Hochschule Esslingen
University of Applied Sciences

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

Tabellenverzeichnis

Listings

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: In [8] werden sequenzielle Daten als Graph dargestellt. Dabei werden die Kantengewichte über die Euklidische Distanz zwischen verschiedenen Datenpunkten dargestellt. Anschließend werden die Knoten mithilfe des Minimum Spanning Tree Algorithmus geclustered. Dan werden über ein voting scheme Ausreißer identifiziert. Wurde auf Aktiendaten und Wetterdaten angewandt.

Ein robuster graphbasierter Algorithmus zur Erkennung und Charakterisierung von Anomalien in verrauschte 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, das 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 Transformation Zeitreihe zu Netzwerk

Ziel unseres Forschungsprojektes ist es unter anderem verschiedene Algorithmen, zur Ausreißer Erkennung in Netzwerken, auf Zeitreihendaten anzuwenden. Als erstes müssen hierzu die Zeitreihen in ein Netzwerk umgewandelt werden, dieser Schritt wird in diesem Kapitel erläutert. Je nach Ausreißer-Erkennung Algorithmus, muss die Transformation leicht unterschiedlich durchgeführt werden. Aus diesem Grund wird in Kap. 2.1, Kap. 2.2 und Kap. 2.3 erläutert wie die Umwandlung für die jeweiligen Algorithmen funktioniert.

2.1 Netsimile

Der erste Schritt der Transformation ist, die Zeitreihe in kleinere Intervalle aufzusplitten. Anschließend kann für jedes der Intervalle ein Netzwerk berechnet werden. Die Länge des Intervalls kann als Hyperparameter an den Algorithmus übergeben werden. Je nach Zeitreihe funktionieren unterschiedliche Intervallgrößen besser oder schlechter. Insofern die Zeitreihe eine Saisonalität aufweist, kann diese bestimmt und als Intervallgröße genutzt werden. todo: Überprüfen ob Saisonalität der richtige Begriff ist.
Um die einzelne Zeitintervalle in ein Netzwerk umzuwandeln, wird zunächst die Distanz zwischen den einzelnen Elementen des Zeitintervalls berechnet. Hierzu wird auf das in [vgl. 1, S. 2-3] vorgestellte Distanzmaß zurückgegriffen. Insofern für p = 2 eingesetzt wird, handelt es sich um die euklidische Distanz. Die Abstände bilden die Kantengewichte zwischen den jeweiligen Elementen im Netzwerk. Die Elemente der Zeitreihe bilden die Knoten des Netzwerks. Die Netzwerke werden intern als Adjazenzmatrizen gespeichert. Des Weiteren kann ein Element der Zeitreihe aus mehreren Werten bestehen z.B. bei multivariaten Zeitreihen.

$$D_{ij} = \left(\sum_k \left| v_k^i - v_k^j \right|^p \right)^{1/p} \tag{2.1}$$

Im nächsten Schritt müssen die Netzwerke in CSV-Dateien gespeichert werden, sodass der Netsimile Algorithmus die Daten einlesen kann. Dazu wird für jede Kante des Netzwerks eine Zeile im der Datei, mit folgendem Format generiert: Ursprungsknoten, Zielknoten, Gewichtung. Für jedes Zeitintervall muss eine einzelne CSV-Datei angelegt werden. Der Netsimile Algorithmus vergleicht...

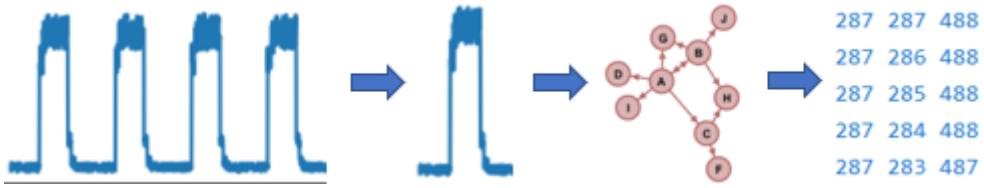


Abb. 2.1: Umwandlung einer Zeitreihe in Netzwerk

todo: Das Netzwerk aus der Grafik noch abändern

2.2 MIDAS

Die Transformation der Zeitreihe in mehrere Netzwerke funktioniert für den MIDAS Algorithmus gleich wie in Kap. 2.1. Allerdings kann der Algorithmus teilweise bessere Ergebnisse erzielen, wenn für p eine Zahl größer als zwei eingesetzt wird. Dadurch werden größere Abstände zwischen Elementen stärker gewichtet.
Außerdem erwartet der MIDAS Algorithmus für die Netzwerkdaten ein anderes Übergabeformat. Hierbei können alle Daten der jeweiligen Zeitabschnitte in einen CSV-File geschrieben werden. Die CSV-Datei muss dabei folgendermaßen strukturiert sein: Ursprungsknoten, Zielknoten, Zeitintervall. Es ist nicht möglich die Kantengewichtung direkt an den Algorithmus zu übergeben. Um die Kantengewichtung trotzdem übergeben zu können, wird die gleiche Kante mehrmals in Abhängigkeit der Gewichtung an den Algorithmus übergeben. Wie funkt Midas ganz kurz..

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

Abb. 2.2: Datensatz Midas

2.3 MIDAS-R

Der Midas-R Algorithmus speichert weitere Features über die Netzwerke. Dazu gehört die Gesamtzahl an Kanten, welche von einem Knoten ausgehen und die Aktuelle Anzahl an Kanten die von einem Knoten ausgehen. Aus diesem Grund sind die Berechnungen zur Ausreißer-Erkennung deutlich umfangreicher. Das Bewirkt, dass die Laufzeit des Algorithmus mit den Daten aus Kap. 2.2 zu lange ist. Deshalb musste ein Lösung gefunden werden um den Umfang der Daten zu reduzieren, während die Informationen dennoch erhalten bleiben. Dazu wurde eine Hauptkomponenten Zerlegung durchgeführt, welche die Größe der Adjazenzmatrix verringert. Der Nachteil hiervon ist das nicht mehr genau gesagt werden kann, welche Kante genau der Ausreißer ist.

todo: Midas R liefert eigentlich mehrere Ausreißer Scores es wäre vielleicht interessant diese einzeln zu betrachten und nicht zusammenaddiert. todo: Hab hier das mit der Hauptkomponentenzerlegung gemacht. Wenn es Ergebnisse hierfür gibt. Kann ich das hier noch erklären

3 Netsimile

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

3.1 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:

$\overline{d}_i = |N(i)|$
: Die Anzahl der Nachbarn (d.h. Grad) von Knoten i, wobei $N(i)$ die Nachbarn von Knoten i beschreibt.

$\overline{c}_i$
: 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.

$|E^{\circ}_{ego(i)}|$
: 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.

3.2 Anwendung auf Enron und Darpa Datensatz

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

3.3 Erweiterung des Algorithmus

todo: Erklären warum ein neues Feature benötigt wird. Vergleich des Algorithmus mit Feature und ohne neues Feature

3.4 Optimierte Implementierung des Algorithmus

Unter Verwendung der Netsimile Implementierung aus Kap. 3.1, 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 Zeitreihe 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

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

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

$|E^{\circ}_{ego(i)}|$
: Geometrischs Mittel 10 Prozent der höchsten Kantengewichte in $ego(i)$.

$|E^{\circ}_{ego(i)}|$
: 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 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. 3.1 zu sehen.

Anwendung auf 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. 3.1

Tab. 3.1: Parameter Netismile Zeitreihen

Parameter	Beschreibung
Periodizität	Wie in Kap. 2.1 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. 3.4 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.

beschrieben.

Tab. 3.2 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 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, das 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. 3.2 nicht berücksichtigt. Im zweidimensionalen

Tab. 3.2: Netsimile Time Series Perfomance

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	****	***

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. Kap. 5.1.1). 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 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.

4.1 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 Zielpaares 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

4.1.1 Erkennung von Mikrocluster

4.1.2 Experimente

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

Einführung

todo: Stichworte sammeln

4.2 Anwendung auf Zeitreihen

Um den MIDAS Algorithmus auf Zeitreihen anwenden zu können muss die Zeitreihe, wie in Kap. 2.2 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. 4.1). 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.

