicht aber die Häufigkeit an Verbindungen, die zwischen zwei Knoten stattgefunden hat. Damit würde ein großer Anteil des Datensatzes verloren gehen, da der Algorithmus keine Mehrfach-Verbindungen zwischen Knoten im Graphen darstellen kann.

Hierfür werden die Mehrfach-Verbindungen aufsummiert und als Kanten-Gewichtung zwischen zwei Knoten dem Graphen hinzugefügt.

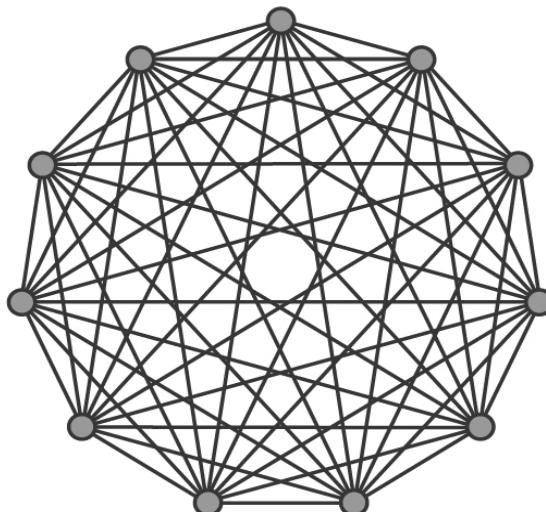
```
1 for i in range(len(e_list)):
2     g.add_edge(e_list[i][0], e_list[i][1], weight=e_list[i][2])
```

List. 4.1: todo: Platzhalter

Anschließend muss die Gewichtung aus diesem Graphen extrahiert werden. Dafür wird die Summe aller ausgehenden Kanten eines Knotens gebildet und als Feature hinzugefügt.

Für die Anwendung des Algorithmus auf Zeitreihen ist die Erweiterung um das Feature Gewichtung ebenfalls relevant, da sonst der Algorithmus auf einen vollständigen Graphen angewendet wird.

Das Problem hierbei ist, dass jeder Knoten eines Graphens 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



Complete graph with 11 vertices

Abb. 4.3: todo: Platzhalter: Graph with 11 vertices

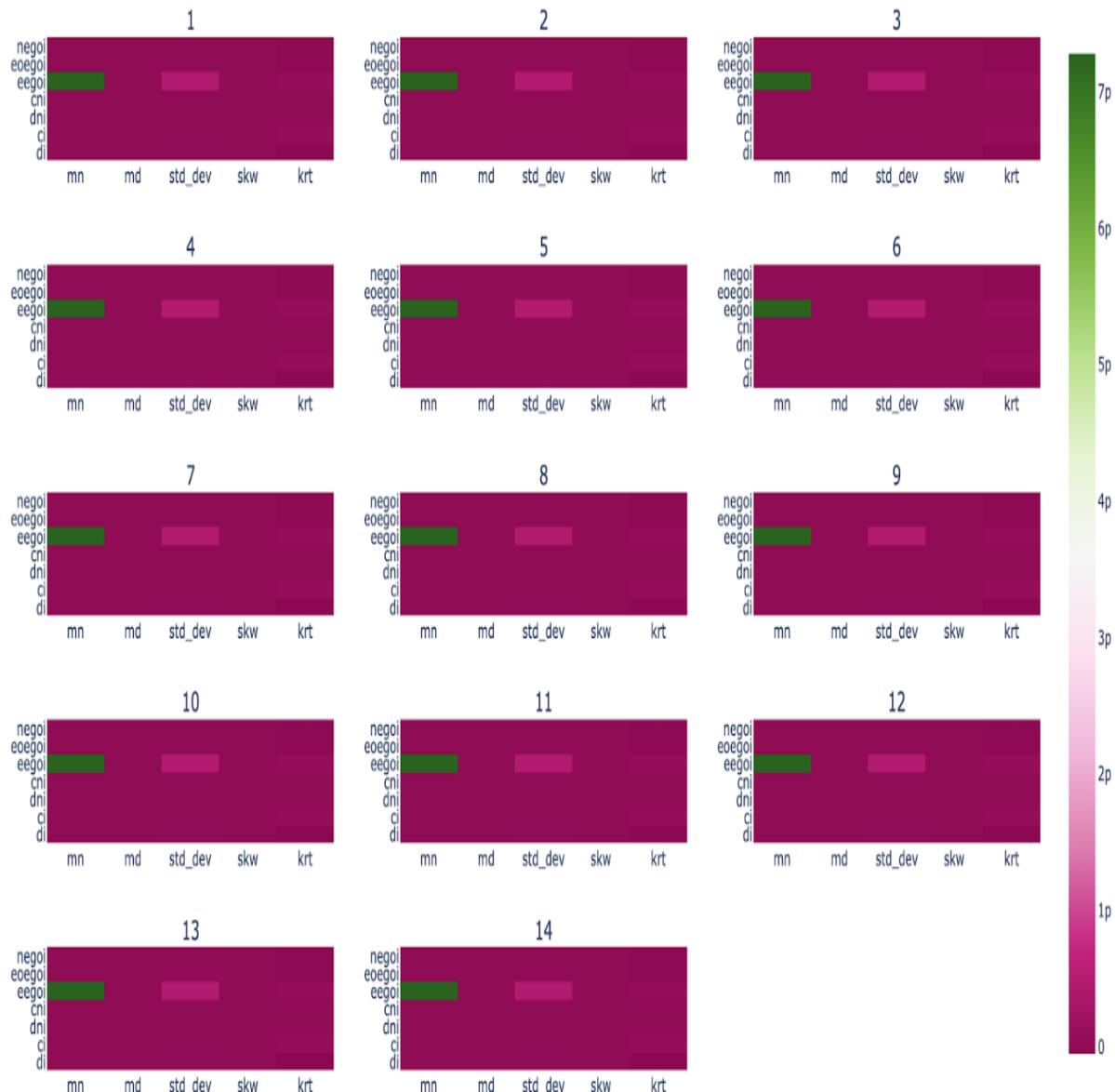
würde sich die Gesamtanzahl an Knoten annähern. Im Folgenden wird das Verhältnis der Features zum Durchschnitt dargestellt.

Man erkennt gut, dass die Features der einzelnen Graphen identisch sind, weshalb die Graphen denselben Threshold haben und somit den Wert 0 teilen.

todo: ToDo: Warum ist die Einheit p vorhanden? Da überall der gleiche Grünton beim gleichen Feature ist, entspricht dies eigentlich dem Durchschnitt und müsste ebenfalls den Wert 0 annehmen ? überprüfen

