technische universität
dortmund

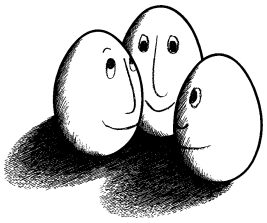
Diplomarbeit

Subspace-Clustering mit parallelen häufigen Mengen

Marcin Skirzynski
März 2012

Gutachter:
Prof. Dr. Katharina Morik
Dipl.-Inf. Nico Piatkowski

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl für Künstliche Intelligenz (LS8)
http://www-ai.cs.tu-dortmund.de



INHALTSVERZEICHNIS

NOTATIONEN

$\boldsymbol{X}$	Datensatz aus N Beispielen
$\vec{x}^{(i)} \in \boldsymbol{X}$	p-dimensionaler Vektor des i-ten Beispieles in $\boldsymbol{X}$
$x_j^{(i)}$	Wert des Beispiels $\vec{x}^{(i)}$ für das Attribut A_j
$\mathcal{C}$	Clustering eines Datensatzes
$\mathbf{C}_i$	i-tes Cluster eines Clusterings
$R = \{a_1, \ldots, a_p\}$	Eine Menge von Items
$T = \{t_1, \ldots, t_N\}$	Eine Menge von Transaktionen
$s(I)$	Support einer Menge von Items I
ξ_{rel}	Relativer Schwellwert für häufige Mengen
$S(T, \xi_{\text{rel}})$	Häufige Mengen für relativen Schwellwert ξ_{rel}
$\boldsymbol{A} = \{A_1, \ldots, A_p\}$	Menge von p Attributen bzw. Merkmalen
$\mathbb{A}^p = A_1 \times \ldots \times A_p \subseteq \mathbb{R}^p$	Reellwertiger Vektorraum für $\boldsymbol{A}$
$\mathbb{S}^k = A_{t_1} \times \ldots \times A_{t_k}$	k-dimensionaler Subspace bzw. Unterraum von $\mathbb{A}^p$
$\boldsymbol{S} \subseteq \boldsymbol{A}$	Attribute eines Subspaces bzw. Unterraumes
$\mathbf{C} = (\boldsymbol{X}, \boldsymbol{S})$	Subspace Cluster mit den Beispielen $\boldsymbol{X}$ im Subspace $\mathbb{S}^k$
I_A^i	i-tes Intervall für Attribut A
$\boldsymbol{P}$	Partitionierung von $\mathbb{A}^p$
$u^{(i)} \in \boldsymbol{P}$	Einheit einer Partitionierung
$u_j^{(i)} = [l_j, h_j)$	Intervall einer Einheit $u^{(i)}$ für ein Attribut A_j
$\vec{x}^{(i)} \sqsubset u^{(j)}$	$\vec{x}^{(i)}$ ist enthalten in Einheit $u^{(j)}$
$s(u^{(i)})$	Support einer Einheit $u^{(i)}$
τ	Schwellwert für dichte Einheiten
$u^{(i)} \boxminus u^{(j)}$	Zwei zusammenhängende Einheiten
$R = \bigwedge_{A_{t_i} \in \boldsymbol{S}} (L_{t_i} \leq A_{t_i} < H_{t_i})$	Region für die Attribute $\boldsymbol{S}$
$R \sqsubset \mathbf{C}$	Region ist in einem Cluster $\mathbf{C}$ enthalten
$\boldsymbol{R} = R^{(1)} \vee \ldots \vee R^{(d)}$	Minimale Beschreibung aus d Regionen
β	Schwellwert bei der Partitionierung
ϵ	Nachbarschaftsradius

1

EINLEITUNG

Clusteranalyse ist ein grundlegendes Verfahren im maschinellen Lernen. Hierbei werden die Daten zu sogenannten *Clustern* gruppiert. Die Objekte eines solchen Clusters sollen zueinander „ähnlich“ und zu Objekten anderer Cluster „unähnlich“ sein. Typischerweise besteht ein Objekt aus n reellwertigen Zahlen, die Merkmale des Objektes genannt werden. In diesem Fall kann beispielsweise der euklidische Abstand als Ähnlichkeitsmaß benutzt werden.

Für Daten mit hohen Dimensionen stoßen solche Verfahren oft an ihre Grenzen. Ein Grund hierfür ist der *Fluch der Dimensionalität*, der insbesondere die Verfahren betrifft, die auf Distanzfunktionen wie die euklidische Abstandfunktion basieren. Eine hohe Dimensionalität erhöht aber auch die Wahrscheinlichkeit, dass bestimmte Merkmale nur ein Rauschen zu den Daten hinzufügen und dadurch ein Clustering erschweren.

Um diese Probleme zu lösen, werden Verfahren des *Subspace Clustering* angewandt. Hierbei werden Teilmengen der Merkmale gesucht, in denen die Daten Cluster bilden. Das Ergebnis eines Subspace Clusterings ist also eine Menge von Objekten, die sich bezüglich einer Teilmenge ihrer Merkmale „ähnlich“ sind.

Die so ausgewählten Merkmale bieten für bestimmte Anwendungen zusätzliche Informationen. Ein Beispiel sei hier die Gen-Expressions-Analyse. Dabei werden Gewebeproben von an Krebs erkrankten Patienten entnommen und durch DNA-Microarrays wird aus diesen Proben ermittelt, wie stark bestimmte Gene exprimiert sind, was angibt, wie „aktiv“ diese Gene sind.

Viele Publikation aus dem Bereich des maschinellen Lernens beschäftigen sich mit diesen Microarray-Daten. Typischerweise sind die Gene dabei die Merkmale und eine Gewebeprobe ein Objekt bzw. Datenpunkt innerhalb des so aufgespannten Raumes. Das Beson-

dere an dieser Anwendung ist, dass die Anzahl der Merkmale ($n \approx 10.000 - 200.000$) dabei deutlich größer ist als die Anzahl der Proben ($m \approx 100$).
Die Anwendung von Subspace Clustering hilft der Biologie diese großen Datenmengen zu analysieren. Das Subspace Clustering würde auf folgende Frage eine Antwort liefern: „Welchen Patienten können hinsichtlich bestimmter Gene gut gruppiert werden?“ Eine weitere Betrachtung der Krankheitsverläufe der Patienten innerhalb einer ermittelten Gruppe und den dazu ausgewählten Genen, könnte Aufschlüsse zur Funktionsweise der Gene liefern.
Die vorliegende Diplomarbeit behandelt Subspace Clustering und die Anwendung auf Microarray-Daten, um solche Fragen zu beantworten. Das hierfür verwendete Verfahren basiert auf den von Agrawal et al. [4] vorgestellten CLIQUE-Algorithmus, der einen starken Bezug zum APRIORI-Verfahren hat, das beim Finden von häufigen Mengen – ein weiteres Problem im Bereich des maschinellen Lernens – verwendet wird. Im Folgenden wird dieser Ansatz analysiert und in einzelne Komponenten abstrahiert. Eine wichtige Komponente ist hier die Partitionierung des Datenraumes, für die mehrere Ansätze untersucht und bewertet werden, um anschließend ein eigenes Verfahren vorzustellen. Schließlich wird das Subspace Clustering vollständig in das Problem der häufigen Mengen überführt, um die Aufgabe mit einem schnellen und parallelen Verfahren zu lösen.

Struktur

In Kapitel 2 werden zunächst die Grundlagen, die zum Verständnis des Microarray-Datensatzes notwendig sind, erläutert. Daraufhin folgt eine Übersicht zum Bereich des maschinellen Lernens mit einem Fokus auf Clustering und häufige Mengen. Anschließend wird MAPREDUCE als Konzept zur parallelen Programmierung vorgestellt.
Kapitel 3 beschäftigt sich mit dem Algorithmus, stellt den Zusammenhang zwischen Subspace Clustering und häufigen Mengen her und wie Subspace Cluster durch das Finden von häufigen Mengen gefunden werden können. Dabei wird das Verfahren in einzelne Module unterteilt und untersucht. Ein Schwerpunkt liegt auf der Partitionierung, für die ein eigenes Verfahren vorgestellt wird. Weiterhin wird vorgestellt, wie das Verfahren durch die Parallelisierung beschleunigt werden kann.
Die neue Partitionierungstechnik, als auch das parallele Suchen der häufigen Mengen, wird in Kapitel 4 experimentell untersucht. Anschließend wird das Verfahren verwendet, um die Microarray-Daten zu untersuchen. Abschließend wird in Kapitel 5 ein Fazit aus den Untersuchungen gezogen.

2

GRUNDLAGEN

2.1 Biologische Grundlagen

Ein wichtiger Faktor beim maschinellen Lernen ist solides Grundlagenwissen im Anwendungsbereich. Nur so können die richtigen Algorithmen gewählt und die Daten geeignet vorverarbeitet werden. Auch bei der Evaluation der Ergebnisse ist dieses Wissen von Bedeutung.
Das folgende Kapitel handelt von diesen notwendigen biologischen Grundlagen (zusammengetragen aus [21]). Zunächst wird beschrieben, wie Erbinformationen gespeichert werden und wie aus diesen Informationen die Bausteine des Lebens entstehen. Dies wird Genexpression genannt und kann durch die beschriebene Microarray-Technologie gemessen werden. Mit dieser Methode werden Daten von an Krebs leidenden Patienten erhoben, die in dieser Arbeit untersucht werden.

2.1.1 Genetik

Zellen sind die kleinsten Bestandteile des menschlichen Körpers (und aller anderen lebenden Organismen). Jegliche Aktivitäten und Prozesse sind im Grunde ein komplexes Zusammenspiel aus vielen verschiedenen Zellen. Sie sind die kleinsten und einfachsten lebenden Einheiten in der Biologie.
Die Entstehung von Leben beginnt dabei mit einer einzelne Zelle. Durch eine wichtige Fähigkeit der Zelle, und zwar der Zellteilung, entstehen mit der Zeit bis zu 100 Billionen von weiteren Zellen. Dabei gibt es viele verschiedene Zelltypen. Beim Menschen sind dies um die 200 Verschiedene, wie beispielsweise eine Muskelzelle oder eine Nervenzelle.

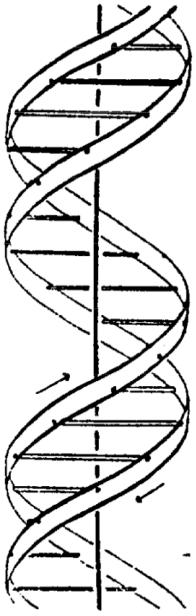


Abbildung 2.1: Schematische Skizze der Doppelhelixstruktur der DNA aus [74]. Zwei Ketten aus Phosphat und Zucker werden durch ein Nukleinbasen-Paar miteinander verbunden.

Die Information, wann eine Zelle eine Nervenzelle wird und wie diese aufgebaut ist, wird von den Eltern durch die **Gene** weitergegeben. Innerhalb einer Zelle befinden sich die Chromosomen, die aus langen **Desoxyribonukleinsäure**-Molekülen (kurz DNS bzw. **DNA** auf englisch) bestehen. Ein Gen ist ein Abschnitt auf diesem Molekül.

In [74] beschreiben die Autoren die molekulare Struktur der DNA als schraubenförmige Doppelhelix, die in der Mitte durch je zwei Nukleinbasen verbunden werden. In Abbildung 2.1 ist die Originalskizze aus [74] abgebildet. Es wird zwischen vier verschiedenen Basen unterschieden: Adenin (A), Guanin (G), Cytosin (C) und Thymin (T). Aufgrund der möglichen Wasserstoffbrücken zwischen den Basen bilden jedoch nur Adenin und Thymin oder Guanin und Cytosin ein Basenpaar, das die Helix zusammenhält. Die jeweiligen Basen werden als komplementär zueinander beschrieben.

Von der DNA zum Protein Wesentliche Bestandteile einer Zelle und verantwortlich für nahezu jeden Prozess innerhalb dieser sind die **Proteine**, die im wesentlichen Moleküle aus Aminosäuren sind. Die Synthese von bestimmten Proteinen ist also verantwortlich für den Typ und die Aufgabe einer Zelle. Die Gene speichern die Information für diese Synthese durch spezielle Sequenzen der Nukleinbasen. Um aus dieser Information ein Protein zu erstellen sind zwei Schritte notwendig. Der gesamte Prozess wird auch **Genexpression** genannt.

Transkription Bei der Transkription werden zunächst die Wasserstoffbrücken der Nukleinbasen im DNA-Molekül aufgespalten, so dass zwei einzelne Stränge, ein kodierender und ein komplementärer Strang, entstehen. An den kodierenden Strang lagern sich komplementäre Basen und Enzyme an, so dass ein neuer Strang entsteht, der Boten-Ribonukleinsäure (kurz Boten-RNS bzw. **mRNA** auf englisch) genannt wird. Dieser wird vom kodierenden Strang abgespalten, so dass der kodierende und der komplementäre Strang wieder ein DNA-Molekül bilden können. Der so entstandene mRNA-Strang ist der DNA sehr ähnlich, hat jedoch statt der Thymin- nun Uracil-Basen (U).

Translation Die mRNA-Moleküle gelangen nun in sogenannte Ribosomen innerhalb der Zelle. Die Aufgabe dieser Ribosomen ist es, Aminosäuren zu erstellen indem komplementäre Basen an den Basen der mRNA-Moleküle angelagert werden. Die so erstellten Aminosäuren bilden nach ihrer Erstellung ein bestimmtes, von den Basen des mRNA-Moleküls codiertes Protein.

Genetischer Code Bei der Übersetzung der mRNA innerhalb eines Ribosoms werden immer Einheiten von drei Basen zu einer Aminosäure übersetzt. Diese Basen werden **Codeons** genannt und die Reihenfolge dieser Basen bestimmt welche Aminosäure erstellt wird.
Es gibt also vier verschiedene Basen, die in einem Codeon an drei verschiedenen Stellen gesetzt sein können. Dadurch können theoretisch $4^3 = 64$ verschiedene Aminosäuren erstellt werden, obwohl es nur 20 verschiedene Aminosäuren gibt. Dies ist notwendig, da mit zwei Basen nur $4^2 = 16$ verschiedene Säuren kodiert werden können. So kodieren mehrere verschiedene Sequenzen die gleiche Aminosäure. Daneben gibt es noch Codeons, die den Start und den Stopp der Proteinsynthese symbolisieren. Mit Hilfe der Transfer-Ribonukleinsäure (tRNA) ordnen sich die so erzeugten Aminosäuren zu Proteinen an.

Introns und Exons Der Sequenzabschnitt der DNA lässt sich funktionell in zwei Bereiche unterteilen. Zum einen gibt es die **Exon**s (**Ex**pressed Regi**on**s) , die den Abschnitt bezeichnen, der die Aminosäuren eines Proteins kodiert. Zwischen diesen Bereichen sind Abschnitte, die **Intron**s (**Int**ervening Regi**on**s) genannt werden. Ein einzelnes Gen auf dem DNA-Strang besteht aus einer Sequenz von mehreren Exons und Introns.
Die Introns werden vor der Übersetzung zur mRNA entfernt (dieser Vorgang wird Spleißen genannt) und dient somit zunächst als Trenner zwischen den Exons. Es scheint, dass sie nichts kodieren. Es wird jedoch vermutet, dass Teile der Introns trotzdem auf bestimmte Art und Weise transkribiert werden und eine regulierende Funktion ausüben. Sie stellen auch den Großteil der DNA dar. Die kodierenden Exons sind lediglich ein kleiner Teil der gesamten DNA. Bei den Menschen sind beispielsweise nur 1.5% der DNA

kodierend. Zusätzlich wird bei der Synthese nur ein Teil der kodierenden Gene verwendet. Dies hängt von der Art der Zelle ab, liegt aber meistens bei 20% der Exons. Die Menge, die zur Genexpression verwendet wird, ist jedoch für jede Zelle einzigartig.

2.1.2 Neuroblastom

Zellteilung ist nicht nur zum Wachstum eines Organismus, sondern auch zur Regeneration wichtig, da Zellen kontinuierlich absterben. Zum Überleben muss also ein Gleichgewicht zwischen neuen, durch Zellteilung entstandenen Zellen und alten, absterbenden Zellen herrschen. Dieser selbstregulierende Prozess wird durch den genetischen Code bei der Zellteilung gesteuert. Durch Defekte an diesem Code kann dieser Prozess gestört werden, so dass ein Ungleichgewicht zu Gunsten der Zellentstehung entsteht. Dieser Defekt kann prinzipiell an jeder Zell- und Gewebeart entstehen, und diese unkontrollierte Gewebeneubildung wird als bösartiger Tumor oder auch Krebs bezeichnet.

Das Neuroblastom ist ein solcher Tumor und eine Erkrankung, die insbesondere im Säuglings- und Kindesalter auftritt. Nach [19] ist dies mit einem Anteil von 5-7% eine der häufigsten Krebserkrankungen in der Kindheit. Die Heilungsaussichten sind recht unterschiedlich. Während es Krankheitsverläufe gibt, die sich ohne weitere Behandlung verbessern, hilft in anderen Fällen selbst der Einsatz von Chemotherapie und Operationen nicht. Ein ausschlaggebender Faktor hierfür ist das Alter. Kleinkinder haben deutlich bessere Erfolgsaussichten zur spontanen Selbstheilung als ältere Patienten.

Genetische Disposition Es wird vermutet, dass Umweltfaktoren kaum Auswirkung auf die Erkrankung und den Krankheitsverlauf haben (siehe [19]). Gene, die das unkontrollierte Wachstum der Zellen fördern, werden Onkogene genannt. Beim Neuroblastom konnte bisher das Onkogen MYCN ermittelt werden (siehe [18, 19, 70]). Eine Amplifikation dieses Gens liegt in 22% der Fälle vor. Unter der Amplifikation eines Gens versteht man die Vermehrung des Genabschnittes und damit eine erhöhte Aktivität bei der Genexpression.

Die Überlebenschancen von Patienten mit ($\sim$ 40-50%) und ohne ($\sim$ 90%) MYCN-Amplifikation variieren dabei recht stark. Deutlich wird das vor allem bei Kleinkindern unter 12 Monaten. Liegt bei ihnen eine Amplifikation vor, dann überleben 90% die weiteren drei Jahre, ansonsten nur 10% (siehe [19]). Bei der Behandlung kann dieses Onkogen zur Risikoeinstufung des Patienten verwendet werden, um Kleinkindern mit geringerem Risiko einer schonenden Behandlung zu unterziehen. Chemotherapien und Operationen sind für Kleinkinder sehr belastend und können sich damit auch negativ auf den Krankheitsverlauf auswirken.

2.1.3 DNA-Microarray

Es gibt einige Methoden die Expression von Genen zu ermitteln. Etabliert hat sich dabei die sogenannte DNA-Microarray-Technologie. Mit dieser Methode können in einem Experiment simultan mehrere tausend Gene bzw. Exons gleichzeitig und kostengünstig untersucht werden.

Experimentelles Vorgehen Der erste Schritt eines Microarray-Experiments ist es von der zu untersuchenden Gewebeprobe die mRNA zu isolieren. Diese instabilen Moleküle werden dann durch einen der Transkription ähnlichen Prozess in cDNA umgewandelt. Zusätzlich werden die Nukleinbasen mit fluoreszierenden Farbstoffen markiert.
Im zweiten Schritt werden einzelne DNA-Stränge (ebenfalls aus dem Gewebe) auf dem Microarray verteilt. Jedes Gen oder Exon erhält dabei einen Platz in diesem Feld. Die cDNA Mixtur wird dann auf dieses Array angewendet.
Im letzten Schritt wird dieses Microarray eingescannt. Für jedes Feld kann dadurch die Fluoreszenz gemessen werden, wodurch erkannt wird wieviele cDNA-Moleküle sich mit dem DNA-Strang verbunden haben. Dies weist auf die Expression des jeweiligen Genes in der Gewebeprobe hin.

2.1.4 Verwendete Daten

Die in dieser Arbeit verwendeten Daten wurden mit dem Microarray-Chip *Affymetrix U95Av2* vom onkologischen Labor der Universitätsklinik Essen erstellt, das sich schon seit Jahren mit Microarraydaten zu Neuroblastoma beschäftigt (siehe u.a. [68]). Der Datensatz beinhaltet 113 verschiedene Gewebeproben für die die Expression von 17 882 Genen bzw. 287 329 Exons ermittelt wurde.

Aufbereitung Nach dem Scan eines Microarrays müssen die entstandenen Bilddaten analysiert und aufbereitet werden. Folgende drei Schritte sind dabei notwendig (siehe auch [35]) und wurden schon im onkologischen Labor durchgeführt.

Entfernung von Hintergrundrauschen Nachdem die Farbintensität ermittelt wurde, ist die erste Aufgabe Rauschen zu ermitteln und zu eliminieren. Bei der Aufnahme entsteht ein Hintergrundrauschen, das technisch nicht zu vermeiden ist. Ein leeres Feld im Array kann jedoch dazu verwendet werden die Intensität des Rauschens zu ermitteln und diesen Wert mit den anderen Feldern zu verrechnen.

Normalisierung Weiterhin sind kleine Varianzen bei der experimentellen Ausführung kaum zu vermeiden. Selbst kleinste Unterschiede in der Menge der isolierten mRNA oder des Farbstoffes, führen zu unterschiedlichen Skalen der Intensität. Um dies für alle Proben auszugleichen, müssen die Daten normalisiert werden.

Aggregation Aus technischen Gründen sind längere Exons auf mehrere Felder aufgeteilt. Im letzten Aufbereitungsschritt müssen diese aggregiert werden, um einen Intensitätswert für das ganze Exon zu erhalten.

Bei den vorliegenden Daten wurden Verfahren gegen das Hintergrundrauschen und zur Normalisierung und Aggregation aus [40] angewandt. Daraus entstand eine $113 \times 17\,882$ große Datenmatrix $\boldsymbol{X}$, für die noch zusätzliche Attribute zum Krankheitsverlauf des Patienten vorhanden sind. Dies beinhaltet zum einen das Alter der Diagnose, einen Rückfall nach einer Behandlung (**Rezidiv**) und letztendlich den Tod des Patienten. Dabei erleiden 38 Patienten einen Rückfall, während 26 an den Folgen der Erkrankung verstorben sind.

2.2 Maschinelles Lernen

In Kapitel 2.1 wurden bisher rein biologische Grundlagen erläutert. Die Aufgabe eines Informatikers ist es nun, die in Kapitel 2.1.4 beschriebenen Daten zu verarbeiten, so dass Biologen daraus Erkenntnisse gewinnen können. Der Erwerb von Wissen durch Erfahrungen wird gemeinhin als Lernen bezeichnet und letztendlich sind Daten nichts anderes als Erfahrung. Dieser Bereich wird in der Informatik maschinelles Lernen genannt und ist ein Teilgebiet der künstlichen Intelligenz. Dieses Kapitel liefert einen groben Überblick über dieses Feld und ist angelehnt an die jeweilige Fachliteratur [13, 38, 72], in der weitere Informationen zu finden sind.

Wenn ein Algorithmus „lernt“, dann bedeutet das nicht, dass er die vorliegenden Daten einfach auswendig lernt, sondern diese verallgemeinert und Gesetzmäßigkeiten findet. Ein Beispiel hierfür ist die Handschrifterkennung bei der Post. Um Briefe anhand der Postleitzahl automatisch vorzusortieren, muss ein Algorithmus in der Lage sein Zahlen zu erkennen. Um das zu lernen, werden zunächst einige Beispiele benötigt, also Bilder von handschriftlichen Zahlen wie in Abbildung 2.2 für die Zahl Sechs beispielhaft zu sehen. Hier wird auch deutlich, wieso reines auswendig lernen wenig hilfreich ist. Eine handgeschriebene Sechs sieht von Fall zu Fall unterschiedlich aus. Menschen haben gelernt dies zu verallgemeinern und erkennen eine Sechs auch, wenn die Form stark abweicht. Dies geschieht also auch dann, wenn sie exakt die gleiche Sechs noch nie vorher gesehen haben. Ein

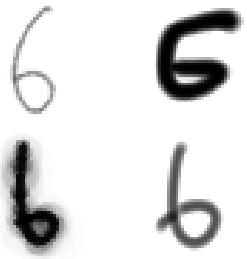


Abbildung 2.2: Handgeschriebene Sechsen mit unterschiedlichen Formen werden durch Verallgemeinerung durch Menschen als die Zahl Sechs erkannt.

Algorithmus zur Handschrifterkennung muss ebenfalls diese Leistung erbringen, um effektiv eingesetzt werden zu können.

2.2.1 Definition und Notation

Für das maschinelle Lernen gibt es viele Definitionen, die unterschiedliche Gesichtspunkte hervorheben. Eine mögliche Definition verwendet Thomas Mitchell in [57]:

Definition 2.2.1 (Maschinelles Lernen). Ein Computerprogramm lernt von Erfahrung **E** bezüglich einer Klasse von Aufgaben **T** und eines Gütemaßes **P**, wenn die Güte einer Aufgabe in **T**, gemessen durch Gütemaß **P**, sich mit Erfahrung **E** verbessert.