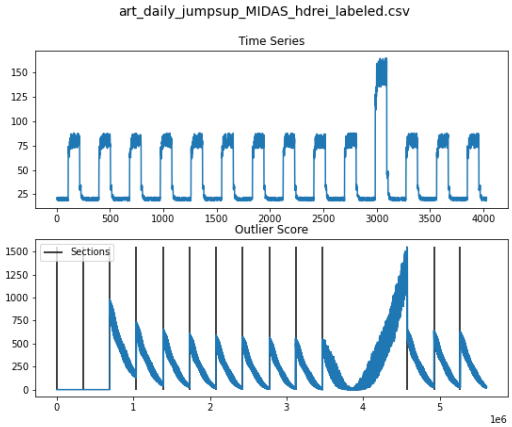
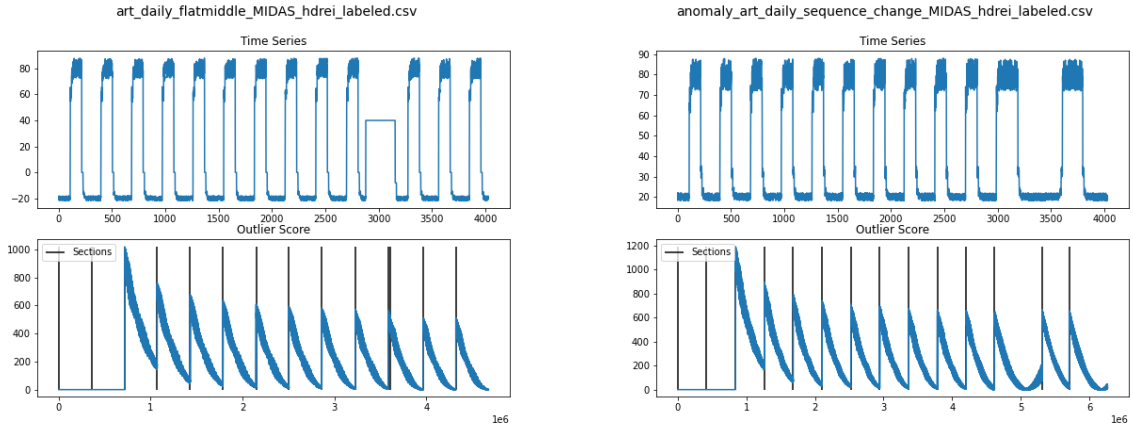


Abb. 4.1: MIDAS Algorithmus angewandt auf Zeitreihe mit einer erhöten Amplitude.

Teilweise führen die Ausreißer auch zu besonders wenigen Kanten (vgl. Abb. 4.2). 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. 4.2). Die Abweichungen sind jedoch so gering, dass es nicht zu einem starken Anstieg des Ausreißer Scores führt.

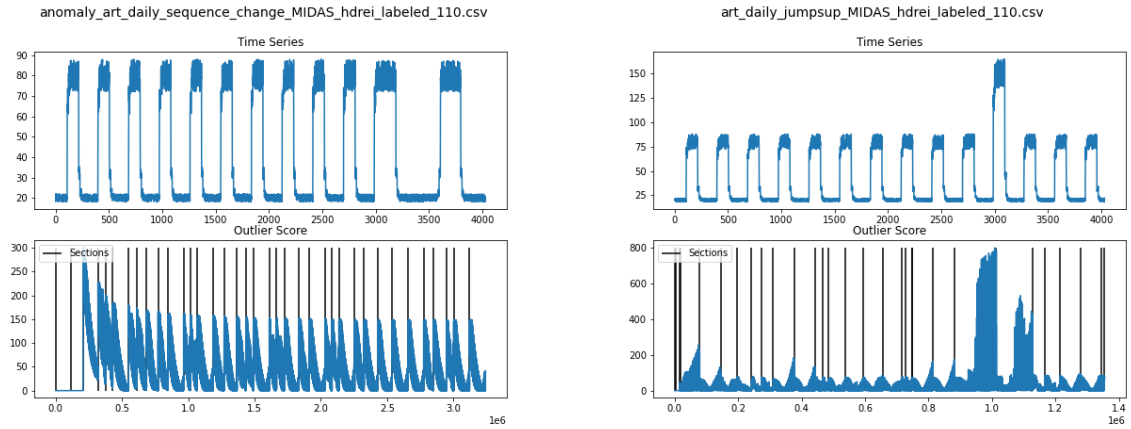


(a) Zeitreihe mit Zyklus Aussetzter **(b)** Zeitreihe mit Frequenzänderung

Abb. 4.2: Ausreißer Erkennung in Zeitreihen MIDAS Algorithmus

Es wurden außerdem Tests durchgeführt um zu Untersuchen, wie sich der Algorithmus bei veränderter Fenstergröße verhält (vgl. Abb. 4.3). Bei den Untersuchungen in Abb. 4.1 und Abb. 4.2 wurde einer Fenstergröße von 288 genutzt, was der Saisonalität der Zeitreihe entspricht. Für dieses Experiment wurde einer Fenstergröße von 110 verwendet. Es konnte festgestellt werden, das 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.

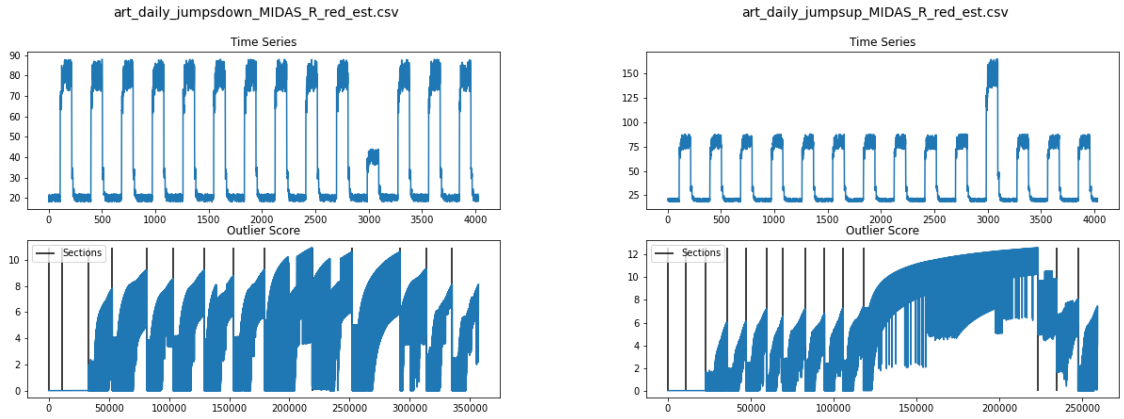
Die anderen Ausreißer Typen werden weiterhin nicht erkannt.



(a) Zeitreihe mit einer Frequenzänderung **(b)** Zeitreihe mit erhöhter Amplitude

Abb. 4.3: 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. 4.4). 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 wir uns durch den Einsatz des MIDAS-R Algorithmus, dass die Ausschläge zu beginn eines jeden Abschnitts aus bleiben, 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. 4.4: Ausreißer Erkennung Zeitreihen MIDAS-R

5 Statische Algorithmen

todo: Labels für die einzelnen Texte umbenennen Es ist mit dieser Art von Algorithmen möglich Ausreißer in einer vollständigen und abgeschlossenen Zeitreihe zu identifizieren. Es werden zwei Algorithmen vorgestellt, ein auf Percolation basierender Algorithmus und ein auf IsoMap basierender Algorithmus. Beide Algorithmen wurden dazu entwickelt Ausreißer in unterschiedlichen Typen von Datensätzen zu erkennen (z.B. Videos, Bilder, Netzwerke). Voraussetzung hierfür ist lediglich, dass eine Distanz zwischen unterschiedlichen Elementen des Datensatzes berechnet werden kann [vgl. 1, S. 2]. Im Folgenden werden die Algorithmen, hinsichtlich ihrer Fähigkeit Ausreißer in Zeitreihen zu identifizieren evaluiert.