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: Struktur - Subsubsection

**Abb. 4.4:** todo: Platzhalter: Heatmap 1

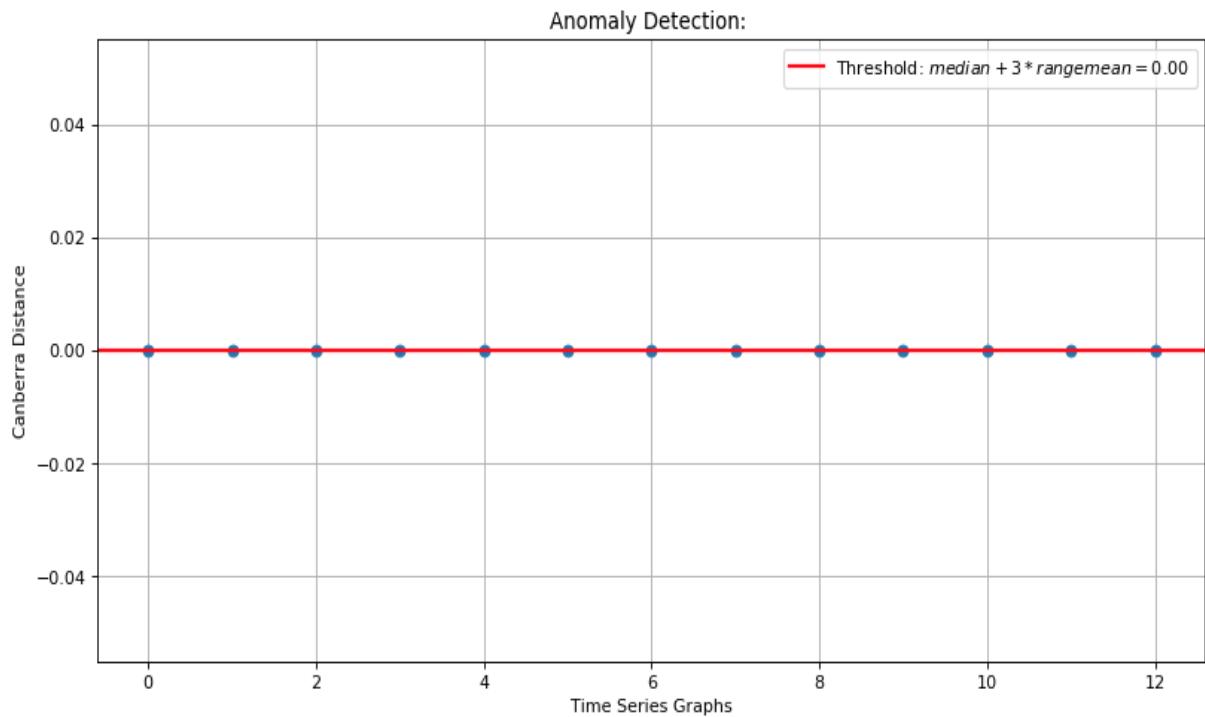


Abb. 4.5: todo: Platzhalter: Anomaly 1

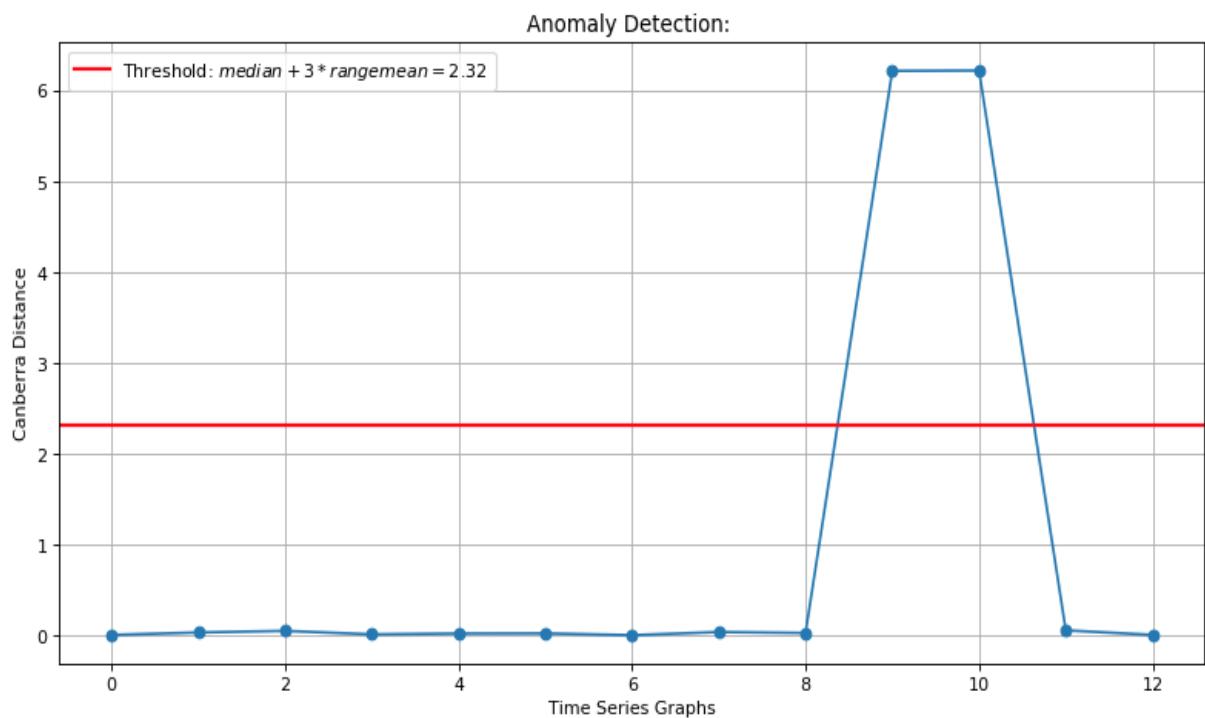


Abb. 4.6: todo: Platzhalter: Anomaly 2

Netzwerk

Enron

- Ausreißer werden erkannt
- todo: Dauer Berechnung

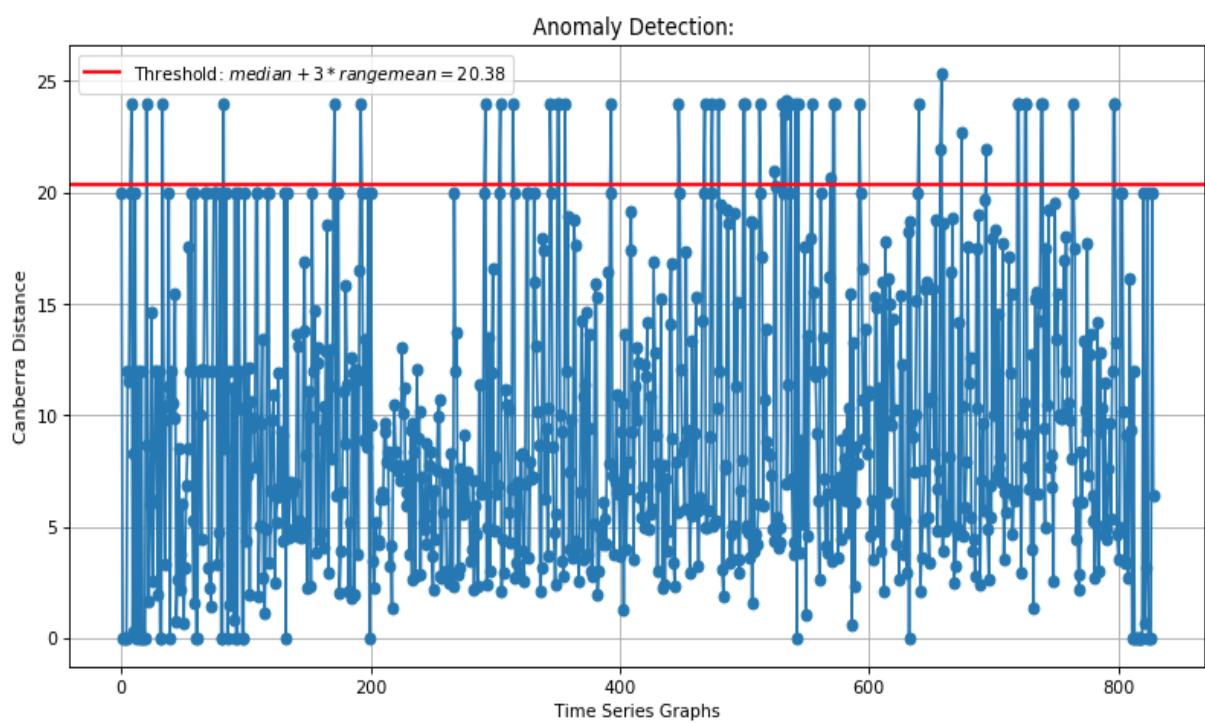


Abb. 4.7: todo: Platzhalter: Anomaly 3

Outlier dates enron

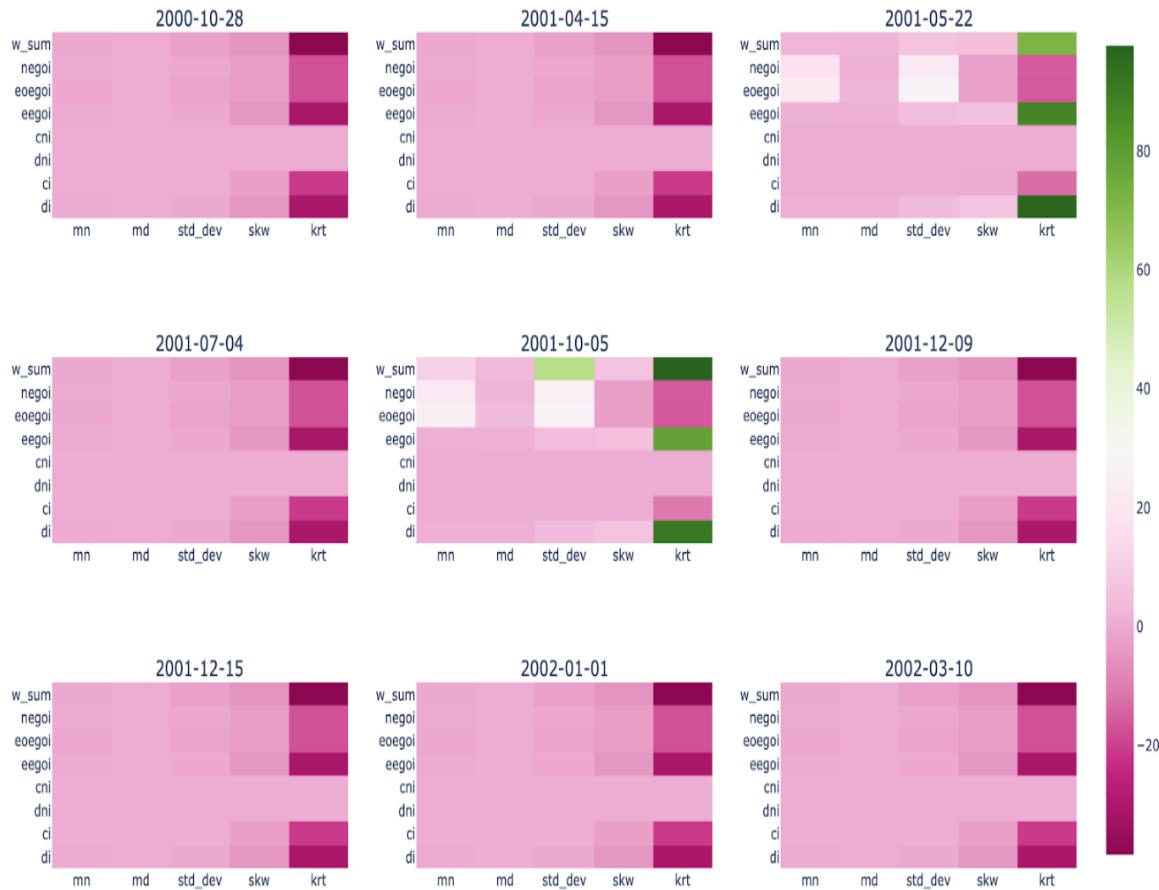


Abb. 4.8: todo: Platzhalter: heatmap 3

Darpa

- Ausreißer werden wahrgenommen
- Dauer Berechnung: 11150.692071 seconds

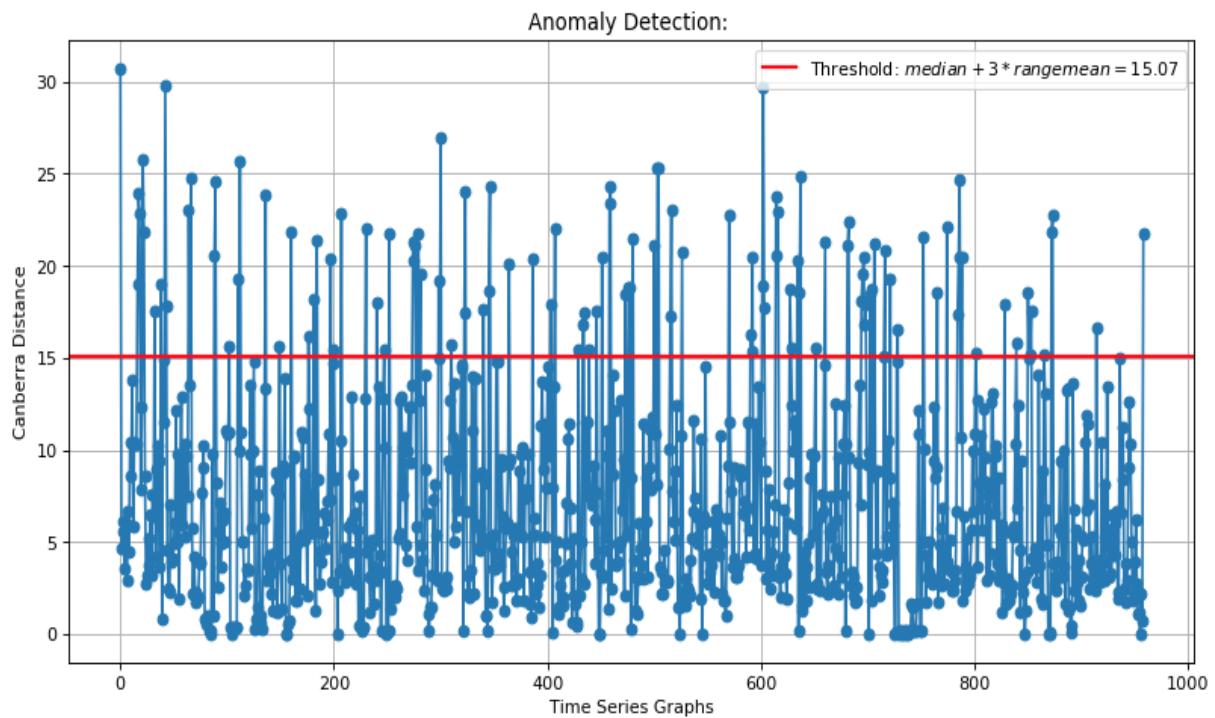


Abb. 4.9: todo: Platzhalter: Anomaly 4

```
112
11150.692071 seconds
u_limit 15.070510845359282

[17, 18, 22, 43, 89, 275, 276, 323, 346, 434, 459, 477, 504, 516, 591, 592, 602, 603, 615, 616, 628, 637, 682, 695, 696, 697, 698, 785, 786, 853, 873]
```

Abb. 4.10: todo: Platzhalter: Dauer 4

Zeitreihendaten

Eindimensional

- Ausreißer werden wahrgenommen
- Dauer Berechnung: 2991.50926781 seconds
- Ausreißer werden wahrgenommen
- Dauer Berechnung: 3691.66552711 seconds
- Ausreißer werden wahrgenommen
- Dauer Berechnung: 3945.95857 seconds
- Ausreißer werden wahrgenommen
- Dauer Berechnung: 3117.28531003 seconds