Bezogen auf das Beispiel der Handschrifterkennung ist die Aufgabe **T** das Erkennen einzelner handschriftlich notierter Zahlen. Als Gütemaß **P** kann die Erkennungsrate verwendet werden und die Erfahrung **E** ist eine Menge von handschriftlichen Beispielen, an denen der Algorithmus typische Charakteristiken des Zahlenbildes lernen kann.
Ein zentraler Aspekt zum Lernen ist dabei die Erfahrung **E**, die einem Lernalgorithmus als Datensatz vorliegt. Die direkt zum Lernen verwendeten Daten werden auch **Trainingsdaten** $\boldsymbol{X}_{\text{train}}$ genannt. Hieraus erstellt das maschinelle Lernverfahren ein **Modell** $\hat{f}$, manchmal auch **Hypothese** genannt. Dieses Modell stellt die aus den Daten gelernten Informationen dar und kann nun auf weiteren Daten, den **Testdaten** $\boldsymbol{X}_{\text{test}}$, angewendet werden. Das Ergebnis der Modellanwendung ist abhängig vom Algorithmus, wird aber in der Regel **Label** $\hat{y}$ genannt. Im Prinzip ist das Modell eine gelernte Funktion, die als Eingabe ein Datenbeispiel $x^{(i)} \in \boldsymbol{X}_{\text{test}}$ bekommt und eine gewünschte Ausgabe $\hat{y}$ liefert (siehe Abbildung 2.3). Am Beispiel der Handschrifterkennung wäre das die erkannte Zahl.
Die Trainingsdaten aus denen das Modell erstellt wird, bestehen aus einer Menge von N **Beispielen**

$$\boldsymbol{X} = \{\vec{x}^{(1)}, \ldots, \vec{x}^{(N)}\}$$

Jedes Beispiel $\vec{x}^{(i)}$ ist dabei ein p-dimensionaler Vektor bei dem $x_j^{(i)}$ den Wert darstellt, den $\vec{x}^{(i)}$ für das **Merkmal** bzw. **Attribut** A_j annimmt. Die Reihenfolge, die Anzahl und der Wertebereich eines jeden Attributes für jedes Beispiel ist über den gesamten Trainingsdatensatz gleich.
Bei dem Wertebereich eines Attributes wird zwischen **numerischen**, **nominalen** und **binominalen** Wertebereichen unterschieden. Numerische Attribute sind meist reellwertige Zahlen, wobei nominale Attribute eine ungeordnete Menge an Kategorien darstellen. Ist ein Attribut binominal gibt es nur zwei mögliche Werte, die ein Beispiel für dieses Attribut annehmen kann, z.B. `ja` und `nein`. Der Wertebereich des Labels kann ebenfalls numerisch, nominal oder binominal sein. Die folgende Arbeit beschränkt sich auf den Fall $\boldsymbol{X} \subseteq \mathbb{R}^p$. Für nicht-numerische Attribute müssen die Algorithmen angepasst werden.

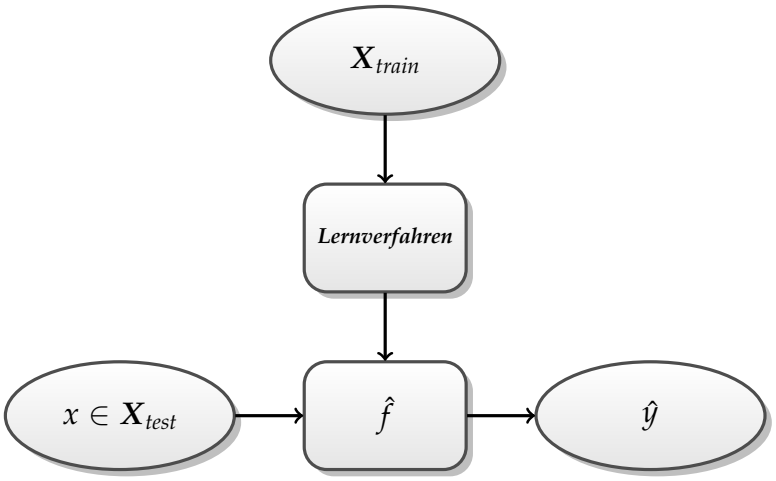


Abbildung 2.3: Ein Lernverfahren erhält Trainingsdaten $\boldsymbol{X}_{\text{train}}$ aus denen ein Modell $\hat{f}$ gebildet wird. Dieses Modell kann auf Testdaten $\boldsymbol{X}_{\text{test}}$ angewendet werden und erstellt ein Label $\hat{y}$ für jedes Beispiel $x \in \boldsymbol{X}_{\text{test}}$.

Oftmals gibt es auch eine einfache, bijektive Abbildung, so dass dann auch numerische Algorithmen verwendet werden können.

Bei der Handschrifterkennung könnten beispielsweise eine Menge von handschriftlichen Zahlen eingescannt und innerhalb eines gleich großen, monochromen Bildes skaliert werden. Dann kann jedes Pixel als ein numerisches Attribut aufgefasst werden, dessen Wert die Helligkeit des Pixels darstellt.

2.2.2 Überwachtes Lernen

Damit das Lernverfahren zur Handschrifterkennung ein Modell lernen kann, das eine Zahl richtig erkennt, muss es wissen, welches Trainingsbeispiel welche Zahl darstellt. Dem Trainingsbeispiel muss also sein „wahres“ Label y zugeordnet werden. Dies stellt eine Zielvorgabe dar, die gelernt und als $\hat{y}$ vorhergesagt werden soll. Diese Zuordnung muss jedoch vorher von jemandem festgelegt werden. Ein sogenannter Experte muss jedes Trainingsbeispiel „labeln“, d.h. jedem Bild eine Zahl zuordnen, die durch dieses Bild dargestellt wird. Dies ist vergleichbar mit dem Lernen eines Kindes, das von seinen Eltern gesagt bekommt, dass der Vierbeiner vor ihm eine Katze ist und kein Hund. Diese Art des Lernens wird auch **überwachtes Lernen** genannt.

Hierbei muss also jedes Beispiel $\vec{x}^{(i)}$ des Trainingsdatensatzes um ein Label-Attribut $y^{(i)}$ erweitert werden:

$$\boldsymbol{X}_{\text{train}} = \{(\vec{x}^{(1)}, y^{(1)}), \ldots, (\vec{x}^{(N)}, y^{(N)})\}$$

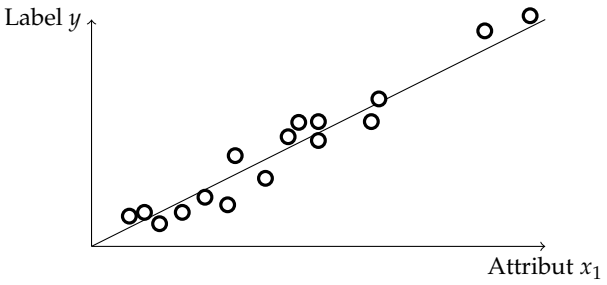


Abbildung 2.4: Beispiel für eine lineare Regression

Dabei ist $y^{(i)}$ ein spezielles Attribut, da es im Modell nach der Lernphase nicht mehr direkt verwendet werden kann. Die Testdaten haben dieses Attribut nämlich nicht. Schließlich soll dies vorhergesagt werden.

Funktionsapproximation Mathematisch betrachtet wird die Annahme gemacht, dass es für das zu lösende Problem eine „wahre“ Funktion $f(\vec{x}) = y$ gibt, für die eine Menge von Beispielen mit ihrem dazugehörigen Funktionswert zur Verfügung stehen. Diese Funktion f ist nicht bekannt und soll nun möglichst gut durch eine geschätzte Funktion $\hat{f}(\vec{x}) = \hat{y}$ approximiert werden und zwar so, dass die vorhandenen Trainingsbeispiele, aber auch neue Daten, möglichst gut vorhergesagt werden sollen. Oder anders ausgedrückt: Es soll für eine Funktion f auf Basis von N Funktionsauswertungen eine approximierte Funktion $\hat{f}$ gefunden werden.

Regression Beim überwachten Lernen wird unterschieden bzgl. des Wertebereiches des Labels. Dieses kann wie jedes andere Attribut entweder numerisch, nominal oder binominal sein. Das Lernen mit einem numerischen Label, also beispielsweise der Vorhersage einer reellwertigen Zahl, wird **Regression** genannt. Ein Beispiel hierfür wäre die Energievorhersage eines regionalen Stromanbieters. Dieser muss schon einen Monat im Voraus bei einem Kraftwerksbetreiber Strom einkaufen. Es muss also möglichst gut vorhergesagt werden, wie viel Strom zu welchen Tageszeiten bzw. Wochentagen gebraucht wird. Eine falsche Vorhersage führt zu höheren Strompreisen, da entweder zu viel Strom gekauft wurde oder fehlender Strom teurer nachgekauft werden muss.

Das maschinelle Lernen anhand der Regression sollte schon aus dem Mathematikunterricht in der Schule bekannt sein. Für eine Menge von Punkten ist eine Regressionsgerade zu finden, die die Punkte möglichst gut wieder spiegelt (siehe auch Abbildung 2.4). Dies

nennt man **lineare Regression**. Hier ist das ermittelte Modell für $\vec{x} \in \mathbb{R}^p$ und einen Gewichtsvektor $\vec{\theta} \in \mathbb{R}^{p+1}$ die Funktion

$$\begin{aligned}\hat{f}(\vec{x};\vec{\theta}) &= \left(\sum_{j=1}^{p} \theta_j x_j\right) + \theta_{p+1} \\ &= \vec{\theta}^{\mathsf{T}}\vec{x} && \text{mit} \quad x_{p+1} = 1 \\ &= \left\langle \vec{\theta}, \vec{x} \right\rangle && \text{mit } \langle\cdot,\cdot\rangle \text{ als Skalarprodukt}\end{aligned}$$

Die Aufgabe **T** ist es, den optimalen Gewichtsvektor $\vec{\theta}^*$ zu ermitteln. Welcher das ist, wird durch ein Gütemaß **P** entschieden. Bei der Regressionsanalyse wird als Gütemaß meistens der quadratische Fehler verwendet:

$$RSS(\hat{f};\vec{\theta}) = \sum_{i=1}^{N} \left(y^{(i)} - \hat{f}(\vec{x}^{(i)};\vec{\theta})\right)^2$$

Dieses Gütemaß ist über die bisherige Erfahrung **E**, also den Datensatz $\boldsymbol{X}_{\text{train}}$, definiert. Diese Funktion soll nun bzgl. $\vec{\theta}$ minimiert werden:

$$\vec{\theta}^* = \min_{\vec{\theta}} RSS(\hat{f};\vec{\theta})$$

Das Lösen von Optimierproblemen ist oft zu beobachten bei Lernalgorithmen, denn Optimierung ist ein wichtiges Hilfsmittel beim maschinellen Lernen.

Modellkomplexität Bei der linearen Regression ist das Modell beschrieben durch eine Klasse von linearen Funktionen, für die der Parametervektor $\vec{\theta}^*$ gelernt werden soll. Für komplexere Daten funktioniert dies nicht, da dann das lineare Modell die zu Grunde liegenden Daten nur unzureichend abbilden kann. In diesem Fall kann die Komplexität des Modells erhöht werden, z.B. durch eine Polynomapproximation:

$$\hat{f}(\vec{x}; m, \vec{\theta}^{(1)}, \ldots, \vec{\theta}^{(m)}) = \sum_{i=1}^{m} \left\langle \vec{\theta}^{(i)}, \vec{x}^{i} \right\rangle$$

Der Parameter m ist dabei der Grad des Polynoms und für $m = 1$ erhalten wir auch ein lineares Regressionsmodell.
Nach dem Approximationssatz von Weierstrass [75] kann jede stetige Funktion gleichmäßig auf einem kompakten Intervall durch Polynome approximiert werden. Dadurch erreichen wir eine große Flexibilität des Modells, die aber mit Vorsicht zu genießen ist. Die „wahre Funktion“ der Daten aus der realen Welt sind oft nicht perfekt. Es muss angenommen werden, dass es ein statistisches Rauschen gibt, was aber nicht ins Modell aufgenommen werden soll. Es wäre zwar möglich eine Funktion zu erstellen, die genau den Daten in Abbildung 2.4 entspricht, dies wäre aber eine sehr chaotische Funktion, die auf $\boldsymbol{X}_{\text{train}}$ zwar keinen Fehler macht, sich auf Testdaten aber sehr schlecht verhält. In Abbildung 2.4 ist eine lineare Funktion plausibler, auch wenn ein Fehler vorhanden ist,

der hier aber als Datenrauschen aufgefasst werden sollte. Das Lernen dieses Rauschens ist vergleichbar mit dem anfangs erwähntem Auswendiglernen. So ein auswendig gelerntes Modell wird auch **überangepasstes** Modell genannt. Somit ist das Finden von m für ein gutes generalisiertes polynomielles Regressionsmodell ein wichtiges Problem. Und dies ist nicht nur bei Regressionsproblemen der Fall. Auch bei der Modellkomplexität anderer Verfahren muss berücksichtigt werden, dass ein gutes Modell so speziell wie nötig, aber so generell wie möglich ist.

Klassifikation Neben der Regression gibt es noch das Lernen von nominalen bzw. binominalen Labeln. Dies wird auch **Klassifikation** genannt. Auch wenn das Beispiel der Handschrifterkennung eine Zahl vorhersagen soll, handelt es sich hierbei um eine Klassifikationsaufgabe. Letztendlich soll nur zwischen 10 verschiedenen Klassen unterschieden werden, also ob es sich bei der Zahl um eine Null, Eins, ..., Neun handelt. Das eigentliche Konzept der Zahl oder das Wissen, dass eine Sechs größer ist als eine Fünf, ist für diese Aufgabe unerheblich.
Eine weiteres Beispiel für eine Klassifikationsaufgabe ist das Erkennen von Spam in einem E-Mail Programm. Nachdem der Benutzer eine Zeit lang den Algorithmus trainiert hat, indem einzelne E-Mails als Spam markiert wurden, kann das E-Mail Programm nun automatisiert zwischen den Klassen „`Spam`“ und „`Nicht Spam`“ unterscheiden und dann unerwünschte E-Mails aussortieren.
Ein simples Klassifikationsmodell für ein Zwei-Klassenproblem besteht aus einer Hyperebene H, die die beiden Klassen voneinander trennt. In der Hesse-Normalform lässt sich diese mit

$$H = \left\{ \vec{x} \in \mathbb{R}^p \mid \left\langle \vec{\beta}, \vec{x} - \vec{a} \right\rangle = 0 \right\} \quad \text{und} \quad \|\vec{\beta}\| = 1$$

beschreiben. Dabei ist $\vec{a}$ ein Stützvektor auf der Hyperebene und $\vec{\beta}$ der Normalenvektor, der orthogonal zur Hyperebene steht. Anders ausgedrückt besteht die Hyperebene aus den Punkten $\vec{x}$, deren Richtungsvektoren $\vec{x} - \vec{a}$ orthogonal zum Normalenvektor $\vec{\beta}$ stehen. Da für das Skalarprodukt $\langle \vec{x}, \vec{y} \rangle = 0$ gilt, falls die Vektoren $\vec{x}$ und $\vec{y}$ orthogonal zueinander stehen, beschreibt H durch die Gleichung $\left\langle \vec{\beta}, \vec{x} - \vec{a} \right\rangle = 0$ eine Hyperebene.
Das Skalarprodukt ist ebenfalls nützlich zur Klassifikation durch eine gegebene Hyperebene. An der Kosinusdefinition des Skalarprodukts

$$\langle \vec{x}, \vec{y} \rangle = \|\vec{x}\| \cdot \|\vec{y}\| \cdot \cos(\measuredangle(\vec{x}, \vec{y}))$$

ist erkennbar, dass das Vorzeichen des Skalarprodukts vom Kosinus des Winkels zwischen den beiden Vektoren abhängt. Wenn der Winkel spitz ist, dann ist der Kosinus positiv, bei einem stumpfen Winkel negativ und bei 90° genau 0. Anders ausgedrückt: Sind zwei Vektoren gleich orientiert, dann ist ihr Skalarprodukt positiv, sind sie dagegen

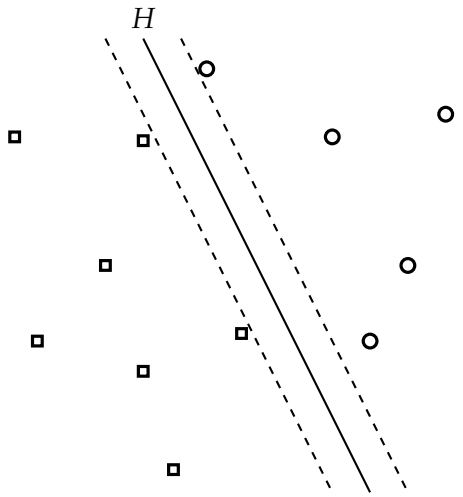


Abbildung 2.5: Zwei Klassen werden durch eine Hyberebene H voneinander getrennt. Die Hyperebene wird von der *Support Vector Machine* so gewählt, dass der kleinste Abstand zwischen einem $x \in \boldsymbol{X}_{\text{train}}$ und H maximal ist.

entgegengesetzt orientiert, dann ist das Skalarprodukt negativ. Somit kann eine Klassifikation mit

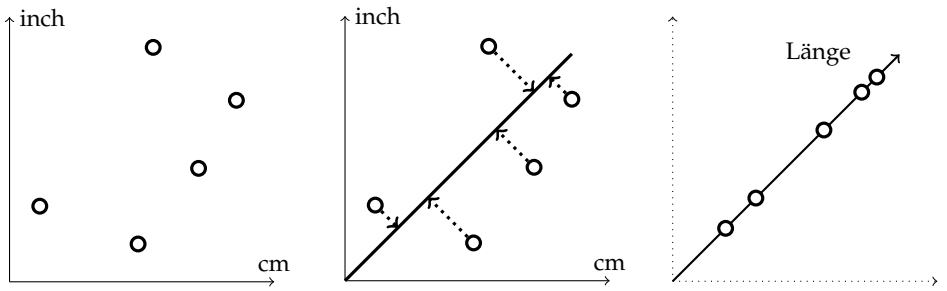
$$\hat{f}(\vec{x}) = \text{sign}\left(\left\langle \vec{\beta}, \vec{x} - \vec{a} \right\rangle\right) \in \{-1, +1\}$$

erfolgen und die beiden Klassen können auf -1 und $+1$ abgebildet werden.
Die Lernaufgabe wird somit wieder ein Optimierungsproblem bei dem die optimalen Parameter $\vec{\beta}$ und $\vec{a}$ gefunden werden sollen. Eine der berühmtesten Ansätze hierfür ist die *Support Vector Machine* [28]. Hierbei wird die trennende Hyperebene so gewählt, dass der kleinste Abstand zwischen $x \in \boldsymbol{X}_{\text{train}}$ und H maximal ist. In Abbildung 2.5 sind neben H zwei gleich weit entfernte, parallele Ebenen zu H abgebildet. Die Parameter werden nun dadurch ermittelt, dass die Breite des Streifens zwischen den beiden Ebenen maximiert wird.

2.2.3 Unüberwachtes Lernen

Im Gegensatz zu überwachtem Lernen, bei dem während des Lernens ein Label $y^{(i)}$ vorliegt, ist dies bei unüberwachtem Lernen nicht der Fall. Es werden also keine Zielwerte vorgegeben. Folglich gibt es nur einen Datensatz $\boldsymbol{X}$ und es wird nicht zwischen $\boldsymbol{X}_{\text{train}}$ und $\boldsymbol{X}_{\text{test}}$ unterschieden. Die Aufgabe besteht darin aus $\boldsymbol{X}$ Strukturen und Muster zu erkennen, um diese zusammenzufassen und möglicherweise wichtige Schlüsselattribute zu erklären.

Dimensionsreduktion Eine Möglichkeit hierfür bietet die Reduktion der Dimension eines Datensatzes. Die Anwendung solcher Verfahren kann unterschiedliche Gründe ha-



(a) Aufgrund von Rundungsfehlern korrelieren die beiden Längenmaße nicht und liegen damit nicht auf einer Geraden.

(b) Nach dem Vorbild der linearen Regression wird eine Gerade mit dem kleinsten Fehler ermittelt und die Punkte werden auf diese Gerade projiziert.

(c) Die auf die Gerade projizierten Punkte bilden eine neue zusammengefasste Dimension, die die Länge darstellt.

Abbildung 2.6: Ein Beispiel für die Dimensionsreduktion, wie sie beispielsweise bei der Hauptkomponentenanalyse durchgeführt wird

ben. Zum einen können Daten dadurch stark komprimiert werden und zum anderen hat dies oftmals Geschwindigkeitsvorteile für weitere Lernalgorithmen, wenn die Dimension des Datensatzes reduziert ist.

Sei $\boldsymbol{X}$ beispielsweise ein Datensatz von Personen, die mit zwei Attributen beschrieben werden. Attribut $\vec{x}_1^{(i)}$ ist die Größe der i-ten Person in Zentimeter und $\vec{x}_2^{(i)}$ die Größe in Zoll. Aufgrund von Rundungsfehlern bei der Datenerfassung liegen die Datenpunkte nicht auf einer perfekten Geraden, sondern sehen aus, wie in Abbildung 2.6 dargestellt. Werden die Punkte auf diese Gerade projiziert, dann kann diese Gerade auch als neue Achse, die allgemein die Größe darstellt, angesehen werden. Dieses so induzierte Attribut kann nun die anderen beiden Attribute ersetzen. In Datensätzen, die hunderte Dimensionen enthalten, teilweise aus verschiedenen Quellen aggregiert werden und somit auch stark redundante Attribute enthalten können, kann dies zu einer starken Reduktion der Dimension und Zusammenfassung führen. Ein Algorithmus zu solcher Dimensionsreduktion ist beispielsweise die **Hauptkomponentenanalyse**.

Clusteranalyse Die wichtigsten Verfahren sind hierbei die **Clustering**-Algorithmen. Die Aufgabe besteht dabei in der Gruppierung von Beispielen in einem Datensatz. Dabei sollen alle Beispiele innerhalb einer Gruppe möglichst „ähnlich“ und Beispiele in unterschiedlichen Gruppen möglichst „unähnlich“ zueinander sein. Da sich diese Arbeit näher mit Clustering beschäftigt, wird dieses Thema im Kapitel 2.3 weiterführend behandelt.

2.2.4 Verstärkendes Lernen

Ein weiterer Bereich des maschinellen Lernens ist das sogenannte verstärkende Lernen. Die grundlegende Idee dahinter ist angelehnt an die Konditionierung in der psychologischen Lerntheorie.
Das Ergebnis eines Algorithmus wird dabei beeinflusst, indem „gutes Verhalten“ belohnt und „schlechtes Verhalten“ bestraft wird. Das Lernen findet nun dadurch statt, dass die erwartete Belohnung maximiert wird.
Diese Art des Lernens ist vor allem dann notwendig, wenn es kein eindeutiges und „richtiges“ Label gibt, das wie beim überwachten Lernen vorhergesagt wird. Ein Algorithmus, der die Steuerung eines mehrbeinigen Laufroboters lernt, kann schwer über explizites Expertenwissen trainiert werden. Über eine Belohnungsfunktion, die das Vorwärtskommen belohnt und das Fallen bestraft, kann ein Algorithmus besser lernen, in welcher Situation welche Bewegung durchgeführt werden muss.

2.3 Clusteranalyse

Eine grundlegende mentale Fähigkeit von Menschen ist das Gruppieren von Entitäten. In der Psychologie wird dies Konzeptbildung genannt und ist wichtig für uns, um die enorme Informationsmenge, die jeden Tag von uns verarbeitet wird, effizient zu bewältigen. Objekte unserer Umwelt werden automatisch abstrahiert und kategorisiert, so dass wir nicht jedes mal darüber nachdenken müssen, was wir vor uns sehen und wie wir uns zu verhalten haben. Insbesondere für die Sprachentwicklung ist das Bilden von Gruppen unvermeidbar. Ein Kind muss zunächst verstehen, was Hunde gemeinsam haben und was sie von Katzen unterscheidet, um einen Hund auch Hund zu nennen.
Carl von Linné beschrieb die Relevanz der Kategorienbildung in seinem aus dem Jahre 1737 stammenden Werk „Genera Plantarum“ folgendermaßen:

> „Alles was sich als wirklich verschieden erkennen läßt, hängt von einer deutlichen Methode ab, nach welcher wir das Ähnliche von dem Unähnlichen unterscheiden. Je natürlicher die Unterscheidungszeichen dieser Methode sind; desto deutlicher werden unsere Begriffe von den Sachen. Je größer die Menge der Gegenstände ist, die unseren Verstand beschäftigen; desto schwerer ist es, eine Methode auszuarbeiten; aber eben so unentbehrlich wird sie auch.“ - Carl von Linné 1737 (deutsche Übersetzung [49])

Das Werk von Carl von Linné beschäftigt sich mit Pflanzengattungen und tatsächlich war die Biologie bzw. Zoologie auch die erste Wissenschaft, die Methoden der Gruppierung verwendete. In der Biologie wird dies jedoch Taxonomie genannt und ist wichtig, um die Pflanzen- und Tierwelt besser zu verstehen und Schlussfolgerungen ziehen zu können.

(a) Original **(b)** k=9 **(c)** k=3

Abbildung 2.7: Segmentierung eines Bildes mit Hilfe von KMEANS in k viele Farbgruppen

Aber auch in anderen Wissenschaften, wie Astronomie, Medizin, Soziologie, Geographie etc., ist dies eine wichtige Methode zum Erkenntnisgewinn.

Es gibt einen wichtigen Unterschied in dieser Methodik zwischen der frühen Wissenschaft und der heutigen. Während in den Anfangszeiten der Biologie oder auch der Astronomie der mühsame Teil der Arbeit darin bestand die Daten zu erfassen und das Auswerten noch per Hand geschah, hat sich dies mittlerweile geändert. Durch moderne Computer- und Sensortechnik können große Datenmengen innerhalb kürzester Zeit generiert werden, so dass die Schwierigkeit nun vielmehr in der Auswertung der Daten besteht. Das manuelle Bearbeiten dieser Daten ist nicht mehr möglich und Wissenschaftler verschiedenster Gebiete sind auf eine automatisierte und algorithmische Herangehensweise angewiesen. Und genau dieser Prozess ist die Aufgabe der Clusteranalyse: das automatisierte Finden von Gruppen von Objekten.

Neben der Anwendung der Clusteranalyse zum Verständnis der Daten, bei der die natürliche Struktur der Daten erfasst werden soll, wird sie auch als Hilfsmittel für weitere Verarbeitung eingesetzt. Das Abstrahieren von individuellen Objekten hin zu einer Charakterisierung durch Prototypen einer Gruppe stellt eine nützliche Zusammenfassung dar.

Ein Beispiel hierfür ist die Bildsegmentierung. Jedes Pixel eines Bildes hat gewisse Farb- bzw. Helligkeitswerte. Für diese Werte können Gruppen gefunden werden, für die ein Prototyp ermittelt werden kann. Jedes Pixel erhält dann die Farbe des Prototypen einer Gruppe in dem der Pixel enthalten ist. Wenn also 9 Farbgruppen in der Clusteranalyse gefunden werden, dann kann das Bild auf 9 Farben reduziert werden und kann somit deutlich einfacher komprimiert werden. In Abbildung 2.7 wurde ein Bild mit Hilfe des KMEANS Algorithmus segmentiert. In 2.7c wurde das ganze Bild auf 3 Farben reduziert durch das Finden von 3 Farbgruppen.

