

Masterarbeit

**Unüberwachte Ausreißerererkennung mit Hilfe
von Submodularen Funktionen**

Philipp-Jan Honysz
Mai 2020

Gutachter:

Prof. Dr. Katharina Morik

Sebastian Buschjäger, M. Sc.

Technische Universität Dortmund

Fakultät für Informatik

Lehrstuhl für Künstliche Intelligenz

<http://www-ai.cs.tu-dortmund.de>

Inhaltsverzeichnis

1	Einleitung	1
1.1	Verwandte Arbeiten	4
1.2	Aufbau der Arbeit	5
2	Submodulare Funktionen	7
2.1	Definition	8
2.2	Optimierung	9
2.2.1	Greedy-Algorithmus	9
2.2.2	Sieve Streaming	11
2.2.3	Weitere Optimierungsverfahren	15
2.3	Beispiele	17
2.3.1	Informative Vector Machine	17
2.3.2	Exemplar-based Clustering	20
3	Ausreißerererkennung	23
3.1	Definition	23
3.2	Lernaufgaben	25
3.3	Strukturierung	27
3.4	Klassische Methoden	29
3.4.1	Isolation Forest	29
3.4.2	Local Outlier Factor	34
3.4.3	Diskussion	37
3.5	Submodulare Methoden	39
3.5.1	Einfache Partitionierung	39
3.5.2	Rekursive Partitionierung	43
3.5.3	Diskussion	49
4	Implementierung	51
4.1	Informative Vector Machine	51
4.1.1	Kernmatrix	52
4.1.2	Determinante	53

4.2	Exemplar-based Clustering	54
4.2.1	General Purpose GPU	57
5	Experimente	69
5.1	Ausreißerererkennung	69
5.1.1	Aufbau	70
5.1.2	Ergebnisse	78
5.1.3	Diskussion	93
5.2	Exemplar-based Clustering auf moderner Hardware	97
5.2.1	Aufbau	97
5.2.2	Ergebnisse	98
5.2.3	Diskussion	103
6	Fazit und Ausblick	107
A	Ausreißerererkennung	111
A.0.1	Maximierung des F1-Scores	111
A.0.2	Maximierung der ROC-AUC	115
	Abbildungsverzeichnis	121
	Algorithmenverzeichnis	123
	Literaturverzeichnis	131
	Erklärung	131

Kapitel 1

Einleitung

Das Erkennen von Ausreißern oder Anomalien in einem statischen Datensatz oder einem Strom von Daten gehört zu den klassischen Disziplinen des maschinellen Lernens [22] und hat eine große praktische Bedeutung. So reicht die Bandbreite möglicher Anwendungen von der Identifikation betrügerischer Kreditkartentransaktionen [3] bis zur Detektion auffälliger Ereignissen in Sensornetzwerken [13] über die Erkennung fehlerhafter Fertigungsprozessen in der Industrie [58]. Diese nutzt beispielsweise detektierte Produktionsanomalien, um die Qualität hergestellter Produkte zu verbessern, den zugrundeliegende Produktionsprozess besser zu verstehen und die Produktion von Ausschussware zu verhindern, um letztlich die Nachhaltigkeit zu verbessern [37, 59].

Ausreißer werden dabei häufig intuitiv über ihren Ausnahmecharakter definiert, das heißt sie treten seltener als „normale“ Beobachtungen auf und weichen hinsichtlich ihrer Merkmalsausprägungen von einer zu definierenden Norm ab [20]. Dies macht es jedoch schwierig, einer Beobachtung die Ausreißereigenschaft datensatz- und anwendungsübergreifend zuzuordnen, da sowohl die *Norm*, die Abweichung davon und die erwartete *Seltenheit* aufwändig für jedes Szenario anzugeben sind. Dieses Problem wird zusätzlich dadurch verstärkt, dass gerade in hochvolumigen Datengenerierungsprozessen oder komplexen Systemen keine markierten Trainingsdaten (engl. *labeled train data*) zu Verfügung stehen, die für individuelle Beobachtungen die Ausreißereigenschaft „markieren“ und die mittels eines sogenannten *überwachten* Lernalgorithmus zur Bestimmung eines Klassifikationsmodells oder der Norm genutzt werden könnten.

Verfahren zur *unüberwachten* Ausreißerererkennung, die auf Grundlage unmarkierter Daten arbeiten, schätzen daher die Norm anhand von bestimmten Annahmen mit dem Ziel eine möglichst zutreffende Prognose zu stellen. Einige der klassischen Methoden versuchen zur Detektion von Ausreißern den Datensatz in disjunkte Mengen zu partitionieren und durch Inspektion ihrer Struktur für individuelle Beobachtungen zu entscheiden, ob diese normal oder anomal sind. Andere Methoden versuchen anhand des Datensatzes ein stochastisches Modell zu inferieren, um für bestimmte Teile des Datenraums zu entscheiden, ob

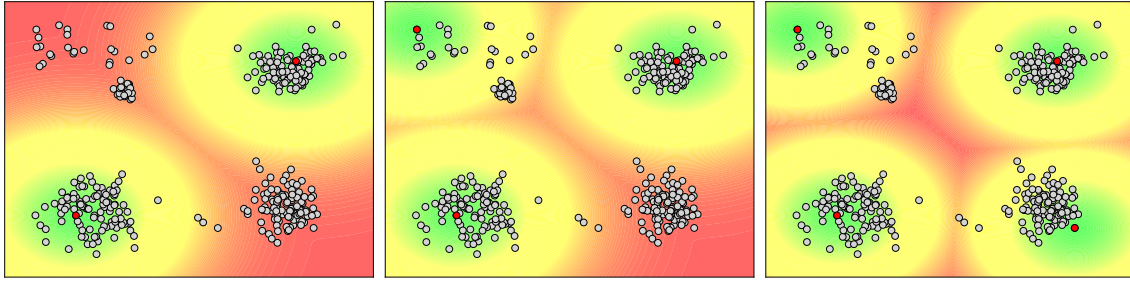


Abbildung 1.1: Ein Beispieldatensatz aus dem mittels der Optimierung einer submodularen Funktion jeweils zwei, drei und vier möglichst diverse Repräsentanten (rot markiert) ausgewählt wurden. Die Flächen zeigen die Ähnlichkeit zu ihrem nächsten Repräsentanten an und stehen hier für ein Maß, das den gemeinsam erklärten Datenraum angibt. Grün bis gelbe Flächen stehen dabei für tendenziell gut erklärte Regionen, während rot-orange Flächen für schlecht erklärte Teile des Datenraums stehen. Es ist erkennbar, dass durch die zusätzliche Auswahl an Repräsentanten die erklärte Fläche zunimmt.

Beobachtungen nach Beurteilung ihrer Auftrittswahrscheinlichkeit eher für eine Ausreißer- oder eine Normalbeobachtung sprechen. Diesen unüberwachten Methoden ist gemein, dass diese sogenannte *intrinsische* Eigenschaften des Datensatzes, wie beispielsweise Distanzen oder Dichteschätzungen, auswerten [20], um die Ausreißerererkennung zu bewerkstelligen.

Eine weitere Möglichkeit den durch den Datensatz aufgespannten Datenraum näher zu untersuchen und im obigen Sinne auch intrinsische Eigenschaften zu entdecken, bieten sogenannte *Datenzusammenfassungen*. Die zugrundeliegende Idee ist hierbei, dass ein möglichst repräsentativer und diverser Ausschnitt des ursprünglichen Datensatzes gefunden wird, sodass möglichst viele interessante Aspekte (und damit auch *normale* als auch *anomale* Ereignisse) des zugrundeliegenden Prozesses in einer kompakten Menge abgebildet werden. Jede Beobachtung dieser Datenzusammenfassung ist dann ein *Repräsentant*, der für eine Region im Datenraum steht und welchem stellvertretend beispielsweise das Attribut einer Ausreißerregion zugeordnet werden kann.

Um solche Zusammenfassungen zu finden, können sogenannte *submodulare Funktionen* betrachtet werden, die ihren mathematisch-theoretischen Ursprung in der sogenannten *Matroiden-* beziehungsweise *Polymatroiden-*Theorie haben [1]: Diese stellen aus formaler Perspektive eine Untergruppe von Mengenfunktion dar, die Mengen auf reelle Werte abbilden. Im Kontext submodularer Funktionen werden diese Werte häufig als *Nutzen* oder quantifizierte *Repräsentativität* hinsichtlich des Datensatzes interpretiert, wobei für diese charakteristisch ist, dass der erzielbare Nutzen mit der Aufnahme weiterer Elemente in die Menge sinkt. Die Idee ist, dass die Aufnahme weiterer Beobachtungen in die Zusammenfassungsmenge den Wert der submodularen Funktion weniger stark steigen lässt. So deckt beispielsweise ein aufgenommener Punkt einen Teil des Datenraums ab, während die Aufnahme eines weiteren Punkts zwar weitere Teile des Raums erklärt aber auch

zu Überlappungen mit anderen Repräsentanten hinsichtlich der Erklärungsfähigkeit führt (vgl. Abbildung 1.1).

Die Frage danach, welche Beobachtungen eines Datensatzes in die Zusammenfassung übernommen werden, führt auf ein NP-hartes Optimierungsproblem [35], welches üblicherweise durch approximative Optimierungsverfahren gelöst wird. Diese weisen trotz Approximation einige komfortable Eigenschaften auf, zu denen Gütegarantien hinsichtlich des erreichbaren Funktionswerts und die Kontrolle budgetrelevanter Parameter gehören. So lässt sich beispielsweise die Speicherkomplexität steuern, was für den Einsatz unter Ressourcenbeschränkung von großer Bedeutung ist. Einige Optimierungsstrategien erlauben außerdem die Identifikation bedeutsamer Zusammenfassungen, selbst wenn die Daten nicht als Stapel sondern als Strom vorliegen. Diese Datenstromszenarien liegen üblicherweise bei sehr hochvolumigen Datengenerierungsprozessen vor und zeichnen sich dadurch aus, dass kein wahlfreier Zugriff möglich ist und jeder Datenpunkt nur einmal beobachtet werden kann. Problematisch ist jedoch häufig, dass auch die approximativen Optimierungsverfahren eine asymptotisch hohe Anzahl an Funktionsauswertungen vornehmen, die eine schnell arbeitende Implementierung der verwendeten submodularen Funktion verlangen, um Datenzusammenfassungen möglichst zügig zu Verfügung zu stellen.

Die Kernfragen sind hierbei, wie individuelle Funktionen die Repräsentativität einer gegebenen Menge modellieren, wie aus einer gefundenen Zusammenfassung eines Datensatzes auf die Ausreißereigenschaft von beliebigen Beobachtungen geschlossen werden kann, wie die Wahl der submodularen Funktion die Qualität der Ausreißerererkennung beeinflusst und damit auch welcher durch die submodulare Funktion modellierte Repräsentativitätsbegriff für die Aufgabe der Detektion von Ausreißern angemessen ist. Aus diesen Fragestellungen sollen folgende Ziele der Arbeit abgeleitet werden:

- Es soll untersucht werden, wie sich submodulare Funktionen zur Ausreißerererkennung verwenden lassen. Für diese Untersuchung ist es essentiell, die formalen Grundlagen dieser Untergruppe von Mengenfunktionen zu etablieren und zu klären, welche approximativen Strategien zur Optimierung existieren. Darüber hinaus soll für einige exemplarisch ausgewählte Funktionen dargelegt werden, wie diese die Repräsentativität einer Menge beurteilen.
- Ferner soll dargelegt werden, wie man sich dem häufig nur intuitiv greifbaren Begriff des *Ausreißers* formal nähern kann, wie Methoden zur Ausreißerererkennung anhand ihrer Lernaufgabe charakterisierbar sind und wie sich diese anhand ihrer Grundannahmen taxonomisch strukturieren lassen. Darüber hinaus soll ein Blick darauf geworfen werden, wie eine durch submodulare Funktionen gefundene Zusammenfassung zum Zwecke der Ausreißerererkennung ausgewertet werden kann und wie in Abgrenzung hierzu klassische Ausreißererkenntungsverfahren arbeiten.

- Weiterhin soll besprochen werden, wie ausgewählte submodulare Funktionen effizient oder unter Ausnutzung eines Grafikprozessors möglichst schnell ausgewertet werden können, um den Laufzeiteinfluss einer unter Umständen hohen Anzahl an Funktionsauswertungen gering zu halten.
- In zwei Experimentierreihen soll empirisch evaluiert werden, ob bestimmte submodulare Funktionen in bestimmten Konfigurationen für den Anwendungsfall der Ausreißerererkennung eine bessere Eignung aufweisen. Weiterhin soll die Effektivität der Auswertung einer speziellen Funktion auf einem Grafikprozessor geprüft werden.

1.1 Verwandte Arbeiten

Submodulare Funktionen werden bereits zur Lösung einer Reihe praktischer Probleme eingesetzt, wozu beispielsweise die effektive Platzierung von Sensoren [6, 31], die Detektion von Ereignissen [49], die empirische Validierung von Software [14] aber auch die Selektion von Merkmalen [60] gehört. Ein weiteres Anwendungsfeld stellt die Anfertigung von Zusammenfassungen datengenerierender Prozesse dar: Dies ist insbesondere im Umfeld hochvoluminöser Daten interessant, in dem die Selektion von repräsentativen Beobachtungen (die manchmal auch *Prototypen* oder *Archetypen* genannt werden) oftmals wünschenswert ist und auch entsprechende Optimierungsverfahren im Datenstrom motiviert [5]. Die durch submodulare Funktionen zusammenfassbaren Daten sind dabei mannigfaltiger Natur und umfassen beispielsweise Videodaten [24, 42], Bilddaten [61] oder Textkorpora [38, 39, 56].

Das Themenfeld der Anomalie- oder Ausreißerererkennung ist hingegen, wie Chandola *et al.* darlegen, eine Thematik, die mindestens seit dem 19. Jahrhundert erforscht wird [11] und seitdem eine Reihe von Erkennungsverfahren aus dem Bereich der Statistik und dem maschinellen Lernen hervorgebracht hat. Dazu gehören beispielsweise Klassifikationsbasierte Ansätze, die unter anderem künstliche neuronale Netze, Bayes-Netze oder Stützvektormethoden einsetzen, Nächste-Nachbarn- und Clustering-basierte Ansätze oder auch statistische Methoden [11]. Es würde den Rahmen dieser Arbeit sprengen sämtliche Anwendungsszenarien und Detektionsmethoden hier zu skizzieren, weswegen an dieser Stelle auf die zusammenfassende Arbeit von Chandola *et al.* [11] verwiesen werden soll, die diese Thematik im Detail beleuchtet.

Die Erkennung von Ausreißern mit submodularen Funktionen ist hingegen weitestgehend unerforscht. Nach Sichtung entsprechender Literatur ließen sich lediglich einige Arbeiten auffinden, die beispielsweise submodulare Funktionen verwenden, um Kernel in kernbasierten Methoden zu bestimmen, die wiederum Ausreißer in Zeitreihen erkennen [66] oder um eingeschleuste Komponenten in integrierten Schaltkreisen zu identifizieren [34]. Einige Ansätze setzen submodulare Funktionen im Sinne eines Nachverarbeitungsschritts ein, um Ausreißer, die mit einem etablierten Verfahren gefunden wurden, aufzubereiten

und dem Benutzer näher zu erklären [23, 57]. Soweit ebenfalls bekannt ist, existieren keine bereits veröffentlichten Arbeiten, die eine Ausreißerererkennung auf Grundlage einer durch submodulare Funktionen identifizierten Zusammenfassung vornehmen: Ein neuartiger Ansatz stammt von Buschjäger *et al.*, der eine durch eine Zusammenfassung induzierte Partitionierung des Raums betrachtet und anhand eines statistischen Tests entscheidet, ob eine bestimmte Partition oder eine bestimmte *Region* ausreißt und somit auffällig ist [9].

1.2 Aufbau der Arbeit

Das Kapitel 2 soll in das Themenfeld der submodularen Funktionen einführen und diese hierfür in Abschnitt 2.1 formal etabliert werden. Im Abschnitt 2.2 sollen dann die Optimierungsprobleme und die Strategien zur Optimierung vorgestellt werden, um anschließend in Abschnitt 2.3 zwei Beispiele für submodulare Funktionen vorzustellen. Das Kapitel 3 beschäftigt sich dann mit der Thematik der Ausreißer und der Ausreißerererkennung. Um dies zu bewerkstelligen sollen in Abschnitt 3.1 Definitionen des Begriffs *Ausreißer* vorgestellt und Mechanismen zu ihrer Entstehung diskutiert werden. In den Abschnitten 3.2 und 3.3 sollen dann die Lernaufgaben und die Strukturierung der Methoden besprochen werden, die in den Abschnitten 3.4 und 3.5 nochmals aufgegriffen werden, um die dort vorgestellten klassischen und submodularen Methoden der Ausreißerererkennung einzuordnen. Im Kapitel 4 wird dann besprochen, wie sich die vorgestellten submodularen Funktionen effizient oder unter Unterstützung moderner Hardware implementieren lassen, um die hohe Anzahl an Funktionsauswertungen nicht zu einem gewichtigen Laufzeitfaktor werden zu lassen. Das Kapitel 5 soll dann die Motivation und die Durchführung einiger Versuchsreihen dokumentieren, die die Effektivität verschiedener Ausreißerererkennungsverfahren anhand einer Menge von Datensätzen und gewählter Metriken messen. Außerdem soll in Abschnitt 5.2 die Effektivität der GPU-Beschleunigung gegenüber der CPU gemessen und dokumentiert werden. Im abschließenden Kapitel 6 soll die Arbeit inhaltlich zusammengefasst, ein Fazit gezogen und ein Ausblick darauf gegeben werden, wie die entwickelten Konzepte weiterentwickelt werden können.

Kapitel 2

Submodulare Funktionen

Submodulare Funktionen stellen eine Klasse von Mengenfunktionen dar, die einer Menge von Beobachtungen einen reellen Wert (den *Nutzen*, engl. *utility*) zuordnen, wobei die Hinzunahme weiterer Beobachtungen in diese Menge den Funktionswert stets weniger stark ansteigen lässt (engl. *diminishing returns property*) und somit einen Sättigungseffekt bewirkt. Ein praktisches Beispiel für den Einsatz dieser Eigenschaft liefern *Krause* und *Golovin* [35]: In einem Netzwerk aus Wasserrohren sollen Sensoren platziert werden, die Kontaminationen des geführten Wassers erkennen. Jeder dieser Sensoren ist dabei in der Lage einen bestimmten Teil des Rohrnetzes abzudecken und damit einen Nutzen zu liefern. Die Hinzunahme weiterer Sensoren erhöht zwar den Nutzen, führt aber auch zu Überlappungen mit bereits überwachten Bereichen, woraus ein geringerer *Nutzengewinn* resultiert. Dies führt auf ein Optimierungsproblem, welches zumeist kardinalitätsbeschränkt ist: So sollen nur eine bestimmte Zahl an Sensoren platziert, aber der Nutzen maximiert werden. Auf ähnliche Weise lässt sich ein Problem motivieren, welches auf der Identifikation von *Datenzusammenfassungen* beruht: Aus einem Datensatz sollen repräsentative Beobachtungen ausgewählt werden, die jeweils einen Teil der vollständigen Daten erklären und eine Zusammenfassung bilden. Die Hinzunahme weiterer Beobachtungen in diese Zusammenfassung erklärt den ursprünglichen Datensatz zwar immer besser, führt allerdings dazu, dass der zusätzlich erklärte Anteil des Datenraums kleiner wird und somit der *Erklärungsgewinn* sinkt. Das Ziel bei der Betrachtung dieses Problems ist die Identifizierung möglichst kompakter und aussagekräftiger Zusammenfassungen. Hierbei stellt sich insbesondere die Frage, wie sich die *Repräsentativität* einer individuellen Menge feststellen lässt.

Zur Klärung dieser Fragen sollen in Abschnitt 2.1 zunächst die formalen Grundlagen der submodularen Funktionen etabliert werden. In Abschnitt 2.2 soll dann das Optimierungsproblem und die dazugehörigen Optimierungsmethoden näher beleuchtet werden, um dann in Abschnitt 2.3 die submodularen Funktionen der INFORMATIVE VECTOR MACHINE (IVM) und des EXEMPLAR-BASED CLUSTERINGS exemplarisch vorzustellen. Diese sollen hinsichtlich des Problems der Identifikation von Datenzusammenfassungen motiviert wer-

den, wobei insbesondere erklärt wird, wie diese technisch die Messung der Repräsentativität einer gegebenen Menge umsetzen.

2.1 Definition

Submodulare Funktionen stellen Mengenfunktionen dar, die sich durch die Eigenschaft verschwindener Nutzengewinne auszeichnen. Um diese Eigenschaft auszudrücken wird der Begriff der *diskreten Ableitung* wie folgt etabliert:

2.1.1 Definition (Diskrete Ableitung [35]). Sei V eine endliche Menge, $f : \mathcal{P}(V) \rightarrow \mathbb{R}$, $S \subseteq V$ und $e \in V$. Dann heißt $\Delta_f(e | S) = f(S \cup \{e\}) - f(S)$ die *diskrete Ableitung* von f mit der Menge S hinsichtlich der Beobachtung e .

Die diskrete Ableitung ist also gerade der *Nutzengewinn*, der durch die Hinzunahme einer bestimmten Beobachtung erzielt wird. Die Menge V wird dabei häufig als *Grundmenge* bezeichnet, da sich alle anderen Mengen, die als Argument für die submodulare Funktion verwendet werden, von der Grundmenge anhand der Teilmengenrelation ableiten.

Mittels der diskreten Ableitung kann der Begriff der submodularen Funktion dann wie folgt definiert werden:

2.1.2 Definition (Submodulare Funktion [35]). Eine Funktion $f : \mathcal{P}(V) \rightarrow \mathbb{R}$ heißt *submodular*, wenn für alle $A \subseteq B \subseteq V$ und $e \in V \setminus B$ folgendes gilt:

$$\Delta_f(e | A) \geq \Delta_f(e | B). \quad (2.1)$$

Ferner sei für den Fall einer leeren Argumentmenge $f(\emptyset) = 0$ definiert.

Diese Definition erlaubt es allerdings, dass die Hinzunahme weiterer Beobachtungen in eine Menge S den Funktionswert sinken lässt. Submodulare Funktionen, die diese Eigenschaft nicht aufweisen, heißen *monoton* und werden wie folgt definiert:

2.1.3 Definition (Monotone submodulare Funktion [35]). Eine submodulare Funktion $f : \mathcal{P}(V) \rightarrow \mathbb{R}$ heißt *monoton*, wenn für alle $A \subseteq B \subseteq V$ gilt, dass $f(A) \leq f(B)$.

Die monotonen submodularen Funktionen stellen eine wichtige Untergruppe der submodularen Funktionen dar. So arbeiten die im nächsten Abschnitt vorgestellten Optimierungsverfahren ausschließlich unter der Annahme, dass die zu optimierende Funktion tatsächlich *monoton* ist, weswegen im Folgenden ausschließlich auf monotone submodulare Funktionen Bezug genommen wird.

2.2 Optimierung

Mit dem in der Einleitung formulierten Verständnis der Bewertung einer Menge durch Zuordnung eines spezifischen *Nutzens* geht der Wunsch der Maximierung dieser Funktion einher. Informell wird also jene Menge gesucht, die die submodulare Funktion hinsichtlich definierter Nebenbedingungen maximiert. Die Nebenbedingungen sind dabei ein essentieller Teil der Problemformulierung, da diese unter anderem die Komplexität der gefundenen Lösung steuern. Betrachtet man beispielsweise das eingangs formulierte Beispiel der Abdeckung eines Rohrnetzwerkes, so ist es unwirtschaftlich mehr Sensoren zu platzieren als für die Entdeckung von Kontaminationen notwendig sind. Genauso möchte man für die Identifikation von Datenzusammenfassungen möglichst wenig Beobachtungen auswählen, um sowohl Kompaktheit als auch Repräsentativität zu garantieren.

Formell lässt sich das Optimierungsproblem, gegeben einer Grundmenge $V = \{\vec{v}_1, \dots, \vec{v}_n\}$ und einer submodularen Funktion $f : \mathcal{P}(V) \rightarrow \mathbb{R}$, wie folgt aufstellen [35]:

$$\max_{S \subseteq V} f(S) \text{ unter Nebenbedingungen auf } S. \quad (2.2)$$

Mögliche Nebenbedingungen sind neben Knapsack- und Matroid-Beschränkungen vor allem Kardinalitätsbeschränkungen [35], auf welchen der Fokus in dieser Arbeit liegt. Dabei wird gefordert, dass für beliebiges $k \in \mathbb{N}$ die Bedingung $|S| \leq k$ erfüllt ist:

$$S_k^* = \max_{S \subseteq V, |S| \leq k} f(S) \quad (2.3)$$

Wie *Krause* und *Golovin* notieren, ist dieses Optimierungsproblem für viele submodulare Funktionen bereits ein NP-hartes Problem [35]. Gegeben einer Grundmenge V könnte es zur Identifikation der global-optimalen Lösung beispielsweise notwendig sein, die Potenzmenge $\mathcal{P}(V)$ zu prüfen, was zu $\mathcal{O}(2^{|V|})$ Evaluationen der submodularen Funktion führt und damit nicht als effizient erachtet werden kann.

Approximative Optimierungsverfahren behelfen sich daher mit der Bereitstellung einer Näherung an die optimale Lösung S_k^* beziehungsweise den dazugehörigen Funktionswert $f(S_k^*)$. In diesem Abschnitt sollen zwei Optimierungsverfahren für submodulare Funktionen vorgestellt werden, die das Problem 2.3 approximativ lösen, nämlich der GREEDY-Algorithmus und das SIEVE STREAMING. Neben dem Funktionsprinzip des jeweiligen Verfahrens sollen auch die gebotenen (theoretischen) Gütegarantien und Laufzeit- und Speichernutzungseigenschaften beleuchtet werden, die für einen Einsatz unter Ressourcenbeschränkung von Bedeutung sind.

2.2.1 Greedy-Algorithmus

Für das Problem der Maximierung submodularer Funktionen lässt sich ein gieriger Algorithmus formulieren, der in jedem Schritt jene Beobachtung auswählt, welche den größten

Nutzengewinn verspricht. Die Hoffnung ist hierbei, dass die so getroffenen *lokalen* Entscheidungen eine *globale* Optimallösung produzieren [25].

Algorithmus 2.1 GREEDY-Algorithmus [35].

Eingabe: Grundmenge V , Kardinalitätsbeschränkung $k \in \mathbb{N}$, Submodulare Funktion f

```

1:  $S_0 \leftarrow \emptyset$ 
2: for  $i \in [1, k]$  do
3:    $S_i = S_{i-1} \cup \left\{ \arg \max_{e \in V \setminus S_{i-1}} \Delta_f(e \mid S_{i-1}) \right\}$ 
4: end for
5: return  $S_k$ 

```

Algorithmus Der GREEDY-Algorithmus entscheidet in jedem Schritt i , welche noch nicht ausgewählte Beobachtung aus der Grundmenge V den größten Nutzengewinn verspricht und fügt diese der Menge hinzu, sodass die Menge S_i resultiert. Bieten zwei oder mehr Beobachtungen den gleichen Nutzengewinn, wird eine dieser Beobachtungen willkürlich ausgewählt. Der GREEDY-Algorithmus bricht ab, wenn die zuvor aufgestellten Kardinalitätsbeschränkungen erfüllt sind.

Garantien Wie Nemhauser *et al.* festgestellt haben, lässt sich für den GREEDY-Algorithmus eine theoretische Gütegarantie angeben, die im Folgenden Theorem dargelegt wird:

2.2.1 Theorem (Approximationsgüte (GREEDY) [45]). Sei $f : \mathcal{P}(V) \rightarrow \mathbb{R}$ die zu optimierende submodulare Funktion, $k \in \mathbb{N}$ die Kardinalitätsbeschränkung nach Problem 2.3 sowie $\{S_i\}_{i \geq 0}$ die durch den GREEDY-Algorithmus produzierten Zwischenlösungen und $l \in \mathbb{N}$. Ferner sei $k = l$ und e die eulersche Zahl. Dann erfüllt GREEDY die folgenden Eigenschaften:

- Die Güte der gefundenen Lösung ist nach unten durch folgenden Ausdruck beschränkt

$$f(S_l) \geq \left(1 - e^{-l/k}\right) f(S_k^*) \quad (2.4)$$

- Nach Gleichung 2.4 beträgt die Approximationsgüte bei $k = l$ folglich mindestens $(1 - e^{-1}) \approx 63,21\%$.

Krause und *Golovin* notieren dabei eine Relaxierung der oben genannten Bedingungen, sodass l und k beliebige positive Ganzzahlen darstellen können. Die Autoren nutzen diese Relaxierung, um die Verbesserung der Approximation dann zu quantifizieren, wenn der GREEDY-Algorithmus *mehr* Beobachtungen auswählen darf, um das Optimalergebnis des ursprünglich aufgestellten Optimierungsproblems 2.3 zu approximieren, also $l > k$.

Wie *Nemhauser* und *Wolsey* zeigen, stellt der GREEDY-Algorithmus für submodulare Funktionen sogar das beste, noch erreichbare Ergebnis bereit: Unter der Voraussetzung, dass nur eine polynomielle Größe von Funktionsauswertungen vorgenommen werden darf, existiert kein Algorithmus, der eine bessere Approximationsgüte als $(1 - e^{-1})$ liefert [44]. Dies ist genau die GREEDY-Approximationsgüte gegeben dem Fall, dass $l = k$ gilt (vgl. Gleichung 2.4).

Laufzeiteigenschaften Der GREEDY-Algorithmus nimmt $\mathcal{O}(|V| \cdot k)$ Funktionsauswertungen vor. Die Anzahl der Iterationen über die gesamte Grundmenge V ist beschränkt durch $\mathcal{O}(k)$, wobei der Speicherverbrauch ebenfalls durch $\mathcal{O}(k)$ beschränkt ist [5].

2.2.2 Sieve Streaming

Wie in Abschnitt 2.2.1 beschrieben liefert der GREEDY-Algorithmus für die Gruppe effizienter, also in polynomieller Zeit operierender Algorithmen, für das Problem 2.3 nachweislich eine Lösung mit bester Approximationsgüte. Trotzdem existieren Szenarien in denen der GREEDY-Algorithmus abzulehnen ist. Ein solches Szenario ist beispielsweise dann gegeben, wenn die Grundmenge V nicht als Stapel von Beobachtungen präsentiert wird, sondern vielmehr als Datenstrom vorliegt. Diese Datenströme sind dadurch charakterisiert, dass jedes Element genau einmal betrachtet wird, wodurch sich nicht die nächste nutzenmaximierende Beobachtung auswählen lässt, da dies den wahlfreien Zugriff auf die Grundmenge verlangen würde. Ein weiterer problematischer Aspekt ist die hohe Anzahl an benötigten Funktionsauswertungen, die der GREEDY-Algorithmus benötigt und die von der Größe der Grundmenge abhängt.

Es wurden daher Algorithmen entwickelt, die die Optimierung submodularer Funktionen beziehungsweise die Lösung des Problems 2.3 auf Grundlage von Datenströmen vornehmen. Zu diesen gehört unter anderem das SIEVE STREAMING von *Badanidiyuru et al.* [5].

Algorithmus Das SIEVE STREAMING-Verfahren lässt sich durch die Idee charakterisieren, den GREEDY-Algorithmus zu „simulieren“. Diese Simulation beruht auf der Erkenntnis, dass im Schritt i des GREEDY-Verfahrens, in dem bereits die Menge S_i ausgewählt wurde, der Grenznutzen des nächsten ausgewählten Elements e_{i+1} durch folgende untere Schranke beschränkt ist [5]:

$$\Delta_f(e_{i+1} | S_i) \geq \frac{f(S_k^*) - f(S_i)}{k} \quad (2.5)$$

Der Kernmechanismus von SIEVE STREAMING ist nun, nur diejenigen Elemente aufzunehmen, die einen ähnlich hohen Nutzengewinn aufweisen, wie GREEDY diesen hervorbringen würde. Aus technischer Perspektive wird dabei jedes im Strom ankommende Element auf

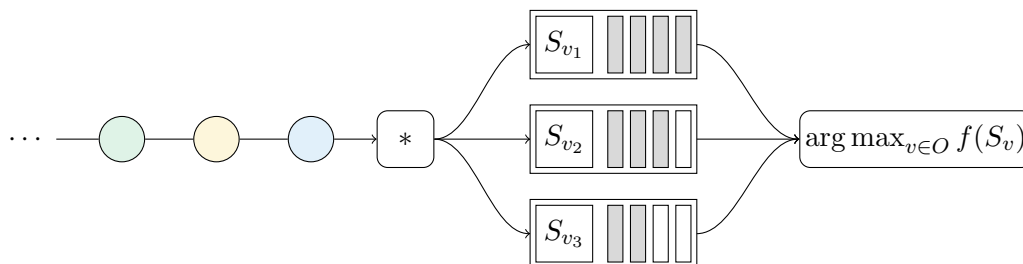


Abbildung 2.1: Kernmechanismus des SIEVE STREAMINGS für die Optimierung von submodularen Funktionen unter Kardinalitätsbeschränkung mit $k = 4$: Ein Strom von Beobachtungen wird mittels eines Multiplikatorelements („*“) an drei verschiedene *Siebe* mit den individuellen Grenzwerten v_1 , v_2 und v_3 verteilt. Diese nehmen eine spezifische Beobachtung auf, sofern der Grenznutzen einen bestimmten Wert überschreitet und das Sieb noch nicht voll ist. Wenn alle Siebe gefüllt oder der Beobachtungsstrom terminiert, wird mittels einer „arg max“-Operation ermittelt, welches Sieb den Funktionswert maximiert. Die Details sind in Algorithmus 2.3 und 2.4 dargelegt.

die Überschreitung eines spezifischen Grenzwertes überprüft (engl. *thresholding*), wobei dieser Grenzwert festzulegen ist.

Ausgehend von Formel 2.5 lässt sich bei Auswahl des *ersten* Elements daher die Überlegung aufstellen, diesen spezifischen Grenzwert zunächst als $f(S_k^*)/k$ zu wählen. Wie Badanidiyuru *et al.* feststellen, ist dieser Ansatz allerdings nicht akzeptabel, falls sich ein einzelnes Element mit Grenzwertüberschreitung am Ende und viele Elemente mit Grenzwertunterschreitung am Anfang des Stroms befinden. In diesem Falle würde die selektierte Menge lediglich aus diesem einzelnen grenzwertüberschreitenden Element bestehen, wodurch eine ungenügende Lösung gefunden wird. Eine Relaxierung des so entworfenen Grenzwertes, bei der beispielsweise nur ein Bruchteil der optimalen Lösung $f(S_k^*)$ herangezogen wird, würde diese Problematik dabei lösen.

Algorithmus 2.2 SIEVE STREAMING bei bekanntem Optimalwert $f(S_k^*)$ [5].

Eingabe: v , sodass $S_k^* \geq v \geq \alpha \cdot S_k^*$.

- 1: $S = \emptyset$
 - 2: **for** $i \in [1, n]$ **do**
 - 3: **if** $\Delta_f(e_i | S) \geq \frac{v/2 - f(S)}{k - |S|}$ **and** $|S| < k$ **then**
 - 4: $S := S \cup \{e_i\}$
 - 5: **end if**
 - 6: **end for**
 - 7: **return** S
-

Diese Intuition kann im folgenden Algorithmus formalisiert werden: Angenommen der Wert der optimalen Lösung $f(S_k^*)$ sei bis zu einem konstanten Faktor $\alpha \in [0,1]$ bekannt. Dann existiert ein v , sodass $f(S_k^*) \geq v \geq \alpha \cdot f(S_k^*)$, was zusammen mit α die oben angedeu-

tete Relaxierung des Grenzwertes darstellt. Wie oben beschrieben, beginnt der Algorithmus dann mit einer leeren Menge S und fügt neue Beobachtungen hinzu, wenn Folgendes gilt:

$$\Delta_f(e_i | S) \geq \frac{v/2 - f(S)}{k - |S|} \quad (2.6)$$

Die vollständige Prozedur wird in Algorithmus 2.2 angegeben, wobei die gebotenen Garantien im folgenden Theorem 2.2.2 dargelegt werden:

2.2.2 Theorem (Eigenschaften des Algorithmus 2.2 [5]). Wird der Algorithmus 2.2 mit den geforderten Eingabeparametern instanziiert, dann erfüllt dieser folgende Eigenschaften:

- Es wird eine Lösungsmenge S gefunden, sodass $|S| \leq k$ und $f(S) \geq \frac{\alpha}{2} f(S_k^*)$.
- Es wird eine Iteration über den Datensatz benötigt und es werden höchstens k Beobachtungen im Speicher abgelegt.

Der beschriebene Algorithmus 2.2 arbeitet mit der Annahme, dass der Optimalwert $f(S_k^*)$ zumindest näherungsweise bekannt ist, was in der Regel nicht der Fall ist. Badanidiyuru *et al.* behelfen sich daher mit der Kenntnis des erreichbaren Maximalwerts der zu optimierenden Funktion gegeben eines einzelnen Elements, formalisiert durch $m = \max_{e \in V} f(\{e\})$. Aus diesem Ausdruck lässt sich dann eine Schätzung für $f(S_k^*)$ angeben, da mit Definition 2.1.2 folgendes gilt:

$$m \leq f(S_k^*) \leq k \cdot m \quad (2.7)$$

Dieser Ausdruck liefert eine obere Schranke und damit eine Näherung an den in Algorithmus 2.2 benötigten Optimalwert $f(S_k^*)$. Ausgehend davon schlagen Badanidiyuru *et al.* zusammen mit der unteren Schranke aus Theorem 2.2.2, $v = km$ und $\alpha = 1/k$ die untere Schranke für die erwartbare Lösung bei Verwendung von Algorithmus 2.2 zu $f(S_k^*)/2k$ vor.

Eine Verbesserung dieser Approximationsgüte lässt sich dadurch erreichen, dass das durch Ausdruck 2.7 aufgespannte Optimalitätsintervall abgetastet wird und so verschiedene „Siebe“ aufgespannt werden, die Beobachtungen mit großem Grenznutzen aussieben [5]. Konkret geschieht dies durch Betrachtung der folgenden Menge O :

$$O = \{(1 + \epsilon)^i \mid i \in \mathbb{Z}, m \leq (1 + \epsilon)^i \leq k \cdot m\} \quad (2.8)$$

Analog zu der erwarteten Eingabe wird von O erwartet, dass ein Grenzwert $v \in O$ existiert, sodass $f(S_k^*) \geq v \geq (1 - \epsilon)f(S_k^*)$, also v eine hinreichend gute Approximation an $f(S_k^*)$ liefert. Damit wird ein weiterer Hyperparameter ϵ eingeführt, der die Qualität dieser Näherung kontrolliert. Zusammen mit der Grenzwertmenge O und den Überlegungen aus Algorithmus 2.2 lässt sich ein neuer Algorithmus angeben, der auf die Kenntnis von $f(S_k^*)$ vollständig verzichtet.

Algorithmus 2.3 SIEVE STREAMING bei bekanntem Maximalwert m [5].

Eingabe: $m = \max_{e \in V} f(\{e\})$.

```

1:  $O = \{(1 + \epsilon)^i \mid i \in \mathbb{Z}, m \leq (1 + \epsilon)^i \leq k \cdot m\}$ 
2: Initialisiere für jedes  $v \in O$  die Menge  $S_v := \emptyset$ .
3: for  $i \in [1, n]$  do
4:   for  $v \in O$  do
5:     if  $\Delta_f(e_i \mid S_v) \geq \frac{v/2 - f(S_v)}{k - |S_v|}$  and  $|S_v| < k$  then
6:        $S_v := S_v \cup \{e_i\}$ 
7:     end if
8:   end for
9: end for
10: return  $\arg \max_{v \in O} f(S_v)$ 

```

Dieser Algorithmus stellt *de facto* eine elegantere Form der Mehrfachinitialisierung des Algorithmus 2.2 bereit. Anstelle für jeden Grenzwert $v \in O$ diesen Algorithmus zu starten und so $\mathcal{O}(|O|)$ Iterationen über die Grundmenge zu benötigen, sind mit diesem Algorithmus nur eine Iteration über die Grundmenge vonnöten. Ferner werden folgende Garantien erfüllt:

2.2.3 Theorem (Eigenschaften des Algorithmus 2.3 [5]). Wird der Algorithmus 2.2 mit den geforderten Eingabeparametern instanziiert, dann erfüllt dieser folgende Eigenschaften:

- Es wird eine Lösungsmenge S gefunden, sodass $|S| \leq k$ und $f(S) \geq (\frac{1}{2} - \epsilon)f(S_k^*)$.
- Es werden höchstens $\mathcal{O}(\frac{\log k}{\epsilon})$ Beobachtungen im Speicher abgelegt.

Zusammen mit Theorem 2.2.3 und der Definition der Grenzwertmenge O aus Gleichung 2.8 lässt sich für die Wahl des Hyperparameters ϵ ein sinnvoller Wertebereich mit $(0, \frac{1}{2})$ angeben, da ansonsten entweder jegliche Gütegarantien verloren gehen oder sich der Ausdruck 2.8 nicht sinnvoll berechnen lässt.

Der aufgestellte Algorithmus 2.3 erfordert nun im Gegensatz zu Algorithmus 2.2 nicht mehr die Kenntnis des Optimalwerts $f(S_k^*)$, verlangt aber die Kenntnis des Maximalwerts m für ein einzelnes Element. Lässt sich dieser nicht aus der konkreten Funktion ableiten, muss dieser aus der Grundmenge V inferiert werden. Eine Möglichkeit dies zu bewerkstelligen ergibt sich durch die Implementierung eines dritten Algorithmus der zwei Iterationen über die gesamte Grundmenge V benötigt: Die erste Iteration ermittelt dabei den Maximalwert m , während die zweite Iteration den Algorithmus 2.3 durchführt. In einem kanonischen Aufbau strömender Daten, in dem jede Beobachtung nur einmal beobachtet werden kann, ist eine solche Vorgehensweise allerdings unzulässig.

Badanidiyuru *et al.* eliminieren diese Problematik, indem ein aktueller Maximalwert m bei Eintreffen von neuen Beobachtungen laufend aktualisiert wird. Dieser Maximalwert wird dann dazu verwendet, um für jede neue Beobachtung eine neue Grenzwertmenge aufzustellen. Jene Ergebnismengen S_v die zu keinem Grenzwert in dieser neuen Grenzwertmenge gehören, werden entfernt. Dies stellt das endgültige SIEVE STREAMING-Verfahren dar, welches in Algorithmus 2.4 beschrieben ist.

Algorithmus 2.4 SIEVE STREAMING [5]

```

1:  $O = \{(1 + \epsilon)^i \mid i \in \mathbb{Z}\}$ 
2: Initialisiere für jedes  $v \in O$  die Menge  $S_v := \emptyset$ .
3:  $m := 0$ 
4: for  $i \in [1, n]$  do
5:    $m := \max(m, f(\{e_i\}))$ 
6:    $O_i = \{(1 + \epsilon)^i \mid m \leq (1 + \epsilon)^i \leq 2 \cdot k \cdot m\}$ 
7:   Entferne alle  $S_v$ , sodass  $v \notin O_i$ .
8:   for  $v \in O_i$  do
9:     if  $\Delta_f(e_i \mid S_v) \geq \frac{v/2 - f(S_v)}{k - |S_v|}$  and  $|S_v| < k$  then
10:       $S_v := S_v \cup \{e_i\}$ 
11:     end if
12:   end for
13: end for
14: return  $\arg \max_{v \in O_n} f(S_v)$ 

```

Der Algorithmus vermeidet nun die Spezifikation jeglicher Informationen über die Daten beziehungsweise die submodulare Funktion und verlangt lediglich die Wahl des Hyperparameters $\epsilon \in (0, \frac{1}{2})$ und einer zu optimierenden Funktion f . Ferner werden folgende Garantien geboten:

2.2.4 Theorem (Eigenschaften des Algorithmus 2.4 [5]). Der Algorithmus 2.4 weist folgende Eigenschaften auf:

- Es wird eine Lösungsmenge S gefunden, sodass $|S| \leq k$ und $f(S) \geq (\frac{1}{2} - \epsilon)f(S_k^*)$.
- Es wird eine Iteration über den Datensatz benötigt und es werden höchstens $\mathcal{O}(\frac{k \log k}{\epsilon})$ Beobachtungen im Speicher abgelegt.

2.2.3 Weitere Optimierungsverfahren

Im vergangenen Abschnitt wurden zwei Algorithmen zur Lösung des Optimierungsproblems 2.3 vorgestellt, nämlich GREEDY und SIEVE STREAMING. Aufgrund der NP-Härte

dieses Problems, stellen beide Lösungsstrategien ein approximatives Resultat bereit. Hierbei wurde dargelegt, dass GREEDY für die Gruppe effizienter Algorithmen das nachweislich beste Resultat bereitstellt, wobei eine Approximationsgüte von mindestens $(1 - e^{-1}) \approx 63,21\%$ erreichbar ist (vgl. Theorem 2.2.1). In den Fällen, in denen der GREEDY-Algorithmus beispielsweise aufgrund des Speicherbedarfs oder der Laufzeit abzulehnen ist, stellt SIEVE STREAMING eine Methode bereit, die auf strömenden Daten operiert. Die hier erreichbare Approximationsgüte tendiert bei kleiner werdendem Hyperparameter ϵ zu 50% (vgl. Theorem 2.2.4).

Ferner existieren weitere Optimierungsmethoden, die das Problem 2.3 lösen können. Ein weiterer Vertreter ist beispielsweise der SALSA-Algorithmus, der von Norouzi-Fard *et al.* eingeführt wurde [46]. Dieser Algorithmus ist ebenfalls ein Datenstromverfahren, wobei SALSA drei verschiedene Prozeduren gleichzeitig auf ein ankommendes Element anwendet und das beste Resultat dieser drei Prozeduren zurückgibt. Diese drei Prozeduren machen dabei verschiedene Annahmen über die Struktur der Optimallösung S^* , wobei ein spezifischer Dichtebegriff verwendet wird. Die Optimallösung gilt dabei als *dicht*, wenn höchstens 1% der Optimalmenge S^* mindestens 10% des Optimallösungswerts $f(S^*)$ erklären. In Abhängigkeit dieser Dichte-eigenschaft lassen sich, ähnlich wie beim SIEVE STREAMING-Verfahren, Grenzwerte entsprechend wählen. Da a-priori nicht bekannt ist, ob die Optimallösung *dicht* ist, werden alle Prozeduren parallel ausgeführt.

Ein weiteres Datenstromverfahren, welches näher an der bereits besprochenen GREEDY-Prozedur liegt, ist der STREAM-GREEDY-Algorithmus von *Gomes* und *Krause* [21]. Dieser Algorithmus nimmt eine Fallunterscheidung auf Grundlage einer Arbeitsmenge A vor, die das aktuelle Ergebnis beschreibt: Sofern A noch nicht so viele Elemente vorhält, wie es die Kardinalitätsbeschränkung k vorsieht, wird jenes Element aus der Menge aller zum jetzigen Zeitpunkt angekommenen Elemente ausgewählt, welches den Nutzen am stärksten maximiert. Ansonsten werden mittels einer gierigen Strategie Elemente aus der Arbeitsmenge ausgetauscht, um den Nutzen zu maximieren.

Ein weiteres zu erwähnendes Verfahren zur Optimierung von submodularen Funktionen auf Grundlage von Datenströmen ist das von *Buschjäger et al.* etablierte THREE SIEVES-Verfahren [9]. Dieses koppelt an den Ideen von *Badanidiyuru et al.* an, berücksichtigt aber lediglich ein Sieb, dessen individueller Grenzwert nicht bei Instanziierung fixiert sondern bei Ablehnung einer vorher zu bestimmenden Anzahl von Beobachtungen angepasst wird.

Welches Optimierungsverfahren letztlich praktisch eingesetzt wird, lässt sich a-priori nur anhand von aufgestellten Anforderungen hinsichtlich der Laufzeit- beziehungsweise Speicherkomplexität oder den Gütegarantien entscheiden. Gerade für ressourcenbeschränkte Systeme oder in Szenarien, die eine definierte Güte hinsichtlich der gefundenen Lösung verlangen, lässt sich die Menge einsatzfähiger Verfahren schnell eingrenzen. Festzuhalten ist aber auch, dass Optimierungsalgorithmen wie GREEDY oder SIEVE STREAMING dem Anwender eine Gütegarantie zur Verfügung stellen, sodass datensatz- und anwendungsab-

hängig vereinzelt auch bessere Ergebnisse beobachtet werden können und sich daher eine empirische Evaluation der Optimierung für eine gegebenen Anwendung anbietet.

Die Betrachtung von GREEDY und SIEVE STREAMING erfolgte an dieser Stelle, da diese in vielerlei Hinsicht grundlegende Ergebnisse im Themenfeld der submodularen Funktionsoptimierung hervorgebracht haben, wozu beispielsweise die maximal-erzielbare Approximationsgüte unter Annahme einer polynomiellen Anzahl an Funktionsauswertungen gehört. Das SIEVE STREAMING-Verfahren fand hier Beachtung da sich dieses gegenüber anderen Datenstromoptimierungsverfahren wie SALSA dadurch auszeichnet, dass eine geringere Anzahl an Hyperparametern zu optimieren sind. Gegenüber dem STREAM-GREEDY-Verfahren ist vorteilhaft, dass die Gütegarantie zwar zu SIEVE STREAMING ähnlich ist, aber explizit erwartet wird, dass die Grundmenge mehrfach im Strom bereitgestellt wird und, dass die submodularen Funktionen einen bestimmten Wertebereich abdeckt, um diese Garantie auch zu erreichen.

2.3 Beispiele

Nachdem im Abschnitt 2.1 die submodularen Funktionen formal etabliert und das wichtige Problem der Optimierung unter Kardinalitätsbeschränkung in Abschnitt 2.2 diskutiert wurde, sollen nun zwei Funktionen exemplarisch vorgestellt werden, nämlich die INFORMATIVE VECTOR MACHINE (IVM) und das EXEMPLAR-BASED CLUSTERING. Hierbei sollen insbesondere formale Aspekte im Vordergrund stehen und begründet werden, wie die verschiedenen submodularen Funktionen die Repräsentativität beziehungsweise die Diversität einer ausgewählten Menge hinsichtlich der zugehörigen Grundmenge messen.

2.3.1 Informative Vector Machine

Ein erstes Beispiel für eine submodulare Funktion stellt die INFORMATIVE VECTOR MACHINE (IVM) dar. Diese ist inspiriert durch ein Problem, welches bei der Betrachtung sogenannter Gaußprozesse entsteht. Diese sollen an dieser Stelle kurz anhand der Ausführungen von *Lawrence* und *Platt* eingeführt werden [36].

Es wird ein Modell unbeobachteter — sogenannter *latent* — Variablen betrachtet: Hierbei sind die Beobachtungen $\vec{y} = (y_1, \dots, y_n)'$ unabhängig von den Eingabedaten $\mathbf{X} = (\vec{x}_1, \dots, \vec{x}_n)'$, wobei zusätzlich ein Vektor latenter Variablen $\vec{f} = (f_1, \dots, f_n)'$ berücksichtigt wird. Die A-priori-Verteilung der latenten Variablen ist dann durch einen Gaußprozess beschrieben:

$$\mathbb{P}(\vec{f} \mid \mathbf{X}, \vec{\theta}) = \mathcal{N}(\vec{0}, \mathbf{K}) \quad (2.9)$$

Hierbei stellt $\mathcal{N}(\vec{\mu}, \mathbf{\Sigma})$ die mehrdimensionale Normalverteilung mit Mittelwertvektor $\vec{\mu}$ und Kovarianzmatrix $\mathbf{\Sigma}$ dar. Die Matrix \mathbf{K} ist dabei eine sogenannte *Kernmatrix*, die

durch einen Vektor $\vec{\theta}$ parametrisiert ist und durch eine Kernfunktion k erzeugt werden kann. Diese stellt dabei für einen Daten- oder Eingaberaum X das Skalarprodukt $\langle \cdot, \cdot \rangle$ in einem sogenannten Merkmalsraum H über eine Abbildung $\Phi : X \rightarrow H$ dar, sodass $k(\vec{x}_i, \vec{x}_j) = \langle \Phi(\vec{x}_i), \Phi(\vec{x}_j) \rangle$ [51]. Für die konkrete Anwendung sind dabei insbesondere die *positiv-definiten Kernel* interessant. *Schölkopf* und *Smola* definieren diese sowohl für den komplexen als auch den reellen Fall, wobei hier nur der reelle Fall, wie folgt, dargestellt wird:

2.3.1 Definition (Kernmatrix [51]). Gegeben sei eine Funktion $k : X \times X \rightarrow \mathbb{R}$ und ein Datensatz $\vec{x}_1, \dots, \vec{x}_n \in X$, dann wird die Matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$ mit den Matrixelementen

$$\mathbf{K}_{i,j} = k(\vec{x}_i, \vec{x}_j) \quad (2.10)$$

die Kernmatrix von k für $\vec{x}_1, \dots, \vec{x}_n \in X$ genannt.