Für beide Algorithmen gilt, dass die Zeitreihe zunächst in ein Netzwerk umgewandelt werden muss. Hierzu wird ebenfalls Formel 1 verwendet. 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.

5.1 IsoMap Basierter Algorithmus

Der Grundgedanke hinter diesem Ansatz ist, dass Informationen über Ausreißer bei der Reduzierung der Dimensionalität mit dem IsoMap Algorithmus 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].

5.1.1 IsoMap

Beim IsoMap handelt es sich um einen Algorithmus zur nichtlinearen Dimensionsreduktion. Zunächst werden beim IsoMap die Nachbarn eines jeden Punktes über K-Nearest Neighbor bestimmt. Anschließend wird jeder Punkt mit den gefundenen Nachbarn verknüpft, wodurch ein neuer Körper entsteht. Daraufhin wird eine neue Distanzmatrix auf dem entstandenen Körper berechnet. Diese Matrix kann auch als geodätische Distanzmatrix bezeichnet werden und wird im weiteren Verlauf des Algorithmus benötigt. Der Zweck des Ablaufs ist es das nichtlineare Zusammenhänge in der anschließenden Dimensionsreduktion erhalten bleiben. Die Dimensionsreduktion erfolgt anschließend über Eigenvektor ? [vgl. 9, S. 3]. todo: Noch nach Seite für Quelle suchen

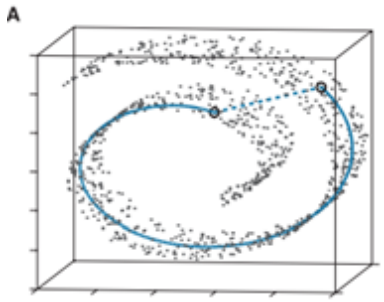


Abb. 5.1: Funktionsweiße IsoMap

5.1.2 IsoMap Basierter Algorithmus

Mithilfe des IsoMap Algorithmus wurden neue Features berechnet. Im nächsten Schritt wird versucht aus diesen Features die ursprüngliche Distanzmatrix zu rekonstruieren. Nun kann die Distanzmatrix aus Kap. 5.1.1 mit dieser Distanzmatrix verglichen werden. Dazu wird die Pearson Korrelation zwischen den jeweiligen Vektoren der Matrizen berechnet. Für Ausreißer wird erwartet, dass die Ähnlichkeit sehr niedrig ist, da die Informationen über sie bei der Reduktion verloren gehen [vgl. 1, S. 3].

5.1.3 Implementierung

Da für die Implementierung des Algorithmus viele Berechnungen mit Matrizen durchgeführt werden müssen, wurde hierzu auf Python/Numpy zurückgegriffen. Für den IsoMap Algorithmus existierte eine sehr gute Implementierung in SciKitLearn, deshalb wurde auf diese zurückgegriffen. An den Algorithmus können verschieden Parameter übergeben werden, es handelt sich hierbei um dieselben Parameter, welche auch an den IsoMap Algorithmus übergeben werden können.

5.1.4 Ergebnisse

Der IsoMap Algorithmus liefert eher schwache Ergebnisse bei der Erkennung von Ausreißern in Zeitreihen. Hauptproblem hierbei ist, dass starke Anstiege in der Zeitreihe zu starken Ausschlägen im Ausreißer Score führt. An den Stellen, an welchen sich tatsächlich Ausreißer befinden, kommt es je nach Ausreißer Typ ebenfalls zu Ausschlägen im Ausreißer Score. Jedoch sind diese Ausschläge etwa so groß wie die der Übergänge. Deshalb ist es nur schwer möglich die Ausreißer zu identifizieren. Das selbe Problem trat auch in [10] bei der Ausreißer Erkennung mit dem Random Walk Algorithmus auf. Um das Problem zu lösen wurde hierbei eine Glättung der Zeitreihe durchgeführt. Dadurch sind die Übergänge zwischen den Abschnitten nicht mehr so plötzlich und werden nicht mehr als Ausreißer markiert [vgl. 10, S. 31,36]. Um Zukünftig bessere Ergebnisse zu erzielen, wäre das ein möglicher Ansatz.
Des Weiteren ist zu erkennen, dass der Algorithmus für einige Ausreißer Typen nicht geeignet ist. Hierzu gehören die Ausreißer Typen, welche sich nicht vom Wertebereich her ändern.

todo: Wie genau bezieht man sich auf seine eigene Ausarbeitung

Tab. 5.1: 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	-

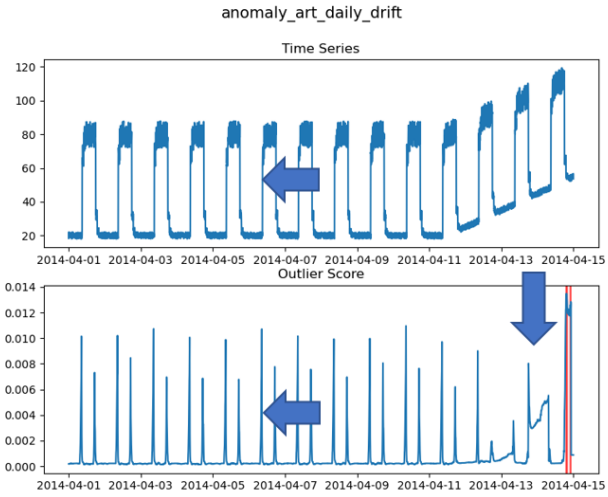


Abb. 5.2: Problem Übergänge

5.2 Perculation

Bei diesem Algorithmus werden schrittweiße die Kanten mit den höchsten Gewichten aus der Distanzmatrix entfernt. Ziel dieses Prozesses ist es Ausreißer vom Hauptcluster 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 [vgl. 1, S. 3].

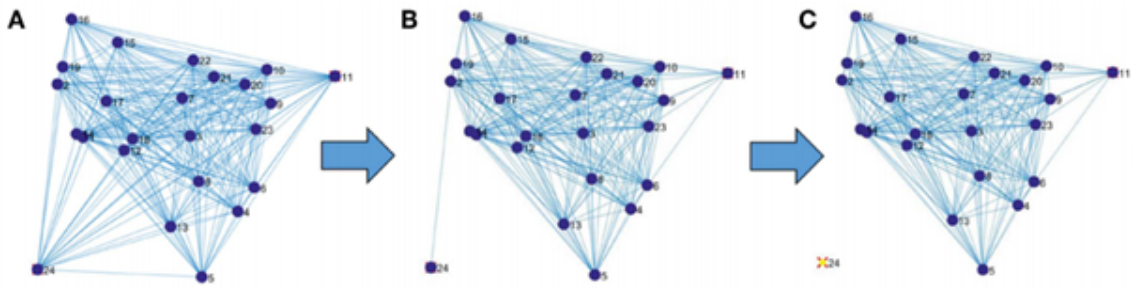


Abb. 5.3: Ablauf Perculation basierter Algorithmus

5.2.1 Implementierung

Aus denselben Gründen wie in Kap. 5.1.3 erläutert, wurde für die Implementierung auf Python/Numpy zurückgegriffen. Da die Distanzmatrix sehr umfangreich werden kann wurden einige Veränderungen an dem Algorithmus vorgenommen, um ihn Performanter zu machen. Eine Modifikation, die vorgenommen wurde, ist das die Kanten nicht einzeln, sondern in Gruppen entfernt werden. Dadurch muss seltener überprüft werden, ob ein Knoten mittlerweile komplett isoliert ist. Außerdem wurde ein Abbruchkriterium implementiert, bei welchem der Algorithmus angehalten wird sobald eine bestimmte Prozentzahl an Kanten entfernt wurde. Dies hat keine Auswirkungen auf die Qualität der Ausreißer Erkennung, da Ausreiser üblicherweise bereits zu beginn des Algorithmus isoliert werden. Der Algorithmus berechnet für jedes Element der Zeitreihe einen Ausreißer Score. Allerdings können die Ausreißer Scores sehr stark schwanken. Deshalb ist es schwierig Ausreißer zu identifizieren, welche sich über mehrere Zeitschritte erstrecken, da kein kontinuierliches Ansteigen des Scores beobachtet werden kann. Eine Möglichkeit, um diese Art der Ausreißer trotzdem zu identifizieren, ist es ein Sliding Window Verfahren einzusetzen. Dabei wird der Ausreißer Score für jedes Element neu berechnet, indem ein Mittelwert über die Zeitpunkte vor einem und nach einem Element gebildet wird. Dadurch werden die Schwankungen im Ausreißer Score abgemildert. Prinzipiell ist der Algorithmus parameterfrei, durch die Veränderungen kann jedoch die Größe des Sliding Window als Parameter übergeben werden.