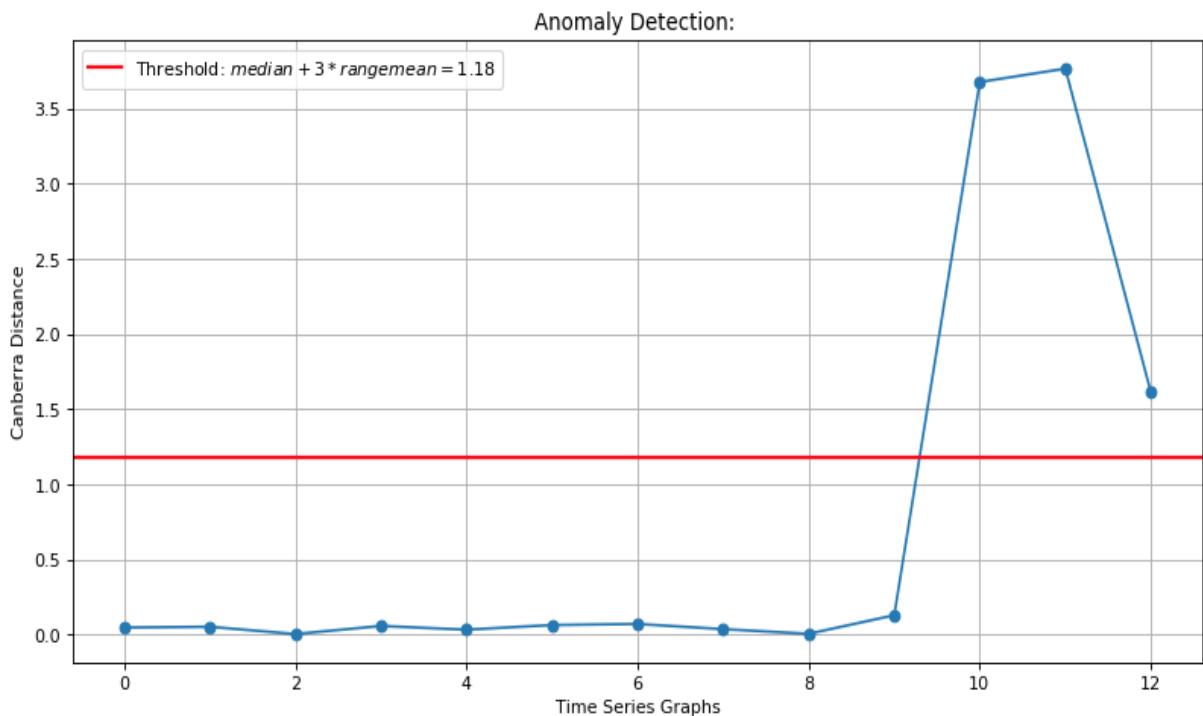


Abb. 4.11: todo: Platzhalter: Sequence Change

```

2
2991.50926781 seconds
u_limit 1.176035680230644

[11, 12]

```

Abb. 4.12: todo: Platzhalter: Sequence Change

- Ausreißer werden wahrgenommen
- Dauer Berechnung: 3768.19052601 seconds

Mehrdimensional Probleme

- es werden Knoten generiert die gar nicht existieren im Datensatz

4.2.4 Optimierte Implementierung des Algorithmus

Unter Verwendung der Netsimile Implementierung aus ??, benötigte die Ausführung des Algorithmus teilweise bis zu 30 Minuten. Aus diesem Grund wurde der Algorithmus von uns neu implementiert. Die Laufzeit konnte dabei auf wenige Sekunden reduziert werden, indem keine Graphen Bibliothek für die Implementierung verwendet wurde. Das heißt die Netzwerke der

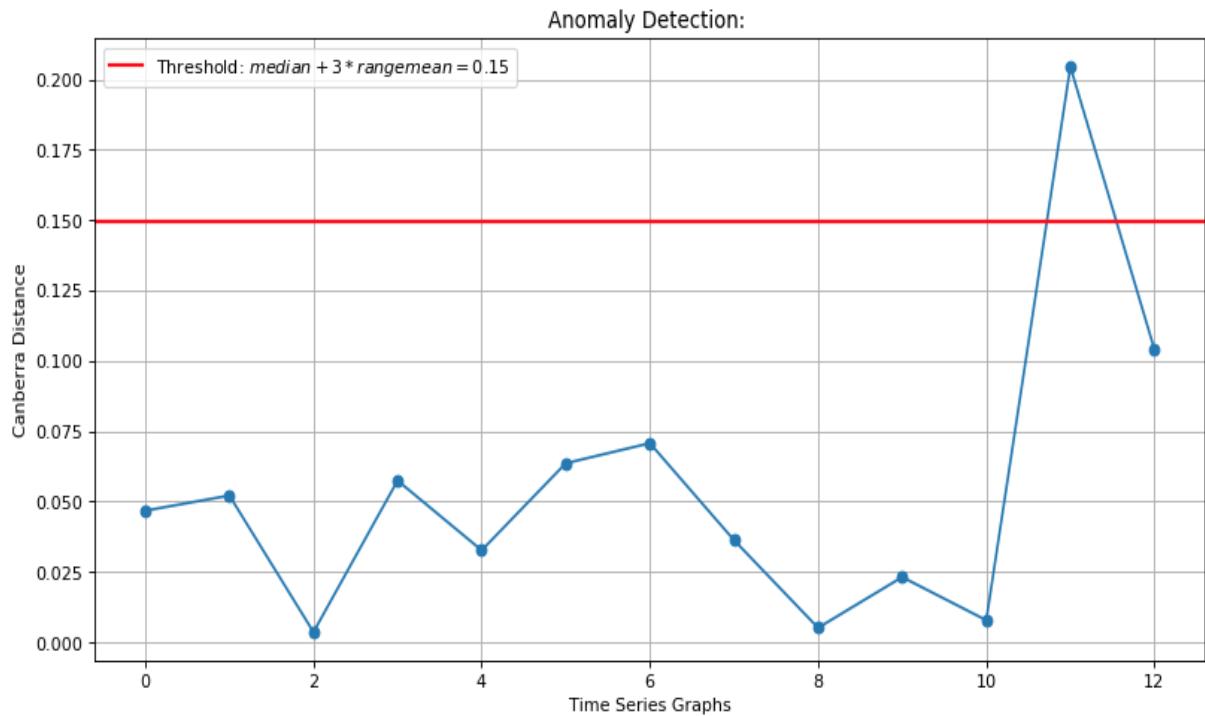


Abb. 4.13: todo: Platzhalter: Peaks

```

1
3691.66552711 seconds
u_limit 0.14978517221349386

[]

```

Abb. 4.14: todo: Platzhalter: Peaks

Zeitreihen werden nicht in ein Graphen Objekt umgewandelt, sondern als Adjazenzmatrix gespeichert. Dadurch können die Features deutlich effizienter berechnet werden. Des Weiteren wurden einige Features neu eingeführt und andere entfernt, sodass lediglich Features verwendet werden die für Fully Connected Graphen geeignet sind. So hat beispielsweise das Feature $|E_{ego(i)}|$ keine Aussagekraft in einem Fully Connected Netzwerk, da jeder Knoten die gleiche Anzahl Kanten in seinem Ego Netzwerk aufweist. Deshalb wurden folgende Features verwendet: **todo: Formeln ändern**

$|E_{ego(i)}^{\circ}|$
Arithmetisches Mittel der Kantengewichte in $ego(i)$.

$|N(ego(i))|$
Geometrisches Mittel der Kantengewichte in $ego(i)$.

$|E_{ego(i)}^{\circ}|$

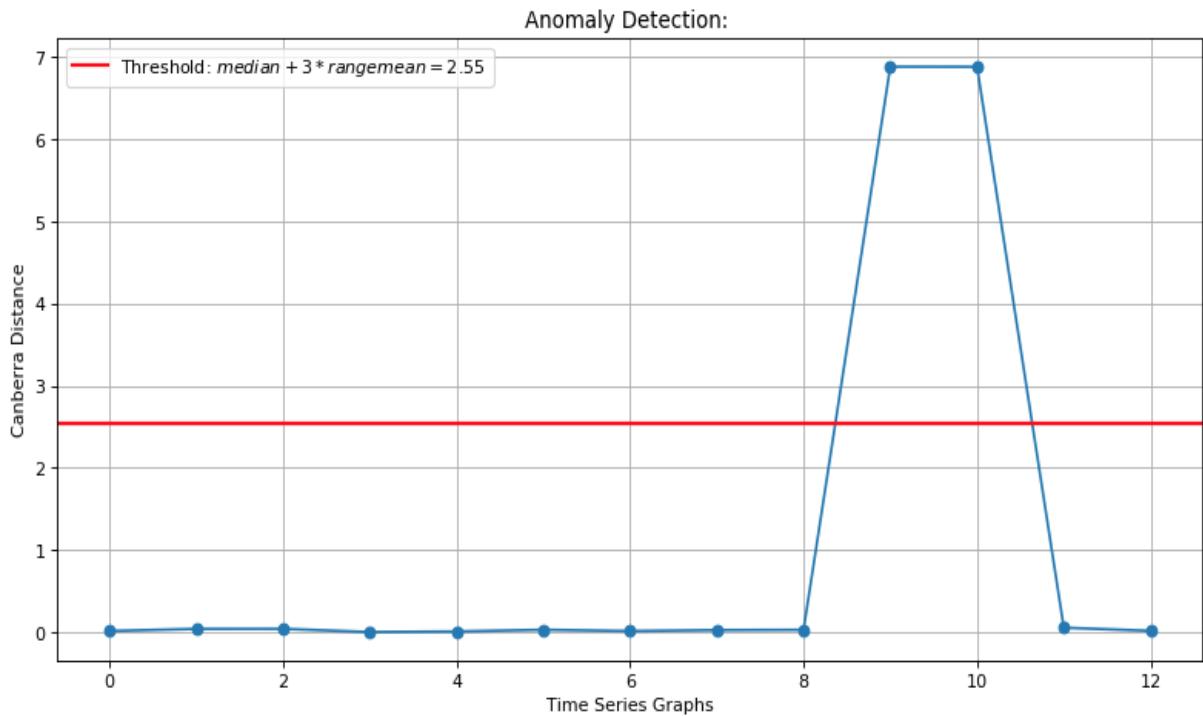


Abb. 4.15: todo: Platzhalter: No Jump

```

2
3945.95857 seconds
u_limit 2.5469650158210424

[10]

```

Abb. 4.16: todo: Platzhalter: No Jump

Geometrisches Mittel 10 Prozent der höchsten Kantengewichte in $ego(i)$.

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

Geometrischer Mittel 20 Prozent der höchsten Kantengewichte in $ego(i)$.

Von diesen Features wurde dann auch den Median, den Mittelwert, die Standardabweichung, die Schiefe, sowie die Kurtosis berechnet. **todo: Bin mir nicht sicher zu welchen Elementen die Canberra 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. 4.1 zu sehen.

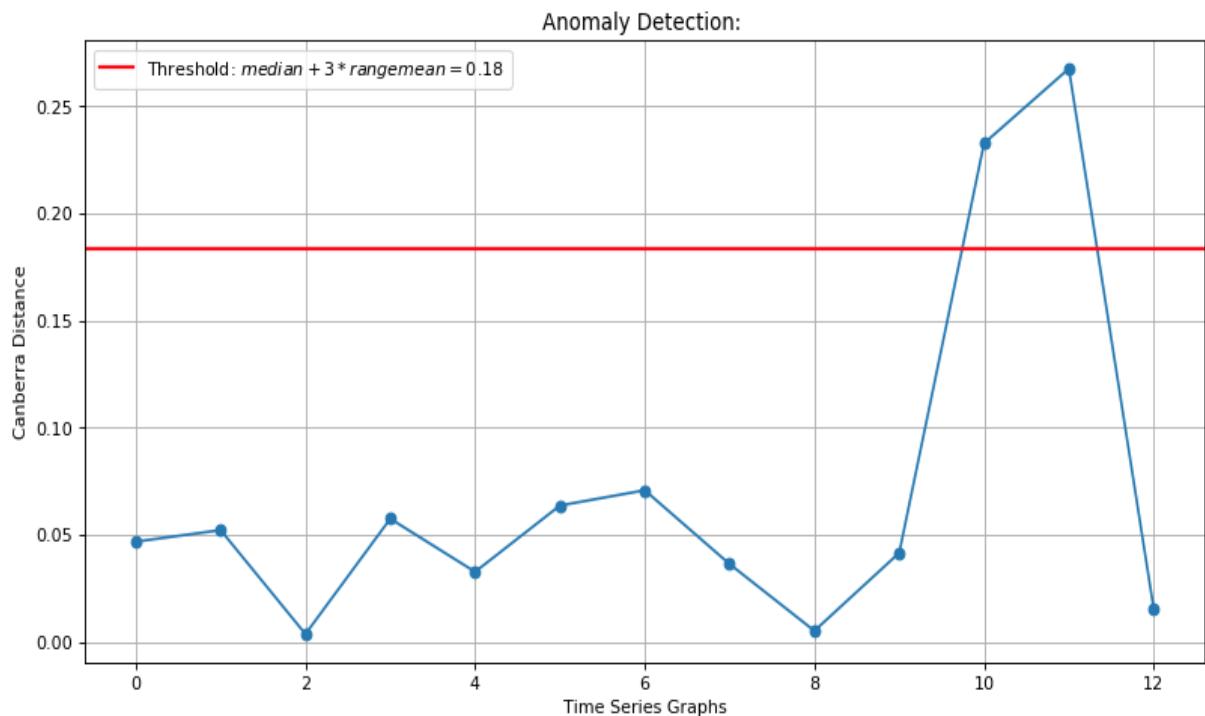


Abb. 4.17: todo: Platzhalter: Increase Noise

```

2
3117.28531003 seconds
u_limit 0.18321387952161292

[11]