2.3.2 Definition (Positiv-definite Matrix [51]). Eine reelle Matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$ heißt *positiv-definit*, falls diese für alle $c_i \in \mathbb{R}$ die folgende Bedingung erfüllt:

$$\sum_{i,j} c_i c_j \mathbf{K}_{i,j} \geq 0 \quad (2.11)$$

2.3.3 Definition (Positiv-definiter Kernel [51]). Sei X eine nicht-leere Menge. Eine Funktion $k : X \times X \rightarrow \mathbb{R}$, welche für alle $n \in \mathbb{N}$ und alle $\vec{x}_1, \dots, \vec{x}_n \in X$ zu einer positiv-definiten Kernmatrix führt, heißt *positiv-definiter Kernel*. Ein solcher Kernel ist nicht-negativ hinsichtlich der Diagonalelemente ($\forall \vec{x} \in X : k(\vec{x}, \vec{x}) \geq 0$) und symmetrisch ($\forall \vec{x}_i, \vec{x}_j \in X : k(\vec{x}_i, \vec{x}_j) = k(\vec{x}_j, \vec{x}_i)$).

Der Kernel wird dann für jede Beobachtung in \mathbf{X} berechnet und als Kovarianzmatrix für die mehrdimensionale Normalverteilung verwendet. Die gemeinsame Wahrscheinlichkeitsverteilung wird nun wie folgt angegeben:

$$\mathbb{P}(\vec{y}, \vec{f} \mid \mathbf{X}, \vec{\theta}) = \mathbb{P}(\vec{f} \mid \mathbf{X}, \vec{\theta}) \prod_{i=1}^n \mathbb{P}(y_i \mid f_i) \quad (2.12)$$

Hierbei gibt $\mathbb{P}(y_i \mid f_i)$ die Wahrscheinlichkeit für das Auftreten einer Beobachtung gegeben einer latenten Variable an und wird daher auch als Modellierung des Rauschens in den Daten („Rauschmodell“, *engl.* noise model) verstanden. Die Berechnung der marginalisierten Likelihood für das Rauschmodell kann in Abhängigkeit der konkreten Verteilung des Modells allerdings eine Laufzeitkomplexität von $\mathcal{O}(n^3)$ aufweisen.

Ein Verfahren, welches die Zahl einzubeziehender Beobachtungen zu reduzieren versucht und eine möglichst aussagekräftige Repräsentation des Datensatzes finden möchte,

ist die INFORMATIVE VECTOR MACHINE. Hierbei wird eine sogenannte *active set selection* durchgeführt, wobei die ausgewählten Beobachtungen das *active set* darstellen. Die Auswahl erfolgt dabei ähnlich zu den bereits vorgestellten gierigen Algorithmen (s. Abschnitt 2.2), wobei das Entscheidungskriterium diesmal nicht durch den erzielten Nutzensgewinn sondern die Abnahme an *Entropie* dargestellt wird.

Die Entropie steht dabei formal für den Erwartungswert des Informationsgehalts einer Zufallsvariable [50], die als Maß für die zu erwartende Unsicherheit verstanden werden kann [54]. Übertragen auf die von der IVM angestrebte Minimierung der Entropie bedeutet dies, dass wir mittels der „gierigen“ Optimierung die Unsicherheit der in der aktiven Menge enthaltenen Beobachtungen reduzieren wollen und so eine möglichst adäquate also „sichere“ Beschreibung des ursprünglichen Datensatzes erhalten wollen.

Die hier angegebene Beschreibung der IVM lässt bereits vermuten, dass der Prozess der *active set selection* submodulare Eigenschaften nach Definition 2.1.2 aufweist. So wird erwartet, dass die Unsicherheit bei einem leeren *active set* besonders hoch ist und diese durch eine sukzessive Aufnahme von Beobachtungen abnimmt, wobei die Entropie bei größer-werdendem *active set* nur noch langsam abnimmt (vgl. *diminishing returns property*). Tatsächlich ist es so, dass die INFORMATIVE VECTOR MACHINE in einer leicht abgewandelten Form monoton submodular ist. Um dies zu zeigen, wird eine beliebige positiv-definite Matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, wie beispielsweise eine Kernmatrix, betrachtet. Dann ist die Entropie

$$H(\mathbf{A}) = \log \det(\mathbf{A}) \quad (2.13)$$

eine submodulare Funktion [33]. Diese Funktion ist sogar monoton submodular, wenn der kleinste Eigenwert λ_{min} der Matrix \mathbf{A} größer eins ist [55]. Durch die Addition der Identitätsmatrix \mathbf{I} auf \mathbf{A} lässt sich diese Eigenschaft garantieren, womit sich zusammen mit einem Regularisierungsparameter eine monotone submodulare Funktion wie folgt angeben lässt:

2.3.4 Definition (Submodulare Funktion – Informative Vector Machine [5]).

Sei \mathbf{I} die Identitätsmatrix, σ ein Regularisierungsparameter und \mathbf{K}_S die Kernmatrix eines Datensatzes $S \subseteq V \subseteq X$ auf Grundlage eines positiv-definiten Kernels, dann ist die INFORMATIVE VECTOR MACHINE (IVM)

$$f(S) = \frac{1}{2} \log \det(\mathbf{I} + \sigma^{-2} \mathbf{K}^S) \quad (2.14)$$

eine monotone submodulare Funktion.

Die Halbierung der logarithmierten Determinante ist aus mathematischer Perspektive für die Eigenschaft der Submodularität nicht erforderlich, erspart jedoch bei der Implementierung die Durchführung einer zusätzlichen Operation (vgl. Abschnitt 4.1.2).

Die submodulare Funktion der INFORMATIVE VECTOR MACHINE misst die Repräsentativität einer ausgewählten Menge indem versucht wird die Entropie zu erfassen, wozu gerade die Unsicherheit der durch die Repräsentanten gegebenen Erklärungsfähigkeit des Datenraums gehört. Ist die Unsicherheit niedrig, dann liegt idealerweise eine besonders diverse und repräsentative Darstellung der zugrundeliegenden Daten vor. Zu beachten ist hierbei, dass diese Unsicherheit auf Grundlage der Kernmatrix \mathbf{K}^S gemessen wird und dieser daher eine kritische Bedeutung zukommt: So spiegelt der positiv-definite Kernel mittels des Skalarprodukts inhärent eine Ähnlichkeitsfunktion wieder, die sich in Abhängigkeit des Kerns steuern lässt. Dies führt im Wesentlichen darauf, dass ein Kernel identifiziert werden muss, der zu den Daten passt: Unbedingt zu vermeiden sind jene Kernel (beziehungsweise jene *parametrisierte* Kernel), die jede Beobachtung zu sich selbst maximal ähnlich aber zu jeder anderen Beobachtung maximal *unähnlich* erscheinen lassen. Aufgrund der Unähnlichkeit beliebiger Beobachtungspaare wird bei arbiträrer Aufnahme von Beobachtungen in eine Menge die Entropie stets minimiert und so die durch die Funktion gemessene Repräsentativität maximiert, obwohl tatsächlich keine repräsentative Teilmenge des Ursprungsdatensatzes gefunden wurde.

2.3.2 Exemplar-based Clustering

Clustering stellt eine klassische Lernaufgabe dar, die eng mit dem Begriff des unüberwachten Lernens verknüpft ist. Informell bezeichnet Clustering das Partitionieren eines gegebenen Datensatzes in Subgruppen, wobei Beobachtungen der gleichen Gruppe möglichst ähnlich und Beobachtungen anderer Gruppen möglichst unähnlich sein sollen. *Ähnlichkeit* wird dabei im Kontext des Clusterings über Distanzfunktionen ausgedrückt, die zwei Beobachtungen einen reellen Wert (die Distanz) zuordnen. Eine formale Definition dieser Lernaufgabe kann wie folgt angegeben werden:

2.3.5 Definition (Clustering [65]). Sei X ein Datenraum und $V \subseteq X$ eine Menge von Beobachtungen. Ferner sei $d : X \times X \rightarrow \mathbb{R}^+$ eine Distanzfunktion, die als solche folgende Eigenschaften erfüllt:

1. $\forall \vec{x} \in X : d(\vec{x}, \vec{x}) = 0$
2. $\forall \vec{x}_1, \vec{x}_2 \in X : d(\vec{x}_1, \vec{x}_2) = d(\vec{x}_2, \vec{x}_1)$

Weiterhin sei $q : \mathcal{P}(\mathcal{P}(X)) \rightarrow \mathbb{R}$ eine Qualitätsfunktion. Die Lernaufgabe des *Clusterings* ist dann, eine Partitionierung $\mathcal{C} = \{C_1, \dots, C_k\}$ zu finden, wobei $\forall i : C_i \subseteq V$ und $q(\mathcal{C})$ maximiert wird.

Es existieren vielfältige Algorithmen, die diese Lernaufgabe lösen. Unterscheidungsmerkmale ergeben sich dabei häufig in der Fähigkeit nicht-konvexe Cluster zu identifizieren, im Laufzeitverhalten und der Notwendigkeit die Anzahl der zu findenden Cluster anzugeben.

Eine Lernaufgabe des Clusterings ist k -Medoids. Dieses Verfahren wählt aus dem zu clusternden Datensatz V mithilfe einer beliebigen Distanzfunktion $d : X \times X \rightarrow \mathbb{R}^+$ genau k Punkte aus, die die sogenannte Menge der Clusterzentren S darstellt. Die Auswahl erfolgt dabei durch Minimierung folgender k -Medoids-Verlustfunktion [21]:

$$L(S) = \frac{1}{|V|} \sum_{\vec{v} \in V} \min_{\vec{s} \in S} d(\vec{v}, \vec{s}) \quad (2.15)$$

Die Partitionierung des Datensatzes erfolgt dann durch Betrachtung jeder Beobachtung aus V , die dann dem Cluster zugeordnet wird, dessen Clusterzentrum am nächsten liegt. Algorithmen wie *Partitioning Around Medoids* (PAM) lösen dieses Problem direkt [32]. Alternativ kann dieses Problem auch durch Optimierung einer submodularen Funktion gelöst werden, die zusammen mit den bereits vorgestellten Optimierungsalgorithmen eine Menge von Clusterzentren ermittelt und wie folgt angegeben wird:

2.3.6 Definition (Submodulare Funktion – Exemplar-based Clustering [21]).

Sei X ein Datenraum, $V \subseteq X$ eine Menge von Beobachtungen, $\vec{e}_0 \in X$ ein Hilfsvektor (beispielsweise der Nullvektor [5]), $S \subseteq V$ und $L(S)$ die in Gleichung 2.15 definierte Verlustfunktion des k -Medoids-Verfahren, dann ist

$$f(S) = L(\{\vec{e}_0\}) - L(S \cup \{\vec{e}_0\}) \quad (2.16)$$

eine monotone submodulare Funktion.

Im Zusammenhang mit Definition 2.3.6 muss die in der Verlustfunktion verwendete Distanzfunktion lediglich nicht-negativ sein [21].

Die Messung der Repräsentativität einer ausgewählten Menge wird technisch mittels Funktion 2.15 und der Messung der mittleren Distanz aller Vektoren zu ihrem nächsten Repräsentanten umgesetzt: Sind diese Distanzen im Mittel sehr hoch – also gerade dann, wenn Vektoren zu ihrem nächsten Repräsentanten äußerst unähnlich sind – dann werden viele Vektoren nicht angemessen repräsentiert und die gefundene Menge ist als eher nicht repräsentativ anzusehen. In diesem Falle steigt der Wert der k -Medoids-Verlustfunktion. Der Ausdruck 2.16 misst nun gerade die Differenz zwischen einer äußerst unrepräsentativen Menge, gegeben durch einen Hilfsvektor \vec{e}_0 beziehungsweise den Nullvektor, und der tatsächlich ausgewählten Menge an Repräsentanten. Je größer diese Differenz ausfällt, desto größer ist auch die Repräsentativität der parametrisierten Menge S . Offensichtlich ist so auch die gemessene Repräsentativität am größten, wenn $S = V$, da

$L(S) = 0$. Umgekehrt ist die Repräsentativität der Menge S am geringsten, wenn $S = \emptyset$, da $f(S) = L(\{\vec{e}_0\}) - L(\{\vec{e}_0\}) = 0$.

Kapitel 3

Ausreißerererkennung

Der Begriff des *Ausreißers* stellt eine Eigenschaft dar, die sich einer Beobachtung als Teil eines Datensatzes zuordnen lässt. Im Rahmen von praktischen Anwendungen lässt sich häufig eine klare intuitive Vorstellung einer *ausreißenden* Beobachtung formulieren, die dann gelegentlich *Anomalie* genannt wird und eine — wie auch immer geartete und idealerweise wohldefinierte — Abweichung von einer *Normalität* beschreibt. Ein Beispiel hierfür sei die Betrugserkennung in einem Strom von Kreditkartentransaktionen: Eine hohe Transaktion im US-amerikanischen Raum stellt für einen Kunden, der üblicherweise Waren geringen Werts in Europa erwirbt, eine Abweichung dessen dar, was für diese Person üblich ist und könnte demnach betrügerisch sein. Ein potenzieller Kreditkartenanbieter könnte diese Transaktion daher zum Schutze des Kunden blockieren.

Abweichend von dieser praktischen Vorstellung eines Ausreißers, soll in diesem Kapitel ein Blick darauf geworfen werden, wie sich dieser Begriff theoretisch etablieren lässt. Hierfür soll in Abschnitt 3.1 eine Definition des Begriffs vorgestellt werden. Anschließend werden in Abschnitt 3.2 und 3.3 die notwendigen Aspekte besprochen, um Methoden zur Ausreißerererkennung umfassend zu charakterisieren. In den darauffolgenden Abschnitten 3.4 und 3.5 werden dann klassische und submodulare Verfahren zur Erkennung von Ausreißern vorgestellt und mittels der zuvor vorgestellten Strukturierung eingeordnet. Des Weiteren wird ein Blick auf die asymptotische Laufzeitkomplexität der Ausreißererkenntnisverfahren geworfen, die ein wichtiges Charakteristikum zur Beurteilung der erwartbaren Laufzeit darstellt.

3.1 Definition

In der Einleitung dieses Kapitels wurde beschrieben, wie sich eine intuitive Vorstellung eines Ausreißers in einem praktischen Kontext entwickeln lässt. In der nun folgenden theoretischeren Betrachtung sei lediglich ein Datensatz verfügbar, der auf Ausreißer untersucht

werden soll. Auch hier lässt sich zunächst eine informelle Idee dessen formulieren, wie eine ausreißende Beobachtung charakterisierbar ist, wie *Hawkins* im Jahr 1980 beschrieb:

3.1.1 Definition (Ausreißer – intuitiv [28]). Ein Ausreißer ist eine Beobachtung, die so stark von anderen Beobachtungen abweicht, dass diese den Verdacht erweckt, sie sei durch einen anderen Mechanismus generiert worden.

Welcher Beobachtung das Attribut des *Ausreißers* zugeordnet werden kann, hängt also von zwei wesentlichen Faktoren ab, die es zu definieren gilt: Einerseits davon, *wie* eine Beobachtung von anderen Beobachtungen abweicht und andererseits davon *wie stark* die Abweichung ist. Diese Faktoren stützen sich im Wesentlichen auf die Analyse der Beobachtungen eines (wie auch immer gearteten) Zufallsprozesses ab, die in einem Datensatz zusammengefasst werden. Eine weitere Möglichkeit sich dem Begriff des *Ausreißers* zu nähern, ist nicht die generierten Beobachtungen zu beleuchten sondern die *datengenerierenden Mechanismen* zu charakterisieren.

Diese Mechanismen werden üblicherweise durch *Wahrscheinlichkeitsverteilungen* beschrieben, die angeben, wie wahrscheinlich das Auftreten von bestimmten Beobachtungen ist. *Hawkins* beschreibt anhand dessen zwei Möglichkeiten, wie Ausreißer in einem Datensatz entstehen können.

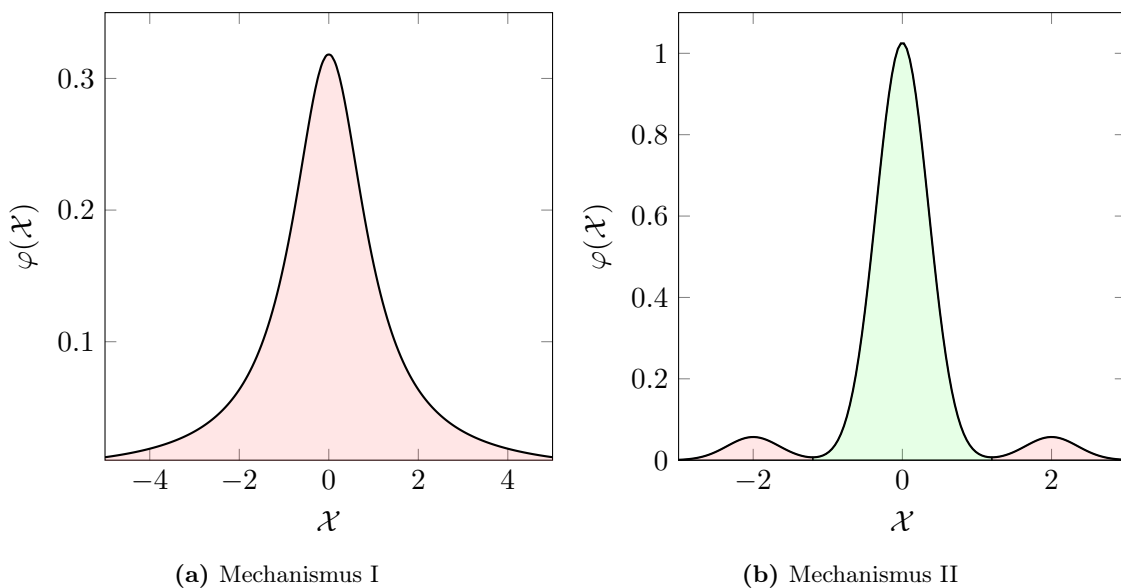


Abbildung 3.1: Visualisierung der beiden ausreißergenerierenden Mechanismen nach *Hawkins* gegeben einer Zufallsvariable \mathcal{X} und der dazugehörigen Verteilung $\varphi(\mathcal{X})$. (a) Es existiert lediglich eine Wahrscheinlichkeitsverteilung, wobei die dazugehörige Dichtefunktion sich an den Rändern nur langsam dem Wert null nähert. (b) Es existiert eine Verteilung, die *gute* Beobachtungen produziert (grün) und eine Verteilung, die *schlechte* Beobachtungen produziert (rot).

Mechanismus I Die Daten stammen aus einer Wahrscheinlichkeitsverteilung, deren Werte der Dichtefunktion sich an den Rändern nur langsam dem Wert null nähern (beispielsweise eine studentsche t -Verteilung). Dabei steht es außer Frage, dass jede Beobachtung in irgendeiner Form fehlerhaft ist [28] (s. Abbildung 3.1a). Davon ausgehend etabliert *Hawkins* ein Konzept von ausreißer-resistenten und ausreißer-anfälligen Verteilungen, wobei die ausreißer-anfälligen Verteilungen genau die Eigenschaft der sich an den Rändern nur langsam dem Wert null annähernden Wahrscheinlichkeitsdichten aufweisen. Wann einer Verteilung die Eigenschaft der Ausreißeranfälligkeit zuzuordnen ist, wird von *Hawkins* formal etabliert [28], hier allerdings nicht näher beleuchtet.

Mechanismus II Der zweite Mechanismus nach *Hawkins* beruht auf der Idee, dass die in einem Datensatz vorhandenen Daten aus *zwei* Wahrscheinlichkeitsverteilungen stammen [28]. Dabei erzeugt eine Verteilung, mit der dazugehörigen Dichtefunktion f_g , die *guten* Beobachtungen und eine weitere Verteilung mit der Dichtefunktion f_s die *schlechten* Beobachtungen. Dann ist die Dichtefunktion der Gesamtverteilung f , gegeben der Wahrscheinlichkeit $p \in [0, 1]$, dass eine Beobachtung *schlecht* ist, durch folgenden Ausdruck definiert:

$$f(x) = (1 - p)f_g(x) + pf_s(x) \quad (3.1)$$

Ein Beispiel für diesen Mechanismus ist in Abbildung 3.1b gegeben: Hierbei erzeugt eine Normalverteilung $\mathcal{N}(0; 0,35)$ die *guten* Beobachtungen. Die *schlechten* Beobachtungen werden durch eine Komposition zweier Normalverteilungen $\mathcal{N}(-2; 0,35)$ und $\mathcal{N}(2; 0,35)$ — eine *Gaußsche Mischverteilung* — erzeugt, die zu gleichen Teilen die resultierende Verteilung dominieren. Mit $p = 10\%$ und Gleichung 3.1 ergibt sich dann genau die Dichtefunktion, die in Abbildung 3.1b sichtbar ist.

3.2 Lernaufgaben

Nachdem im vergangenen Abschnitt der Fokus auf den Mechanismen lag, die dafür verantwortlich sein können, dass Ausreißer in Daten beobachtet werden können, soll nun die methodische Detektion von Ausreißern in den Vordergrund rücken. Neben den tatsächlichen Detektionsansätzen, ist dabei zur korrekten Charakterisierung der vorliegenden Methode vor allem die *Lernaufgabe* von entscheidender Bedeutung: Diese lässt sich insbesondere intuitiv durch die Frage danach charakterisieren, was eine durch eine Methode gelernte Funktion praktisch leisten soll. Ferner lässt sich formal untersuchen, wie eine gelernte Funktion strukturiert ist, also welche Definitions- und welche Zielmenge unterstützt wird und welche Eingabedaten benötigt werden, um diese Funktion zu finden. Im Folgenden werden die für die Ausreißerererkennung wichtigen Aspekte zur Bestimmung der Lernaufgabe wiedergegeben.

Novelty Detection vs. Ausreißerererkennung In dieser Arbeit liegt der Fokus auf der *Ausreißerererkennung*, die zunächst von der sogenannten *Novelty Detection* abzugrenzen ist. Die Aufgabe der Ausreißerererkennung liegt darin, auf Grundlage eines Datensatzes zu entscheiden, ob individuelle Beobachtungen ausreißend sind oder nicht. Hierbei wird angenommen, dass der zu untersuchende Datensatz bereits ausreißende Beobachtungen enthält und es diese daher zu identifizieren gilt. Dies unterscheidet sich wesentlich von der Novelty Detection, die auf Grundlage eines Datensatzes *bekannte* Muster und Regionen charakterisiert und davon abweichend bei Modellanwendung neue oder unbekannte Beobachtungen, die während des Trainingsprozesses nicht vorhanden waren, identifiziert [41]. Dabei ist es natürlich möglich, dass Novelty Detection auch zur Erkennung von Ausreißern verwendet wird: Voraussetzung ist dabei jedoch, dass beim Training eines entsprechenden Novelty Detection-Modells ausschließlich *normale* Beobachtungen verwendet werden, die bei der Modellapplikation entsprechend zur Unterscheidung herangezogen werden. Da es bei hochvolumigen und hochdimensionalen Prozessen oftmals schwierig ist, gerade diese für die Modellselektion essentiellen Informationen über die Ausreißereigenschaft individueller Beobachtungen zu bestimmen, soll im weiteren Verlauf ausschließlich auf die Lernaufgabe der Ausreißerererkennung Bezug genommen werden.

Bewertungsräume Grundlage der Ausreißerererkennung ist die Identifikation einer Funktion $f : X \rightarrow B$ anhand eines Datensatzes \mathcal{D} , die einer Beobachtung aus einem beliebigen Datenraum X eine Bewertung aus dem Bewertungsraum B zuordnet. Dieser Bewertungsraum wird von Methoden zur Ausreißerererkennung unterschiedlich definiert, wobei zwei Varianten vorherrschend sind: Zum einen können Beobachtungen eine rein binäre Ausreißereigenschaft zugeordnet werden, wobei diese dann entweder als *ausreißend* oder *nicht-ausreißend* markiert werden. Eine formale Etablierung dieser binären Markierung von Ausreißern könnte durch $B = \{+1, -1\}$ gegeben sein, bei dem „-1“ einen Ausreißer signalisiert, während „+1“ eine Normalbeobachtung anzeigt. Methoden, die eine binäre Markierung von Ausreißern durchführen, haben den Vorteil eine definitive Bewertung einer Beobachtung durchzuführen, die wenig Interpretationsspielraum übrig lässt. Manchmal ist es jedoch wünschenswert, die durch die Methode getroffene Entscheidung in einem weiteren Schritt entweder manuell (beispielsweise durch den Bediener einer Maschine) oder automatisch (beispielsweise durch die Wahl eines empirisch gewählten Grenzwerts) bewerten zu lassen. In diesem Falle kann es empfehlenswert sein, auf Methoden zurückzugreifen, die eine reelle Ausreißerbewertung (also $B = \mathbb{R}$) vornehmen. Eine so für eine spezifische Beobachtung quantifizierte Ausreißereigenschaft kann nun interpretiert werden, wobei genau geprüft werden muss, wie die verwendete Methode diesen Wert errechnet hat, um besagte Interpretation adäquat zu gestalten.

Datensätze Es soll nun ein Blick auf den zur Identifikation der Funktion f benötigten Datensatz \mathcal{D} geworfen werden. Nach *Goldstein* und *Uchida* können hierbei drei Varianten unterschieden werden [20]: Die erste Variante beschreibt die *überwachte Ausreißererken-*
nung, bei der markierte Trainingsdaten zu Verfügung stehen. Rein formal wird hierbei ein Datensatz $\mathcal{D} = \mathcal{D}_L = \{(\vec{x}_1, y_1), \dots, (\vec{x}_n, y_n)\} \subset X \times B$ angenommen, wobei X dem bereits bekannten Datenraum und B dem Bewertungsraum entspricht. Diese Lernaufgabe ist sehr ähnlich zu einer klassischen Klassifikationsaufgabe, wobei die starke Ungleichverteilung der im Datensatz enthaltenen Klassen (d.h. *viele* normale Beobachtungen, *wenige* ausreißende Beobachtungen) ein zusätzlicher Teil der Problemformulierung darstellt und die Menge der anwendbaren Methoden limitiert [20].

Die *halbüberwachte Ausreißererken-*
nung ist äquivalent zu der obigen informellen Beschreibung der *Novelty Detection*, falls diese zur Ausreißererken-

nung angewendet werden soll. Dabei soll jedoch betont werden, dass die halbüberwachte Ausreißererken-

nung keineswegs äquivalent zur *Novelty Detection* ist, da diese theoretisch beliebige Regionen von anderen Regionen abgrenzen kann, wobei diese Regionen nicht notwendigerweise *Normalre-*
gionen sein müssen. Die halbüberwachte Ausreißererken-

nung impliziert gerade diese Identifikation von Normalregionen, die anhand eines unmarkierten Datensatz, der ausschließlich *normale* Beobachtungen enthält, geschätzt wird. Formal kann dieser Datensatz durch $\mathcal{D} = \mathcal{D}_{UL}^+ = \{\vec{x} \in X \mid \vec{x} \text{ ist normal bzw. kein Ausreißer}\} \subset X$ angegeben werden. Die Bewertung erfolgt dann durch Prüfung, ob eine fragliche Beobachtung zu dem gelernten Modell von *Normalität* passt oder nicht.

Die *unüberwachte Ausreißererken-*
nung, auf welcher der Fokus in dieser Arbeit liegt, macht keine Annahme über den präsentierten Datensatz und schätzt die Ausreißereigen-

schaft für eine gegebene Beobachtung anhand der intrinsischen Eigenschaften des Datensatzes, wozu beispielsweise Dichte oder Distanzen gehören [20]. Der Datensatz \mathcal{D} ist un-

markiert, also $\mathcal{D} = \mathcal{D}_{UL} = \{\vec{x}_1, \dots, \vec{x}_n\} \subset X$. Da keine Informationen hinsichtlich der Ausreißereigen-

schaft von präsentierten Beobachtungen erforderlich ist, gehört die unüber-

wachte Ausreißererken-

nung zu den flexibelsten der vorgestellten Lernaufgaben, verlangt aber auch, dass das gewonnene Modell beziehungsweise die gelernte Funktion ausgiebig validiert wird. Im Gegensatz zu den überwachten Lernmethoden stehen hier nämlich keine „Wahrheitsinformationen“ in Form von Labeln zu Verfügung, die bei der Modellselektion einbezogen werden können.

3.3 Strukturierung

Die Frage nach der Lernaufgabe konzentriert sich vor allem auf die streng theoretischen Aspekte dessen, was bei einer Ausreißererken-

nung geleistet werden soll: Der Fokus liegt auf Formalia, wie der Strukturierung des Lerndatensatzes und der durch eine Methode gele-

isteten Bewertung. Für eine nähere Charakterisierung eines Verfahrens soll erneut der Blick

auf *Hawkins'* Definition eines Ausreißers geworfen werden (s. Definition 3.1.1): Zwar ist diese Definition intuitiv greifbar, allerdings erlaubt sie nicht, einen Ausreißer mit absoluter Sicherheit und unabhängig von dem Verfahren als solchen zu kennzeichnen. Methoden, die eine Ausreißerererkennung bewerkstelligen wollen, müssen daher diese Definition gewissermaßen *implementieren* und realisieren so ein individuelles Verständnis dafür, wie sich ein Ausreißer präsentiert. Dies wird ausgenutzt, um Verfahren, die ein ähnliches Verständnis implementieren, gemeinsam zu gruppieren und dementsprechend andere Verfahren voneinander abzutrennen. Dies ist nicht nur sinnvoll, um eine „Taxonomie von Methoden zur Ausreißerererkennung“ aufzubauen, sondern auch in einem rein praktischen Umfeld von Bedeutung: So lassen sich beispielsweise für ein Problem geeignete Verfahren selektieren, wobei klassische Kriterien, wie Laufzeit, Speichernutzung und Interpretierbarkeit von Bedeutung sein können.

Im Folgenden soll daher eine Gruppierung von Methoden der unüberwachten Ausreißerbeziehungsweise Anomalieerkennung präsentiert werden, wie sie gemäß *Goldstein* und *Uchida* [20] beziehungsweise *Chandola et al.* [11] aufgestellt wurde:

1. **Nächste Nachbarn-basiert:** Unter Verwendung eines Distanz- oder Ähnlichkeitsmaßes arbeiten diese Methoden auf Grundlage der Annahme, dass *normale* Beobachtungen in *dichteren* Regionen auftreten und Ausreißer deutlich weiter von ihren nächsten Nachbarn entfernt sind. Eine reelle Ausreißerbewertung findet dann dadurch statt, dass entweder die Distanz einer Beobachtung zu ihrem *i*-nächsten Nachbarn herangezogen wird oder aber die relative Dichte jeder Beobachtung berechnet wird.
2. **Clustering-basiert:** Zum Wesen des *Clusterings* gehört es einen Datensatz derart zu partitionieren, dass ähnliche oder nah-beieinander liegende Beobachtungen in jeweils eine eigene Menge (dem *Cluster*) eingeordnet werden (vgl. Definition 2.3.5). Um auf Grundlage eines identifizierten Clusterings eine Ausreißerererkennung durchzuführen, werden Annahmen aufgestellt von denen *Chandola et al.* drei unterscheidet [11]: Die erste Annahme ist, dass alle *normalen* Beobachtungen sich in einem Cluster befinden während *ausreißende* Beobachtungen zu keinem Cluster gehören. Die zweite mögliche Annahme ist, dass normale Beobachtungen sich in relativer Nähe zum mittleren Punkt des Clusters — dem sogenannten *Zentroiden* — befinden während Ausreißer weiter von diesem entfernt liegen. Die dritte Annahme zeichnet sich durch die Idee aus, dass normale Beobachtungen zu großen und dichten Clustern gehören, während Ausreißer eher in kleinen oder dünn besiedelten Clustern eingeordnet werden.
3. **Statistische Methoden:** Diese Gruppe von Methoden arbeitet im Wesentlichen mit *Hawkins'* Vorstellungen dessen, wie verschiedene Mechanismen Ausreißer in Datensätzen erzeugen (vgl. Abschnitt 3.1). Die Schlüsselannahme dieser Methoden ist daher, dass normale Beobachtungen in tendenziell wahrscheinlicheren Bereichen eines

zugrundeliegenden stochastischen Modells liegen, während Ausreißer in tendenziell eher unwahrscheinlicheren Bereichen zu finden sind [11].

4. **Spektral-/ Unterraum-basiert:** Diese Methoden suchen eine Projektion der zu untersuchenden Daten in einen kleiner dimensionierten Unterraum, in dem die ursprünglichen Merkmale durch eine angemessene Kombination approximativ dargestellt werden [11]. Die Annahme ist hierbei, dass durch die Projektion in einen Unterraum Normal- und Ausreißerbeobachtungen deutlich anders auftreten [11] und sich damit leichter voneinander trennen lassen.
5. **Klassifikations-basiert:** Klassifizierer sind üblicherweise überwachte oder mindestens halb-überwachte Lernverfahren. *Goldstein* und *Uchida* benennen dennoch eine Methode, die als unüberwachtes Verfahren ein Klassifikationsmodell bestimmt, nämlich die *One-Class SVM* [52]: Dieses Lernverfahren lernt auf Grundlage eines Datensatzes eine Trennebene, die alle Beobachtungen von der Komplementmenge separiert, sodass sich feststellen lässt, ob eine neue Beobachtung in diese Region gehört oder nicht. Dies ist konform mit der oben formulierten Vorstellung der *Novelty Detection*. Erlaubt man bei der Modellbestimmung jedoch, dass einzelne Beobachtungen auch außerhalb dieser Trennebene liegen dürfen, wird aus einer *harten Trennebene* eine *weiche Trennebene*. Die Hoffnung ist dann, dass die Ausreißer gerade außerhalb der Trennebene liegen und sich so identifizieren lassen. So wird eine unüberwachte Ausreißerererkennung ermöglicht.

Dabei sei jedoch erwähnt, dass diese Kategorisierung von Verfahren nicht unbedingt absolut scharf ist und sich Algorithmen teilweise auch in mehr als eine Kategorie einordnen lassen.

3.4 Klassische Methoden

Nachdem nun im Kontext der Ausreißerererkennung die möglichen Lernaufgaben und die Strukturierung der vorhandenen Methoden präsentiert wurde, sollen im nun folgenden Abschnitt klassische Methoden präsentiert werden. Hierzu wird im Abschnitt 3.4.1 der ISOLATION FOREST und in Abschnitt 3.4.2 der LOCAL OUTLIER FACTOR vorgestellt, die zwei populäre Vertreter unüberwachter Lernverfahren zur Ausreißerererkennung darstellen. Abschließend sollen in Abschnitt 3.4.3 die beiden Verfahren kurz zusammengefasst, eingeordnet und diskutiert werden.

3.4.1 Isolation Forest

Die Methode des ISOLATION FOREST (IF) stellt eine unüberwachte Methode zur Ausreißerererkennung dar, die im Jahr 2008 von Liu *et al.* vorgeschlagen wurde und grundsätzlich die

Intuition verfolgt, Ausreißer in einem Datensatz zu isolieren anstatt die *normalen* Regionen zu charakterisieren [40]. Hierfür werden zufällige Baumstrukturen aufgebaut, die ähnlich zu gewöhnlichen Indexstrukturen, wie beispielsweise k - d -Bäumen, eine Partitionierung des Datenraums induzieren. Aus der Strukturierung der jeweils generierten Bäume leiten Liu *et al.* eine reelle Ausreißerbewertung für jede individuelle Beobachtung ab. Formal wird für einen gegebenen ungelabelten Datensatz $\mathcal{D}_{UL} = \{\vec{x}_1, \dots, \vec{x}_n\} \subset X$ eines beliebigen Datenraums X eine Bewertungsfunktion $r : \mathcal{D}_{UL} \rightarrow \mathbb{R}$ gefunden, die einer Beobachtung nicht nur eine binäre Ausreißereigenschaft bescheinigt sondern auch angibt, wie stark die Beobachtung innerhalb des Datensatzes ausreißt.

Als Vertreter der sogenannten *Ensemble*-Methoden kombinieren IFs Modelle eines Basislernalers, die häufig als *schwach* bezüglich ihrer Fehlerrate charakterisiert werden [27], zu einem *starken* Metamodell. Dieses Metamodell trifft die Klassifikations- oder Regressionsentscheidung (jeweils bei Vorhersage von diskreten oder kontinuierlichen Werten) durch die gewichtete oder ungewichtete Entscheidung der jeweiligen Basismodelle, die für ein bestimmtes Ergebnis „votieren“. Motiviert werden diese Methoden dabei häufig durch eine verbesserte Vorhersagegenauigkeit, die bei Nutzung eines einzelnen Modells unter Umständen nicht zu erzielen wäre [26]. Konkret handelt es sich beim ISOLATION FOREST um eine sogenannte *Bagging*-Ensemblemethode, bei der zur Bestimmung jedes einzelnen Basismodells lediglich eine Teilmenge des gesamten Datensatzes herangezogen wird [27].

Isolation Trees

Das in dem Falle der IFs verwendete Basismodell ist der sogenannte ISOLATION TREE (IT). Dieser arbeitet mit der Intuition, dass Ausreißer, die tendenziell seltener als normale Beobachtungen auftreten, von anderen Beobachtungen isoliert werden können. Bei einer Isolation aller Punkte eines Datensatz in eine eigene Partition beobachten Liu *et al.* empirisch, dass in einem zufällig generierten Baum, der die Partitionierung des Raums beschreibt, für ausreißende Beobachtungen deutlich kürzere Pfade von der Wurzel bis zur entsprechenden Partition zu erwarten sind [40]. Diese Beobachtung soll zur Ausreißerererkennung ausgenutzt werden und hierfür der IT formal wie folgt definiert werden:

3.4.1 Definition (Isolation Tree (IT) [40]). Sei T der Knoten eines IT, dann ist T entweder ein *externer Knoten* ohne Kindknoten oder aber ein *interner Knoten* mit genau zwei Kindknoten und einer Testbedingung. Eine Testbedingung besteht dabei aus einem Merkmal q und einem Splitwert p , sodass die Testbedingung die Beobachtungen des internen Knotens T in den linken Kindknoten T_l und den rechten Kindknoten T_r aufteilt.

Ein IT ist dabei stets ein Binärbaum mit entweder genau zwei Kindknoten oder aber keinen Kindknoten, der durch rekursive Aufteilung eines ungelabelten Datensatzes $\mathcal{D}_{UL} =$

$\{\vec{x}_1, \dots, \vec{x}_n\} \subset X$ derart aufgebaut wird, dass jeweils zufällig ein Merkmal und ein Splitwert bestimmt wird, der die Partitionierung bestimmt. Die Rekursion bricht in einem Knoten genau dann ab, wenn entweder nur noch eine Beobachtung in der Partition liegt, der Baum eine vorher definierte Höhe erreicht hat oder alle Beobachtungen in der Partition identisch sind.

Auf Grundlage des ITs soll nun eine Anomalieerkennung hinsichtlich einer gegebenen Beobachtung durchgeführt werden: Wie bereits beschrieben, sollen hierzu die Pfadlängen genutzt werden, die bei anomalen Beobachtung besonders kurz sind. Der Begriff der *Pfadlänge* wird dabei wie folgt definiert:

3.4.2 Definition (Pfadlänge [40]). Sei $\vec{x} \in \mathcal{D}_{\text{UL}}$ eine Beobachtung, dann ist $h(\vec{x})$ die Pfadlänge dieser Beobachtung, definiert durch die Anzahl der Kanten, die von der Wurzel bis zum externen Knoten eines gegebenen IT traversiert werden.

Um nun anhand der Pfadlängen den Grad der Anomalie für eine gegebene Beobachtung zu quantifizieren, soll ein Blick auf die *durchschnittliche Pfadlänge* $c(n)$ bei n Beobachtungen eines IT geworfen werde. Diese wird aus der durchschnittlichen Pfadlänge bei erfolgloser Suche in einem binären Suchbaum abgeleitet und durch Liu *et al.* mit folgendem Ausdruck angegeben [40]:

$$c(n) = 2H(n-1) - (2(n-1)/n) \quad (3.2)$$

Dabei repräsentiert $H(n)$ die n -te Partialsumme der harmonischen Reihe.

Mittels des Ausdrucks $c(n)$ kann nun eine Normalisierung der Pfadlängen von der Wurzel bis zu einem Blattknoten durchgeführt werden, um sowohl unter- als auch überdurchschnittliche Abweichungen festzustellen und so eine Anomalie bezüglich aller Beobachtung jenes Blattknotens zu quantifizieren. Liu *et al.* benennen hierfür eine Scoringfunktion, die wie folgt definiert wird:

3.4.3 Definition (Anomaliescore [40]). Sei $\vec{x} \in \mathcal{D}_{\text{UL}}$ eine Beobachtung, $n = |\mathcal{D}_{\text{UL}}|$, $h(\vec{x})$ die Pfadlänge der Beobachtung \vec{x} und $c(n)$ die durchschnittliche Pfadlänge eines IT gemäß Gleichung 3.2, dann ist

$$s(\vec{x}, n) = 2^{-\frac{E(h(\vec{x}))}{c(n)}} \quad (3.3)$$

der Anomaliescore der Beobachtung \vec{x} . Dabei ist $E(h(\vec{x}))$ eine durchschnittliche Pfadlänge gegeben mehrerer ITs.

Der Anomaliescore bildet denn auf die Menge $(0, 1]$ ab, falls $0 < h(x) \leq n-1$, wobei sich gemäß Liu *et al.* für diesen Score nun drei Aussagen machen lassen [40]:

1. Falls s für gegebene Beobachtungen *sehr nah* an 1 liegen, dann sind diese definitiv Ausreißer.

2. Falls s für gegebene Beobachtungen *deutlich kleiner* als 0,5 ist, dann können diese *ziemlich sicher* als normale Beobachtungen angesehen werden.
3. Falls für alle Beobachtungen \vec{x} der Anomaliescore $s(\vec{x}, n) \approx 0,5$, dann enthält der gesamte Datensatz keinen trennbaren Ausreißer.

Für das eingangs vorgestellte *Ensemble* von ITs, muss nun noch geklärt werden, wie eine Menge von ITs gemeinsam zu einem ISOLATION FOREST (IFs) trainiert werden und wie die Modellapplikation vonstatten geht.

Training des Isolation Forest

Zum Trainieren eines IFs wird, wie auch schon beim IT, ein ungelabelter Datensatz $\mathcal{D}_{UL} = \{\vec{x}_1, \dots, \vec{x}_n\} \subset X$ über einem Datenraum X betrachtet. Für eine gegebene Anzahl der zu trainierenden IT $t \in \mathbb{N}$, wird aus dem Datensatz \mathcal{D}_{UL} eine Stichprobe der Größe $\psi \in \mathbb{N}$ gezogen, wobei diese Stichprobengröße ebenfalls vom Benutzer zu spezifizieren ist. Diese Stichprobe (engl. *sample*) wird für jeden zu trainierenden IT separat gezogen und für das Training des Baums verwendet, wobei dieses Training solange erfolgt, bis alle Beobachtungen entweder isoliert sind oder der Baum die Maximalthöhe $l = \lceil \log_2 \psi \rceil$ erreicht hat. Das Training des ITs wird in Algorithmus 3.1 beschrieben, wohingegen der Aufbau des Ensembles aus ISOLATION TREES zu einem ISOLATION FOREST in Algorithmus 3.2 angegeben ist. Diese beiden Algorithmen sind dazu geeignet, um einen ISOLATION FOREST auf Grundlage eines Datensatzes aufzubauen und diesen für die Prüfung beliebiger Beobachtungen im Rahmen einer Modellapplikation zu nutzen.

Algorithmus 3.1 Training des ISOLATION TREES [40]

Eingabe: Datensatz: \mathcal{D}_{UL} , Merkmale in \mathcal{D}_{UL} : Q , Aktuelle Baumhöhe: $e \in \mathbb{N}$,

Maximale Baumhöhe: $l \in \mathbb{N}$

- 1: **function** IT(\mathcal{D}_{UL}, e, l)
 - 2: **if** $e \geq l$ **or** $|\mathcal{D}_{UL}| \leq 1$ **then**
 - 3: **return** EXTERNNODE{Size $\leftarrow |\mathcal{D}_{UL}|$ }
 - 4: **else**
 - 5: Wähle zufällig ein Splitattribut $q \in Q$.
 - 6: Wähle zufällig einen Splitwert p anhand der Min- und Maxwerte von q in \mathcal{D}_{UL} .
 - 7: $\mathcal{D}_{UL}^L \leftarrow \text{FILTER}(\mathcal{D}_{UL}, q < p)$
 - 8: $\mathcal{D}_{UL}^R \leftarrow \text{FILTER}(\mathcal{D}_{UL}, q \geq p)$
 - 9: LeftNode \leftarrow IT($\mathcal{D}_{UL}^L, e + 1, l$)
 - 10: RightNode \leftarrow IT($\mathcal{D}_{UL}^R, e + 1, l$)
 - 11: **return** INTERNALNODE {LeftNode, RightNode, SplitAtt $\leftarrow q$, SplitVal $\leftarrow p$ }
 - 12: **end if**
 - 13: **end function**
-

Algorithmus 3.2 Training des ISOLATION FORESTS [40]

Eingabe: Datensatz: \mathcal{D}_{UL} , $t \in \mathbb{N}$: Anzahl der Bäume, $\psi \in \mathbb{N}$: Stichprobengröße

```

1: function IF( $\mathcal{D}_{UL}, t, \psi$ )
2:   Forest  $\leftarrow \emptyset$ 
3:    $l = \lceil \log_2 \psi \rceil$ 
4:   for  $i \in [1, t]$  do
5:      $\mathcal{D}'_{UL} \leftarrow \text{SAMPLE}(\mathcal{D}_{UL}, \psi)$ 
6:     Forest  $\leftarrow$  Forest  $\cup$  IT( $\mathcal{D}'_{UL}, 0, l$ )
7:   end for
8:   return Forest
9: end function

```

Anwendung des Isolation Forest

Um den Anomaliescore für beliebige Beobachtungen zu inferieren, wird nun die bereits aus Definition 3.4.3 bekannte durchschnittliche Pfadlänge $E(h(\vec{x}))$ genutzt. Die Pfadlänge für einen einzelnen Baum wird dabei durch Algorithmus 3.3 errechnet, der im Wesentlichen für eine gegebene Beobachtung entsprechend der Splitattribute und -werte den Baum traversiert und die Höhe mitzählt. Eine Besonderheit ergibt sich, wenn der externe Knoten, an dem der Baum terminiert, mehr als eine Beobachtung enthält: In diesem Falle würde die Pfadlänge deutlich kürzer ausfallen, obwohl noch weitere Beobachtungen zu partitionieren sind. Um zu verhindern, dass Bäume vollständig ausgebaut werden müssen (was beispielsweise für ressourcenbeschränkte Systeme nicht gangbar sein kann) und Pfadlängen unterschätzt werden (was zu fälschlicherweise als Ausreißer markierten Beobachtungen führen kann), wird die tatsächliche Pfadlänge mit Gleichung 3.2 näherungsweise korrigiert.

Algorithmus 3.3 Bestimmung der Pfadlänge in einem ISOLATION TREE [40]

Eingabe: Beobachtung: $\vec{x} \in \mathcal{D}_{UL}$, Isolation Tree: T , Aktuelle Pfadlänge: $e \in \mathbb{N}$ (= 0 bei initialem Aufruf)

```

1: function PFADLÄNGE( $\vec{x}, T, e$ )
2:   if  $T$  ist ein externer Knoten then
3:     return  $e + c(T.size)$   $\triangleright T.size$  entspricht der Anzahl der Beobachtungen in  $T$ .
4:   end if
5:    $a \leftarrow T.SplitAtt$ 
6:   if  $\vec{x}^a < T.SplitValue$  then
7:     return PFADLÄNGE( $\vec{x}, T.LeftNode, e + 1$ )
8:   else
9:     return PFADLÄNGE( $\vec{x}, T.RightNode, e + 1$ )
10:  end if
11: end function

```

Die Anomaliebewertung ergibt sich dann durch die so errechneten, durchschnittlichen Pfadlängen für einen IF und der in Definition 3.4.3 angegebenen Formel und Bewertungsregeln.

Laufzeitkomplexität

Den Autoren zufolge liegt die Laufzeitkomplexität für das Trainieren des Ensembles aus ISOLATION TREES zu einem ISOLATION FOREST bei $\mathcal{O}(t\psi \log \psi)$, wohingegen die Zeitkomplexität für die Evaluierung des Ensembles bei einer Testdatengröße von n bei $\mathcal{O}(nt \log \psi)$ liegt [40].

3.4.2 Local Outlier Factor

Der LOCAL OUTLIER FACTOR (LOF) ist eine unüberwachte Methode zur Bestimmung von ausreißenden Beobachtungen [8], die im Jahr 2000 von Breunig *et al.* eingeführt wurde. Analog zum ISOLATION FOREST-Verfahren wird für einen unmarkierten Datensatz \mathcal{D}_{UL} eines beliebigen Datenraums X eine Bewertungsfunktion $r : \mathcal{D}_{UL} \rightarrow \mathbb{R}$ gefunden, die einer Beobachtung eine reelle Ausreißerbewertung zuordnet.

Kern dieses Ansatzes ist die Vorstellung, dass Ausreißer nicht nur *global*, also relativ zum gesamten Datensatz, sondern auch relativ zu einer *lokalen* Punkteumgebung existieren können. Grundlage ist hierbei stets eine Distanzfunktion $d : X \times X \rightarrow \mathbb{R}^+$, die als solche die in Definition 2.3.5 aufgeführten Eigenschaften erfüllt. Zur Kategorie der globalen Ausreißer gehören beispielsweise $DB(pct, d_{\min})$ -Ausreißer, die von Breunig *et al.* wie folgt definiert werden:

3.4.4 Definition ($DB(pct, d_{\min})$ -Ausreißer [8]). Eine Beobachtung $p \in \mathcal{D}_{UL}$ heißt $DB(pct, d_{\min})$ -Ausreißer, gdw. mindestens pct Prozent der Beobachtungen in \mathcal{D}_{UL} weiter als d_{\min} entfernt sind, also wenn die Menge $\{q \in \mathcal{D}_{UL} \mid d(p, q) \leq d_{\min}\}$ kleiner oder genau so groß ist, wie $(100 - pct)\%$ des Datensatzes \mathcal{D}_{UL} .

Die Autoren zeigen dabei anhand eines Beispiels, welches an dieser Stelle reproduziert werden soll, dass diese Definition eines *globalen Ausreißers* zwar unter bestimmten Bedingungen bedeutsam und angemessen sein kann, aber für komplex-strukturierte Datensätze häufig eine andere Art von Ausreißern vorhanden ist. In Abbildung 3.2 ist hierzu eine Beispielsituation in einem zweidimensionalen Raum abgebildet. Die visuelle Inspektion des Scatterplots liefert für die Punkte o_1 und o_2 die nach Definition 3.1.1 erforderliche Abweichung von anderen Beobachtungen, in dem Falle den Clustern C_1 und C_2 , um diese Punkte als Ausreißer zu charakterisieren. Wendet man die Definition 3.4.4 an, dann ist aber lediglich der Punkt o_1 ein sinnvoller Ausreißer, da nur für diesen Punkt eine Parameterkombination aus d_{\min} und pct angegeben werden kann, sodass o_2 ein Ausreißer ist aber die Beobachtungen in C_2 es nicht sind [8].