5.2.2 Ergebnisse

Die Qualität der Ausreißer Erkennung mit dem Perculation Algorithmus kann großenteils als gut bis sehr gut bezeichnet werden. Lediglich die Ausreißer Typen Einzelne Peaks, Signal Aussetzer und Frequenzänderung können vom Algorithmus nicht erkannt werden. Bei den einzelnen Peaks

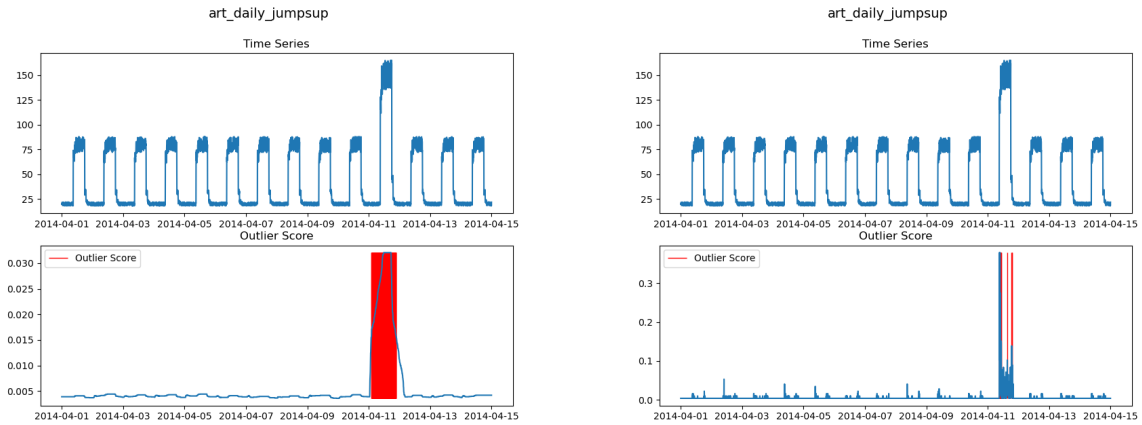


Abb. 5.4: Vergelich Perculation Algorithmus mit Sliding Window Verfahren und ohne Sliding Window Verfahren

Tab. 5.2: 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	-

liegt das an der Verwendung des Sliding Window Verfahren, dadurch werden die Ausschläge im Ausreißer Score weggemittelt und können nur noch sehr schlecht identifiziert werden. Wird jedoch kein Sliding Window Verfahren angewandt können die Ausreißer sehr gut identifiziert werden. Signal Aussetzer und Frequenzänderungen können vom Perculation Algorithmus nicht identifiziert werden, weil die Werte der Zeitreihe hierbei nicht von den Werten der restlichen Zeitreihe abweichen.

todo: Die richtigen Ergebnisse rein machen und bisschen was drüber schreiben

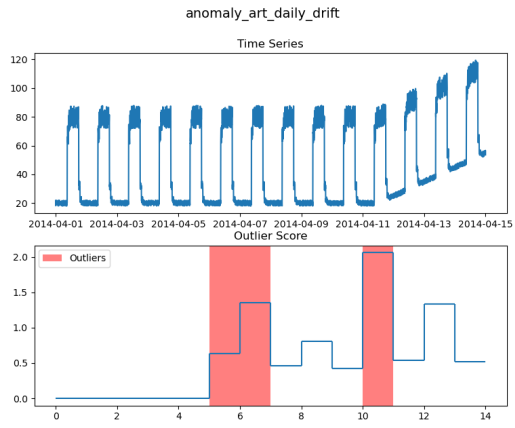
todo: Die Bilder vielleicht noch überarbeiten, sodass sie schöner aussehen. Vielleicht auch noch die Tabelle mit den Sternen rein machen. Vielleicht die beiden Graphiken zu einer Zusammenführen.

todo: Fragestellung: Inwieweit können vielleicht auch andere Datensätze in Graphen umgewandelt werden, sodass z.B. der Netismile darauf angewendet werden kann.

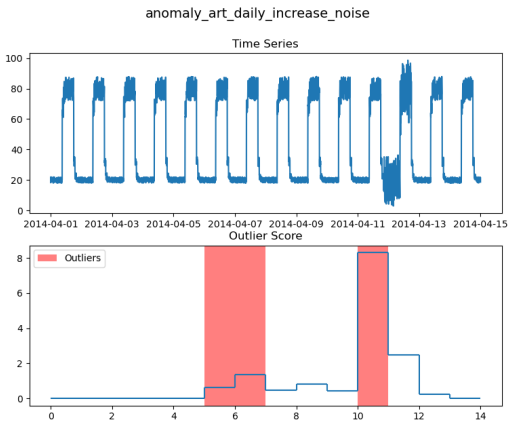
A Netsimile

A.1 Eindimensionales Signal

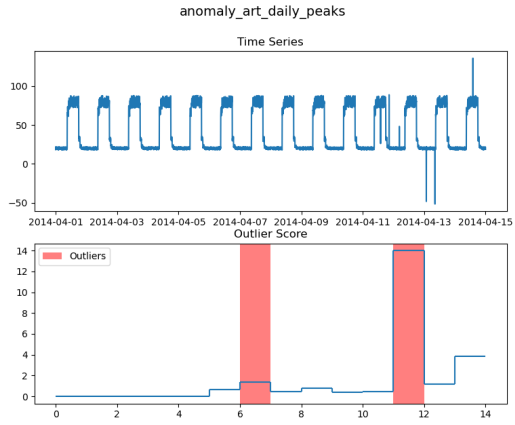
todo: Wrong picture for daily peaks. Change that the sixed element is not always an outlier