```

Abb. 4.18: todo: Platzhalter: Increase Noise

Tab. 4.1: Parameter Netismile Zeitreihen

| Parameter | Beschreibung |
|--------------|---|
| Periodizität | Wie in Kap. 3.1 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 Kap. 4.2.4 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. |

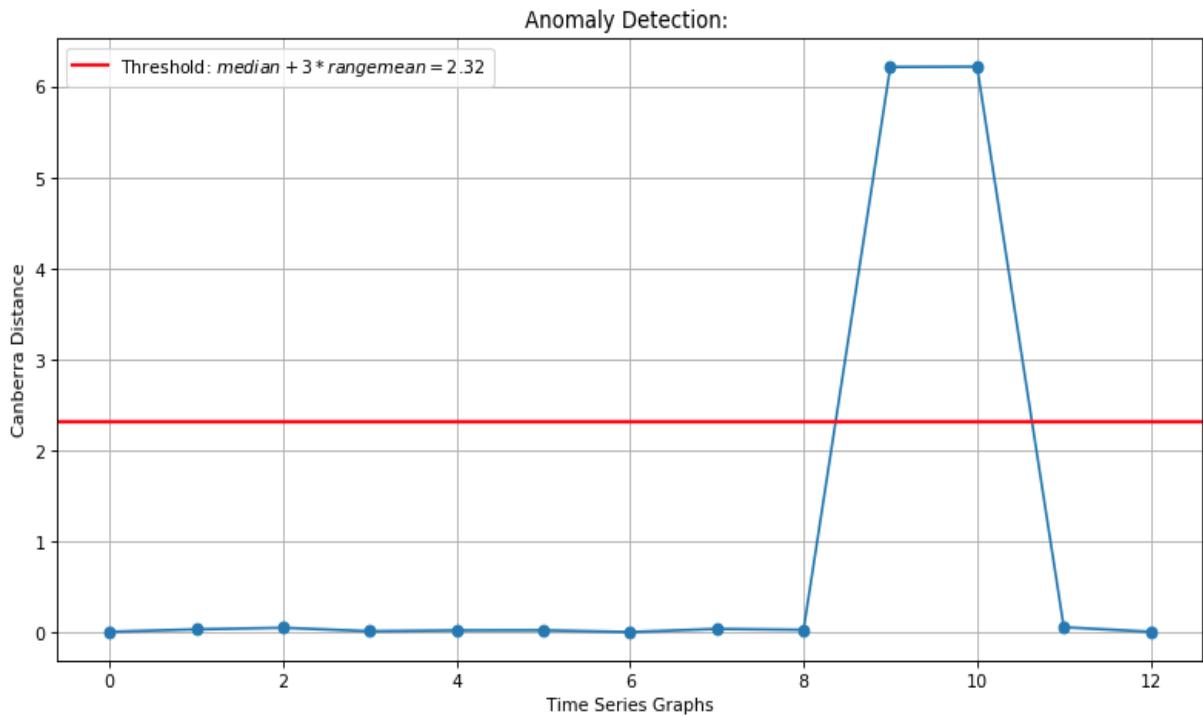


Abb. 4.19: todo: Platzhalter: Flatmiddle

```

2
3768.19052601 seconds
u_limit 2.315237566494818

[10]

```

Abb. 4.20: todo: Platzhalter: Flatmiddle

Ausreißer-Erkennung in Zeitreihen

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 Ausreißererkennung 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. 4.1 beschrieben.

Tab. 4.2 zeigt die Ergebnisse der Tests. Es ist zu erkennen, dass die Qualität der Ausreißererkennung 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

Tab. 4.2: Netsimile Time Series Performance

| 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 | **** | *** |

kontinuierlichen Zunahme der Amplitude lediglich der Anfang des Ausreißers detektiert. Aus diesem Grund wurde eine Bewertung mit drei Sternen vergeben. Bei der Betrachtung der Graphiken in [Kap. A.1](#) und [Kap. A.2](#) 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. 4.2](#) 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.

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. todo: Related Work: Sedanspot, RHSS

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. 3, 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.

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. 3, S. 3]

todo: chi-squared

Erkennung von Mikrocluster

4.3.2 Ausreißer-Erkennung in Graphen

todo: ausformulieren, bilder einfügen - DARPA und ENRON Datensätze - Jumpsup Datensatz von Marcus

-DARPA AUC 91-ENRON identisch mit SEDANSPOT-labels außer September 2000

Schwierigkeit geeignete Datensätze zu finden, dazu gibt es ein Paper

Wenn man die Anomalyscores als gewichtete nimmt, kommen Graphen in Networkx raus in denen man die anomalous nodes identifizieren kann dabei sollten es Edges sein..

Einführung

todo: Stichworte sammeln

4.3.3 Ausreißer-Erkennung in Zeitreihen

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, das 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, das 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. 4.24). 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 das 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.

Teilweise führen die Ausreißer auch zu besonders wenigen Kanten (vgl. Abb. 4.25). 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). todo: Noch Pfeil in Graphik einfügen Des weiteren ergibt sich durch die Ausreißer eine leicht veränderte Anzahl an Kanten in dem Abschnitt mit dem Ausreißer (vgl.