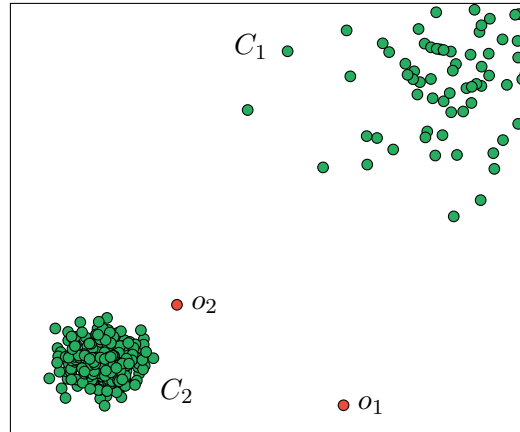


Abbildung 3.2: Ein zweidimensionaler Beispieldatensatz, der aus zwei Clustern C_1 und C_2 nicht-ausreißender Punkte (grün) und zwei Ausreißern o_1 und o_2 besteht (Abbildung angelehnt an [8]).

Ein anderes Verständnis von Ausreißern, die eine korrekte Trennung der in Abbildung 3.2 dargestellten Punkte in ausreißende und nicht-ausreißende Beobachtungen ermöglicht, bieten im Gegensatz zu den globalen Ausreißern die sogenannten *lokalen Ausreißer*, die sich weniger auf den gesamten Datensatz sondern vielmehr auf die Punkteumgebung beziehen. Um den Begriff des lokalen Ausreißers zu formalisieren, wird zunächst der Begriff der k -Distanz und der k -Distanz-Nachbarschaft wie folgt definiert:

3.4.5 Definition (k -Distanz einer Beobachtung [8]). Sei $k \in \mathbb{N}$ und $d : X \times X \rightarrow \mathbb{R}$ eine Distanzfunktion über einem Datenraum X , dann ist die k -Distanz einer Beobachtung $p \in \mathcal{D}_{\text{UL}} \subset X$ (geschrieben: $k\text{-dist}(p)$), definiert durch die Distanz $d(p, o)$ zwischen p und einer Beobachtung $o \in \mathcal{D}_{\text{UL}}$, sodass:

1. Für mindestens k Beobachtungen $o' \in \mathcal{D}_{\text{UL}} \setminus \{p\}$ gilt, dass $d(p, o') \leq d(p, o)$ und
2. Für mindestens $k - 1$ Beobachtungen $o' \in \mathcal{D}_{\text{UL}} \setminus \{p\}$ gilt, dass $d(p, o') < d(p, o)$.

3.4.6 Definition (k -Distanz-Nachbarschaft einer Beobachtung [8]). Sei $d_k = k\text{-dist}(p)$ die k -Distanz einer Beobachtung $p \in \mathcal{D}_{\text{UL}} \subset X$ und $d : X \times X \rightarrow \mathbb{R}$ eine Distanzfunktion, dann ist die k -Distanz-Nachbarschaft von p jeder Punkt aus \mathcal{D}_{UL} der nicht weiter als d_k entfernt ist, also

$$N_{k\text{-dist}(p)}(p) = \{q \in \mathcal{D}_{\text{UL}} \setminus \{p\} \mid d(p, q) \leq d_k\} \quad (3.4)$$

Ferner wird das Konzept einer Erreichbarkeitsdistanz eingeführt, die einen Glättungseffekt auf die tatsächliche, durch eine Distanzfunktion berechnete Distanz zweier Beobachtungen ausübt. Dieser Glättungseffekt wird beabsichtigt, um Beobachtungen, die besonders *nahe*

(im Sinne der Distanzfunktion) an einer zu untersuchenden Beobachtung liegen, die gleiche Distanz zuzuordnen und so statistische Fluktuationen zu reduzieren [8].

3.4.7 Definition (Erreichbarkeitsdistanz [8]). Sei $k \in \mathbb{N}$ und $d : X \times X \rightarrow \mathbb{R}$ eine Distanzfunktion, dann ist die Erreichbarkeitsdistanz einer Beobachtung $p \in \mathcal{D}_{UL}$ hinsichtlich einer Beobachtung $o \in \mathcal{D}_{UL}$ definiert durch

$$\text{reach-dist}_k(p, o) = \max \{k\text{-dist}(o), d(p, o)\} \quad (3.5)$$

Die Stärke dieses Glättungseffekts wird dabei durch den Parameter k kontrolliert. Wird dieser Parameter vergrößert, dann vergrößert sich auch die k -Distanz womit die Glättung auf *mehr* Beobachtungen Anwendung findet. Umgekehrt vermindert sich die Glättung, wenn der Parameter niedriger gewählt wird.

Wie bereits erwähnt, liegt der Kern in der Entdeckung von lokalen Ausreißern darin, die lokale Punkteumgebung einer spezifischen Beobachtung näher zu untersuchen. Insbesondere bei Betrachtung von Definition 3.4.5 wird deutlich, dass dabei die intuitive Vorstellung von dichteren und weniger dichten Punktereignissen von großer Bedeutung ist. So würde beispielsweise für zwei repräsentative Punkte der Cluster C_1 und C_2 der in Abbildung 3.2 dargestellten Beispielsituation für eine feste Wahl des k -Parameters stark variierende k -Distanzen resultieren, da für beide Cluster unterschiedlich viele Punkte in einen gleich dimensionierten Unterraum fallen. So müssten für eine repräsentative Beobachtung des (visuell auch erkennbaren) weniger dichten Clusters C_1 bei festem k für die Bestimmung der k -Distanz weiter entfernt liegende Beobachtungen einbezogen werden, als es beim Cluster C_2 der Fall wäre. Daher schlagen Breunig *et al.* eine Untersuchung der Dichte der Punkteumgebung vor, wobei hier ein zentraler Mechanismus des sogenannten *dichtebasierten Clusterings* zum Tragen kommt, nämlich die Parametrisierung einer minimalen Anzahl von Punkten $\text{minPts} \in \mathbb{N}$.

Mit diesem Parameter lässt sich für eine feste Beobachtung $p \in \mathcal{D}_{UL}$ und einer anderen Beobachtung $o \in N_{k\text{-dist}(\text{minPts})(p)}$ zusammen mit der in Definition 3.4.7 gegebenen Erreichbarkeitsdistanz $\text{reach-dist}_{\text{minPts}}(p, o)$ ein Maß für die Dichte der benachbarten Punkte angeben, welches als *lokale Erreichbarkeitsdichte* wie folgt definiert wird:

3.4.8 Definition (Lokale Erreichbarkeitsdichte [8]). Sei $p \in \mathcal{D}_{UL}$ und $\text{minPts} \in \mathbb{N}$, dann ist die lokale Erreichbarkeitsdichte dieses Punkts $\text{lrd}_{\text{minPts}}(p)$ durch folgenden Ausdruck definiert:

$$\text{lrd}_{\text{minPts}}(p) = \left(\frac{\sum_{o \in N_{k\text{-dist}(\text{minPts})(p)}} \text{reach-dist}_{\text{minPts}}(p, o)}{|N_{k\text{-dist}(\text{minPts})(p)}|} \right)^{-1} \quad (3.6)$$

Die lokale Erreichbarkeitsdichte gibt dabei das Inverse der durchschnittlichen Erreichbarkeitsdistanz auf Grundlage der $minPts$ -nächsten Nachbarn an. Auf Grundlage dieser Definition lässt sich nun der lokale Ausreißerfaktor (engl. *local outlier factor*, LOF) bestimmen, der für jeden spezifischen Punkt ein reelles Maß dafür angibt, wie stark dieser ausreißt:

3.4.9 Definition (Local Outlier Factor (LOF) [8]). Sei $p \in \mathcal{D}_{UL}$ und $minPts \in \mathbb{N}$, dann ist der lokale Ausreißerfaktor oder *Local Outlier Factor* (LOF) für p definiert durch:

$$LOF_{minPts}(p) = \frac{\sum_{o \in N_{k-dist(minPts)}(p)} \frac{lrd_{minPts}(o)}{lrd_{minPts}(p)}}{|N_{k-dist(minPts)}(p)|} \quad (3.7)$$

Der LOF gibt das durchschnittliche Verhältnis der lokalen Erreichbarkeitsdichten eines zu untersuchenden Punktes p und den $minPts$ -nächsten Nachbarn an. Diese Formel beinhaltet dabei die Intuition, dass eine geringe Erreichbarkeitsdichte von p aber hohe Erreichbarkeitsdichten der benachbarten Punkte gerade stark für einen Ausreißer sprechen und der LOF-Wert dann größer eins ist. Dies ist gerade die Situation, die für den Punkt o_2 in der Abbildung 3.2 erwartet wird. Andernfalls liegt der LOF für einen nicht-ausreißenden Punkt, der innerhalb eines Clusters liegt, näherungsweise bei eins [8] oder sogar darunter, falls der Punkt in einer besonders dichten Region liegt.

Laufzeitkomplexität

Die Laufzeitkomplexität des LOCAL OUTLIER FACTORS wird ausgehend von Formel 3.7 bestimmt: Es ist leicht einsehbar, dass die Gesamtkomplexität von der Berechnung der k -Distanz beziehungsweise der k -Distanz-Nachbarschaften dominiert wird.

Bei Nutzung eines sogenannten *Brute Force*-Ansatzes, bei dem die k -nächsten Beobachtungen gefunden werden sollen, muss ein zu untersuchender Punkt p mit jedem anderen Punkt verglichen werden, was bei einer Datensatzgröße n zu einer Laufzeit von $\mathcal{O}(n)$ führt. Möchte man den LOF für alle Beobachtungen im Datensatz bestimmen, so steigt die Laufzeitkomplexität daher zu $\mathcal{O}(n^2)$.

Darüber hinaus diskutieren Breunig *et al.* die Möglichkeit hierfür Indexstrukturen einzusetzen, die abhängig von der Dimensionalität der Daten die Frage nach den k -nächsten Nachbarn in konstanter, logarithmischer oder linearer Zeit beantworten können [8].

3.4.3 Diskussion

In den beiden vorangegangenen Abschnitten wurden zwei Methoden zur Ausreißererken- nung vorgestellt, nämlich der ISOLATION FOREST und der LOCAL OUTLIER FACTOR.

Der ISOLATION FOREST arbeitet, wie für Ensemble-Methoden typisch, mit mehreren Modellen (den ISOLATION TREES), die gemeinsam ausgewertet werden. Jeder dieser Bäume wird dabei anhand einer Teilmenge der verfügbaren Merkmale trainiert, die, genauso wie das benötigte Splitattribut, zufällig bestimmt wird. Die Induktion dieser Bäume erfolgt dabei entfernt nach den Prinzipien einer divisiven hierarchischen Clusteranalyse, wobei auf die Verwendung eines Distanzmaßes verzichtet wird und lediglich eine Ordnung auf den jeweiligen Splitattributen verlangt wird. Dies macht das Verfahren etwas flexibler und einfacher, setzt aber auch voraus, dass in den *meisten* Fällen — das Ensemble kann Fehler bis zu einem gewissen Grad „glätten“ — Ausreißer oder Ausreißerpartitionen korrekt isoliert werden. In Anlehnung an *Hawkins'* intuitive Definition eines Ausreißers, werden diese in einem einzelnen Knoten eines ISOLATION TREES idealerweise dadurch identifiziert, dass diese sich anhand eines Splitwerts von einer „Normalpartition“ unterscheiden und so die Abweichung ausgedrückt wird.

Die Einordnung in die taxonomische Strukturierung der Ausreißererkenntnisverfahren nach Abschnitt 3.3 stellt sich jedoch schwierig dar, da der ISOLATION FOREST sich ausschließlich auf Zufallsentscheidungen beruft: Es wird weder eine Unterraumprojektion gesucht noch ein zu der One-Class-SVM ähnliches Optimierungsproblem gelöst, welches einzelne Beobachtungen anhand einer (weichen) Trennebene von anderen trennt. Die Einordnung dieser Methode in die Gruppe der Clustering-basierten Verfahren ist ebenfalls auszuschließen, da Clustering inhärent mit der Verwendung von Distanzmaßen verknüpft ist und diese hier explizit nicht verwendet werden. Die gleiche Argumentation gilt auch für die Nächste-Nachbarn-basierten Methoden. Denkbar ist eine Einordnung in den Bereich der statistischen Methoden: Die Autoren Liu *et al.* stützen sich bei der Etablierung im Wesentlichen darauf ab, dass Ausreißer in geringer Zahl und anders in Erscheinung treten („few and different“ [40]) und somit diese Beobachtungen in unwahrscheinlicheren Bereichen des Datenraums liegen. Die Konstruktion des ISOLATION FORESTS könnte daher inhärent die Annahme beinhalten, dass die Wahrscheinlichkeit einen ausreichend reinen Split (also eine möglichst geringe Vermischung von ausreißenden und normalen Beobachtungen zwischen den neuen Partitionen) zu erhalten bei der Baumkonstruktion hoch genug ist, um eine reine Zufallswahl des Splitkriteriums zu rechtfertigen, wobei gleichzeitig Fehler über das Ensemble geglättet werden. Zusammen mit der Vorstellung von seltener auftretenden Ausreißern könnte dies auf kürzere Pfadlängen im Sinne der Definition 3.4.2 führen. Soweit an dieser Stelle bekannt ist, fehlen hierzu allerdings theoretische Erklärungen, die diese Hypothese stützen, weswegen eine definitive konkrete Einordnung in die taxonomische Strukturierung nach Abschnitt 3.3 nicht erfolgen soll.

Der LOCAL OUTLIER FACTOR hingegen ist gemäß der taxonomischen Strukturierung nach Abschnitt 3.3 eine kanonische *Nächste-Nachbarn*-Methode (vgl. Abschnitt 3.3) [20], bei der die Zuordnung der Ausreißereigenschaft für eine individuelle Beobachtung durch Inspektion der Punkteumgebung erfolgt. Die nach *Hawkins'* geforderte Abweichung, die

den Verdacht eines Ausreißers erweckt, wird also dadurch erkannt, dass eine Beobachtung in einer definierten Umgebung weiter entfernt liegt, als andere Beobachtungen. Die Größe dieser Umgebung und auch die Wahl einer adäquaten Distanzfunktion stellen zu optimierende Hyperparameter dar, die sich gerade in hochdimensionalen Räumen mitunter schwierig gestalten kann.

3.5 Submodulare Methoden

Das Verständnis von submodularen Funktionen war in den vergangenen Abschnitten durch eine relativ abstrakte Perspektive auf die gewonnenen Mengen geprägt. So lag der Fokus auf der Identifizierung von *optimalen Teilmengen*, die eine definierte Funktion hinsichtlich einer Grundmenge unter Kardinalitätsbeschränkungen optimiert. Eine etwas konkretere Perspektive auf die optimierten Mengen lässt sich dadurch gewinnen, dass, wie bereits angedeutet, diese Mengen als Zusammenfassung eines Datensatzes betrachtet werden. Von solch einer Zusammenfassung erwartet man intuitiv, dass diese repräsentative Beobachtung des anfangs zu Verfügung gestellten Datensatzes bereit hält und damit sowohl übliche als auch unübliche Repräsentanten beinhaltet. Diese Vorstellung suggeriert auf natürliche Weise den Einsatz der Zusammenfassung für eine Ausreißerererkennung. Der zu verwendende *Zusammenfassungsbegriff* wird dabei durch die submodulare Funktion festgelegt: So versucht die INFORMATIVE VECTOR MACHINE durch die Verwendung einer Kernfunktion und der dazugehörigen Kernmatrix die Entropie, also die zu erwartende Unsicherheit, zu minimieren und so eine adäquate Repräsentation zu finden. Das EXEMPLAR-BASED CLUSTERING versucht Gruppen unähnlicher Beobachtungen voneinander zu trennen, wobei jede Gruppe durch eine Beobachtung repräsentiert wird.

In diesem Abschnitt sollen nun Methoden präsentiert werden, um die Lücke zwischen einer gegebenen Zusammenfassung und der Ausreißerererkennung mittels submodularen Funktionen zu schließen. Kern der beiden Ansätze ist eine Partitionierung des Datenraums in *Regionen* anhand der Repräsentanten, wobei jeweils eine sogenannte einfache und rekursive Partitionierungsstrategie vorgestellt wird.

3.5.1 Einfache Partitionierung

Buschjäger *et al.* stellen eine Ausreißerererkennung namens SUMMARY OUTLIER DETECTION (SOD) vor, die eine einfache Partitionierungsstrategie implementiert [9]. Für eine gegebene Zusammenfassung S einer Grundmenge $V \subset X$ eines Datenraums X , die anhand der Lösung des Optimierungsproblems 2.3 gefunden wurde, soll einem individuellen Punkt die Wahrscheinlichkeit, dass dieser ein Ausreißer ist, zugeordnet werden. Formal soll also eine Bewertungsfunktion $r : X \rightarrow [0, 1]$ gefunden werden.

Um eine individuelle Beobachtung zu bewerten, wird nun eine Partitionierung des Datenraums betrachtet, die zusammen mit einer Distanzfunktion $d : X \times X \rightarrow \mathbb{R}^+$ durch die

Vektoren in S induziert wird. Ein Vektor $\vec{x} \in X$ gehört dabei zu einer Partition, die durch den Vektor $\vec{s}_p \in S$ aufgespannt wird, genau dann wenn $\forall \vec{s} \in S \setminus \{\vec{s}_p\} : d(\vec{v}, \vec{s}) \geq d(\vec{v}, \vec{s}_p)$, informell also genau zum nächsten Repräsentanten gegeben der Distanzfunktion. Die genutzte Distanzfunktion ist dabei ein Parameter, der bei Modellapplikation gewählt wird. Dabei wird idealerweise die Wahl derart getroffen wird, dass die bei der Optimierung der Funktion genutzte Ähnlichkeits- beziehungsweise Distanzsemantik ebenfalls zur Partitionierung herangezogen wird. So arbeitet der RBF-Kernel üblicherweise mit der quadratischen, euklidischen Distanz zwischen zwei Vektoren, die dann auch bei der Anwendung des Modells verwendet werden könnte.

Für eine Menge von Beobachtungen wird nun gezählt, wie viele der Vektoren in die entsprechenden Partitionen fallen, womit eine Wahrscheinlichkeit für das Auftreten von Beobachtungen in bestimmten Regionen geschätzt wird. Formal wird dies durch das Einführen eines Zählers c_i für jede Partition \vec{s}_i realisiert, der inkrementiert wird sobald ein Vektor der Menge in die entsprechende Partition fällt, sodass diese Wahrscheinlichkeit wie folgt angegeben werden kann:

$$\mathbb{P}(i = \arg \min \{d(\vec{v}, \vec{s}_i) \mid \vec{s}_i \in S\}) \approx \frac{c_i}{\sum_{\vec{s}_i \in S} c_i} \quad (3.8)$$

Um diese Zähler adäquat zu bestimmen, wird im Stapelbetrieb (bei dem wahlfreier Zugriff auf sämtliche Daten möglich ist) ein zweites mal die Grundmenge herangezogen und gezählt, wie häufig Elemente der Grundmenge in die entsprechenden Partitionen fallen. Bei Betrachtung eines Datenstroms ist ein wahlfreier Zugriff auf die Daten inhärent nicht möglich: Buschjäger *et al.* schlagen daher vor, dass das Zählen begonnen wird, sobald eine Menge von Repräsentanten aus dem Datenstrom abgeleitet wurde. Jene Elemente, die dann weiterhin in jenem Datenstrom ankommen, werden dann zur Bestimmung der Zähler herangezogen.

Die eigentliche Ausreißerererkennung wird nun durch Betrachtung eines statistischen Tests bewerkstelligt: Gegeben sei nun die zu einer vorhandenen Zusammenfassung S ermittelte Randverteilung $\mathbb{P}(\vec{s}_i \mid S)$ für jeden Repräsentanten $\vec{s}_i \in S$. Nun kann eine Nullhypothese $\mathbb{P}_0(\vec{s}_i \mid S)$ aufgestellt werden, die zunächst einmal eine Annahme über die Verteilung der Daten ausdrückt und als eine Vorstellung von Normalität aufgefasst werden kann: Diese Verteilung kann entweder durch einen Experten geschätzt werden, der die ermittelte Zusammenfassung der Grundmenge seinem Domänenwissen entsprechend untersucht oder aber vereinfachend als Gleichverteilung also $\mathbb{P}_0(\vec{s}_i \mid S) = \frac{1}{|S|}$ angenommen werden. Für die Ausreißerererkennung gilt es nun entweder diese Nullhypothese für jede Partition entsprechend zu widerlegen oder zu bestätigen: Eine Bestätigung der Hypothese würde bedeuten, dass ein Verhalten beobachtet wird, dass mit dem ursprünglich formulierten Erwartungen konform ist und daher für eine „Normalpartition“ spricht. Eine Ablehnung spricht hingegen für eine „Ausreißerpartition“.

Algorithmus 3.4 SUMMARY OUTLIER DETECTION [9].

```

1: function KENNZEICHNEREGIONEN( $S$ )
2:    $c_i \leftarrow 0, \forall i \in \{1, \dots, k\}$  ▷ Initialisiere die Zähler.
3:   for  $\vec{v} \in V$  do
4:      $i \leftarrow \arg \min \{d(\vec{v}, \vec{s}_i) \mid \vec{s}_i \in S\}$ 
5:      $c_i \leftarrow c_i + 1$ 
6:   end for
7:   for  $i \in \{1, \dots, |S|\}$  do
8:     if  $\frac{c_i}{|V|} \leq \mathbb{P}_0$  then
9:        $p_i \leftarrow \text{BINOMIALTEST}(\frac{c_i}{|V|}, \mathbb{P}_0)$  ▷ Berechne den  $p$ -Wert für die Partition.
10:      if  $p_i < p_{\text{thr}}$  then
11:         $p_i \leftarrow 0$  ▷ Kennzeichne die Partition als Ausreißerregion.
12:      else
13:         $p_i \leftarrow 1$  ▷ Kennzeichne die Partition als Normalregion.
14:      end if
15:    else
16:       $p_i \leftarrow 1$  ▷ Kennzeichne die Partition als Normalregion.
17:    end if
18:  end for
19: end function
20: KENNZEICHNEREGIONEN(EXTRAHIEREZUSAMMENFASSUNG( $\mathcal{D}_{\text{UL}}$ ))

```

Die Prüfung der aufgestellten Nullhypothese erfolgt dabei mittels eines zweiseitigen Binomialtests. Hierbei werden die Hypothesenpaare $H_0 : \mathbb{P} = \mathbb{P}_0$ und $H_1 : \mathbb{P} \neq \mathbb{P}_0$ gegeneinander getestet und ein sogenannter p -Wert ermittelt. Dieser liefert das kleinste Signifikanzniveau α , welches gerade noch zur Ablehnung der Nullhypothese H_0 führt [30]. Das Signifikanzniveau ist dabei eine Wahrscheinlichkeit für das Begehen eines Fehlers erster Art, also das Ablehnen der Nullhypothese obwohl diese eigentlich zutreffend wäre [30]. Ein hoher p -Wert lässt sich also derart interpretieren, dass die Nullhypothese tendenziell nicht abgelehnt werden kann und somit die getestete Partition als normal bewertet werden sollte. Ein niedriger p -Wert spricht hingegen für die Ablehnung von H_0 und somit für eine Ausreißerpartition. Wann ein p -Wert „niedrig genug“ empfunden wird, um eine Partition entsprechend als Ausreißer zu markieren, entscheidet ein Schwellwert p_{thr} der vorgegeben wird und entsprechend unterschritten werden muss. Berechnet wird dieser p -Wert für einen zweiseitigen Test, durch Betrachtung des folgenden Ausdrucks:

$$p = 2 \cdot \min \{ \mathbb{P}(T \leq t \mid H_0), \mathbb{P}(T \geq t \mid H_0) \} \quad (3.9)$$

Die Zufallsvariable T beschreibt hier die Anzahl der Beobachtungen, die in eine zu prüfende Partition gefallen sind, von der wir unter der Nullhypothese H_0 ausgehen, dass diese

binomialverteilt ist. Damit lassen sich die in Gleichung 3.9 angegebenen Wahrscheinlichkeitsausdrücke mit der Wahrscheinlichkeitsfunktion der Binomialverteilung $B(x | p, n) = \binom{n}{x} p^x (1-p)^{n-x}$ und der Definition der Verteilungsfunktion sowie $p = \mathbb{P}_0$ und $n = |V|$ wie folgt berechnen:

$$\mathbb{P}(T \leq t | H_0) = \sum_{i=0}^t B(i | p, n) \quad (3.10)$$

$$\mathbb{P}(T \geq t | H_0) = \sum_{i=t}^n B(i | p, n) - \sum_{i=0}^{t-1} B(i | p, n) \quad (3.11)$$

Eine durch Buschjäger *et al.* beschriebene zusätzliche Verfeinerung der oben genannten Prozedur ergibt sich dadurch, dass nur besonders dünn besetzte Partitionen (ausgedrückt durch die Randverteilung) für den genannten statistischen Test erwogen werden. Der Grund hierfür liegt in der Charakteristik des verwendeten Tests: Ist $\mathbb{P}(\vec{s}_i | S)$ deutlich größer oder deutlich kleiner als $\mathbb{P}_0(\vec{s}_i | S)$, kann es vorkommen, dass besonders dichte Partitionen fälschlicherweise als Ausreißerregionen markiert werden. Dies ist aber nicht konform mit der intuitiven Vorstellung einer ausreißenden Region (vgl. Definition 3.1.1), von der wir eine starke Abweichung von anderen Beobachtungen erwarten.

Zusammen mit der hier angegebenen Beschreibung lässt sich eine vollständige Prozedur formulieren, die in Algorithmus 3.4 angegeben ist.

Laufzeitkomplexität

Geht man an dieser Stelle davon aus, dass eine Zusammenfassung S bereits vorliegt und die Komplexität der Bestimmung dieser nicht berücksichtigt werden muss, so ergeben sich ausgehend von Algorithmus 3.4 zwei zentrale Laufzeitfaktoren: Die Bestimmung der Zählerwerte (Zeilen 3 bis 5) erfordern den Vergleich jedes Elements der Grundmenge V mit jedem Repräsentanten $\vec{s} \in S$, was zu einer Laufzeitkomplexität von $\mathcal{O}(|V| \cdot |S|)$ führt. Die Berechnung des Binomialtests beziehungsweise der beidseitigen p -Werte wird durch die Bestimmung der entsprechenden Binomialkoeffizienten $\binom{n}{x}$ dominiert: Bei direkter Nutzung der Formel $\binom{n}{x} = \frac{n!}{x!(n-x)!}$ liegt mit $\Theta(2\binom{n}{x} - 1)$ [43] eine super-polynomielle Laufzeit vor, die schon für relativ geringe n und gerade bei $n = |V|$ nicht mehr praktikabel ist. Die Nutzung dynamischer Programmierung reduziert die Laufzeitkomplexität auf $\Theta(nx)$ [43]. Die Komplexität der Bewertung aller Regionen, die durch S repräsentiert werden, führt somit unter Betrachtung der Gleichung 3.9 zu einer Zeitkomplexität von $\mathcal{O}(|S| \cdot |V|^2)$, was auch die Gesamtkomplexität des Algorithmus 3.4 beschreibt. An dieser Stelle soll jedoch nicht unerwähnt bleiben, dass die in den Gleichungen 3.10 und 3.11 durchgeführte Berechnung der Verteilungsfunktion für die Binomialverteilung auch anhand der sogenannten *regularisierten unvollständigen Betafunktion* erfolgen kann, die auf die Berechnung des Binomialkoeffizienten verzichtet, aber stattdessen die Berechnung beziehungsweise die Ap-

proximation eines Integrals verlangt [63], weswegen dies hier nicht näher besprochen werden soll.

3.5.2 Rekursive Partitionierung

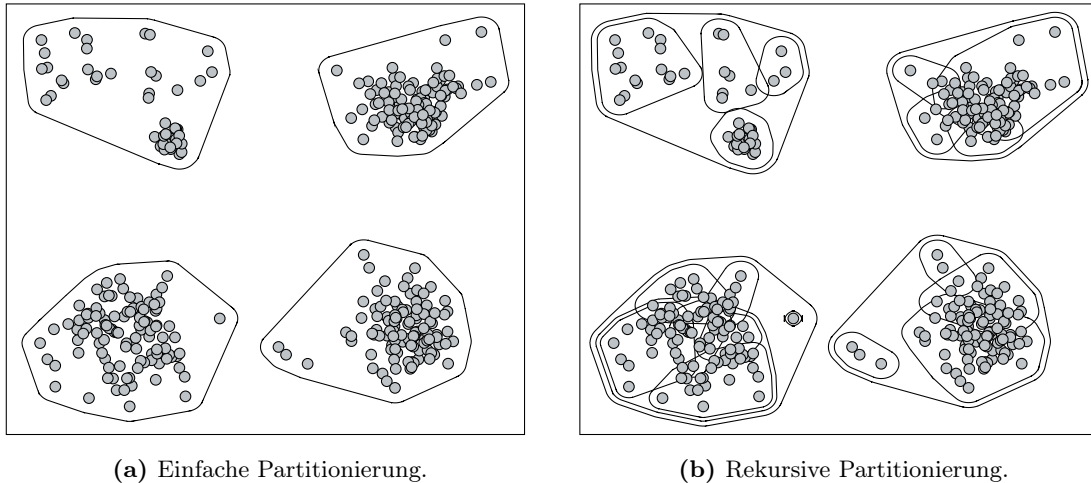


Abbildung 3.3: Darstellung der einfachen und rekursiven Partitionierungsstrategie bei Nutzung einer jeweils identisch konfigurierten submodularen Funktion (EXEMPLAR-BASED CLUSTERING, vgl. Abschnitt 2.3.2), einer Zusammenfassungsgröße von $k = 4$ und des GREEDY-Optimierers. **a)** Es wird eine Zusammenfassung gefunden, die jeweils einen Repräsentanten für jeden der vier visuell vorhandenen Cluster vorhält. Entsprechend werden vier Partitionen, erkennbar durch die eingezeichneten Grenzen, identifiziert. **b)** Aufbauend auf der einfachen Partitionierung wird in den jeweiligen Regionen eine erneute Partitionierung durchgeführt, die zur Identifizierung weiterer Subgruppen führt. Diese könnten beispielsweise im industriellen Umfeld als eine (Sub-)Gruppe von Maschinenstörungen interpretiert werden.

Das in Abschnitt 3.5.1 beschriebene Verfahren lässt sich als *einfach partitionierend* charakterisieren. Es wird eine globale Perspektive auf den Datensatz eingeführt, bei der eine gefundene Zusammenfassung eines Datensatzes dazu genutzt wird, um den Datenraum in entsprechende Unterräume zu partitionieren und diese mittels eines statischen Tests als Ausreißer- und Normalregionen zu kennzeichnen. Hierfür ist es notwendig, dass die zur Partitionierung benötigte Zusammenfassung bereits adäquate Regionen findet. Dies kann sich in Abhängigkeit der gewählten Funktion und des Optimierungsverfahrens als schwierig herausstellen, da die erforderlichen Parameter sorgfältig gewählt werden müssen.

Eine Verbesserung der SUMMARY OUTLIER DETECTION ließe sich möglicherweise dadurch erzielen, dass der Datenraum nicht *einfach partitioniert* sondern mittels eines rekursiven Ansatzes *mehrfach partitioniert* wird. In einer bereits identifizierten Partition wird eine erneute Partitionierung durchgeführt, um diese zu inspizieren und dort weitere für die Ausreißerererkennung interessante Strukturen zu entdecken. Neben dieser feineren Abtastung der im Datensatz vorhandenen Regionen wird so auch das Problem der Pa-

parameterwahl abgeschwächt: Anstatt zu verlangen, dass bereits die erste Partitionierung alle interessanten Regionen identifiziert, wäre es mit dem rekursiven Ansatz möglich, eine bereits gefundene Region näher zu inspizieren und dort weitere für die Ausreißererken- nung bedeutsame Strukturen zu entdecken. So können auch anfangs suboptimal gewählte Parameter zu erfolgreichen Resultaten führen.

In diesem Abschnitt soll dieser rekursive Ansatz mit dem Namen RECURSIVE SUB- MODULAR PARTITIONING (RSP) vorgestellt werden. Diese Methode baut dabei auf der im Abschnitt 3.5.1 vorgestellten Methode auf, führt aber die beschriebene wiederholte Parti- tionierung ein. Hierfür wird ausschließlich die Situation des Stapelbetriebs betrachtet, in der ein wahlfreier Zugriff auf die Daten möglich ist. Gegeben einer Grundmenge $\mathcal{D}_{UL} \subset X$, die auf Ausreißer untersucht werden soll, und einem beliebigen Datenraum X wird der RSP-Baum eingeführt, der die Partitionierung von \mathcal{D}_{UL} adäquat beschreiben kann:

3.5.1 Definition (RSP-Baum). Sei T der Knoten eines RSP-Baums, der eine Men- ge von Vektoren $V(T) \subseteq \mathcal{D}_{UL}$ und einen Repräsentanten $R(T) \in V(T)$ beinhaltet. Seien weiterhin $C_T = \{T_1, \dots, T_n\}$ die zu T gehörigen Kindknoten. Falls $C_T = \emptyset$, dann ist T ein sogenannter Blattknoten. Falls $C_T \neq \emptyset$, dann gilt für zwei Kindkno- ten $T_i, T_j \in C_T$ mit $T_i \neq T_j$, dass $V(T_i) \cap V(T_j) = \emptyset$. Weiterhin gilt dann, dass $\bigcup_{T_i} V(T_i) = V(T)$. Damit stellen die Kindknoten gemeinsame eine Partitionierung von $V(T)$ dar.

Induktion des Baums

Die Induktion des Baums erfolgt nun mittels der in Abschnitt 3.5.1 formulierten Idee der Partitionierung anhand von Zusammenfassungen. Der Wurzelknoten des RSP-Baums R beinhaltet zunächst sämtliche Beobachtungen des Eingabedatensatzes $V(R) = \mathcal{D}_{UL}$, wobei der initiale Repräsentant durch den Zentroiden \vec{c} der Eingabedaten gegeben ist:

$$\vec{c} = \frac{1}{|V(R)|} \sum_{\vec{v} \in V(R)} \vec{v} \quad (3.12)$$

Nun wird mittels eines der in Abschnitt 2.2 vorgestellten Optimierungsverfahren das Pro- blem 2.3 gelöst und so eine Zusammenfassung $S \subseteq \mathcal{D}_{UL}$ mit $|S| = k$ identifiziert. Jeder der Vektoren in S stellt einen Repräsentanten dar, der mittels einer Distanzfunktion eine Partition des Eingaberaums induziert, sodass für den initialen Wurzelknoten k Kindknoten erzeugt werden. Für jeden Kindknoten wird diese Prozedur wiederholt, bis eine definierte Abbruchbedingung erfüllt ist. Für den Fall, dass die Partition eines Knotens T weniger oder genauso viele Beobachtungen enthält als zur Bildung der Zusammenfassung vorgesehen sind, also $V(T) \leq k$, wird der Knoten ebenfalls nicht weiter bearbeitet oder partitioniert.

Es sei erwähnt, dass die in Algorithmus 3.5 gewählte Partitionierungsstrategie (s. Zeile 10) theoretisch dazu führen kann, dass Beobachtungen mehr als einem Knoten zugewiesen

Algorithmus 3.5 Induktion des RSP-BAUMS.

Eingabe: Datensatz \mathcal{D}_{UL} , Distanzfunktion d , Repräsentanten pro Partition $k \in \mathbb{N}$

```

1: function FINDERSPBAUM( $\mathcal{D}_{UL}, d$ )
2:    $Q \leftarrow \{\}$  ▷ Warteschlange zu bearbeitender Knoten.
3:    $\vec{c}_R \leftarrow \frac{1}{|\mathcal{D}_{UL}|} \sum_{\vec{v} \in \mathcal{D}_{UL}} \vec{v}$  ▷ Berechne den Zentroiden für den Wurzelknoten.
4:    $Q.\text{Insert}(\text{RSPNODE}(\mathcal{D}_{UL}, \vec{c}_R))$  ▷ Erzeuge Wurzelknoten  $R$  mit  $V(R) = \mathcal{D}_{UL}$ 
5:   while  $Q \neq \emptyset$  do
6:      $T \leftarrow Q.\text{Pop}()$  ▷ Entferne Knoten aus Warteschlange zur Partitionierung.
7:     if  $\text{ABBRUCHBEDINGUNG}(T)$  nicht erfüllt then
8:        $S \leftarrow \text{EXTRAHIEREZUSAMMENFASSUNG}(V(T))$ 
9:       for all  $\vec{s} \in S$  do
10:         $P \leftarrow \{\vec{v} \in V(T) \mid \forall \vec{s}_i \in S \setminus \{\vec{s}\} : d(\vec{v}, \vec{s}) \leq d(\vec{v}, \vec{s}_i)\}$ 
11:         $C \leftarrow \text{RSPNODE}(P, \vec{s})$  ▷ Erzeuge neuen Knoten mit der Partition  $P$ .
12:         $C_T \leftarrow C_T \cup \{C\}$  ▷  $C$  wird Kindknoten von  $T$ .
13:        if  $|P| > k$  then
14:           $Q.\text{Insert}(C)$  ▷ Füge  $C$  der Warteschlange zur Bearbeitung hinzu.
15:        end if
16:      end for
17:    end if
18:  end while
19:  return  $R$ 
20: end function

```

werden können. Diese Situation wird programmatisch ausgeschlossen und zur Vereinfachung hier nicht aufgeführt, sodass eine korrekte Partitionierung nach Definition 3.5.1 garantiert wird.

Für die Induktion des RSP-Baums ist es erforderlich, dass eine Abbruchbedingung spezifiziert wird. Diese Abbruchbedingung operiert dabei auf Grundlage einer Menge von Vektoren, für die die Abbruchentscheidung getroffen wird. In Anlehnung an Definition 3.1.1 sind dabei verschiedene Varianten denkbar:

3.5.2 Definition (RSP-Bauminduktion — Abbruchbedingungen). Sei T der Knoten eines RSP-Baums und der T_R der dazugehörige Wurzelknoten, wobei $V(T)$ die Beobachtungen enthält und $\vec{c}_T = R(T)$ der zu T gehörige Repräsentant ist. Für die Untersuchung von T auf einen Abbruch der Rekursion seien nun folgende Abbruchbedingungen definiert:

1. **Streuung der Partition:** Sei $t_d \in \mathbb{R}^+$ eine akzeptable Streuung innerhalb einer Partition und $d : X \times X \rightarrow \mathbb{R}^+$ eine Distanzfunktion, dann ist diese Abbruchbedingung *erfüllt*, wenn folgender Ausdruck erfüllt ist:

$$\frac{1}{|V(T)|} \sum_{\vec{v} \in V(T)} d(\vec{v}, \vec{c}_T) \leq t_d \quad (3.13)$$

2. **Mittlerer Kernfunktionswert:** Sei $t_k \in \mathbb{R}^+$ der mittlere Kernfunktionswert der für eine Partition mindestens erreicht werden muss und k eine Kernfunktion. Diese Abbruchbedingung ist dann *erfüllt*, wenn folgender Ausdruck erfüllt ist:

$$\frac{1}{|V(T)|} \sum_{\vec{v} \in V(T)} k(\vec{v}, \vec{c}_T) \geq t_k \quad (3.14)$$

3. **Baumhöhe:** Sei t_h die maximale Baumhöhe, die für einen RSP-Baum erreicht werden darf. Die Abbruchbedingung ist dann *erfüllt*, wenn folgender Ausdruck erfüllt ist:

$$h(T_R) > t_h \quad (3.15)$$

Die Funktion $h(T_R)$ berechnet dabei, die Höhe des Baums ausgehend von einem (Wurzel-)Knoten T_R und wird induktiv wie folgt definiert:

$$h(T) = \begin{cases} 0, & \text{falls } T \text{ ein Blattknoten} \\ \max \{1 + h(T_i) \mid \forall T_i \in C_T\}, & \text{sonst} \end{cases} \quad (3.16)$$

Die ersten beiden in Definition 3.5.2 aufgeführten Abbruchbedingungen lassen sich dabei ähnlich motivieren. Die erste Abbruchbedingung (*Streuung der Partition*) formalisiert vor allem die Idee der Identifikation eines intuitiv möglichst kompakten Clusters. Zusammen mit Definition 3.1.1 sollen so Partitionen mit Beobachtungen, die stark von den anderen Beobachtungen in der selben Partition abweichen, weiter aufgeteilt werden. Da hier Beobachtungen unter Verwendung eines Distanzbegriffs miteinander verglichen werden, könnte es sinnvoll sein, diese Abbruchbedingung zusammen mit der submodularen Funktion des EXEMPLAR-BASED CLUSTERINGS zu verwenden, die auf der gleichen Grundlage operiert.

Die zweite Abbruchbedingung (*Mittlerer Kernfunktionswert*) hingegen beruht auf der von Kernfunktionen oder auch im weiteren Sinne von Skalarprodukten eingeführten Idee des Vergleichs von Beobachtungen anhand eines Ähnlichkeitsbegriffs. Die Konzepte von Distanzen und Ähnlichkeiten sind dabei miteinander verwandt, da diese im Allgemeinen das Inverse des jeweils anderen Konzepts darstellen. Dies erklärt auch die starke Ähnlichkeit zwischen der beiden Abbruchbedingungen in Definition 3.5.2. Statt nun aber mittlere Distanzen zu verwenden, die für eine möglichst hohe intuitive Kompaktheit der Partition einen definierten Wert *unterschreiten* müssen, werden hier mittlere Kernfunktionswerte

verwendet, die einen definierten Wert *überschreiten* müssen. Der Grund hierfür liegt in der Vorstellung, dass Partitionen aus möglichst ähnlichen Beobachtungen bestehen und einzelne Beobachtungen, die unähnlicher sind, weiter im Baum aufgetrennt werden sollen. Ähnlich wie auch die erste Abbruchbedingung wird dies mit der intuitiven Vorstellung von Ausreißern motiviert (s. Definition 3.1.1).

Die dritte Abbruchbedingung (*Baumhöhe*) zielt darauf ab, die Komplexität des resultierenden Baums zu reduzieren, indem die Baumhöhe beschränkt wird. Dies soll zum einen dazu dienen, dass der Baum insgesamt weniger Knoten und Kanten enthält und sich damit auch auf ressourcenbeschränkten Systemen abspeichern lässt und zum anderen, um zu verhindern, dass Bäume vollständig ausgebaut werden. Dies trägt zur Reduktion der Zeit bei, die zum Aufbau und zur Auswertung des Baums in einer nachfolgenden Ausreißerererkennung benötigt wird.

Ausreißerererkennung

Die Ausreißerererkennung erfolgt nun mittels der bereits in Abschnitt 3.5.1 formulierten Ideen bei Verwendung einer einfachen Partitionierung. Statt nun jedoch jene Regionen zu berücksichtigen, die nach einmaliger Partitionierung entstehen, werden jene Regionen berücksichtigt, die nach mehrfacher Partitionierung entstehen. Konkret bedeutet dies, dass die Blattknoten des RSP-Baums für die Einordnung in Normal- oder Ausreißerregionen verwendet werden: Die Hoffnung ist hierbei, dass diese durch die in den Blattknoten gespeicherten Repräsentanten induzierte Partitionierung von \mathcal{D}_{UL} zu einer detaillierteren und damit aussagekräftigeren Ausreißerererkennung führt. Eine Abwandlung von Algorithmus 3.4, der aber einen RSP-Baum berücksichtigt, kann wie folgt angegeben werden:

Algorithmus 3.6 SUMMARY OUTLIER DETECTION (bei Verwendung eines RSP-Baums).

Eingabe: Datensatz \mathcal{D}_{UL} , Distanzfunktion d

- 1: $T \leftarrow \text{FINDERSPBAUM}(\mathcal{D}_{UL}, d)$
 - 2: $L \leftarrow \text{FINDEBLATTKNOTEN}(T)$
 - 3: $R \leftarrow \{R(l) \mid \forall l \in L\}$ ▷ Extrahiere Repräsentanten aus Blattknoten.
 - 4: $\text{KENNEZEICHNEREGIONEN}(R)$ ▷ s. Algorithmus 3.4
-

Hierbei wird zunächst auf Grundlage des Datensatzes \mathcal{D}_{UL} , der auf Ausreißer untersucht werden soll, und einer Distanzfunktion d ein RSP-Baum aufgebaut, wobei der Algorithmus 3.5 verwendet wird. Diese Prozedur liefert den Wurzelknoten T und damit den Baum der im nachfolgenden Schritt traversiert wird, um die Menge der Blattknoten L zu finden. Aus dieser Menge werden dann die Repräsentanten extrahiert, welche zusammen mit Algorithmus 3.4 für die Kennzeichnung von Normal- und Ausreißerregionen verwendet werden.

Es ist wichtig festzuhalten, dass durch die Verwendung der Blattknoten und der Bewertungsstrategien der einfachen Partitionierung (vgl. Abschnitt 3.5.1) die Abbruchbedingung zu einem kritischen Parameter avanciert. Die Abbruchbedingung kontrolliert zusammen mit dem Parameter k , der die Anzahl der Repräsentanten pro Partition angibt, wie viele Regionen gefunden werden. Unbedingt zu verhindern ist dabei, dass jeder Punkt einen eigenen Blattknoten zugeteilt bekommt und damit eine eigene Region angibt. In diesem Falle würde eine für die Ausreißerererkennung bedeutungslose Zusammenfassung an die Kennzeichnungsroutine übergeben werden. Die Anzahl der Repräsentanten pro Partition und die Abbruchbedingung müssen dabei empirisch durch eine experimentelle Evaluation gewählt werden, um diese Situation zu vermeiden.

Laufzeitkomplexität

Um die Laufzeitkomplexität der RSP-Prozeduren zu analysieren, wird zunächst ein Blick auf die Bauminduktion in Algorithmus 3.5 geworfen: Um den Repräsentanten des Wurzelknotens zu finden, werden bei $n = |\mathcal{D}_{UL}|$ Beobachtungen $\mathcal{O}(n)$ Operationen durchgeführt. Anschließend werden für jeden Knoten mit m Beobachtungen zunächst geprüft, ob die Abbruchbedingung erfüllt ist, wobei dies für die Abbruchbedingungen eins und zwei $\mathcal{O}(m)$ Operationen einnimmt. Die dritte Abbruchbedingung kann aus dem Baumgenerierungsprozess abgeleitet werden, nimmt daher konstante Zeit ein und ist für diese Analyse ohne Bedeutung. Ist die Abbruchbedingung nicht erfüllt, wird eine Zusammenfassung S extrahiert, wobei hier analog zu Abschnitt 3.5.1 angenommen wird, dass dies ebenfalls in konstanter Zeit erfolgt. Anschließend muss jeder der m Beobachtungen im aktuellen Knoten einem der Repräsentanten zugeordnet werden, was $\mathcal{O}(km)$ Operationen erfordert (vgl. Zeilen neun und zehn). Die weiteren Operationen erfordern lediglich konstante Zeit und werden hier nicht weiter berücksichtigt. Da sich die Beobachtungen des Datensatzes ausgehend vom Wurzelknoten auf die Kindknoten im partitionierenden Sinne verteilen, kann mit der Laufzeit für *einen* m -elementigen Knoten $\mathcal{O}(m+km) = \mathcal{O}(km)$ die Gesamtkomplexität für den Aufbau des RSP-Baums zu $\mathcal{O}(kn \log n)$ angegeben werden.

Nun wird die Ausreißerererkennung mit RSP-Bäumen anhand Algorithmus 3.6 betrachtet: Der Komplexität der RSP-Bauminduktion wurde im vorherigen Absatz besprochen, wobei nun zusätzlich die Identifizierung der Blattknoten erforderlich ist. Obwohl dies nicht explizit beschrieben wurde, lässt sich bei der Bauminduktion eine Liste vorhalten, in der sämtliche Blattknoten abgespeichert sind. Diese Operation in Zeile zwei des Algorithmus 3.6 lässt sich daher in konstanter Zeit durchführen, wozu ebenfalls die Extraktion der Repräsentanten gehört (Zeile drei). Die Komplexität der Kennzeichnung aller gefundenen Region nimmt nun gemäß Abschnitt 3.5.1 für eine Menge von Repräsentanten R die Komplexität $\mathcal{O}(|R| \cdot n^2)$ ein, was auch die Gesamtkomplexität des Algorithmus darstellt. Wichtig ist an dieser Stelle festzuhalten, dass aufgrund der baumartigen, rekursiven Partitionierung

im Gegensatz zur einfachen Partitionierung $|R| \neq k$ sondern R exponentiell mit $\mathcal{O}(k^{\log n})$ wächst.

3.5.3 Diskussion

Das Verfahren der SUMMARY OUTLIER DETECTION ist in vielerlei Hinsicht eine statistische Methode zur Ausreißerererkennung, wie sie nach Abschnitt 3.3 taxonomisch charakterisiert wird. So funktioniert dieses Verfahren im Kern derart, dass durch eine Menge gefundener Repräsentanten der dazugehörige Datenraum partitioniert wird und anhand des zugrundeliegenden Datensatzes eine Dichte und somit eine Wahrscheinlichkeitsverteilung geschätzt wird. Anstatt nun aber unwahrscheinlichere Bereiche des Datenraums als Ausreißerregionen zu identifizieren, ermöglicht dieses Verfahren den Vergleich der gefundenen Verteilung mit einer Expertenhypothese anhand eines statistischen Tests, dessen Ergebnis die Ausreißereigenschaft für eine ganze Region angibt.

Eine Erweiterung dieses Verfahrens stellt das vorgestellte RECURSIVE SUBMODULAR PARTITIONING (RSP) dar, welches Elemente Clustering-basierter Ausreißererkenntungsverfahren implementiert, um so bedeutsamere und differenziertere Regionen zu identifizieren, die eine qualitativ bessere Vorhersage ermöglichen. Charakteristisch im RSP-Verfahren ist insbesondere das aus hierarchischen Clusteringverfahren bekannte divisive Partitionieren des Datenraums, bei dem die eingegebenen Daten wiederholt voneinander „abgetrennt“ werden. Gegenüber dem klassischen, hierarchischen Clustering ist hier allerdings vorteilhaft, dass die Partitionierung an bestimmten Knoten automatisch abgebrochen wird und eine gefundene Partitionierung der Daten, die beispielsweise als Dendrogramm dargestellt wird, nicht mehr manuell inspiziert werden muss. Weiterhin ist vorteilhaft, dass das Verfahren nicht an Distanzfunktionen gebunden ist und somit beispielsweise auch kernbasierte Ansätze wie die INFORMATIVE VECTOR MACHINE eingesetzt werden können.

Kapitel 4

Implementierung

Submodulare Funktionen und die konkrete Evaluierung einer bestimmten Teilmenge stellen den Kern der in Abschnitt 2.2 vorgestellten Optimierungsverfahren dar und müssen daher zum Erzielen von akzeptablen Laufzeiten sorgfältig implementiert werden. So benötigt beispielsweise der GREEDY-Algorithmus gegeben einer Grundmenge V für das Lösen des Problems 2.3 $\mathcal{O}(|V| \cdot k)$ Funktionsauswertungen. In diesem Kapitel soll daher besprochen werden, wie sich die beiden hier vorgestellten submodularen Funktionen INFORMATIVE VECTOR MACHINE und EXEMPLAR-BASED CLUSTERING effizient implementieren lassen oder wie sich moderne Hardware, wie beispielsweise Grafikprozessoren, für die Funktionsauswertung nutzen lassen, um so geringe Laufzeiten zu erreichen.

4.1 Informative Vector Machine

Gegeben sei eine Teilmenge $S \subseteq V$ mit $V \subseteq X$ eines beliebigen Datenraums X . Zusätzlich sei ein positiv-definites Kernel K beziehungsweise die dazugehörige Kernmatrix \mathbf{K}^S und ein Regularisierungsparameter $\sigma \in \mathbb{R}$ gegeben. Dann ist die submodulare Funktion der INFORMATIVE VECTOR MACHINE durch folgenden Ausdruck definiert:

$$f(S) = \frac{1}{2} \log \det(\mathbf{I} + \sigma^{-2} \mathbf{K}^S) \quad (4.1)$$

Die Addition der Identitätsmatrix \mathbf{I} erfolgt lediglich auf der Hauptdiagonale der Kernmatrix, woraus eine Laufzeit von $\mathcal{O}(|S|)$ resultiert. Die Multiplikation des Regularisierungsparameters und der Kernmatrix wird nur dann durchgeführt, wenn $\sigma \neq 1$, andernfalls muss jedes Element der Matrix aktualisiert werden, was zu einer Laufzeit von $\mathcal{O}(|S|^2)$ führt. Da Matrixbasisoperationen üblicherweise bereits durch Softwarepakete ausreichend optimiert sind, soll der Fokus hier auf zwei anderen Laufzeitfaktoren liegen, nämlich der Berechnung der Kernmatrix und der Determinante. Hierfür soll besprochen werden, welche unterschiedlichen Berechnungsstrategien existieren, welche Unterschiede sich in der asymptotischen Laufzeitkomplexität ergeben und wie sich eine effiziente Implementierung realisieren lässt.