Nach der Taxonomie des maschinellen Lernens (siehe Kapitel 2.2) wird die Clusteranalyse auch unüberwachtes Lernen genannt. Es gibt jedoch eine natürliche Verbindung zwischen der Clusteranalyse und der Klassifikation beim überwachten Lernen. Die Clus-

teranalyse erstellt Gruppen von Beispielen und die Klassifikation weist neuen Entitäten Gruppen zu. Die Gruppenzugehörigkeit eines Beispiels ist letztendlich nichts anderes als ein maschinell erstelltes Label.

2.3.1 Definitionen

Das Ziel der Clusteranalyse ist es, für ein Datensatz $\boldsymbol{X} = \{x^{(1)}, \ldots, x^{(N)}\}$ die Beispiele $x^{(i)}$ so in Gruppen aufzuteilen, dass Beispiele innerhalb der selben Gruppe „ähnlich“ und Beispiele aus unterschiedlichen Gruppen „unähnlich“ sind. Der Begriff der Ähnlichkeit ist ein zentraler Begriff in der Clusteranalyse [72, 31].

Definition 2.3.1 (Ähnlichkeitsmaß). Eine Funktion $sim : \boldsymbol{X} \times \boldsymbol{X} \to \mathbb{R}$ ist ein metrisches **Ähnlichkeitsmaß**, wenn $\exists s_0 \in \mathbb{R} : \forall x, y \in \boldsymbol{X} : -\infty < sim(x, y) \leq s_0 < \infty$ und

$$\forall x \in \boldsymbol{X}: \quad sim(x, x) = s_0 \tag{2.1}$$

$$\forall x, y \in \boldsymbol{X}: \quad sim(x, y) = sim(y, x) \tag{2.2}$$

$$\forall x, y, z \in \boldsymbol{X}: \quad sim(x, z) \leq sim(x, y) + sim(y, z) \tag{2.3}$$

Je größer der Funktionswert von sim desto ähnlicher sind sich zwei Beispiele. Diese Funktion soll symmetrisch sein (Zeile 2.2) und die Dreieckungleichung (Zeile 2.3) erfüllen. Weiterhin wird ein Wert s_0 benötigt, der angibt, ab wann zwei Objekte gleich sind (Zeile 2.1). Es ist nicht ganz klar welchen Wert s_0 haben soll. Neben dem Ähnlichkeitsmaß kann aber auch noch ein Unähnlichkeitsmaß definiert werden.

Definition 2.3.2 (Unähnlichkeitsmaß). Eine Funktion $sim^{-1} : \boldsymbol{X} \times \boldsymbol{X} \to \mathbb{R}$ ist ein metrisches **Unähnlichkeitsmaß**, wenn $\exists d_0 \in \mathbb{R} : \forall x, y \in \boldsymbol{X} : -\infty < d_0 \leq sim^{-1}(x, y) < \infty$ für das gilt:

$$\forall x \in \boldsymbol{X}: \quad sim^{-1}(x, x) = d_0 \tag{2.4}$$

$$\forall x, y \in \boldsymbol{X}: \quad sim^{-1}(x, y) = sim^{-1}(y, x) \tag{2.5}$$

$$\forall x, y, z \in \boldsymbol{X}: \quad sim^{-1}(x, z) \leq sim^{-1}(x, y) + sim^{-1}(y, z) \tag{2.6}$$

Beim Unähnlichkeitsmaß sind sich also zwei Beispiele ähnlicher je kleiner die Funktion sim^{-1} wird. Die Anforderung an die Symmetrie und die Dreiecksungleichung bleiben gleich. Der Wert d_0 stellt den minimalen Unähnlichkeitswert dar, der mit $d_0 = 0$ auch sinnvoll gesetzt werden kann. Dies wäre dann die intuitive Vorstellung einer **Distanzfunktion**, bei der die „Entfernung“ zweier Beispiele zur Bestimmung der Ähnlichkeit verwendet wird.
Ist $\boldsymbol{X}$ ein endlichdimensionaler Vektorraum $\mathbb{K}^m$, dann ist eine Distanzfunktion definiert über die p-Norm.

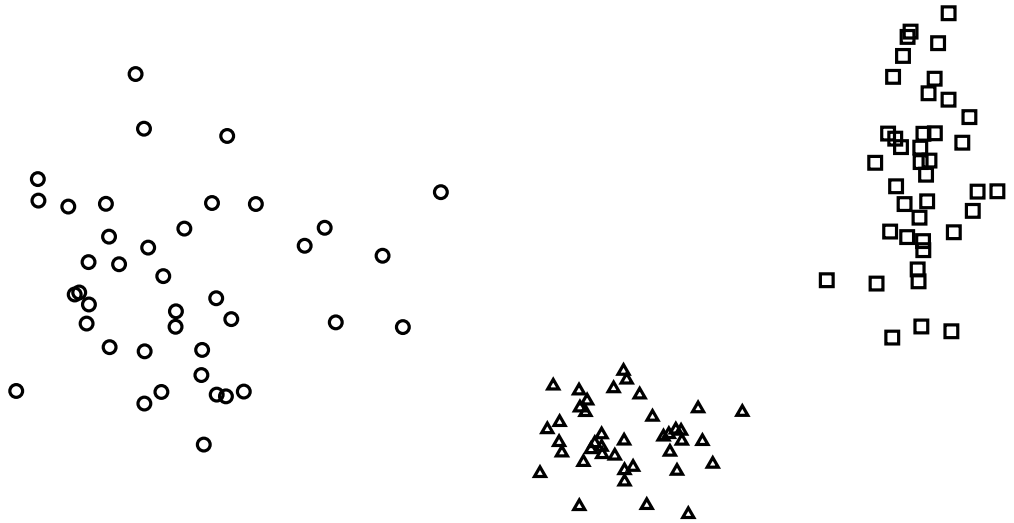


Abbildung 2.8: Ein Datensatz im zweidimensionalen und reellwertigen Raum lässt sich intuitiv unter der Verwendung der euklidischen Distanz in drei Cluster gruppieren.

Definition 2.3.3 (L_p-Norm). Sei $\mathbb{K}^m$ ein m-dimensionaler Vektorraum, $x \in \mathbb{K}^m$ und $p \geq 1$, dann ist

$$\|x\|_p = \left(\sum_{i=0}^{m} |x_i|^p \right)^{\frac{1}{p}}$$

die L_p-Norm des Vektors x.

Definition 2.3.4 (Distanzfunktion). Seien $x, y \in \mathbb{K}^m$ zwei Vektoren und $\|\cdot\|_p : \mathbb{K}^m \to \mathbb{R}$ eine L_p-Norm, dann ist

$$d(x, y) = \|x - y\|_p$$

eine Distanz- bzw. Abstandsfunktion.

Die Definition 2.3.4 bildet eine Klasse von Funktionen, die zur Abstandsbestimmung verwendet werden kann. In der Clusteranalyse werden meistens die Normen L_1 (**Manhattan Distanz**), L_2 (**Euklidische Distanz**) und L_∞ (**Tschebyschow Distanz**) verwendet.
Obwohl noch eine Vielzahl weiterer Maße für die Ähnlichkeit bzw. Unähnlichkeit existieren und auch ihre Anwendung findet, wird im folgenden der Begriff der Distanzfunktion verwendet. Falls nicht anders angegeben handelt es sich dann um die durch die L_2 Norm induzierte euklidische Distanz.

Problem 1 (Clusteranalyse). Sei $\boldsymbol{X} = \{x^{(1)}, \ldots, x^{(N)}\}$ ein Datensatz und $\boldsymbol{C} = \{\mathbf{C}_1, \ldots, \mathbf{C}_k\}$ eine Menge von Gruppen bzw. **Clustern** $\mathbf{C}_i$, dann finde ein **Clustering** $\mathcal{C} : \boldsymbol{X} \to \boldsymbol{C}$, so dass der **Intra-Cluster Abstand**

$$\sum_{\mathbf{C}_i} \left(\sum_{\mathcal{C}(x^{(a)}) \in \mathbf{C}_i} \left(\sum_{\mathcal{C}(x^{(b)}) \in \mathbf{C}_i} d(x^{(a)}, x^{(b)}) \right) \right)$$

minimal und der **Inter-Cluster Abstand**

$$\sum_{\mathbf{C}_i} \left(\sum_{\mathbf{C}_j \neq \mathbf{C}_i} \left(\sum_{\mathcal{C}(x^{(a)}) \in \mathbf{C}_i} \left(\sum_{\mathcal{C}(x^{(b)}) \in \mathbf{C}_j} d(x^{(a)}, x^{(b)}) \right) \right) \right)$$

maximal ist.

Das Problem 1 definiert die Clusteranalyse als ein Optimierungsproblem, dass die erste intuitive Idee widerspiegelt, Gruppen mit möglichst „ähnlichen“ Beispielen zu finden. Es sei trotzdem zu beachten, dass es sich hierbei nicht um eine präzise Definition aller Clustering-Algorithmen handelt. Manche Verfahren lösen nicht exakt dieses Problem, sondern weisen beispielsweise ein Beispiel mehreren Clustern zu. Auch dieses Vorgehen kann plausibel sein. Generell ist es bei der Clusteranalyse schwer zu entscheiden, welches Vorgehen richtig und welches falsch ist. Dies hängt stark vom Anwendungsgebiet ab.

Beispiel In Abbildung 2.8 ist ein Beispiel für ein Clustering mit $\boldsymbol{X} \subset \mathbb{R}^2$. Deutlich zu unterscheiden sind hier 3 unterschiedliche Cluster. Bei dieser Unterscheidung wird intuitiv die euklidische **Distanzfunktion** (L_2 Norm) als Maß für die Ähnlichkeit von Objekten verwendet.

2.3.2 Kategorisierung von Clusteringverfahren

Es gibt mehrere Ansätze, die Clusteringalgorithmen verfolgen und die sich teils stark voneinander unterscheiden. Sie lassen sich dabei in unterschiedliche Kategorien unterteilen [31, 72].
Eine erste Unterscheidung kann bezüglich der Art der Zuweisung eines Beispiels zu einem Cluster sein. **Partitionierende** Clusteringverfahren unterteilen die Daten in sich nicht überlappende Cluster. Jedes Beispiel ist also in genau einem Cluster. Der ganze Datenraum wird dabei vollständig partitioniert.
Nicht immer macht es jedoch Sinn ein Beispiel einem Cluster zuzuordnen. Bei echten Daten gibt es auch Beispiele, die als Ausreißer bzw. Rauschen aufgefasst werden können. Diese sollen keinem Cluster zugeordnet werden. Dieses **exklusive** Clustering kann jedoch auch als partitionierendes Clustering aufgefasst werden, wenn die Ausreißer eine eigenständige Gruppe bilden. Trotzdem müssen hierfür die Algorithmen angepasst werden, da solche Ausreißercluster keine typischen Cluster darstellen, sondern nur den Raum zwischen den restlichen Clustern füllen.
Weiterhin gibt es aber auch Situationen bei denen Ausreißer nicht berücksichtigt werden sollen und jedes Beispiel einem echten Cluster zugeordnet werden soll. **Überlappendes** Clustering kann einem Beispiel mehrere Gruppen zuweisen, wenn sich dieses zwischen zwei Clustern befindet.

Fuzzy Clustering weist jedem Beispiel $x^{(i)}$ alle Cluster zu. Diese „Mitgliedschaft“ zu einem Cluster $\mathbf{C}_j$ wird jedoch noch mit einem Gewicht $w_{\mathbf{C}_j}(x^{(i)}) \in [0,1]$ versehen, wobei 0 auf praktisch keine Relevanz zu diesem Cluster hindeutet und 1 einen starken Bezug ausdrückt. Zusätzlich wird noch oft gefordert, dass

$$\forall x^{(i)} \in \boldsymbol{X} : \sum_{\mathbf{C}_j} w_{\mathbf{C}_j}(x^{(i)}) = 1$$

gelten muss. Damit kann die Zugehörigkeit zu einem Cluster auch als Wahrscheinlichkeit aufgefasst werden, so dass dies oft auch **probabilistisches** Clustering genannt wird.
Ein weiterer Fall der mehrfachen Zuweisung ist das **hierarchische** Clustering. Cluster können hier auch Unter-Cluster beinhalten, in denen die Beispiele dann erneut zugewiesen werden. Dies ist dann ein Clustering auf mehreren Ebenen, wobei innerhalb einer Ebene oftmals ein partitionierendes Clustering durchgeführt wird.
Neben der Art der Zuweisung kann noch zwischen den verschiedenen Typen der Clusterrepräsentation unterschieden werden. Es stellt sich also die Frage, wie ein Cluster nun konkret dargestellt wird. Ein Clustering ist dabei beispielsweise die Menge aller Punkte, die zu sich selbst ähnlicher sind, als zu den Punkten anderer Gruppen. Neue Entitäten können dann dem Cluster zugewiesen werden, deren Beispiele am ähnlichsten sind. Sollen nicht alle Beispiele bei der Repräsentation gespeichert werden, ist eine weitere Möglichkeit, nur einen **Prototypen** eines Clusters zu speichern. Dies ist oftmals der Zentroid, also der Mittelwert aller zugehörigen Beispiele eines Clusters. Eine weitere Möglichkeit ist das **dichte-basierte** Vorgehen bei dem ein Cluster eine dichte Region von Beispielen ist, die von weniger dichten Regionen umgeben ist.
Im folgenden werden 2 bekannte Clustering-Algorithmen vorgestellt.

2.3.3 KMEANS

Einer der bekanntesten Clustering-Algorithmen ist KMEANS [53]. Es handelt sich dabei um ein Prototyp-basiertes und partitionierendes Verfahren. Der Benutzer muss hierbei die Anzahl der Cluster k angeben.
Das grundlegende Vorgehen ist dabei sehr simpel. Der Algorithmus 2.1 erstellt zunächst k initiale Zentroiden, die zufällig verteilt werden. Jeder Punkt des Datensatzes wird dem nächsten Zentroiden zugeordnet, wobei als Distanzfunktion in der Regel die euklidische Distanz verwendet wird. Anschließend werden die Zentroiden neu berechnet, indem der Mittelwert aller zugeordneten Beispiele ermittelt wird. Da sich die Zentroiden dabei bewegen können, wird nun überprüft, ob sich an der Zuordnung irgendetwas geändert hat. Falls ja, dann werden die Zentroiden erneut ermittelt, ansonsten endet KMEANS.
KMEANS ist stark von der initialen Wahl der k Zentroiden abhängig. Je nach Wahl können so auch unterschiedliche Clusterings entstehen. Die besten Ergebnisse entstehen wenn

Algorithmus 2.1 KMEANS

Eingabe: $X = \{x^{(1)}, \ldots, x^{(N)}\}, k \geq 1$
Ausgabe: k Zentroiden c_i
  Erstelle k viele zufällige, initiale Zentroiden c_i
  repeat
    Weise jedem Beispiel den nächsten Zentroiden c_i zu
    Berechne den Mittelwertvektor aller zugewiesenen Beispiele neu
  until Zentroiden ändern sich nicht
  return Zentroiden c_i

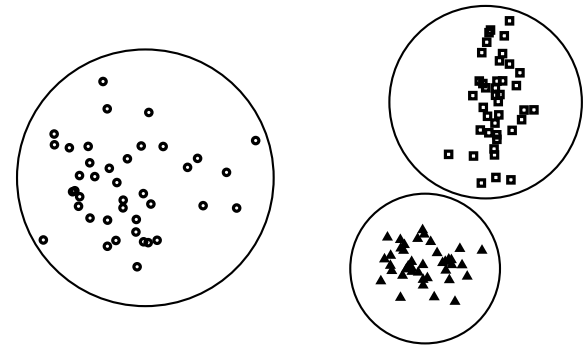
der Raum gut durch die Zentroiden abgedeckt wird. Im Allgemeinen wird auch eine Stichprobe aus k Beispielen gewählt, die die Zentroiden darstellen.

2.3.4 DBSCAN

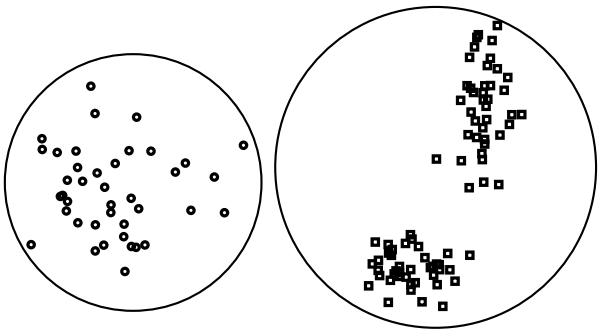
Ein dichte-basiertes Verfahren wird in [67] unter dem Namen DBSCAN vorgestellt. Dadurch, dass Cluster als dichte Regionen identifiziert werden, kann das Verfahren mit Rauschen in den Daten innerhalb der nicht dichten Regionen gut umgehen.
Die Dichte eines Beispiels wird als die Anzahl der Punkte in seiner Nachbarschaft definiert. Die Nachbarschaft ist dabei durch einen Radius angegeben. Durch diese Dichte können drei verschiedene Arten von Datenpunkten unterschieden werden. Ein sogenannter *Core Point* ist ein Punkt, dessen Dichte einen vorgegebenen Schwellwert überschreitet. Sind also genügend weitere Beispiele in der Nähe, dann wird ein Beispiel ein *Core Point*.
Ein *Border Point* dagegen ist ein Beispiel, das selbst kein *Core Point* ist, aber sich innerhalb seiner Nachbarschaft ein solcher befindet. Diese Punkte stellen damit das Randgebiet eines Clusters dar. Ein Punkt, der weder dicht, noch in der Nachbarschaft eines dichten Punktes liegt, wird als *Noise Point* bezeichnet.
DBSCAN ermittelt für jeden Punkt, um welche Punktart es sich dabei handelt und erstellt anhand dieser Klassifizierung die eigentlichen Cluster. Zwei *Core Point*s innerhalb ihrer Nachbarschaft sind dabei in einem Cluster. Dies setzt sich transitiv fort, so dass alle durch die dichte Nachbarschaft verbundenen Punkte einen Cluster bilden. *Border Point*s erhalten den Cluster ihres nächsten *Core Point*s. Die restlichen Punkte werden als Rauschen ignoriert.

2.3.5 Evaluation

Die Evaluation von gelernten Modellen ist bei vielen Lernaufgaben ein integraler Bestandteil und enthält wohldefinierte und akzeptierte Gütemaße. Bei der Clusteranalyse



(a) $k = 3$



(b) $k = 2$

Abbildung 2.9: Der gleiche Datensatz wird mit dem gleichen Algorithmus KMEANS aber mit unterschiedlichem Parameter analysiert. Beide Ergebnisse sind für sich plausibel und gruppieren die Daten.

ist die Situation schwieriger. Verschiedene Typen von Clusteringverfahren liefern oftmals sehr unterschiedliche Ergebnisse, die sich schlecht vergleichen lassen: Wie lässt sich beispielsweise ein hierarchisches Clustering mit einem Prototyp-basierten Clustering vergleichen?

Aber selbst, wenn die Methoden vom gleichen Typ sind, bleibt die Frage, welches Clustering besser ist und wieso? Dies ist selbst bei Ergebnissen vom gleichen Algorithmus jedoch mit einem anderen Parameter nicht einfach entscheidbar. Bei KMEANS beispielsweise muss zum Start die Anzahl der Cluster k angegeben werden. Hieraus ergeben sich unterschiedliche Clusterings mit unterschiedlichem Detailgrad. Abbildung 2.9 zeigt die Anwendung von KMEANS auf dem gleichen Datensatz mit unterschiedlichem k. Die beiden unterschiedlichen Gruppierungen widersprechen sich nicht, sondern stellen unterschiedliche Abstraktionsstufen dar. Dies kann man vergleichen mit den Clusterings, die beim hierarchischen Clusterverfahren erstellt werden. Für $k = 1$ erstellt KMEANS das generellste Clustering und für $k = N$ das speziellste.

Die Frage nach einem Maß, das beide Clusterings sinnvoll unterscheidet und bewertet, ist nicht allgemeingültig zu beantworten. Bei der Evaluation eines Clusterings werden

aus diesen Gründen Gütemaße auch Indizes genannt, die Informationen bezüglich verschiedener Eigenschaften eines Clusterings liefern. Es wird dabei zwischen externen und internen Indizes unterschieden [71].

Externe Indizes Ein Verfahren, das es zu evaluieren gilt, kann an Datensätzen getestet werden, für die schon Klassenlabels vorhanden sind. Eine Struktur der Daten ist also schon vorgegeben und mit einem Vergleich der Klassifikation mit dem Clustering kann ermittelt werden, wie gut das Clusteringverfahren diese Klassifikation nachbildet und sich damit auf Datensätzen verhält, die eine ähnliche Struktur haben. Ob Clusterings wie beispielsweise in Abbildung 2.9b oder 2.9a gut bewertet werden, entscheidet somit der Datensatz zur Evaluation.
Prinzipiell kann so die Güte eines Clusteringverfahren wie bei der Klassifikation gemessen werden, wobei darauf geachtet werden muss, dass nicht nur der Label-Name verglichen wird, da das Clustering für die ermittelten Gruppen vollkommen andere Namen wählen kann. Maße, die sich stark an die Gütemaße der Klassifikation orientieren, sind nach [71]:

Entropy Die Entropie ist ein gängiges Maß, das in der Informationstheorie entwickelt wurde und eine Aussage über die Informationsdichte macht. Für ein Clustering $\mathcal{C} = \{\mathbf{C}_1, \ldots, \mathbf{C}_k\}$ mit $\mathbf{C}_i \subseteq \boldsymbol{X}$ und einer Klassifikation $\mathcal{Y} = \{\boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_l\}$ mit $\boldsymbol{Y}_j \subseteq \boldsymbol{X}$ gibt die Entropie an wie viele Beispiele der gleichen Klasse $\boldsymbol{Y}_j$ einem Cluster $\mathbf{C}_i$ angehören. Für ein Cluster $\mathbf{C}_i$ ist die Entropie also definiert als

$$e_i = -\sum_{j=1}^{l} p_{ij} \log_2 p_{ij}$$

wobei p_{ij} die Wahrscheinlichkeit angibt, dass ein Objekt des Clusters $\mathbf{C}_i$ der Klasse $\boldsymbol{X}_j$ angehört mit

$$p_{ij} = \frac{|\mathbf{C}_i \cap \boldsymbol{Y}_j|}{|\mathbf{C}_i|}$$

Für die Gesamtentropie des Clusterings $\mathcal{C}$ ergibt sich dann durch $\boldsymbol{Y}$ die gewichtete Summe:

$$e = \sum_{i=1}^{k} \frac{|\mathbf{C}_i|}{N} e_i$$

Purity Bei diesem Maß wird jedes Cluster $\mathbf{C}_i$ der Mehrheitsklasse zugeordnet, indem

$$p_i = \max_j p_{ij} = \max_j \frac{|\mathbf{C}_i \cap \boldsymbol{Y}_j|}{|\mathbf{C}_i|}$$

berechnet wird. Es wird also die Klasse gewählt, die die höchste Wahrscheinlichkeit in diesem Cluster besitzt und diese Wahrscheinlichkeiten werden für alle Cluster gewichtet aufsummiert:

$$p = \sum_{i=1}^{k} \frac{|\mathbf{C}_i|}{N} p_i$$

	Gleicher Cluster	Unterschiedliche Cluster
Gleiche Klasse	f_{11}	f_{10}
Unterschiedliche Klasse	f_{01}	f_{00}

Tabelle 2.1: Kontingenztabelle für den Vergleich von Clustering und Klassifikation. f_{11} ist die Anzahl der Beispielpaare, die im gleichen Cluster sind und das gleiche Klassenlabel teilen.

Precision Ein schon bekanntes Maß aus der Klassifikation ist die *Precision* eines Clusters $\mathbf{C}_i$ bezüglich einer Klasse $\boldsymbol{Y}_j$ das mit

$$precision(\mathbf{C}_i, \boldsymbol{Y}_j) = p_{ij} = \frac{|\mathbf{C}_i \cap \boldsymbol{Y}_j|}{|\mathbf{C}_i|}$$

berechnet wird. Die *Precision* gibt also den Anteil einer Klasse innerhalb eines Clusters an.

Recall Ebenfalls aus der Klassifikation bekannt ist der *Recall*, der für eine Klasse $\boldsymbol{Y}_j$ seinen totalen Anteil innerhalb des Clusters $\mathbf{C}_i$ angibt:

$$recall(\mathbf{C}_i, \boldsymbol{Y}_j) = \frac{|\mathbf{C}_i \cap \boldsymbol{Y}_j|}{|\boldsymbol{Y}_j|}$$

F-measure Die beiden Maße *Precision* und *Recall* können kombiniert werden zu einem Maß, das angibt, wie sehr ein Cluster $\mathbf{C}_i$ nur Beispiele einer Klasse beinhaltet und wie viele aller Beispiele dieser Klasse in diesem Cluster sind:

$$F(\mathbf{C}_i, \boldsymbol{Y}_j) = \frac{2 \cdot precision(\mathbf{C}_i, \boldsymbol{Y}_j) \cdot recall(\mathbf{C}_i, \boldsymbol{Y}_j)}{precision(\mathbf{C}_i, \boldsymbol{Y}_j) + recall(\mathbf{C}_i, \boldsymbol{Y}_j)}$$

Neben den an die Klassifikation angelehnten externen Maßen gibt es noch Maße, die an der Ähnlichkeit der Beispiele untereinander orientiert sind. Bei einem guten Clustering sollte für jeweils zwei Beispiele aus $\boldsymbol{X}$ gelten, dass, wenn sie im gleichen Cluster sind, sie auch in derselben Klasse sein sollten und umgekehrt. Eine Möglichkeit dies auszudrücken ist die Γ **Statistik**. Hierfür werden zwei $N \times N$ Matrizen M_1 und M_2 erstellt, so dass für den Eintrag in Zeile i und Spalte j in M_1 gilt, dass dort eine 1 steht, wenn Beispiel $\vec{x}^{(i)}$ und $\vec{x}^{(j)}$ im gleichen Cluster sind und 0 sonst. Gleiches gilt bei M_2. Mit der Berechnung des Korrelationskoeffizienten der Daten von M_1 und M_2 kann nun die Γ Statistik berechnet werden. Je höher diese Korrelation ist, desto besser ist das Clustering.
Aufbauend auf diesem Prinzip kann mit Hilfe der Tabelle 2.1 der **Rand-Index**

$$rand = \frac{f_{00} + f_{11}}{f_{00} + f_{01} + f_{10} + f_{11}}$$

und der **Jaccard-Index**

$$jaccard = \frac{f_{11}}{f_{01} + f_{10} + f_{11}}$$

berechnet werden. Diese beiden Indizes bilden die am häufigsten verwendeten Evaluationsmaße für das Clustering [71].