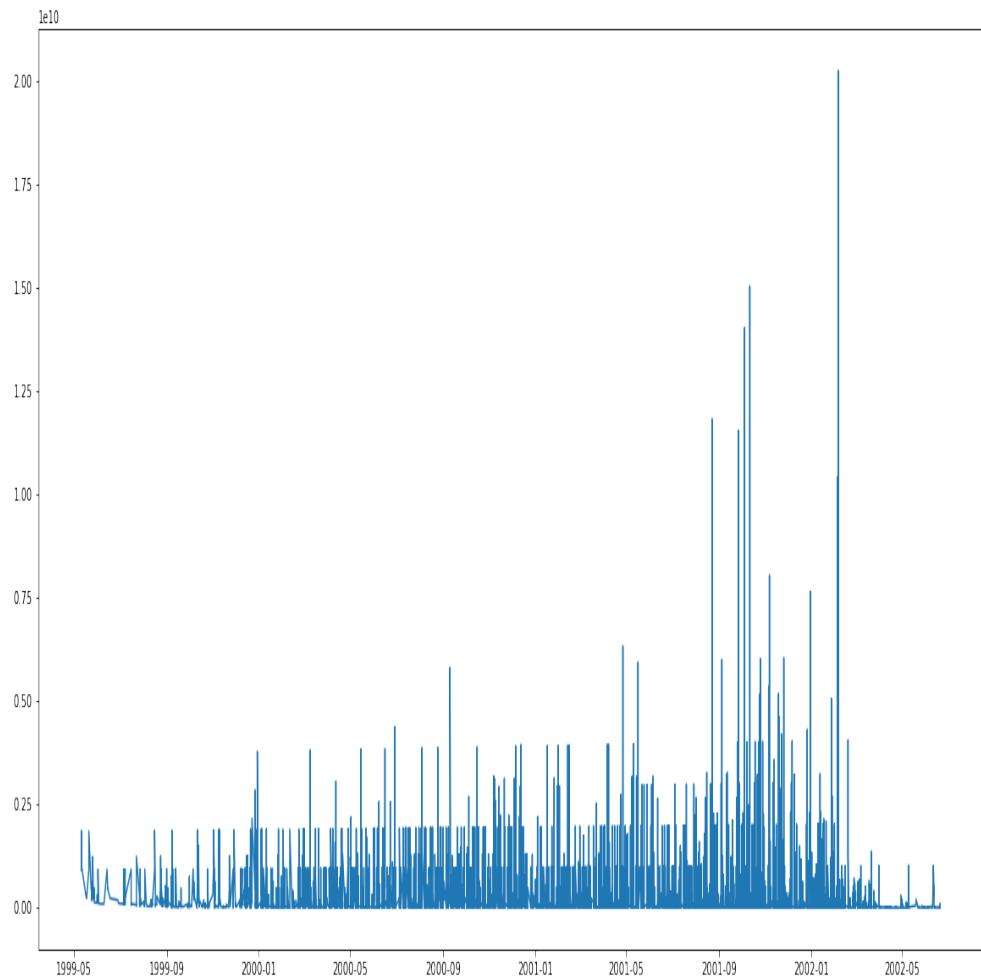


Abb. 4.21: todo: Platzhalter: Enron

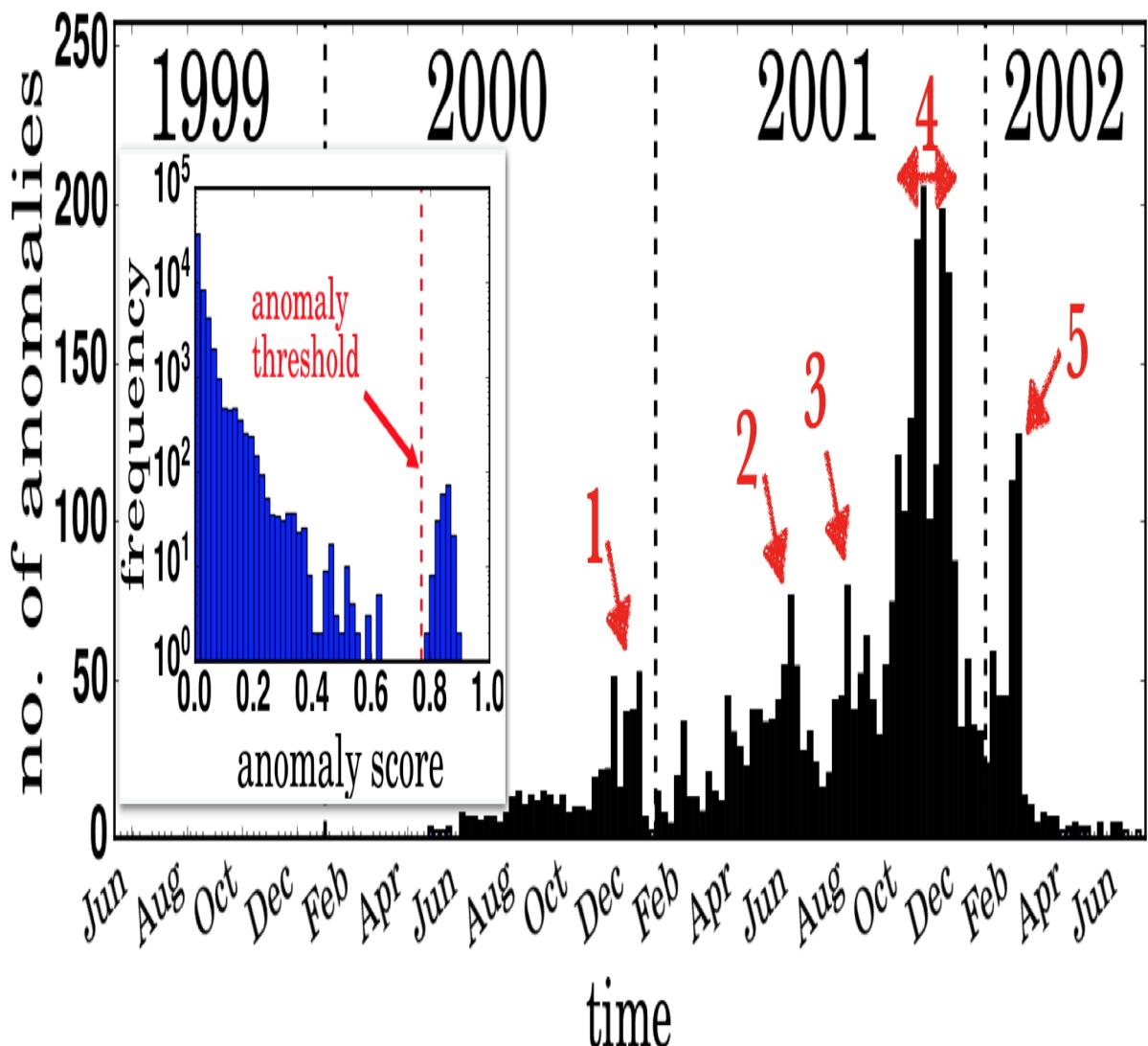


Abb. 4.22: todo: Platzhalter: Enron

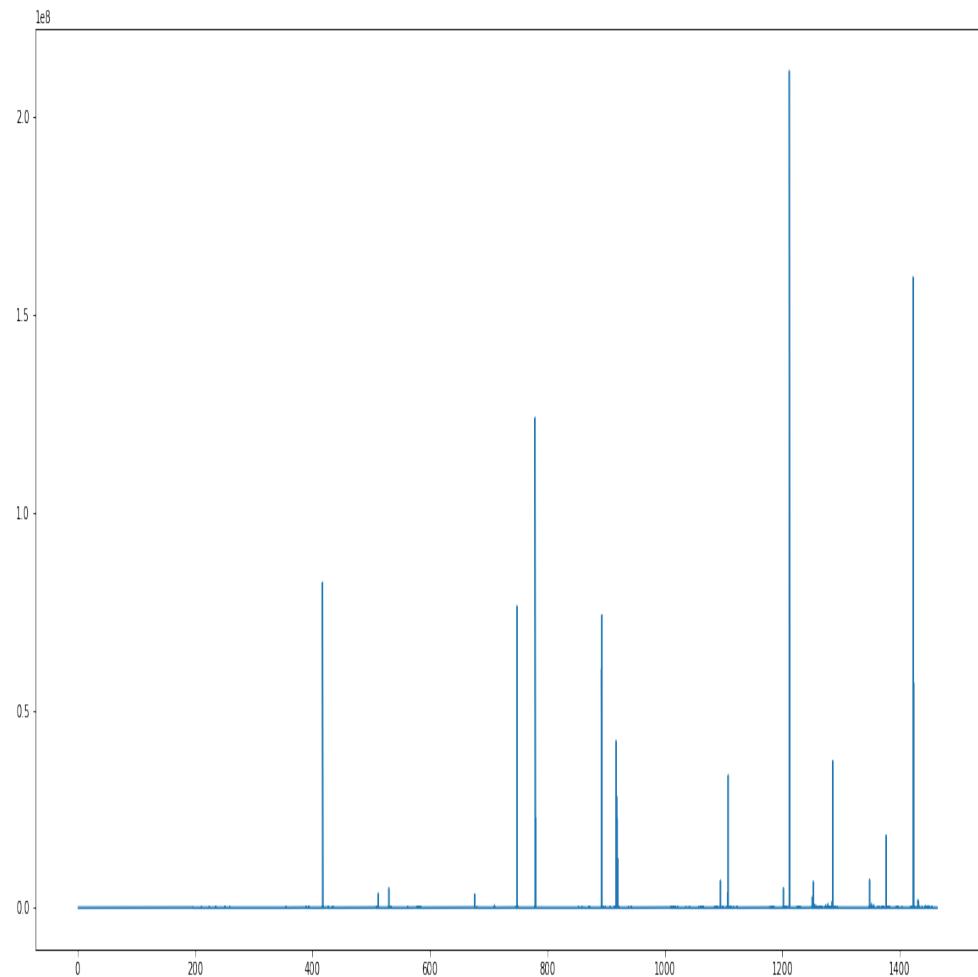


Abb. 4.23: todo: Platzhalter: Darpa

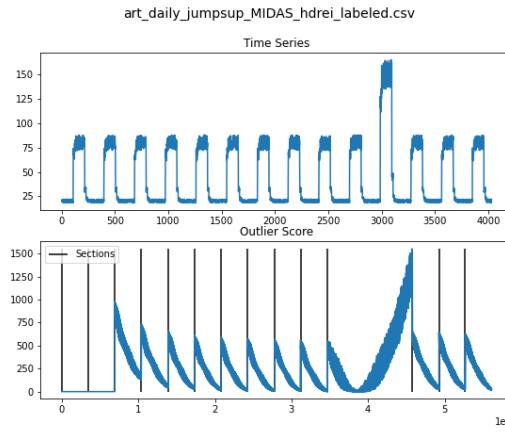
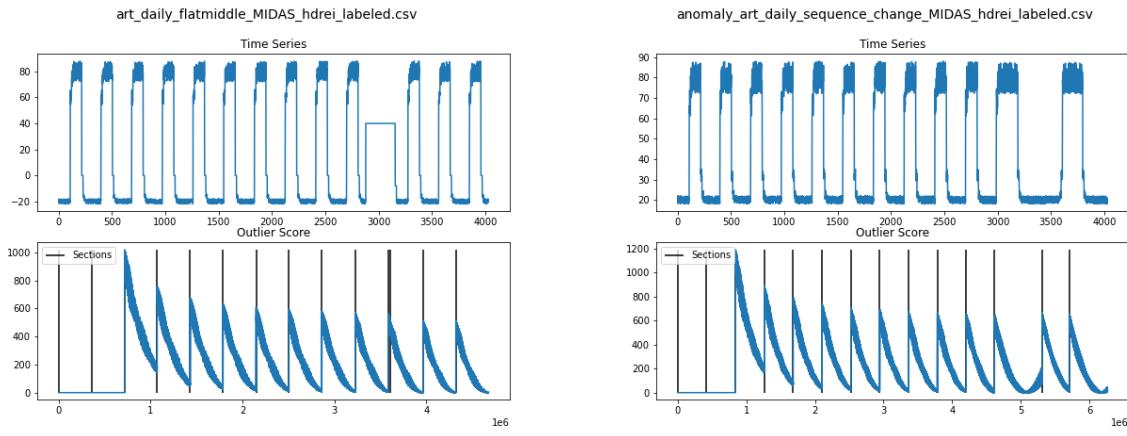


Abb. 4.24: MIDAS Algorithmus angewandt auf Zeitreihe mit einer erhöten Amplitude.

Abb. 4.25). Die Abweichungen sind jedoch so gering, dass es nicht zu einem starken Anstieg des Ausreißer Scores führt.



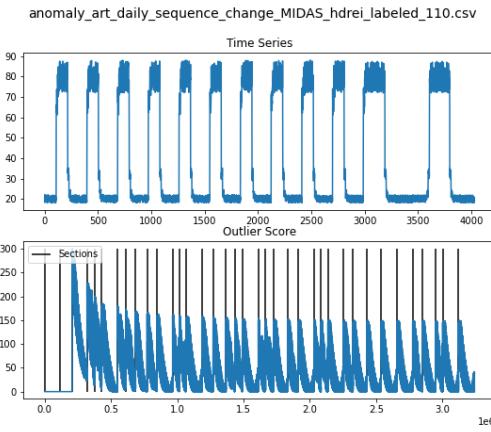
(a) Zeitreihe mit Zyklus Aussetzter

(b) Zeitreihe mit Frequenzänderung

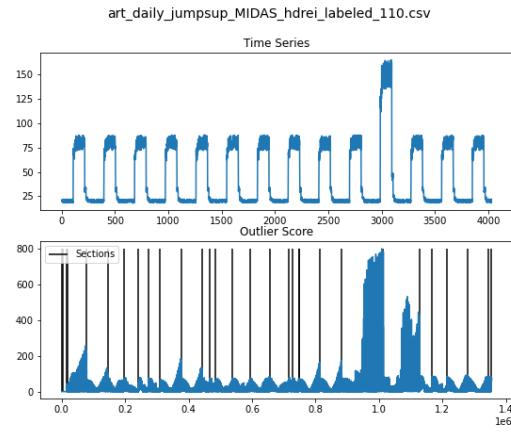
Abb. 4.25: Ausreißer Erkennung in Zeitreihen MIDAS Algorithmus

Es wurden außerdem Tests durchgeführt um zu untersuchen, wie sich der Algorithmus bei veränderter Fenstergröße verhält (vgl. Abb. 4.26). Bei den Untersuchungen in Abb. 4.24 und Abb. 4.25 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 erbringt. 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.

In einem nächsten Schritt wurde untersucht inwiefern der MIDAS-R Algorithmus zu einer Verbesserung bei der Ausreißer Erkennung beitragen kann (vgl. Abb. 4.27). Der MIDAS-R Algorithmus berücksichtigt bei der Berechnung des Ausreißer Scores für den aktuellen Abschnitt auch die Daten aus der jüngsten Vergangenheit(vorangegangene Abschnitte). Aus diesem Grund erhofften



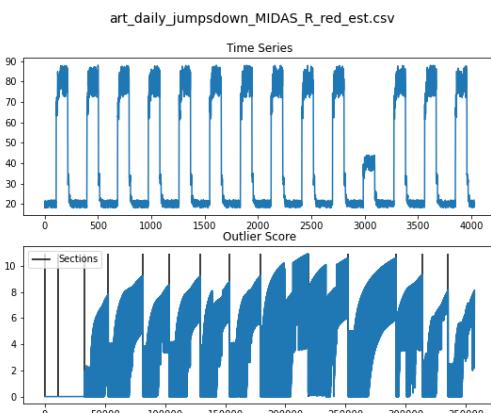
(a) Zeitreihe mit einer Frequenzänderung



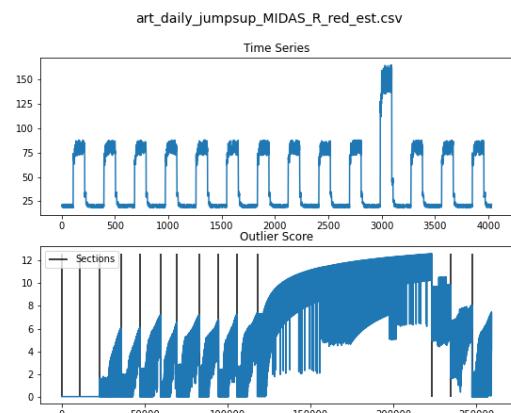
(b) Zeitreihe mit erhöhter Amplitude

Abb. 4.26: Ausreißer Erkennung Zeitreihen MIDAS Algorithmus Fenstergröße 110

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 auch die durch den MIDAS-R Algorithmus eingeführten Features zu keiner Verbesserung der Ergebnisse geführt haben. **todo: Vielleicht könnte eine Verbesserung erreicht werden wenn andere Features eingeführt werden würden.**



(a) Zeitreihe mit geringerer Amplitude



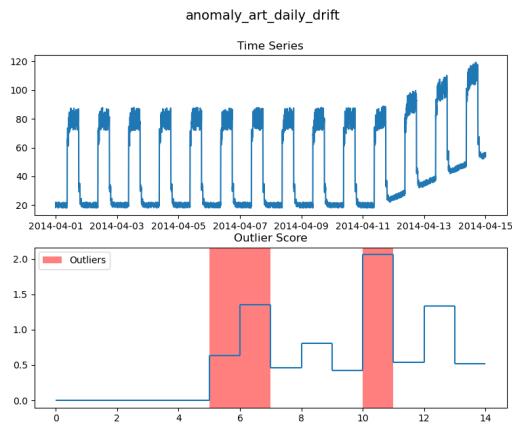
(b) Zeitreihe mit erhöhter Amplitude

Abb. 4.27: Ausreißer Erkennung Zeitreihen MIDAS-R

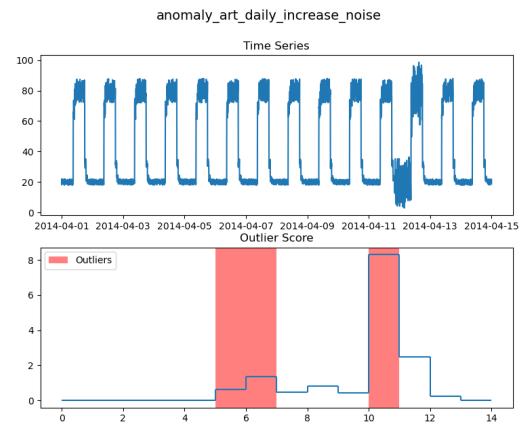
A Netsimile

A.1 Eindimensionales Signal

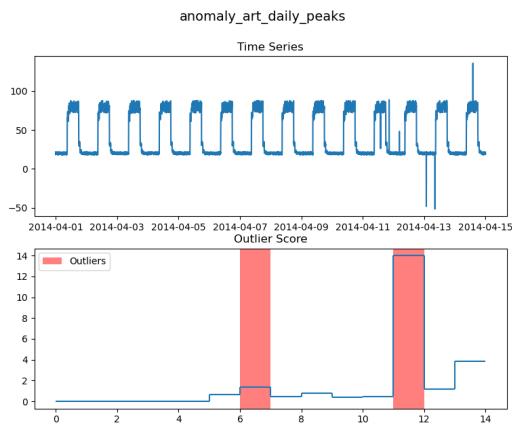
todo: Wrong picture for daily peaks. Change that the sixed element is not always an outlier