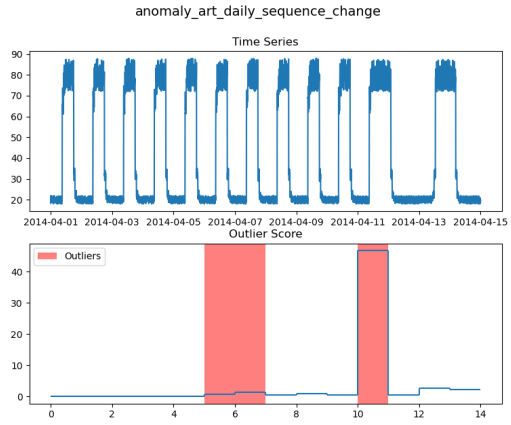
(a) Caption for sub-figure1



(b) Caption for sub-figure1



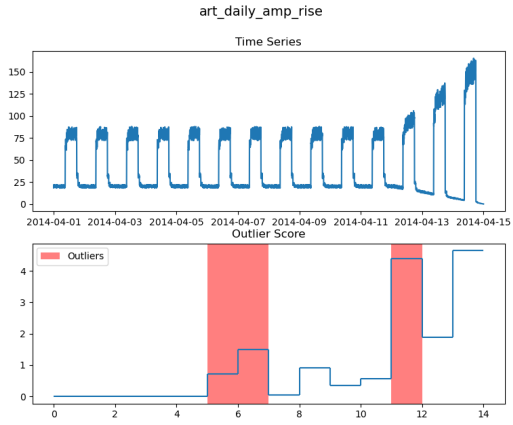
(c) Caption for sub-figure1



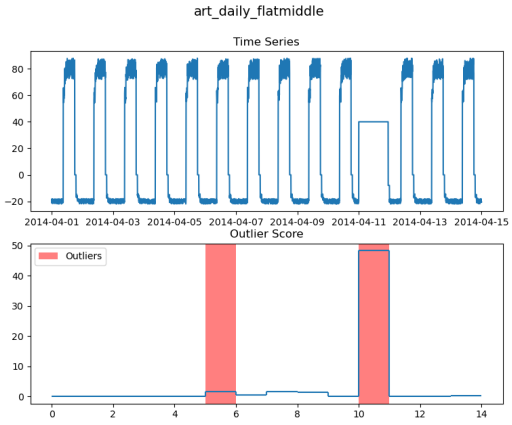
(d) Caption for sub-figure1

A.2 Zweidimensionales Signal

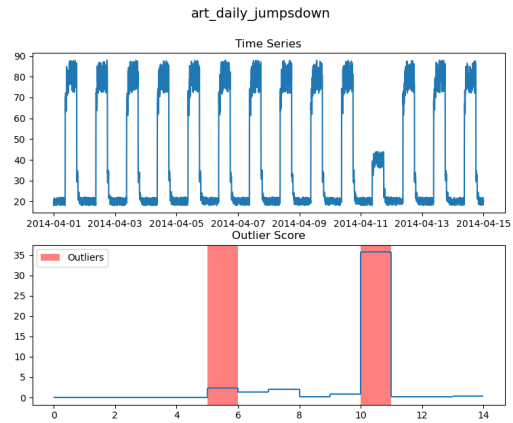
todo: Wrong picture for daily peaks. Change that the sixed element is not always an outlier