Interne Indizes Während externe Indizes vorteilhaft sind, um neue Algorithmen auf bekannten Datensätzen zu evaluieren, muss ein Clusteringverfahren in der Praxis auf Datensätze angewandt werden, die unbekannt sind. Interne Indizes geben hier ohne zusätzliche Informationen zum Datensatz wichtige Hinweise zur Qualität des Clusterings an. Diese Informationen beziehen sich hauptsächlich auf die Kompaktheit der ermittelten Cluster, die **Cluster Kohäsion**

$$cohesion(\mathbf{C}_i) = \sum_{\substack{\vec{x}\in\mathbf{C}_i \\ \vec{y}\in\mathbf{C}_i}} proximity(\vec{x},\vec{y})$$

und Trennung der Cluster, die **Cluster Separation**

$$separation(\mathbf{C}_i,\mathbf{C}_j) = \sum_{\substack{\vec{x}\in\mathbf{C}_i \\ \vec{y}\in\mathbf{C}_j}} proximity(\vec{x},\vec{y})$$

Die Proximitätsfunktion *proximity* kann entweder über ein Ähnlichkeits- oder über ein Unähnlichkeitsmaß (siehe Definitionen 2.3.1 und 2.3.2) definiert werden. Abhängig von dieser Wahl, also ob s_0 oder d_0 die optimale Güte darstellen, ist, ob kleinere oder größere Werte der Kohäsion bzw. Separation besser sind. Für Distanzfunktionen (einem Unähnlichkeitsmaß) wäre ein möglichst kleiner Kohäsionswert und ein möglichst großer Separationswert gut.
Bei einem Prototyp-basierenden Clustering kann für ein Cluster $\mathbf{C}_i$ mit einem Zentroiden $\boldsymbol{c}_i$ die Berechnung schneller ausgeführt werden:

$$cohesion(\mathbf{C}_i) = \sum_{\vec{x}\in\mathbf{C}_i} proximity(\vec{x},\boldsymbol{c}_i)$$
$$separation(\mathbf{C}_i,\mathbf{C}_j) = proximity(\boldsymbol{c}_i,\boldsymbol{c}_j)$$

Zur allgemeinen Evaluation können die Kohäsions- und Separationswerte für jedes Cluster ermittelt und gewichtet aufsummiert werden. Die Summe kann gleichgewichtet sein oder die Größe der Cluster einbeziehen, aber auch eine quadratische Summe wird verwendet.
Es macht aber auch Sinn, diese Maße für einzelne Cluster zu verwenden, um ein berechnetes Clustering weiter zu analysieren und zu verbessern. Wenn die Kohäsion eines Clusters nicht sehr gut ist im Vergleich zu den anderen Clustern, dann kann dieser Cluster in weitere Cluster aufgeteilt werden. Bei zwei Clustern mit guter Kohäsion, aber schlechter Separation, können beide Cluster zu einem verschmolzen werden. Beide Methoden sind insbesondere bei einem Verfahren mit einer vorher festgelegten Anzahl von Clustern, wie z.B. bei KMEANS, nützlich.

Der **Silhouetten-Koeffizient**

$$silhouette(\vec{x}^{(i)}) = \frac{b_i - a_i}{\max(a_i, b_i)}$$

vereint die Kohäsion und Separation zu einem Index, der für die einzelnen Beispiele eines Clusters berechnet wird. Dabei ist a_i der durchschnittliche Abstand von $\vec{x}^{(i)}$ zu den anderen Punkten im gleichen Cluster und stellt damit den Kohäsionbegriff für ein Beispiel dar. Der Faktor b_i dagegen ist das Minimum der durchschnittlichen Abstände von $\vec{x}^{(i)}$ zu anderen Beispielen in Clustern in denen $\vec{x}^{(i)}$ nicht enthalten ist. Dies ist ähnlich zum Separationbegriff. Dieser Koeffizient ist im Intervall $[-1, +1]$, wobei ein negativer Wert auf ein prinzipiell schlechtes Clustering hindeutet, da dies bedeutet, dass $b_i < a_i$ ist und damit die Separation und Kohäsion des Beispiels nicht gut sind. Dieser Koeffizient kann dann auch auf ein ganzes Cluster angewendet werden. Ein guter Cluster hat für seine Beispiele gute Silhouetten-Koeffizienten.

Ähnlich zu der Korrelation der Matrizen M_1 und M_2 bei den externen Indizes kann der **Korrelationskoeffizient** zwischen der $N \times N$ Distanzmatrix D und der idealen $N \times N$ Matrix I als Maß verwendet werden. Die Distanzmatrix gibt an der Zeile i und der Spalte j den Abstand zwischen $\vec{x}^{(i)}$ und $\vec{x}^{(j)}$ an. In der Matrix I steht an dieser Stelle 0, wenn $\vec{x}^{(i)}$ und $\vec{x}^{(j)}$ im gleichen Cluster sind und 1 sonst. Eine hohe Korrelation dieser Matrizen deutet darauf, dass Beispiele in einem Cluster nahe sind.

Clustertendenz des Datensatzes Jedes Clusteringverfahren findet in den Daten Cluster und das unabhängig davon, ob es überhaupt Strukturen in den Daten gibt. Das bedeutet, dass die Algorithmen Cluster selbst in gleichverteilt zufälligen Daten finden. Wenn die Indizes auf ein schlechtes Verhalten hinweisen, kann dies also auch bedeuten, dass sich die Daten nicht gut gruppieren lassen und nicht, dass das Verfahren schlecht ist.

Maße der Clustertendenz versuchen mit Hilfe von statistischen Tests zu ermitteln, wie gut die Daten zur Gruppierung tendieren. Sie erstellen aber nicht direkt ein Clustering. Ein Beispiel hierfür ist die **Hopkins Statistik** [10]. Bei dieser Statistik werden gleichverteilt p Punkte im Datenraum gezogen und bilden die Menge U, sowie eine Stichprobe $W \subseteq \boldsymbol{X}$ mit der Kardinalität p. Für beide Mengen wird der Abstand zum nächsten Nachbarn in $\boldsymbol{X}$ berechnet. Sei u_i der Abstand des i-ten Punktes in U und w_i in W, dann ist die Hopkins Statistik

$$H = \frac{\sum_{i=1}^{p} w_i}{\sum_{i=1}^{p} u_i + \sum_{i=1}^{p} w_i}$$

Die Daten haben eine Tendenz zur zufälligen Verteilung, wenn die Abstände der Punkte zu den nächsten Nachbarn aus U und W ungefähr gleich sind. In diesem Fall ist $H \approx 0.5$. Werte um 1 oder 0 deuten auf gute Clustertendenz hin.

2.4 Finden häufiger Mengen

Das Finden häufiger Mengen stammt ursprünglich aus der **Assoziationsanalyse**. Ein weiterer gängiger Begriff, der den Ursprung dieser Analyse hervorhebt, ist **Warenkorbanalyse**. Die erste Erwähnung findet sich in [5] und ist seitdem Gegenstand vieler Publikationen.
Eine mögliche Datenbasis für so eine Analyse ist in Tabelle 2.2 als Transaktionsdatenbank dargestellt. Jede Zeile dieser Datenbank repräsentiert einen Einkauf in einem Supermarkt und wird Transaktion genannt. Die Besitzer des Supermarktes sind nun daran interessiert diese Daten zur Optimierung des Geschäftes zu nutzen. Oder anders ausgedrückt: Sie wollen die Warenkörbe ihrer Kunden analysieren, um Zusammenhänge im Einkaufsverhalten zu erkennen.
Ein bewährtes Mittel hierfür sind **Assoziationsregeln**. Mit den Daten aus Tabelle 2.2 als Basis ist eine plausible Regel, die erstellt werden kann:

$$\{\text{Bier}\} \rightarrow \{\text{Chips}\}$$

Wenn Personen Bier gekauft haben, dann wurden dazu auch oft Chips verkauft. Supermärkte verwenden dieses Wissen zur Sortierung und Platzierung ihrer Produktpalette.
Die Erkenntnis, dass Chips und Bier gemeinsam gekauft werden, ist nicht unbedingt überraschend. Bei einer großen Auswahl an Produkten ergeben sich dennoch oft interessante Zusammenhänge. Ein berühmtes Beispiel in der Fachliteratur ist die Geschichte einer angeblichen Studie, die durch solche Assoziationsregeln den Zusammenhang zwischen einer bestimmten Marke Bier und Windeln gefunden haben soll. Ob es diese Studie tatsächlich so gab ist Gegenstand vieler Diskussionen. Fest steht jedoch, dass durch diese Art der Analyse Zusammenhänge gefunden werden können, die nicht offensichtlich sind und gewinnbringend eingesetzt werden können.
Ein weiteres Beispiel für den Nutzen solcher Regeln ist das Empfehlungssystem von Einkaufsseiten wie beispielsweise Amazon. Das Einkaufsverhalten der Kunden kann dazu

TID	Produkte
1	{Bier, Chips, Milch}
2	{Milch, Zucker}
3	{Bier, Chips, Schokolade}
4	{Bier, Chips, Pampers}
5	{Bier, Chips}
6	{Bier}

Tabelle 2.2: Beispiel für eine Transaktionsdatenbank eines Supermarktes. Jede Zeile stellt eine Transaktion, also den Kauf der folgenden Produkte, dar.

TID	a_1	a_2	a_3	a_4	a_5	a_6
1	1	0	0	0	0	0
2	0	1	0	1	0	1
3	1	1	1	0	1	0
4	1	1	0	1	1	0
5	0	0	1	1	0	0
6	1	0	1	0	1	0

Tabelle 2.3: Beispiel für eine binäre Repräsentation einer Transaktionsdatenbank. Jede Zeile repräsentiert eine Transaktion, also beispielsweise den Einkauf eines Kunden. TID ist die Identifikationsnummer der Transaktion und die Attribute $a_1, \ldots, a_6$ stellen die Items dar.

benutzt werden, erstaunlich gute Empfehlungen bei der Suche und dem Einkauf auf solchen Seiten zu erstellen. Aber auch außerhalb des Marketings lässt sich die Assoziationsanalyse verwenden, um interessante Muster zu erkennen. Insbesondere in der Bioinformatik oder in der medizinischen Diagnostik können solche Verfahren eingesetzt werden. Der erste Schritt bei der Erstellung von Assoziationsregeln ist das Finden von Produkten, die häufig zusammen gekauft werden. Dies sind die gesuchten häufigen Mengen. Erst mit Hilfe dieser Mengen können aussagekräftige Regeln erstellt werden. Relevant für diese Arbeit sind jedoch nur die häufigen Mengen, so dass sich das folgende Kapitel nur mit dieser Thematik beschäftigt.

2.4.1 Definitionen

Die Daten für die Assoziationsanalyse werden häufig **Transaktionsdaten** genannt.

Definition 2.4.1 (Transaktionen). Sei $R = \{a_1, \ldots, a_p\}$ eine Menge von Items, dann ist $T = \{t_1, \ldots, t_N\}$ eine Menge von Transaktionen mit $t_i \subseteq R$.

Ein Beispiel für eine solche Transaktionsdatenbank ist in Tabelle 2.3 zu sehen. Jede Zeile repräsentiert eine Transaktion, also zum Beispiel einen Einkauf eines Kunden. Ein enthaltenes Element in einer Transaktion wird **Item** genannt und wird als binominales Attribut repräsentiert. Entweder ein Item ist Teil einer Transaktion oder nicht. Dies wird auch als eine binäre Repräsentation einer Transaktionsdatenbank bezeichnet.

Definition 2.4.2 (Support). Eine Transaktion t_i unterstützt eine Menge $I \subseteq R$, genau dann, wenn $I \subseteq t_i$. Dann ist der **Support** dieser Menge I definiert als

$$s(I) = |\{t \in T : I \subseteq t\}|$$

Der Support einer Menge I stellt damit die Häufigkeit dieser Menge I dar.

Definition 2.4.3 (Häufige Mengen). Für einen absoluten Schwellwert $\xi_{\text{abs}} \in \mathbb{R}_{\geq 0}$ bzw. relativen Schwellwert $\xi_{\text{rel}} \in [0,1]$ ist eine Menge I häufig, wenn

$$s(I) \geq \xi_{\text{abs}} \qquad \text{bzw.} \qquad \frac{s(I)}{|T|} \geq \xi_{\text{rel}}$$

und somit ist die Menge der häufigen Mengen

$$S(T, \xi_{\text{abs}}) = \{I \subseteq R : s(I) \geq \xi_{\text{abs}}\} \qquad \text{bzw.} \qquad S(T, \xi_{\text{rel}}) = \{I \subseteq R : \frac{s(I)}{|T|} \geq \xi_{\text{rel}}\}$$

Mit den bisherigen Definitionen kann nun das Problem des Findens von häufigen Mengen formalisiert werden.

Problem 2 (Finden von häufigen Mengen). Sei T eine Transaktionsdatenbank und $\xi_{\text{abs}} \in \mathbb{R}_{\geq 0}$ ein absoluter Schwellwert bzw. $\xi_{\text{rel}} \in [0,1]$ ein relativer Schwellwert, dann finde $S(T, \xi_{\text{abs}})$ bzw. $S(T, \xi_{\text{rel}})$.

Beispiel Für den Support der Mengen aus den Transaktionsdaten T in Tabelle 2.3 gilt:

$$\begin{aligned} s(\{a_1\}) &= 4 & s(\{a_1, a_2\}) &= 2 \\ s(\{a_2\}) &= 3 & s(\{a_1, a_3\}) &= 2 \\ &\vdots & &\vdots \\ s(\{a_6\}) &= 1 & s(\{a_1, \ldots, a_6\}) &= 0 \end{aligned}$$

Mit einem absoluten Schwellwert $\xi_{\text{abs}} = 3$ sind die häufigen Mengen

$$S(T, 3) = \{\{a_1\}, \ldots, \{a_5\}, \{a_1, a_5\}\}$$

2.4.2 Komplexität des Suchraumes

Das Finden häufiger Mengen scheint zunächst eine einfache Aufgabe zu sein, da hierfür „nur gezählt" werden muss. Zu beachten ist jedoch, dass der Suchraum des Problems 2 exponentiell in der Menge der Items ist (siehe auch Abbildung 2.10). Ein naives Vorgehen bei dem zunächst alle Teilmengen von R gebildet werden und für jede Teilmenge gezählt wird, welche Transaktionen diese unterstützen, müsste folglich $2^{|R|}$ viele Teilmengen zählen.
Bei Betrachtung von großen Datenmengen wird somit ein scheinbar einfaches Zählproblem schnell sehr schwierig. Angenommen ein Supermarkt hätte ein Sortiment von 1000 Produkten, dann ist die Anzahl zu zählender Teilmengen $\approx 10^{300}$. Zum Vergleich: die geschätzte Anzahl der Atome im Universum ist $\approx 10^{80}$. Große Einkaufshäuser im Internet haben noch weit mehr als 1000 Produkte im Sortiment.

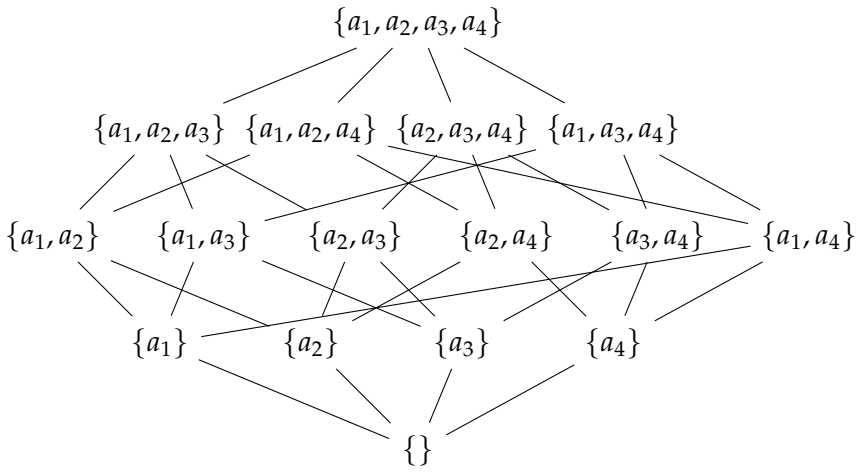


Abbildung 2.10: Der Suchraum der häufigen Mengen bildet mit der Teilmengenbeziehung einen Verband und ist exponentiell in der Anzahl der Items – hier $2^{|R|} = 2^{|\{a_1,a_2,a_3,a_4\}|} = 16$.

Es ist jedoch auch gar nicht nötig den gesamten Suchraum zu betrachten. Eine Möglichkeit den Suchraum einzuschränken ist es nur die Teilmengen zu betrachten, die sich aus den vorhandenen Transaktionen bilden lassen (siehe [9]). Dies schränkt den Suchraum für spärlich besetzte Datenbanken stark ein, ist für dichter besetzte Transaktionsdaten aber weiterhin nicht praktikabel.

2.4.3 APRIORI

Ein wichtiger Ansatz zur Einschränkung des Suchraumes wird bei APRIORI [7] genutzt. Die Kernidee hierbei ist die Verwendung einer simplen Eigenschaft der Beziehung zwischen Teilmengen und ihrer Häufigkeit.

Proposition 2.4.4 (Monotonie). *Sei T eine Transaktionsdatenbank mit den Items R, dann gilt für zwei Mengen $X, Y \subseteq R$*

$$X \subseteq Y \Rightarrow s(Y) \leq s(X)$$

Für die häufigen Mengen kann aus der Proposition 2.4.4 geschlossen werden, dass jede Teilmenge einer häufigen Menge ebenfalls häufig sein muss. Das bedeutet aber auch, dass jede Obermenge einer nicht häufigen Menge ebenfalls nicht häufig ist. Wenn ein Produkt selten gekauft wurde, dann waren auch alle Einkäufe, die dieses Produkt beinhalten, ebenfalls selten. Wie stark dadurch der Suchraum eingeschränkt wird, ist in Abbildung 2.11 zu sehen.
Diese Suchraumeinschränkung nutzt APRIORI indem zunächst die 1-elementigen häufigen Mengen L_1 ermittelt werden. Aus L_1 wird dann die Kandidatenmenge C_2 generiert.

Dies sind dann alle 2-elementigen Mengen, die überhaupt noch häufig sein können. Für diese Kandidaten muss die Häufigkeit durch Zählen in der Transaktionsdatenbank noch ermittelt werden. Die häufigen Kandidaten bilden die Menge L_2, aus denen dann die Kandidaten C_3 generiert werden. Dieser Vorgang wird solange wiederholt bis keine neuen Kandidaten mehr generiert werden (siehe auch Algorithmus 2.2).

Algorithmus 2.2 APRIORI

Eingabe: T, R, ξ_{abs}
Ausgabe: $S(T, \xi_{\text{abs}})$

```
k ← 1
C_1 ← {{I} : I ∈ R}
L_1 ← {I ∈ C_1 : |{t ∈ T : I ⊆ t}| ≥ ξ_abs}
while C_k ≠ {} do
    C_{k+1} ← APRIORIGEN(L_k, ξ_abs, T)
    L_{k+1} ← {I ∈ C_{k+1} : |{t ∈ T : I ⊆ t}| ≥ ξ_abs}
    k ← k + 1
end while
return ⋃_i L_i
```

Bei der Generierung der Kandidaten (Algorithmus 2.3) wird nicht jede Menge aus L_k mit jeder anderen Menge vereinigt, sondern nur die Mengen, die die selben $k-1$ Elemente haben. Dadurch, dass alle Teilmengen einer häufigen Menge ebenfalls häufig sein

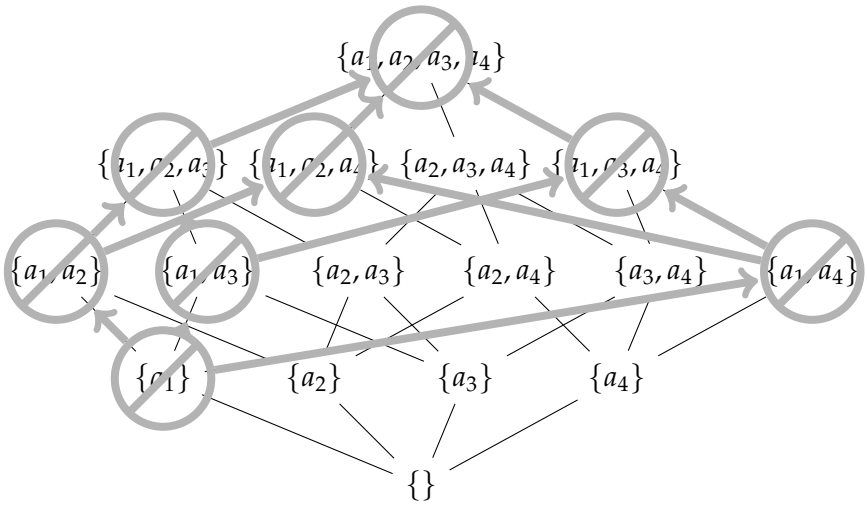


Abbildung 2.11: Durch die Monotonie-Eigenschaft wird der Suchraum eingeschränkt. Das Wissen über die Nicht-Häufigkeit von $\{a_1\}$ schließt die Häufigkeit für die Hälfte des Suchraumes aus (von 2^4 auf 2^3 mögliche Mengen).

müssen, werden alle möglichen Kandidaten so generiert. Anschließend werden noch alle k-elementigen Teilmengen des Kandidaten überprüft, ob sie in L_k sind.

Algorithmus 2.3 APRIORIGEN

Eingabe: L_k, ξ_{abs}, T
Ausgabe: C_{k+1}

```
C_{k+1} ← {}
for l_1 ← {i_1, ..., i_{k-1}, i_k} ∈ L_k do
    for l_2 ← {i_1, ..., i_{k-1}, i'_k} ∈ L_k do
        l ← {i_1, ..., i_{k-1}, i_k, i'_k}
        for c_k ∈ {k-elementige Teilmengen von l} do
            if c_k ∈ L_k then
                C_{k+1} ← C_{k+1} ∪ {l}
            end if
        end for
    end for
end for
return C_{k+1}
```

Nach der Veröffentlichung des APRIORI-Algorithmus erschienen viele Arbeiten, die sich mit dem Finden von häufigen Mengen beschäftigten. Denn auch wenn der Suchraum durch das Ausnutzen der Monotonie-Eigenschaft drastisch verkleinert wurde, ist die Laufzeit aus zwei Gründen noch problematisch.

Kandidatengenerierung Im Falle von vielen häufigen Mengen – beispielsweise bei einem kleinen Schwellwert – werden noch viele Kandidaten erzeugt, die überprüft werden müssen. Wenn zum Beispiel im ersten Lauf 10^4 1-elementige häufige Mengen gefunden wurden, werden 10^7 2-elementige Kandidaten generiert, deren Häufigkeit bestimmt werden muss. Auch das Finden von großen häufigen Mengen ist sehr aufwändig. Für eine 100-elementige häufige Menge müssen $2^{100} - 2 \approx 10^{30}$ Kandidaten generiert werden.

Mehrfacher Datenbankdurchlauf Für die Erstellung der Menge L_{k+1} muss die Häufigkeit der Kandidaten aus C_k bestimmt werden. Hierfür muss wiederholt aus der Datenbank gelesen werden. Dies ist insbesondere für große Datenmengen, die nicht in den Speicher passen, nicht praktikabel, da dies mit häufigen und sehr langsamen Plattenzugriffen verbunden ist.

TID	Items in Transaktion	Sortierte häufige Items in Transaktion
1	$\{f, a, c, d, g, i, m, p\}$	$\{f, c, a, m, p\}$
2	$\{a, b, c, f, l, m, o\}$	$\{f, c, a, b, m\}$
3	$\{b, f, h, j, o\}$	$\{f, b\}$
4	$\{b, c, k, s, p\}$	$\{c, b, p\}$
5	$\{a, f, c, e, l, p, m, n\}$	$\{f, c, a, m, p\}$

Tabelle 2.4: Für jede Transaktion wird eine Liste der häufigen Items erstellt und nach ihrer globalen Häufigkeit sortiert. Dies benötigt zwei Datenbankdurchläufe. Für dieses Beispiel ist $\xi_{\text{abs}} = 2$.

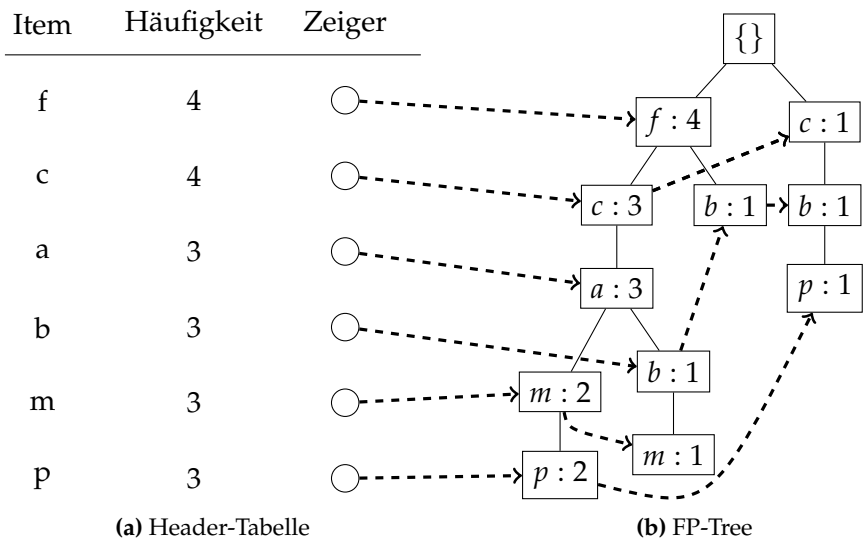
2.4.4 FPGROWTH

Eine der erfolgreichsten Folgearbeiten, die sich mit den Problemen der Kandidatengenerierung und Datenbankdurchläufen beschäftigte ist FPGROWTH[37]. Hierbei wird mit nur zwei Datenbankdurchläufen eine Baumdatenstruktur FPTREE (siehe Abbildung 2.12b) erstellt, die daraufhin rekursiv abgearbeitet wird, um die häufigen Mengen zu ermitteln. Bei diesem Abarbeiten werden keine Kandidaten generiert.