4.1.1 Kernmatrix

Um die Implementierung der Kernmatrix adäquat zu diskutieren wird nicht eine einzelne Funktionsauswertung $f(S)$ sondern eine potenziell unendliche Folge von Funktionsauswertungen $(f(S_i))_{i \in \mathbb{N}}$ betrachtet, wobei zunächst o. B. d. A. eine feste Kardinalität aller zu evaluierenden Mengen $|S| = k$ angenommen wird. Der naive Ansatz zur Berechnung der Kernmatrix $\mathbf{K}^S \in \mathbb{R}^{k \times k}$ berechnet für eine gegebene Menge S diese vollständig neu, was zu einer Laufzeit von $\mathcal{O}(k^2)$ führt, da die gesamte Kernmatrix aufgefüllt werden muss. Eine einfache Optimierung lässt sich dadurch erzielen, dass die Symmetrie des positiv-definiten Kernels ausgenutzt wird und nur eine der beiden Dreiecksmatrizen von \mathbf{K}^S berechnet wird. Dies führt theoretisch zu einer Halbierung der benötigten Laufzeit.

Tatsächlich ist der naive Ansatz nur dann angemessen, wenn $\forall S_i, S_{i+1} : S_i \cap S_{i+1} = \emptyset$, also sich keine zwei aufeinanderfolgenden Mengen in einer Kette von Funktionsausführungen Beobachtungen teilen. Dies ist für die in Abschnitt 2.2 vorgestellten Optimierungsverfahren eine Annahme, die nicht gilt: So würde in der GREEDY-Optimierung zur Entscheidung über die Aufnahme des nächsten Elements eine Menge S_{i-1} betrachtet werden und zu dieser sukzessive verschiedene, noch nicht aufgenommene Elemente hinzugefügt werden, bis feststeht, welches Marginalelement den Grenznutzen maximiert. Dies führt dazu, dass die Menge S_{i-1} für eine hinreichend lange Kette von Funktionsevaluierungen eine echte Teilmenge bildet und sich daraus potentielle Beschleunigungen der Berechnung der Kernmatrix ergeben können. Eine ähnliche Situation ergibt sich im SIEVE STREAMING-Verfahren entlang der Siebe.

Eine mögliche, weitere Optimierung ist daher, dass die gesamte Kernmatrix zwischen zwei Funktionsauswertungen zwischengespeichert wird und nur diejenigen Matrixeinträge geändert werden, die nun auf unterschiedlichen Beobachtungen beruhen. Es werden daher nun zwei aufeinanderfolgende Funktionsauswertungen $f(S_i)$ und $f(S_{i+1})$ betrachtet, wobei die Kernmatrix \mathbf{K}^{S_i} bereits vollständig vorliegt und die nun aktualisiert werden muss, da sich $\Delta = S_{i+1} \setminus S_i, \delta = |\Delta|$ Elemente zwischen den Mengen geändert haben. Sei nun $I = \{i_1, \dots, i_l\}, l \leq k$ die Indexmenge jener Beobachtungen aus S_{i+1} , die in S_i nicht vorhanden waren und für die nun die Kernmatrix aktualisiert werden muss. Für ein festes $\iota \in I$ sind die zu aktualisierenden Zellen $\mathbf{K}_{i,j}^{S_i}$ gerade jene für die gilt, dass $i = \iota$ oder $j = \iota$. Effektiv wird dabei genau eine Zeile und eine Spalte der Kernmatrix aktualisiert, was zu einer Laufzeit von $\mathcal{O}(k)$ für ein geändertes Element beziehungsweise $\mathcal{O}(k\delta)$ für alle veränderten Elemente führt. Dies setzt voraus, dass die Berechnung von Δ effizient und in konstanter Zeit erfolgt. Ändern sich alle Elemente zwischen den Funktionsausführungen, also $\delta = k$, dann ist die Laufzeit äquivalent zum naiven Ansatz mit $\mathcal{O}(k^2)$.

Ändern sich die Kardinalitäten der auszuwertenden Mengen zwischen zwei Funktionsauswertungen $f(S_i)$ und $f(S_{i+1})$, müssen bei einer Verkleinerung der Nachfolgemenge S_{i+1} Zeilen und Spalten der zwischengespeicherten Kernmatrix \mathbf{K}^{S_i} entsprechend entfernt

werden, wobei die reale Laufzeit von Faktoren wie der verwendeten Matriximplementierung, dem Speicherlayout, eventuell benötigten Nachberechnungen und der Frage, ob eine Betrachtung von Dreiecksmatrizen erfolgt, abhängig ist und daher hier nicht besprochen werden soll. Tatsächlich ist es sogar so, dass die in Abschnitt 2.2 vorgestellten Optimierungsverfahren stets $S_i \subseteq S_{i+1}$ garantieren und daher nur eine Vergrößerung der Kernmatrix zu erwarten ist. In diesem Falle wird die (symmetrische) Kernmatrix um eine Spalte und eine Zeile erweitert und die fehlenden Einträge nachberechnet, was zu linearer Laufzeit in der (neuen) Zeilen- beziehungsweise Spaltengröße der Kernmatrix führt.

4.1.2 Determinante

Der zweite gewichtige Laufzeitfaktor bei der Evaluation der IVM ist die Berechnung der Determinante einer bereits bestimmten Kernmatrix $\mathbf{K} = \mathbf{I} + \sigma^{-2}\mathbf{K}^S \in \mathbb{R}^{|S| \times |S|}$. Verfahren zur Determinantenberechnung allgemeiner symmetrischer Matrizen, wie der Laplace'sche Entwicklungssatz weisen eine Laufzeitkomplexität von $\mathcal{O}(|S|!)$ auf, was die Operation gerade für große Matrizen teuer macht. Eine Reduktion der Laufzeit kann dadurch erzielt werden, dass die aus Definition 2.3.1 bekannte Tatsache ausgenutzt wird, dass \mathbf{K} positiv-definit ist¹. In diesem Falle lässt sich nämlich die sogenannte *Cholesky-Zerlegung* anwenden:

4.1.1 Definition (Cholesky-Zerlegung [53]). Sei $\mathbf{A} \in \mathbb{R}^{n \times n}$ eine positiv-definite, quadratische Matrix. Dann ist

$$\mathbf{A} = \mathbf{L}\mathbf{L}' \quad (4.2)$$

die *Cholesky-Zerlegung* von \mathbf{A} , wobei \mathbf{L} eine Linksdreiecksmatrix darstellt. Die Komplexität dieser Operation liegt bei $\mathcal{O}(n^3)$.

Es wird nun also die Cholesky-Zerlegung der Kernmatrix \mathbf{K} betrachtet:

$$\mathbf{K} = \mathbf{L}\mathbf{L}'$$

Da \mathbf{L} nun eine Dreiecksmatrix ist, lässt sich die Determinante dieser Matrix wie folgt schreiben:

$$\det(\mathbf{L}) = \prod_{i=1}^k L_{i,i}$$

¹Es wird an dieser Stelle nicht explizit gezeigt, dass \mathbf{K} tatsächlich positiv-definit ist: Es lässt sich aber mit Definition 2.3.2 einsehen, dass diese Eigenschaft gegeben der positiv-definiten Kernmatrix \mathbf{K}^S durch die Addition der Identitätsmatrix \mathbf{I} mit ausschließlich nicht-negativen Matrixelementen und der Multiplikation mit einem Regularisierungswert $\sigma > 0$ nicht verloren geht.

Für die komplette Kernmatrix \mathbf{K} lässt sich die Determinante daher unter Ausnutzung von $\det(\mathbf{AB}) = \det(\mathbf{A}) \det(\mathbf{B})$ und $\det(\mathbf{A}) = \det(\mathbf{A}')$ für beliebige quadratische Matrizen \mathbf{A} und \mathbf{B} wie folgt angeben:

$$\det(\mathbf{K}) = \det(\mathbf{LL}') = \det(\mathbf{L}) \det(\mathbf{L}') = \det(\mathbf{L}) \det(\mathbf{L}) = \prod_{i=1}^k L_{i,i} \prod_{i=1}^k L_{i,i}$$

Da die IVM nicht nur die Determinante sondern auch den Logarithmus dessen berücksichtigt, lässt sich mit den Logarithmengesetzen eine weitere Vereinfachung vornehmen:

$$\log(\det(\mathbf{K})) = \log\left(\prod_{i=1}^k L_{i,i} \prod_{i=1}^k L_{i,i}\right) = 2 \log\left(\prod_{i=1}^k L_{i,i}\right) = 2 \log\left(\sum_{i=1}^k L_{i,i}\right)$$

Die INFORMATIVE VECTOR MACHINE vermeidet die Verdopplung der logarithmierten L -Matrixspur indem diese durch Halbierung des errechneten Werts wieder entfernt wird.

4.2 Exemplar-based Clustering

Die Implementierung der submodularen Funktion für das EXEMPLAR-BASED CLUSTERING erfolgt anhand der Definition 2.3.6 und wird im folgenden Algorithmus dargelegt:

Algorithmus 4.1 EXEMPLAR-BASED CLUSTERING (CPU)

Eingabe: Datensätze V und $S \subseteq V$, Distanzfunktion d

```

1: function L( $V, S$ )
2:    $\sigma \leftarrow 0$ 
3:   for all  $v \in V$  do
4:      $t \leftarrow \text{FLT\_MAX}$ 
5:     for all  $s \in S$  do
6:        $t \leftarrow \min(t, d(s, v))$ 
7:     end for
8:      $\sigma \leftarrow \sigma + t$ 
9:   end for
10:  return  $\frac{\sigma}{|V|}$ 
11: end function
12: return  $L(V, \{\vec{e}_0\}) - L(V, S \cup \{\vec{e}_0\})$ 

```

Es ist hierbei zu sehen, dass der Algorithmus in dieser (naiven) Implementierung eine Komplexität von $\mathcal{O}(|V| \cdot |S|)$ aufweist. Diese hohe Komplexität weist einen problematischen Charakter auf, der insbesondere dann deutlich wird, wenn entweder die Grundmenge V und die zu evaluierenden S groß sind oder aber die Optimierungsroutinen viele Funktionsevaluationen vornehmen (vgl. Abschnitt 2.2). Daher sollen Techniken untersucht werden, die die benötigte Laufzeit reduzieren.

Indexstrukturen Das in Algorithmus 4.1 dargestellte Verfahren ist ein sogenannter *Brute Force*-Ansatz. Hierbei wird für jedes Paar aus Grundmengenvektor und Evaluationsmengenvektor die entsprechende Distanz berechnet und so schrittweise das Minimum gefunden. Dieser Brute-Force-Ansatz könnte dadurch verbessert werden, dass helfende Datenstrukturen eingesetzt werden, die die Zeitkomplexität verringern. Tatsächlich ist die L -Funktion des EXEMPLAR-BASED CLUSTERINGS durch eine zentrale Query gekennzeichnet, nämlich die Frage danach, welcher Punkt aus der Evaluationsmenge S dem aktuell gewählten Grundmengenvektor am nächsten ist. Solche Queries heißen *k-nächste-Nachbarn-Queries* und werden beispielsweise durch k - d -Bäume unterstützt. Diese werden auf Grundlage eines Datensatzes $\mathcal{D} \subset X$ aufgebaut und führen eine baumartige Partitionierung des Datensatzes entlang jeweils *verschiedener* Dimensionen des Eingaberaums X durch. Anschließend kann mithilfe dieser Datenstruktur für *beliebige* Vektoren $\vec{x} \in X$ erfragt werden, welcher Vektor aus \mathcal{D} diesem am nächsten ist. Für eine Anwendung dieser Indexstruktur innerhalb der L -Funktion wäre es also erforderlich, dass diese auf S aufgebaut wird, sodass erfragt werden kann, welcher Vektor $\vec{s} \in S$ zu einer Query-Beobachtung $\vec{v} \in V$ am nächsten ist. Da sich jedoch S (im Gegensatz zu V) mit jedem Funktionsaufruf ändert, müsste die Indexstruktur bei jedem Aufruf neu aufgebaut werden. Daher wird die Möglichkeit der Verwendung von Indexstrukturen an dieser Stelle nicht weiter verfolgt.

CPU-Parallelisierung Eine weitere Möglichkeit der beschleunigten Ausführung von Algorithmus 4.1 besteht in der Ausnutzung von Parallelität, die durch heutige CPU-Mehrkern- und NUMA²-Architekturen einfach implementierbar ist. Tatsächlich ist das EXEMPLAR-BASED CLUSTERING in der oben pseudokodifizierten Form trivial parallelisierbar, indem die äußere Schleife der L -Funktion durch mehrere Ausführungsfäden ausgeführt wird. Eine Realisierung dieser Strategie findet sich in Algorithmus 4.2, wobei zu beachten ist, dass eine Parallelisierung der inneren Schleife nicht möglich ist, da das Ausführungsergebnis intern von der Berechnung des (vorherigen) Minimums abhängt (vgl. Zeile 6, Algorithmus 4.2).

CPU-Mengenparallelisierung Ein weitere Möglichkeit zur Parallelisierung ergibt sich im Zusammenhang mit der Betrachtung der in Abschnitt 2.2 vorgestellten Optimierer: Tatsächlich ist es so, dass jede dieser Optimierer stets die Funktionswerte mehrere Mengen auswertet. So prüft GREEDY anhand einer bereits ausgewählten Menge im Sinne einer gierigen Strategie, welches noch nicht ausgewählte Element der Grundmenge den Nutzensgewinn am stärksten maximieren würde und bildet dementsprechende zu evaluierende Mengen, deren Funktionswerte jeweils in einer Differenzbetrachtung zur bereits ausgewählten Menge die Entscheidung liefern. Der SIEVE STREAMING-Algorithmus muss hingegen

²engl. *non-uniform memory access*: CPU-Verbundstruktur, in dem Prozessoren abhängig von der konkreten Speicheradresse nicht die gleichen Zugriffszeiten auf den Speicher garantiert werden.

Algorithmus 4.2 EXEMPLAR-BASED CLUSTERING (CPU, parallel)

Eingabe: Datensätze V und $S \subseteq V$, Distanzfunktion d

```

1: function LPARALLEL( $V, S$ )
2:    $\sigma \leftarrow$  new Array für jedes  $v \in V$ .
3:   for all  $v \in V$  in parallel do
4:      $t \leftarrow$  FLT_MAX
5:     for all  $s \in S$  do
6:        $t \leftarrow \min(t, d(s, v))$ 
7:     end for
8:      $\sigma_v \leftarrow \sigma_v + t$ 
9:   end for
10:   $\Sigma \leftarrow$  reduce  $\sigma$  by sum in parallel
11:  return  $\Sigma$ 
12: end function
13: return LPARALLEL( $V, \{\vec{e}_0\}$ ) – LPARALLEL( $V, S \cup \{\vec{e}_0\}$ )

```

bei Ankunft eines neuen Elements parallel mehrere Mengen oder *Siebe*, entsprechend ihrem zugeordnetem Grenzwert, aktualisieren (vgl. Algorithmus 2.3 und 2.4). Folglich ist es für diesen Anwendungsfall idealer einen Algorithmus zu entwerfen, der eine Menge von Eingabemengen $S_{\text{multi}} = \{S_1, \dots, S_l\} \subset \mathcal{P}(\mathcal{P}(V))$ betrachtet und dem entsprechend eine Menge von Funktionswerten $F = \{f(S_1), \dots, f(S_l)\}$ liefert. Ein CPU-paralleler Algorithmus, der dieses Problem löst, kann wie folgt angegeben werden:

Algorithmus 4.3 EXEMPLAR-BASED CLUSTERING (CPU, mengen-parallel)

Eingabe: Datensätze V und $S_{\text{multi}} = \{S_1, \dots, S_l\} \subset \mathcal{P}(\mathcal{P}(V))$, Distanzfunktion d

```

1:  $F \leftarrow$  new Array für jedes  $S_i \in S_{\text{multi}}$ .
2: for all  $S_i \in S_{\text{multi}}$  in parallel do
3:    $F_i \leftarrow L(V, \{\vec{e}_0\}) - L(V, S_i)$ 
4: end for
5: return  $F$ 

```

Die Komplexität dieses Problems wächst gegenüber dem Algorithmus 4.1 zu $\mathcal{O}(|V| \cdot |S| \cdot |S_{\text{multi}}|)$ und begründet verstärkt die Notwendigkeit für ein schnelles Lösungsverfahren.

Beschleunigung Die durch den Algorithmus 4.2 erreichbare Beschleunigung liegt bei $|V|$, während sie bei Algorithmus 4.3 bei $|V| \cdot |S_{\text{multi}}|$ liegt. Diese Beschleunigung kann bei größeren Datensätzen und heutigen Mehrkern-CPU-Architekturen allerdings nicht erreicht werden, da nicht genügend Recheneinheiten zu Verfügung stehen. Ein NUMA-Verbund von CPUs oder ein klassischer Rechnerverbund sieht zwar prinzipiell keine Limitierung

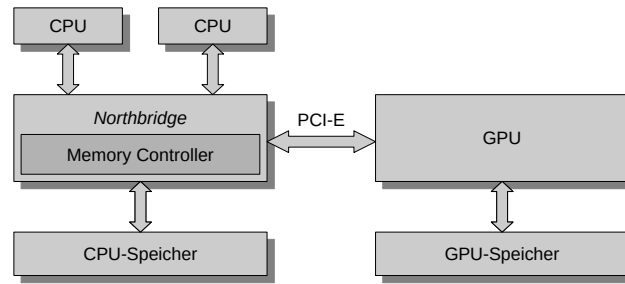


Abbildung 4.1: Schematische Darstellung der Integration einer GPU in ein Zweikern-CPU-System mit einer sogenannten *Northbridge* über eine PCI-Express (PCI-E)-Verbindung [64].

hinsichtlich der einsatzfähigen CPUs vor, weist allerdings andere Nachteile auf, wie variierende Speicherlatenzen und damit einhergehende Lokalitätseffekte, die es bei der Implementierung zu berücksichtigen gilt. Um dieses Problem zu lösen und weitere Parallelität nutzbar zu machen, können spezialisierte Koprozessoren genutzt werden, die architektonisch auf das Berechnen von derartigen Algorithmen ausgelegt sind. Neben Xeon Phi und *Field Programmable Gate Arrays* (FPGAs) gehören vor allem *Graphics Processing Units* (GPUs) in diese Gruppe. Diese sollen aufgrund ihrer breiten Verfügbarkeit hier evaluiert werden.

4.2.1 General Purpose GPU

GPUs stellen Koprozessoren dar, die typischerweise für das Verarbeiten von Bilddaten verwendet werden und darauf spezialisiert sind, gleichartige Operationen, wie beispielsweise der massenhaften Berechnung von Distanzen wie sie hier erforderlich ist, auf vielen Daten parallel auszuführen. Dieses Schema wird entsprechend der Flynn'schen Klassifikation *Single Instruction Multiple Data* (SIMD) [19] oder *Datenparallelismus* genannt, welches sich im Kontext von Grafikprozessoren neben der Berechnung von reinen Bilddaten auch für allgemeine Probleme anwenden lässt. Dies führt auf den Begriff der *general purpose GPU* (GPGPU).

Integration Da sich eine GPU als Koprozessor in ein bestehendes System integriert, soll kurz anhand der Abbildung 4.1 beleuchtet werden, wie diese Integration bewerkstelligt wird. Üblicherweise existiert bereits eine Architektur aus einer CPU, die wie hier im Beispiel entweder zwei oder meistens mehr CPU-Kerne bereitstellt, und die über eine sogenannte *Northbridge* mit dem CPU-(Haupt-)Speicher verbunden ist. Im Kontext der GPGPU-Programmierung wird dieser Teil des Hardwaresystems schlicht *host* genannt. Die Integration der GPU, welche in diesem Kontext *device* genannt wird, erfolgt dabei über die PCI-Express-Schnittstelle. Dies ist besonders bedeutsam, da das meistens notwendige zu Verfügung stellen von Nutzdaten (wie beispielsweise die Grundmenge V oder zu evaluierende Mengen S_{multi}) immer über diese Schnittstelle erfolgen muss und daher

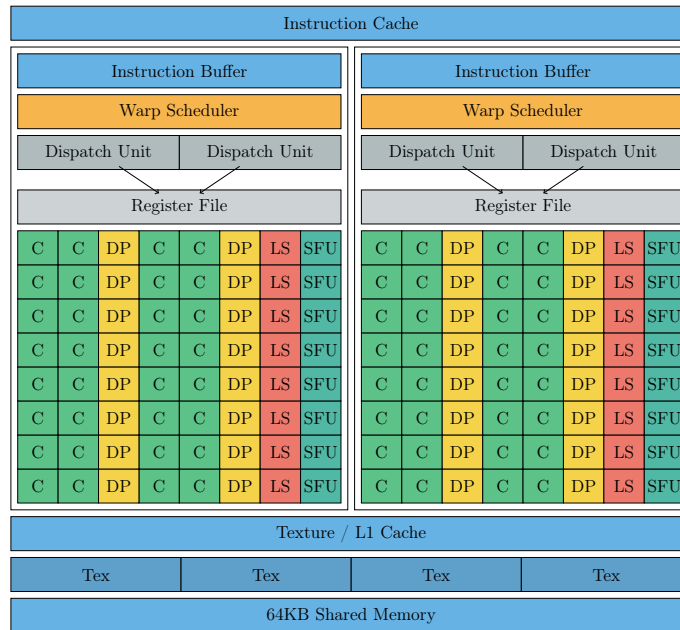


Abbildung 4.2: Streaming Multiprocessor (SM) einer GP100-GPU des Herstellers NVIDIA (eigene Darstellung nach [15]).

einen beschränkenden Charakter hinsichtlich der erreichbaren Geschwindigkeit darstellen kann. Dies fällt besonders dann stark ins Gewicht, wenn viele Datentransferoperationen durchgeführt werden müssen.

Rechenarchitektur Um zu verstehen, warum GPUs für das Lösen von hochparallelen Problemen geeignet sind, woher diese ihre Rechenkraft beziehen und welche Limitierung existieren, soll ein exemplarischer Blick auf die *Pascal*-GPU-Architektur des Herstellers NVIDIA geworfen werden. Die GP100-GPU, die diese Architektur implementiert, besteht neben speicherrelevanten Bestandteilen wie einem *Memory Controller* vor allem aus einem Array von 60 sogenannten *Streaming Multiprocessors* (SM), die die Berechnung von Instruktionen übernehmen [15] und daher für das Verständnis von GPUs als hochparalleles Rechenwerk von besonderer Bedeutung sind.

Der in Abbildung 4.2 schematisch dargelegte SM besteht dabei neben einem *Instruction Cache* und einem *Texture Cache*, die jeweils Instruktionen und Daten zwischenspeichern, insbesondere aus zwei sogenannten *processing blocks*, in denen die Rechenarbeit stattfindet. Diese beinhalten ein Array von Ausführungskernen (engl. *execution cores* oder „CUDA-Cores“), welche je nach Typus mannigfaltige Aufgaben übernehmen: Die Berechnungskerne (C) dienen dem Berechnen von Instruktionen der Integer- oder 32-bit-Gleitkommaarithmetik, während die Double-Precision-Kerne (DP) dies für die 64-bit-Gleitkommaarithmetik übernehmen. Die *Special Functions Units* (SFUs) berechnen hingegen sogenannte *transzendente Funktionen*, wie die Logarithmusfunktion. Load-Store-

Kerne dienen dem Laden und Speichern von Daten. Welche dieser Ausführungskerne zur Berechnung einer Instruktion ausgewählt wird, bestimmt dabei die sogenannte *Dispatch Unit*, die den Pool ausführbarer Instruktion wiederum vom *Warp Scheduler* bezieht. Dieser wählt dabei einen sogenannten *warp* – eine Gruppe von 32 Threads, die jeweils an der gleichen Adresse starten und die gleiche Instruktion ausführen (Datenparallelismus) – aus, der innerhalb des jeweiligen Processing Blocks ausgeführt werden soll. Es wird hierbei deutlich, dass sich die GPU vor allem durch die Vielzahl an Ausführungskernen für das Lösen von hochparallelen Aufgaben empfiehlt. Hierbei ist allerdings das Ungleichgewicht von 32-bit-Gleitkommaeinheiten zu 64-bit-Gleitkommaeinheiten festzuhalten, welches stets dazu führt, dass Berechnungen mit doppelter Präzision tendenziell langsamer sind, als Berechnungen einfacher Präzision.

Neben den Berechnungskernen und den Caches ist allerdings auch der sogenannte *geteilte Speicher* (engl. *shared memory*) ein wichtiger Hardwarebestandteil, welcher kurz erwähnt werden soll: In diesen können bei der GP100-GPU bis zu 64KB Nutzdaten geladen werden, die sich direkt im SM befinden (engl. *on-chip memory*) und auf diese daher besonders latenzarm zugegriffen werden kann. Dies steht im Kontrast zu dem *globalen Speicher* (engl. *global memory*), der deutlich größere Datenmengen speichern kann, sich aber nicht auf dem GPU-Chip (engl. *off-chip memory*) befindet und bei dem Datenzugriffe deutlich latenzbehafteter ausfallen.

Programmiermodell Nachdem nun die GPU aus der Perspektive der Hardware betrachtet wurde, soll nun ein Blick auf das Programmiermodell geworfen werden. Historisch sind dabei insbesondere CUDA und OpenCL prominente Vertreter von Frameworks, die das Programmieren von Code für GPUs ermöglichen. Der Fokus soll im Folgenden auf CUDA liegen.

Im Gegensatz zur CPU-basierten Mehrfädigkeit, in denen sich Threads quasi beliebig organisieren lassen (beispielsweise über Fork-Join-Konzepte, Jobpools oder Producer-Consumer-Systeme) muss im CUDA-Framework eine Abbildung der abzuarbeitenden, parallelen Aufgabe in eine Gitter-Block-Struktur gefunden werden. Das Gitter (engl. *grid*) besteht dabei aus bis zu drei Dimensionen, welches wiederum die Blöcke enthält. Ein Block kann ebenfalls aus bis zu drei Dimensionen besteht und enthält maximal 1024 Threads (s. Abbildung 4.3). Der Programmcode, der dann auf der GPU parallel ausgeführt und auch *kernel* genannt wird, wird dann entsprechend dieser Struktur, die vom Programmierer in Form einer *Kernelkonfiguration* vorgegeben wird, instanziiert. Formal lässt sich diese Kernelkonfiguration als $C = (D_g, D_b)$ mit $D_g = (g_x, g_y, g_z) \in \mathbb{N}^3$ der Dimensionierung des Gitters und $D_b = (b_x, b_y, b_z) \in \mathbb{N}^3$ der Dimensionierung jedes Blocks im Gitter angeben.

Jedem Thread ist dabei bekannt, welche Position $P_t = (t_x^*, t_y^*, t_z^*)$ dieser in einem konkreten Block $P_b = (b_x^*, b_y^*, b_z^*)$ einnimmt und wie dieser Block dimensioniert ist. Diese Information ist essentiell, da der Kernel-ausführende Thread daraus ableitet, welche Auf-

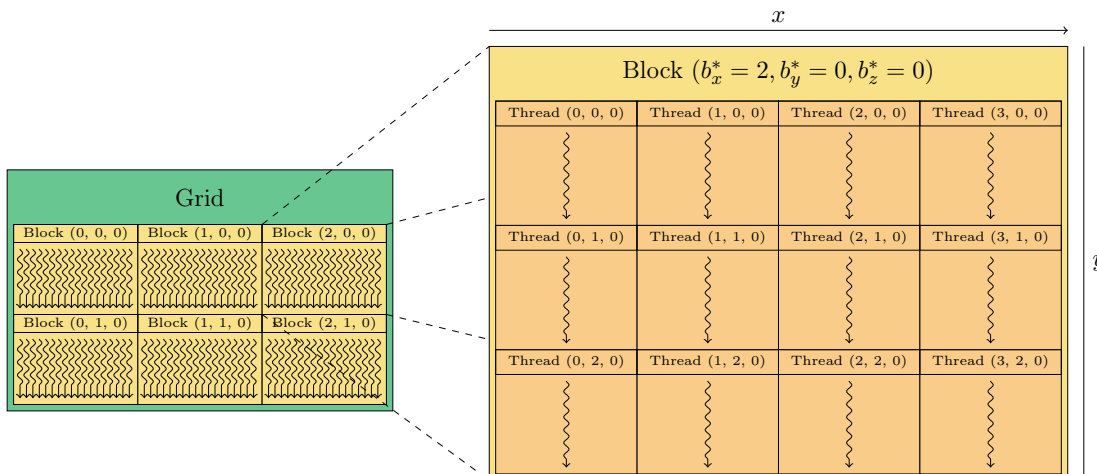


Abbildung 4.3: Aus einer Kernelkonfiguration $C = (D_g, D_b)$ resultierende Gitter-Block-Struktur mit $D_g = (g_x = 3, g_y = 2, g_z = 1)$ und $D_b = (b_x = 4, b_y = 3, b_z = 1)$, wie sie zur Berechnung eines GPU-Kernels herangezogen werden könnte (eigene Darstellung nach [16]).

gabe jeder individuelle Thread abzuarbeiten hat und an welcher Position im Speicher ein Ergebnis abgelegt wird.

Ferner existieren im CUDA-Programmiermodell einfache Möglichkeiten zur blockweisen Barriersynchronisation, mit der garantiert werden kann, dass alle Threads eines Blocks einen bestimmten Codeabschnitt passiert haben, bevor nachfolgende Codeabschnitte ausgeführt werden können. Ein Anwendungsfall für diese Form der Synchronisation stellt das Laden von Nutzdaten in den bereits erwähnten geteilten Speicher dar: Jedem Block kann dabei, wie oben bereits erwähnt, eine begrenzte Menge dieses besonders schnellen Speichers *exklusiv* zu Verfügung gestellt werden, wobei die Daten erst bei der Ausführung des Kernels dort hinein geladen werden. Um zu garantieren, dass alle Threads eines Blocks *ihre* Nutzdaten in den geteilten Speicher geladen haben, wird die Barriersynchronisation genutzt.

Algorithmus Nun soll ein GPU-Algorithmus für das mengenparallele Problem des EXEMPLAR-BASED CLUSTERINGS angegeben werden, wobei wie in der Definition 2.3.6 eine Grundmenge $V = \{\vec{v}_1, \dots, \vec{v}_n\} \subset X$ eines beliebigen Datenraums X , die k -Medoids-Verlustfunktion $L : \mathcal{P}(X) \rightarrow \mathbb{R}$, ein Hilfsvektor $\vec{e}_0 \in X$ und die dazugehörige submodulare Funktion $f : \mathcal{P}(V) \rightarrow \mathbb{R}$ betrachtet wird. Aus technischer Perspektive seien alle benötigten Daten im globalen Speicher der GPU vorhanden. Die Funktionen sind dabei im Einzelnen wie folgt gegeben:

$$L(S) = \frac{1}{|V|} \sum_{\vec{v} \in V} \min_{s \in S} d(v, s) \quad (4.3)$$

$$f(S) = L(\{\vec{e}_0\}) - L(S \cup \{\vec{e}_0\}) \quad (4.4)$$

Die zu implementierende Distanzfunktion $d(\cdot, \cdot)$ sei hierbei das Quadrat der euklidischen Distanz, die im Folgenden kurz definiert werden soll:

4.2.1 Definition (Euklidische Distanz). Seien $\vec{x}, \vec{y} \in X$ zwei Vektoren aus einem Datenraum X mit Skalarprodukt $\langle \cdot, \cdot \rangle$, dann ist die euklidische Distanz $d_{\text{Euklid}}(\vec{x}, \vec{y})$ eine Distanzfunktion (vgl. Definition 2.3.5), die durch den Ausdruck

$$d_{\text{Euklid}}(\vec{x}, \vec{y}) = \|\vec{x} - \vec{y}\| \quad (4.5)$$

gegeben ist, wobei $\|\cdot\|$ die euklidische Norm ist, die wie folgt definiert wird [7]:

$$\|\vec{x}\| = \langle \vec{x}, \vec{x} \rangle \quad (4.6)$$

Es ist hierbei zu erkennen, dass der Teilausdruck $L(\{\vec{e}_0\})$ der submodularen Funktion f bei fixem Hilfsvektor $\vec{e}_0 \in X$ lediglich von der Grundmenge V abhängt, von der annehmbar ist, dass diese a-priori bekannt ist. Da sich die Zeitkomplexität dieses Ausdrucks lediglich auf $\mathcal{O}(|V|)$ beläuft, wird der resultierende Wert auf konventionelle Art von der CPU vorberechnet und steht damit für *alle* zukünftigen Funktionsevaluationen von f zu Verfügung. Der Fokus der Implementierung liegt daher im Folgenden auf dem zweiten Teilausdruck $L(S \cup \{\vec{e}_0\})$, der von der tatsächlich zu evaluierenden Menge S abhängt und die die Komplexität des Algorithmus 4.1 begründet. Da hier das mengenparallele Problem (vgl. Algorithmus 4.3) gelöst werden soll, wird hier analog zur bisherigen Problemstellung eine Menge von Mengen $S_{\text{multi}} = \{S_1, \dots, S_l\}$ betrachtet, für die unabhängig voneinander der Funktionswert berechnet werden soll.

Damit die GPU möglichst viele unabhängige Aufgaben erhält, wird eine Dekomposition dieser Funktion, wie folgt, betrachtet:

$$L_{\vec{v}_i}(S) = \frac{\min_{\vec{s} \in S} d(\vec{v}_i, \vec{s})}{|V|} \quad (4.7)$$

$$L(S) = L_{\vec{v}_1}(S) + \dots + L_{\vec{v}_n}(S) \quad (4.8)$$

Mittels dieser Dekomposition lässt sich im Folgenden zusammen mit S_{multi} eine Arbeitsmatrix A definieren, die als Grundlage für die später zu bestimmende Gitter-Block-Struktur dient:

$$\begin{array}{c} \xrightarrow{\text{Elemente der Grundmenge (x-Richtung)}} \\ \left(\begin{array}{cccc} L_{\vec{v}_1}(S_1) & L_{\vec{v}_2}(S_1) & \cdots & L_{\vec{v}_n}(S_1) \\ L_{\vec{v}_1}(S_2) & L_{\vec{v}_2}(S_2) & \cdots & L_{\vec{v}_n}(S_2) \\ \vdots & \vdots & \ddots & \vdots \\ L_{\vec{v}_1}(S_l) & L_{\vec{v}_2}(S_l) & \cdots & L_{\vec{v}_n}(S_l) \end{array} \right) = A \\ \downarrow \text{Mengen aus } S_{\text{multi}} \text{ (y-Richtung)} \end{array} \quad (4.9)$$

Eine Zeilenreduktion mittels der Summe liefert dann, wie in Gleichung 4.8 vorgesehen, den Funktionswert für jede Menge aus S_{multi} . Die so definierte Arbeitsmatrix A dient nun

$$A = \begin{pmatrix} L_{\vec{v}_1}(S_1) & L_{\vec{v}_2}(S_1) & L_{\vec{v}_3}(S_1) & L_{\vec{v}_4}(S_1) \\ L_{\vec{v}_1}(S_2) & L_{\vec{v}_2}(S_2) & L_{\vec{v}_3}(S_2) & L_{\vec{v}_4}(S_2) \\ L_{\vec{v}_1}(S_3) & L_{\vec{v}_2}(S_3) & L_{\vec{v}_3}(S_3) & L_{\vec{v}_4}(S_3) \\ L_{\vec{v}_1}(S_4) & L_{\vec{v}_2}(S_4) & L_{\vec{v}_3}(S_4) & L_{\vec{v}_4}(S_4) \end{pmatrix}$$

Abbildung 4.4: Beispielhafte Arbeitsmatrix A zur Lösung eines Problems des EXEMPLAR-BASED CLUSTERINGS mit $V = \{v_1, \dots, v_4\}$ und $S_{\text{multi}} = \{S_1, \dots, S_4\}$ mit dazu berechneter Kernelkonfiguration. Alle Spalten der Matrix A werden in jeweils einem Block (gelb) von vier Threads gruppiert, wobei jeder Thread des Blocks genau eine Zelle bearbeitet.

zum Aufbau der Kernelkonfiguration $C = (D_g, D_b)$, die so gewählt wird, dass jeder GPU-Thread genau eine Zelle der Matrix A bearbeitet. Dabei sollen jedoch die Möglichkeiten des geteilten Speichers genutzt werden.

Wie in A deutlich wird, arbeitet jede Spalte i der Matrix auf Grundlage des gleichen Elements der Grundmenge $\vec{v}_i \in V$. Dabei kann leicht erkannt werden, dass alle Prozesse, die jeweils auf *einer* Spalte der Matrix A arbeiten, zum Zwecke der Distanzberechnung in Gleichung 4.7 häufige Zugriffe auf diesen Vektor vornehmen, die sich leicht durch die Nutzung von latenzarmen geteiltem Speicher *cachen* lassen, der jedoch blockexklusiv ist und daher die Kernelkonfiguration determiniert. Die Idee sei nun, eben genannte Prozesse, die die Werte einer Spalte i in der Arbeitsmatrix A berechnen in einem Block zu gruppieren und den dazugehörigen $\vec{v}_i \in V$ im schnelleren geteilten Speicher bereitzustellen, von wo aus Zugriffe im Vergleich zum globalen Speicher günstiger erfolgen können. Um zu verhindern, dass ein Grundmengenvektor mehrfach in den knappen geteilten Speicher geladen wird, muss die Dimension des Blocks in Richtung der Mengen aus S_{multi} (also in y -Richtung, vgl. Ausdruck 4.9) maximiert werden, da ansonsten mehrere Blöcke instanziiert werden müssen, die den gleichen Grundmengenvektor als Grundlage besitzen. Dabei müssen die Restriktionen des Programmiermodells beachtet werden, sodass nicht mehr als 1024 Threads pro Block instanziiert werden und der verfügbare, geteilte Speicher pro Block $\beta \in \mathbb{N}^+$ nicht überschritten wird. Konkret lässt sich dann die Ausdehnung *jedes* Blocks $D_b = (b_x, b_y, b_z)$, zusammen mit dem Speicherverbrauch für einen Vektor der Grundmenge $\gamma \in \mathbb{N}^+$, wie folgt angeben:

$$b_x = \min \left\{ \left\lfloor \frac{1024}{b_y} \right\rfloor, \left\lfloor \frac{\beta}{\gamma} \right\rfloor \right\} \quad (4.10)$$

$$b_y = \min \{1024, |S_{\text{multi}}|\} \quad (4.11)$$

$$b_z = 1 \quad (4.12)$$

Der Konfigurationsparameter b_y enthält dann genau die oben beschriebene Maximierung: Es werden entweder so viele Threads in y -Richtung erzeugt, wie Mengen in S_{multi} vorhanden sind oder aber höchstens 1024 Threads, um den Anforderungen des Programmiermodells

zu genügen. Der Parameter b_x fügt sich dann dem bereits gewählten b_y -Parameter: Es werden in x -Richtung entweder so viele Threads erzeugt, wie noch entsprechend dem b_y -Parameter möglich sind oder aber, wie viele Grundmengenenelemente noch in den geteilten Speicher dieses Blocks passen. Da die Arbeitsmatrix lediglich zweidimensional ist, gilt $b_z = 1$.

Die Dimensionierung des Gitters $D_g = (g_x, g_y, g_z)$ erfolgt dann so, dass das Problem, welches durch die Arbeitsmatrix A beschrieben ist, gelöst wird:

$$g_x = \left\lceil \frac{|V|}{b_X} \right\rceil \quad (4.13)$$

$$g_y = \left\lceil \frac{|S_{\text{multi}}|}{b_Y} \right\rceil \quad (4.14)$$

$$g_z = 1 \quad (4.15)$$

Sollten Threads existieren, die sich aufgrund ihrer Indizierung und der Dimensionierung des dazugehörigen Blocks, keiner Zelle in der Arbeitsmatrix 4.9 zuordnen lassen, wird über einen entsprechenden Konditionalausdruck die Ausführung des Fadens verhindert.

Ausgehend von der nun vollständig bestimmten Kernelkonfiguration $C = (C_G, C_B)$ kann nun ein GPU-paralleles Lösungsverfahren zum EXEMPLAR-BASED CLUSTERING angegeben werden. Jede Threadinstanz, die den angegebene Algorithmus 4.4 ausführt, bearbeitet nun genau eine Zelle der Arbeitsmatrix A und legt das berechnete Ergebnis in genau dieser wieder ab, wofür der entsprechende Speicherraum allokiert wird.

In den Zeilen 1 und 2 wird dabei zunächst anhand der Koordinaten der konkreten Blockinstanz im Gitter P_b , der Koordinaten der konkreten Threadinstanz im Block P_t und der Blockausdehnung D_b die jeweilige Spalte i und Zeile j der Arbeitsmatrix A bestimmt, für die dieser Thread den Wert der dekompositionierten Verlustfunktion berechnen soll. Unabhängig der Blockzugehörigkeit lädt jeder erste Thread in y -Richtung den dazugehörigen Grundmengenvektor (vgl. Matrix 4.9) in den geteilten Speicher (Zeilen 3 bis 5) und macht ihn so für die anderen Threads des selben Blocks verfügbar. Die anderen Threads des gleichen Blocks warten derweilen darauf, dass dieser Prozess abgeschlossen wird (Zeile 6). Anschließend wird der entsprechende (Teil-)Wert der Verlustfunktion für die konkrete Zelle der Arbeitsmatrix berechnet (Zeile 7) und der Wert an der jeweiligen Position abgelegt (Zeile 8). Weitere Berechnungen auf der GPU werden nicht betrachtet. Eine Summenreduktion der Arbeitsmatrix A in Spaltenrichtung auf der CPU mittels konventioneller Methoden liefert in $\mathcal{O}(|V|)$ Operationen den Ergebnisvektor $\vec{e} \in \mathbb{R}^{|S_{\text{multi}}|}$, der die Werte der submodularen Funktion für jede einzelne, eingegebene Menge bereit hält.

Speicherorganisation Bisher wurde angenommen, dass sich die zur Berechnung der Funktion benötigten Daten in irgendeiner Form im globalen Speicher der GPU befinden. Es wurde lediglich beschrieben, dass der zu einem Block gehörende Grundmengenvektor aus dem globalen in den geteilten Speicher geladen wird. In diesem Abschnitt soll daher

Algorithmus 4.4 EXEMPLAR-BASED CLUSTERING (GPU)

Eingabe: Grundmenge $V \subset X^d$, Mengen S_{multi} , Blockinstanz $P_b = (b_x^*, b_y^*, b_z^*)$, Threadinstanz $P_t = (t_x^*, t_y^*, t_z^*)$, Blockausdehnung $D_b = (b_x, b_y, b_z)$

- 1: $i \leftarrow b_x \cdot b_x^* + t_x^*$ ▷ Index des zu bearbeitenden Grundmengenelements $\vec{v}_i \in V$.
- 2: $j \leftarrow b_y \cdot b_y^* + t_y^*$ ▷ Index der zu bearbeitenden Menge $S_j \in S_{\text{multi}}$.
- 3: **if** $t_y^* = 0$ **then**
- 4: **load** $\vec{v}_i \in V$ **from** global memory **into** shared memory
- 5: **end if**
- 6: **synchronize** threads **in** block ▷ Warten, bis $\vec{v}_i \in V$ im geteilten Speicher liegt.
- 7: $d_{\min} \leftarrow \text{FP_MAX}$
- 8: **for all** $\vec{s}_i \in S_j$ **do**
- 9: $d \leftarrow 0$
- 10: **for** $k = 1$ **to** d **do**
- 11: $d \leftarrow (\vec{v}_i[k] - \vec{s}_i[k])^2$ ▷ s. Gleichung 4.7.
- 12: **end for**
- 13: $d_{\min} \leftarrow \min(d_{\min}, d)$
- 14: **end for**
- 15: $A_{j,i} = \frac{d_{\min}}{|V|}$ ▷ Speichere den Wert der Verlustfunktion in der Arbeitsmatrix.

präzisiert werden, wie die Grundmenge V und die zu evaluierenden Mengen S_{multi} auf dem Speicher der GPU abgelegt werden. Hierfür wird die Grundmenge V in eine Grundmengenmatrix \mathbf{V} überführt und S_{multi} als Menge von Evaluationsmatrizen $\{\mathbf{S}_1, \dots, \mathbf{S}_l\}$ betrachtet.

GPU-Speicher unterscheiden sich hinsichtlich der Zugriffsmethodik nicht von herkömmlichen CPU-Hauptspeichern: Es steht ein für die Applikation allozierter Speicherraum zu Verfügung, wobei eine eindimensionale Adressierung den Ort anzeigt, aus dem Datenwörter gelesen oder in den Datenwörter geschrieben werden sollen. Während sich Vektoren als eindimensionale Datenstruktur direkt in einem solchen Speicher (hintereinander) ablegen lassen, stellen Matrizen aufgrund ihrer zweidimensionalen Adressierung (anhand von Zeilen und Spalten) *komplexe* Datenstrukturen dar, für die diese Form der linearen Adressierung nicht gangbar ist. Um diese Situation aufzulösen, muss die Matrix für den GPU-Koprozessor in einen Vektor überführt werden und eine Abbildung gefunden werden, sodass sich jedes Matrixelement, identifiziert durch Zeilen- und Spaltenposition, in diesem Vektor adressieren lässt. Diese Operation wird hier *Vektorisierung* genannt. Die beiden dominierenden Lösungsstrategien sind dabei jeweils die zeilen- und spaltenweise Speicherung, bei denen entweder alle Zeilen oder alle Spalten nacheinander abgelegt werden. Welche Speicherungsform dabei bessere Resultate liefert, hängt stark vom zugrundeliegenden Problem ab und ist beispielsweise bereits für Datenbanken und CPU-Cacheeffekte untersucht worden

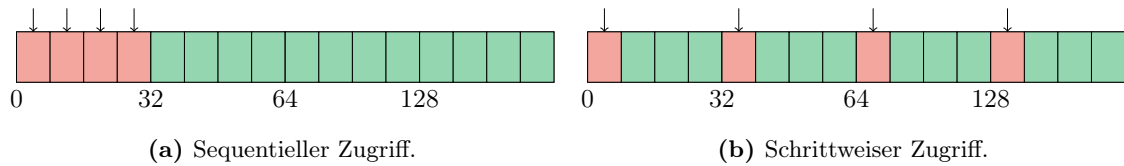


Abbildung 4.5: Beispielhafter GPU-Speicherbereich bestehend aus vier Segmenten zu je 32 Byte. (a) Es werden vier 8 Byte-Wörter aus einem Speichersegment des globalen Speichers angefordert, was zu genau einer Speichertransaktion führt. (b) Es werden vier 8 Byte-Wörter aus vier Speichersegmenten des globalen Speichers angefordert, was zu vier Speichertransaktionen führt.

[2]. Für den hier implementierten Algorithmus wird die Grundmengenmatrix \mathbf{V} spaltenweise vektorisiert und keine weiteren Optimierungen durchgeführt, da diese verhältnismäßig klein ist, bereits manuell in den als Cache fungierenden geteilten Speicher geladen wird und ansonsten keine weiteren Zugriffe auf diese Matrix durchgeführt werden.

Eine andere Situation ergibt sich bei den Evaluationsmatrizen, da diese in hoher Zahl vorkommen und sich aufgrund ihrer Größe nicht im geteilten Speicher cachen lassen. Zugriffe müssen daher zwangsläufig auf den globalen Speicher erfolgen und hierfür einige Effizienzüberlegungen angestellt werden. Im CUDA-Framework wird der globale Speicher in Segmente fester Größe (üblicherweise 32 Byte) zerlegt, sodass sich beispielsweise im Doppelpräzisionsbetrieb vier Gleitkommazahlen in einem Segment ablegen lassen. Greifen Threads eines *warps* lediglich auf ein Segment zu, werden diese Speicherzugriffe in eine Speichertransaktion verschmolzen (engl. *coalesced*), was sich vorteilhaft auf den erzielbaren Datendurchsatz auswirkt (s. Abbildung 4.5a). Umgekehrt werden mehr Speichertransaktionen durchgeführt und so der Datendurchsatz beeinträchtigt, wenn sich die benötigten Daten in vielen unterschiedlichen Segmenten des globalen Speichers vorfinden (s. Abbildung 4.5b). Es ist daher darauf zu achten, dass Threads eines *warps* auf Daten möglichst weniger Segmente zugreifen, Zugriffe sequenzialisiert werden und so die beschriebenen Lokalitätseffekte genutzt werden. Hinzukommt, dass vom Host durchgeführte Speicheraktionen, wie beispielsweise das Kopieren von Nutzdaten aufgrund der Systemarchitektur (vgl. Abbildung 4.1) relativ teure Operationen darstellen und die volle Bandbreite der PCI-E-Verbindung nur dann ausgeschöpft wird, wenn ausreichend große Daten kopiert werden. Für das Speichern der Evaluationsmatrizen ist es unter Berücksichtigung der genannten Aspekte daher erforderlich, dass die Daten gemeinsam gespeichert und in möglichst wenig Operationen auf die GPU übertragen werden.

Um dies zu bewerkstelligen, werden alle Evaluationsmatrizen, die sonst separat auf die GPU geladen werden müssten, auf dem Host in eine einzelne (große) Matrix überführt. Dies minimiert die Anzahl der Kopiervorgänge und sorgt für eine angemessene Auslastung der PCI-E-Verbindung. Anstelle nun aber alle Matrizen lediglich zu konkatenieren, soll unter Berücksichtigung des Algorithmus 4.4 darauf geachtet werden, dass Speicherzugriffe verschmolzen werden können, um die Zahl der Transaktionen zu reduzieren. Um dies zu

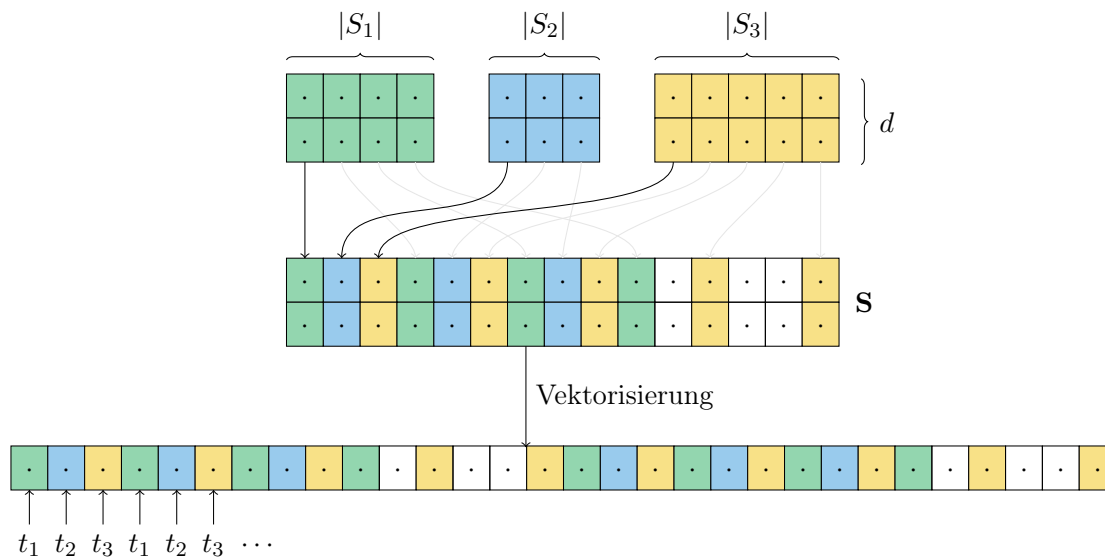


Abbildung 4.6: Drei Evaluationsmatrizen S_1 , S_2 und S_3 mit jeweils vier, drei und fünf Elementen und einer Dimensionalität von $d = 2$ werden in eine einzelne Matrix S überführt. Anschließend wird diese Matrix zeilenweise in einen Vektor überführt und damit *vektoriert*. Die Threads t_1 , t_2 und t_3 sind den jeweiligen Evaluationsmengen oder -matrizen S_1 , S_2 und S_3 zugeordnet und greifen demnach nacheinander auf die Informationen dieses Vektors zu, weswegen die Zugriffe in möglichst wenigen Speichertransaktionen verschmolzen werden.

bewerkstelligen wird ein Blick auf die für den Datenzugriff relevante Zeile 11 des Algorithmus 4.4 geworfen: In dieser Zeile wird die Differenz zwischen einem Grundmengenvektor und einem Evaluationsmengenvektor jeweils am Index k berechnet und anschließend quadriert. Der Zugriff auf den Grundmengenvektor erfolgt über den geteilten Speicher und ist hier ohne Bedeutung. Der Wert des entsprechenden Evaluationsmengenvektors muss aus dem globalen Speicher erfolgen und soll nun optimiert werden. Wie bereits erwähnt, ergibt sich die Möglichkeit der Verschmelzung von Zugriffen auf den globalen Speicher innerhalb eines *warp*: Diese Gruppe von Threads entsteht aus der Partitionierung der Threads eines Blocks und führt stets eine gemeinsame Instruktion aus. Diese Instruktion sei nun das Laden des benötigten Werts $\vec{s}_i[k]$ mit $\vec{s}_i \in S_j$ aus dem Speicher, wobei k und i für jeden Thread im *warp* gleich ist. Zwischen den Threads im Block unterscheidet sich nun lediglich die betrachtete Evaluationsmenge S_j . Um den Verschmelzungseffekt zu erzielen, müssen daher jene Daten der verschiedenen Mengen S_j , für die k (die aktuell betrachtete Dimension) und i (der Index des betrachteten Vektors) gleich sind, sequentiell im Speicher abgelegt werden.