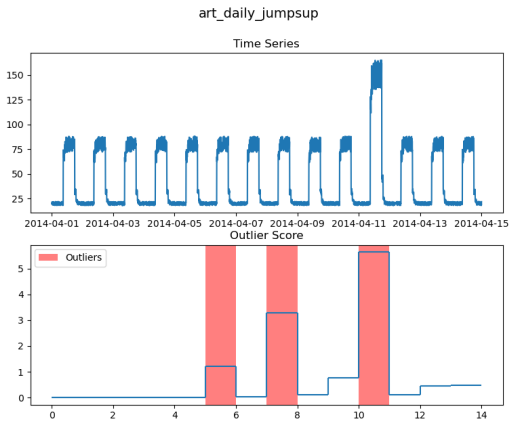
(e) Caption for sub-figure1



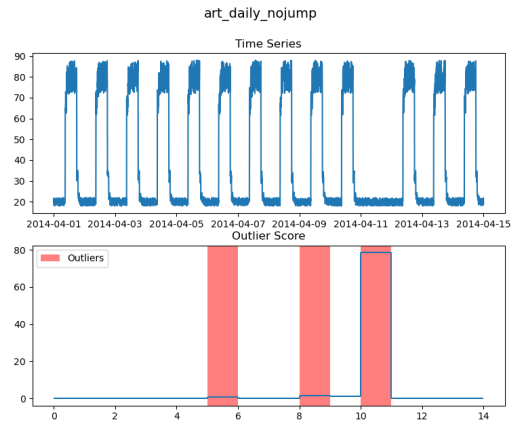
(f) Caption for sub-figure1



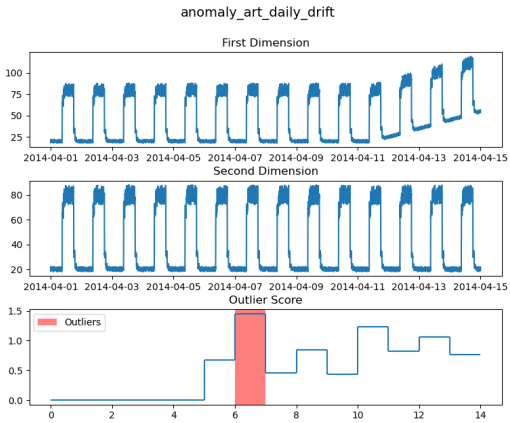
(g) Caption for sub-figure1



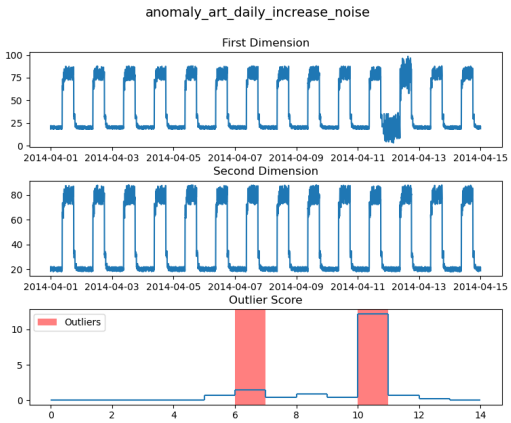
(h) Caption for sub-figure1



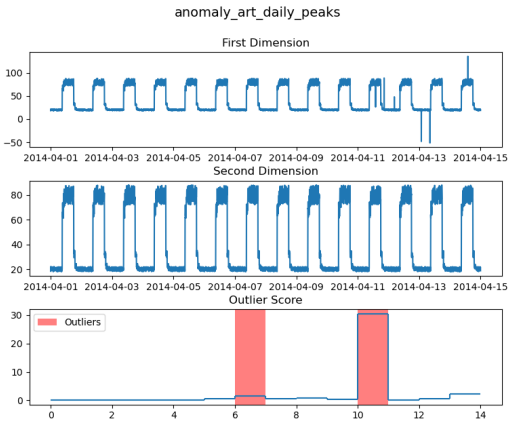
(i) Caption for sub-figure1



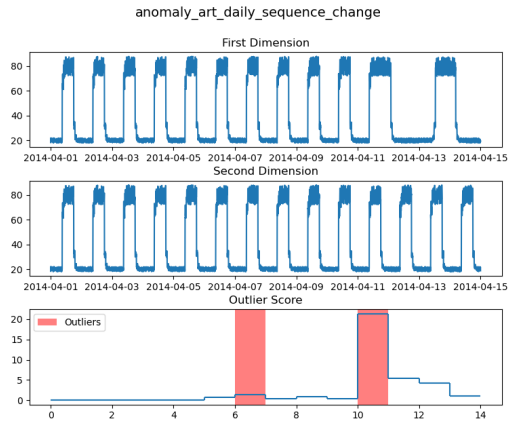
(a) Caption for sub-figure1



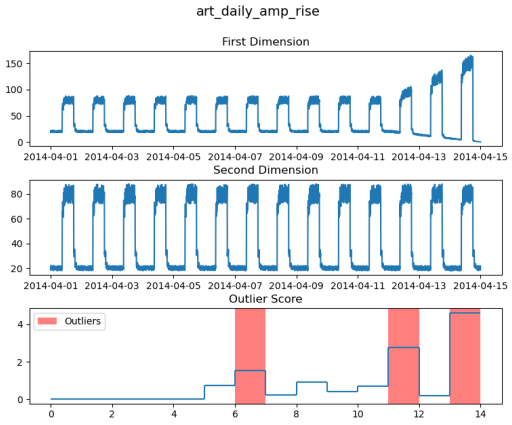
(b) Caption for sub-figure1



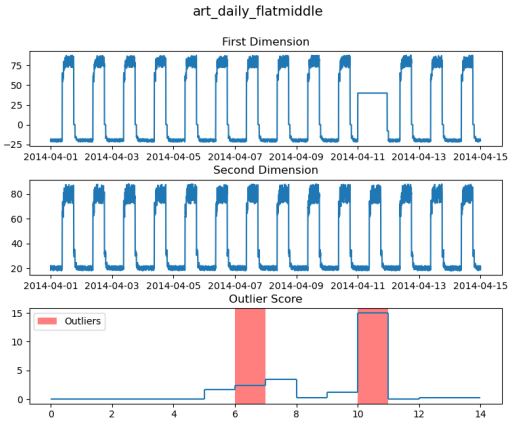
(c) Caption for sub-figure1



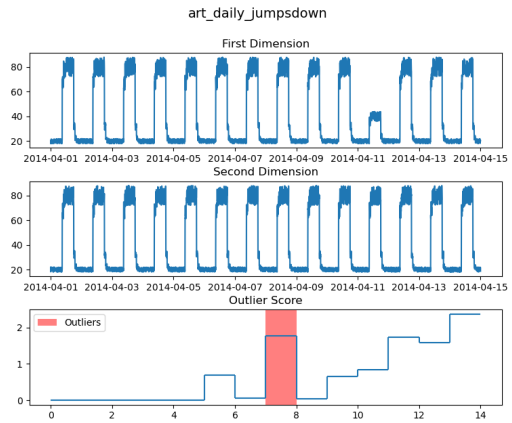
(d) Caption for sub-figure1



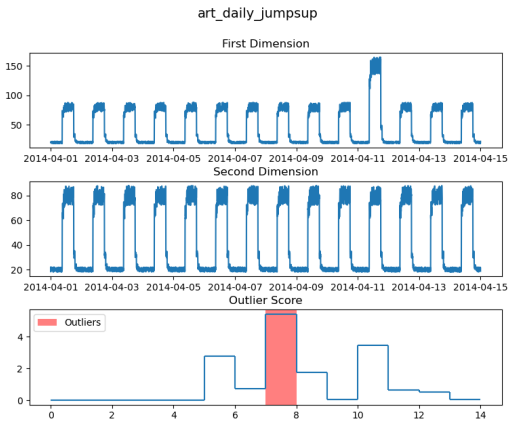
(e) Caption for sub-figure1



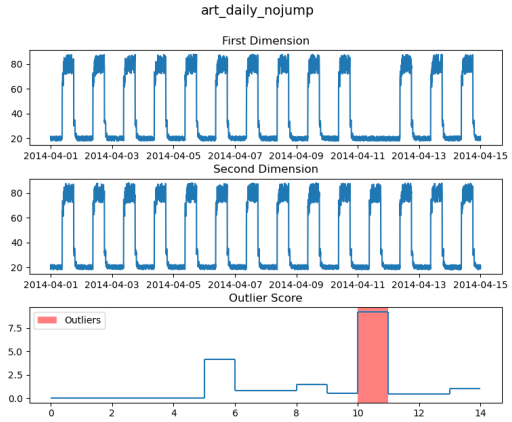
(f) Caption for sub-figure1



(g) Caption for sub-figure1

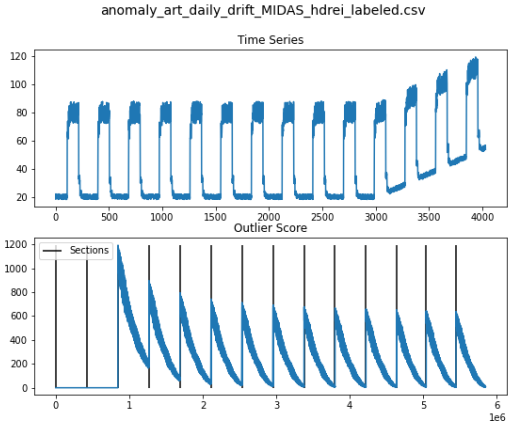


(h) Caption for sub-figure1

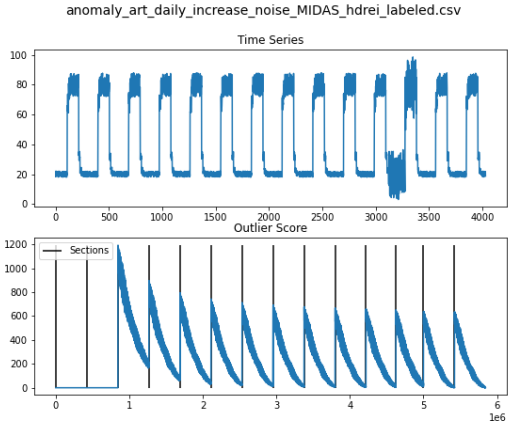


(i) Caption for sub-figure1

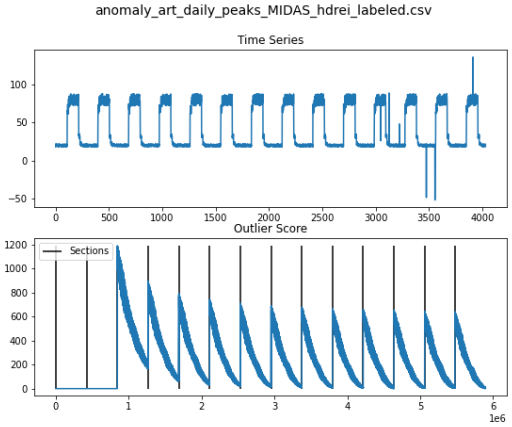
B Midas



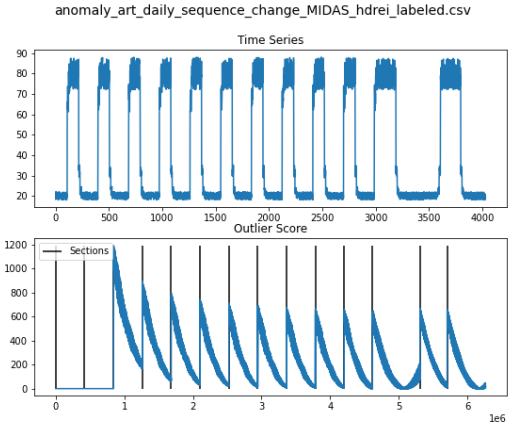