Baumdatenstruktur FPTREE

Der erste Datenbanklauf wird benötigt, um alle 1-elementigen häufigen Mengen zu ermitteln. Diese werden zusammen mit ihrer Häufigkeit gespeichert und nach dieser sortiert. Beim zweiten Lauf wird aus jeder Transaktion eine Liste der häufigen Items erstellt, die in dieser Transaktion enthalten sind. Diese Liste ist nach der globalen Häufigkeit sortiert (siehe Tabelle 2.4). Anschließend wird diese Liste dem FPTREE hinzugefügt.
Die Baumdatenstruktur wird anfangs mit einem leeren Wurzelknoten $\{\}$ erstellt. Jeder hinzugefügte Knoten speichert ein Item, einen Zähler und einen Zeiger auf einen anderen Knoten. Zusätzlich zur Baumstruktur wird noch eine Header-Tabelle benötigt, in der die häufigen Items mit ihrer Häufigkeit und einem Zeiger auf einen Knoten im Baum gespeichert werden.

Hinzufügen einer Transaktion Beim Hinzufügen einer sortierten Transaktionsliste zu einem FPTREE wird zunächst das erste Item der Liste gewählt. Dies ist das häufigste Item in der aktuellen Transaktion. Für dieses Item wird überprüft, ob ein Kind des Wurzelknotens dieses Item enthält. Falls dies der Fall ist, dann wird dessen Zähler um eins erhöht, ansonsten wird ein Knoten mit dem Zählerwert eins erstellt. Wurde ein neuer Knoten erstellt, wird als nächstes überprüft, ob die Header-Tabelle das aktuelle Item enthält. Wenn das Item enthalten ist, dann wird entlang des Zeigers traversiert bis ein Knoten erreicht wird, dessen Zeiger auf keinen weiteren Knoten zeigt. Für diesen Kno-



(a) Header-Tabelle **(b)** FP-Tree

Abbildung 2.12: Initiale Erstellung eines FP-Trees: Für jedes häufige Items gibt es einen Eintrag in der Header-Tabelle (sortiert nach der Häufigkeit) und einen Zeiger, der auf das erste Vorkommen des Items im FPTREE zeigt. Jeder Eintrag im Baum enthält ein Item, einen Zähler und einen Zeiger, der auf die nächste Position des selben Items im Baum zeigt.

ten wird ein Zeiger auf den Kindknoten des aktuellen Items erstellt. Falls es noch keinen Eintrag in der Header-Tabelle gibt, wird dieser erstellt, so dass der Zeiger direkt auf den Knoten mit dem aktuellen Item zeigt. Anschließend wird diese Prozedur mit dem nächsten Element der Liste und dem Kindknoten wiederholt. Dies geschieht so lange bis die Transaktionsliste leer ist. Ein Beispiel für einen so erstellten FPTREE ist in Abbildung 2.12 zu sehen. Der Algorithmus 2.4 zeigt die Methode FPINSERT, die für das Erstellen des FPTREEs zuständig ist.

Komprimierung durch FPTREE

Ein FPTREE ist im Prinzip ein Präfixbaum, der die Transaktionen als Wörter hinzufügt. Durch die Sortierung nach der Häufigkeit soll eine gute Komprimierung erreicht werden. Wie stark ein FPTREE komprimiert, hängt von den Daten ab. Im besten Fall sind alle Transaktionen gleich und der FPTREE besteht aus einem Pfad. Im schlimmsten Fall sind alle Transaktionen unterschiedlich. Dann ist der FPTREE sogar größer als die Originaldatenbank, da noch zusätzlicher Speicher für die Zeiger und Zähler benötigt wird. Im Unterschied zu APRIORI ist die Größe jedoch linear zur Datenbank beschränkt. Im schlimmsten Fall würde APRIORI exponentiell viele Kandidaten erstellen.

Algorithmus 2.4 FPINSERT

Eingabe: Liste von Items l, Knoten n eines FPTREE

```
i ← erstes Item von l
if n hat Kind c mit Item i then
    Inkrementiere Zähler von c
else
    Erstelle Kind c für n mit Zähler 1
    if Header-Tabelle des FPTREEs hat Eintrag für i then
        Folge den Zeigern bis zum Knoten p der keinen Zeiger mehr hat
        Erstelle Zeiger in p der auf c zeigt
    else
        Erstelle Eintrag in Header-Tabelle
        Erstelle Zeiger des Eintrages, der auf c zeigt
    end if
end if
Entferne erstes Element von l
if l nicht leer then
    FPINSERT(l, c)
end if
```

Ein weiteres Kriterium für die Komprimierung der Daten ist die Ordnung der Items. Die Häufigkeit wird gewählt, da hier die beste Komprimierung zu erwarten ist. Dies muss aber nicht immer der Fall sein [37].

Häufige Mengen mit Hilfe eines FPTREEs

Nach der Erstellung einer kompakten Datenstruktur werden die häufigen Mengen durch einen rekursiven Algorithmus aus dem FPTREE extrahiert. Dieser Algorithmus funktioniert nach dem *divide-and-conquer*-Prinzip.
Für jedes Item in der Header-Tabelle wird zunächst eine *conditional pattern-base* erstellt. Hierbei handelt es sich um einen Ausschnitt aus der Datenbank unter der Annahme, dass dieses Item in den Transaktionen enthalten ist. Hierfür folgt FPGROWTH dem Zeiger eines Items aus der Header-Tabelle und speichert für jeden Knoten im Baum die Items des Pfades an dem er hängt. Abhängig vom Zähler des Knotens wird dies wiederholt.
Beim FPTREE aus Abbildung 2.12 wäre dies für das Item m die Menge

$$\{\{f, c, a\}, \{f, c, a\}, \{f, c, a, b\}\}$$

Dies kann als vollkommen neue Datenbank angesehen werden bei der alle Transaktionen gespeichert sind, die m enthalten. Anschließend wird aus dieser neuen Datenbank

$$\{f, c, a\}$$
$$\{f, c, a\}$$
$$\{f, c, a, b\}$$

(a) *Conditional pattern-base* bezüglich m

(b) *Conditional* FPTREE bezüglich m

Abbildung 2.13: Aus Abbildung 2.12 erstellte *condtional pattern-base* und *conditional* FPTREE bezüglich m. 2.13a bildet eine neue Datenbank aus der 2.13b erstellt wird. Dabei wird b nicht hinzugefügt, da der absolute Schwellwert von $\xi_{\text{abs}} = 2$ nicht überschritten wird.

ein *conditional* FPTREE (siehe Abbildung 2.13) erstellt. Dabei werden die Transaktionen in einem neuen FPTREE hinzugefügt. Dabei müssen vorher alle nicht häufigen Items entfernt werden. Bei der Datenbank bezüglich m taucht b nur einmal auf und wird deshalb wegen $\xi_{\text{abs}} = 2$ nicht verwendet.

Auf dem so erzeugten FPTREE und der dazugehörigen neuen Header-Tabelle wird das Verfahren rekursiv aufgerufen. Dies geschieht so lange bis der FPTREE entweder leer ist oder nur noch aus einem Pfad besteht. Dabei ist zu beachten, dass bei den rekursiven Aufrufen die neuen FPTREEs bezüglich einer Menge von Items erstellt werden. Algorithmus 2.5 zeigt die Methode FPMINE, die die häufigen Mengen erzeugt.

Sobald der erzeugte FPTREE leer ist, können nun aus der Menge, zu der der FPTREE erstellt wurde, alle Teilmengen gebildet werden, die dann als häufige Menge ausgegeben werden. Falls der FPTREE aus nur einem Pfad besteht, kann dieser Vorgang ebenfalls abgebrochen werden und die Items in diesem Pfad werden ebenfalls zu der Erstellung der häufigen Mengen durch Teilmengenbildung verwendet. Der FPTREE in Abbildung 2.13b ist bezüglich m erstellt worden und besteht aus nur noch einem Pfad mit den Items f, c, a. Folglich sind alle Teilmengen von $\{f, c, a, m\}$ häufige Mengen. Für den kompletten Algorithmus 2.6 müssen nun für alle erstellten *conditional* FPTREEs die häufigen Mengen gesammelt und zurückgegeben werden.

Algorithmus 2.5 FPMINE

Eingabe: FPTREE, Präfix-Item-Menge I
Ausgabe: Häufige Mengen im FPTREE bezüglich I

```
if FPTREE ist leer then
    return alle Teilmengen von I
else if FPTREE hat nur einen Pfad then
    return alle Teilmengen der Items im Pfad ∪ alle Teilmengen von I
else
    for Item i ∈ Header-Tabelle von FPTREE do
        while Zeiger p von i nicht leer do
            i ← Ziel von p
            while Zähler c von i nicht 0 do
                Füge Präfixpfad von i zur conditional pattern-base P_I hinzu
                Dekrementiere Zähler c
            end while
        end while
        CONDFPTREE← conditional FPTREE von P_I
        I ← I ∪ {i}
        return FPMINE(CONDFPTREE, I)
    end for
end if
```

Algorithmus 2.6 FPGROWTH

Eingabe: T, ξ_{abs}
Ausgabe: $S(T, \xi_{\text{abs}})$

```
F ← häufige Items aus T sortiert nach Häufigkeit
FPTREE← {}
for t ∈ T do
    tf ← Liste häufiger Items aus t sortiert nach Reihenfolge in F
    FPINSERT(tf,FPTREE.{})
end for
return FPMINE(FPTREE, {})
```

2.5 Parallele Algorithmen

Im Jahre 1965 sagte Moore in [58] eine regelmäßige Verdoppelung der Schaltkreisdichte auf einem Computerchip voraus. Dies ist als das Mooresche Gesetz bekannt und wird heute oftmals als eine Verdoppelung der Rechengeschwindigkeit in Abständen von 12 bis 18 Monaten gedeutet. Während bisher eine Verdoppelung der Transistoren und Erhöhung der Taktfrequenz innerhalb eines Prozessors für die Erfüllung dieser Faustregel verantwortlich waren, stößt dieses Vorgehen langsam an physikalische Grenzen (siehe [65]).
Um weitere Leistungssteigerungen zu ermöglichen, werden Multikern- und Multiprozessorsysteme immer bedeutender. Dies sind Systeme in denen mehrere Berechnungen parallel durchgeführt werden. Das Problem ist hierbei, dass die bisherige Software von diesem Paradigmenwechsel nicht automatisch profitiert, wie sie es von immer schnelleren Prozessoren getan hat. Die Algorithmen müssen solche Systeme direkt unterstützen. Allerdings erweist sich das Entwickeln von parallelen Algorithmen als komplex und fehleranfällig (siehe [55]). Zusätzlich ist der Geschwindigkeitszuwachs eines parallelisierten Algorithmus beschränkt. Diesen Effekt beschreibt beispielsweise das Amdahlsche Gesetz [8]. Zusätzlich sind viele Algorithmen auch nicht vollständig parallelisierbar und müssen teilweise weiterhin seriell verarbeitet werden.
Ein weiteres Problem ist der Verwaltungsmehraufwand durch die Parallelisierung. Die einzelnen Prozessoren müssen sich je nach Algorithmus miteinander koordinieren und diese Kommunikation untereinander kostet Zeit. Dies verschärft sich zunehmend bei Parallelisierung, die über Rechnergrenzen hinaus stattfindet.

Flynnsche Taxonomie Es gibt viele verschiedene Architekturen, die eine parallele Berechnung ermöglichen. In [33] wurden Computer in vier verschiedene Bereiche eingeordnet, die eine grobe Einordnung der parallelen Architekturen ermöglicht. Diese Einteilung ist seitdem als die Flynnsche Taxonomie bekannt.

Single Instruction, Single Data (SISD) Dies ist das traditionelle von Neumann Modell bei dem ein Prozessor sequentiell einen Strom von Instruktionen abarbeitet. Jede Instruktion hat dabei Auswirkungen auf ein Datum im lokalen Speicher. Dies ist also das Modell, für das die traditionellen sequentielle Algorithmen geschrieben werden.

Single Instruction, Multiple Data (SIMD) Bei einem **SIMD** Modell wird ebenfalls ein Strom von Instruktionen angenommen, der sequentiell abgearbeitet wird. Jede dieser Instruktionen wird jedoch nicht von einem Prozessor auf dem selben Datum angewandt, sondern auf mehreren Daten gleichzeitig. Video- oder Bildbearbeitungsalgorithmen profitieren beispielsweise von solchen Architekturen, da oftmals die

gleiche Instruktion auf viele verschiedene Bildpunkte gleichzeitig ausgeführt werden muss. Folglich arbeiten Grafikkarten zumindest teilweise nach diesem Prinzip.

Multiple Instruction, Single Data (MISD) In diese Kategorie fallen nur sehr wenige und sehr spezielle Systeme. Mehrere Recheneinheiten bearbeiten dabei dasselbe Datum. Dies ist beispielsweise dann sinnvoll, wenn redundante Berechnungen durchgeführt werden müssen, die dann zur Fehlererkennung und -behebung verwendet werden. In der Praxis haben **MISD**-Architekturen jedoch kaum eine Bedeutung.

Multiple Instruction, Multiple Data (MIMD) Dies ist die gängigste Architektur, die zur parallelen Berechnung verwendet wird. Mehrere Prozessoren bearbeiten dabei verschiedene Daten. Dies ist beispielsweise bei Multikern- und Multiprozessorsystemen der Fall. Da sich solche Systeme denselben Speicher teilen, werden diese auch als *Shared Memory* Systeme bezeichnet. Demgegenüber stehen Systeme bei denen jeder Prozessor über eigenen Speicher verfügt, wie beispielsweise Cluster von vielen verschiedenen Rechnern. Der Vorteil gegenüber *Shared Memory* Systemen ist, dass ein Cluster deutlich besser skaliert. Das Hinzufügen von weiteren Rechnern ist also deutlich einfacher. Der Nachteil ist jedoch, dass die Kommunikationskosten und der Aufwand der Implementierung in diesem Fall deutlich höher sind.

2.5.1 MapReduce

Der Suchmaschinenbetreiber Google musste sich schon früh damit befassen enorme Datenmengen zu verarbeiten. Dabei handelt es sich um mehrere Petabyte an Daten, die nur noch unter Verwendung von massiver Parallelisierung effizient verarbeitet werden können. Inspiriert von funktionalen Programmiersprachen entwickelte Google ein Framework namens MapReduce und veröffentlichte es 2004 [29].

Das grobe Konzept von MapReduce, wie in Abbildung 2.14 skizziert, ist es, die Funktionalität eines Programmes in zwei Funktionen abzubilden: Eine Map- und eine Reduce-Funktion. Die Daten liegen dabei auf verschiedenen Rechner in einem Cluster. Jeder Rechner in diesem Cluster führt die Map-Funktion auf seinen Teil der Daten aus, erzeugt Zwischenergebnisse und verschickt diese anhand eines Schlüssels aggregiert weiter. Diese Ergebnisse werden sortiert nach den Schlüsseln zu verschiedenen Instanzen der Reduce-Funktion verschickt, so dass alle Ergebnisse eines Schlüssel innerhalb einer Funktion vorhanden sind. Die Aufgabe der Reduce-Funktion ist es nun, diese Zwischenergebnisse zu dem endgültigen Ergebnis zu aggregieren. Dabei ist es durchaus erlaubt, mehrere Ebenen von Map- und Reduce-Funktion zu verwenden, so dass die Ausgabe der Reduce-Schicht für andere Map-Funktionen als Eingabe dient.

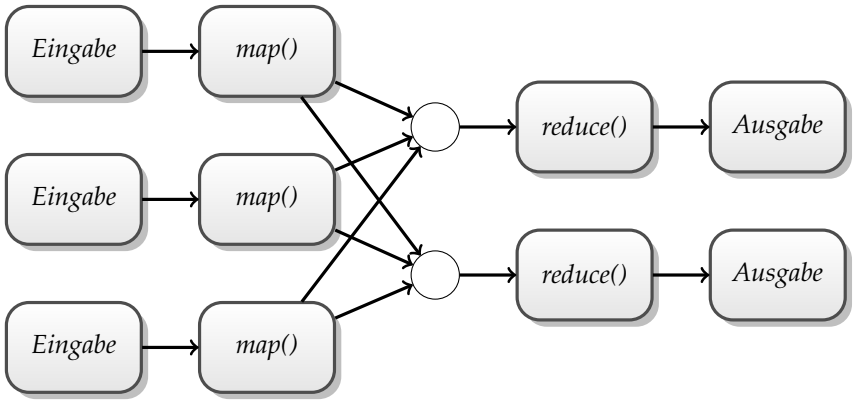


Abbildung 2.14: MAPREDUCE Konzept zur parallelen Berechnung

Die Aufgabe des Programmieres ist es nun, einen Algorithmus auf die zwei Funktionen abzubilden, während die MAPREDUCE-Infrastruktur sich sowohl um die Verteilung der Daten, als auch die Anwendung der Funktionen auf verschiedenen Rechner kümmert.
Ein sehr simples Beispiel eines solchen Programmes ist das Zählen von Wörtern, um die am häufigsten benutzen Wörter zu finden. Dies kann bei einer Suchmaschine dazu verwendet werden, die häufigsten Suchbegriffe zu ermitteln. Bei einer Datenmenge von mehreren Petabytes kann dies zu einem Problem werden. Insbesondere, da solche Datenmengen auch nicht auf einem einzelnen Rechner gesammelt, sondern schon über den Cluster verteilt sind. Jede Instanz der Map-Funktion erhält nun beispielsweise eine Text-Datei, die Wörter enthält und schon auf dem jeweiligen Rechner vorhanden ist. Für jedes Wort wird ein Schlüssel erzeugt. Der Wert, der diesem Schlüssel zugewiesen ist, ist die Häufigkeit des Wortes innerhalb dieses Textdokumentes. Die so erzeugten Zwischenergebnisse werden an verschiedene Reduce-Funktionen auf unterschiedlichen Rechnern im Cluster verteilt. Jede Funktion erhält für einen Schlüssel alle Zwischenergebnisse der Map-Funktionen und summiert sie zu gemeinsamen Zählerständen auf, die wiederum an eine weitere Reduce-Funktion verschickt werden, die alle Summen aller Reduce-Funktionen aggregiert und somit die k häufigsten Wörter ausgeben kann.
Der Vorteil an diesem Konzept ist die Abstraktion der eigentlichen Parallelisierung. Der Entwickler muss den Algorithmus zwar auf die Map- und Reduce-Funktionen abbilden, aber wenn dies geschehen ist, kann das Framework diese auf beliebig vielen Daten skaliert anwenden indem weitere Rechner zum Cluster hinzugefügt werden. Weiterhin kümmert sich das Framework dann um benötigte Datenredundanzen und fehlgeschlagene Prozesse. Bei einer Anzahl von mehreren tausend Rechner steigt die Wahrscheinlichkeit für einen Ausfall stark an. Solche müssen erkannt und durch erneute Berechnungen kompensiert werden.
Ein weiterer Vorteil ist, dass ein auf dieses Konzept abgebildeter Algorithmus auch auf einem Shared-Memory System verwendet werden kann. Bei einem lokalen MAPREDU-

CE-Framework übernimmt ein Prozess die Rolle eines Rechners im Cluster und bearbeitet seinen Teil der Daten im Speicher. Dies kann insbesondere dann sinnvoll sein, wenn die Daten noch von einem einzelnen Rechner mit unterschiedlichen Prozessoren bzw. Kernen parallel bearbeitet können, so dass die Größe der Daten die erhöhten Kommunikationskosten zwischen den einzelnen Rechnern nicht rechtfertigt. Aus der Sicht des Algorithmus ist es unerheblich, ob dieser auf einem Cluster oder lokal durch mehrere Prozessoren ausgeführt wird. Falls die Datenmenge ansteigt und eine weitere Skalierung notwendig ist, kann derselbe Algorithmus auf einem Cluster ausgeführt werden, der beliebig durch Hinzufügen von weiteren Rechner skaliert.
Ein bekanntes und nach dem Vorbild von Google erstelltes Open Source MAPREDUCE Framework ist HADOOP [17], das mit MAHOUT [60] auch eine Bibliothek von Verfahren des maschinellen Lernens enthält.

3

SUBSPACE CLUSTERING

3.1 Motivation

Durch immer neuere Technologien ergeben sich mehr Möglichkeiten, immer einfacher und schneller Daten zu erzeugen. Die Wichtigkeit dieser Daten kennzeichnet das Informationszeitalter. Wie aus Informationen Wissen entsteht ist in Kapitel 2.2 beschrieben, doch die traditionellen Ansätze zeigen mit der wachsenden Komplexität der Datensätze immer mehr Schwächen. Dabei ist insbesondere die größer werdende Dimensionalität der Daten problematisch, da sich dadurch nicht nur die Laufzeit der Algorithmen, sondern auch die Qualität der Ergebnisse verschlechtert.

3.1.1 Fluch der Dimensionalität

Für die qualitative Verschlechterung gibt es mehrere Gründe. Der Mensch entwickelt durch seine Wahrnehmung der Umwelt eine geometrische Intuition des zwei- und dreidimensionalen Raumes. Bei einem Blick auf einen zweidimensionalen Plot von Daten wird intuitiv die Distanz der Objekte verwendet, um diese zu gruppieren. Diese Wahrnehmung wurde in vielen traditionellen Algorithmen übernommen. Doch diese Intuition schlägt in höheren Dimensionen fehl. Anschauliche Beispiele zur Illustration der Verfahren lassen sich nicht immer auf höhere Dimensionen generalisieren.
Ein Beispiel für so eine falsche Intuition lässt sich anhand einer mehrdimensionalen Hyperkugel veranschaulichen (siehe [13]). Das Volumen einer Hyperkugel mit Radius r wird berechnet mit

$$V_p(r) = K_p r^p$$

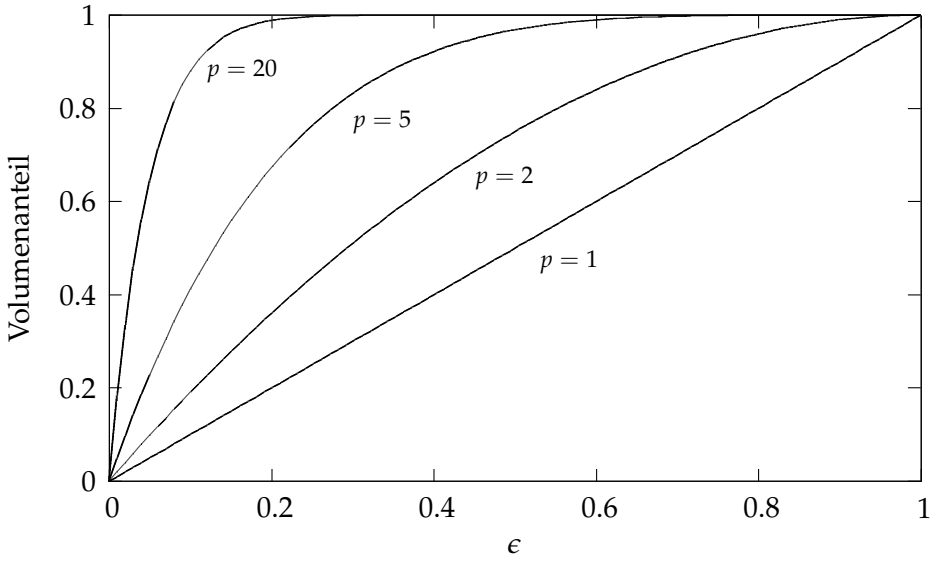


Abbildung 3.1: Volumenanteil in der Hülle einer Hyperkugel zwischen dem Radius $r = 1$ und $r = 1 - \epsilon$ in unterschiedlichen Dimensionen.

wobei K_p ein von der Dimension p abhängiger Term ist. Dann ist der Volumenanteil V_{frac} der Hyperkugel zwischen dem Radius $r = 1$ und $r = 1 - \epsilon$

$$V_{frac} = \frac{V_p(1) - V_p(1-\epsilon)}{V_p(1)} = 1 - (1-\epsilon)^p$$

In Abbildung 3.1 ist V_{frac} abhängig von ϵ für mehrere Dimensionen abgebildet. Zu beobachten ist, dass für $\epsilon = 0.1$ der Volumenanteil im 20-dimensionalen Raum schon 80% beträgt. Das bedeutet, dass mit steigender Dimensionalität der Großteil des Volumens in einer dünnen äußeren Schicht liegt. Folglich liegt der Großteil von gleichverteilten Daten in einer hochdimensionalen Hyperkugel ebenfalls in einem dünnen Randbereich der Hyperkugel. Für Verfahren , die auf die Lokalität und Nachbarschaft der Beispiele basieren, muss dies berücksichtigt werden, sobald auf hochdimensionalen Daten gearbeitet werden soll.

Seit [11] ist dieses Phänomen als **Fluch der Dimensionalität** bekannt. Eine weitere Beobachtung ist die steigende Spärlichkeit der Daten. Eine Erhöhung der Dimensionalität führt zu exponentiellem Wachstum des Volumens. Das Volumen eines Hyperwürfels mit Kantenlänge a ist beispielsweise a^p und um darin eine gleiche Datendichte zu erlangen, müsste die Anzahl der Beispiele auch exponentiell steigen. Dies ist aber selten der Fall. Bei den Microarray-Daten (siehe Kaptiel 2.1.4) gibt es beispielsweise viele Merkmale, jedoch nur sehr wenige Beispiele. In [12] wurde sogar gezeigt, dass mit der Erhöhung der Dimensionalität der kleinste Abstand zwischen zwei Punkten sich dem größtem Abstand annähert. Wenn sich die Abstände der Punkte untereinander kaum noch unterscheiden, dann ist dies vor allem für distanzbasierte Verfahren problematisch.