Technisch erzielt wird dies dadurch, dass im Round-Robin-Verfahren eine Evaluationsmenge gewählt wird und in dieser Menge jeweils der nächste noch nicht betrachtete Vektor in die nächste Spalte der Evaluationsmengenmatrix S geschrieben. Ist in einer Evaluationsmenge kein Vektor mehr vorhanden, um diesen in die Matrix S zu schreiben, bleibt der

entsprechende Eintrag schlicht leer. Anschließend wird die Matrix \mathbf{S} zeilenweise vektorisiert. Das Vorgehen ist in Abbildung 4.6 grafisch illustriert. Dieses Verfahren weist zwar den Nachteil auf, dass durch das Freilassen von Feldern in der Evaluationsmatrix bei Mengen unterschiedlicher Länge ein gewisser „Verschnitt“ entsteht, allerdings wird hierdurch auch die Adressierung vereinfacht, da keine variablen Größen von Evaluationsmengen berücksichtigt werden müssen. So ist a-priori bekannt, dass eine Zeile in der Matrix \mathbf{S} (die eine Dimension aller gemeinsamen Vektoren beinhaltet) genau $m = |S_{\text{multi}}| \cdot \max\{|S| \mid S \in S_{\text{multi}}\}$ Einträge umfasst. Da die Matrix zeilenweise vektorisiert wird, müssen zur Adressierung einer individuellen Dimension k somit $k \cdot m$ Einträge im Vektor übersprungen werden. Um innerhalb einer bestimmten Dimension einen Vektor $\vec{s}_i \in S_j$ zu adressieren, müssen zusätzlich $i \cdot |S_{\text{multi}}| + j$ Einträge übersprungen werden.

Gleichzeitig ist aber auch zu erwähnen, dass das Problem des Verschnitts gerade für den GREEDY-Algorithmus, der viele Evaluationsmengen zur Optimierungsentscheidung prüft, nicht existent ist, da alle zu evaluierenden Mengen die gleiche Kardinalität aufweisen. Anders ist es im SIEVE STREAMING-Verfahren, bei dem die Anzahl der Elemente in den jeweiligen Sieben abhängig von dem tatsächlich zugeordneten Grenzwert stark variieren kann, dort allerdings pro neu ankommendem Element im Datenstrom in Abhängigkeit des gewählten Approximationsparameters ϵ aber auch tendenziell weniger Mengen zu prüfen sind.

Kapitel 5

Experimente

Die in den vergangenen Kapiteln etablierten Konzepte sollen nun in einer Reihe von Experimenten empirisch evaluiert werden. Hierbei soll zunächst in Abschnitt 5.1 das Themenfeld der Ausreißerererkennung bearbeitet werden, wobei insbesondere die Frage im Vordergrund steht, ob submodulare Methoden zur Ausreißerererkennung kompetitiv zu klassischen Methoden sind, welche Funktionen für diese Aufgabe adäquate Zusammenfassungen hervorbringen und wie sich die rekursive Partitionierungsstrategie im Vergleich zur einfachen Partitionierungsstrategie auf die Vorhersagegüte auswirkt. In einer zweiten Experimentierreihe, die in Abschnitt 5.2 vorgestellt wird, soll untersucht werden, ob durch die GPU-Implementierung des EXEMPLAR-BASED CLUSTERINGS eine Beschleunigung gegenüber der CPU-Implementierung erzielt werden kann und wie groß diese ausfällt.

Für beide Experimentierreihen wird zunächst der Aufbau der Versuche und die Parameterwahl diskutiert, bevor die Ergebnisse beschrieben und anschließend in einem diskutierenden Abschnitt interpretiert und bewertet werden.

5.1 Ausreißerererkennung

In dieser Experimentierreihe sollen die in den vergangenen Abschnitten vorgestellten Ausreißerererkennungsverfahren evaluiert werden. Konkret sollen diese Verfahren auf eine Reihe von Datensätzen angewendet werden, denen jeweils eine unterschiedliche Erkennungsaufgabe zugrunde liegt. Darauf aufbauend werden passende Metriken gemessen, die die erzielbare Vorhersageleistung in der Ausreißerererkennung feststellen. Die zentrale Frage hierbei ist, ob es eine eindeutige Präferenz hinsichtlich eines Verfahrens oder hinsichtlich einer bestimmten Parametrisierung gibt. Insbesondere soll festgestellt werden, ob eine bestimmte submodulare Funktion mit bestimmten Parametern (beispielsweise mit einer bestimmten Kernfunktion) tendenziell eher dazu geeignet ist, um eine Ausreißerererkennung zu bewerkstelligen. Dies führt im weiteren Sinne auf die Frage, ob eine submodulare Funktion für diese Aufgabe bessere *Zusammenfassungen* eines Datensatzes liefert. Des Weiteren sollen in

Abhängigkeit der konkret untersuchten Eigenschaften Hypothesen aufgestellt werden, für die in Abschnitt 5.1.2 geeignete Daten präsentiert werden sollen, die dann zur Diskussion der Hypothese in Abschnitt 5.1.3 verwendet werden.

Um die empirische Evaluation der Ausreißerererkennung und die Validierung der Hypothesen durchzuführen, wird in Abschnitt 5.1.1 der Aufbau dieser Experimentierreihe dargelegt. In Abschnitt 5.1.2 werden dann die erzielten Ergebnisse präsentiert, um diese in Abschnitt 5.1.3 hinsichtlich der oben präsentierten Fragestellungen zu diskutieren.

5.1.1 Aufbau

In diesem Abschnitt soll der Aufbau der Experimentierreihe zur Ausreißerererkennung beschrieben werden. Hierzu werden zunächst die in den Experimenten zu verwendenden Datensätze charakterisiert. Anschließend soll besprochen werden, welche Algorithmen zur Ausreißerererkennung verwendet werden und wie die Parameter für diese Verfahren gewählt beziehungsweise optimiert werden. Zuletzt sollen Metriken präsentiert werden, um das Ausreißerererkennungsergebnis zu bewerten.

Datensätze

Verwendet werden verschiedene Datensätze, die in einem jeweils konkreten Anwendungsfall die Erkennung von Ausreißern verlangen. Diese Datensätze sind in Tabelle 5.1 abgebildet und stammen zum großen Teil aus der Arbeit von Campos *et al.* [10] und wurden auf der dazugehörigen Webseite¹ verfügbar gemacht. Bei jedem dieser Datensätzen wurde stets auf die normalisierte Version zurückgegriffen, die um Duplikate bereinigt wurde und höchstens 2% Ausreißer enthält. Die Attribute jener Datensätze, die nicht von Campos *et al.* zusammengestellt wurden, wurden auf das Intervall $[0, 1]$ skaliert, bevor diese für die Ausreißerererkennung verwendet wurden.

Für jeden dieser Datensätze stehen Labelinformationen zur Verfügung, die jedoch nicht für das Anwenden der vorgestellten Methoden Verwendung finden sondern lediglich zum Messen der Vorhersagequalität eingesetzt werden. Diese Label sind dabei binärer Natur, geben also nicht an wie „stark“ eine Beobachtung ausreißt sondern nur, ob dies der Fall ist oder nicht (also $\mathbb{B} = \{+1, -1\}$, s. Abschnitt 3.2).

Konfiguration

Um einen angemessenen Vergleich zwischen den Methoden zu gewährleisten, muss für jeden zu prüfenden Algorithmus eine Parametersuche oder zumindest eine geschickte Parameterwahl erfolgen. In diesem Abschnitt soll daher kurz dargelegt werden, welche Im-

¹<https://www.dbs.ifi.lmu.de/research/outlier-evaluation/DAMI/> (zuletzt zugegriffen am 05.03.2020)

#	Name	N	d	N^{-1}	Aufgabe
1	Annthyroid	7200	21	534	Schilddrüsenfunktionsklassifizierung
2	Cardiotocography	2126	21	471	Herzfrequenzklassifikation
3	Pageblocks	5473	10	560	Dokumentenanalyse
4	Pendigits	9868	16	20	Bildklassifikation
5	Spambase	4601	57	1813	Klassifikation von Spam
6	Waveform	3443	21	100	Synthetische Daten
7	Mammography	11183	6	260	Identifikation von Brustkrebs

Tabelle 5.1: Die zur Ausreißerererkennung verwendeten Datensätze, charakterisiert anhand ihres Namens, der Anzahl der Beobachtungen im Datensatz N , der Anzahl vorkommender Dimensionen d , der Anzahl vorkommender Ausreißer N^{-1} und der Aufgabe. Die Datensätze 1 - 6 entstammen Campos *et al.* [10]. Der Datensatz *Mammography* wurde von Liu *et al.* [40] verwendet.

plementierung eines Verfahrens genutzt wurde und wie zu wählende Parameter bestimmt beziehungsweise optimiert wurden.

Isolation Forest Zur Ausreißerererkennung mittels des ISOLATION FORESTS wird die Implementierung genutzt, die durch die `scikit-learn`-Bibliothek zur Verfügung gestellt wird [47]. Die zu wählenden Parameter sind hierbei die Anzahl zu trainierender Bäume t sowie die Stichprobengröße ψ . Zusätzlich wird die durch die Implementierung offerierte Möglichkeit genutzt, die Zahl der nutzbaren Merkmale ϕ pro Isolation Tree zu limitieren. Sei N nun die Anzahl der Beobachtungen in einem Datensatz und d die Anzahl der Dimensionen, dann wird zur Evaluation des Isolation Forests das Kreuzprodukt der Parameterkombinationen aus $t \in \{2^7, 2^8, 2^9, 2^{10}\}$, $\psi \in \{\min(2^8, N)\}$, $\phi \in \{\sqrt{d}\}$ verwendet.

Local Outlier Factor Die Ausreißerererkennung mittels des LOCAL OUTLIER FACTORS wird ebenfalls durch die Implementierung durchgeführt, die durch das `scikit-learn`-Paket bereitgestellt wird. Um das Verfahren anwenden zu können, muss insbesondere die zu betrachtende Nachbarschaftsgröße $minPts$ gewählt werden. Die zu evaluierenden Nachbarschaftsgrößen seien $minPts \in \{5, 10, 20, 50, 100\}$. Die verwendete Distanzfunktion ist die euklidische Distanz (vgl. Definition 4.2.1).

Summary Extraction Die in Abschnitt 3.5.1 vorgestellte SUMMARY OUTLIER DETECTION sowie die in Abschnitt 3.5.2 vorgestellte Erweiterung mit RSP-Bäumen verlangen (intuitiv) die Extraktion einer Zusammenfassung über dem gegebenen Datensatz beziehungsweise (abstrakt) die Lösung des Optimierungsproblems 2.3. Hierzu wurden in Abschnitt 2.2 zwei Optimierungsstrategien vorgestellt, nämlich der GREEDY-Algorithmus sowie das SIEVE STREAMING. Für die Evaluation der Ausreißererkennungsperformanz soll auf GREEDY und SIEVE STREAMING mit $\epsilon = 0.1$ zurückgegriffen werden. Da das SIEVE STREAMING-

Verfahren eigentlich ein Datenstromverfahren ist, welches einen potenziell unendlichen Datenstrom betrachtet, kann unter Umständen nach einmaliger Iteration über den Datensatz kein Sieb existieren, das mit k Elementen gefüllt ist, was nach Definition 2.1.3 zu schlechteren Funktionswerten führen kann. Es werden daher bis zu zwanzig Iterationen über den Datensatz gestattet und somit eine Erweiterung des Ansatzes von Badanidiyuru *et al.* betrachtet. Diese Iterationen werden solange durchgeführt, bis alle geöffneten Siebe k Elemente aufweisen, diese also *geschlossen* sind. Sollte nach diesen Iterationen immer noch kein geschlossenes Sieb vorhanden sein, wird das beste, geöffnete Sieb zurückgegeben. Aus diesem Grund wird das SIEVE STREAMING-Verfahren auch ausschließlich im Zusammenhang mit der SUMMARY OUTLIER DETECTION *ohne* RSP-Bäume durchgeführt, da sich die Größe des Datensatzes mit zunehmender Baumtiefe pro Baumknoten stark verringert.

Der Einfluss der mitunter hohen Anzahl der Funktionsauswertungen (vgl. Abschnitt 2.2.1) wird durch eine effiziente beziehungsweise eine durch moderne Hardware unterstützte Implementierung möglichst gering gehalten (s. Kapitel 4). Zur Summary Extraction werden die submodularen Funktionen der INFORMATIVE VECTOR MACHINE (IVM) und des EXEMPLAR-BASED CLUSTERINGS eingesetzt.

Um die IVM auszuwerten wird eine Kernmatrix benötigt, die durch einen positiv-definiten Kernel induziert wird. Da Kernel beziehungsweise Kernfunktionen über das Skalarprodukt und die implizite Projektion in einen Merkmalsraum ein individuelles Verständnis von Ähnlichkeit zwischen Beobachtungen realisieren, gilt es diesen Parameter für die Ausreißerererkennung zu optimieren. Dabei ist insbesondere zu vermeiden, dass zwar die gemessene Repräsentativität maximiert aber effektiv bedeutungslose Repräsentanten identifiziert werden (vgl. Abschnitt 2.3.1).

Für die experimentelle Evaluation soll zunächst der RBF-Kernel verwendet werden, der formal wie folgt etabliert wird:

5.1.1 Definition (RBF-Kernel [62]). Seien $\vec{x}, \vec{y} \in X$ zwei Vektoren aus einem Datenraum X mit Skalarprodukt $\langle \cdot, \cdot \rangle$ sowie einem Skalierungsparameter $\sigma > 0$, dann ist

$$k_{\text{RBF}} = \exp\left(-\frac{d_{\text{Euklid}}(\vec{x}, \vec{y})^2}{2\sigma^2}\right) \quad (5.1)$$

der RBF-Kernel. Dieser Kernel ist positiv-definit.

Ferner soll ebenfalls der Matérn-Kernel evaluiert werden. Dieser Kernel lässt sich in einer generalisierten Form darstellen, der unter anderem von einem Parameter $\nu > 0$ abhängt. Wird dieser Parameter beliebig gewählt, muss der Kernel unter anderem mittels der modifizierten Besselfunktion teuer berechnet werden. *Rasmussen* und *Williams* geben aber Vereinfachungen an, die die Komplexität des Kerns für feste $\nu \in \{\frac{1}{2}, \frac{3}{2}, \frac{5}{2}\}$ deutlich reduzieren und außerdem für Anwendungen im maschinellen Lernen den Autoren zufolge

womöglich am interessantesten sind [48]. Diese speziellen Matérn-Kernel sollen nun wie folgt definiert werden:

5.1.2 Definition (Matérn-Kernel [48]). Sei $r = d(\vec{x}, \vec{y})$ die Distanz zwischen zwei Vektoren $\vec{x}, \vec{y} \in X$ in einem Datenraum X , dann ist der Matérn-Kernel $k_{\text{Matérn}}$ für $\nu \in \{\frac{1}{2}, \frac{3}{2}, \frac{5}{2}\}$ und einem Skalierungsparameter $l > 0$ wie folgt definiert:

$$k_{\text{Matérn}}^{\nu=1/2} = \exp\left(-\frac{r}{l}\right) \quad (5.2)$$

$$k_{\text{Matérn}}^{\nu=3/2} = \left(1 + \frac{\sqrt{3}r}{l}\right) \exp\left(-\frac{\sqrt{3}r}{l}\right) \quad (5.3)$$

$$k_{\text{Matérn}}^{\nu=5/2} = \left(1 + \frac{\sqrt{5}r}{l} + \frac{5r^2}{3l^2}\right) \exp\left(-\frac{\sqrt{5}r}{l}\right) \quad (5.4)$$

Der Matérn-Kernel ist positiv-definit [12].

Zur konkreten Evaluation der IVM mit dem Matérn-Kernel wird die euklidische Distanzfunktion d_{Euklid} verwendet.

Wie in Abschnitt 2.3.1 dargelegt, müssen eventuelle Parameter eines Kernels sorgsam gewählt werden, damit eine für den Anwendungsfall der Ausreißererkenung tatsächlich bedeutsame repräsentative Zusammenfassung der Daten gefunden wird. Da sowohl der RBF- als auch der Matérn-Kernel einen Skalierungsparameter beinhalten, der a-priori gewählt werden muss, soll dieser deshalb optimiert werden, wofür die Menge $\{\frac{0.01}{\sqrt{d}}, \frac{0.5}{\sqrt{d}}, \frac{0.75}{\sqrt{d}}, \frac{1.0}{\sqrt{d}}, \frac{2.0}{\sqrt{d}}, \frac{5.0}{\sqrt{d}}, \frac{7.5}{\sqrt{d}}, \frac{10.0}{\sqrt{d}}, \frac{12.5}{\sqrt{d}}, \frac{15.0}{\sqrt{d}}\}$ bei gegebener Dimensionalität des Datensatzes d herangezogen werden soll. Des Weiteren wird abweichend von Definition 5.1.2 eine zur `scikit-learn`-Bibliothek kompatible Implementierung des Matérn-Kernels verwendet, bei der nicht die Distanz r skaliert wird, sondern die Dimensionen der eingegebenen Vektoren \vec{x} und \vec{y} individuell mittels eines Skalierungsvektors \vec{l} skaliert werden, sodass effektiv jeder Eintrag eines zu skalierenden Vektors \vec{a}_i durch den korrespondierenden Eintrag im Skalierungsvektor \vec{l}_i dividiert wird. Für die hier durchgeführten Experimente wird der Skalierungsvektor \vec{l} mit den zu optimierenden Skalierungsparametern gefüllt.

Für die Evaluation der submodularen Funktion des EXEMPLAR-BASED CLUSTERINGS wird lediglich die Spezifikation einer Distanzfunktion erfordert (vgl. Abschnitt 2.3.2). Diese sei hiermit auf die quadratische, euklidische Distanz festgelegt.

Darüber hinaus soll neben den Optimierungsverfahren GREEDY und SIEVE STREAMING, die eine submodulare Funktion zur Repräsentantenwahl optimieren, ebenfalls eine reine Zufallsauswahl von Beobachtungen der Grundmenge erfolgen, um festzustellen, wie sich eine zufällige und damit unbedeutsame Auswahl von Repräsentanten auf die Vorhersagequalität auswirkt. Ein gegebenes Experiment, welches auf der Zufallsauswahl von Repräsentanten beruht, wird dabei fünf mal wiederholt, sodass eine möglichst robuste Beurteilung der erzielten Metriken und Funktionswerte möglich ist.

Ferner gilt es noch zur Lösung des Problems 2.3 festzulegen, wie viele Punkte bei der Bildung der Zusammenfassung berücksichtigt werden sollen. Da a-priori unklar ist, wie viele Punkte zu einer für die Ausreißerererkennung bedeutungsvollen Zusammenfassung führen, soll dieser Parameter ebenfalls optimiert werden und hierfür die Menge $k \in \{10, 20, \dots, 100\}$ abgetastet werden. Die SUMMARY OUTLIER DETECTION mit RSP-Bäumen betrachtet hiervon aufgrund des exponentiellen Wachstums gefundener Regionen abweichend die Menge $k \in \{2, 3, \dots, 15\}$.

Summary Outlier Detection Um die in Abschnitt 3.5.1 beschriebene Ausreißerererkennung durchzuführen, wird eine innerhalb dieser Arbeit in der Programmiersprache C++ entwickelte Implementierung genutzt. Die hierfür benötigten Zusammenfassung werden methodisch derart extrahiert, wie es im Abschnitt zuvor beschrieben wurde und somit auch die dort dargelegten Optimierungsstrategien angewandt. Ferner wird der Grenzwert für die p -Wertunterschreitung auf $p_{\text{thr}} = 0,05$ und die zur Partitionsbestimmung benötigte Distanzfunktion auf die quadratische, euklidische Distanz festgelegt.

Recursive Submodular Partitioning Die Ausreißerererkennung mit RSP-Bäumen stellt eine Erweiterung der SUMMARY OUTLIER DETECTION dar, womit die dort beschriebene Konfiguration und Parametrisierung hier ebenfalls Anwendung finden soll. Zusätzlich zu spezifizieren ist jedoch die Abbruchbedingung bei der Induktion des Baums. Diese seien die Streuung der Partition, der mittlere Kernfunktionswert und die maximale Baumhöhe (s. Definition 3.5.2). Die Abbruchbedingung „Baumhöhe“ wird mit verschiedenen Baumhöhen $t_h \in \{1, \dots, 5\}$ evaluiert. Des Weiteren sind die jeweiligen Grenzwerte t_d und t_k adäquat zu wählen. Diese lassen sich jedoch in Abhängigkeit des Datensatzes und der konkret verwendeten Funktion nur schwer a-priori bestimmen, weswegen an dieser Stelle ebenfalls eine Optimierung durchgeführt werden soll. Betrachtet wird daher die für die Berechnung der Abbruchbedingung erforderliche Funktion $e : X \times X \rightarrow \mathbb{R}$, wobei für den gesamten zu untersuchenden Datensatz \mathcal{D}_{UL} nun sowohl der kleinste vorkommende Wert $e_{\min} = \min \{e(\vec{x}, \vec{y}) \mid \forall \vec{x}, \vec{y} \in \mathcal{D}_{\text{UL}}\}$ als auch der größte vorkommende Wert $e_{\max} = \max \{e(\vec{x}, \vec{y}) \mid \forall \vec{x}, \vec{y} \in \mathcal{D}_{\text{UL}}\}$ berechnet wird. Das durch die beiden Werte aufgespannte Intervall $[e_{\min}, e_{\max}]$ wird gleichförmig abgetastet, wobei fünf Schritte vorgenommen werden. Da die Berechnung von e_{\min} und e_{\max} jeweils eine Zeitkomplexität von $\mathcal{O}(|\mathcal{D}_{\text{UL}}|^2)$ aufweisen und dies gerade für große Datensätze teuer werden kann, wird jeder Datensatz auf höchstens 1000 Punkte reduziert, wobei die Punkte aus dem Ursprungsdatensatz mittels einer simulierten Gleichverteilung ohne Zurücklegen bereits gezogener Beobachtungen gewählt werden. Für die Berechnung der Abbruchbedingung „Streuung der Partition“ wird die euklidische Distanz verwendet. Die Abbruchbedingung „Mittlerer Kernfunktionswert“ verwendet die Kernfunktion der IVM. Sofern die submodulare Funktion des EXEMPLAR-BASED CLUSTERINGS verwendet wird, wird standardmäßig der RBF-Kernel

verwendet, dessen Skalierungsparameter σ , wie im obigen Abschnitt dargelegt, optimiert wird.

Metriken

Jedes der zu evaluierenden Ausreißererkennungsverfahren liefert nun einen Vektor $\vec{y}_p \in \mathbb{R}^n$, der die Bewertung jeder einzelnen Beobachtung im eingegebenen Datensatz \mathcal{D}_{UL} mit $n = |\mathcal{D}_{UL}|$ enthält. Um festzustellen, wie gut die Ausreißererkennung einer Methode auf einem gegebenen Datensatz operiert, muss dieser vorhergesagte Vektor \vec{y}_p mit den „wahren“ Labelinformationen $\vec{y}_t \in \{+, -\}^n$ verglichen werden. Intuitiv wird bei der Messung der Klassifikationsleistung üblicherweise verlangt, dass die getroffene Vorhersage möglichst nah an der durch den Datensatz präsentierten Wahrheit liegt, wobei diese „Nähe“ zu definieren ist. Schwierigkeiten ergeben sich beispielsweise dadurch, dass die *Wahrheit* durch den Datensatz in der Regel durch einen Binärvektor angezeigt wird, der unter Umständen mit einer reellen Bewertung durch den Prädiktor, also hier der entsprechenden Methode zur Ausreißererkennung, zu vergleichen ist. Eine Lösung hierfür kann die Betrachtung von Schwellwertüber- oder unterschreitungen darstellen, der den reellen Bewertungsvektor binarisiert, wobei durch den Schwellwert jedoch effektiv ein zusätzlicher Hyperparameter eingeführt wird. Doch selbst wenn ausschließlich binäre Vektoren miteinander verglichen werden, muss auf die Wahl einer adäquaten Metrik geachtet werden, damit gerade die für den Anwendungsfall der Ausreißererkennung kritischen Beobachtungen gemessen werden.

Um die Bewertung der Vorhersageleistung zu bewerkstelligen, wird im weiteren Verlauf dieser Experimentierreihe auf zwei Metriken, nämlich die ROC-AUC und den F1-Score, zurückgegriffen. Es soll betont werden, dass im Rahmen dieser Arbeit eine rein qualitative Bewertung der Metriken erfolgen soll, da eine quantitativ-interpretierende Analyse der Vorhersagegüten eine genaue Kenntnis des zu untersuchenden Anwendungsfalls erfordern würde, die hier nicht vorhanden ist.

ROC-AUC Zur Erklärung dieser Metrik soll auf die Ausführungen von *Fawcett* zurückgegriffen werden [18], der zunächst die sogenannten *Receiver Operating Characteristics* (ROC)-Plots erläutert. Diese stellen zweidimensionale Plots dar, wobei die Richtig-positiv-Rate (engl. *true positive rate*, *TP rate*) auf der x -Achse und die Falsch-positiv-Rate (engl. *false positive rate*, *FP rate*) auf der y -Achse eingezeichnet werden. Grundlage für die Berechnung dieser Raten ist die sogenannte Konfusionsmatrix, die in Tabelle 5.2 aufgeführt ist.

Wahrheit Vorhersage	Positiv (P)	Negativ (N)
Positiv	Richtig-positiv (TP)	Falsch-positiv (FP)
Negativ	Falsch-negativ (FN)	Richtig-negativ (TN)

Tabelle 5.2: Konfusionsmatrix bei binärer Bewertung von Beobachtungen [18].

Die Schätzung der TP- und FP-Rate erfolgt dabei (ohne formale Definition) durch die Berechnung der beiden nachfolgenden Ausdrücke:

$$\text{TP rate} \approx \frac{TP}{P} \quad (5.5)$$

$$\text{FP rate} \approx \frac{FP}{N} \quad (5.6)$$

Für den Fall, dass ein Ausreißererkenntungsverfahren mit binären Bewertungen arbeitet, können diese beiden Raten gegeben eines Datensatzes ermittelt werden und als Punktepaar in den ROC-Plot eingezeichnet werden. Werden verschiedene Punktepaare für verschiedene Verfahren ermittelt, kann durch visuelle Inspektion dieses Plots festgestellt werden, welche Verfahren besser sind als andere: So gilt ein Punkt gegenüber einem anderen Punkt als besser, wenn die TP-Rate höher oder die FP-Rate niedriger ist.

Werden nun statt binärer Bewertungen auch reelle Bewertungsräume betrachtet, führt dies auf das mächtigere Konzept der ROC-AUC (engl. *area under curve*). Für jedes Verfahren, welches einen reellen Vektor mit Bewertungen produziert, kann dieser nun durch die Wahl eines Schwellwerts in einen binären Vektor überführt werden. Für verschiedene Schwellwerte können dabei jeweils verschiedene Punktepaare, wie oben beschrieben, produziert und so eine ROC-Kurve gefunden werden. Wird die Fläche gemessen, die unter dieser ROC-Kurve liegt, führt dies auf die Metrik der ROC-AUC. Höhere Werte für die ROC-AUC sind dabei wünschenswerter, da dies auf eine bessere durchschnittliche Klassifikations- oder Erkennungsleistung (gemessen an der TP- und FP-Rate) der zugrundeliegenden Methode hindeutet, wobei der resultierende Wert grundsätzlich im Intervall $[0, 1]$ liegt. Die Messung dieser Fläche ist eine nicht-triviale Operation, die an dieser Stelle durch die `scikit-learn`-Bibliothek [47] unterstützt wird.

Diese Metrik ist in jenen Situationen vorteilhaft, bei denen der Erfolg eines Verfahrens von der adäquaten Wahl eines Schwellwerts abhängt. Durch die Inspektion des ROC-Plots lässt sich zum einen feststellen, ob und, wenn zutreffend, welche Schwellwerte das gegebene Problem angemessen lösen. Unabhängig davon lassen sich auch ohne manuelle Inspektion der Kurve mittels der ROC-AUC-Metrik feststellen, ob im Mittel adäquate Grenzwerte existieren. Dies kann in einer Bewertung auch als Charakteristikum hierfür dienen, wie „sensibel“ ein Verfahren gegenüber der Grenzwertwahl ist. Existiert beispielsweise im ROC-Plot ein Grenzwert, der das Problem angemessen löst, während alle anderen Grenzwerte

ungenügende Ergebnisse liefern, dann darf das Verfahren als sensibel gegenüber der Grenzwahl gelten.

F_1 -Score Der ROC-AUC-Score verlangt, dass die Bewertung jeder individuellen Beobachtung durch einen reellen Wert erfolgt. Ein Ansatz, der mit Binärbewertungen arbeitet, ist der sogenannte F_1 -Score. Grundlage für diesen Score ist die sogenannte Präzision P und der Recall R [17], die durch die Ausdrücke

$$P = \frac{TP}{TP + FP} \quad (5.7)$$

$$R = \frac{TP}{TP + FN} \quad (5.8)$$

berechnet werden. Beide Metriken bewegen sich hierbei im Wertebereich $[0, 1]$, wobei jeweils eine Präzision beziehungsweise ein Recall von 100% wünschenswert ist. Die genannten Ausdrücke messen die Erkennungsperformance dabei anhand verschiedener Faktoren: Wird die Anzahl richtig-positiver Vorhersagen fixiert, dann verschlechtert sich die Präzision lediglich in jenen Fällen, in denen mehr falsch-positive Vorhersagen getroffen werden. Hinsichtlich des Recalls ist die Situation ähnlich, wohingegen nicht die falsch-positiven Vorhersagen die Metrik verschlechtern, sondern die falsch-negativen Vorhersagen. Da Ausreißer in der Lernaufgabe der Ausreißerererkennung typischerweise deutlich unterrepräsentiert sind und je nach praktischem Anwendungsfall eine hohe Bedeutung zukommen, ist man gerade daran interessiert, diese ausreißenden Beobachtung zu identifizieren, was auch entsprechend gemessen werden soll. Daher wird im Sinne der obigen Metriken die Ausreißerklasse als „positive“ Klasse geführt. Würde man nun lediglich Präzision zur Messung der Erkennungsleistung heranziehen, dann würden gerade die falsch-negativ erkannten Beobachtungen vernachlässigt werden, also jene, die eigentlich ausreißend sind aber vom Verfahren nicht als solche erkannt werden. Andersherum ist es beim Recall: Dort würden gerade die falsch-positiv erkannten Beobachtungen ohne Betrachtung bleiben, also jene Fälle, in denen Beobachtungen eigentlich keine Ausreißer sind, aber als solche erkannt werden.

Dieses Ungleichgewicht zwischen den Metriken versucht der F_1 -Score zu beheben, indem das harmonische Mittel zwischen den beiden Metriken Precision und Recall gebildet wird [17]. Dieser ist dann durch folgenden Ausdruck gegeben:

$$F_1 = 2 \frac{PR}{P + R} \quad (5.9)$$

Postprocessing

Aufgrund der großen Anzahl an durchgeführten Experimenten ist es notwendig, dass die erzielten Ergebnisse geeignet nachbearbeitet werden, sodass erkennbar ist, welche Verfahren mit welchen Parametern die jeweils besten Resultate erzielen. Zu berücksichtigen ist hierbei einerseits, dass beispielsweise beim Arbeitsschritt der *Summary Extraction* mit einer Zufallsauswahl mehrere Läufe durchgeführt wurden, dass einige Datensätze anhand

Datensatz	Run	Split	Methode	f	$f(S)$	Abbruch	$t_d/t_k/t_h$	k	Kernel	Sigma	$minPts$	t	Laufzeit [s]	ROC-AUC	F1-Score
Waveform	2	v06	Random (SOD-RSP)	IVM	—	—	t_d	0.5424	13	$k_{\text{Matérn}}^{\nu=5/2}$	1.0910	—	0.2293	0.6442	0.1589
SpamBase	—	v04	Isolation Forest	—	—	—	—	—	—	—	—	512	0.6121	0.7824	0.0000
PenDigits	3	v04	Random (SOD-RSP)	EBC	—	—	t_k	0.25	11	—	—	—	8.3721	0.5659	0.0000
Cardiotoco.	—	v10	Local Outlier Factor	—	—	—	—	—	—	—	—	5	0.0542	0.7412	0.2150
Cardiotoco.	2	v02	Random (SOD)	IVM	13.583	—	—	—	40	k_{TRBF}	0.2182	—	0.0130	0.5644	0.0315
Anthyroid	3	v01	Random (SOD-RSP)	EBC	—	—	t_k	0.2842	10	—	—	—	0.0292	0.4182	0.0169
PageBlocks	—	v04	Greedy (SOD)	IVM	7.440	—	—	—	70	$k_{\text{Matérn}}^{\nu=3/2}$	1.5811	—	12.9199	0.7937	0.2625
PenDigits	4	v01	Random (SOD-RSP)	IVM	—	—	t_h	2	13	$k_{\text{Matérn}}^{\nu=5/2}$	0.2500	—	0.0659	0.3969	0.0008
...

Tabelle 5.3: Ausschnitt aus der resultierenden Ergebnistabelle für die im vergangenen Abschnitt beschriebenen Experimente. Jede hier dargestellte Zeile stellt eine durchgeführte Experimentkonfiguration dar, die hinsichtlich eines Datensatzes beziehungsweise (falls vorhanden) des dazugehörigen *Splits* evaluiert wurde. Aufgrund der immensen Größe dieser Tabelle muss zum Zwecke der Ergebnisdiskussion eine adäquate Aggregations- beziehungsweise Selektionsstrategie gefunden werden.

verschiedener Aufteilungen (engl. *splits*) präsentiert werden und dass zwei verschiedene Metriken definiert wurden, die unabhängig voneinander zu optimieren sind.

Die in dieser Arbeit verwendete Strategie reduziert zunächst die ermittelte Laufzeit, den ROC-AUC-Wert, den F1-Score und den Funktionswert wiederholter Ausführungen der gleichen Experimentkonfiguration, wie es hier ausschließlich bei der Zufallsauswahl der Fall ist, mittels des arithmetischen Mittels. Anschließend wird je nach betrachteter Metrik unter allen Experimenten einer Ausreißererennungsmethode, die für den gleichen Datensatz und den gleichen Split durchgeführt wurden und (gegebenenfalls) die gleiche submodulare Funktion, die gleiche Abbruchbedingung (bei Verwendung von RSP-Bäumen) und die gleiche Kardinalitätsbeschränkung k verwenden, jeweils die Experimentkonfiguration selektiert, die die beste Metrik hervorgebracht hat.

In der resultierenden Tabelle werden dann die Laufzeiten, der ROC-AUC-Wert, der F1-Score und der Funktionswert für alle identischen Experimentkonfigurationen, die verschiedene Splits eines Datensatzes betreffen, arithmetisch gemittelt. Darauffolgend wird für jede jeweils betrachtete Metrik unter allen Experimenten, die den gleichen Datensatz, die gleiche Ausreißererennungsmethode und (gegebenenfalls) die gleiche submodulare Funktion verwenden, jeweils die Konfiguration gesucht, die besagte Metrik maximiert.

Das Ergebnis dieser Nachbearbeitungsprozedur wird in den nachfolgenden Abschnitten präsentiert und diskutiert.

5.1.2 Ergebnisse

In diesem Abschnitt sollen die Ergebnisse der im vorgegangenen Abschnitt vorgestellten Experimentierreihe diskutiert werden. Wie bereits erläutert, wurden für die Datensätze eine hohe Anzahl an Experimenten durchgeführt, um einen angemessenen Vergleich zwischen den Methoden zu ermöglichen und die optimierbaren Parameter entsprechend zu würdigen. Da auch die aggregierte Ergebnistabelle zu umfangreich ist, um sie an dieser Stelle

vorzustellen, sollen im folgenden für die Arbeit eine Reihe interessanter Aspekte vorgestellt werden, die aus dieser Aggregation gewonnen wurden. Die aggregierten Ergebnistabellen, die entsprechend nach den F1- und den ROC-AUC-Metriken generiert wurden, finden sich für den interessierten Leser im Anhang A.0.1 und A.0.2 der Arbeit.

Vergleich verschiedener submodularer Funktionen

Zunächst soll ein Blick auf die Ausreißerererkennung mit submodularen Methoden geworfen und verglichen werden, wie sich die Wahl der Funktion auf die Vorhersageleistung, die jeweils durch den F1-Score und die ROC-AUC gemessen wird, auswirkt. Hierfür wird die Differenz zwischen den besten erzielten Vorhersageleistungen bei Verwendung der INFORMATIVE VECTOR MACHINE und des EXEMPLAR-BASED CLUSTERINGS gebildet, wobei jeweils zwischen dem Datensatz, dem submodularen Erkennungsverfahren und dem Optimierer differenziert wird. Das Ergebnis dieser Betrachtung ist in Tabelle 5.4 abgebildet.

Datensatz	$\Delta_{\text{EBC-IVM}}$ (F1-Score)					$\Delta_{\text{EBC-IVM}}$ (ROC-AUC)				
	Greedy		Random		SieveStr.	Greedy		Random		SieveStr.
	SOD	SOD-RSP	SOD	SOD-RSP	SOD	SOD	SOD-RSP	SOD	SOD-RSP	SOD
Anthyroid	-0.03	-0.06	-0.02	0.08	-0.01	-0.08	-0.09	-0.08	-0.03	-0.04
Cardiotocography	-0.17	-0.17	-0.06	0.00	-0.09	-0.13	-0.12	-0.12	-0.01	-0.07
Mammography	-0.09	-0.06	-0.05	-0.02	-0.29	0.03	0.00	-0.12	-0.02	-0.18
PageBlocks	-0.22	-0.05	-0.04	-0.03	-0.26	-0.02	-0.09	-0.15	-0.03	-0.14
PenDigits	-0.04	-0.06	-0.01	-0.10	-0.02	-0.10	0.00	-0.19	0.00	0.00
Spambase	-0.25	-0.36	-0.03	0.00	-0.18	-0.27	-0.25	-0.08	-0.02	-0.19
Waveform	0.00	-0.17	-0.05	0.01	-0.08	-0.24	-0.05	-0.11	0.00	-0.09

Tabelle 5.4: Vergleich der erzielten Vorhersagegüten pro Datensatz und pro zugrundeliegendem Ausreißerererkennungsverfahren und Funktionsoptimierer. Die Tabelle gibt jeweils die Differenz zwischen den erzielten Metriken bei Verwendung des EXEMPLAR-BASED CLUSTERINGS (EBC) und der INFORMATIVE VECTOR MACHINE (IVM) an, wobei ein negativer Wert eine Verschlechterung bei Nutzung des exemplarbasierten Clusterings anzeigt. Die fett markierten Einträge kennzeichnen eine Verbesserung der Vorhersagegüte bei Nutzung der submodularen Funktion des EXEMPLAR-BASED CLUSTERINGS.

Anhand der Tabelle lässt sich beobachten, dass das EXEMPLAR-BASED CLUSTERING, so wie es hier eingesetzt wurde, in den meisten Fällen die Vorhersageleistung gegenüber der INFORMATIVE VECTOR MACHINE nicht verbessert. Eine nähere Betrachtung der erzielten Ergebnisse lässt sich zunächst anhand der Optimierer beziehungsweise des dazugehörigen Erkennungsverfahrens durchführen: So konnte hinsichtlich des F1-Scores weder bei Nutzung des GREEDY-Optimierers noch beim SIEVE STREAMING-Verfahren, unabhängig vom submodularen Ausreißerererkennungsverfahren, eine Verbesserung durch das EXEMPLAR-BASED CLUSTERING erzielt werden. Lediglich bei der Zufallsauswahl, die die Funktion für die Optimierung *nicht* berücksichtigt, konnten Verbesserungen beobachtet werden, was zu klären ist. Wendet man sich der ROC-AUC-Metrik zu, so konnte lediglich bei Nutzung des

GREEDY-Optimierers und der gewöhnlichen SUMMARY OUTLIER DETECTION ohne RSP-Bäume bei genau einem Datensatz eine Verbesserung der Metrik um 0,03 Punkte erreicht werden.

Ausgehend von den Datensätzen lässt sich feststellen, dass für die Datensätze *Cardiotocography*, *PageBlocks*, *PenDigits* und *Spambase* in keiner der getesteten Varianten aus Optimierungsverfahren und Ausreißererkenntnismethodik eine Verbesserung der Metriken F1-Score und ROC-AUC durch das EXEMPLAR-BASED CLUSTERING erreicht wurde. Für die Datensätze *Anthyroid* und *Waveform* konnte eine Verbesserung bei Nutzung einer Zufallswahl beobachtet werden, die es ebenfalls zu klären gilt. Hinsichtlich des Datensatzes *Mammography* konnte die bereits oben beschriebene Verbesserung um 0,03 Punkte der ROC-AUC bei Nutzung des GREEDY-Optimierers und der SUMMARY OUTLIER DETECTION durch das EXEMPLAR-BASED CLUSTERING erreicht werden.

Vergleich verschiedener submodularer Ausreißererkenntnisverfahren

Als nächstes soll der Einfluss der zusätzlichen Nutzung von RSP-Bäumen bei der SUMMARY OUTLIER DETECTION (SOD) untersucht werden. Hierfür wird für jede Kombination aus Datensatz, Optimierungsverfahren und submodularer Funktion, die jeweils beste Vorhersagegüte ermittelt, die bei gewöhnlicher SOD und bei SOD-RSP erzielt werden konnte und die Differenz zwischen diesen beiden Güten gebildet. Diese wird so berechnet, dass eine positive Differenz eine Verbesserung der Metrik durch SOD-RSP anzeigt. Das Ergebnis dieser Berechnung ist in Tabelle 5.5 abgebildet.

Optimierer Funktion	$\Delta_{\text{SOD-RSP} - \text{SOD}}$ (F1-Score)				$\Delta_{\text{SOD-RSP} - \text{SOD}}$ (ROC-AUC)			
	Greedy		Random		Greedy		Random	
	EBC	IVM	EBC	IVM	EBC	IVM	EBC	IVM
Anthyroid	0.02	0.05	0.28	0.18	0.09	0.09	0.20	0.15
Cardiotocography	0.05	0.05	0.28	0.22	0.07	0.05	0.29	0.17
Mammography	0.22	0.19	0.46	0.44	0.12	0.15	0.21	0.11
PageBlocks	0.24	0.06	0.41	0.40	-0.02	0.04	0.26	0.14
PenDigits	0.02	0.04	0.11	0.20	0.11	0.00	0.27	0.08
Spambase	-0.01	0.10	0.38	0.35	-0.01	-0.03	0.21	0.15
Waveform	0.10	0.27	0.46	0.41	0.23	0.03	0.31	0.20

Tabelle 5.5: Vergleich der erzielten Vorhersagegüten pro Datensatz, zugrundeliegendem Funktionsoptimierer und submodularer Funktion. Die Tabelle gibt jeweils die Differenz zwischen den erzielten Metriken bei Verwendung einer gewöhnlichen SUMMARY OUTLIER DETECTION und der Erweiterung mit RSP-Bäumen wieder, wobei ein positiver Wert eine Verbesserung bei Nutzung des RSP-Verfahrens anzeigt. Die fett markierten Einträge kennzeichnen eine Verschlechterung der Vorhersagegüte bei Nutzung des SOD-RSP-Verfahrens.

Anhand der Tabelle lässt sich zunächst erkennen, dass die zusätzliche rekursive Partitionierung des Raums in den meisten Fällen einen positiven Effekt auf die SUMMARY

OUTLIER DETECTION ausübt. Die größtmögliche Verbesserung der F1-Metrik, die durch die RSP-Technik erzielt wurde, liegt bei 0,46 Punkten, wohingegen die geringste Verbesserung durch RSP mit 0,02 Punkten beziffert werden kann. In einem Fall führte die RSP-Technik zu einer Verschlechterung der F1-Metrik um 0,01 Punkte. Darüber hinaus lassen sich die größten Verbesserung der F1-Metrik durch die RSP-Technik auf Grundlage einer reinen Zufallsauswahl von Repräsentanten beobachten. Wendet man sich nun der ROC-AUC-Metrik zu, so lässt sich die größtmögliche Verbesserung durch RSP mit 0,31 Punkten beziffern, wohingegen die geringste Verbesserung mit 0,03 Punkten gemessen wurde. In einem Fall wurde durch RSP keine Verbesserung gegenüber der einfach partitionierenden SOD-Methodik beobachtet. In drei Fällen verschlechterte sich die ROC-AUC-Metrik, wobei die größte Verschlechterung mit 0,03 Punkten beziffert werden kann. Wie auch bei der F1-Metrik können die größten Verbesserungen der ROC-AUC-Metrik durch RSP auf Grundlage einer reinen Zufallsauswahl beobachtet werden.

Betrachtet man die Tabelle 5.5 aus der Perspektive der Datensätze, so lässt für *Annthyroid*, *Cardiotocography*, *Mammography*, *PenDigits* und *Waveform* unabhängig des eingesetzten Optimierers oder der eingesetzten Funktion festhalten, dass die RSP-Technik in den meisten Fällen zu einer Verbesserung führt, zumindest aber konkurrenzfähig zur gewöhnlichen SUMMARY OUTLIER DETECTION ist. Beim Datensatz *PageBlocks* konnte lediglich für die ROC-AUC-Metrik, den GREEDY-Optimierer und der submodularen Funktion des EXEMPLAR-BASED CLUSTERINGS eine Verschlechterung um 0,02 Punkte durch die RSP-Technik beobachtet werden. Hinsichtlich des *Spambase*-Datensatzes konnte in drei Fälle eine Verschlechterung der ROC-AUC- und F1-Metriken beobachtet werden, wobei die größte Verschlechterung bei 0,03 Punkten lag und die größte Verbesserung durch RSP mit 0,38 Punkten gemessen wurde.

Vergleich klassischer und submodularer Ausreißererkenntungsverfahren

Nun soll betrachtet werden, wie sich die durch die F1- und die ROC-AUC-Metrik gemessene Vorhersagegüte zwischen den hier betrachteten klassischen Methoden, also LOCAL OUTLIER FACTOR (LOF) und ISOLATION FOREST (IF), und den submodularen Methoden entwickelt. Hierfür wird zunächst für jeden Datensatz die beste Metrik gesucht, die jeweils durch SOD, SOD-RSP und eine der klassischen Methoden erzielt wurde. Anschließend wird zwischen diesen besten Metriken die Differenz gebildet, sodass ein positiver Wert jeweils eine Verbesserung durch SOD oder SOD-RSP anzeigt. Das Ergebnis dieses Vorgehens ist in den Tabellen 5.6a (für die F1-Metrik) und 5.6b (für die ROC-AUC-Metrik) abgebildet.

Zunächst soll ein Blick auf die F1-Metrik geworfen werden: Für die Metrikdifferenzen zwischen SOD und den klassischen Methoden lässt sich zunächst festhalten, dass in den meisten Fällen eine Verbesserung durch die beste SOD-Konfiguration gegenüber der besten klassischen Methode erzielt wurde. Die größte Verbesserung lag hier bei 0,2 Punkten,

Datensatz	SOD		SOD-RSP		Klassisch		F1-Score-Differenz	
	Optimierer	F1-Score	Optimierer	F1-Score	Methode	F1-Score	$\Delta_{\text{SOD - Klassisch}}$	$\Delta_{\text{SOD-RSP - Klassisch}}$
Anthyroid	Greedy	0.07	Random	0.32	IF	0.21	-0.14	0.11
Cardiotocography	Greedy	0.27	Random	0.36	LOF	0.25	0.02	0.11
Mammography	Sieve Str.	0.42	Random	0.60	IF	0.22	0.20	0.38
PageBlocks	Greedy	0.42	Random	0.52	LOF	0.39	0.03	0.13
PenDigits	Greedy	0.05	Random	0.22	LOF	0.09	-0.04	0.13
PenDigits	Sieve Str.	0.05	Random	0.22	LOF	0.09	-0.04	0.13
Spambase	Greedy	0.34	Random	0.46	IF	0.14	0.20	0.32
Waveform	Sieve Str.	0.27	Random	0.55	IF	0.11	0.16	0.44

(a) Vergleich mittels der F1-Metrik.

Datensatz	SOD		SOD-RSP		Klassisch		ROC-AUC-Differenz	
	Optimierer	ROC-AUC	Optimierer	ROC-AUC	Methode	ROC-AUC	$\Delta_{\text{SOD - Klassisch}}$	$\Delta_{\text{SOD-RSP - Klassisch}}$
Anthyroid	Random	0.61	Random	0.76	IF	0.84	-0.23	-0.08
Cardiotocography	Greedy	0.85	Random	0.91	LOF	0.85	0.00	0.06
Mammography	Sieve Str.	0.84	Random	0.89	LOF	0.83	0.01	0.06
PageBlocks	Sieve Str.	0.92	Greedy	0.94	LOF	0.96	-0.04	-0.02
PenDigits	Greedy	1.00	Greedy	1.00	LOF	0.98	0.02	0.02
PenDigits	Sieve Str.	1.00	Greedy	1.00	LOF	0.98	0.02	0.02
Spambase	Sieve Str.	0.92	Greedy	0.88	LOF	0.83	0.09	0.05
Waveform	Greedy	0.85	Greedy	0.89	LOF	0.75	0.10	0.14

(b) Vergleich mittels der ROC-AUC-Metrik.