(a) Caption for sub-figure1



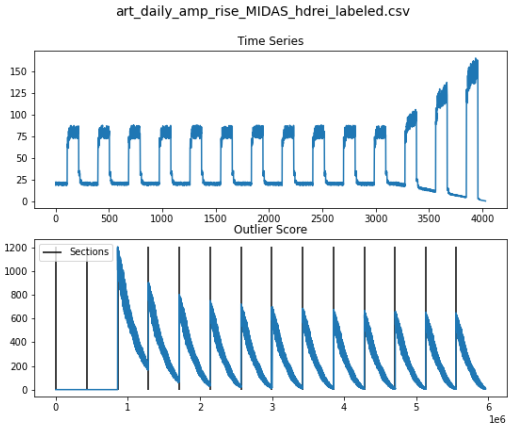
(b) Caption for sub-figure1



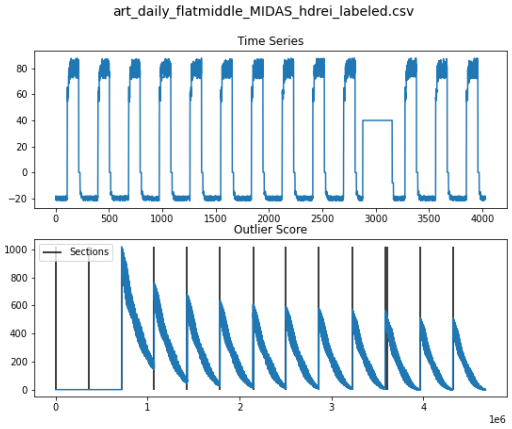
(c) Caption for sub-figure1



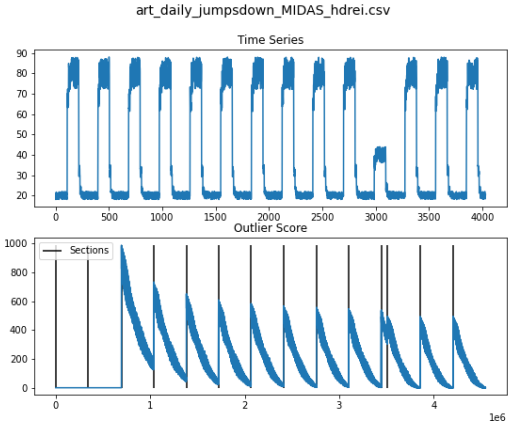
(d) Caption for sub-figure1



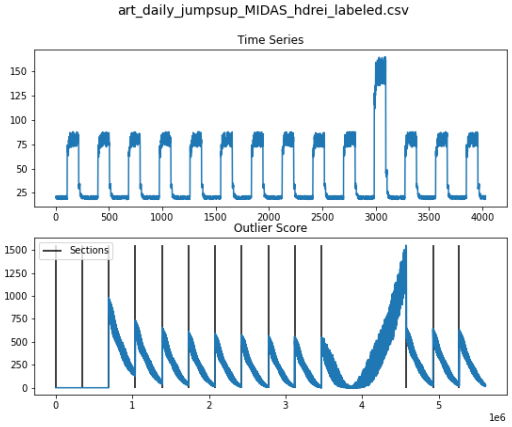
(e) Caption for sub-figure1



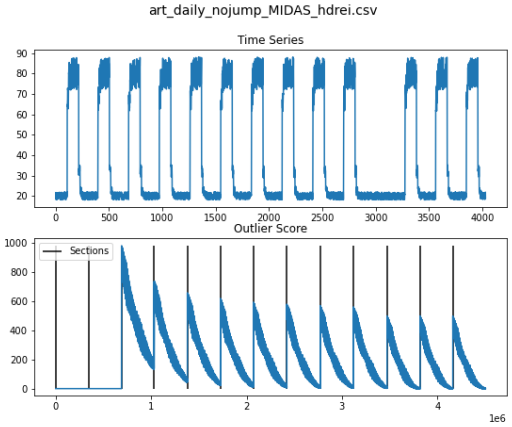
(f) Caption for sub-figure1



(g) Caption for sub-figure1



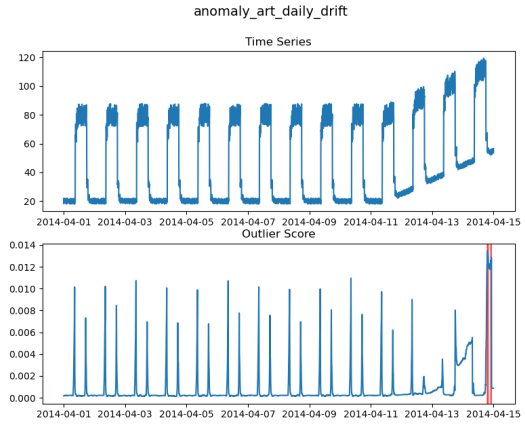
(h) Caption for sub-figure1



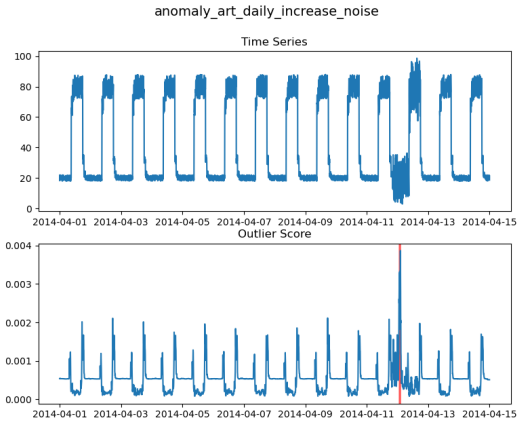
(i) Caption for sub-figure1

C Isomap

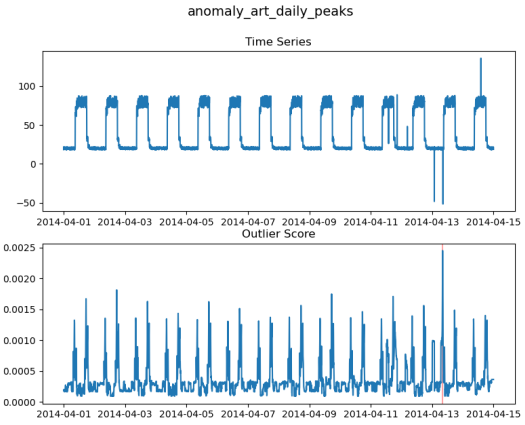
C.1 Eindimensionales Signal



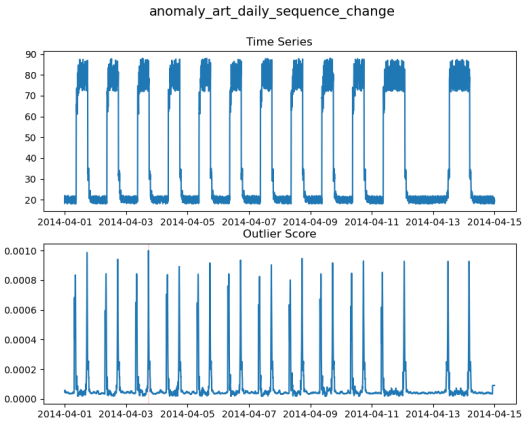
(a) Caption for sub-figure1



(b) Caption for sub-figure1

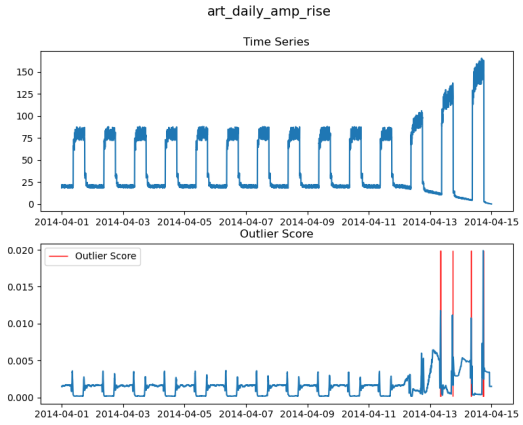


(c) Caption for sub-figure1

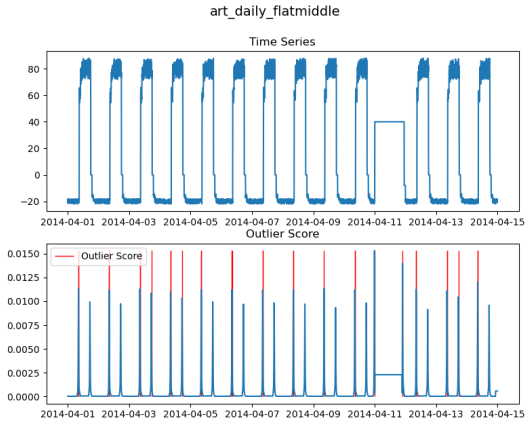


(d) Caption for sub-figure1

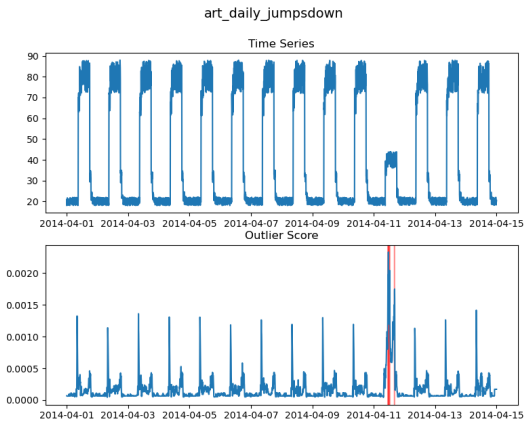
todo: In einigen Bilden fehlt die Legende. Vielleicht noch ein Paar bessere Ergebnisse zu erziehlen