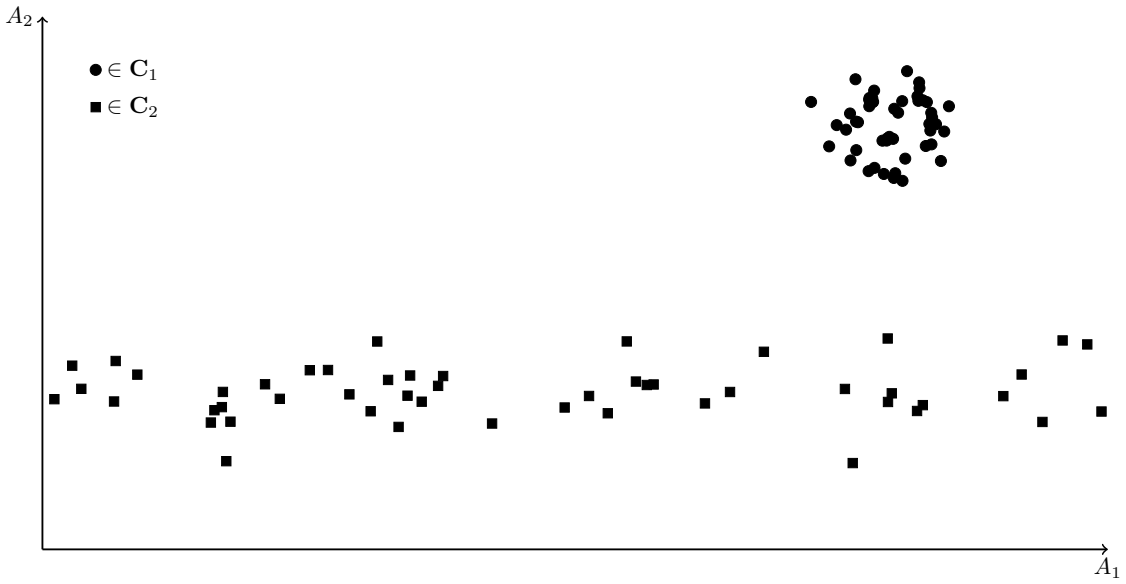


Abbildung 3.2: Beispiele aus zwei Clustern $\mathbf{C}_1$ und $\mathbf{C}_2$. Bei den Beispielen aus $\mathbf{C}_1$ sind beide Attribute A_1 und A_2 normalverteilt, bei $\mathbf{C}_2$ ist nur A_2 normalverteilt. Attribut A_1 ist dagegen über den gesamten Bereich gleichverteilt. In beiden Cluster sind gleich viele Beispiele enthalten, durch das Rauschen ist $\mathbf{C}_2$ jedoch nicht so kompakt wie $\mathbf{C}_1$ und ermöglicht deswegen kein so gutes Clustering.

Rauschen der Daten Moderne Technologie führt nicht nur dazu, dass die Anzahl der Datenbeispiele größer wird, sondern auch die Dimensionalität. Durch beispielsweise immer mehr Sensoren können mehr Merkmale erfasst werden, in der Erwartung, dass diese Merkmale unter Umständen relevant sind. Dies vergrößert aber auch die Wahrscheinlichkeit, dass es viele Merkmale gibt, die nicht relevant sind.

Manche Merkmale fügen dem Datensatz beispielsweise nur gleichverteiltes Rauschen hinzu. Dies erschwert das Finden von kompakten Clustern, da solche Attribute die Daten „auseinander ziehen“. Zudem kann dies auch nur für eine bestimmte Gruppe gelten. Für weitere Cluster kann dieses Merkmal wieder relevant sein. In Abbildung 3.2 gibt es beispielsweise zwei Cluster. Für Cluster $\mathbf{C}_1$ führen beide Attribute zu einem guten und dichten Cluster. Beim zweiten Cluster $\mathbf{C}_2$ zeigt sich, dass das Attribut A_1 nur ein gleichverteiltes Rauschen hinzufügt. Bei der alleinigen Verwendung des Attributes A_2 entsteht jedoch ein kompakter Cluster.

3.1.2 Dimensionsreduktion

Um die auftretenden Probleme einer hohen Dimensionalität zu verringern, können Methoden der Merkmalsselektion und -transformation angewandt werden. Dies geschieht als Vorverarbeitung vor dem eigentlichen Clustering.

Bei der Selektion von Merkmalen werden diejenigen Attribute ermittelt, die die höchste Relevanz gegenüber der jeweiligen Lernaufgabe haben [14, 69]. Solche Methoden verbessern die Ergebnisse bei manchen Datensätzen, doch lösen sie nicht das grundsätzliche Problem. Es handelt sich dabei um eine sogenannte globale Methode, die für ein Clustering auf dem ganzem Merkmalsraum angewandt wird. Verschiedene Cluster können jedoch in unterschiedlichen, sich überlappenden Teilmengen des Merkmalsraumes vorhanden sein. Damit kann jedes Merkmal für ein bestimmtes Cluster relevant und für andere Cluster irrelevant sein, so dass sich die Merkmalsselektion nur bedingt eignet.
In Kapitel 2.2.3 wurde als eine Möglichkeit zur Reduzierung der Dimensionalität die Hauptkomponentenanalyse vorgestellt. Dabei werden vorhandene Merkmale, die stark miteinander korrelieren, zu einem neuen Merkmal kombiniert. Es findet also eine Transformation der Merkmale statt, die zwar ebenfalls zu einer Reduktion der Merkmale führt, jedoch auch problematisch ist. Zum einen werden bei diesem Verfahren die Merkmale nicht vollständig entfernt, sondern nur in andere überführt. Das bedeutet aber auch, dass ein Rauschen der Daten weiterhin vorhanden ist und ein gutes Clustering erschwert. Darüber hinaus sind die neu erzeugten Merkmale nicht immer so intuitiv interpretierbar, wie beim Beispiel mit den Größen in Kapitel 2.2.3. Oftmals sind Cluster, die in einer echten Teilmenge der Merkmalen gefunden werden, interessanter als in einer Menge von neu erzeugten Merkmalen, da aus der Menge der ursprünglichen Merkmale bessere Rückschlüsse auf die ursprünglichen Daten gezogen werden können.
Es zeigt sich somit, dass der Fluch der hohen Dimensionalität nicht bloß durch das Hinzufügen von Vorverarbeitungsschritten gelöst werden kann, sondern neue Verfahren benötigt werden. **Subspace Clustering** ist der allgemeine Begriff für Clustering Verfahren, die die Reduktion der Dimensionalität lokal durchführen und integrieren.

3.2 Definitionen

Einen Überblick zur Clusteranalyse mit integrierter Behandlung von hochdimensionalen Daten liefert [46]. Dabei wird das Ziel dieser Analyse folgendermaßen definiert.

Definition 3.2.1 (Clusteranalyse auf hochdimensionalen Daten). Das generelle Ziel von Clusteralgorithmen für hochdimensionale Daten ist das Finden von Clustern in beliebig orientierten Unterräumen des ursprünglichen Merkmalraumes.

Der Begriff des beliebig orientierten Unterraumes ist sehr allgemein gefasst und soll verdeutlichen, dass nicht nur Teilmengen des Merkmalraumes gewählt werden können, sondern auch beliebige Linearkombinationen der Merkmale. Dies wird auch **Correlation Clustering** genannt und ist eine Klasse von Algorithmen, die die Merkmalstransformation integrieren. Beispiele für Verfahren, die auf der Hauptkomponentenanalyse basieren, sind [3, 16, 1].

Auch wenn das Entfernen von Rauschen durch zusätzliche, eingebettete Merkmalsselektion gewährleistet werden kann, bleibt je nach Anwendungsfall ein entscheidender Nachteil beim Correlation Clustering. Eine leichte Interpretierbarkeit der Ergebnisse ist bei beliebigen Linearkombinationen der Merkmale weiterhin nicht gegeben.

Alternativ hierzu können die Daten aus dem ursprünglichen Merkmalsraum in einen Unterraum projiziert werden, der achsenparallel ist. Anders ausgedrückt bedeutet das, dass eine echte Teilmenge der ursprünglichen Attribute gewählt wird, die dann einen neuen Raum induziert. In diesem Raum soll sich dann der Datensatz besser gruppieren lassen. Für solche Methoden gibt es in der gängigen Literatur keine einheitliche Bezeichnung. Die gängigen Begriffe sind **Axis-parallel Clustering**, **Projected Clustering** oder **Subspace Clustering**, wobei Subspace Clustering auch als Oberbegriff verwendet wird, der auch das Correlation Clustering umfasst. In [4] wird für das projizierende Clustering der Begriff Subspace Clustering verwendet und da sich die vorliegende Arbeit zu einem großen Teil auf [4] bezieht, wird im Folgenden die Bezeichnung Subspace Clustering verwendet.

In der englischen Literatur wird der Begriff **Subspace** für einen Unterraum des ursprünglichen Raumes verwendet. Zu beachten ist jedoch, dass ein Unterraum in diesem Kontext nicht vollständig der allgemeinen Definition des mathematischen Unterraumes bzw. Untervektorraumes entspricht. In dieser Arbeit wird folgende Definition verwendet.

Definition 3.2.2 (Subspace bzw. Unterraum). Für einen Raum $\mathbb{A}^p = A_1 \times \ldots \times A_p \subseteq \mathbb{R}^p$ mit den Attributen $\boldsymbol{A} = \{A_1, \ldots, A_p\}$ ist $\mathbb{S}^k = A_{t_1} \times \ldots \times A_{t_k}$ mit $t_k \leq p$ und $\forall i < j : t_i < t_j$ und den Attributen $\boldsymbol{S} = \{A_{t_1}, \ldots, A_{t_k}\} \subseteq \boldsymbol{A}$ ein Subspace bzw. Unterraum von $\mathbb{A}^p$.

Es handelt sich also um einen Unterraum, der durch eine Teilmenge der Attribute induziert wird. Darüber hinaus wird noch eine Projektion der Beispiele von $\boldsymbol{X}$ in diesen neuen Raum $\mathbb{S}$ benötigt.

Definition 3.2.3 (Projektion auf einen Subspace). Sei $\boldsymbol{X}$ ein Datensatz aus $\mathbb{A}^p$ und $\mathbb{S}^k$ ein Subspace von $\mathbb{A}^p$, so ist $\pi : \mathbb{A}^p \to \mathbb{S}^k$ eine orthogonale Projektion von $\boldsymbol{X}$ auf den Unterraum $\mathbb{S}^k$, falls $\forall \vec{x}^{(i)} \in \boldsymbol{X} : \vec{x}^{(i)} - \pi(\vec{x}^{(i)}) \perp \mathbb{S}^k$.

Mit Hilfe dieser Projektion wird nun ein Subspace Cluster definiert. Dabei handelt es sich um einen Cluster, der gefunden wird, wenn $\boldsymbol{X}$ auf einen Subspace projiziert wird.

Definition 3.2.4 (Subspace Cluster). Sei $\boldsymbol{X}$ aus $\mathbb{A}^p \subseteq \mathbb{R}^p$ und $\mathbb{S}^k$ ein Subspace von $\mathbb{A}^p$ mit den Attributen $\boldsymbol{S}$, dann ist $\mathbf{C} = (\boldsymbol{X}, \boldsymbol{S})$ ein Subspace Cluster, falls $\boldsymbol{X}$ projiziert auf $\mathbb{S}$ nach einem Ähnlichkeits- oder Unähnlichkeitsmaß einen Cluster bildet.

3.2.1 Top-Down und Bottom-Up Verfahren

Die Restriktion von beliebig orientierten Unterräumen zu Subspaces beschränkt den zu durchsuchenden Suchraum. Für einen p-dimensionalen Datensatz $\boldsymbol{X}$ ist die Anzahl der möglichen m-dimensionalen Subspaces $\binom{p}{m}$ für $1 \leq m \leq p$. Damit ist die Anzahl aller möglichen Subspaces

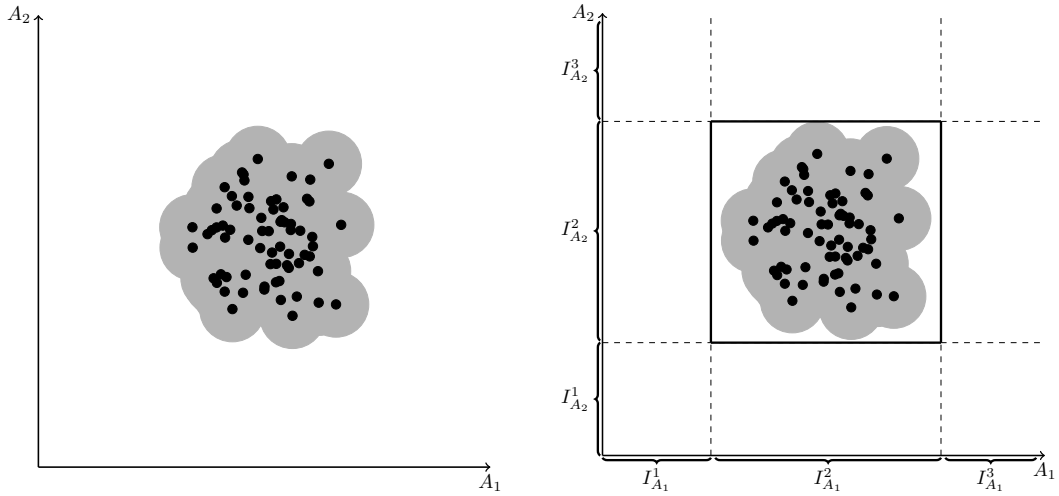
$$\sum_{m=1}^{p} \binom{p}{m} = 2^p - 1$$

und damit exponentiell zur Dimensionalität der Daten. Ein naives Vorgehen, das über alle Subspaces iteriert und dort nach Clustern sucht, ist damit für hochdimensionale Daten nicht praktikabel. Nach [46] gibt es dabei 2 Ansätze, um den exponentiellen Suchraum zu erforschen.

Top-Down Bei den sogenannten **Top-Down**-Ansätzen werden zunächst Approximationen von Clustern im vollständigen Merkmalsraum gesucht. Jeder Cluster erhält für jedes Merkmal ein Gewicht, das zunächst für alle Attribute gleich ist. In mehreren Iterationen werden für jeden Cluster die Gewichte angepasst und damit im Laufe der Zeit bestimmte Subspaces ausgewählt. Die Anpassung der Gewichte folgt dabei bestimmten Gütekriterien für die jeweiligen Cluster. Beispiele für diesen Ansatz sind [2, 15].

Bottom-Up Im Gegensatz dazu gruppiert ein **Bottom-Up**-Ansatz zunächst die Daten für jedes einzelne Merkmal. Dabei werden Clusterkriterien verwendet, die eine Eigenschaft aufweisen, die der Monotonieeigenschaft der häufigen Mengen ähnlich ist. Wenn die Daten $\boldsymbol{X}$ sich in einem Subspace mit den Attributen $\boldsymbol{S}$ gruppieren lassen (beispielsweise bezüglich eines Dichtekriteriums), dann muss dies auch für alle Subspaces mit den Attributen $\boldsymbol{T} \subset \boldsymbol{S}$ gelten. So kann ähnlich zu APRIORI (siehe Kapitel 2.4.3) der Suchraum stark eingegrenzt werden. Verfahren, die diesem Ansatz folgen, sind beispielsweise [4, 25, 59, 50, 43].

Dadurch, dass Top-Down Algorithmen zunächst auf allen Merkmalen ein initiales Clustering erzeugen, haben solche Verfahren Probleme mit der Laufzeit [61]. Um dieses Verhalten zu verbessern, müssen Sampling- und Approximationstechniken verwendet werden. Weiterhin werden in vielen Fällen schwer zu ermittelnde Parameter, wie beispielsweise die Anzahl der Cluster und die Größe der Subspaces benötigt. Nach [61, 46] finden Top-Down-Algorithmen partitionierende Cluster in sich nicht überlappenden Subspaces. Dies ist für manche Anwendung nützlich, wenn z.B. eine Segmentierung der Daten vorgenommen werden soll (dies ist oftmals im Marketing der Fall). Bei explorativen und wissenserzeugenden Aufgaben ist diese Restriktion nicht vorteilhaft. Es lässt sich jedoch sagen, dass Top-Down-Verfahren performanter sind als Bottom-Up Ansätze, wenn möglichst große Cluster in großen Subspaces gefunden werden sollen [61]. Dies spiegelt ei-



(a) Beispiele und ihre gesamte Nachbarschaft, die als dichte Region aufgefasst werden kann

(b) Eine dichte Region wird eingegrenzt durch zwei Intervalle $I^2_{A_1}$ und $I^2_{A_2}$.

Abbildung 3.3: Ein Cluster kann auch durch Intervalle repräsentiert werden.

nes der Probleme des APRIORI Algorithmus wieder, der für große häufige Mengen nicht effizient ist, da hierfür exponentiell zur Kardinalität der zu findenden Menge viele Teilmengen untersucht werden müssen (siehe Kapitel 2.4.3).

3.2.2 Subspace Clustering und häufige Mengen

Der Ansatz bei den Bottom-Up-Algorithmen deutet an, dass es eine starke Verbindung zwischen häufigen Mengen und Subspace Clustering gibt (siehe auch [73]). Ein Attribut wird hierbei als Item aufgefasst, für das die Häufigkeit gleich 1 ist, falls für dieses Attribut ein Cluster nach dem Clusterkriterium gefunden wurde und 0 sonst. Mit einem Schwellwert von 1 bilden die häufigen Mengen die Subspaces, in denen Cluster gefunden werden können.
Dies ist eine Art Metaverfahren, da für die einzelnen Subspaces weiterhin ein traditioneller Clustering-Algorithmus gestartet wird, um Cluster zu ermitteln. Es gibt aber noch eine direktere Verbindung. Die Abbildung 3.3a zeigt ein Clustering, dass durch ein dichtebasiertes Verfahren, wie beispielsweise DBSCAN ermittelt wurde. Das umrandete Gebiet stellt die dichte Region dar, also eine Region in der es viele Beispiele gibt. Diese Region kann auch durch ein Rechteck eingeschlossen werden. Das Rechteck stellt eine gröbere Repräsentation des Clusters dar. Diese Repräsentation hat jedoch den Vorteil, dass der Cluster $\mathbf{C}_1$ nun durch eine Konjunktion der zwei Intervalle $I^2_{A_1}$ und $I^2_{A_2}$ angegeben werden kann.

$$\vec{x} \in \mathbf{C}_1 \Leftrightarrow x_{A_1} \in I^2_{A_1} \wedge x_{A_2} \in I^2_{A_2}$$

Das Intervall $I^2_{A_1}$ repräsentiert den Bereich des Clusters bezüglich des Attributes A_1 und $I^2_{A_2}$ bezüglich A_2. Eine Projektion der Beispiele auf A_1 bzw. A_2 zeigt, dass alle Beispiele innerhalb des Intervalls $I^2_{A_1}$ bzw. $I^2_{A_2}$ liegen. Damit ist in beiden Intervallen die Datendichte hoch. Dies ist folgerichtig, da die Konjunktion der Clusterrepräsentation ein Gebiet beschreibt, das in den Intervallen $I^2_{A_1}$ *und* $I^2_{A_2}$ dicht ist und damit auch in den jeweiligen Intervallen für sich dicht sein muss. Damit bilden diese Intervalle zwei neue Cluster im Subspace mit A_1 bzw. A_2.

$$\vec{x} \in \mathbf{C}_2 \Leftrightarrow x_{A_1} \in I^2_{A_1} \qquad \text{bzw.} \qquad \vec{x} \in \mathbf{C}_3 \Leftrightarrow x_{A_2} \in I^2_{A_2}$$

Die drei Subspace Cluster werden dadurch beschrieben durch

$$\mathbf{C}_1 = (\{\vec{x} \in \boldsymbol{X} : x_{A_1} \in I^2_{A_1} \wedge x_{A_2} \in I^2_{A_2}\}, \quad \{A_1, A_2\}) \tag{3.1}$$

$$\mathbf{C}_2 = (\{\vec{x} \in \boldsymbol{X} : x_{A_1} \in I^2_{A_1}\}, \quad \{A_1\}) \tag{3.2}$$

$$\mathbf{C}_3 = (\{\vec{x} \in \boldsymbol{X} : x_{A_2} \in I^2_{A_2}\}, \quad \{A_2\}) \tag{3.3}$$

Ein Bottom-Up-Algorithmus berechnet zunächst die Cluster $\mathbf{C}_2$ und $\mathbf{C}_3$ für die Subspaces $\{A_1\}$ und $\{A_2\}$ indem er die dichten Intervalle $I^2_{A_1}$ und $I^2_{A_2}$ ermittelt. Im nächsten Schritt werden diese Intervalle konjugiert, um die so entstandene Region auf hohe Dichte zu überprüfen. Dies ähnelt sehr der Kandidatengenerierung beim Finden von häufigen Mengen.
Das Problem, die Subspace Cluster auf diese Art zu finden, kann sogar vollständig in das Problem des Findens von häufigen Mengen überführt werden. Ein Intervall sei hierbei ein Item mit dem Support

$$s(\{I_{A_i}\}) = \left|\{\vec{x} \in \boldsymbol{X} : x_{A_i} \in I_{A_i}\}\right|$$

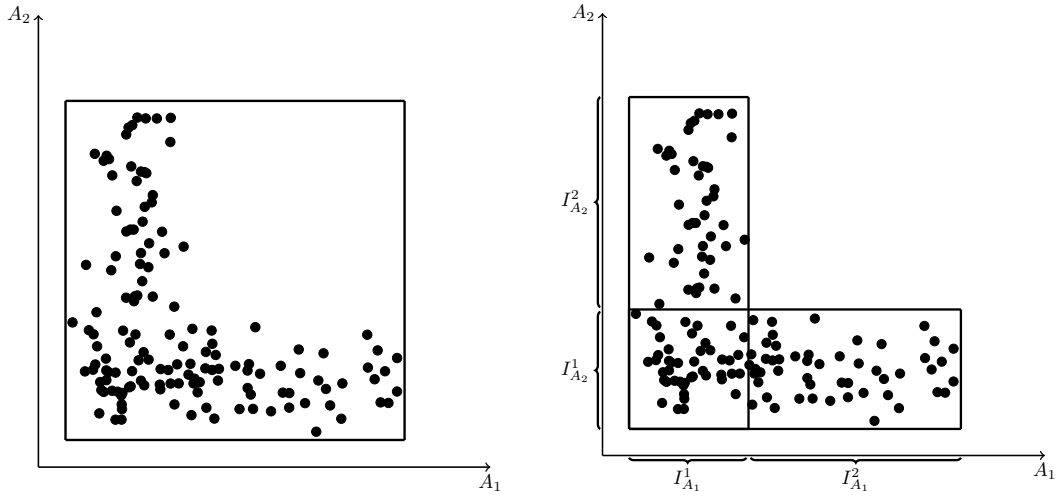
bzw. für eine Konjunktion von Intervallen $\bigwedge_{1 \le i \le k} I_{A_i}$

$$s(\{I_{A_1}, \ldots, I_{A_k}\}) = \left|\{\vec{x} \in \boldsymbol{X} : \bigwedge_{1 \le i \le k} x_{A_i} \in I_{A_i}\}\right|$$

Bei der Angabe eines Schwellwertes für die benötigte Dichte eines Intervall, kann das Finden von Subspace Clustern durch das Finden von häufigen Mengen (von Intervallen) gelöst werden (siehe auch Kapitel 3.5). Mit einem passenden Schwellwert werden so für die Daten in Abbildung 3.3a die häufigen Mengen

$$\{\{I^2_{A_1}\}, \{I^2_{A_2}\}, \{I^2_{A_1}, I^2_{A_2}\}\}$$

gefunden, aus denen die Subspace Cluster $\mathbf{C}_1, \mathbf{C}_2, \mathbf{C}_3$ (siehe Gleichungen 3.1, 3.2 und 3.3) abgeleitet werden.



(a) Die Eingrenzung durch zwei Intervalle erzeugt ein Cluster, das einen großen Bereich umfasst, in dem kein Beispiel vorhanden ist.

(b) Durch die Konjunktion von mehreren Intervallen kann ein Cluster genauer repräsentiert werden.

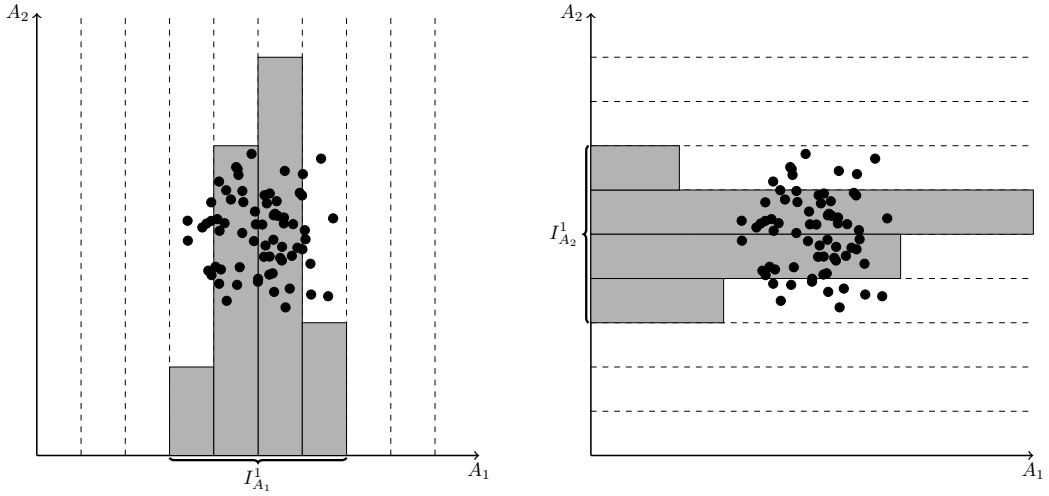
Abbildung 3.4: Für komplexe Cluster muss die Modellkomplexität erhöht werden, um den Cluster besser zu beschreiben.