Tabelle 5.6: Vergleich jener Ausreißererkenntungsverfahren, die für einen gegebenen Datensatz die jeweils beste Metrik produziert haben. Die Verfahren wurden danach getrennt, ob diese das SOD-Verfahren mit oder ohne RSP-Bäume verwendet haben oder ob das Verfahren zu den hier verglichenen klassischen Methoden, also ISOLATION FOREST (IF) oder LOCAL OUTLIER FACTOR (LOF) gehört. Die Metrikdifferenzspalten geben jeweils an, wie groß die Differenz zwischen einer submodularen Methode (also SOD oder SOD-RSP) und der jeweils besten klassischen Methode ist. Fett gedruckte Einträge geben dabei eine Verschlechterung an, die durch eine der submodularen Methoden erzielt wurde.

wohingegen die kleinste Verbesserung bei 0,02 Punkten lag. In drei Fällen konnte eine Verschlechterung der Metrik beobachtet werden, wobei die größte Verschlechterung bei 0,14 Punkten und die kleinste Verschlechterung bei 0,04 Punkten lag. Weiterhin ist erkennbar, dass bei der Nutzung von SOD ohne RSP-Bäumen der jeweils beste F1-Wert in den meisten Fällen durch den GREEDY-Algorithmus gefolgt vom SIEVE STREAMING-Verfahren erzielt wurde. Eine Zufallsauswahl der Repräsentanten führte hier bei keinem der betrachteten Datensätze zu einer besten F1-Metrik. Anders verhält es sich jedoch bei SOD mit Nutzung von RSP-Bäumen: Hier führte ausschließlich die Zufallsauswahl von Repräsentanten bei der Baumgenerierung zu besten F1-Metriken: Ebenfalls konnte hier ausschließlich eine Verbesserung des F1-Werts gegenüber der jeweils besten klassischen Methode beobachtet werden: Die größte Verbesserung lag hier bei 0,44 Punkten, während die geringste Verbesserung bei 0,11 Punkten lag. Die Datensätze *Cardiotocography*, *Mammography*, *PageBlocks*, *Spambase* und *Waveform* profitierten hinsichtlich der F1-Metrik in unterschiedlichem Maße

von der SOD-Technik, unabhängig davon ob RSP-Bäume verwendet wurden oder nicht. Bei den Datensätzen *Annthyroid* und *PenDigits* konnte lediglich eine Verbesserung durch SOD-RSP erzielt werden, wobei sowohl für die klassischen Methoden als auch die SOD-Methoden F1-Metriken beobachtet wurden, die kleiner als ein Zehntel des Maximalwerts der Metrik betragen. Aus der Tabelle 5.6a geht weiterhin hervor, dass für den Datensatz *PenDigits* durch den GREEDY- und den SIEVE STREAMING-Algorithmus eine gleich gute Metrik erzielt werden konnte, weswegen dieser Datensatz doppelt vorhanden ist. Des Weiteren kann angesichts der F1-Metrik innerhalb der klassischen Methoden keine eindeutige Präferenz hinsichtlich des Ausreißererkenntnisverfahrens festgestellt werden.

Wird die ROC-AUC-Metrik betrachtet, so lässt sich für die Metrikdifferenz zwischen SOD (ohne RSP) und den klassischen Methoden festhalten, dass in den meisten Fällen eine Verbesserung der Metrik erreicht werden konnte, wobei die Verbesserungen häufig weniger als ein Zehntel des Metrikmaximalwerts betragen: So lag die größte Verbesserung durch SOD bei 0,1 Punkten, wohingegen die kleinste Verbesserung bei 0,01 Punkten lag. In einem Fall konnte durch SOD keine Verbesserung erzielt werden, sodass diese Methode mit dem jeweils besten klassischen Verfahren eine vergleichbare Vorhersageleistung erreicht hat. In zwei Fällen hat die SOD-Methodik eine Verschlechterung erzielt, wobei die größte Verschlechterung bei 0,23 Punkten und die kleinste Verschlechterung bei 0,04 Punkten lag. Ferner ist bei Nutzung des SOD-Verfahrens hinsichtlich der ROC-AUC-Metrik keine eindeutige Präferenz zu einem Optimierer zu erkennen, wobei hier (anders als bei Betrachtung der F1-Metrik) in einem Fall auch die reine Zufallsauswahl eine beste ROC-AUC erreicht hat. Betrachtet man nun die Metrikdifferenzen, die durch SOD-RSP erreicht wurden, so ergibt sich ein ähnliches Bild, wie bei SOD ohne RSP-Bäume: In den meisten Fällen konnte durch SOD-RSP eine Verbesserung der ROC-AUC-Metrik erreicht werden, wobei die größte Verbesserung bei 0,14 Punkten und die geringste Verbesserung bei 0,02 Punkten lag. In zwei Fällen hat SOD-RSP die Metrik verschlechtert, wobei die größte Verschlechterung bei 0,08 Punkten und die geringste Verschlechterung bei 0,02 Punkten lag. Für SOD-RSP kann hinsichtlich der ROC-AUC-Metrik keine eindeutige Präferenz hinsichtlich eines zu wählenden Optimierers festgehalten werden. Im Gegensatz zur F1-Metrik scheint hier aber auch in einigen Situationen eine gierige Auswahl der Repräsentanten gegenüber einer reinen Zufallsauswahl präferabel zu sein. Hinsichtlich der klassischen Methoden kann bei Ziel der Maximierung der ROC-AUC beobachtet werden, dass LOF deutlich häufiger ausgewählt wird, als der Isolation Forest. Mit Blick auf die Datensätze kann für *Cardiotocography*, *Mammography*, *PenDigits*, *Spambase* und *Waveform* festgehalten werden, dass SOD (ob mit oder ohne RSP-Bäumen) mindestens eine kompetitive Vorhersageleistung zur jeweils besten klassischen Methode liefert. Für die Datensätze *Annthyroid* und *PageBlocks* lieferte SOD in sämtlichen Szenarien eine Verschlechterung der ROC-AUC-Metrik. Weiterhin ist festzuhalten, dass (wie bei der F1-Metrik) hinsichtlich der Nutzung von SOD ohne RSP-Bäume sowohl der GREEDY-Algorithmus als auch das SIEVE STREAMING-Verfahren

für den SOD-Bewertungsalgorithmus Repräsentanten liefern, die zu einer Maximierung der ROC-AUC führen.

Identifizierung präferabler Parameter für die submodulare Ausreißerererkennung

Nachdem zuletzt der Fokus auf die durch die verschiedenen Verfahren und Verfahrenskombinationen produzierten Metriken lag, soll nun ein Blick darauf geworfen werden, welche Parameter für die submodularen Funktionen die jeweils besten F1- und ROC-AUC-Metriken produziert haben. Dadurch soll erörtert werden, ob es eine eindeutige Präferenz hinsichtlich einer bestimmten Konfiguration des SOD-Verfahrens gibt. Die Ergebnisse sind in Tabelle 5.7a und 5.7b aufgeführt.

	Greedy (SOD)			Greedy (SOD-RSP)			Sieve Streaming (SOD)			
	k	Funktion	Kernel	k	Abbruchbedingung	Funktion	Kernel	k	Funktion	Kernel
Anthyroid	40	IVM	$k_{\text{Matérn}}^{\nu=1/2}$	12	Mittlere Kernfunktion	IVM	k_{RBF}	80	IVM	$k_{\text{Matérn}}^{\nu=1/2}$
Cardiotocography	70	IVM	$k_{\text{Matérn}}^{\nu=1/2}$	3	Maximale Baumhöhe	IVM	k_{RBF}	30	IVM	$k_{\text{Matérn}}^{\nu=1/2}$
Mammography	30	IVM	$k_{\text{Matérn}}^{\nu=1/2}$	11	Maximale Baumhöhe	IVM	$k_{\text{Matérn}}^{\nu=1/2}$	10	IVM	$k_{\text{Matérn}}^{\nu=1/2}$
PageBlocks	10	IVM	k_{RBF}	5	Mittlere Kernfunktion	IVM	k_{RBF}	100	IVM	$k_{\text{Matérn}}^{\nu=1/2}$
PenDigits	50	IVM	$k_{\text{Matérn}}^{\nu=1/2}$	2	Streuung in Partition	IVM	k_{RBF}	50	IVM	$k_{\text{Matérn}}^{\nu=1/2}$
Spambase	20	IVM	k_{RBF}	4	Maximale Baumhöhe	IVM	$k_{\text{Matérn}}^{\nu=1/2}$	10	IVM	k_{RBF}
Waveform	100	IVM	$k_{\text{Matérn}}^{\nu=1/2}$	2	Streuung in Partition	IVM	$k_{\text{Matérn}}^{\nu=3/2}$	20	IVM	$k_{\text{Matérn}}^{\nu=1/2}$

(a) Maximierung der F1-Metrik.

	Greedy (SOD)			Greedy (SOD-RSP)			Sieve Streaming (SOD)			
	k	Funktion	Kernel	k	Abbruchbedingung	Funktion	Kernel	k	Funktion	Kernel
Anthyroid	100	IVM	$k_{\text{Matérn}}^{\nu=1/2}$	12	Mittlere Kernfunktion	IVM	k_{RBF}	70	IVM	$k_{\text{Matérn}}^{\nu=1/2}$
Cardiotocography	10	IVM	$k_{\text{Matérn}}^{\nu=1/2}$	6	Mittlere Kernfunktion	IVM	$k_{\text{Matérn}}^{\nu=1/2}$	30	IVM	k_{RBF}
Mammography	20	EBC	-	9	Maximale Baumhöhe	IVM	$k_{\text{Matérn}}^{\nu=1/2}$	60	IVM	k_{RBF}
PageBlocks	100	IVM	k_{RBF}	11	Mittlere Kernfunktion	IVM	$k_{\text{Matérn}}^{\nu=5/2}$	100	IVM	$k_{\text{Matérn}}^{\nu=1/2}$
PenDigits	90	IVM	$k_{\text{Matérn}}^{\nu=1/2}$	5	Streuung in Partition	IVM	$k_{\text{Matérn}}^{\nu=3/2}$	90	IVM	$k_{\text{Matérn}}^{\nu=1/2}$
Spambase	80	IVM	$k_{\text{Matérn}}^{\nu=1/2}$	14	Maximale Baumhöhe	IVM	$k_{\text{Matérn}}^{\nu=1/2}$	80	IVM	$k_{\text{Matérn}}^{\nu=1/2}$
Waveform	100	IVM	$k_{\text{Matérn}}^{\nu=1/2}$	6	Streuung in Partition	IVM	$k_{\text{Matérn}}^{\nu=1/2}$	70	IVM	k_{RBF}

(b) Maximierung der ROC-AUC-Metrik.

Tabelle 5.7: Vergleich der Parameter, die bei Nutzung der SUMMARY OUTLIER DETECTION auf Grundlage des GREEDY-Algorithmus (mit und ohne RSP-Bäume) sowie des SIEVE STREAMINGS (ohne RSP-Bäume) hinsichtlich verschiedener Datensätze zu einer Maximierung der jeweiligen Metrik und damit der Vorhersageleistung geführt haben.

Zunächst sollen die selektierten Konfiguration beim Ziel der Maximierung der F1-Metrik beleuchtet werden. Wendet man sich dem SOD-Verfahren auf Grundlage des GREEDY-Algorithmus zu, so lässt sich für die Kardinalitätsbeschränkung k , also der Anzahl in eine Zusammenfassung aufgenommener Beobachtungen, keine eindeutige Präferenz für einen Wert oder einen Wertebereich ausmachen. Des Weiteren wird, wie bereits die Untersuchungen zu Tabelle 5.4 vermuten lassen, die INFORMATIVE VECTOR MACHINE als submodulare Funktion bevorzugt. Hinsichtlich des dazugehörigen Kernels, wird in den meisten Fällen

der Matérn-Kernel $k_{\text{Matérn}}^{\nu=1/2}$ bevorzugt, wobei in zwei Fällen auch der RBF-Kernel selektiert wurde. Betrachtet man nun die SOD-RSP-Ausreißerererkennung auf Grundlage des GREEDY-Algorithmus, so wird für die Kardinalitätsbeschränkung offenbar ein kleiner Wert $k \leq 5$ präferiert. Hinsichtlich der Abbruchbedingung lässt sich keine eindeutige Präferenz erkennen. Wie auch beim SOD-Verfahren (ohne RSP-Bäume) wird beim SOD-RSP-Verfahren die IVM bevorzugt, wobei allerdings keine eindeutige Präferenz hinsichtlich des zu nutzenden Kernels erkennbar ist. Häufiger als bei SOD ist allerdings der RBF-Kernel selektiert worden. Nun soll die SUMMARY OUTLIER DETECTION auf Grundlage der SIEVE STREAMING-Optimiert betrachtet werden: Für die Kardinalitätsbeschränkung lässt sich keine eindeutige Präferenz erkennen, wobei hier, wie bei der GREEDY (SOD) die IVM als submodulare Funktion und der $k_{\text{Matérn}}^{\nu=1/2}$ -Kernel bevorzugt wird.

Nun sollen die Konfigurationen betrachtet werden, die hinsichtlich der ROC-AUC-Metrik zu einer Maximierung geführt haben. Betrachtet man zunächst die SUMMARY OUTLIER DETECTION ohne RSP-Bäume auf Grundlage der GREEDY-Optimierung so lässt sich für die Kardinalitätsbeschränkung eine Präferenz mit $k \geq 80$ ausmachen, wobei erwartungsgemäß (vgl. Tabelle 5.4) am häufigsten die IVM zu einer Maximierung der Metrik geführt hat. Analog zur Betrachtung der F1-Metrik wurde auch hier am häufigsten der $k_{\text{Matérn}}^{\nu=1/2}$ -Kernel bevorzugt, weswegen dieser als präferabel gelten kann. Wendet man sich dem SOD-RSP-Verfahren auf Grundlage der GREEDY-Optimierung zu, so lässt sich für den k -Parameter keine eindeutige Präferenz ableiten. Gegenüber der Maximierung der F1-Metrik scheinen aber größere Zusammenfassungen pro RSP-Baumknoten begünstigt zu werden. Als submodulare Funktion wird die IVM bevorzugt, wobei als Kernel verschiedene Matérn-Kernelvariationen und insbesondere der $k_{\text{Matérn}}^{\nu=1/2}$ -Kernel am häufigsten ausgewählt wurden. Für die Abbruchbedingung lässt sich ebenfalls keine Präferenz feststellen. Zuletzt wird das SOD-Verfahren auf Grundlage des SIEVE STREAMINGS beleuchtet, wobei hier größere k -Parameter mit $k \geq 60$ bevorzugt ausgewählt wurden. Als Funktion wurde, wie bei anderen Konfiguration schon festgestellt, die IVM bevorzugt. Beim dazugehörigen Kernel lässt sich keine eindeutige Präferenz feststellen, wobei aber der $k_{\text{Matérn}}^{\nu=1/2}$ -Kernel am häufigsten, gefolgt vom k_{RBF} -Kernel, ausgewählt wurde.

Identifizierung präferabler Parameter im zufallsbasierten SOD-RSP-Verfahren

Die im vergangenen Abschnitt diskutierten Tabellen 5.7a und 5.7b verglichen die Parameter verschiedener submodularer Ausreißerererkennungsverfahren auf Grundlage verschiedener Optimierungsverfahren, wobei der Fokus auf jenen Verfahren lag, die, neben der passenden Wahl der Kardinalitätsbeschränkung, vor allem eine angemessene Konfiguration der submodularen Funktion verlangen, um bedeutungsvolle Zusammenfassungen zu erreichen. Da besonders häufig die INFORMATIVE VECTOR MACHINE als submodulare Funktion hohe Vorhersagegüten erreicht hat, wurde gerade die Wahl des Kernels näher betrachtet. Wie

aus den Tabellen 5.6a und 5.6b und der dazugehörigen Beschreibung jedoch hervorgeht, liefert gerade bei Betrachtung der F1-Metrik die zufällige Auswahl von Repräsentanten und der Bewertung der Datensätze mittels des SUMMARY OUTLIER DETECTION-Verfahrens mit RSP-Bäumen häufig, gegenüber der nicht-zufälligen und der funktionsoptimierenden Auswahl von Repräsentanten, die bessere Vorhersagegüte. Daher sollen die besten Parameter im RANDOM (SOD-RSP) für jeden Datensatz nochmals in Tabelle 5.8 gesondert betrachtet werden.

	F1-Score			ROC-AUC		
	k	Abbruchbedingung	Metrik	k	Abbruchbedingung	Metrik
Anthyroid	3	Mittlere Kernfunktion	0.32	3	Streuung in Partition	0.76
Cardiotocography	4	Streuung in Partition	0.36	8	Mittlere Kernfunktion	0.91
Mammography	2	Streuung in Partition	0.60	13	Mittlere Kernfunktion	0.89
PageBlocks	2	Streuung in Partition	0.52	2	Mittlere Kernfunktion	0.93
PenDigits	9	Streuung in Partition	0.22	7	Streuung in Partition	1.00
Spambase	2	Streuung in Partition	0.46	5	Mittlere Kernfunktion	0.85
Waveform	5	Mittlere Kernfunktion	0.55	6	Mittlere Kernfunktion	0.88

Tabelle 5.8: Vergleich der durch die F1- beziehungsweise ROC-AUC-Metrik gemessenen, besten erzielten Vorhersagegüte auf Grundlage des RANDOM (SOD-RSP)-Verfahrens und der dazugehörigen Parameter.

Zunächst werden die besten, erzielten F1-Metriken betrachtet: Hierbei scheinen zunächst hinsichtlich der Kardinalitätsbeschränkung k eher kleine Werte mit $k \leq 5$ bevorzugt zu werden, wobei in den meisten Fällen die Abbruchbedingung „Streuung in Partition“ zu der besten F1-Vorhersagegüte geführt hat. Hinsichtlich der ROC-AUC-Metrik fallen die meisten selektieren Kardinalitätsbeschränkungen in das Intervall $k \in [5, 8]$, wobei im Gegensatz zur F1-Metrik nicht die Abbruchbedingung „Streuung in Partition“ sondern die Abbruchbedingung „Mittlere Kernfunktion“ am häufigsten die besten Resultate geliefert hat. Metrikübergreifend wurde für keinen der dargestellten Datensätze die Abbruchbedingung „Baumhöhe“ ausgewählt.

Einfluss der Kardinalitätsbeschränkung auf die Vorhersagegüte

Nun soll betrachtet werden, wie sich die Wahl der Kardinalitätsbeschränkung k bei der Nutzung submodularer Methoden zur Ausreißererknennung auf die Vorhersagequalität auswirkt. Es wird ausschließlich die INFORMATIVE VECTOR MACHINE betrachtet. Das Ziel dieser Untersuchung erfolgt dabei vor Betrachtung folgender Hypothese:

1. Nach Definition 2.1.3 ist bekannt, dass der Funktionswert der hier verwendeten submodularen Funktion mit Hinzunahme weiterer Beobachtungen steigt. Dadurch, dass repräsentativere Mengen gefunden werden aber, wie bei der SUMMARY OUTLIER DETECTION verlangt, trotzdem $k \ll n$, sollte sich ebenfalls die Vorhersagegüte mit steigendem Funktionswert verbessern.

Um diese Hypothese empirisch zu prüfen, soll ausschließlich die SUMMARY OUTLIER DETECTION auf Grundlage des GREEDY-Optimierers betrachtet werden. Der betrachtete Kernel sei der $k_{\text{Matérn}}^{\nu=1/2}$ -Kernel, da dieser in vielen Fällen (vgl. Tabelle 5.7) zu den besten, innerhalb der Versuchsreihe ermittelten Ergebnissen geführt hat. Festzulegen ist nun noch der Skalierungsparameter σ , dem eine erhebliche Bedeutung zukommt, da dieser die Ähnlichkeiten zwischen Beobachtungen kontrolliert und entscheidend für den Erfolg der Ausreißerererkennung ist (vgl. Abschnitt 2.3.1). Für die empirische Evaluation an dieser Stelle soll derjenige σ -Parameter gewählt werden, der für $k = 10$ die jeweils betrachtete Metrik maximiert hat: Diese Wahl erfolgt vor dem Hintergrund einen Parameter zu finden, der die Ähnlichkeiten zwischen den Daten ausreichend gut beurteilen konnte. Das Ergebnis dieser Aufbereitung ist in Abbildung 5.1 grafisch dargestellt.

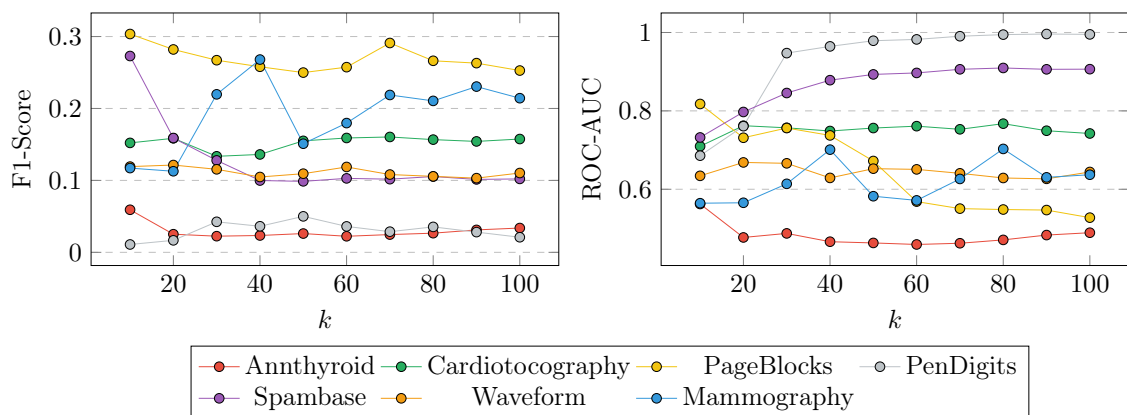


Abbildung 5.1: Vergleich der bei Wahl eines bestimmten Parameters k erreichte Metrik unter Nutzung des $k_{\text{Matérn}}^{\nu=1/2}$ -Kernels, verschiedener Datensätze, der SUMMARY OUTLIER DETECTION ohne RSP-Bäume und dem GREEDY-Optimierer.

Zunächst sollten die erreichten F1-Scores bei Variierung der Kardinalitätsbeschränkung k für verschiedene Datensätze verglichen werden. Man stellt fest, dass in einigen Szenarien die Aufnahme weiterer Punkte die Metrik nicht stark steigen oder fallen lässt. Diese Aussage lässt sich zumindest für die Datensätze *Anthyroid*, *PenDigits*, *Cardiotocography* und *Waveform* treffen. Für den Datensatz *PageBlocks* lässt sich ein leicht fallender Trend erkennen, wohingegen sich für den *Spambase*-Datensatz ein zunächst stark fallender Trend beobachten lässt, der sich mit weiterer Aufnahme von Beobachtungen bei $F_1 = 0,1$ stabilisiert. Hinsichtlich des *Mammography*-Datensatzes lässt sich ein steigender Trend beobachten, wobei schon mit $k = 40$ eine Vorhersagegüte erkennbar ist, die durch weitere Aufnahme von Beobachtungen in die Zusammenfassung nicht mehr erreicht wurde.

Wendet man sich der ROC-AUC-Metrik zu, so lassen sich in einigen Fällen ähnliche Beobachtungen, wie bei der Betrachtung der F1-Metrik machen: So zeigen die Datensätze *Anthyroid*, *Waveform* und *Cardiotocography* wenig Varianz bei Variierung der Kardinalitätsbeschränkung k . Für den Datensatz *PageBlocks* lässt sich ein eher linear-fallender

Trend ausmachen, während für den Datensatz *Mammography* kein eindeutiger Trend angegeben werden kann. Interessanterweise lässt sich hinsichtlich der Datensätze *PenDigits* und *SpamBase* ein logarithmisch-steigender Trend erkennen, der den Verdacht nahe legt, dass durch die Hinzunahme weiterer Beobachtungen ein gewisser Sättigungseffekt hinsichtlich der Metrik eintritt. Dies würde der *diminishing returns*-Eigenschaft der submodularen Funktion entsprechen, wobei sich an dieser Stelle nicht eindeutig klären lässt, ob dieses Verhalten tatsächlich der Submodularität geschuldet ist.

Einfluss der Optimierer auf den Funktionswert

Nun soll untersucht werden, welchen Einfluss die verschiedenen Optimierungsverfahren zur submodularen Funktionsmaximierung auf den tatsächlich erreichten Funktionswert haben. Die Untersuchung wird dabei auf Grundlage folgender Hypothese durchgeführt:

2. Gemäß Abschnitt 2.2 sollte der GREEDY-Algorithmus den besten Funktionswert gefolgt vom SIEVE STREAMING-Verfahren garantieren. Eine reine Zufallsauswahl von Punkten sollte nicht-repräsentative Zusammenfassungen liefern, die zu durchweg schlechteren Funktionswerten führen sollte.

Betrachtet wird die INFORMATIVE VECTOR MACHINE mit dem $k_{\text{Matérn}}^{\nu=1/2}$ -Kernel. Wie in Abschnitt 2.3.1 beschrieben, ist es notwendig den Skalierungsparameter σ derart fest zu wählen, dass dieser zu den Daten passt. Um dies zu bewerkstelligen wird unter allen evaluierten Parameter jener gewählt, der für $k = 10$ eine sinnvolle Ausreißerererkennung ermöglicht hat, hier gemessen durch eine Maximierung des F1-Scores. Das Ergebnis dieser Prozedur ist in Abbildung 5.2 grafisch dargestellt.

Aufgeführt sind die Datensätze *Annthyroid*, *Mammography*, *PageBlocks* und *Spambase*. Für die Datensätze *Waveform*, *Cardiotocography* und *PenDigits* konnte kein sinnvoller Plot angegeben werden, da sich die Wertereihen vollständig überlagerten. Der Grund hierfür liegt möglicherweise darin, dass durch die oben beschriebene Prozedur kein angemessener Kernel beziehungsweise Skalierungsparameter gefunden werden konnte, der zu den Datensätzen passt. Für diese Untersuchung wird sich daher auf die abgebildeten Daten beschränkt.

Grundsätzlich lässt sich erkennen, dass der GREEDY-Algorithmus in allen hier betrachteten Szenarien zu den besten beobachteten Funktionswerten führt. In Abhängigkeit der Datensätze ergibt sich dabei eine unterschiedlich große Verschlechterung gegenüber dem nächst-schlechteren Optimierungsverfahren: Diese soll an dieser Stelle nur qualitativ beurteilt werden, da die quantitative Beurteilung der Verbesserung genaue Kenntnis über die anwendungsfall-spezifische Bedeutung des Funktionswerts verlangen würde.

Betrachtet man zunächst den *Annthyroid*-Datensatz, so lässt sich gegenüber dem nächst-schlechteren Optimierungsverfahren, dem SIEVE STREAMING, eine recht große Verbesserung durch GREEDY erkennen. Das SIEVE STREAMING-Verfahren scheint gegenüber der

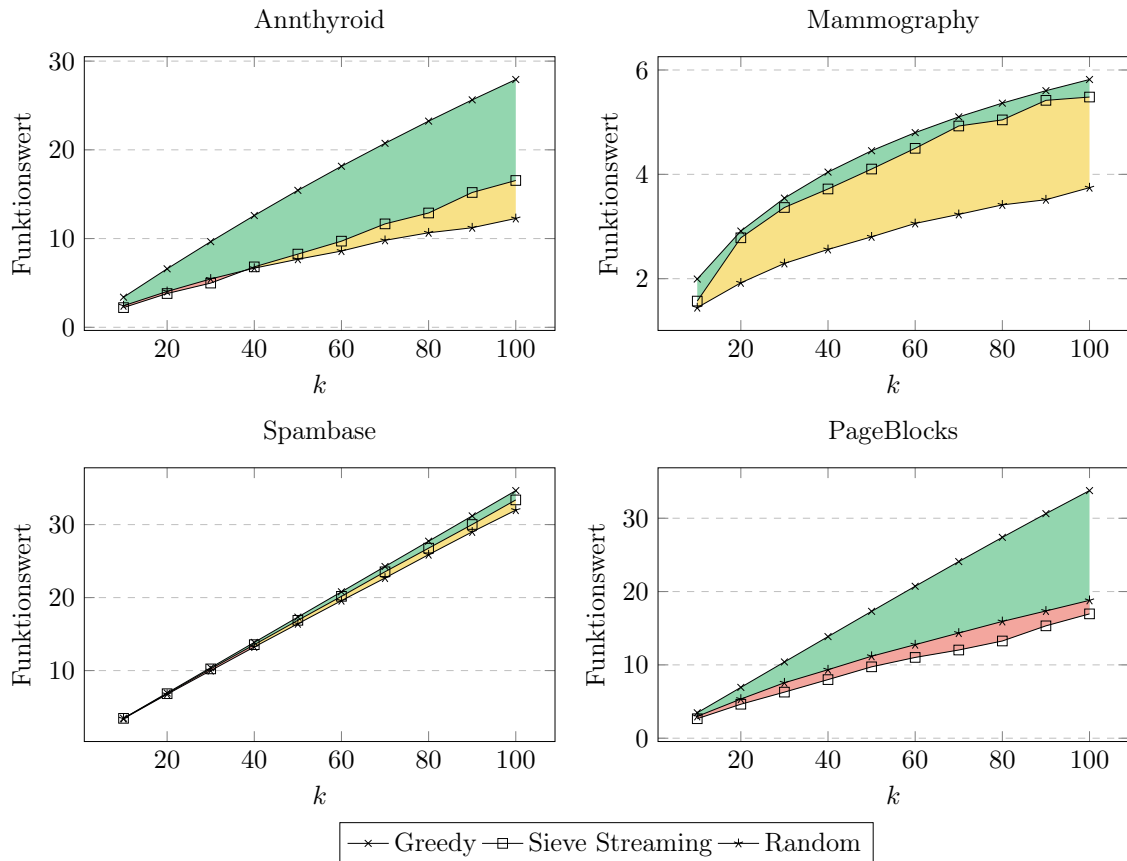


Abbildung 5.2: Vergleich der erzielten Funktionswerte bei Nutzung der INFORMATIVE VECTOR MACHINE, des $k_{\text{Matérn}}^{\nu=1/2}$ -Kernels, einem festen Skalierungsparameter σ , der Optimierer GREEDY und SIEVE STREAMING und einer reinen Zufallsauswahl von Repräsentanten („Random“). Die grüne Fläche repräsentiert die Verbesserung des Funktionswert bei Nutzung des GREEDY-Algorithmus gegenüber dem SIEVE STREAMING-Verfahren, während die gelbe Fläche die Verbesserung des SIEVE STREAMING-Verfahrens gegenüber der Zufallsauswahl wiedergibt. Rote Flächen geben eine Verschlechterung des SIEVE STREAMINGS gegenüber der reinen Zufallswahl von Repräsentanten an.

Zufallsauswahl geringfügige Verbesserungen erst ab $k = 40$ zu erzielen, wobei für größer werdende Zusammenfassungen die Verbesserung zunimmt. Für $k < 40$ produziert die Zufallsauswahl bessere Funktionswerte.

Hinsichtlich des *Mammography*-Datensatzes lässt sich durch SIEVE STREAMING eine deutliche Verbesserung gegenüber der reinen Zufallsauswahl beobachten. Die Funktionswerte sind dabei in relativer Nähe zu jenen, die der GREEDY-Algorithmus hervorgebracht hat. Die Zufallsauswahl von Repräsentanten führt zu erkennbar schlechteren Funktionswerten als GREEDY oder SIEVE STREAMING.

Im *Spambase*-Datensatz liegen die Funktionswerte gerade für kleine k sehr dicht beieinander, sodass qualitative Unterschiede kaum auszumachen sind. Für größer werdende k ist zu erkennen, dass die Differenzen zwischen den Verfahren größer werden, sodass erkennbar

ist, dass GREEDY die besseren Funktionswerte, gefolgt von SIEVE STREAMING und der Zufallsauswahl, hervorbringt. Ob diese erkennbar geringen Verbesserungen der Verfahren zueinander tatsächlich für die Anwendung bedeutsam sind, bleibt an dieser Stelle offen.

Der *PageBlocks*-Datensatz zeigt in Abgrenzung zu den anderen betrachteten Datensätzen, dass das SIEVE STREAMING durchweg schlechtere Funktionswerte hervorgebracht hat, als die reine Zufallsauswahl von Repräsentanten. Der GREEDY-Algorithmus verbessert dabei gegenüber dieser Zufallsauswahl den Funktionswert deutlich, wobei die Differenz mit Zunahme des k -Parameters größer zu werden scheint.

Laufzeitvergleich klassischer und submodularer Ausreißererkenntungsverfahren

Abschließend sollen die erzielten Laufzeiten der verschiedenen Verfahren verglichen werden: Hierzu werden der kleinste und größte Datensatz betrachtet, die auf die gleiche Art und Weise gesplittet und normalisiert wurden und des Weiteren ähnlich dimensioniert sind, nämlich *Cardiotocography* und *PenDigits*. Zunächst soll ein Blick auf die submodularen Methoden zur Ausreißererkenntung, also insbesondere den verschiedenen Kombinationen aus Ausreißererkenntungsmethode und zugrundeliegendem Optimierungsverfahren, geworfen werden. Um den Einfluss der verschiedenen submodularen Optimierungs- und Bewertungsverfahren adäquat herauszustellen, werden methodenübergreifend alle weiteren Parameter fixiert, sodass lediglich die INFORMATIVE VECTOR MACHINE mit RBF-Kernel betrachtet wird. Da die Laufzeit der SUMMARY OUTLIER DETECTION mit RSP-Bäumen ganz wesentlich von der Baumhöhe abhängt, soll an dieser Stelle für diese Methodenkombination ausschließlich die Abbruchbedingung „Baumhöhe“ mit $t_h = 3$ betrachtet werden, da die anderen Abbruchbedingungen „Mittlere Kernfunktion“ und „Streuung in Partition“ in Abhängigkeit der konkret gefundenen Partitionen und im Datensatz vorhandenen Muster unterschiedlich hohe Bäume produzieren können. Unter allen verbliebenen Konfigurationen wurde die mittlere Laufzeit ermittelt, welche in Abbildung 5.3 in Abhängigkeit der Kardinalitätsbeschränkung k geplottet sind. Die Untersuchung der Laufzeit soll anhand folgender Hypothese erfolgen:

3. Hinsichtlich der submodularen Methoden zur Ausreißererkenntung sollte bei Nutzung einer reinen Zufallsauswahl von Repräsentanten die Laufzeit mit Zunahme der Kardinalitätsbeschränkung k konstant bleiben, wohingegen bei der GREEDY-Optimierung zusammen mit der IVM und den Laufzeiteigenschaften nach Abschnitt 4.1 ein superlineares Wachstum zu erwarten ist. Für das SIEVE STREAMING-Verfahren lässt sich keine Hypothese formulieren, da die konkrete Laufzeit von der Ordnung der Elemente im Datensatz und der Grenzwertwahl abhängt. Hinsichtlich des LOCAL OUTLIER FACTORS ist aufgrund der von der *scikit-learn*-Bibliothek verwendeten Indexstruktur ein eher logarithmisches Wachstum zu erwarten, wohin gegen beim ISOLATION FO-

REST eher ein lineares Wachstum mit Zunahme der Menge zu trainierender Bäume beobachtbar sein müsste.

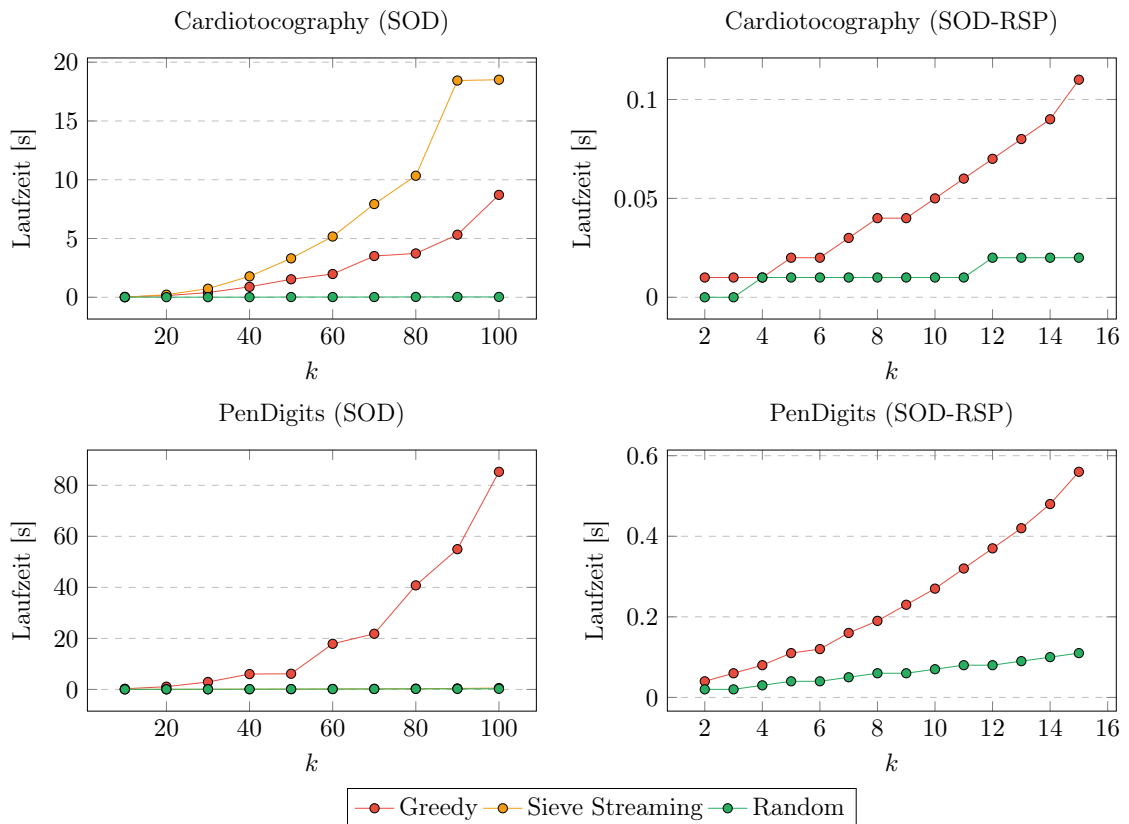


Abbildung 5.3: Erzielte mittlere Laufzeiten der SUMMARY OUTLIER DETECTION mit RSP-Bäumen („SOD-RSP“) und ohne RSP-Bäume („SOD“) auf Grundlage der Optimierungsverfahren GREEDY, SIEVE STREAMING und einer reinen Zufallsauswahl von Repräsentanten. Die Laufzeiten sind in Abhängigkeit der Kardinalitätsbeschränkung k und in Sekunden angegeben und wurden auf Grundlage der Datensätze *Cardiotocography* und *PenDigits* erzielt. Für SOD-RSP wurde keine Optimierung anhand des SIEVE STREAMING-Verfahrens berücksichtigt.

Wendet man sich zunächst der SUMMARY OUTLIER DETECTION ohne RSP-Bäume zu, so erkennt man, dass unabhängig des Datensatzes die reine Zufallsauswahl von Repräsentanten zu einer konstanten Laufzeit in der Ausreißererkennung, unabhängig der Wahl des k -Parameters, führt. Hinsichtlich der GREEDY-Optimierung lässt sich erkennen, dass mit Zunahme der Anzahl auszuwählender Repräsentanten ein mitunter deutliches über-lineares Steigerungsverhalten der Laufzeit vorhanden ist. Für den *Cardiotocography*-Datensatz konnte beobachtet werden, dass die GREEDY-Optimierung schneller war als das SIEVE STREAMING-Verfahren, wohingegen im (größeren) *PenDigits*-Datensatz, das SIEVE STREAMING mit der gleichen, niedrigen und konstanten Laufzeit wie die Zufallsauswahl von Repräsentanten in Erscheinung getreten ist und GREEDY eine mit der Anzahl an Repräsentanten über-linear steigende Laufzeit erreicht hat. Bemerkenswert ist ebenfalls, dass bei Verwen-

derung des *Cardiotocography*-Datensatzes und des SIEVE STREAMING-Verfahrens die Laufzeit für $k = 90$ und $k = 100$ ein gewisses Plateau erreicht zu haben scheint und diese nicht so stark steigt, wie es für kleinere k Parameter vorher der Fall gewesen ist.

Nun soll die SUMMARY OUTLIER DETECTION mit RSP-Bäumen betrachtet werden: Auch hier lässt sich unabhängig des Datensatzes für den GREEDY-Algorithmus mit Zunahme der Kardinalitätsbeschränkung k ein über-lineares Steigerungsverhalten beobachten. Hinsichtlich der Zufallsauswahl von Beobachtungen ist bemerkenswert, dass die Laufzeit abhängig des k -Parameters nun nicht mehr konstant bleibt, sondern ein gewisses Wachstum beobachtet werden kann. Im größeren *PenDigits*-Datensatz ist die GREEDY-Optimierung ausnahmslos für alle k -Parameter echt langsamer als es bei der Zufallsauswahl der Fall gewesen ist, wohingegen im kleineren *Cardiotocography*-Datensatz zumindest für $k = 4$ die Zufallsauswahl eine mit der gierigen Optimierung vergleichbare Laufzeit erreicht hat.

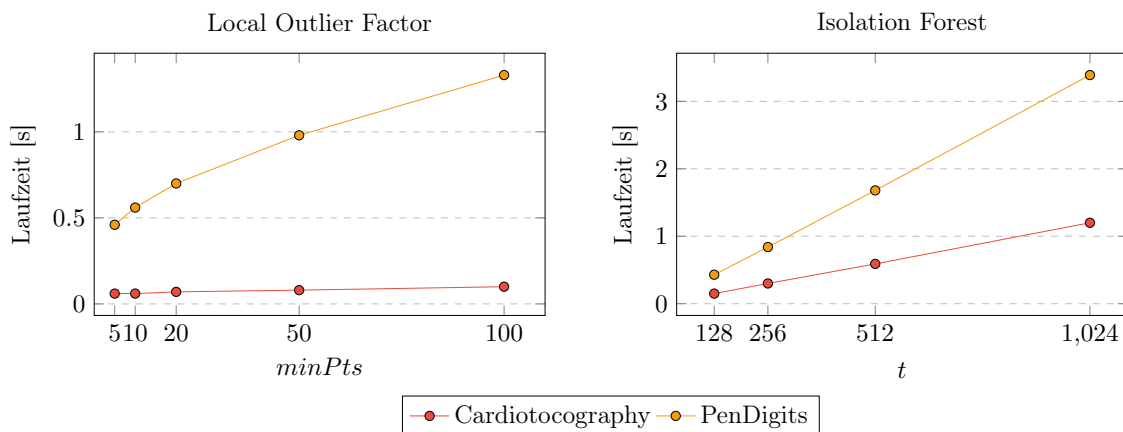


Abbildung 5.4: Erzielte mittlere Laufzeiten des LOCAL OUTLIER FACTORS bei Variation der Nachbarschaftsgröße $minPts$ und des ISOLATION FORESTS bei Variation der Anzahl zu trainierender Bäume t . Als Grundlage wurden die Datensätze *Cardiotocography* und *PenDigits* betrachtet.

Zuletzt soll geprüft werden, wie sich die Laufzeit des LOCAL OUTLIER FACTORS (LOF) und des ISOLATION FORESTS (IF) in Abhängigkeit der Datensätze *Cardiotocography* und *PenDigits* und der laufzeitrelevanten Parameter verhalten. Dies seien für LOF die Nachbarschaftsgröße $minPts$ und für IF die Anzahl zu trainierender Bäume t . Die grafische Illustration der Laufzeiten finden sich in Abbildung 5.4.

Wird zunächst der LOF betrachtet, stellt man für den *PenDigits*-Datensatz tendenziell eine eher empirisch logarithmisch wachsende Laufzeit fest. Im *Cardiotocography*-Datensatz sind die mittleren Laufzeiten auch für $minPts = 100$ derart gering, dass sich nur schwer eine sinnvolle Wachstumsaussage treffen lässt, wobei auch hier das Wachstum eher logarithmisch beschränkt zu sein scheint. Im Vergleich zwischen den Datensätzen führt der größere *PenDigits*-Datensatz gegenüber dem *Cardiotocography*-Datensatz auch zu messbar höheren Laufzeiten.

Bei Betrachtung des ISOLATION FOREST-Algorithmus lässt sich für die Datensätze *Cardiotocography* und *PenDigits* ein ausnahmslos linearer Kurvenverlauf erkennen. Die Kurvenverläufe unterscheiden sich hierbei zwischen den Datensätzen anhand der Verschiebung auf der Laufzeitachse und der Steigung der Laufzeit. So ist im kleineren *Cardiotocography*-Datensatz eine geringere Steigung und eine niedrigere Verschiebung auf der Laufzeitachse zu erkennen, wohingegen beim *PenDigits*-Datensatz sowohl die Steigung als auch die Verschiebung größer ausfällt.

5.1.3 Diskussion

Im vergangenen Abschnitt wurden verschiedene Aspekte der submodularen Ausreißererken- nung hinsichtlich der erzielbaren Vorhersagegüte herausgestellt und ein Vergleich zwischen diesen und klassischen Verfahren vollzogen. Bei Betrachtung der submodularen Funktio- nen INFORMATIVE VECTOR MACHINE (IVM) und des EXEMPLAR-BASED CLUSTERINGS (EBC) stellte sich heraus, dass die IVM in den meisten Fällen die bessere Vorhersagegüte (unabhängig der Betrachtung von F1-Score oder ROC-AUC) lieferte, wobei die Metrikdif- ferenzen mit bis zu 0,36 Punkten durchaus beträchtlich zu sein scheinen. Daraus lässt sich jedoch nicht grundsätzlich schließen, dass das EBC für die Ausreißererken- nung mit sub- modularen Methoden ungeeignet ist: Vielmehr ist es denkbar, dass es für eine hinreichend gute Vorhersagegüte mit EBC erforderlich ist, weitere Parameteroptimierungen durchzu- führen. So wurden für die IVM diverse Kernel und Kernelparameter evaluiert, während für das EBC lediglich die quadratische, euklidische Distanz angenommen wurde. Tatsächlich ist die in Abschnitt 2.3.2 gegebene Definition des EBC hinsichtlich der zu verwendenden Unähnlichkeitsfunktion äußerst liberal und verlangt im Gegensatz zu herkömmlichen Di- stanzfunktionen lediglich die Nichtnegativität der Unähnlichkeit gegeben zweier Beobach- tungspaare. Für eine weitere Prüfung der Eignung des EBC ist es daher möglich, dass wei- tere Distanzfunktionen evaluiert werden oder aber aus Kerneln Unähnlichkeitsfunktionen konstruiert werden, die dann ähnlich wie die IVM optimiert werden.

An dieser Stelle soll weiterhin erörtert werden, weshalb die Wahl der Funktion bei einer reinen Zufallsauswahl von Beobachtungen die Metriken um bis zu 0,19 Punkte schwanken lässt. Nach Prüfung der aggregierten und selektierten Ergebnistabelle ist ersichtlich, dass die IVM und das EBC für einen Datensatz beziehungsweise einen spezifischen Datensatz- split nicht die gleiche Anzahl an Experimenten durchgeführt wurde. Wie im Abschnitt *Postprocessing* dargelegt ist, werden aber gerade für einen Split jene Experimente und jene Konfigurationen gesucht, die die Metrik maximieren. Da für die IVM mehr Experimente durchgeführt wurden, ist dort auch die Wahrscheinlichkeit größer, dass eine Zufallsauswahl gefunden wird, die die Metrik maximiert, sodass die Schwankungen zustande kommen.

Im Vergleich zwischen SOD und SOD-RSP konnten in den meisten Fällen Verbesse- rungen der Vorhersagegüten durch die Erweiterung mit RSP-Bäumen beobachtet, wobei

diese Verbesserungen beispielsweise bei Betrachtung der F1-Metrik bis zu 0,46 Punkte beitragen konnten und damit als bedeutsam angesehen werden können. Diese Erkenntnis lässt vermuten, dass die RSP-Technik durch ihre rekursive Inspektionsstrategie bereits identifizierter Regionen für die Ausreißerererkennung bedeutsamere Partitionen findet. Kritisch zu würdigen ist jedoch, dass durch RECURSIVE SUBMODULAR PARTITIONING ein zusätzlich vergrößerter Parameterraum entsteht, der einer zeitlich deutlich aufwändigeren Optimierung zu unterziehen ist, wozu insbesondere die Identifikation einer adäquaten Abbruchbedingung gehört. Ein gegenüber dem gewöhnlichen SOD-Verfahren zusätzliches Problem ist vor allem, dass logarithmisch in der Baumhöhe mehrfach der gesamte Datensatz neuen Repräsentanten zuzuordnen ist, was bei SOD nur einmal erfolgen muss.

Im Vergleich der klassischen und submodularen Verfahren zur Ausreißerererkennung konnte festgestellt werden, dass die submodularen Ansätze in den meisten Fällen gegenüber dem ISOLATION FOREST oder dem LOCAL OUTLIER FACTOR bessere Vorhersagegüten liefern konnten. Betrachtet man zunächst das Ziel der Maximierung der F1-Metrik so haben die klassischen Methoden mit maximal 0,39 Punkten und oftmals mit $F_1 < 0,25$ Punkten niedrige Vorhersagegüten produziert, wohingegen die submodularen Methoden die F1-Metrik der jeweils beste klassische Methode um mindestens 0,11 Punkte und bei Betrachtung des *Mammography*- und *Spambase*-Datensatz sogar um mindestens 0,32 Punkte verbessert hat. Interessanterweise sorgte hier gerade das SOD-RSP-Verfahren mit einer reinen zufallsbasierten Auswahl von Repräsentanten, die keine submodulare Funktion zur Repräsentantenwahl berücksichtigt, für die größten positiven Differenzen zu den klassischen Methoden. Denkbar ist, dass dieses Verfahren weniger über die submodularen Funktionen bedeutungsvolle Repräsentanten und damit bedeutungsvolle Partitionen findet, sondern die Abbruchbedingung kontrolliert, wann Regionen ähnlich oder kompakt genug sind, um sich für eine Ausreißerererkennung zu qualifizieren. Diese These wird sowohl dadurch unterstützt, dass bei einer Betrachtung von SOD ohne RSP-Bäume die zufallsbasierte Auswahl für keinen der betrachteten Datensätze die beste Vorhersagegüte produziert hat als auch dadurch, dass gerade bei den besten Zufallsbäumen stets die Wahl auf die Abbruchbedingungen „Mittlere Kernfunktion“ und „Streuung in Partition“ aber nie auf „Baumhöhe“ fiel (vgl. Tabelle 5.8). Vorteilhaft am zufallsbasierten SOD-RSP-Verfahren ist tatsächlich, dass die zusätzliche Laufzeitkomplexität, die üblicherweise durch die Optimierung der Funktion entsteht, entfällt und die Bäume dadurch vergleichsweise schnell aufgebaut werden können. Zu beachten ist allerdings auch, dass durch das exponentielle Wachstum der gefundenen Regionen die Evaluation des statistischen Tests gegenüber der gewöhnlichen SOD-Technik mehr Zeit in Anspruch nimmt (vgl. Abschnitt 3.5.1). Wendet man sich der ROC-AUC-Metrik zu, stellt man weiterhin fest, dass die submodularen Methoden zwar bessere Vorhersagegüten als die klassischen Methoden liefern, die Differenzen aber oftmals ausgesprochen gering ausfallen und die klassischen Methoden mit meistens ROC-AUC $> 0,83$ kompetitive Vorhersagen liefern aber weniger Parameteroptimierungsaufwand erfordern.

Hinsichtlich des Einflusses der Kardinalitätsbeschränkung auf die Vorhersagegüte konnte die aufgestellte Hypothese eins nicht eindeutig bestätigt werden. Es konnte insbesondere kein datensatzübergreifender Trend festgestellt werden, bei dem sich die Metrik mit Zunahme der Beobachtungen und damit auch mit Zunahme des Funktionswertes verbessert hat. In Abhängigkeit der betrachteten Datensätze und der jeweiligen Metriken konnte ein mannigfaltiges Verhalten bei Variation des k -Parameters beobachtet werden: Bei Betrachtung vieler Datensätze konnte festgestellt werden, dass sich die Metrik bei Variierung der Kardinalitätsbeschränkung kaum verändert hat, was auf den Schluss hindeutet, dass der Wahl des k -Parameters bei diesen Datensätzen keine kritische Bedeutung zukommt, was möglicherweise aber auch mit ungünstig gewählten Parametern zusammenhängt. Bei einigen Datensätzen hingegen konnte durch die Aufnahme weiterer Punkte ein Sättigungseffekt erzielt werden, die darauf hindeuten, dass mehr Repräsentanten in einigen Szenarien zu besseren Vorhersagegüten führt. Auch ist zu erwähnen, dass die SUMMARY OUTLIER DETECTION hinsichtlich einiger Datensätze sensibel auf die Kardinalitätsbeschränkung reagiert hat, sodass sich für einige spezifische k -Parameter lokale Maxima hinsichtlich der Metrik ergeben haben. Zusammenfassend lässt sich festhalten, dass eine Optimierung der Kardinalitätsbeschränkung wohl sinnvoll ist, um die bestmögliche Vorhersagegüte zu erreichen, aber der adäquaten Wahl der submodularen Funktion und ihrer Parameter (vor allem des Kernels bei der IVM) mehr Beachtung geschenkt werden sollte.