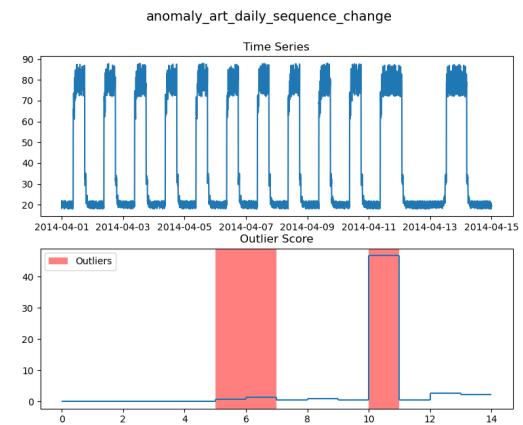
(a) Caption for sub-figure1



(b) Caption for sub-figure1



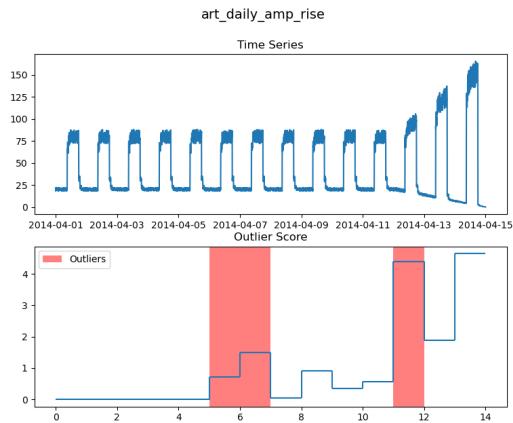
(c) Caption for sub-figure1



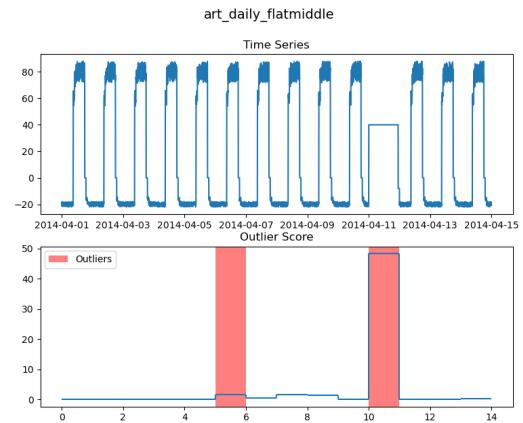
(d) Caption for sub-figure1

A.2 Zweidimensionales Signal

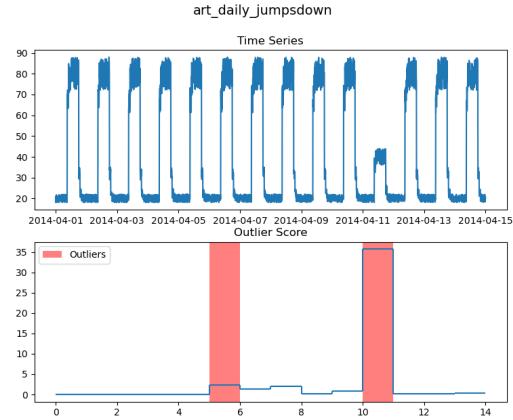
todo: Wrong picture for daily peaks. Change that the sixed element is not always an outlier