Modellkomplexität Bei den obigen Überlegungen werden zwei vereinfachende Annahmen gemacht. Die erste Annahme ist, dass ein Cluster sich gut mit Hilfe eines sehr einfachen Modells darstellen lässt. Das Modell ist hierbei das Rechteck, das den Cluster einschließt. Diese Modellkomplexität ist aber zu gering, um Cluster wie in Abbildung 3.4a plausibel zu beschreiben. Um dieses Problem zu lösen, muss die Modellkomplexität erhöht werden. Das Modell kann, wie in Abbildung 3.4b zu sehen, mit drei Rechtecken beschrieben werden und umfasst den Cluster deutlich besser. Die Clusterbeschreibung muss hierfür zu

$$\vec{x} \in \mathbf{C}_1 \Leftrightarrow \left(\vec{x}_{A_1} \in I^1_{A_1} \wedge \vec{x}_{A_2} \in I^1_{A_2}\right) \vee \left(\vec{x}_{A_1} \in I^2_{A_1} \wedge \vec{x}_{A_2} \in I^1_{A_2}\right) \vee \left(\vec{x}_{A_1} \in I^1_{A_1} \wedge \vec{x}_{A_2} \in I^2_{A_2}\right)$$

erweitert werden und stellt damit eine disjunktive Normalform dar.
Durch die Verwendung weiterer Disjunktionen, kann die Modellkomplexität so weit erhöht werden, dass beliebige Regionen repräsentiert werden können. Wie in Kapitel 2.2.2 beschrieben, muss auch hier zwischen der Flexibilität und Komplexität eines Modells abgewogen werden, damit der Cluster gut abgegrenzt wird, aber kein überangepasstes Modell entsteht.

Diskretisierung der Merkmale Das zweite Problem ist, dass im oberen Beispiel die Intervalle schon vorgegeben waren. Der Cluster war schon bekannt und wurde mit den Intervallen eingegrenzt. Wenn auf den Daten gelernt wird, ist diese Information natürlich nicht vorhanden. Der Wertebereich des numerischen Merkmales muss in Einheiten



(a) Diskretisierung und Histogramm für A_1

(b) Diskretisierung und Histogramm für A_2

Abbildung 3.5: Diskretisierung von Attributen und Erstellung von Histogrammen

unterteilt werden. Dies ist die sogenannte Diskretisierung bzw. Partitionierung des Merkmalraumes.
Für diese so ermittelten Einheiten kann nun ein Histogramm, wie in Abbildung 3.5 zu sehen, erstellt werden. Das bedeutet, dass für jede Einheit, die Anzahl der darin enthaltenen Beispiele gezählt werden. Überschreitet eine Einheit einen vorgegebenen Schwellwert, gilt sie als häufig. Benachbarte Einheiten werden zu Intervallen zusammengefasst.

3.2.3 Problemdefinition

Die Eingabe für das Subspace Clustering ist ein Datensatz $\boldsymbol{X} = \{\vec{x}^{(1)}, \ldots, \vec{x}^{(N)}\}$ aus dem numerischen und beschränkten Raum $\mathbb{A}^p = A_1 \times \ldots \times A_p \subset \mathbb{R}^p$, wobei A_i das i-te Attribut ist. Für ein Beispiel $\vec{x}^{(i)} \in \boldsymbol{X}$ ist $\vec{x}^{(i)}_j$ der Wert des Beispiels für das Attribut A_j.

Definition 3.2.5 (Partitionierung in Einheiten). Sei $\mathbb{A}^p = A_1 \times \ldots \times A_p \subset \mathbb{R}^p$ ein numerischer und beschränkter Raum, dann ist $\boldsymbol{P} = \{u^{(1)}, \ldots, u^{(n)}\}$ eine Partitionierung von $\mathbb{A}^p$ in Einheiten $u^{(i)} = \{u^{(i)}_1, \ldots, u^{(i)}_p\}$ mit dem nach rechts offenen Intervall $u^{(i)}_j = [l_j, h_j)$ für ein Attribut A_j.

Aus wievielen Einheiten eine Partition besteht, hängt davon ab, wie breit die abgedeckten Intervalle auf den jeweiligen Attributen sind und in welchen Bereich $\mathbb{A}$ beschränkt ist. Da es sich jedoch um eine Partitionierung des gesamten Merkmalraumes handelt, ist jedes Beispiel in einer solchen Einheit enthalten.

Definition 3.2.6 ($\vec{x} \sqsubset u$). Sei $\vec{x}$ ein Beispiel aus einem Datensatz $\boldsymbol{X} \subseteq \mathbb{A}^p$ mit den Attributen $\boldsymbol{A} = \{A_1, \ldots, A_p\}$ und $u \in \boldsymbol{P}$ eine Einheit aus einer Partitionierung von $\mathbb{A}^p$, dann ist $\vec{x} \sqsubset u$ ($\vec{x}$ **ist enthalten in** u), falls

$$\forall A_i \in \boldsymbol{A} : l_i \leq x_{A_i} < h_i$$

Mit der binären Relation $\sqsubset$ wird nun der Support einer Einheit definiert.

Definition 3.2.7 (Support einer Einheit). Sei $\boldsymbol{X}$ ein Datensatz aus einem numerischen und beschränkten Raum $\mathbb{A}^p$ und u eine Einheit einer Partitionierung von $\mathbb{A}^p$, dann ist der Support von u definiert durch

$$s(u) = \frac{|\{\vec{x} \in \boldsymbol{X} : \vec{x} \sqsubset u\}|}{|\boldsymbol{X}|}$$

Durch den Support einer Einheit kann – äquivalent zu der Definition eines häufigen Items beim Finden von häufigen Mengen – eine dichte Einheit definiert werden.

Definition 3.2.8 (Dichte Einheit). Sei $\boldsymbol{X}$ ein Datensatz aus einem numerischen und beschränkten Raum $\mathbb{A}^p$ und u eine Einheit einer Partitionierung von $\mathbb{A}^p$, dann ist u mit einem Dichteschwellwert $\tau \in [0, 1]$ eine **dichte Einheit**, falls

$$s(u) > \tau$$

Bei den bisherigen Definitionen wird der vollständige Merkmalsraum $\mathbb{A}^p$ des Datensatzes $\boldsymbol{X}$ betrachtet. Eine Einheit lässt sich jedoch auch über ein Subspace $\mathbb{S}^k$ mit den Attributen $\boldsymbol{S}$ definieren, indem alle Beispiele in diesen Unterraum projiziert werden. Für $\mathbb{S}^k$ gelten dann die selben Definitionen wie für $\mathbb{A}^p$.

Definition 3.2.9 (Gemeinsame Oberfläche von Einheiten). Zwei k-dimensionale Einheiten $u^{(1)} = \{u^{(1)}_{t_1}, \ldots, u^{(1)}_{t_k}\}$ und $u^{(2)} = \{u^{(2)}_{t_1}, \ldots, u^{(2)}_{t_k}\}$ haben eine **gemeinsame Oberfläche** ($u^{(1)} \boxminus u^{(2)}$), falls ein Attribut A_{t_i} existiert, so dass entweder $h^{(1)}_{t_i} = l^{(2)}_{t_i}$ oder $h^{(2)}_{t_i} = l^{(1)}_{t_i}$ gilt und $u^{(1)}_{t_j} = u^{(2)}_{t_j}$ für alle anderen Attribute A_{t_j} .

Eine gemeinsame Oberfläche entspricht der intuitiven und geometrischen Interpretation von sich an einer Seite berührenden Einheiten. Ein weitergehender und transitiver Begriff ist der der zusammenhängenden Einheiten.

Definition 3.2.10 (Zusammenhängende Einheiten). Zwei k-dimensionale Einheiten $u^{(1)}$ und $u^{(2)}$ sind zusammenhängend, falls sie eine gemeinsame Oberfläche haben oder weitere k-dimensionale Einheiten $u^{(3)}, \ldots, u^{(j)}$ (mit $j \geq 3$) existieren, so dass

$$u^{(1)} \boxminus u^{(3)} \wedge \ldots \wedge u^{(j)} \boxminus u^{(2)}$$

Mit Hilfe der zusammenhängenden und dichten Einheiten kann ein Cluster nun im Sinne eines dichte-basierten Verfahrens definiert werden und damit die Definition 3.2.4 eines Subspace Clusters konkretisieren. Ein Cluster ist dann eine Menge von zusammenhängenden, dichten Einheiten.

Definition 3.2.11 (Subspace Cluster aus dichte Einheiten). Ein Cluster im numerischen und beschränkten Subspace $\mathbb{S}^k$ von $\mathbb{A}^p$ ist die maximale Menge von k-dimensionalen, zusammenhängenden und dichten Einheiten in $\mathbb{S}^k$.

Für die über dichte Einheiten definierten Subspace Cluster gilt eine Eigenschaft, die wichtig für den Bottom-Up-Ansatz des CLIQUE-Algorithmus (siehe Kapitel 3.3) ist und schon aus Kapitel 2.4.3 über die häufigen Mengen bekannt ist.

Proposition 3.2.12 (Monotonie). *Für einen Subspace Cluster aus zusammenhängenden, dichten Einheiten in $\mathbb{S}^k$ gilt, dass eine Projektion der dichten Einheiten auf jeden $k-1$-dimensionalen Unterraum von $\mathbb{S}^k$ ebenfalls dicht ist und einen Subspace Cluster bildet.*

Beweis. Für einen Subspace Cluster aus zusammenhängenden und dichten Einheiten mit den Attributen $\boldsymbol{S}$ gilt für den Support jeder dieser Einheiten u

$$s(u) \geq \tau$$

Nach der Definition 3.2.7 gilt

$$s(u) = \frac{|\{\vec{x} \in \boldsymbol{X} : \forall A_i \in \boldsymbol{S} : l_i \leq x_{A_i} < h_i\}|}{|\boldsymbol{X}|}$$

Bei einer Projektion auf einen $k-1$-dimensionalen Subspace Cluster, gilt für die neue Attributmenge $\boldsymbol{T}$, dass sie ein Attribut weniger beinhaltet, also $|\boldsymbol{T}| = |\boldsymbol{S}| - 1$. Das bedeutet für die Berechnung des neuen Supports, dass alle Beispiele $\vec{x} \in \boldsymbol{X}$ in einem Intervall weniger beinhaltet sein müssen als für $\boldsymbol{S}$. Folglich ist der Support mindestens genau so hoch.
Seien nun $A_{t_j} = \boldsymbol{S} \setminus \boldsymbol{T}$ und $u^{(1)}$ und $u^{(2)}$ zwei Einheiten mit einer gemeinsamen Oberfläche. Falls A_{t_j} das Attribut ist, für das entweder $h_{t_j}^{(1)} = l_{t_j}^{(2)}$ oder $h_{t_j}^{(2)} = l_{t_j}^{(1)}$ gilt, dann werden $u^{(1)}$ und $u^{(2)}$ auf die gleiche Einheit projiziert, da alle anderen Intervalle gleich sind. Für den anderen Fall behalten $u^{(1)}$ und $u^{(2)}$ eine gemeinsame Oberfläche, da weiterhin ein Attribut existiert für das entweder $h_{t_j}^{(1)} = l_{t_j}^{(2)}$ oder $h_{t_j}^{(2)} = l_{t_j}^{(1)}$ gilt und für alle anderen Attribute die Intervalle weiterhin gleich bleiben. Folglich bildet eine Projektion von zusammenhängenden und dichten Einheiten auf einen $k-1$-dimensionalen Subspace ebenfalls zusammenhängende und dichte Einheiten, die einen Subspace Cluster bilden. □

Für eine intuitive Repräsentation der Cluster durch eine disjunktive Normalform, wird in [4] der Begriff der Regionen verwendet. Die Idee ist, dass eine Region mit einer Konjunktion von Intervallen dargestellt werden kann. Eine Disjunktion dieser Konjunktionen kann dann, wie in Kapitel 3.2.2 beschrieben, ein Cluster beschreiben.

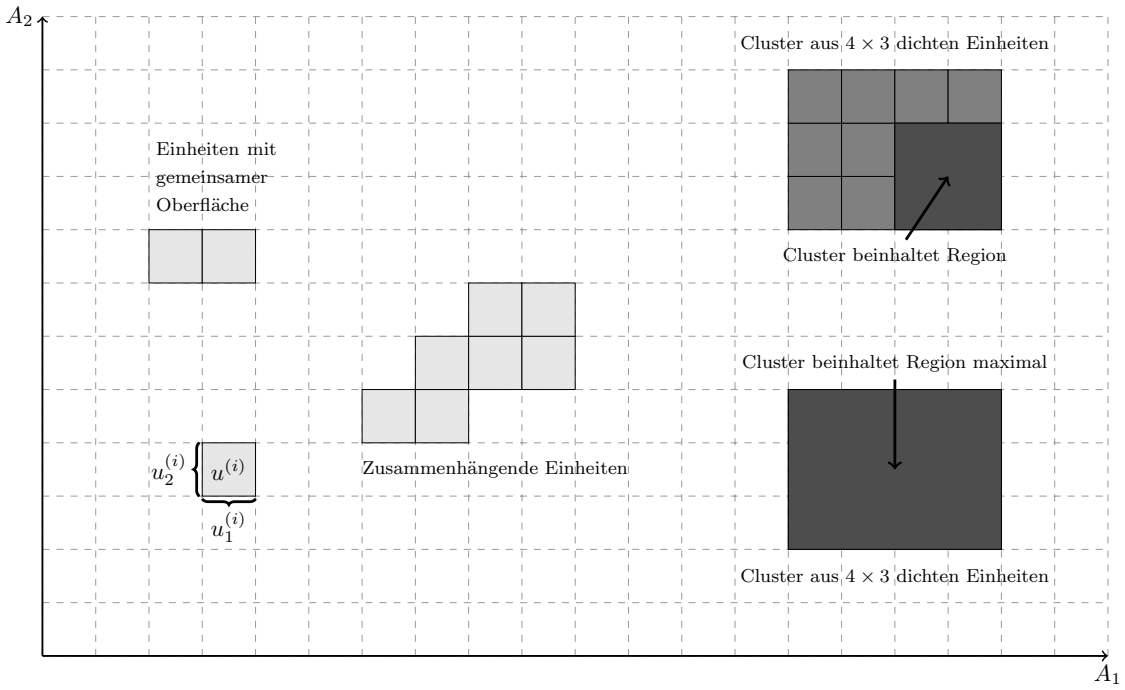


Abbildung 3.6: Definition zum Subspace Clustering mit häufigen Mengen visualisiert. Das gestrichelte Raster stellt die Partitionierung in gleich große Einheiten des Raumes $A_1 \times A_2$ dar.

Definition 3.2.13 (Region). Eine k-dimensionale Region

$$R = \bigwedge_{A_{t_i} \in \boldsymbol{S}} (L_{t_i} \leq A_{t_i} < H_{t_i})$$

mit den Intervallen $[L_{t_i}, H_{t_i})$ im k-dimensionalen Subspace mit den Attributen $\boldsymbol{S}$ ist eine achsenparallele k-dimensionale Menge von zusammenhängenden Einheiten, die einen k-dimensionalen Hyperwürfel bildet.

Eine Beschreibung eines Clusters erfolgt über solche Regionen. Dabei ist nach folgender Definition eine Region in einem Cluster enthalten.

Definition 3.2.14 ($\boldsymbol{R} \sqsubset \mathbf{C}$). Eine Region R ist in einem Cluster $\mathbf{C}$ enthalten ($R \sqsubset \mathbf{C}$) , wenn $R \cap \mathbf{C} = R$. Sie ist maximal enthalten, wenn keine echte Obermenge von R ebenfalls $\mathbf{C}$ enthält.

Für die vollständige Repräsentation eines Clusters durch Regionen wird die minimale Beschreibung verwendet.

Definition 3.2.15 (Minimale Beschreibung). Die minimale Beschreibung eines Clusters **C** ist eine nicht redundante Disjunktion von Regionen

$$\boldsymbol{R} = R^{(1)} \vee \ldots \vee R^{(d)} = \bigvee_{1 \leq j \leq d} \left(\bigwedge_{A_{t_i} \in A_t} \left(L_{t_i}^{(j)} \leq A_{t_i} < H_{t_i}^{(j)} \right) \right)$$

die **C** maximal enthalten.

Problem 3 (Subspace Clustering mit häufigen Mengen). Sei $\boldsymbol{X}$ ein Datensatz aus einem numerischen und beschränkten Raum $\mathbb{A}^p$, $\boldsymbol{P}$ eine Partitionierung von $\mathbb{A}^p$ und $\tau \in [0,1]$ ein Dichteschwellwert, dann finde für alle Subspaces $\mathbb{S}^i$ mit $1 \leq i \leq p$ von $\mathbb{A}^p$ die minimale Beschreibung aller Cluster.

3.3 Algorithmus

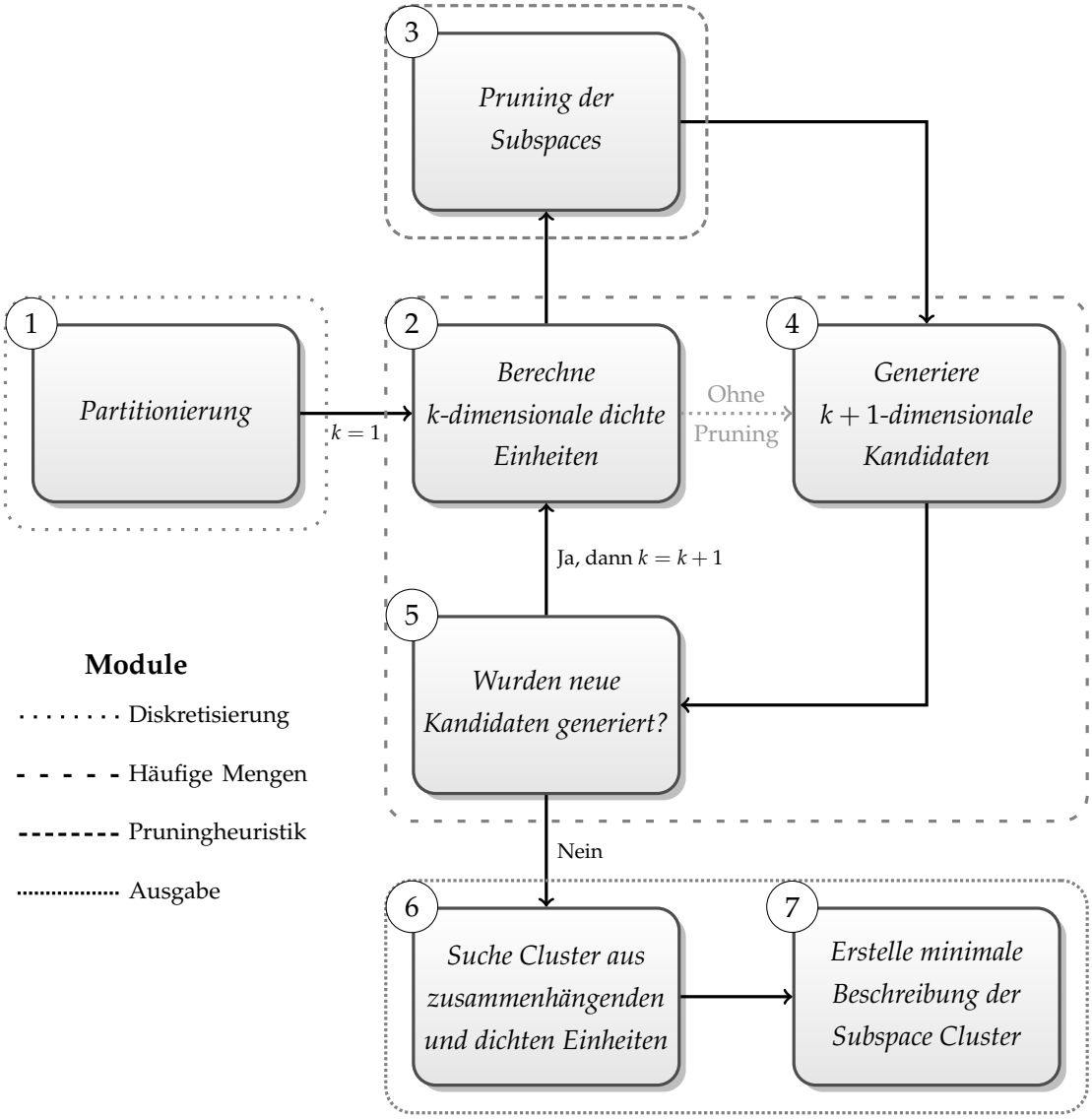
Eine der ersten und wegweisenden Arbeiten im Bereich des Subspace Clusterings ist [4] und beschreibt den CLIQUE-Algorithmus. Dabei handelt es sich um ein Bottom-Up-Verfahren, das den Raum in gleich große Einheiten unterteilt und dann mit einem APRIORI-ähnlichen Verfahren die Dichte in unterschiedlichen Subspaces bestimmt, um so Subspace Cluster zu ermitteln und eine minimale Beschreibung dieser Cluster auszugeben. Es folgten einige Arbeiten, die CLIQUE an einigen Stellen modifizieren. Die zwei bekanntesten Arbeiten sind [25, 59], welche sich zwei unterschiedlichen Aspekten von CLIQUE widmen. Es lässt sich bei allen Arbeiten jedoch ein algorithmisches Grundgerüst erkennen, das in diesem Kapitel abstrahiert werden soll, um einen Rahmen zu schaffen, der es ermöglicht, an den wichtigen Stellen neue Methoden zu entwickeln und zu evaluieren. Nach [4] lässt sich CLIQUE zunächst in drei Schritte unterteilen:

I. Finde Subspaces mit Clustern Dieser Schritt bildet die Brücke zwischen häufigen Mengen und Subspace Clustern. Der Merkmalsraum wird in Einheiten unterteilt, um dann die dichten Einheiten zu ermitteln. Beim Reduzieren des Suchraumes werden weitere Heuristiken eingesetzt, die nur Subspaces verwenden, die in irgendeiner Form „interessant“ sind.

II. Finde Cluster innerhalb des Subspaces Bei den gefundenen dichten Einheiten werden alle zusammenhängenden Einheiten gesucht, die dann die Cluster bilden. Dies ist äquivalent mit dem Suchen der Zusammenhangskomponenten eines Graphen in der Graphentheorie.

III. Generiere eine minimale Beschreibung der Cluster Um eine gut verständliche Repräsentation der gefundenen Cluster zu erhalten, wird in diesem Schritt aus den gefundenen zusammenhängenden, dichten Einheiten die kleinste Anzahl an Regionen gesucht, die als minimale Beschreibung ausgegeben wird.

Algorithmus 3.1 CLIQUE



Um einzelne Schritte zu isolieren und dann zu optimieren, wird der Algorithmus in dieser Arbeit in vier Module bzw. sieben Schritte unterteilt. Der Algorithmus 3.1 zeigt eine schematische Darstellung von CLIQUE. Anhand dieser Abbildung wird der Algorithmus in diesem Kapitel erläutert und in den darauf folgenden Kapiteln innerhalb der einzelnen Module näher betrachtet.

1. Partitionierung Der erste Schritt des Algorithmus ist die Partitionierung des Raumes in Einheiten. Beim maschinellen Lernen wird dieses Vorgehen auch Diskretisierung von numerischen Merkmalen genannt. Bei CLIQUE wird jedes Merkmal in gleich große Intervalle bzw. Einheiten unterteilt. Diese Intervalle bilden in den weiteren Schritten die Items, die gezählt werden. Die Wahl der Anzahl dieser Intervalle beeinflusst deswegen stark die Performanz des gesamten Algorithmus. Aber auch die Qualität des Ergebnises ist von der Wahl der Einheiten abhängig. Folglich besitzt dieser Schritt viel Potential zur Analyse und Optimierung und bildet somit ein eigenständiges Modul, das in Kapitel 3.4 näher behandelt wird.

2. Berechne k-dimensionale dichte Einheiten Nachdem die Partitionierung für alle Merkmale abgeschlossen ist, werden die dichten Einheiten berechnet. Dies ist gleichzusetzen mit dem Finden von häufigen Mengen. In der ersten Iteration werden die eindimensionalen dichten Einheiten bzw. einelementigen Mengen gesucht. Hierfür wird mit einem Datenbankdurchlauf gezählt, wie viele Beispiele innerhalb jeder eindimensionalen Einheit enthalten sind. Dies ist der Support dieser Einheit. Wie in Abbildung 3.5 zu sehen, kann dies mit einem Histogramm visualisiert werden. Alle Einheiten, deren Support über einen als Parameter gegebenen Schwellwert τ liegen, sind dicht.

3. Pruning der Subspaces Gibt es für ein Attribut keine dichten Einheiten, dann kann innerhalb dieses eindimensionalen Subspaces, der durch dieses Attribut induziert wird, kein Cluster gefunden werden. Nach [4] werden aber auch Subspaces verworfen, die dichte Einheiten enthalten. Die Idee dahinter ist, nur „interessante“ Subspaces zu behalten und durch das Verwerfen der restlichen Unterräume und somit auch den dichten Einheiten, den Algorithmus zu beschleunigen. Dieses Vorgehen wird **Pruning** genannt.
In der Tat beschleunigt jedes Verwerfen einer dichten Einheit das Verfahren maßgeblich. Wie in den Grundlagen zu den häufigen Mengen (siehe Kapitel 2.4.3 zum Algorithmus APRIORI bzw. Abbildung 2.11) erwähnt, halbiert das Verwerfen einer einelementigen Menge den gesamten Suchraum. Dieses Prinzip bildet die Basis von APRIORI und soll durch das Pruning von Subspaces weiter verbessert werden. CLIQUE verwendet hier ein Pruning nach dem *Minimal Description Length* Prinzip, während Algorithmen wie ENCLUS [25] ein Entropie-basiertes Verfahren verwenden. Es gibt aber auch Arbeiten, wie

beispielsweise MAFIA [59], die auf dieses Pruning verzichten, da sie befürchten dadurch wichtige Cluster nicht mehr zu finden.
In Kapitel 3.7 wird dieses Modul weiter erläutert.