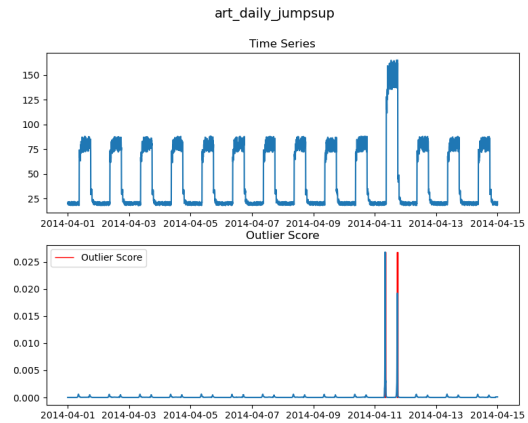
(e) Caption for sub-figure1



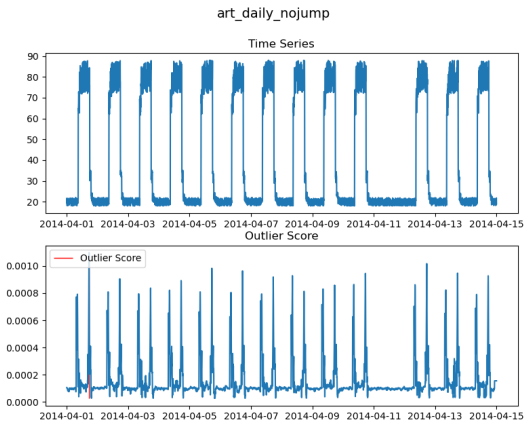
(f) Caption for sub-figure1



(g) Caption for sub-figure1



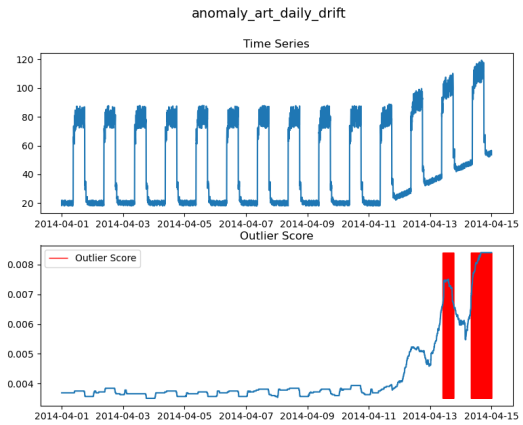
(h) Caption for sub-figure1



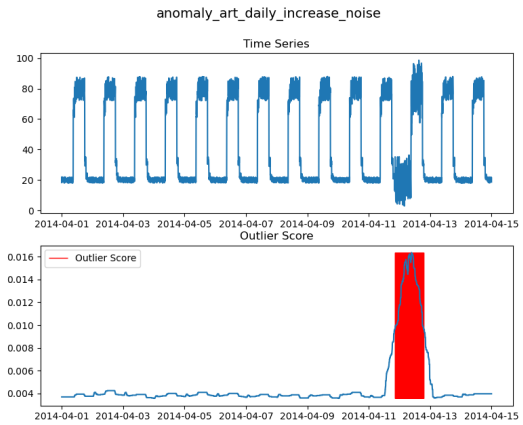
(i) Caption for sub-figure1

D Perculation

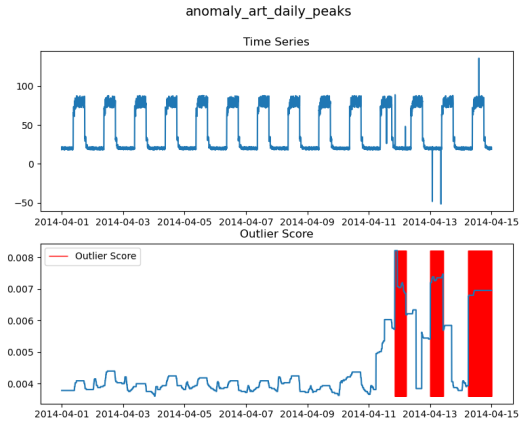
D.1 Sliding Window



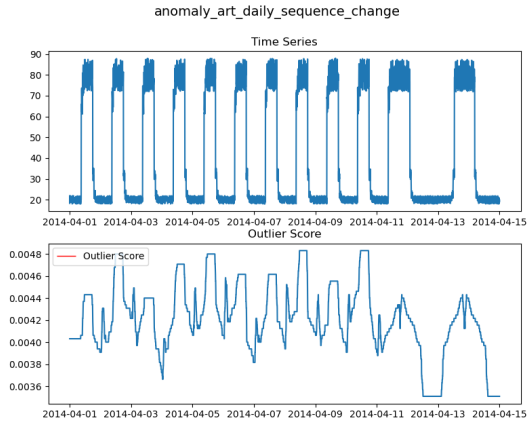
(a) Caption for sub-figure1



(b) Caption for sub-figure1

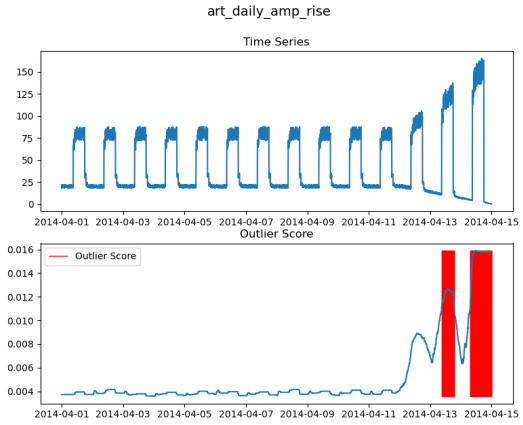


(c) Caption for sub-figure1

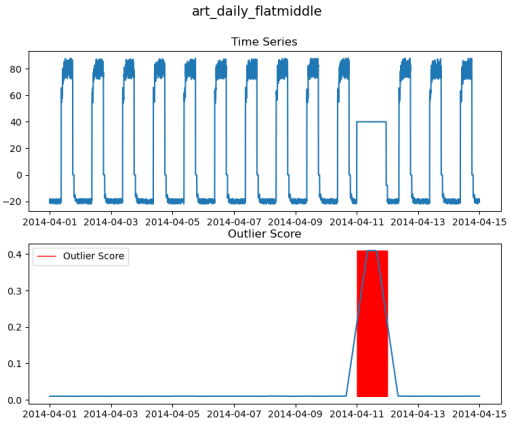


(d) Caption for sub-figure1

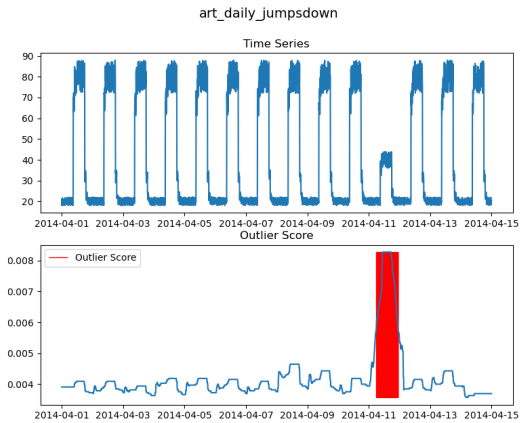
todo: Nim mir nicht sicher ob ich das ohne sliding window auch noch einfÃ¼gen soll. Vielleicht kann ich oben ja einmal einen Vergleich mit sliding window und ohne sliding window rein machen



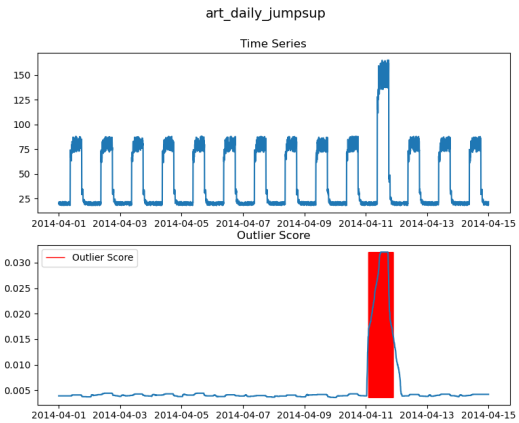
(e) Caption for sub-figure1



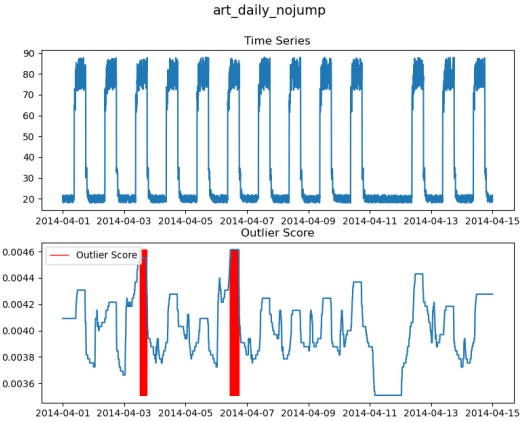
(f) Caption for sub-figure1



(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'.