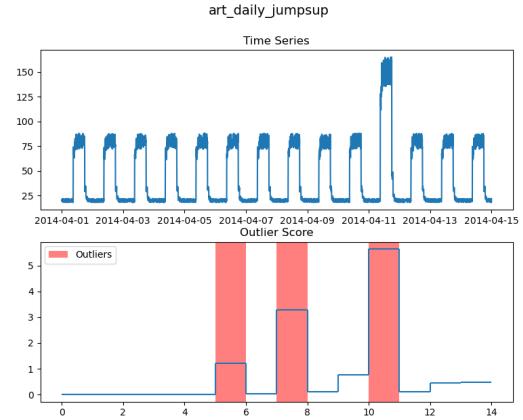
(e) Caption for sub-figure1



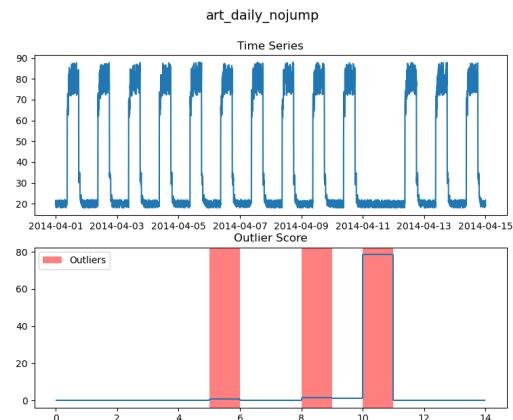
(f) Caption for sub-figure1



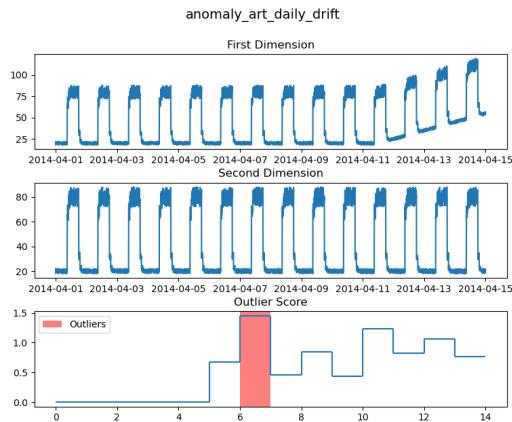
(g) Caption for sub-figure1



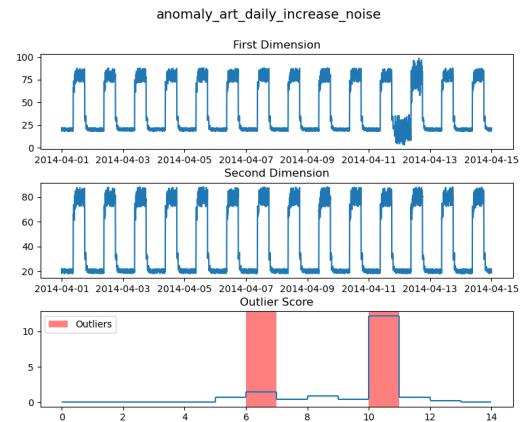
(h) Caption for sub-figure1



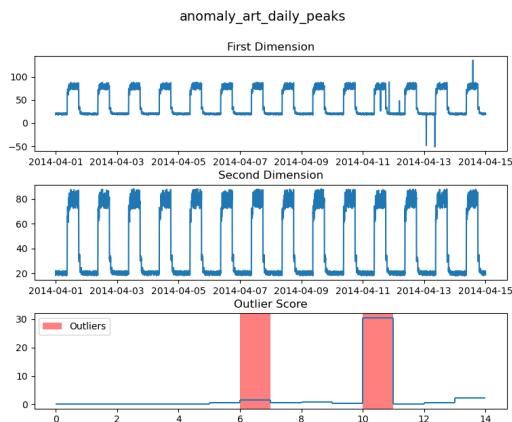
(i) Caption for sub-figure1



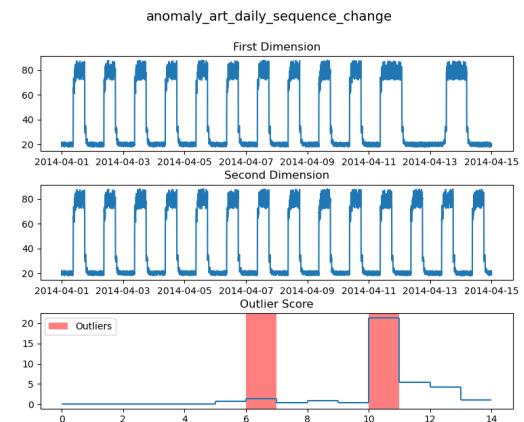
(a) Caption for sub-figure1



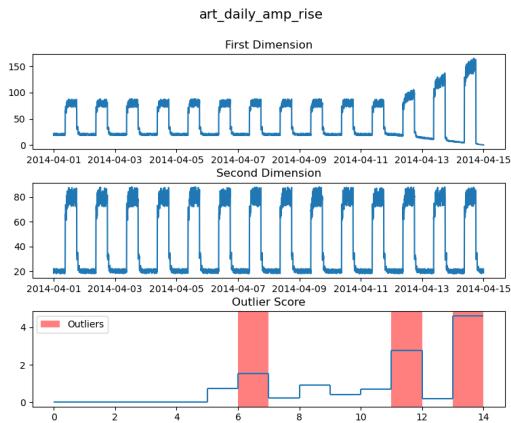
(b) Caption for sub-figure1



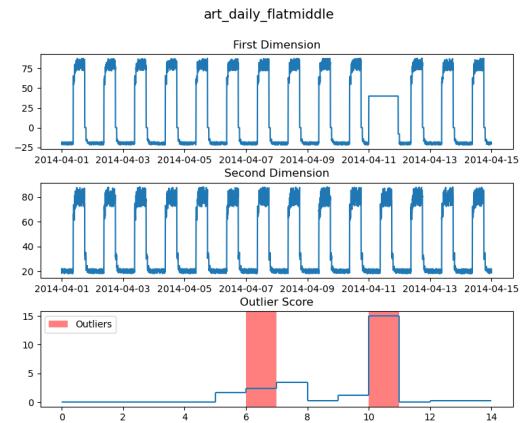
(c) Caption for sub-figure1



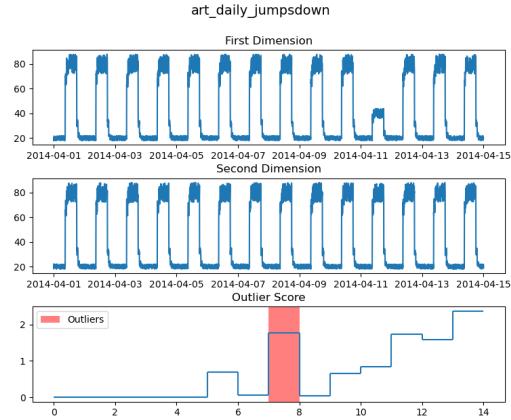
(d) Caption for sub-figure1



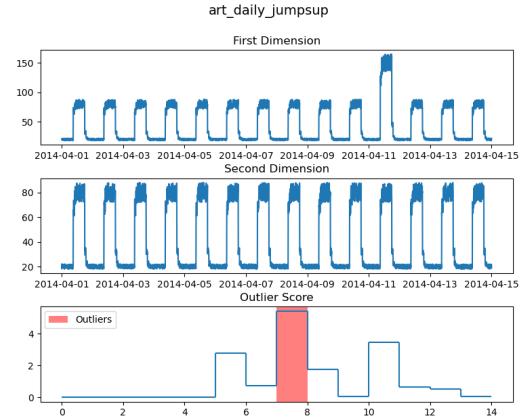
(e) Caption for sub-figure1



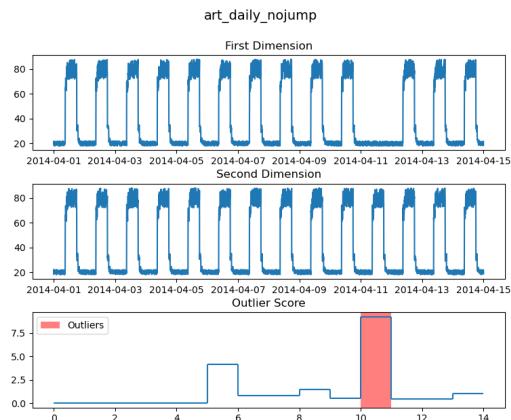
(f) Caption for sub-figure1



(g) Caption for sub-figure1

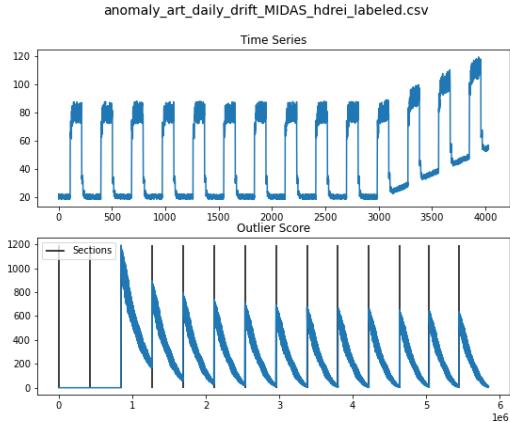


(h) Caption for sub-figure1

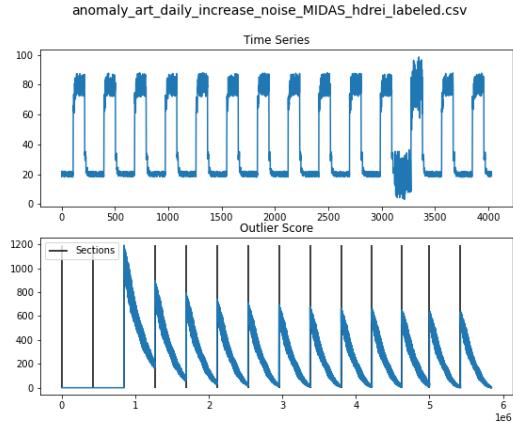


(i) Caption for sub-figure1

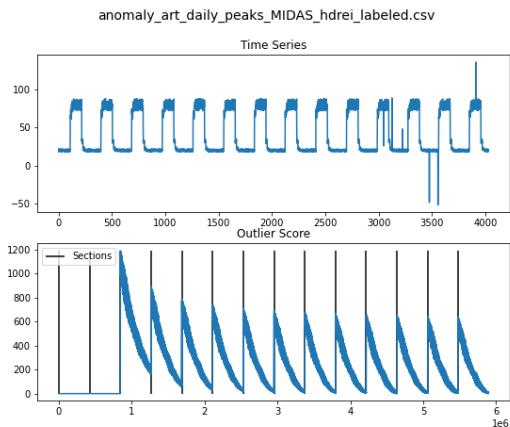
B Midas



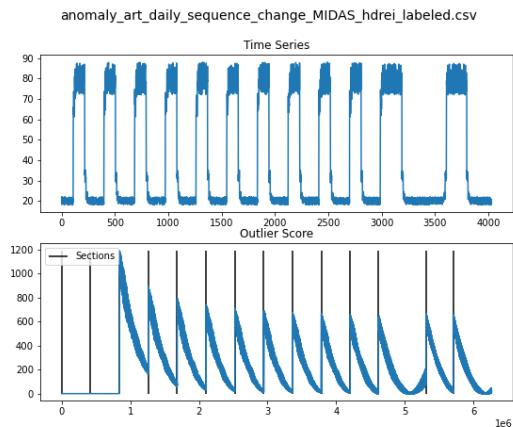
(a) Caption for sub-figure1



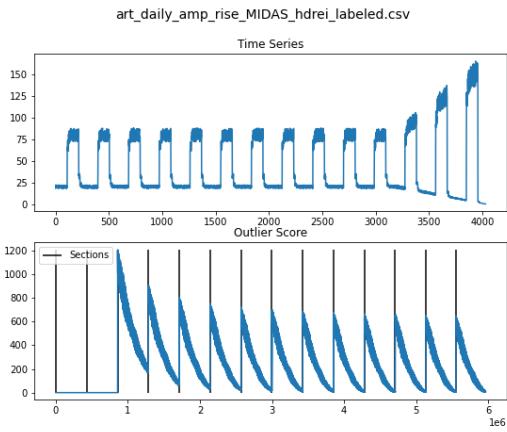
(b) Caption for sub-figure1



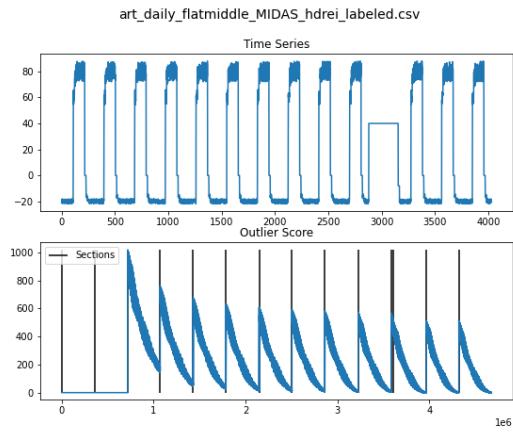
(c) Caption for sub-figure1



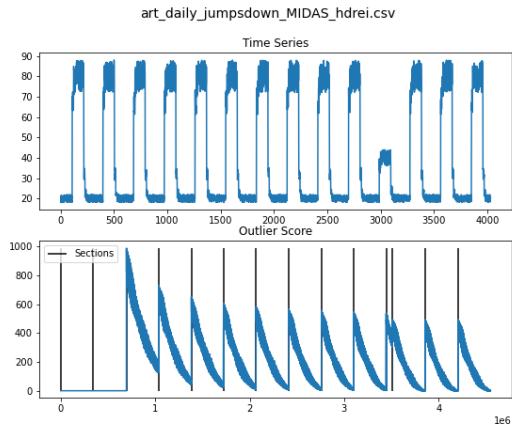
(d) Caption for sub-figure1



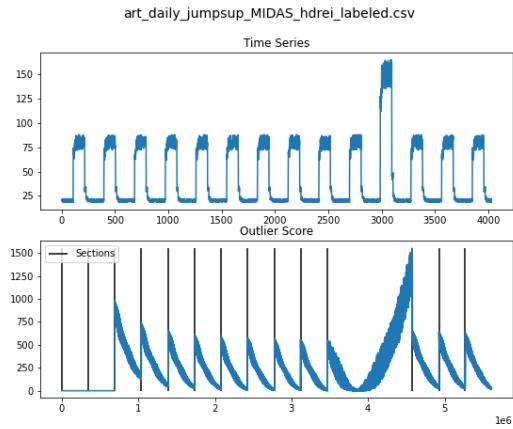
(e) Caption for sub-figure1



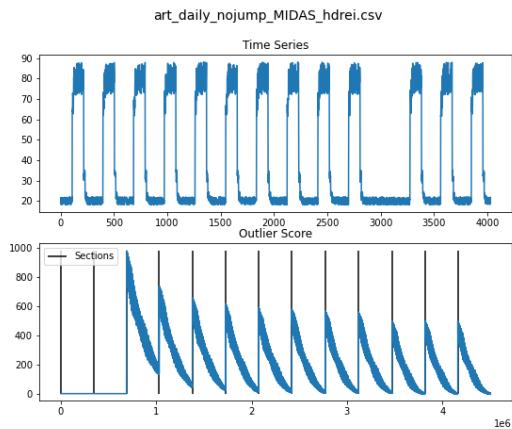
(f) Caption for sub-figure1



(g) Caption for sub-figure1



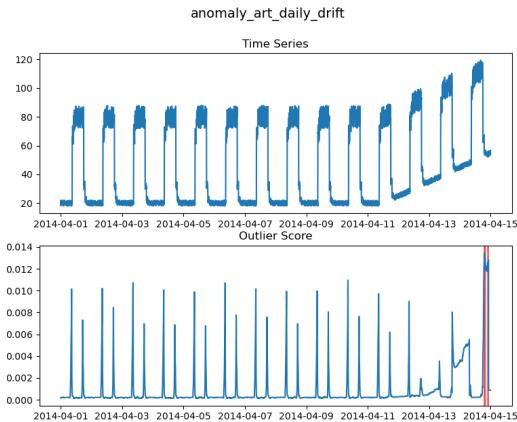
(h) Caption for sub-figure1



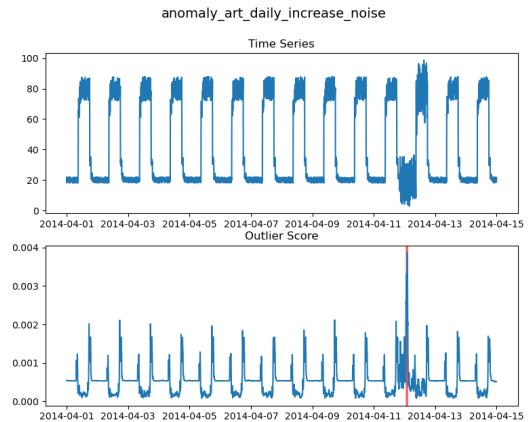
(i) Caption for sub-figure1

C Isomap

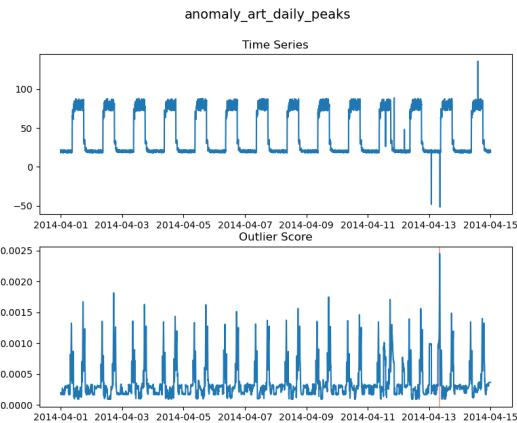
C.1 Eindimensionales Signal



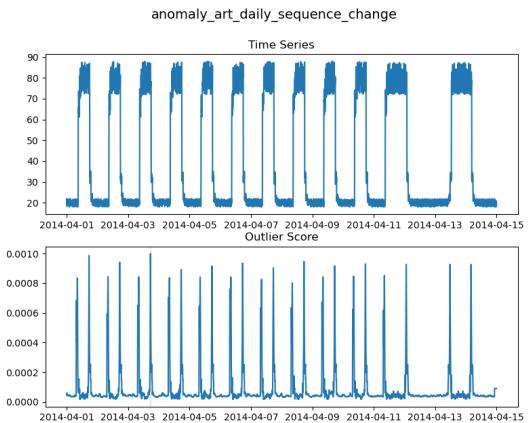
(a) Caption for sub-figure1



(b) Caption for sub-figure1

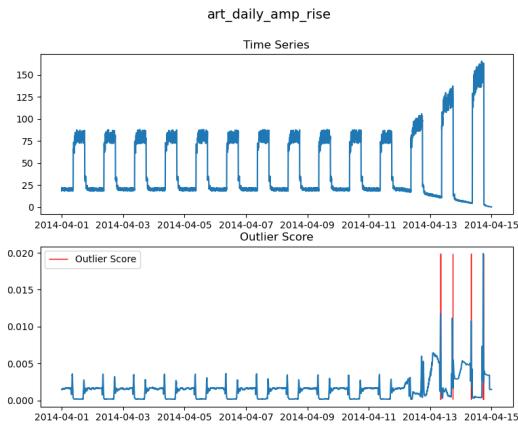


(c) Caption for sub-figure1

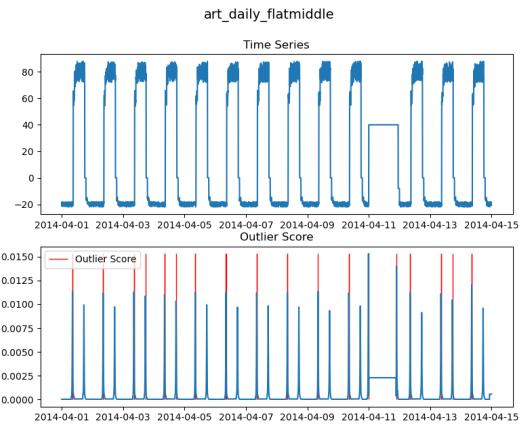


(d) Caption for sub-figure1

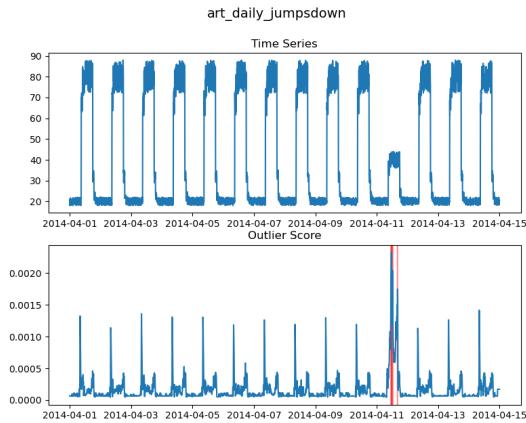
todo: In einigen Bildern fehlt die Legende. Vielleicht noch ein Paar bessere Ergebnisse zu erzielen