Als Nächstes soll der Einfluss der verschiedenen submodularen Optimierungsstrategien auf den Funktionswert diskutiert werden. Die aufgestellte Hypothese zwei ließ sich dabei nur begrenzt bestätigen: In jedem der hier betrachteten Fälle konnte beobachtet werden, dass die GREEDY-Optimierung zu den besten Funktionswerten geführt hat. Darüber hinaus war das SIEVE STREAMING-Verfahren in vielen Fällen, wie es die theoretischen Gütegarantien erwarten lassen, schlechter als GREEDY. Die Funktionswertdifferenzen, die zwischen diesen Verfahren lagen und für verschiedene Datensätze beobachtet werden konnten, variierten teils erheblich. Die Beurteilung des Einfluss dieser Funktionswertdifferenzen auf die Qualität der Ausreißerererkennung lässt sich jedoch nicht beurteilen, da die Beziehung zwischen Funktionswert und Metrik unklar ist. Im *PageBlocks*-Datensatz konnte des Weiteren beobachtet werden, dass das SIEVE STREAMING-Verfahren der Zufallswahl von Repräsentanten unterlegen war: Die Gründe können nicht eindeutig geklärt werden, wobei zwei Hypothesen naheliegen: Entweder wurden tatsächlich ungünstige Beobachtungen selektiert oder aber der hier betrachtete Kernel und die Grenzwerte, die sich aus diesem bestimmen, wurden für diesen Datensatz ungünstig gewählt, sodass sich die Siebe nicht vollständig füllen ließen. Im letzteren Falle würde für die gefundene Zusammenfassung S gelten, dass $|S| < k$ was mit Definition 2.1.3 in schlechteren Funktionswerten resultiert. In jedem Falle betont diese Mutmaßung, dass bei Nutzung der INFORMATIVE VECTOR MACHINE der Wahl des Kernels eine überaus kritische Bedeutung zukommt. Zusammenfassend lässt sich sagen, dass die GREEDY-Optimierung tatsächlich eine adäquate Optimierung der

submodularen Funktion erlaubt und weniger Hyperparameter aufweist, als es bei SIEVE STREAMING der Fall ist. In den Szenarien, in denen GREEDY abzulehnen ist, muss, insbesondere wenn endliche Datensätze betrachtet werden, darauf geachtet werden, dass die Grenzwerte adäquat gewählt sind, um zu vermeiden, dass Siebe offen bleiben und damit weniger Beobachtungen als beabsichtigt selektiert werden.

Zuletzt sollen die erzielten, mittleren Laufzeiten der submodularen Verfahren betrachtet werden. Die aufgestellte Hypothese drei ließ sich hierbei vollumfänglich bestätigen: Wendet man sich zunächst den Größenverhältnissen der Laufzeiten zwischen SOD-RSP und SOD zu, so stellt man fest, dass die SOD-RSP-Laufzeit datensatzunabhängig deutlich geringer als beim SOD-Pendant der Fall gewesen ist, was gerade bei den oftmals besseren Vorhersagegütern bedeutsam ist. Gerade das zufallsbasierte SOD-RSP-Verfahren, welches passable Metriken hervorgebracht hat, zeigt ein lediglich lineares Steigerungsverhalten der Laufzeiten an, während sich dieses bei GREEDY (SOD-RSP) mindestens empirisch überlinear darstellt. Betrachtet man das SUMMARY OUTLIER DETECTION-Verfahren ohne RSP-Bäume stellt man fest, dass gerade das SIEVE STREAMING-Verfahren unterschiedliche empirische Laufzeiten hervorbringt: So ist im kleineren *Cardiotocography*-Datensatz eine höhere Laufzeit als bei der GREEDY-Optimierung beobachtbar, während diese im größeren *PenDigits*-Datensatz konstant mit der Laufzeit der zufallsbasierten Auswahl ist. Eine Hypothese, die dieses Verhalten erklärt, ist möglicherweise im datenstrombasierten Modell des SIEVE STREAMINGS zu suchen: In Abhängigkeit der konkreten Anordnung der Elemente im Datenstrom können bereits die ersten k Elemente des Stroms die Siebe vollständig füllen, sodass das Verfahren terminiert. Bei anderen Datensätzen könnte es erforderlich sein, dass mehrfach über den Datensatz iteriert wird, um die Siebe zu schließen, was zu entsprechend höherer Laufzeit führt. Dies könnte auch unter Umständen erklären, warum in Abbildung 5.3 beim *Cardiotocography*-Datensatz und SOD-RSP für $k = 90$ und $k = 100$ ein Laufzeitplateau vorhanden ist: So könnten nach Aufnahme der 90 Elemente die weitere Aufnahme verbleibender Elemente besonders schnell erfolgen, sodass sich keine Zunahme der Laufzeit mehr beobachten lässt.

Abschließend sollen die Laufzeiten der klassischen Verfahren ISOLATION FOREST (IF) und LOCAL OUTLIER FACTOR (LOF) gewürdigt werden: Betrachtet man zunächst den IF so lässt sich ein absolut idealtypischer linearer Kurvenverlauf der Laufzeit in Abhängigkeit der Anzahl zu trainierender Bäume erkennen, der sich durch die Laufzeitkomplexität beim Trainieren $\mathcal{O}(t\psi \log \psi)$ und beim Anwenden des Ensembles $\mathcal{O}(nt \log \psi)$ bei jeweils fixer Stichprobengröße ψ und fixer Testdatensatzgröße n erklären lässt. Hinsichtlich des LOFs lässt sich ein logarithmischer Kurvenverlauf in Abhängigkeit der Nachbarschaftsgröße $minPts$ erkennen. Da, wie bereits in Abschnitt 3.4.2 erläutert, die Laufzeit durch die k -Nachbarschaftsquery dominiert wird und sich diese häufig durch Baumstrukturen beschleunigen lassen, die eine logarithmische Laufzeitkomplexität produzieren, liegt an dieser Stelle der Verdacht nahe, dass dies hier der Fall ist.

5.2 Exemplar-based Clustering auf moderner Hardware

Im Abschnitt 4.2 wurden mehrere Implementierungen der submodularen Funktion des EXEMPLAR-BASED CLUSTERINGS diskutiert und insbesondere eine GPU-basierte Variante vorgestellt, die mit der hohen Anzahl der Funktionsauswertungen bei Nutzung des GREEDY-Algorithmus motiviert wurde. In diesem Abschnitt soll die Effektivität des GPU-Algorithmus experimentell evaluiert werden. Die zentralen Fragen sind hierbei, in welchen Szenarien die Evaluation der Funktion auf der GPU zu einer geringeren Laufzeit führt, wie stark der sogenannte *Speedup* gegenüber den CPU-Implementierungen ist und welches Laufzeitverhalten sich empirisch messen lässt. So wird für die mengenparallele Implementierung des EXEMPLAR-BASED CLUSTERINGS bei Variierung eines laufzeitrelevanten Parameters und Fixierung aller restlichen Parameter asymptotisch ein linearer Kurvenverlauf erwartet, da die Laufzeitkomplexität durch $\mathcal{O}(|V| \cdot |S| \cdot |S_{\text{multi}}|)$ gegeben ist (vgl. Algorithmus 4.3).

Um eine detaillierte Laufzeituntersuchung der in Abschnitt 4.2 vorgestellten Implementierungen des EXEMPLAR-BASED CLUSTERINGS vorzunehmen, soll eine mehrdimensionale Analyse besagter Parameter und Laufzeitfaktoren durchgeführt werden, die einen Einfluss auf die real-messbare Laufzeit des Algorithmus ausüben. Wie der genannte Komplexitätsausdruck bereits signalisiert, gehören hierzu die Anzahl der zu evaluierenden Mengen in S_{multi} , die Anzahl der Elemente in der Grundmenge $|V| \in \mathbb{N}$, sowie die Anzahl der Elemente $k = |S|$, die sich in jeder Menge aus S_{multi} befinden. Ferner ergibt sich durch die Disparität der durch die GPU bereitgestellten Gleitkommaeinheiten einfacher und doppelter Präzision (vgl. Abschnitt 4.2.1) ein weiterer hardware-abhängiger Laufzeitfaktor, der hier ebenfalls untersucht werden soll.

5.2.1 Aufbau

In dem hier genutzten Experimentaufbau seien die drei problemabhängigen Laufzeitfaktoren zunächst fest mit $|S_{\text{multi}}| = 1000$, $k = 100$ und $|V| = 50000$. Zusätzlich wird in jedem Versuch untersucht, wie sich Nutzung einfacher Präzision (FP32) und doppelter Präzision (FP64) auf die Laufzeit auswirkt.

Für die Versuche werden den Parametern entsprechende Zufallsdaten erzeugt, die als Eingabe für den Algorithmus dienen, wobei jede Probleminstanz (beschrieben durch die gewählten Parametern) stets die gleichen Daten enthält, sodass ein adäquater Vergleich möglich ist. Gemessen wird die vollständige Zeit, die zur Lösung des Problems notwendig ist, wozu jedoch nicht die Datengenerierung gehört. Zur Messung der CPU-Laufzeit wird eine ein- und mehrfädige Implementierung (engl. *single and multithread*) herangezogen und hierfür jeweils Algorithmus 4.1 und 4.3 verwendet. Zur Messung der GPU-Beschleunigung wird der Algorithmus verwendet, wie er in Abschnitt 4.2 beschrieben wird. Neben der reinen Zeitmessung, wird außerdem der Speedup des mehrfädigen und des GPU-Algorithmus gegenüber der einfädigen Implementierung ermittelt. Sei t_{ST} die Zeit, die die einfädige Im-

plementierung zur Berechnung der submodularen Funktion benötigt hat und t die Zeit, die entweder der GPU- oder der mehrfädige Algorithmus benötigt hat, dann ist der Speedup S gegeben durch $S = \frac{t_{ST}}{t}$.

In den nun folgenden Abschnitten werden Versuche vorgestellt, in den jeweils einer der Parameter gezielt variiert und die jeweils anderen Parameter mit den genannten Werten fixiert werden. Die erste Versuchsreihe variiert dabei die Anzahl zu evaluierender Mengen mit $|S_{\text{multi}}| \in \{100, 500, 1000, 2000, \dots, 20000\}$. Die zweite Versuchsreihe variiert die Anzahl der Elemente in der Grundmenge mit $|V| \in \{1000, 5000, 10000, 25000, 50000, 100000, 150000, \dots, 1000000\}$. In der dritten Versuchsreihe wird die Anzahl der Elemente variiert, die sich in jeder Evaluationsmenge befindet, mit $k \in \{10, 25, 50, 75, 100, 200, \dots, 2000\}$ variiert.

Die Ergebnisse wurden auf einer typischen Desktop-CPU des Typs i7-6700 des Herstellers Intel und einer GPU des Typs GeForce GTX 1080 Ti des Herstellers NVIDIA erzielt. Sofern die CPU-mehrfädige Implementierung genutzt wurde, wurden die verfügbaren 4 CPU-Kerne (plus vier Fäden durch simultane Mehrfädigkeit, SMT) vollständig ausgenutzt.

5.2.2 Ergebnisse

Experiment		Einfache Präzision – FP32			Doppelte Präzision – FP64		
		Min.	$\mu \pm \sigma$	Max.	Min.	$\mu \pm \sigma$	Max.
Variation $ S_{\text{multi}} $	Multithread	3,66	$3,8 \pm 0,08$	3,98	3,43	$3,62 \pm 0,11$	3,74
	GPU	144,19	$149,9 \pm 2,6$	155,24	7,23	$7,72 \pm 0,31$	8,05
Variation $ V $	Multithread	3,44	$3,68 \pm 0,19$	4,12	3,06	$3,33 \pm 0,17$	3,74
	GPU	92,96	$155,77 \pm 14,67$	172,4	2,8	$7,64 \pm 1,07$	8,46
Variation k	Multithread	1,06	$2,23 \pm 0,96$	3,85	1,02	$2,04 \pm 0,88$	3,81
	GPU	103,91	$233,86 \pm 120,95$	489,1	4,83	$11,2 \pm 4,41$	19,26

Tabelle 5.9: Erzielte Speedups des EXEMPLAR-BASED CLUSTERINGS gegenüber der einfädigen Implementierung bei Variierung der beschriebenen Laufzeitfaktoren und der Nutzung einfacher und doppelter Präzision. Angegeben sind der minimale und der maximale Speedup sowie der durchschnittliche Speedup und die dazugehörige Standardabweichung, jeweils symbolisiert durch μ und σ .

Variation der Anzahl zu evaluierender Mengen Betrachtet man zunächst die Laufzeitergebnisse bei Variation der Anzahl zu evaluierender Mengen, so stellt man für den Fall einfacher Präzision zunächst fest, dass der GPU-Algorithmus für alle getesteten Parameter durchweg schneller als die einfädige oder mehrfädige CPU-Implementierung ist. Der gemittelte Speedup des GPU-Algorithmus gegenüber der einfädigen Implementierung lag etwa bei Faktor 150, wobei die Standardabweichung etwa bei Faktor drei liegt. Eine ähnliche Aussage lässt sich ebenfalls beim Vergleich der mehrfädigen CPU-Implementierung gegenüber der einfädigen CPU-Implementierung machen: Diese ist ebenfalls bei sämtlichen Parametervariationen schneller, wobei der Speedup lediglich etwa bei Faktor vier liegt mit

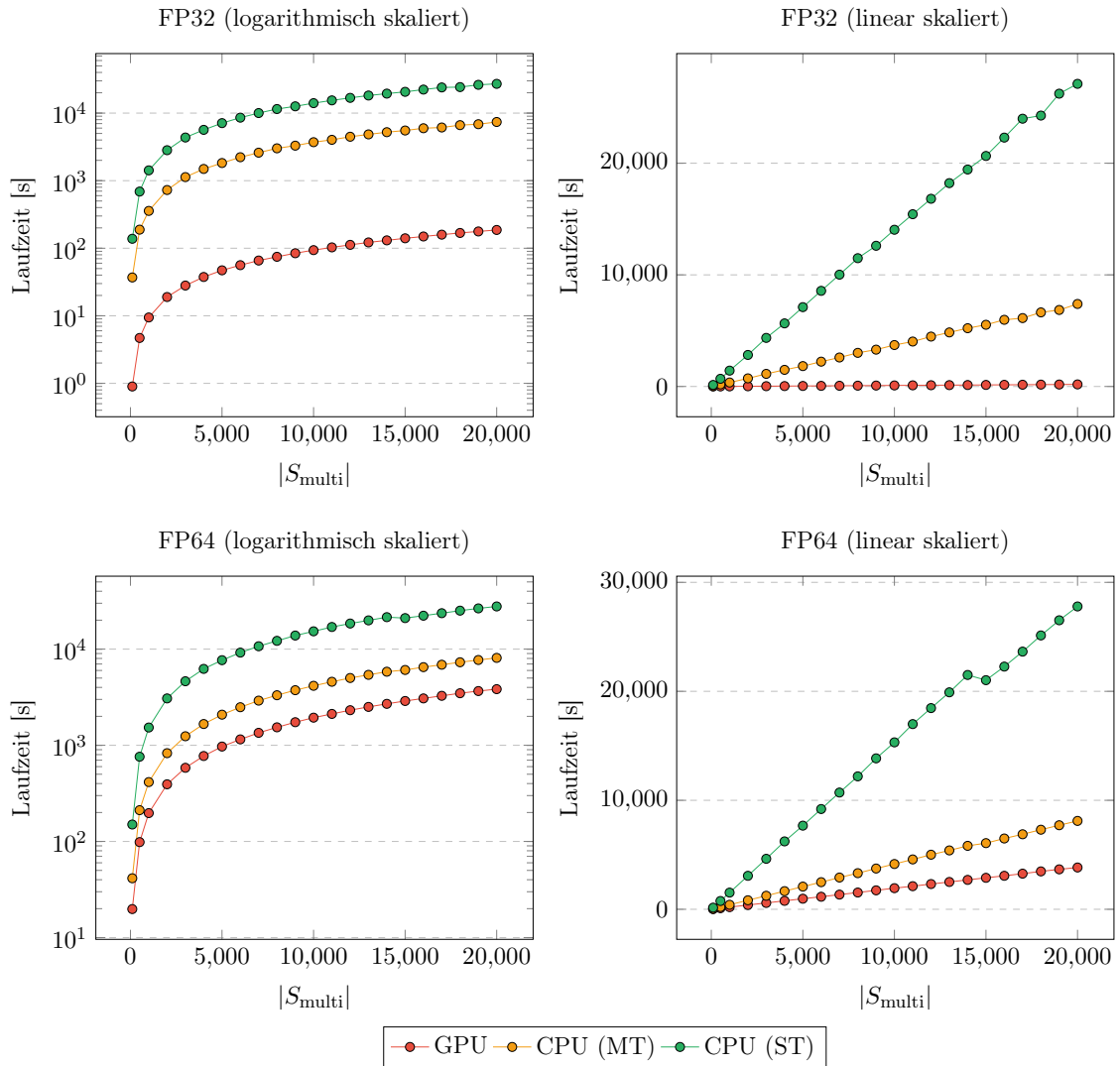


Abbildung 5.5: Erzielte Laufzeiten der submodularen Funktion des EXEMPLAR-BASED CLUSTERINGS auf Grundlage der einfädigen (ST) und mehrfädigen (MT) CPU-Implementierungen sowie der GPU-Implementierung unter Variierung der Anzahl der zu evaluierenden Mengen $|S_{\text{multi}}| \in \mathbb{N}$

einer Standardabweichung von 0,08. Grundsätzlich lässt sich für alle verglichenen Algorithmen ein nahezu linearer Kurvenverlauf bei Steigerung der zu evaluierenden Mengen feststellen.

Eine vergleichbare Situation ergibt sich bei Nutzung doppelter Präzision: Es ist ein linearer Kurvenverlauf zu erkennen, wobei der GPU-Algorithmus einen gegenüber der einfachen Präzision geringeren Speedup erfährt, allerdings in den verglichenen Szenarien gegenüber den CPU-Implementierungen immer noch am schnellsten ist. Der Speedup des GPU-Algorithmus liegt nunmehr bei Faktor 7,72 mit einer Standardabweichung von Faktor 0,31. Die mehrfädige Implementierung arbeitet im FP64-Bereich etwas langsamer und erzielt einen geringeren Speedup von Faktor 3,62 bei einer Standardabweichung von Fak-

tor 0,11. Im Vergleich mit der GPU-Implementierung ist der verlorene Speedup der CPU-Implementierung bei Nutzung doppelter Präzision allerdings weniger groß ausgeprägt.

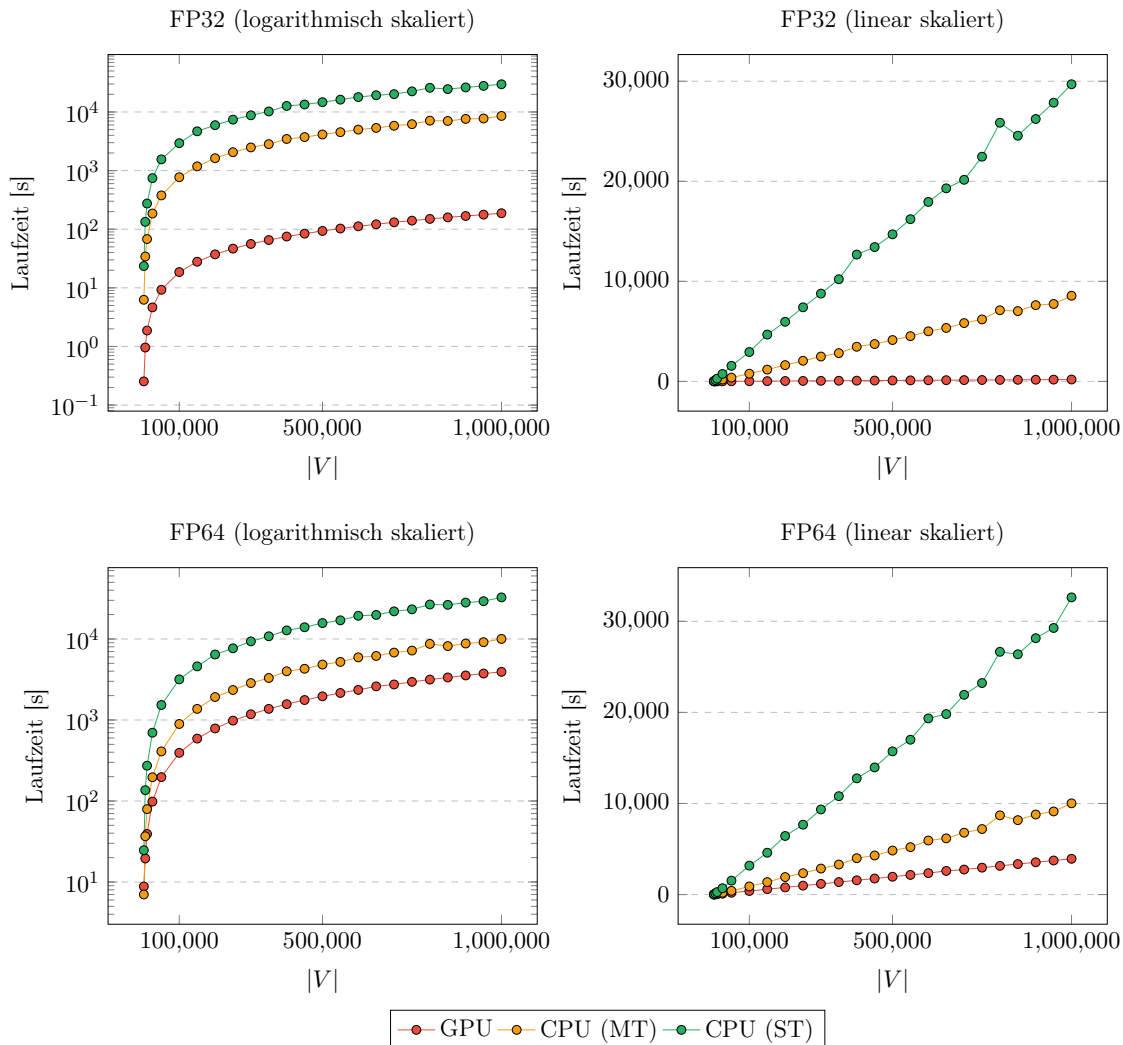


Abbildung 5.6: Erzielte Laufzeiten der submodularen Funktion des EXEMPLAR-BASED CLUSTERINGS auf Grundlage der einfädigen (ST) und mehrfädigen (MT) CPU-Implementierungen sowie der GPU-Implementierung unter Variierung der Anzahl der Elemente $|V| \in \mathbb{N}$ in der Grundmenge V .

Variation der Anzahl der Elemente in der Grundmenge Die Variation der Anzahl der Elemente in der Grundmenge bringt ein ähnliches Bild wie die Variation von $|S_{\text{multi}}|$ hervor. Bei Betrachtung der einfachen Präzision ist der GPU-Algorithmus in den getesteten Parametervariationen stets am schnellsten, gefolgt von der mehrfädigen Implementierung und der einfädigen Implementierung. Der Speedup des GPU-Algorithmus liegt bei Faktor 155,77 mit einer Standardabweichung von Faktor 14,67. Die Standardabweichung ist im Vergleich zu der Versuchsreihe „Variation von $|S_{\text{multi}}|$ “ größer und weist auf einen nicht-

konstanten Speedup hin. Tatsächlich es so, dass die CPU-basierten Implementierung und die einfädige Implementierung im Besonderen (da diese für die Berechnung des Speedups herangezogen wird) ein irreguläres Steigerungsverhalten der Laufzeit aufweisen, die die Varianz des Speedups erklären. Dies lässt sich beispielsweise für $|V| = 800000$ beobachten.

Darüber hinaus ist die mehrfädige CPU-Implementierung stets schneller als die einfädige CPU-Implementierung, wobei der Speedup im Mittel bei Faktor 3,68 bei einer Standardabweichung von Faktor 0,19 lag. Auffallend ist, dass der Laufzeitplot der mehrfädigen Implementierung zwar eine lineare Abhängigkeit von $|V|$ suggeriert, aber vereinzelt Datenpunkte von diesem Verhalten abverhalten. Dies lässt sich auch für die einfädige CPU-Implementierung beobachten, wobei dort stets die gleiche Konfiguration betroffen zu sein scheint (beispielsweise $|V| = 800000$), aber die Ausprägung dieser Abweichung vom suggerierten linearen Verhalten hinsichtlich der einfädigen CPU-Implementierung weniger stark ausfällt.

Dies lässt sich ebenfalls für die Versuchsreihe der doppelten Präzision beobachten, allerdings beeinflusst dieses Verhalten die Messung der Standardabweichung des erreichten Speedups hinsichtlich der GPU-Implementierung weniger stark. Dies liegt daran, dass der Speedup, der durch die GPU gegenüber der einfädigen CPU-Implementierung erzielt wird, deutlich geringer als bei Nutzung einfacher Präzision ausfällt. So liegt der Speedup der GPU bei Faktor 7,64 mit einer Standardabweichung von Faktor 1,07. Nichtsdestotrotz ist die GPU-Implementierung in den meisten evaluierten Parameterkombinationen sowohl der einfädigen als auch der mehrfädigen CPU-Implementierung überlegen. Es ließ sich lediglich für $|V| = 1000$ beobachten, dass die mehrfädige Implementierung der GPU-Implementierung überlegen ist.

Variation der Anzahl der Elemente in jeder Evaluationsmenge Die letzte Versuchsreihe variierte für eine feste Anzahl an Evaluationsmengen $|S_{\text{multi}}| = 1000$ die Anzahl der Elemente k , die in jedem $S \in S_{\text{multi}}$ enthalten ist. Die Betrachtung der GPU-Laufzeiten suggeriert eine lineare Abhängigkeit der Laufzeit von dem Parameter k , wobei der Speedup im Mittel beim Faktor 233,86 und die Standardabweichung durch den Faktor 120,95 gegeben ist. Diese Standardabweichung lässt sich, wie in der vorherigen Versuchsreihe, durch ein irreguläres Steigerungsverhalten der Laufzeit bei der einfädigen Implementierung erklären. Hierfür lässt sich interessanterweise beobachten, dass für $k \in [10, 900]$ ein linearer Kurvenverlauf hinsichtlich der Laufzeit zu erkennen ist, wohingegen für alle nachfolgenden, größeren k das Laufzeitverhalten von dieser Linearität abweicht. Dieses Verhalten lässt sich nur bei der einfädigen aber nicht bei der mehrfädigen Implementierung beobachten.

Der GPU-Algorithmus ist bei allen hier getesteten Parametern schneller als die einfädige und die mehrfädige CPU-Implementierung. Die mehrfädige CPU-Implementierung suggeriert wie die GPU-Laufzeitkurve eine lineare Abhängigkeit, wobei der Speedup bei Faktor 2,23 mit einer Standardabweichung von Faktor 0,96 liegt. Die einfädige CPU-

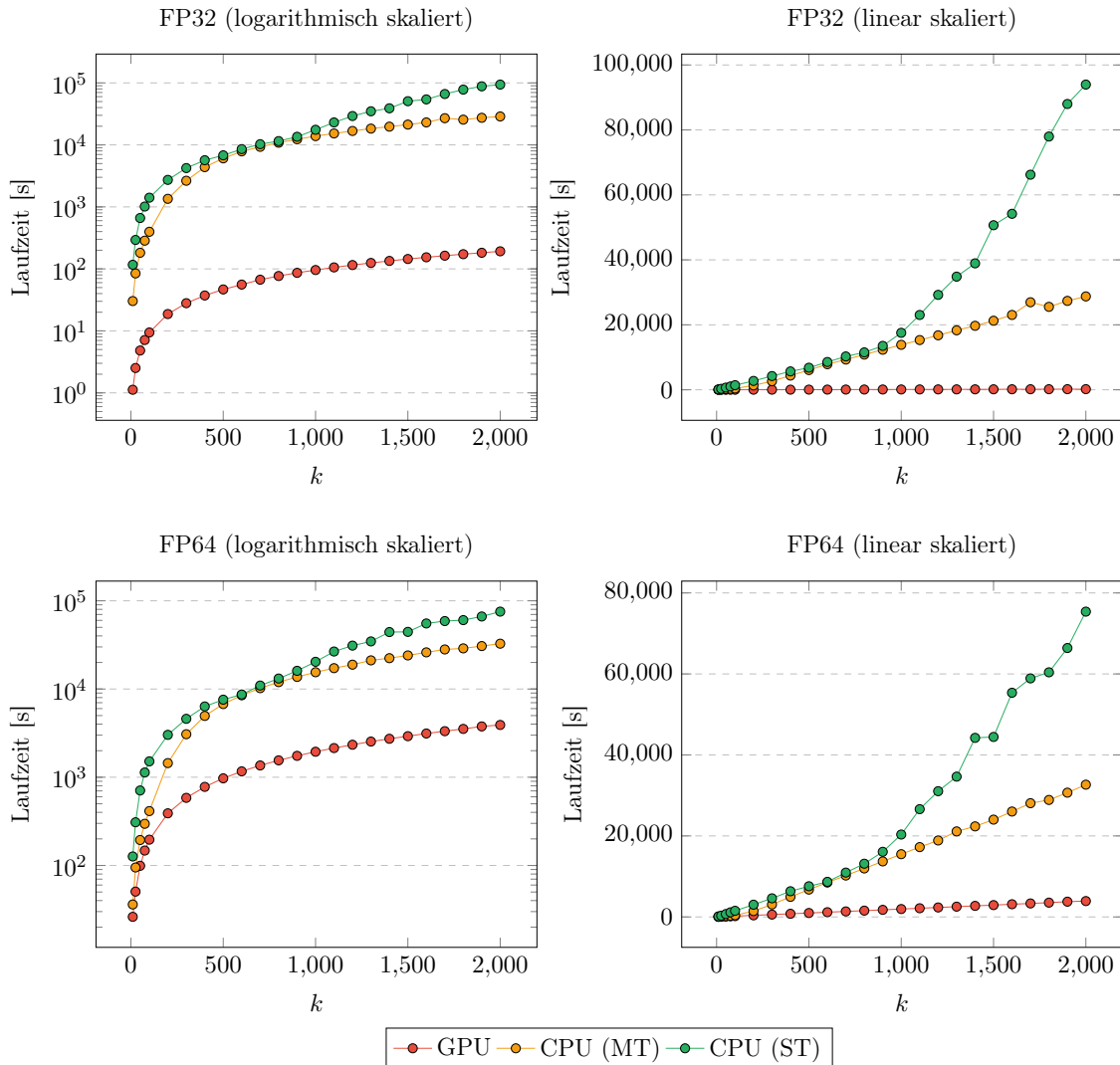


Abbildung 5.7: Erzielte Laufzeiten der submodularen Funktion des EXEMPLAR-BASED CLUSTERINGS auf Grundlage der einfädigen (ST) und mehrfädigen (MT) CPU-Implementierungen sowie der GPU-Implementierung unter Variierung der Anzahl der Elemente in jeder Evaluationsmenge $k \in \mathbb{N}$. Die Laufzeitskala ist logarithmisch skaliert.

Implementierung weist interessanterweise für den genannten Wertebereich $k \in [500, 900]$ eine niedrigere Laufzeit auf, als durch den Kurvenverlauf zu erwarten wäre. Für diesen Bereich nähert sich die Laufzeit der einfädigen Implementierung stark der Laufzeit der mehrfädigen Implementierung an, wenngleich sich bei der Mehrfädigkeit eine faktisch geringere Laufzeit beobachten lässt.

Diese Beobachtung lässt sich bei Betrachtung der Ergebnisse bei Nutzung doppelter Präzision in besonderem Maße wiederholen. In diesem Falle ist es so, dass die Laufzeitdifferenz für $k = 600$ zwischen den CPU-Algorithmen nur noch etwa 174 Sekunden beträgt. Entfernt lässt sich dies auch durch den erzielten Speedup der mehrfädigen CPU-

Implementierung beobachten, der bei Faktor 2,04 liegt und damit kleiner als bei Nutzung einfacher Präzision ist. Unterstrichen wird dies außerdem durch die kleinere Standardabweichung, die bei Faktor 0,88 liegt. Die mehrfädige CPU-Implementierung ist dennoch in allen getesteten Szenarien schneller als die einfädige Implementierung, während die GPU-Implementierung das Problem schneller als beide CPU-Implementierungen lösen konnte. Der Speedup der GPU-Implementierung verkleinert sich gegenüber der einfachen Präzision auf Faktor 11,2 mit einer Standardabweichung von Faktor 4,41.

5.2.3 Diskussion

Im vergangenen Abschnitt wurden Laufzeitergebnisse zu der Variation laufzeitkritischer Parameter bei der Auswertung der submodularen Funktion des EXEMPLAR-BASED CLUSTERINGS präsentiert. Hierbei fiel auf, dass die Laufzeit der GPU-Implementierung in den untersuchten Szenarien linear mit steigender Problemgröße steigt und das Problem gegenüber den genutzten CPU-Implementierung oftmals am schnellsten lösen konnte. Die Größe des erreichbaren Speedups hing dabei äußerst stark von der benutzten Präzision bei der Gleitkommaarithmetik ab: So erzielte die GPU-Implementierung in allen hier getesteten Variationsszenarien bei Nutzung einfacher Präzision erhebliche Speedups von mindestens Faktor 150, die jedoch bei Nutzung doppelter Präzision nicht mehr beobachtet werden konnten. Eine Erklärung hierfür ist, dass die *Streaming Multiprocessors* (SM) einer GPU der Pascal-Mikroarchitektur lediglich halb so viele Doppelpräzisions- wie Einfachpräzisions-Berechnungskerne zu Verfügung stellen. Des Weiteren müssen bei Nutzung doppelter Präzision auch doppelt so viele Nutz- und Ergebnisdaten über die PCI-E-Verbindung zwischen CPU und GPU kopiert werden. Der Wechsel von einfacher Präzision zu doppelter Präzision führt auf der CPU zu keinen derart drastischen Performanceeinbußen. Für eine weitere Nutzung dieser submodularen Funktion wäre es daher interessant zu klären, ob und wie stark die Nutzung einfacher Präzision die Qualität der gewonnen Funktionswerte für den praktischen Betrieb beeinflusst. Sollte es erforderlich sein, dass doppelte Präzision zur Lösung des Problems herangezogen werden muss, dann könnte es nach Beurteilung der gemessenen Speedups entweder notwendig sein eine GPU mit höherer Doppelpräzisionsleistung einzusetzen oder aber auf die GPU-Implementierung zu verzichten und eine klassische CPU oder einen CPU-Verbund zu verwenden, die mehr Rechenkerne zu Verfügung stellen, um so den Speedup der mehrfädigen Implementierung weiter zu steigern.

Welche Strategie dabei sinnvoll ist, könnte dadurch festgestellt werden, dass der erreichbare Speedup mit dem Energieverbrauch ins Verhältnis gesetzt wird. So könnte festgestellt werden, welche Implementierung nicht nur schneller sondern auch (energie-)effizienter arbeitet. So liegt die thermische Verlustleistung (engl. *thermal design point*, TDP) der als Maß für den Energieverbrauch bei der Dimensionierung von entsprechenden Kühlösungen verwendet wird [29], für die hier verwendete GeForce GTX 1080 Ti bei 320 Watt,

die gerade bei der Problemlösung mit einfacher Präzision häufig auch ausweislich des entsprechenden NVIDIA-Tools erreicht wurden. Der i7-6700-Prozessor ist hingegen lediglich für eine TDP von 65 Watt spezifiziert. Gleichzeitig konnte die GPU in einigen Szenarien mit einfacher Präzision das Problem im Mittel aber 234-mal schneller lösen als die einfädige CPU-Implementierung. In den Szenarien doppelter Präzision, in denen die GPU deutlich langsamer als bei Nutzung einfacher Präzision ist, konnte aber auch ein geringerer Energieverbrauch durch die GPU beobachtet werden. Eine weitere Beobachtung dieser Problematik erfolgt an dieser Stelle nicht, sollte jedoch bei der Frage, welche Implementierung eingesetzt wird, nicht vernachlässigt werden.

Hinsichtlich des erreichten Speedups konnten für die mehrfädige CPU-Implementierung einige interessante Beobachtungen gemacht werden: Wie bereits in Abschnitt 5.2.1 geschildert, ist die verwendete CPU eine Vierkern-CPU. Ein Erklärungsansatz warum der durch die Mehrfädigkeit erzielte Speedup selbst in den besten Versuchsreihen kleiner als vier ist, liefert das Amdahlsche Gesetz [4], welches ein theoretisches Modell für die Berechnung des erzielbaren Speedups liefert: Kern dieses Modells ist die Dekomposition der Laufzeit eines Programms in einen sequentiellen Teil r_s und einen parallelen Teil r_p mit $r_s + r_p = 1$, wobei n die Anzahl zusätzlich zu Verfügung gestellter Ressourcen, wie beispielsweise CPU-Prozessoren, darstellt. Der Speedup ist dann nach *Amdahl* beschränkt durch $\frac{1}{r_s + \frac{r_p}{n}}$. Für die Experimente, die $|S_{\text{multi}}|$ und $|V|$ variieren, konnte festgestellt werden, dass der empirisch gemessene Speedup der Mehrfädigkeit ziemlich nah an dem durch die vier Prozessorkerne und $r_s = 0$ gegebenen Ideal vollständiger Parallelisierung liegt. Die doppelte Präzision scheint hier jeweils den sequentiellen Anteil leicht zu vergrößern, was sich in einem geringeren gemessenen Speedup bemerkbar macht. Die Variation des Parameters k scheint den sequentiellen Teil des Programms am stärksten zu vergrößern, da der gemessene Speedup lediglich bei etwa Faktor zwei liegt. Eine Verbesserung könnte erzielt werden, in dem die hier verwendeten (naiven) CPU-Implementierungen beispielsweise von SIMD-Funktionalitäten Gebrauch machen, die durch moderne Prozessoren zu Verfügung gestellt werden.

Eine weitere interessante Beobachtung konnte bei den Versuchen gemacht werden, in denen der Parameter k variiert wurde: Dort haben sich im Intervall $k \in [500, 900]$ die Laufzeiten der ein- und mehrfädigen Implementierung angenähert, bis diese für höhere k wieder auseinanderdrifteten und die mehrfädige CPU-Variante wieder deutlich schneller war (s. Abbildung 5.7). Für das angegebene Intervall lässt sich festhalten, dass der durch die mehrfädige Implementierung erreichte Speedup äußerst gering ist. Denkbar ist beispielsweise, dass es zu ungünstigen Cacheeffekten auf der CPU kommt, bei denen benötigte Daten häufiger als sonst von der nächstniedrigeren Ebene in der Speicherhierarchie angefordert werden. Interessant ist auch, dass für $k > 900$ der Kurvenverlauf die durch die asymptotische Laufzeitanalyse erwartete Linearität verlässt und für größer werdende k starke Nichtlinearitäten beobachtet werden konnten. Auch dies könnte ein Indikator für schlechte Cacheperformance sein. Insgesamt ließen sich bei den CPU-basierten Implementierungen

häufiger irreguläre Laufzeitmuster beobachten, bei denen einzelne Messungen deutlich mehr Zeit in Anspruch nahmen, als es bei anderen Messungen der Fall war. Dies konnte man dabei für gleiche $|V|$ der einfädigen und die mehrfädigen CPU-Implementierung beobachten, wobei der Effekt sich bei der einfädigen Implementierung weniger stark ausdrückt. Eine Erklärung könnten hier ebenfalls ungünstige Zugriffe auf Caches und Hauptspeicher sein. Zu erwähnen ist ebenfalls, dass die GPU dieses Verhalten nicht zeigte und die Laufzeit empirisch linear mit der Problemgröße wuchs. Dies lässt sich möglicherweise dadurch erklären, dass die GPU für die Lösung dieses Problems viele Threads gleichzeitig startet und bestimmte Threads bewusst verdrängt, falls diese gerade auf Daten warten, um andere (ausführbereite) Prozessfäden abzuarbeiten. Es werden also die Speicherzugriffslatenzen, die der CPU in einigen Szenarien mutmaßlich Probleme bereiten, versteckt.

Ein weiterer Punkt, der hier im Experimenterteil nicht näher untersucht wurde aber dennoch praxisrelevant ist, ist der, dass die Problemgröße nicht beliebig groß werden können und hardware-basierte Limitierung schneller erreicht werden als es bei CPU-basierten Algorithmen der Fall ist. Der stärkste limitierende Faktor bei GPUs ist der verfügbare Grafikspeicher, auf dem die Problemdaten abgelegt werden und der heutzutage oftmals nicht größer als 32GB ist. CPUs hingegen können mit deutlich größeren Arbeitsspeichern Probleme lösen, die nicht mehr auf die GPU geladen werden können. Abhilfe können beispielsweise Kompressionsstrategien schaffen, die die Datenmenge reduzieren jedoch die Gefahr irregulärer Zugriffsmuster birgt, die gerade auf GPUs teuer sind. Eine weitere Lösungsstrategie stellen Multi-GPU-Systeme dar, die eine Partitionierung des zugrundeliegenden Problems betrachten, was jedoch ein dafür ausgelegtes Algorithmen-Design verlangt. Darüber hinaus verstärken Multi-GPU-Systeme das Problem des höheren Energieverbrauchs, der beim Rechnen entsteht.

Kapitel 6

Fazit und Ausblick

In der vorliegenden Arbeit wurde die Thematik der Erkennung ausreißender Datenpunkte besprochen, wobei das praxisrelevante Szenario des Lernens ohne markierte Beobachtungen beleuchtet und insbesondere die Verwendung submodularer Funktionen, die einen individuellen Repräsentativitätsbegriff ausgewählter Beobachtungen implementieren, für diesen Einsatzzweck berücksichtigt wurde. Hierzu wurde in Kapitel 2 das Themenfeld dieser spezifischen Klasse von Funktionen detailliert betrachtet und erläutert, wie sich diese formal definieren lassen, wie die submodularen Funktionen `INFORMATIVE VECTOR MACHINE` und `EXEMPLAR-BASED CLUSTERING` die Repräsentativität einer Menge von Datenpunkten quantifizieren und wie eine Optimierung im Stapelbetrieb mittels des `GREEDY`-Algorithmus und im Datenstrombetrieb mittels des `SIEVE STREAMING`-Verfahrens möglich ist. Hierbei wurden insbesondere die Gütegarantien der Optimierer und die Speicherkomplexität, die für einen Einsatz unter Ressourcenbeschränkung von Bedeutung sind, herausgestellt. In Kapitel 3 wurden Definitionen des Begriffs *Ausreißer* vorgestellt und besprochen, wie sich die Lernaufgabe der Ausreißerererkennung ausdrücken lässt und wie dazugehörige Methoden strukturiert werden können. Anschließend wurden zwei exemplarische, klassische Verfahren zur Ausreißerererkennung, nämlich der `ISOLATION FOREST` und der `LOCAL OUTLIER FACTOR` vorgestellt, um anschließend die sogenannte `SUMMARY OUTLIER DETECTION` als submodulare Methode zu präsentieren. Hierzu wurde ergänzend das `RECURSIVE SUBMODULAR PARTITIONING`-Verfahren etabliert, welches anhand einer Hierarchie von Zusammenfassungen, die auf Grundlage verschiedener Datenpartitionen identifiziert wurden, eine genauere Ausreißerererkennung leisten soll. Sowohl klassische als auch submodulare Methoden zur Ausreißerererkennung wurden strukturell anhand ihres Bewertungsmechanismus in die vorgestellten Gruppen bestehender Verfahren eingeordnet. In Kapitel 4 wurde gezeigt, wie sich die zum Teil asymptotische große Zahl an Funktionsevaluierungen durch schnelle Implementierungen ausgleichen lassen: Hinsichtlich der `INFORMATIVE VECTOR MACHINE` wurde beleuchtet, wie sich Berechnungen einsparen lassen und wie die Eigenschaft der positiv-definiten Kernmatrix eine effizientere Berechnung der Determinante ermöglicht.

Für das EXEMPLAR-BASED CLUSTERING wurde erörtert, ob und welche Möglichkeiten zur Laufzeitreduzierung mittels des herkömmlichen Hauptprozessors vorhanden sind, welche Beschleunigungen erzielt werden können und wie Grafikprozessoren aufgrund ihrer Architektur das Problem massiv-parallel lösen können. Hierzu wurde ein spezifischer Algorithmus entwickelt, der beabsichtigt auf die technischen Details dieser Prozessoren abgestimmt zu sein und die Hardwaremerkmale bestmöglich ausnutzen soll. In Kapitel 5 wurde zunächst eine Experimentierreihe durchgeführt, deren Ziel es war, die Vorhersageleistung, gemessen durch den F1-Score und die ROC-AUC, für verschiedene Verfahren und Datensätze zu messen, um festzustellen, ob die submodularen Methoden zur Ausreißerererkennung zu den klassischen Methoden vergleichbare Vorhersagegüten und Laufzeiten bieten und ob grundsätzlich präferable Konfigurationen zur submodularen Ausreißerererkennung existieren. Hierbei zeigte sich, dass die SUMMARY OUTLIER DETECTION unabhängig davon, ob RSP-Bäume verwendet wurden oder nicht, verglichen mit den klassischen Verfahren ISOLATION FOREST und LOCAL OUTLIER FACTOR wettbewerbsfähige Vorhersagegüten hervorbrachten, die nur für einzelne Datensätze schlechtere Metriken lieferten. Gerade die Erweiterung des SOD-Verfahrens mit RSP-Bäumen lieferte in vielen Fällen eine gegenüber dem reinen SOD-Verfahren nochmals verbesserte Vorhersageleistung. Es stellte sich hierbei heraus, dass insbesondere eine reine Zufallswahl von Repräsentanten mit Dispersions- beziehungsweise Kernel-kontrollierenden Abbruchbedingungen zu guten Metriken führte. Unter jenen Verfahren, die eine submodulare Funktion zur Repräsentantenwahl heranziehen und keine Zufallsentscheidung treffen, schnitt insbesondere die INFORMATIVE VECTOR MACHINE gut ab, während das EXEMPLAR-BASED CLUSTERING, mutmaßlich aufgrund mangelnder Optimierung, häufig zu schlechteren Vorhersagegüten führte. Des Weiteren wurden die Hypothesen geprüft, ob die zunehmende Aufnahme von Beobachtungen und damit die verbundene Steigerung des Funktionswertes zu besseren Vorhersagegüten führt, was sich aber nicht eindeutig in dem hier beleuchteten Szenario empirisch zeigen ließ. Hingegen weitestgehend bestätigen ließ sich die Hypothese, dass die GREEDY-Optimierung bessere Funktionswerte gefolgt vom SIEVE STREAMING-Verfahren liefert. Hinsichtlich der Laufzeitbetrachtung ließ sich feststellen, dass gerade die submodulare Ausreißerererkennung mit RSP-Bäumen sehr kurze Ausführungsdauern hervorgebracht hat, die mit den klassischen Methoden vergleichbar sind, wobei zu beachten ist, dass diese aufgrund des größeren Parameterraums auch zeitintensiver zu optimieren sind. Hinsichtlich des SOD-Verfahrens ohne RSP-Bäume konnte für die Optimierer, die akzeptable Metriken produziert haben (also insbesondere *keine* Zufallsentscheidung verwenden), festgestellt werden, dass diese gerade für große Repräsentantenmengen deutlich mehr Zeit in Anspruch nehmen, ein über-lineares Wachstum mit der Kardinalität dieser Menge aufzeigen, aber oftmals schlechter abschnitten als die baumbasierte RSP-Variante. Gleichwohl gilt es hier zu beachten, dass der Parameterraum gegenüber SOD-RSP *kleiner* ist und daher weniger Zeit für die Modellsuche

verwendet werden muss. Zusammenfassend ließ sich hier sagen, dass die beobachtbaren Laufzeiten der in Hypothese drei formulierten Erwartungen gefolgt ist.

In einer zweiten Experimentierreihe wurde der durch die Implementierung auf einem Grafikprozessor erzielte Speedup des EXEMPLAR-BASED CLUSTERINGS untersucht: Es zeigte sich, dass gegenüber einem herkömmlichen, in Desktop-PCs zu findenden Hauptprozessor die Implementierung durchweg schneller als eine nicht-parallelisierte Implementierung arbeitete. In den meisten Fällen arbeitete die GPU-Implementierung auch schneller als eine CPU-mehrfädige Implementierung. Abhängig der verwendeten Gleitkommapräzision ließen sich Speedups von bis zu Faktor 489 erzielen. Es ließ sich des Weiteren feststellen, dass eine starke Disparität zwischen dem erzielbaren Speedup bei Verwendung einfacher und doppelter Präzision existiert, die letztlich zu dem Schluss führt, dass bei Verwendung der hier betrachteten Grafikkarte und doppelter Präzision der Einsatz einer reinen CPU-basierten Implementierung bei Verfügbarkeit einer höheren Zahl von Rechenkernen Sinn ergeben kann. Abzuwägen sind neben den erzielbaren Speedups vor allem die Energieverbräuche der jeweils verwendeten (Ko-)Prozessoren, die gerade bei Grafikprozessoren mitunter sehr hoch sein können, jedoch an dieser Stelle nicht weiter untersucht wurden.

Ausblick Die Experimentierreihe zur Ausreißerererkennung mit submodularen Funktionen lieferte vielversprechende Ergebnisse, die ganz grundsätzlich nahelegen, dass die Bewertung von Beobachtungen hinsichtlich ihrer Ausreißereigenschaft mit dieser Methodik sinnvoll ist: Eine Weiterentwicklung der hier besprochenen Techniken könnte daher dahingehend erfolgen, dass beispielsweise untersucht wird, wie sich die submodulare Ausreißerererkennung, abweichend von der hier experimentell evaluierten Variante der Daten im Stapelbetrieb, als reines Datenstromverfahren umsetzen lässt. So wäre es beispielsweise notwendig, dass in der SUMMARY OUTLIER DETECTION (SOD) bei der Auswahl und Aktualisierung der Repräsentanten auch die entsprechende Dichteschätzung mittels der Zählvariablen aktualisiert oder mindestens verworfen wird. Weiterhin interessant wäre es zu verstehen, warum die SOD mit RSP-Bäumen gerade mit einer zufälligen Auswahl von Repräsentanten besonders gute Vorhersagegüten liefert. So ist zu erwarten, dass durch die zufallsbasierte Partitionierung in den jeweiligen Baumknoten die Abbruchbedingung erst deutlich tiefer im Baum greift, was zu tendenziell größeren Bäumen führt, die abgespeichert werden müssen. Für ressourcenbeschränkte Systeme kann dies eine ernstzunehmende Barriere darstellen. Idealerweise werden durch die Optimierung der submodularen Funktion Repräsentanten ausgewählt, die derart divers sind, dass die Dispersion beziehungsweise die mittleren Kernwerte der gefundenen Partitionen bereits initial sehr klein sind und somit zum einen bedeutsame Regionen und zum anderen kleine Bäume identifiziert werden, die genügsam im Speicherbedarf sind und bessere Vorhersagegüten liefern. Da dies hier nicht der Fall gewesen ist, könnte eine Untersuchung dieses Phänomens die Vorhersagegüten des präsentierten SOD-RSP-Verfahrens nochmals verbessern. Des Weiteren wäre es interessant

zu sehen, wie sich die SUMMARY OUTLIER DETECTION verhält, wenn nicht ein einzelner Baum des RECURSIVE SUBMODULAR PARTITIONING verwendet sondern ein mittels eines *Bagging*-Ensembles trainierter Wald von RSP-Bäumen herangezogen wird. In diesem Falle würden die Ausreißerentscheidungen, die jeweils auf Grundlage eines Baums getroffen wurden, *Bagging*-typisch zwischen den Bäumen des Walds gemittelt werden.

Hinsichtlich der submodularen Funktion des EXEMPLAR-BASED CLUSTERINGS wäre es weiterhin interessant zu verstehen, wie sich die Wahl der Gleitkommapräzision auf die Qualität ausgewählter Zusammenfassungen und damit auf die Ausreißererkenntnis auswirkt, da dies die Geschwindigkeit der Ausführung auf der GPU deutlich determiniert. Weiterhin wäre die Nutzung alternativer Distanzmaße ein interessantes Untersuchungsobjekt: So ist diese Funktion, wie bereits beschrieben, äußerst liberal im Umgang mit Distanzfunktionen sodass ausschließlich Nichtnegativität verlangt wird. Denkbar wäre es daher diesen Parameter der Distanzfunktion intensiver zu optimieren und beispielsweise auch Kernfunktionen zu verwenden, die bereits in der Formulierung der INFORMATIVE VECTOR MACHINE zum tragen kommt und dort akzeptable Ergebnisse hervorgebracht hat.