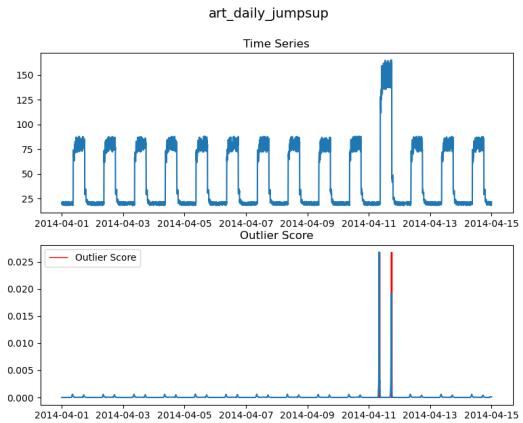
(e) Caption for sub-figure1



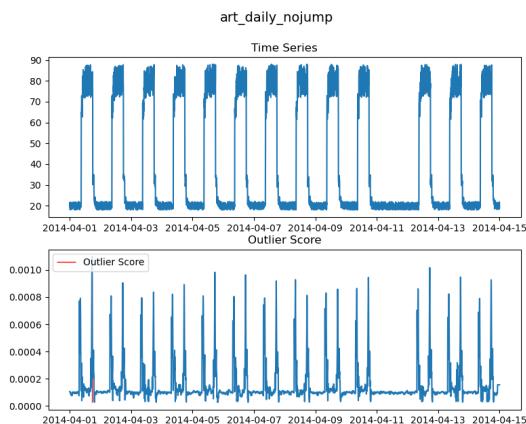
(f) Caption for sub-figure1



(g) Caption for sub-figure1



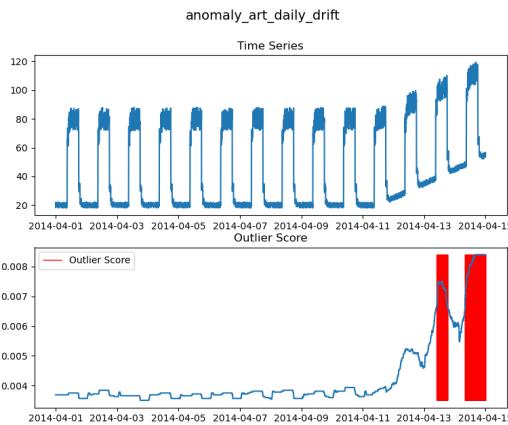
(h) Caption for sub-figure1



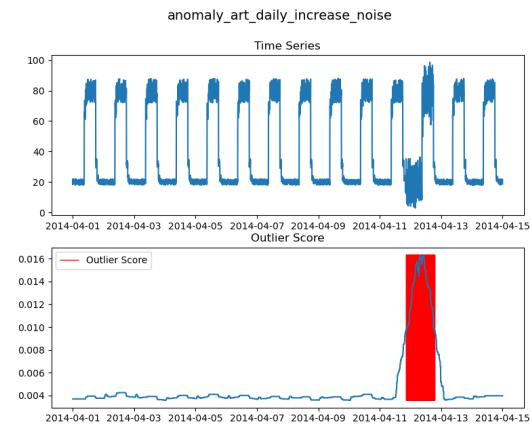
(i) Caption for sub-figure1

D Perculation

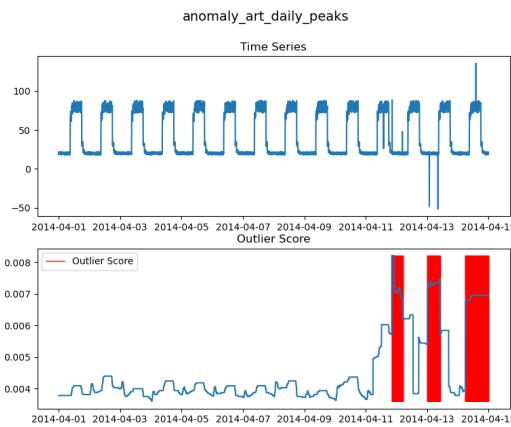
D.1 Sliding Window



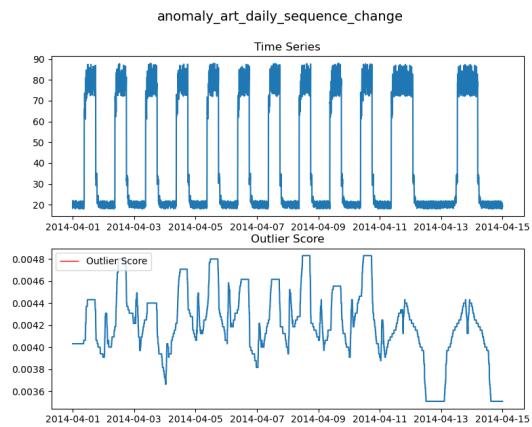
(a) Caption for sub-figure1



(b) Caption for sub-figure1

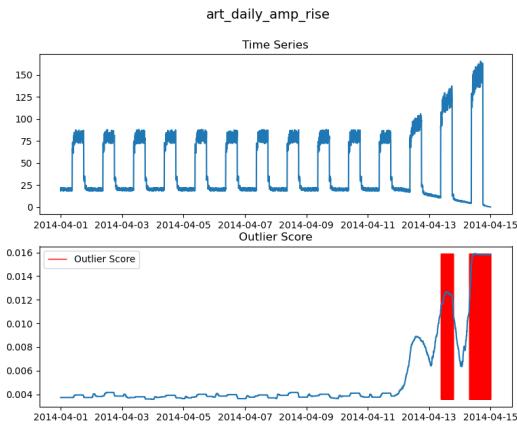


(c) Caption for sub-figure1

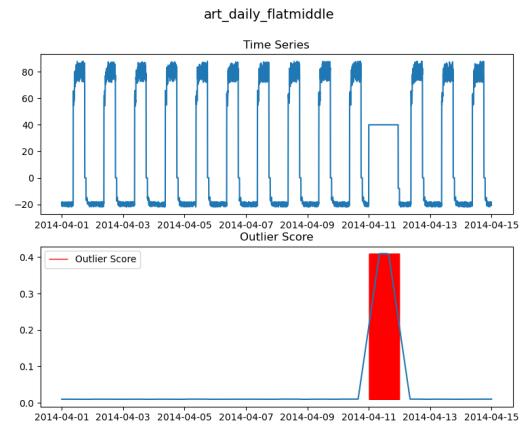


(d) Caption for sub-figure1

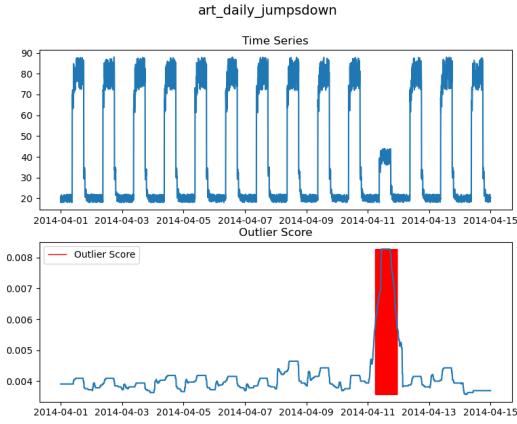
todo: Nim mir nicht sicher ob ich das ohne sliding window auch noch einfÃ¼gen soll. Vielleicht kann ich oben ja einmal einen Vergleich mit sliding window und ohne sliding window rein machen



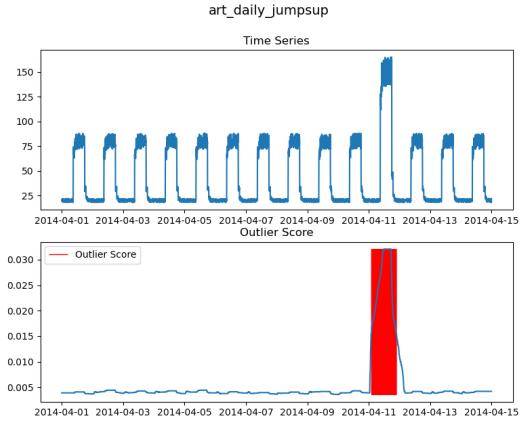
(e) Caption for sub-figure1



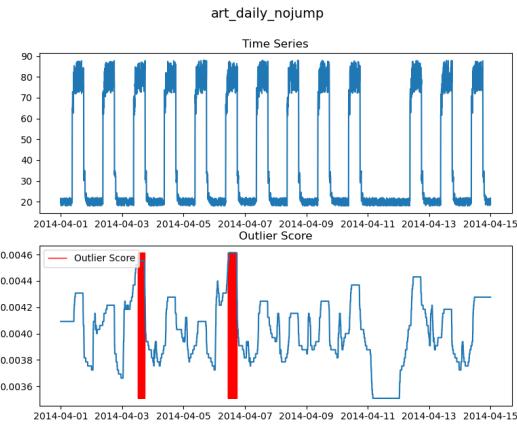
(f) Caption for sub-figure1



(g) Caption for sub-figure1



(h) Caption for sub-figure1



(i) Caption for sub-figure1

Literaturverzeichnis

- [1] 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>
- [2] 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>
- [3] 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’.
- [4] Chandola, V., Banerjee, A. and Kumar, V. [2012], ‘Anomaly detection for discrete sequences: A survey’, *IEEE Transactions on Knowledge and Data Engineering* **24**(5), 823–839.
- [5] 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.
- [6] Hawkins, S., He, H., Williams, G. and Baxter, R. [2002], Outlier detection using replicator neural networks, in Y. Kambayashi, W. Winiwarter and M. Arikawa, eds, ‘Data Warehousing and Knowledge Discovery’, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 170–180.
- [7] Munir, M., Siddiqui, S. A., Dengel, A. and Ahmed, S. [2019], ‘Deepant: A deep learning approach for unsupervised anomaly detection in time series’, *IEEE Access* **7**, 1991–2005.
- [8] Rahmani, A., Afra, S., Zarour, O., Addam, O., Koochakzadeh, N., Kianmehr, K., Alhajj, R. and Rokne, J. [2014], ‘Graph-based approach for outlier detection in sequential data and its application on stock market and weather data’, *Knowledge-Based Systems* **61**, 89–97.
URL: <https://www.sciencedirect.com/science/article/pii/S0950705114000574>
- [9] 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>
- [10] Uzun, Kielman, E. [2020], ‘Anomalie-erkennung in graphen’.