Anhang A

Ausreißerererkennung

III

An dieser Stelle werden die aggregierten und selektierten Ergebnistabellen der in Abschnitt 5.1 diskutierten Experimente präsentiert. Hierbei werden zwei Tabellen präsentiert, die, entsprechend den in Abschnitt 5.1.1 dargelegten Postprocessing-Schritten mit dem Ziel der Maximierung des F1-Score und der ROC-AUC-Metrik erzeugt wurden.

A.0.1 Maximierung des F1-Scores

Datensatz	Methode	Funktion	Kernel	Sigma	k	Abbruchbedingung	minPts	t	F1-Score	Laufzeit [s]	f(S)
Anthyroid	Greedy (SOD)	EBC	-	-	10	-	-	-	0.040524	3.31	1.11943
Anthyroid	Greedy (SOD)	IVM	$k_{\text{Matérn}}^{\nu=1/2}$	0.436436	40	-	-	-	0.072562	3.97	13.8617
Anthyroid	Greedy (SOD-RSP)	EBC	-	-	5	Maximale Baumhöhe	-	-	0.064068	1.39	-
Anthyroid	Greedy (SOD-RSP)	IVM	k_{RBF}	0.109109	12	Mittlere Kernfunktion	-	-	0.119338	2.25	-
Anthyroid	Isolation Forest	-	-	-	0	-	-	128	0.210657	0.27	-
Anthyroid	Local Outlier Factor	-	-	-	0	-	5	-	0.129383	0.34	-
Anthyroid	Random (SOD)	EBC	-	-	80	-	-	-	0.046360	0.12	1.18153

Fortsetzung auf nächster Seite

Datensatz	Methode	Funktion	Kernel	Sigma	k	Abbruchbedingung	$minPts$	t	F1-Score	Laufzeit [s]	$f(S)$
Amnthyroid	Random (SOD)	IVM	$k_{Matern}^{\nu=1/2}$	0.109109	90	-	-	-	0.063598	0.12	26.3053
Amnthyroid	Random (SOD-RSP)	EBC	-	-	3	Mittlere Kernfunktion	-	-	0.322581	0.01	-
Amnthyroid	Random (SOD-RSP)	IVM	$k_{Matern}^{\nu=1/2}$	2.18218	5	Mittlere Kernfunktion	-	-	0.239067	0.01	-
Amnthyroid	Sieve Streaming (SOD)	EBC	-	-	60	-	-	-	0.039691	411.18	1.25028
Amnthyroid	Sieve Streaming (SOD)	IVM	$k_{Matern}^{\nu=1/2}$	0.00218218	80	-	-	-	0.049225	0.22	27.7259
Cardiotocography	Greedy (SOD)	EBC	-	-	40	-	-	-	0.104917	1.78	2.73728
Cardiotocography	Greedy (SOD)	IVM	$k_{Matern}^{\nu=1/2}$	0.109109	70	-	-	-	0.274510	4.46	24.2601
Cardiotocography	Greedy (SOD-RSP)	EBC	-	-	11	Maximale Baumhöhe	-	-	0.156425	0.21	-
Cardiotocography	Greedy (SOD-RSP)	IVM	k_{RBF}	0.163663	3	Maximale Baumhöhe	-	-	0.327869	0.01	-
Cardiotocography	Isolation Forest	-	-	-	0	-	-	128	0.166990	0.16	-
Cardiotocography	Local Outlier Factor	-	-	-	0	-	5	-	0.249773	0.06	-
Cardiotocography	Random (SOD)	EBC	-	-	80	-	-	-	0.074841	0.03	2.72538
Cardiotocography	Random (SOD)	IVM	$k_{Matern}^{\nu=5/2}$	0.436436	80	-	-	-	0.133905	0.03	21.9334
Cardiotocography	Random (SOD-RSP)	EBC	-	-	5	Mittlere Kernfunktion	-	-	0.351648	0.00	-
Cardiotocography	Random (SOD-RSP)	IVM	$k_{Matern}^{\nu=1/2}$	2.72772	4	Streuung in Partition	-	-	0.353556	0.00	-
Cardiotocography	Sieve Streaming (SOD)	EBC	-	-	100	-	-	-	0.143123	43.28	2.71676
Cardiotocography	Sieve Streaming (SOD)	IVM	$k_{Matern}^{\nu=1/2}$	0.00218218	30	-	-	-	0.237063	0.02	10.3972
PageBlocks	Greedy (SOD)	EBC	-	-	100	-	-	-	0.197268	46.13	0.819361
PageBlocks	Greedy (SOD)	IVM	k_{RBF}	3.16228	10	-	-	-	0.421456	0.10	1.6624
PageBlocks	Greedy (SOD-RSP)	EBC	-	-	14	Mittlere Kernfunktion	-	-	0.433213	2.17	-
PageBlocks	Greedy (SOD-RSP)	IVM	$k_{Matern}^{\nu=3/2}$	0.237171	5	Streuung in Partition	-	-	0.484305	0.06	-
PageBlocks	Isolation Forest	-	-	-	0	-	-	128	0.232498	0.20	-
PageBlocks	Local Outlier Factor	-	-	-	0	-	20	-	0.391234	0.09	-
PageBlocks	Random (SOD)	EBC	-	-	100	-	-	-	0.082862	0.10	0.810092
PageBlocks	Random (SOD)	IVM	$k_{Matern}^{\nu=1/2}$	0.158114	10	-	-	-	0.121786	0.01	3.09359
PageBlocks	Random (SOD-RSP)	EBC	-	-	2	Mittlere Kernfunktion	-	-	0.488479	0.01	-
PageBlocks	Random (SOD-RSP)	IVM	$k_{Matern}^{\nu=5/2}$	0.158114	2	Streuung in Partition	-	-	0.516854	0.00	-
PageBlocks	Sieve Streaming (SOD)	EBC	-	-	100	-	-	-	0.124177	119.25	0.808324
PageBlocks	Sieve Streaming (SOD)	IVM	$k_{Matern}^{\nu=1/2}$	2.37171	100	-	-	-	0.381188	329.21	6.0919
PenDigits	Greedy (SOD)	EBC	-	-	100	-	-	-	0.014940	262.17	58673.7

Fortsetzung auf nächster Seite

Datensatz	Methode	Funktion	Kernel	Sigma	k	Abbruchbedingung	minPts	t	F1-Score	Laufzeit [s]	f(S)
PenDigits	Greedy (SOD)	IVM	$k_{\text{Matém}}^{\nu=1/2}$	0.0025	50	-	-	-	0.050021	6.52	17.3287
PenDigits	Greedy (SOD-RSP)	EBC	-	-	12	Streuung in Partition	-	-	0.032077	8.86	-
PenDigits	Greedy (SOD-RSP)	IVM	k_{RBF}	0.25	2	Streuung in Partition	-	-	0.090361	0.13	-
PenDigits	Isolation Forest	-	-	-	0	-	-	128	0.009101	0.44	-
PenDigits	Local Outlier Factor	-	-	-	0	-	50	-	0.091669	1.02	-
PenDigits	Random (SOD)	EBC	-	-	80	-	-	-	0.010269	0.16	57383
PenDigits	Random (SOD)	IVM	$k_{\text{Matém}}^{\nu=5/2}$	0.0025	90	-	-	-	0.015862	0.17	31.1916
PenDigits	Random (SOD-RSP)	EBC	-	-	4	Streuung in Partition	-	-	0.119816	0.02	-
PenDigits	Random (SOD-RSP)	IVM	$k_{\text{Matém}}^{\nu=3/2}$	1.875	9	Streuung in Partition	-	-	0.218487	0.03	-
PenDigits	Sieve Streaming (SOD)	EBC	-	-	30	-	-	-	0.027240	307.32	56173.9
PenDigits	Sieve Streaming (SOD)	IVM	$k_{\text{Matém}}^{\nu=1/2}$	0.0025	50	-	-	-	0.050021	0.11	17.3287
Spambase	Greedy (SOD)	EBC	-	-	80	-	-	-	0.092498	41.11	0.158353
Spambase	Greedy (SOD)	IVM	k_{RBF}	0.132453	20	-	-	-	0.339181	0.28	6.93147
Spambase	Greedy (SOD-RSP)	EBC	-	-	15	Maximale Baumhöhe	-	-	0.083817	1.86	-
Spambase	Greedy (SOD-RSP)	IVM	$k_{\text{Matém}}^{\nu=1/2}$	0.00132453	4	Maximale Baumhöhe	-	-	0.440367	0.02	-
Spambase	Isolation Forest	-	-	-	0	-	-	1024	0.144928	1.22	-
Spambase	Local Outlier Factor	-	-	-	0	-	10	-	0.106927	0.44	-
Spambase	Random (SOD)	EBC	-	-	100	-	-	-	0.080004	0.09	0.120004
Spambase	Random (SOD)	IVM	$k_{\text{Matém}}^{\nu=1/2}$	0.00132453	80	-	-	-	0.108304	0.05	27.72
Spambase	Random (SOD-RSP)	EBC	-	-	3	Mittlere Kernfunktion	-	-	0.455446	0.01	-
Spambase	Random (SOD-RSP)	IVM	$k_{\text{Matém}}^{\nu=3/2}$	0.132453	2	Streuung in Partition	-	-	0.456140	0.01	-
Spambase	Sieve Streaming (SOD)	EBC	-	-	100	-	-	-	0.115601	306.77	0.146875
Spambase	Sieve Streaming (SOD)	IVM	k_{RBF}	1.65567	10	-	-	-	0.291253	0.01	1.67125
Waveform	Greedy (SOD)	EBC	-	-	90	-	-	-	0.253865	35.60	5.41785
Waveform	Greedy (SOD)	IVM	$k_{\text{Matém}}^{\nu=1/2}$	0.109109	100	-	-	-	0.257848	18.74	34.6573
Waveform	Greedy (SOD-RSP)	EBC	-	-	3	Maximale Baumhöhe	-	-	0.354062	0.24	-
Waveform	Greedy (SOD-RSP)	IVM	$k_{\text{Matém}}^{\nu=3/2}$	0.00218218	2	Streuung in Partition	-	-	0.524590	0.44	-
Waveform	Isolation Forest	-	-	-	0	-	-	512	0.108108	0.76	-
Waveform	Local Outlier Factor	-	-	-	0	-	5	-	0.003883	0.33	-
Waveform	Random (SOD)	EBC	-	-	90	-	-	-	0.083011	0.07	5.36699
Waveform	Random (SOD)	IVM	$k_{\text{Matém}}^{\nu=1/2}$	1.63663	30	-	-	-	0.130945	0.02	4.07645

Fortsetzung auf nächster Seite

Datensatz	Methode	Funktion	Kernel	Sigma	k	Abbruchbedingung	$minPts$	t	F1-Score	Laufzeit [s]	$f(S)$
Waveform	Random (SOD-RSP)	EBC	-	-	5	Mittlere Kernfunktion	-	-	0.545455	0.01	-
Waveform	Random (SOD-RSP)	IVM	$k_{Matern}^{\nu=5/2}$	1.63663	3	Streuung in Partition	-	-	0.539130	0.00	-
Waveform	Sieve Streaming (SOD)	EBC	-	-	20	-	-	-	0.185960	37.26	5.29895
Waveform	Sieve Streaming (SOD)	IVM	$k_{Matern}^{\nu=1/2}$	2.18218	20	-	-	-	0.266304	0.79	2.79876
Mammography	Greedy (SOD)	EBC	-	-	10	-	-	-	0.184681	2.86	0.395591
Mammography	Greedy (SOD)	IVM	$k_{Matern}^{\nu=1/2}$	0.816497	30	-	-	-	0.271078	1.70	6.24925
Mammography	Greedy (SOD-RSP)	EBC	-	-	9	Maximale Baumhöhe	-	-	0.400000	2.67	-
Mammography	Greedy (SOD-RSP)	IVM	$k_{Matern}^{\nu=1/2}$	0.816497	11	Maximale Baumhöhe	-	-	0.464768	0.29	-
Mammography	Isolation Forest	-	-	-	0	-	-	128	0.222222	0.37	-
Mammography	Local Outlier Factor	-	-	-	0	-	100	-	0.194595	0.29	-
Mammography	Random (SOD)	EBC	-	-	10	-	-	-	0.115180	0.02	0.382283
Mammography	Random (SOD)	IVM	k_{RBF}	4.08248	20	-	-	-	0.161892	0.03	1.58953
Mammography	Random (SOD-RSP)	EBC	-	-	4	Mittlere Kernfunktion	-	-	0.575517	0.01	-
Mammography	Random (SOD-RSP)	IVM	$k_{Matern}^{\nu=3/2}$	3.06186	2	Streuung in Partition	-	-	0.599379	0.01	-
Mammography	Sieve Streaming (SOD)	EBC	-	-	60	-	-	-	0.131816	214.26	0.39952
Mammography	Sieve Streaming (SOD)	IVM	$k_{Matern}^{\nu=1/2}$	2.04124	10	-	-	-	0.418741	0.03	1.57099

A.0.2 Maximierung der ROC-AUC

Datensatz	Methode	Funktion	Kernel	Sigma	k	Abbruchbedingung	$mimPts$	t	ROC-AUC	Laufzeit [s]	$f(S)$
Amnthyroid	Greedy (SOD)	EBC	-	-	20	-	-	-	0.496753	9.11	1.2081
Amnthyroid	Greedy (SOD)	IVM	$k_{Matern}^{\nu=1/2}$	1.63663	100	-	-	-	0.581449	42.21	20.4714
Amnthyroid	Greedy (SOD-RSP)	EBC	-	-	13	Mittlere Kernfunktion	-	-	0.581987	21.40	-
Amnthyroid	Greedy (SOD-RSP)	IVM	k_{RBF}	0.109109	12	Mittlere Kernfunktion	-	-	0.669786	2.23	-
Amnthyroid	Isolation Forest	-	-	-	0	-	-	128	0.835854	0.26	-
Amnthyroid	Local Outlier Factor	-	-	-	0	-	5	-	0.736190	0.35	-
Amnthyroid	Random (SOD)	EBC	-	-	90	-	-	-	0.533925	0.13	1.19054
Amnthyroid	Random (SOD)	IVM	$k_{Matern}^{\nu=5/2}$	0.163663	100	-	-	-	0.609930	0.14	25.8157
Amnthyroid	Random (SOD-RSP)	EBC	-	-	4	Streuung in Partition	-	-	0.729482	0.01	-
Amnthyroid	Random (SOD-RSP)	IVM	k_{RBF}	1.63663	3	Streuung in Partition	-	-	0.762365	0.01	-
Amnthyroid	Sieve Streaming (SOD)	EBC	-	-	60	-	-	-	0.504986	411.18	1.25028
Amnthyroid	Sieve Streaming (SOD)	IVM	$k_{Matern}^{\nu=1/2}$	0.00218218	70	-	-	-	0.548028	0.16	24.2601
Cardiotocography	Greedy (SOD)	EBC	-	-	30	-	-	-	0.717720	1.07	2.71784
Cardiotocography	Greedy (SOD)	IVM	$k_{Matern}^{\nu=1/2}$	0.00218218	10	-	-	-	0.850085	1.18	3.46574
Cardiotocography	Greedy (SOD-RSP)	EBC	-	-	9	Streuung in Partition	-	-	0.784109	0.15	-
Cardiotocography	Greedy (SOD-RSP)	IVM	$k_{Matern}^{\nu=1/2}$	0.218218	6	Mittlere Kernfunktion	-	-	0.900283	0.22	-
Cardiotocography	Isolation Forest	-	-	-	0	-	-	128	0.827146	0.16	-
Cardiotocography	Local Outlier Factor	-	-	-	0	-	5	-	0.853793	0.06	-
Cardiotocography	Random (SOD)	EBC	-	-	20	-	-	-	0.614861	0.01	2.59818
Cardiotocography	Random (SOD)	IVM	k_{RBF}	1.09109	50	-	-	-	0.736233	0.02	6.8502
Cardiotocography	Random (SOD-RSP)	EBC	-	-	7	Mittlere Kernfunktion	-	-	0.902278	0.01	-
Cardiotocography	Random (SOD-RSP)	IVM	k_{RBF}	1.63663	8	Mittlere Kernfunktion	-	-	0.907353	0.01	-
Cardiotocography	Sieve Streaming (SOD)	EBC	-	-	20	-	-	-	0.773265	16.53	2.60489
Cardiotocography	Sieve Streaming (SOD)	IVM	k_{RBF}	1.09109	30	-	-	-	0.842081	0.02	5.36764
PageBlocks	Greedy (SOD)	EBC	-	-	100	-	-	-	0.874295	46.13	0.819361
PageBlocks	Greedy (SOD)	IVM	k_{RBF}	0.316228	100	-	-	-	0.896551	31.44	26.0188
PageBlocks	Greedy (SOD-RSP)	EBC	-	-	15	Streuung in Partition	-	-	0.852645	2.25	-
PageBlocks	Greedy (SOD-RSP)	IVM	$k_{Matern}^{\nu=5/2}$	0.237171	11	Mittlere Kernfunktion	-	-	0.938913	0.63	-

Fortsetzung auf nächster Seite

Datensatz	Methode	Funktion	Kernel	Sigma	k	Abbruchbedingung	$minPts$	t	ROC-AUC	Laufzeit [s]	$f(S)$
PageBlocks	Isolation Forest	-	-	-	0	-	-	1024	0.928542	1.62	-
PageBlocks	Local Outlier Factor	-	-	-	0	-	100	-	0.957243	0.20	-
PageBlocks	Random (SOD)	EBC	-	-	20	-	-	-	0.642007	0.02	0.794658
PageBlocks	Random (SOD)	IVM	$k_{Matern}^{\nu=3/2}$	0.158114	20	-	-	-	0.791880	0.02	5.78992
PageBlocks	Random (SOD-RSP)	EBC	-	-	8	Mittlere Kernfunktion	-	-	0.898233	0.09	-
PageBlocks	Random (SOD-RSP)	IVM	$k_{Matern}^{\nu=5/2}$	0.158114	2	Mittlere Kernfunktion	-	-	0.927886	0.22	-
PageBlocks	Sieve Streaming (SOD)	EBC	-	-	30	-	-	-	0.780108	68.73	0.802909
PageBlocks	Sieve Streaming (SOD)	IVM	$k_{Matern}^{\nu=1/2}$	4.74342	100	-	-	-	0.920102	230.94	3.74426
PenDigits	Greedy (SOD)	EBC	-	-	100	-	-	-	0.892930	262.17	58673.7
PenDigits	Greedy (SOD)	IVM	$k_{Matern}^{\nu=1/2}$	0.0025	90	-	-	-	0.996127	53.89	31.1916
PenDigits	Greedy (SOD-RSP)	EBC	-	-	3	Streuung in Partition	-	-	0.998510	6.73	-
PenDigits	Greedy (SOD-RSP)	IVM	$k_{Matern}^{\nu=3/2}$	0.0025	5	Streuung in Partition	-	-	0.999914	0.52	-
PenDigits	Isolation Forest	-	-	-	0	-	-	128	0.818260	0.46	-
PenDigits	Local Outlier Factor	-	-	-	0	-	20	-	0.977569	0.60	-
PenDigits	Random (SOD)	EBC	-	-	90	-	-	-	0.724673	0.18	57511.9
PenDigits	Random (SOD)	IVM	$k_{Matern}^{\nu=1/2}$	0.125	100	-	-	-	0.919117	0.28	34.6573
PenDigits	Random (SOD-RSP)	EBC	-	-	14	Streuung in Partition	-	-	0.996824	0.44	-
PenDigits	Random (SOD-RSP)	IVM	$k_{Matern}^{\nu=1/2}$	0.0025	7	Streuung in Partition	-	-	0.998599	0.22	-
PenDigits	Sieve Streaming (SOD)	EBC	-	-	80	-	-	-	0.994206	638.57	57588.8
PenDigits	Sieve Streaming (SOD)	IVM	$k_{Matern}^{\nu=1/2}$	0.0025	90	-	-	-	0.996127	0.40	31.1916
Spambase	Greedy (SOD)	EBC	-	-	80	-	-	-	0.641936	41.11	0.158353
Spambase	Greedy (SOD)	IVM	$k_{Matern}^{\nu=1/2}$	0.00132453	80	-	-	-	0.909472	7.35	27.7259
Spambase	Greedy (SOD-RSP)	EBC	-	-	9	Maximale Baumhöhe	-	-	0.627189	1.21	-
Spambase	Greedy (SOD-RSP)	IVM	$k_{Matern}^{\nu=1/2}$	0.00132453	14	Maximale Baumhöhe	-	-	0.876447	0.30	-
Spambase	Isolation Forest	-	-	-	0	-	-	512	0.823286	0.64	-
Spambase	Local Outlier Factor	-	-	-	0	-	50	-	0.828998	0.51	-
Spambase	Random (SOD)	EBC	-	-	100	-	-	-	0.622642	0.09	0.120004
Spambase	Random (SOD)	IVM	k_{RBF}	1.65567	60	-	-	-	0.707564	0.04	3.62119
Spambase	Random (SOD-RSP)	EBC	-	-	2	Mittlere Kernfunktion	-	-	0.836098	0.00	-
Spambase	Random (SOD-RSP)	IVM	$k_{Matern}^{\nu=1/2}$	0.993399	5	Mittlere Kernfunktion	-	-	0.853500	0.01	-
Spambase	Sieve Streaming (SOD)	EBC	-	-	100	-	-	-	0.729101	306.77	0.146875

Fortsetzung auf nächster Seite

Datensatz	Methode	Funktion	Kernel	Sigma	k	Abbruchbedingung	$minPts$	t	ROC-AUC	Laufzeit [s]	$f(S)$
Spambase	Sieve Streaming (SOD)	IVM	$k_{Matern}^{\nu=1/2}$	0.662266	80	-	-	-	0.918267	0.26	14.8147
Waveform	Greedy (SOD)	EBC	-	-	80	-	-	-	0.614711	28.41	5.41438
Waveform	Greedy (SOD)	IVM	$k_{Matern}^{\nu=1/2}$	0.109109	100	-	-	-	0.852271	19.44	34.6573
Waveform	Greedy (SOD-RSP)	EBC	-	-	2	Maximale Baumhöhe	-	-	0.839719	0.15	-
Waveform	Greedy (SOD-RSP)	IVM	$k_{Matern}^{\nu=1/2}$	0.00218218	6	Streuung in Partition	-	-	0.886472	0.40	-
Waveform	Isolation Forest	-	-	-	0	-	-	512	0.720629	0.75	-
Waveform	Local Outlier Factor	-	-	-	0	-	100	-	0.750883	0.41	-
Waveform	Random (SOD)	EBC	-	-	80	-	-	-	0.571838	0.06	5.3604
Waveform	Random (SOD)	IVM	$k_{Matern}^{\nu=1/2}$	1.63663	30	-	-	-	0.686481	0.02	4.07645
Waveform	Random (SOD-RSP)	EBC	-	-	5	Mittlere Kernfunktion	-	-	0.881532	0.01	-
Waveform	Random (SOD-RSP)	IVM	$k_{Matern}^{\nu=1/2}$	2.18218	6	Mittlere Kernfunktion	-	-	0.882707	0.01	-
Waveform	Sieve Streaming (SOD)	EBC	-	-	100	-	-	-	0.752689	107.08	5.3888
Waveform	Sieve Streaming (SOD)	IVM	k_{RBF}	3.27327	70	-	-	-	0.839065	50.13	3.70168
Mammography	Greedy (SOD)	EBC	-	-	20	-	-	-	0.751883	6.74	0.400378
Mammography	Greedy (SOD)	IVM	k_{RBF}	0.204124	80	-	-	-	0.726666	37.49	23.2001
Mammography	Greedy (SOD-RSP)	EBC	-	-	9	Maximale Baumhöhe	-	-	0.870281	2.67	-
Mammography	Greedy (SOD-RSP)	IVM	$k_{Matern}^{\nu=1/2}$	2.04124	9	Maximale Baumhöhe	-	-	0.872270	0.43	-
Mammography	Isolation Forest	-	-	-	0	-	-	128	0.826551	0.37	-
Mammography	Local Outlier Factor	-	-	-	0	-	100	-	0.827604	0.29	-
Mammography	Random (SOD)	EBC	-	-	10	-	-	-	0.664952	0.02	0.382283
Mammography	Random (SOD)	IVM	$k_{Matern}^{\nu=1/2}$	4.08248	30	-	-	-	0.783689	0.05	1.88568
Mammography	Random (SOD-RSP)	EBC	-	-	10	Mittlere Kernfunktion	-	-	0.873754	0.03	-
Mammography	Random (SOD-RSP)	IVM	$k_{Matern}^{\nu=3/2}$	0.204124	13	Mittlere Kernfunktion	-	-	0.891947	0.03	-
Mammography	Sieve Streaming (SOD)	EBC	-	-	50	-	-	-	0.657020	192.13	0.398499
Mammography	Sieve Streaming (SOD)	IVM	k_{RBF}	0.408248	60	-	-	-	0.838267	1.72	8.77544

Abbildungsverzeichnis

1.1	Ein Beispieldatensatz aus dem mittels der Optimierung einer submodularen Funktion jeweils zwei, drei und vier möglichst diverse Repräsentanten (rot markiert) ausgewählt wurden. Die Flächen zeigen die Ähnlichkeit zu ihrem nächsten Repräsentanten an und stehen hier für ein Maß, dass den gemeinsam erklärten Datenraum angibt. Grün bis gelbe Flächen stehen dabei für tendenziell gut erklärte Regionen, während rot-orange Flächen für schlecht erklärte Teile des Datenraums stehen. Es ist erkennbar, dass durch die zusätzliche Auswahl an Repräsentanten die erklärte Fläche zunimmt.	2
2.1	Kernmechanismus des SIEVE STREAMINGS für die Optimierung von submodularen Funktionen unter Kardinalitätsbeschränkung mit $k = 4$: Ein Strom von Beobachtungen wird mittels eines Multiplikatorelements („*“) an drei verschiedene <i>Siebe</i> mit den individuellen Grenzwerten v_1, v_2 und v_3 verteilt. Diese nehmen eine spezifische Beobachtung auf, sofern der Grenznutzen einen bestimmten Wert überschreitet und das Sieb noch nicht voll ist. Wenn alle Siebe gefüllt oder der Beobachtungsstrom terminiert, wird mittels einer „arg max“-Operation ermittelt, welches Sieb den Funktionswert maximiert. Die Details sind in Algorithmus 2.3 und 2.4 dargelegt.	12
3.1	Visualisierung der beiden ausreißergenerierenden Mechanismen nach <i>Hawkins</i> gegeben einer Zufallsvariable \mathcal{X} und der dazugehörigen Verteilung $\varphi(\mathcal{X})$. (a) Es existiert lediglich eine Wahrscheinlichkeitsverteilung, wobei die dazugehörige Dichtefunktion sich an den Rändern nur langsam dem Wert null nähert. (b) Es existiert eine Verteilung, die <i>gute</i> Beobachtungen produziert (grün) und eine Verteilung, die <i>schlechte</i> Beobachtungen produziert (rot).	24
3.2	Ein zweidimensionaler Beispieldatensatz, der aus zwei Clustern C_1 und C_2 nicht-ausreißender Punkte (grün) und zwei Ausreißern o_1 und o_2 besteht (Abbildung angelehnt an [8]).	35

- 3.3 Darstellung der einfachen und rekursiven Partitionierungsstrategie bei Nutzung einer jeweils identisch konfigurierten submodularen Funktion (EXEMPLAR-BASED CLUSTERING, vgl. Abschnitt 2.3.2), einer Zusammenfassungsgröße von $k = 4$ und des GREEDY-Optimierers. **a)** Es wird eine Zusammenfassung gefunden, die jeweils einen Repräsentanten für jeden der vier visuell vorhandenen Cluster vorhält. Entsprechend werden vier Partitionen, erkennbar durch die eingezeichneten Grenzen, identifiziert. **b)** Aufbauend auf der einfachen Partitionierung wird in den jeweiligen Regionen eine erneute Partitionierung durchgeführt, die zur Identifizierung weiterer Subgruppen führt. Diese könnten beispielsweise im industriellen Umfeld als eine (Sub-)Gruppe von Maschinenstörungen interpretiert werden. 43
- 4.1 Schematische Darstellung der Integration einer GPU in ein Zweikern-CPU-System mit einer sogenannten *Northbridge* über eine PCI-Express (PCI-E)-Verbindung [64]. 57
- 4.2 Streaming Multiprocessor (SM) einer GP100-GPU des Herstellers NVIDIA (eigene Darstellung nach [15]). 58
- 4.3 Aus einer Kernelkonfiguration $C = (D_g, D_b)$ resultierende Gitter-Block-Struktur mit $D_g = (g_x = 3, g_y = 2, g_z = 1)$ und $D_b = (b_x = 4, b_y = 3, b_z = 1)$, wie sie zur Berechnung eines GPU-Kernels herangezogen werden könnte (eigene Darstellung nach [16]). 60
- 4.4 Beispielhafte Arbeitsmatrix A zur Lösung eines Problems des EXEMPLAR-BASED CLUSTERINGS mit $V = \{v_1, \dots, v_4\}$ und $S_{\text{multi}} = \{S_1, \dots, S_4\}$ mit dazu berechneter Kernelkonfiguration. Alle Spalten der Matrix A werden in jeweils einem Block (gelb) von vier Threads gruppiert, wobei jeder Thread des Blocks genau eine Zelle bearbeitet. 62
- 4.5 Beispielhafter GPU-Speicherbereich bestehend aus vier Segmenten zu je 32 Byte. **(a)** Es werden vier 8 Byte-Wörter aus einem Speichersegment des globalen Speichers angefordert, was zu genau einer Speichertransaktion führt. **(b)** Es werden vier 8 Byte-Wörter aus vier Speichersegmenten des globalen Speichers angefordert, was zu vier Speichertransaktionen führt. 65
- 4.6 Drei Evaluationsmatrizen \mathbf{S}_1 , \mathbf{S}_2 und \mathbf{S}_3 mit jeweils vier, drei und fünf Elementen und einer Dimensionalität von $d = 2$ werden in eine einzelne Matrix \mathbf{S} überführt. Anschließend wird diese Matrix zeilenweise in einen Vektor überführt und damit *vektoriert*. Die Threads t_1 , t_2 und t_3 sind den jeweiligen Evaluationsmengen oder -matrizen \mathbf{S}_1 , \mathbf{S}_2 und \mathbf{S}_3 zugeordnet und greifen demnach nacheinander auf die Informationen dieses Vektors zu, weswegen die Zugriffe in möglichst wenigen Speichertransaktionen verschmolzen werden. 66

5.1 Vergleich der bei Wahl eines bestimmten Parameters k erreichte Metrik unter Nutzung des $k_{\text{Matérn}}^{\nu=1/2}$ -Kernels, verschiedener Datensätze, der SUMMARY OUTLIER DETECTION ohne RSP-Bäume und dem GREEDY-Optimierer. 87

5.2 Vergleich der erzielten Funktionswerte bei Nutzung der INFORMATIVE VECTOR MACHINE, des $k_{\text{Matérn}}^{\nu=1/2}$ -Kernels, einem festen Skalierungsparameter σ , der Optimierer GREEDY und SIEVE STREAMING und einer reinen Zufallsauswahl von Repräsentanten („Random“). Die grüne Fläche repräsentiert die Verbesserung des Funktionswert bei Nutzung des GREEDY-Algorithmus gegenüber dem SIEVE STREAMING-Verfahren, während die gelbe Fläche die Verbesserung des SIEVE STREAMING-Verfahrens gegenüber der Zufallsauswahl wiedergibt. Rote Flächen geben eine Verschlechterung des SIEVE STREAMINGS gegenüber der reinen Zufallswahl von Repräsentanten an. 89

5.3 Erzielte mittlere Laufzeiten der SUMMARY OUTLIER DETECTION mit RSP-Bäumen („SOD-RSP“) und ohne RSP-Bäume („SOD“) auf Grundlage der Optimierungsverfahren GREEDY, SIEVE STREAMING und einer reinen Zufallsauswahl von Repräsentanten. Die Laufzeiten sind in Abhängigkeit der Kardinalitätsbeschränkung k und in Sekunden angegeben und wurden auf Grundlage der Datensätze *Cardiotocography* und *PenDigits* erzielt. Für SOD-RSP wurde keine Optimierung anhand des SIEVE STREAMING-Verfahrens berücksichtigt. 91

5.4 Erzielte mittlere Laufzeiten des LOCAL OUTLIER FACTORS bei Variation der Nachbarschaftsgröße $minPts$ und des ISOLATION FORESTS bei Variation der Anzahl zu trainierender Bäume t . Als Grundlage wurden die Datensätze *Cardiotocography* und *PenDigits* betrachtet. 92

5.5 Erzielte Laufzeiten der submodularen Funktion des EXEMPLAR-BASED CLUSTERINGS auf Grundlage der einfädigen (ST) und mehrfädigen (MT) CPU-Implementierungen sowie der GPU-Implementierung unter Variierung der Anzahl der zu evaluierenden Mengen $|S_{\text{multi}}| \in \mathbb{N}$ 99

5.6 Erzielte Laufzeiten der submodularen Funktion des EXEMPLAR-BASED CLUSTERINGS auf Grundlage der einfädigen (ST) und mehrfädigen (MT) CPU-Implementierungen sowie der GPU-Implementierung unter Variierung der Anzahl der Elemente $|V| \in \mathbb{N}$ in der Grundmenge V 100

5.7 Erzielte Laufzeiten der submodularen Funktion des EXEMPLAR-BASED CLUSTERINGS auf Grundlage der einfädigen (ST) und mehrfädigen (MT) CPU-Implementierungen sowie der GPU-Implementierung unter Variierung der Anzahl der Elemente in jeder Evaluationsmenge $k \in \mathbb{N}$. Die Laufzeitskala ist logarithmisch skaliert. 102

Algorithmenverzeichnis

2.1	GREEDY-Algorithmus [35].	10
2.2	SIEVE STREAMING bei bekanntem Optimalwert $f(S_k^*)$ [5].	12
2.3	SIEVE STREAMING bei bekanntem Maximalwert m [5].	14
2.4	SIEVE STREAMING [5]	15
3.1	Training des ISOLATION TREES [40]	32
3.2	Training des ISOLATION FORESTS [40]	33
3.3	Bestimmung der Pfadlänge in einem ISOLATION TREE [40]	33
3.4	SUMMARY OUTLIER DETECTION [9].	41
3.5	Induktion des RSP-BAUMS.	45
3.6	SUMMARY OUTLIER DETECTION (bei Verwendung eines RSP-Baums).	47
4.1	EXEMPLAR-BASED CLUSTERING (CPU)	54
4.2	EXEMPLAR-BASED CLUSTERING (CPU, parallel)	56
4.3	EXEMPLAR-BASED CLUSTERING (CPU, mengen-parallel)	56
4.4	EXEMPLAR-BASED CLUSTERING (GPU)	64

Literaturverzeichnis

- [1] *Chapter I - Introduction*. In: FUJISHIGE, SATORU (Herausgeber): *Submodular Functions and Optimization*, Band 58 der Reihe *Annals of Discrete Mathematics*, Seiten 3 – 20. Elsevier, 2005.
- [2] ABADI, DANIEL J., SAMUEL R. MADDEN und NABIL HACHEM: *Column-stores vs. Row-stores: How Different Are They Really?* In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, Seiten 967–980, New York, NY, USA, 2008. ACM.
- [3] ALESKEROV, E., B. FREISLEBEN und B. RAO: *CARDWATCH: a neural network based database mining system for credit card fraud detection*. In: *Proceedings of the IEEE/IAFE 1997 Computational Intelligence for Financial Engineering (CIFEr)*, Seiten 220–226, March 1997.
- [4] AMDAHL, GENE M.: *Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities*. In: *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), Seite 483–485, New York, NY, USA, 1967. Association for Computing Machinery.
- [5] BADANIDIYURU, ASHWINKUMAR, BAHARAN MIRZASOLEIMAN, AMIN KARBASI und ANDREAS KRAUSE: *Streaming submodular maximization: Massive data summarization on the fly*. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, Seiten 671–680. ACM, 2014.
- [6] BELLALA, GOWTHAM, MANISH MARWAH, MARTIN ARLITT, GEOFF LYON und CULLEN BASH: *Following the Electrons: Methods for Power Management in Commercial Buildings*. In: *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, Seite 994–1002, New York, NY, USA, 2012. Association for Computing Machinery.
- [7] BEUTELSPACHER, ALBRECHT: *Lineare Algebra: Eine Einführung in die Wissenschaft der Vektoren, Abbildungen und Matrizen*, Kapitel Skalarprodukte, Seiten 295–335. Springer Fachmedien Wiesbaden, Wiesbaden, 2014.

- [8] BREUNIG, MARKUS M., HANS-PETER KRIEGEL, RAYMOND T. NG und JÖRG SANDER: *LOF: Identifying Density-Based Local Outliers*. In: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, SIGMOD '00*, Seite 93–104, New York, NY, USA, 2000. Association for Computing Machinery.
- [9] BUSCHJÄGER, SEBASTIAN, PHILIPP-JAN HONYSZ und KATHARINA MORIK: *Very Fast Streaming Submodular Maximization and its Application to Outlier Detection (submitted)*. In: *Proceedings of 26th SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2020)*.
- [10] CAMPOS, GUILHERME O., ARTHUR ZIMEK, JÖRG SANDER, RICARDO J. G. B. CAMPELLO, BARBORA MICENKOVÁ, ERICH SCHUBERT, IRA ASSENT und MICHAEL E. HOULE: *On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study*. *Data Mining and Knowledge Discovery*, 30(4):891–927, 2016.
- [11] CHANDOLA, VARUN, ARINDAM BANERJEE und VIPIN KUMAR: *Anomaly Detection: A Survey*. *ACM Comput. Surv.*, 41(3):15:1–15:58, Juli 2009.
- [12] CHEN, JIE, LEI WANG und MIHAI ANITESCU: *A Fast Summation Tree Code for Matérn Kernel*. *SIAM Journal on Scientific Computing*, 36(1):A289–A309, 2014.
- [13] CHEN, P., S. YANG und J. A. MCCANN: *Distributed Real-Time Anomaly Detection in Networked Industrial Sensing Systems*. *IEEE Transactions on Industrial Electronics*, 62(6):3832–3842, June 2015.
- [14] CHEN, YIZHEN, MING YING, DAREN LIU, ADIL ALIM, FENG CHEN und MEI-HWA CHEN: *Effective Online Software Anomaly Detection*. In: *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2017*, Seite 136–146, New York, NY, USA, 2017. Association for Computing Machinery.
- [15] CORPORATION, NVIDIA: *NVIDIA Tesla P100 Whitepaper*, 2016.
- [16] CORPORATION, NVIDIA: *CUDA C Programming Guide*, 2019.
- [17] DERCZYNSKI, LEON: *Complementarity, F-score, and NLP Evaluation*. In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, Seiten 261–266, Portorož, Slovenia, Mai 2016. European Language Resources Association (ELRA).
- [18] FAWCETT, TOM: *An introduction to ROC analysis*. *Pattern Recognition Letters*, 27(8):861 – 874, 2006. *ROC Analysis in Pattern Recognition*.
- [19] FLYNN, M. J.: *Some Computer Organizations and Their Effectiveness*. *IEEE Transactions on Computers*, C-21(9):948–960, 1972.

- [20] GOLDSTEIN, MARKUS und SEIICHI UCHIDA: *A Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Multivariate Data*. PLOS ONE, 11(4):1–31, 04 2016.
- [21] GOMES, RYAN und ANDREAS KRAUSE: *Budgeted Nonparametric Learning from Data Streams*. In: *Proceedings of the 27th International Conference on International Conference on Machine Learning, ICML'10*, Seiten 391–398, USA, 2010. Omnipress.
- [22] GRUBBS, FRANK E.: *Procedures for Detecting Outlying Observations in Samples*. Technometrics, 11(1):1–21, 1969.
- [23] GUPTA, NIKHIL, DHIVYA ESWARAN, NEIL SHAH, LEMAN AKOGLU und CHRISTOS FALOUTSOS: *Beyond Outlier Detection: LookOut for Pictorial Explanation*. In: BERLINGERIO, MICHELE, FRANCESCO BONCHI, THOMAS GÄRTNER, NEIL HURLEY und GEORGIANA IFRIM (Herausgeber): *Machine Learning and Knowledge Discovery in Databases*, Seiten 122–138, Cham, 2019. Springer International Publishing.
- [24] GYGLI, M., H. GRABNER und L. VAN GOOL: *Video summarization by learning sub-modular mixtures of objectives*. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Seiten 3090–3098, 2015.
- [25] H., CORMEN THOMAS, LEISERSON CHARLES E., RIVEST RONALD und STEIN CLIFFORD: *Algorithmen - Eine Einführung*. De Gruyter Oldenbourg CY 2017 PB.
- [26] HANSEN, L.K. und P. SALAMON: *Neural Network Ensembles*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 12:993–1001, 1990.
- [27] HASTIE, TREVOR, ROBERT TIBSHIRANI und ROBERT FRIEDMAN: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2008.
- [28] HAWKINS, D. M.: *Identification of Outliers*, Seiten 1–12. Springer Netherlands, Dordrecht, 1980.
- [29] HENNESSY, JOHN L. und DAVID A. PATTERSON: *Computer Architecture, Fifth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th Auflage, 2011.
- [30] HENZE, NORBERT: *Statistische Tests*, Seiten 267–293. Springer Fachmedien Wiesbaden, Wiesbaden, 2018.
- [31] HOOL, BRYAN, DHIVYA ESWARAN, HYUN AH SONG, AMRITANSHU PANDEY, MARKO JEREMINOV, LARRY PILEGGI und CHRISTOS FALOUTSOS: *GridWatch: Sensor Placement and Anomaly Detection in the Electrical Grid*. In: BERLINGERIO, MICHELE, FRANCESCO BONCHI, THOMAS GÄRTNER, NEIL HURLEY und GEORGIANA IFRIM

- (Herausgeber): *Machine Learning and Knowledge Discovery in Databases*, Seiten 71–86, Cham, 2019. Springer International Publishing.
- [32] KAUFMANN, L. und P.J. ROUSSEEUW: *Clustering by means of medoids*, Seiten 405–416. Elsevier Science Publishers, 1997.
- [33] KO, CHUN-WA, JON LEE und MAURICE QUEYRANNE: *An Exact Algorithm for Maximum Entropy Sampling*. *Operations Research*, 43(4):684–691, 1995.
- [34] KOUSHANFAR, F. und A. MIRHOSEINI: *A Unified Framework for Multimodal Submodular Integrated Circuits Trojan Detection*. *IEEE Transactions on Information Forensics and Security*, 6(1):162–174, 2011.
- [35] KRAUSE, ANDREAS und DANIEL GOLOVIN: *Submodular function maximization*. *Tractability: Practical Approaches to Hard Problems*, 3(19):8, 2012.
- [36] LAWRENCE, NEIL D. und JOHN C. PLATT: *Learning to Learn with the Informative Vector Machine*. In: *Proceedings of the Twenty-First International Conference on Machine Learning, ICML '04*, Seite 65, New York, NY, USA, 2004. Association for Computing Machinery.
- [37] LIEBER, DANIEL, BENEDIKT KONRAD, JOCHEN DEUSE, MARCO STOLPE und KATHARINA MORIK: *Sustainable Interlinked Manufacturing Processes through Real-Time Quality Prediction*. In: DORNFELD, DAVID A. und BARBARA S. LINKE (Herausgeber): *Leveraging Technology for a Sustainable World*, Seiten 393–398, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
- [38] LIN, H., J. BILMES und S. XIE: *Graph-based submodular selection for extractive summarization*. In: *2009 IEEE Workshop on Automatic Speech Recognition Understanding*, Seiten 381–386, 2009.
- [39] LIN, HUI und JEFF BILMES: *A Class of Submodular Functions for Document Summarization*. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1, HLT '11*, Seite 510–520, USA, 2011. Association for Computational Linguistics.
- [40] LIU, F. T., K. M. TING und Z. ZHOU: *Isolation Forest*. In: *2008 Eighth IEEE International Conference on Data Mining*, Seiten 413–422, Dec 2008.
- [41] MARKOU, MARKOS und SAMEER SINGH: *Novelty detection: a review—part 1: statistical approaches*. *Signal Processing*, 83(12):2481 – 2497, 2003.
- [42] MIRZASOLEIMAN, BAHARAN, STEFANIE JEGELKA und ANDREAS KRAUSE: *Streaming Non-Monotone Submodular Maximization: Personalized Video Summarization on the Fly*, 2018.

- [43] NEAPOLITAN, RICHARD und KUMARSS NAIMIPOUR: *Foundations of Algorithms Using C++ Pseudocode, Third Edition*. Jones and Bartlett Publishers, Inc., USA, 2008.
- [44] NEMHAUSER, G. L. und L. A. WOLSEY: *Best Algorithms for Approximating the Maximum of a Submodular Set Function*. Mathematics of Operations Research, 3(3):177–188, 1978.
- [45] NEMHAUSER, GEORGE L, LAURENCE A WOLSEY und MARSHALL L FISHER: *An analysis of approximations for maximizing submodular set functions—I*. Mathematical Programming, 14(1):265–294, 1978.
- [46] NOROUZI-FARD, ASHKAN, JAKUB TARNAWSKI, SLOBODAN MITROVIC, AMIR ZANDIEH, AIDA MOUSAVIFAR und OLA SVENSSON: *Beyond 1/2-Approximation for Submodular Maximization on Massive Data Streams*. CoRR, abs/1808.01842, 2018.
- [47] PEDREGOSA, F., G. VAROQUAUX, A. GRAMFORT, V. MICHEL, B. THIRION, O. GRISSEL, M. BLONDEL, P. PRETTENHOFER, R. WEISS, V. DUBOURG, J. VANDERPLAS, A. PASSOS, D. COURNAPEAU, M. BRUCHER, M. PERROT und E. DUCHESNAY: *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, 12:2825–2830, 2011.
- [48] RASMUSSEN, CE. und CKI. WILLIAMS: *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, USA, Januar 2006.
- [49] ROZENSHTAIN, POLINA, ARIS ANAGNOSTOPOULOS, ARISTIDES GIONIS und NIKOLAJ TATTI: *Event Detection in Activity Networks*. In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14*, Seite 1176–1185, New York, NY, USA, 2014. Association for Computing Machinery.
- [50] SCHÄFFLER, STEFAN: *Die Entropie*, Seiten 23–36. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.
- [51] SCHÖLKOPF, BERNHARD und ALEXANDER J. SMOLA: *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, Kapitel Kernels. MIT Press, Cambridge, MA, USA, 2001.
- [52] SCHÖLKOPF, BERNHARD und ALEXANDER J. SMOLA: *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, Kapitel Single-Class Problems: Quantile Estimation and Novelty Detection, Seiten 227–250. MIT Press, 2001.

- [53] SCHWARZ, HANS RUDOLF und NORBERT KÖCKLER: *Lineare Gleichungssysteme, direkte Methoden*, Seiten 30–90. Vieweg+Teubner Verlag, Wiesbaden, 2011.
- [54] SHANNON, C. E.: *A mathematical theory of communication*. The Bell System Technical Journal, 27(3):379–423, July 1948.
- [55] SHARMA, DRAVYANSH, AMIT DESHPANDE und ASHISH KAPOOR: *On Greedy Maximization of Entropy*. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, Seite 1330–1338. JMLR.org, 2015.
- [56] SIPOS, RUBEN, ADITH SWAMINATHAN, PANNAGA SHIVASWAMY und THORSTEN JOACHIMS: *Temporal Corpus Summarization Using Submodular Word Coverage*. In: *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, Seite 754–763, New York, NY, USA, 2012. Association for Computing Machinery.
- [57] SONG, FEI, BOYAO ZHOU, QUAN SUN, WANG SUN, SHIWEN XIA und YANLEI DIAO: *Anomaly Detection and Explanation Discovery on Event Streams*. In: *Proceedings of the International Workshop on Real-Time Business Intelligence and Analytics*, BIRTE '18, New York, NY, USA, 2018. Association for Computing Machinery.
- [58] STOJANOVIC, L., M. DINIC, N. STOJANOVIC und A. STOJADINOVIC: *Big-data-driven anomaly detection in industry (4.0): An approach and a case study*. In: *2016 IEEE International Conference on Big Data (Big Data)*, Seiten 1647–1652, Dec 2016.
- [59] STOLPE, MARCO, HENDRIK BLOM und KATHARINA MORIK: *Sustainable Industrial Processes by Embedded Real-Time Quality Prediction*, Seiten 201–243. Springer International Publishing, Cham, 2016.
- [60] THOMA, MARISA, HONG CHENG, ARTHUR GRETTON, JIAWEI HAN, HANS-PETER KRIEGEL, ALEXANDER J. SMOLA, LE SONG, PHILIP S. YU, XIFENG YAN und KARSTEN M. BORGWARDT: *Near-optimal supervised feature selection among frequent sub-graphs*. In: APTE, CHID, HAESUN PARK, KE WANG und MOHAMMAD J. ZAKI (Herausgeber): *Proceedings of the 9th SIAM International Conference on Data Mining*, Band 2, Seite 1087, Philadelphia, PA, 2009. SIAM. 9th SIAM International Conference on Data Mining (SDM 2009); Conference Location: Sparks, NV, USA; Conference Date: April 30 - May 2, 2009.
- [61] TSCHIATSCHKEK, SEBASTIAN, RISHABH K IYER, HAOCHEN WEI und JEFF A BILMES: *Learning Mixtures of Submodular Functions for Image Collection Summarization*. In: GHARAMANI, Z., M. WELLING, C. CORTES, N. D. LAWRENCE und K. Q.

- WEINBERGER (Herausgeber): *Advances in Neural Information Processing Systems 27*, Seiten 1413–1421. Curran Associates, Inc., 2014.
- [62] VERT, JP., K. TSUDA und B. SCHÖLKOPF: *A Primer on Kernel Methods*, Seiten 35–70. MIT Press, Cambridge, MA, USA, 2004.
- [63] WADSWORTH, GEORGE P. and JOSEPH G. BYRAN: *Introduction to probability and random variables*. 1960.
- [64] WILT, NICHOLAS: *The CUDA handbook : a comprehensive guide to GPU programming*. Addison-Wesley, Upper Saddle River, NJ, 2013.
- [65] WROBEL, STEFAN, THORSTEN JOACHIMS and KATHARINA MORIK: *Maschinelles Lernen und Data Mining*. In *Handbuch der Künstlichen Intelligenz*, pages 405–472. de Gruyter, 2013.
- [66] XU, ZHAO, LORENZO VON RITTER and KRISTIAN KERSTING: *Adaptive streaming anomaly analysis*. In *Proceedings of NIPS 2016 Workshop on Artificial Intelligence for Data Science*, 2016.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 10. Mai 2020

Philipp-Jan Honysz

**Eidesstattliche Versicherung
(Affidavit)**

Honysz, Philipp-Jan

165350

Name, Vorname
(Last name, first name)

Matrikelnr.
(Enrollment number)

Ich versichere hiermit an Eides statt, dass ich die vorliegende ~~Bachelorarbeit~~ Masterarbeit* mit dem folgenden Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present ~~Bachelor's~~ Master's* thesis with the following title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution.

Titel der ~~Bachelor-/~~Masterarbeit*:
(Title of the ~~Bachelor's/~~ Master's* thesis):


Unüberwachte Ausreißererkennung mit Hilfe von Submodularen Funktionen

*Nichtzutreffendes bitte streichen
(Please choose the appropriate)

Dortmund, 10.05.2020

Ort, Datum
(Place, date)

Unterschrift
(Signature)



Belehrung:

Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird ggf. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin“) zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:

Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to €50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, section 63, subsection 5 of the North Rhine-Westphalia Higher Education Act (*Hochschulgesetz*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

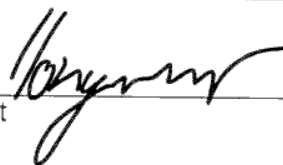
As may be necessary, TU Dortmund will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:**

Dortmund, 10.05.2020

Ort, Datum
(Place, date)

Unterschrift
(Signature)



**Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.