4. Generiere $k+1$-dimensionale Kandidaten Aus den verbleibenden dichten Einheiten werden die Einheiten generiert, die nach der Monotonie-Eigenschaft ebenfalls dicht sein können. Alle anderen $k+1$-dimensionalen Einheiten können nicht dicht sein, da ihre k-dimensionale Projektion nicht dicht ist. Dies ist eine direkte Folgerung aus Proposition 3.2.12.
Seien beispielsweise $\boldsymbol{D}_k = \{u^{(1)}, \ldots, u^{(l)}\}$ eine Menge von k-dimensionalen, dichten Einheiten mit $u^{(i)} = \{u_1^{(i)}, \ldots, u_k^{(i)}\}$, dann werden die Kandidaten aus denjenigen dichten Einheiten gebildet, die $k-1$ gleiche Dimensionen haben.
Dies ist fast identisch mit dem Vorgehen beim Finden von häufigen Mengen. Der Unterschied besteht darin, dass ein Kandidat nicht aus dichten Einheiten, die auf den selben Attributen definiert wurden, gebildet wird. Würde man beispielsweise APRIORI direkt adaptieren, so dass die Items die Einheiten wären, dann würden auch Kandidaten überprüft werden, die durch $u^{(1)} = \{u_1^{(1)}\}$ (Intervall für A_1) und $u^{(2)} = \{u_1^{(2)}\}$ (dies ist ebenfalls ein Intervall für A_1) generiert werden. Dies wäre also eine Einheit $u^{(3)} = \{u_1^{(1)}, u_1^{(2)}\}$. Diese kann jedoch nicht häufig sein. Der Support ist sogar immer gleich 0, da kein Beispiel für das selbe Attribut in zwei sich nicht überlappenden Intervallen sein kann.

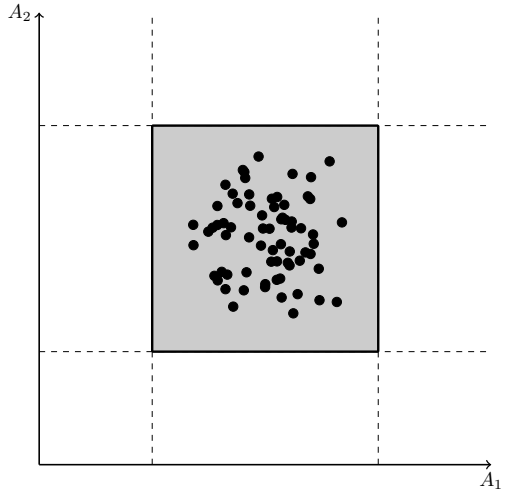
5. Wurden neue Kandidaten generiert? In diesem Schritt wird überprüft, ob Kandidaten für die nächste Iteration generiert wurden. Ist dies der Fall, dann wird k inkrementiert und für die Kandidaten wird mit einem erneuten Datenbankdurchlauf in Schritt 2 ihr Support berechnet, um die höherdimensionalen dichten Einheiten zu erhalten. Dies wird solange wiederholt bis keine neuen Kandidaten generiert wurden. In diesem Fall wird in den Schritten 6 und 7 die Ausgabe generiert.
Die Schritte 2, 4 und 5 bilden den Teil, der dem Finden von häufigen Mengen und damit einem eigenen Modul entspricht. Wie bekannte Algorithmen, wie beispielsweise FPGROWTH, angewandt werden können, um dieses Problem effizient zu lösen, wird in Kapitel 3.5 beschrieben.

6. Suche Cluster aus zusammenhängenden und dichten Einheiten Wenn keine neuen Kandidaten generiert wurden, muss nun die Ausgabe generiert werden. Das bisherige Zwischenergebnis ist eine Menge von Subspaces und eine Menge von dichten Einheiten für jeden dieser Unterräume. In diesem Schritt ist die Aufgabe, für jeden dieser Unterräume die dazugehörigen dichten Einheiten so in Teilmengen aufzuteilen, dass jede Menge von dichten Einheiten zusammenhängend ist und damit einen Cluster bildet.

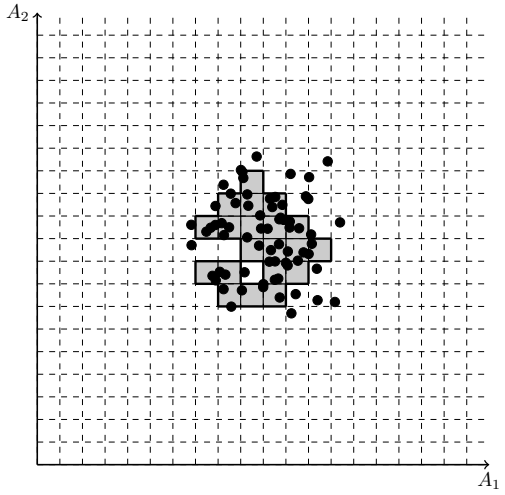
7. Erstelle die minimale Beschreibung der Subspace Cluster Für jede Menge von zusammenhängenden und dichten Einheiten soll nun die intuitive Repräsentation des Cluster durch eine Menge von Regionen erstellt werden. Damit kann jeder Cluster in disjunktiver Normalform dargestellt werden (siehe Kapitel 3.2.3).
Die Schritte 6 und 7 können als Modul zur Erstellung der Ausgabe zusammengefasst werden. Von der Komplexität ist dieses Problem nicht einfach und wird von [4] approximiert gelöst. Kapitel 3.8 beschreibt das nähere Vorgehen.

3.4 Partitionierung

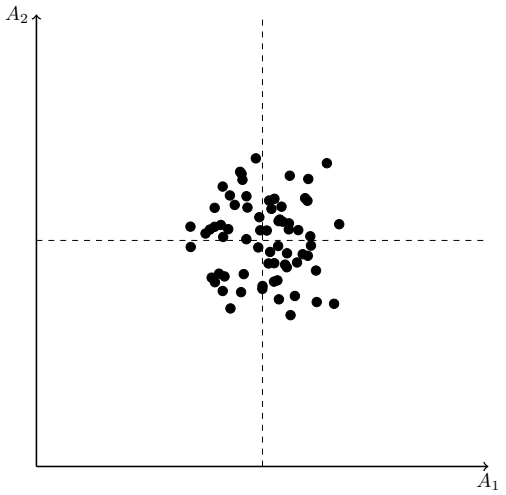
Die Partitionierung des Raumes in die Einheiten stellt den ersten Schritt im Algorithmus dar. Zugleich beeinflusst das Ergebnis dieses Moduls sowohl erheblich die Performanz des Algorithmus, als auch die Qualität des Endergebnisses.
Da jedes Intervall als ein Item angesehen werden kann, hat die Partitionierung eine direkte Auswirkung auf die Anzahl der verschiedenen Items und damit auch auf den gesamten Suchraum. Da der Suchraum exponentiell zur Menge der Items ist, hat die Reduzierung der Intervalle einen großen Einfluss auf diese Größe und damit auch auf die Laufzeit des Algorithmus.
Abbildung 3.7 verdeutlicht diese Unterschiede. Während die Einheiten in 3.7a sehr grob, aber auf den vorhandenen Cluster zugeschnitten sind, ist in Abbildung 3.7b eine sehr feine Unterteilung gewählt. Bei der groben Einteilung wurde der vorhandene Cluster nicht sehr genau, jedoch vollständig von einer dichten Einheit erfasst. Dafür gibt es nur einen Suchraum von 2^6 Mengen, der selbst mit dem naiven Vorgehen (siehe Kapitel 2.4.2) schnell untersucht werden kann. Der Suchraum der feinen Einteilung ist deutlich aufwändiger zu bearbeiten. Mit einem zu hohen relativen Schwellwert werden nicht einmal alle Beispiele von dichten Einheiten abgedeckt. Im Cluster sind sogar Löcher von nicht dichten Einheiten enthalten. Das Modell scheint hier zu komplex zu sein, um eine qualitativ hochwertige Repräsentation des Clusters zu liefern.
Im schlimmsten Fall kann durch eine ungünstige Partitionierung auch gar kein Cluster gefunden werden. In Abbildung 3.7c wurde der Cluster durch die Diskretisierung in 4 gleich große Einheiten aufgeteilt, wodurch der relative Schwellwert für jede Einheit zu groß wird und keine dichte Einheit gefunden wird, obwohl ein Cluster vorhanden ist. Die Beispiele in Abbildung 3.7 sind zwar konstruiert, unterstreichen aber die Bedeutung dieses Moduls.
Beim maschinellen Lernen wird die Partitionierung des Wertbereiches Diskretisierung genannt. In [51, 45] werden hierzu verschiedene Techniken vorgestellt. Dabei wird eine Kategorisierung in überwachte und unüberwachte Methoden vorgenommen. Wie bei den in Kapitel 2.2 vorgestellten Lernverfahren, wird auch hier unterschieden, ob bei der Diskretisierung ein Label verwendet wird (überwacht) oder nicht (unüberwacht). In der



(a) Grobe Partitionierung auf jeweils 3 Intervalle pro Attribut. Der zu durchsuchende Suchraum beim Finden der häufigen Mengen ist 2^6. Der Cluster ist nur grob umrissen.



(b) Feine Partitionierung auf jeweils 20 Intervalle pro Attribut. Der zu durchsuchende Suchraum beim Finden der häufigen Mengen ist 2^{40}. Der Cluster ist sehr fein umrissen und besitzt sogar ein Loch.



(c) Grobe Partitionierung bei dem keine dichte Einheit gefunden wird, da der Cluster genau so geteilt wird, dass der Schwellwert zu hoch ist.

Abbildung 3.7: Unterschiedliche Partitionierung erzielt unterschiedliche Ergebnisse in der Laufzeit und in der Qualität der Ergebnisse

Literatur wird überwiegend an den überwachten Methoden geforscht. Dies ist darin begründet, dass Diskretisierung hauptsächlich bei Klassifikationsverfahren verwendet wird und dort die Verwendung des Labels zu besseren Ergebnissen führt (siehe [30]). Clustering ist jedoch unüberwachtes Lernen, bei dem keine Klasseninformation verfügbar ist. Dies schließt viele bekannte Verfahren wie z.B. die Entropie-Diskretisierung aus. Im Folgenden werden mögliche Methoden zur unüberwachten Diskretisierung vorgestellt.

3.4.1 Unüberwachte Diskretisierung

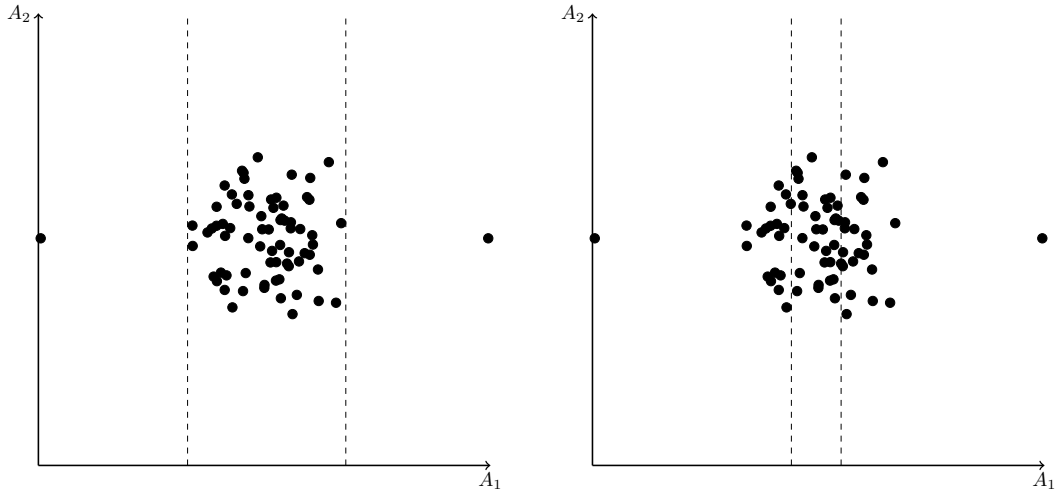
Eine unüberwachte Diskretisierung wird oft bei der Erstellung von Histogrammen verwendet, um Häufigkeitsverteilungen darzustellen. Ohne jegliche Klasseninformation wird der Wertebereich in Intervalle unterteilt und für jeden Bereich die Häufigkeit der Beispiele in diesem Bereich zugewiesen, um in Diagrammen wie Abbildung 3.5 zu erstellen.
Neben der rein graphischen Darstellung werden Histogramme auch bei der Datenbankoptimierung verwendet, um Anfragen zu beschleunigen (siehe [44]). In diesem Bereich entstehen oftmals neue Methoden. Die bekanntesten werden im Folgenden vorgestellt.

3.4.1.1 EqualWidth

Die einfachste Methode zur Diskretisierung ist das Einteilen des Raumes in gleich lange Intervalle. Als Parameter wird dabei die Anzahl der Intervalle b benötigt. Für ein nicht beschränktes und numerisches Attribut A wird dabei der kleinste und der größte Wert ermittelt (L_A und H_A), so dass für die Intervall-Länge

$$I_w = \frac{H_A - L_A}{b}$$

gilt. Für ein beschränktes Attribut kann für L_A und H_A auch die untere bzw. obere Schranke gewählt werden.
Dieses simple Prinzip hat zwei wichtige Nachteile (siehe [36, 30, 51, 45]). Zum einen ist die Wahl des Parameters b, der die Anzahl der Intervalle vorgibt, nicht einfach und intuitiv zu treffen. Abbildungen 3.7b ($b = 20$) und 3.7c ($b = 2$) zeigen, wie stark diese Wahl die Ergebnisse beeinflusst. Ohne Kenntnisse der Daten ist es schwer ein passendes b zu finden und selbst mit dieser Kenntnis ist die Antwort auf die Frage, was ein gutes b ist, nicht klar zu beantworten.
Der zweite Nachteil ist, dass diese Methode anfällig gegenüber Ausreißern ist. Dadurch, dass die kleinsten und größten Werte verwendet werden, ist ein einzelner Ausreißer, der extreme Werte annimmt, ausreichend, um die Größe der Intervalle und damit auch das Ergebnis der anschließenden Lernaufgabe stark zu beeinflussen. Dies macht die Wahl eines passenden Parameters b noch schwieriger.



(a) Die beiden Ausreißer führen hier zu einer guten Diskretisierung, falls die kleinsten und größten Werte eines Datensatzes zur Bestimmung der Intervall-Länge bei EQUALWIDTH verwendet werden. Typischerweise verursachen Ausreißer jedoch schlechte Partitionierungen des Raumes.

(b) Durch die Aufteilung nach der Häufigkeit (EQUALFREQ) wird der Cluster in drei Intervalle unterteilt. Dies kann bei zu niedrigem Schwellwert dazu führen, dass keine dichte Einheit gefunden wird.

Abbildung 3.8: Für einen Datensatz mit zwei Ausreißern wird mit EQUALWIDTH und EQUALFREQ diskrektisiert. Bei beiden Verfahren ist $b = 3$.

In [4] wird für CLIQUE diese Methode verwendet. Dies führt dazu, dass diese beiden Nachteile auch für das Subspace Clustering relevant sind. Die Laufzeit der EQUALWIDTH Diskretisierung selbst ist jedoch recht gering. Für jedes Attribut muss über alle Beispiele iteriert werden, um den kleinsten bzw. größten Wert zu finden, so dass für die Laufzeit $\mathcal{O}(|A||\boldsymbol{X}|)$ angegeben werden kann.

3.4.1.2 EQUALFREQ

Eine weitere simple Methode, die die Daten bei der Diskretisierung miteinbezieht und zumindest das Problem der Ausreißer bei der EQUALWIDTH Diskretisierung behebt, ist die Einteilung nach der Häufigkeit. Der Parameter zur Anzahl der Intervalle bleibt, doch werden nun die Beispiele für jedes Attribut sortiert, um daraufhin die Grenzen der Intervalle so festzulegen, dass in jedem Intervall gleich viele Beispiele enthalten sind. Einige wenige extreme Beispiele fallen somit nicht mehr ganz so stark ins Gewicht. Das jeweilige Intervall kann dadurch nur sehr groß werden. Für die Laufzeit bedeutet das einen zusätzlichen Aufwand zur Sortierung der Beispiele.
Zusätzlich bleibt der weiterhin schwierig zu wählende Parameter b, der in bestimmten Situationen mit dieser Methode ebenfalls zu sehr schlechten Ergebnissen führt. In Ab-

bildung 3.8 werden Daten mit Hilfe der EQUALWIDTH- und der EQUALFREQ-Methode diskretisiert. In beiden Fällen ist $b = 3$. Während dies in 3.8a trotz bzw. gerade wegen der zwei Ausreißern zu einem guten Ergebnis führt, da der gesamte Cluster direkt durch ein dichtes Intervall hinreichend beschrieben werden kann, führt dies bei der Aufteilung nach der Häufigkeit in Abbildung 3.8b zu einem Aufteilen der Datenpunkte in drei unterschiedliche Einheiten, was je nach Schwellwert dazu führt, dass keine der Einheiten mehr dicht ist.

Auch wenn gegenüber EQUALWIDTH nur noch das Problem des zu wählenden Parameters b bleibt, ist diese Methode dennoch ungeeignet um nach häufigen Mengen zu suchen. Der Grund ist die Konstruktion der Intervalle an sich, die versucht Intervalle mit gleicher Häufigkeit zu finden. Für alle Intervalle gilt somit der selbe Support und damit sind alle Intervalle entweder häufig oder nicht.

3.4.1.3 V-OPTIMAL

Die Schwächen der EQUALWIDTH- und EQUALFREQ-Methoden wirkten sich auch bei der Datenbankoptimierung aus, was zur Entwicklung von weiteren Methoden führte. Dabei sollten die Daten stärker einbezogen werden. Das V-OPTIMAL Histogramm [39] ist dafür ein Beispiel.

Die grundlegende Idee ist, dass für ein Intervall eine Varianz, also ein Maß für die Streuung der Beispiele innerhalb des Intervalls, berechnet werden kann. Wenn die Intervalle so gewählt werden, dass die Summe aller Varianzen minimiert wird, dann werden damit Sprünge in der Verteilung der Daten innerhalb des Intervalls minimiert, was somit zu homogeneren Intervallen führt. Es wird folglich die Funktion

$$V_A = \sum_{i=1}^{b} w_{I_A^i} V_{I_A^i}$$

für jedes Attribut $A \in \boldsymbol{A}$, das in b Intervalle eingeteilt wird, minimiert. Dabei ist $V_{I_A^i}$ die Varianz des Intervalls I_A^i und $w_{I_A^i}$ die Anzahl der Beispiele innerhalb des Intervalls, um der Varianz ein Gewicht zuzuweisen.

In Abbildung 3.9 sind zwei Cluster dargestellt. Das Attribut A_1 soll nun in zwei Intervalle $I_{A_1}^1$ und $I_{A_1}^2$ diskretisiert werden. Es muss also der Schnittpunkt c zwischen diesen beiden Intervallen gefunden werden. Für alle möglichen Schnittpunkte wurde die Funktion V_{A_1} abgebildet. Es ist zu erkennen, dass die Varianz zwischen den beiden Clustern minimal ist, so dass V-OPTIMAL den Schnittpunkt in diesem Bereich auswählen wird.

Ein Schnittpunkt, der genau in der Mitte des minimalen Plateaus liegt, würde zu Intervallen führen, die zwar den Cluster umfassen, jedoch auch größere leere Bereiche. Optimal für das Finden von dichten Einheiten wären also zwei Schnittpunkte. Zum einen der kleinste minimale Wert und zum anderen der größte, so dass der leere Bereich zwi-

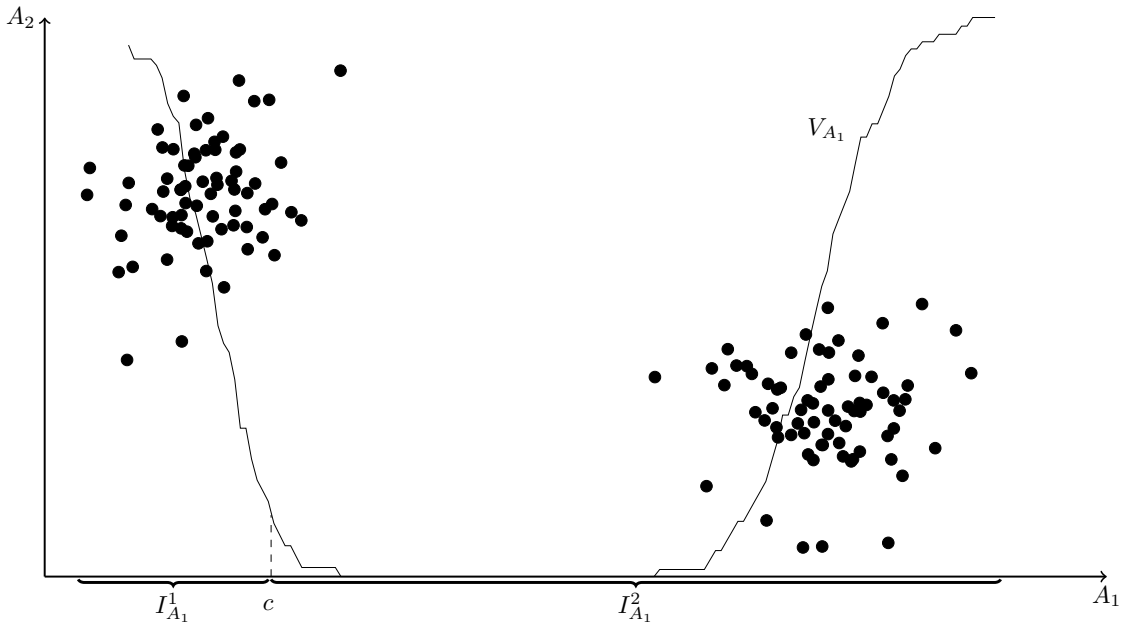


Abbildung 3.9: Für das Attribut A_1 werden zwei Intervalle $I^1_{A_1}$ und $I^2_{A_1}$ mit der variablen Grenze c gebildet. Abhängig von c wird für die beiden Intervalle die gewichtete Summe der beiden Varianzen V_{A_1} berechnet.

schen den beiden Clustern als einzelnes Intervall ausgegeben wird und die beiden Cluster durch engere Intervalle beschrieben werden.

Für eine gute Diskretisierung bezüglich dichter Einheiten muss die Anzahl der Schnittpunkte festgelegt und ein Optimierungsalgorithmus gestartet werden. Bei der Datenbankoptimierung wird dies nur für wenige Attribute durchgeführt und die Anzahl der Schnittpunkte ist fest. Dabei werden zwar auch nur approximative Algorithmen verwendet, doch im Kontext des Subspace Clusterings und der hohen Dimensionalität ist die Komplexität dieses Verfahrens sehr hoch.

3.4.1.4 MaxDiff

Die gewichtete Summe der Varianz ist dann klein, wenn der Schnittpunkt zwischen zwei Gruppen von Beispielen liegt. Eine weitere Möglichkeit, in diesem Bereich Schnittpunkte zu finden, wird ebenfalls in der Datenbankoptimierung verwendet und heißt MaxDiff[62]. Dabei werden die Schnittpunkte an den Stellen gesetzt, an denen die Differenz der Beispiele am größten ist. Dies wäre in Abbildung 3.9 an den Enden des Plateaus. Im Vergleich zu V-Optimal ist die Berechnung nicht so aufwändig. Alle Beispiele müssen bezüglich des zu diskretisierenden Attributes sortiert werden. Ein weiterer Lauf über die Beispiele findet dann die größten Abstände zwischen zwei Punkten. Anschließend

werden durch die Angabe von b oder eines Schwellwertes für die Mindestdifferenz die Intervalle definiert.

3.4.1.5 Clustering

Sowohl MAXDIFF, als auch V-OPTIMAL, finden bei klar abgegrenzten Gruppen gute Intervalle. Im Prinzip werden durch diese Intervalle Cluster beschrieben und ausgegeben. In der Tat werden auch Cluster-Algorithmen wie KMEANS oder DBSCAN (siehe Kapitel 2.3) auf ein zu unterteilendes Attribut angewandt. Die Grenzen der gefundenen Cluster werden dann als Schnittpunkte zur Erstellung der Intervalle verwendet.

3.4.1.6 Nachteile der klassischen Histogramm-Diskretisierung

Die Nachteile von EQUALWIDTH führen zu der Frage, ob andere Methoden zu besseren Intervallen führen. Wie bereits erwähnt ist EQUALFREQ durch die gleich verteilten Häufigkeiten gänzlich ungeeignet und Verfahren wie V-OPTIMAL sehr aufwändig. MAXDIFF und der Clustering-Ansatz klingen zunächst nach einer performanteren Alternative mit ähnlichen Ergebnissen.
Es gibt jedoch einen entscheidenden Nachteil, der für V-OPTIMAL, MAXDIFF und das Clustering gilt. Bei klar zu gruppierenden Beispielen finden alle Verfahren Grenzen von Intervallen, die eine gute Einteilung zum Finden von dichten Einheiten ermöglichen. Bei Subspace-Clustering werden sich in den hochdimensionalen Daten viele Cluster überlappen. Keine der vorgestellten unüberwachten Techniken geht auf diesen Sachverhalt ein.
Dies lässt sich gut am Beispiel in Abbildung 3.10 beobachten. Die optimale Diskretisierung ist mit den gestrichelten Linien gekennzeichnet und beinhaltet 4 Intervalle. Dadurch können bei einer passenden Diskretisierung von A_2 alle Cluster genau beschrieben werden.
Bei einer Einteilung in gleich lange Intervalle bei EQUALWIDTH sind 4 Intervalle nicht ausreichend. Erst bei einer größeren Anzahl ist die „Auflösung“ hoch genug, um die Grenzen zwischen den Clustern zu finden. Dies führt aber zu einer hohen Anzahl von zu zählenden Einheiten. Die von EQUALFREQ erzeugten Intervalle sind auch nicht zufriedenstellend und können aufgrund ihrer identischen Häufigkeit nicht verwendet werden.
Bei MAXDIFF werden die Intervalle rein zufällig gewählt, da es keine nennenswerten Unterschiede in den benachbarten Beispielen gibt. Ähnlich würde sich auch ein Clustering-Verfahren verhalten, da die Daten projiziert auf A_1 als ein Cluster angesehen werden. Ein ähnliches Problem hat auch V-OPTIMAL.
Sich überlappende Cluster sind jedoch in komplexen und hochdimensionalen Daten zu erwarten, so dass die vorgestellten und geläufigen unüberwachten Diskretisierungstechniken als nicht gut geeignet angesehen werden müssen.

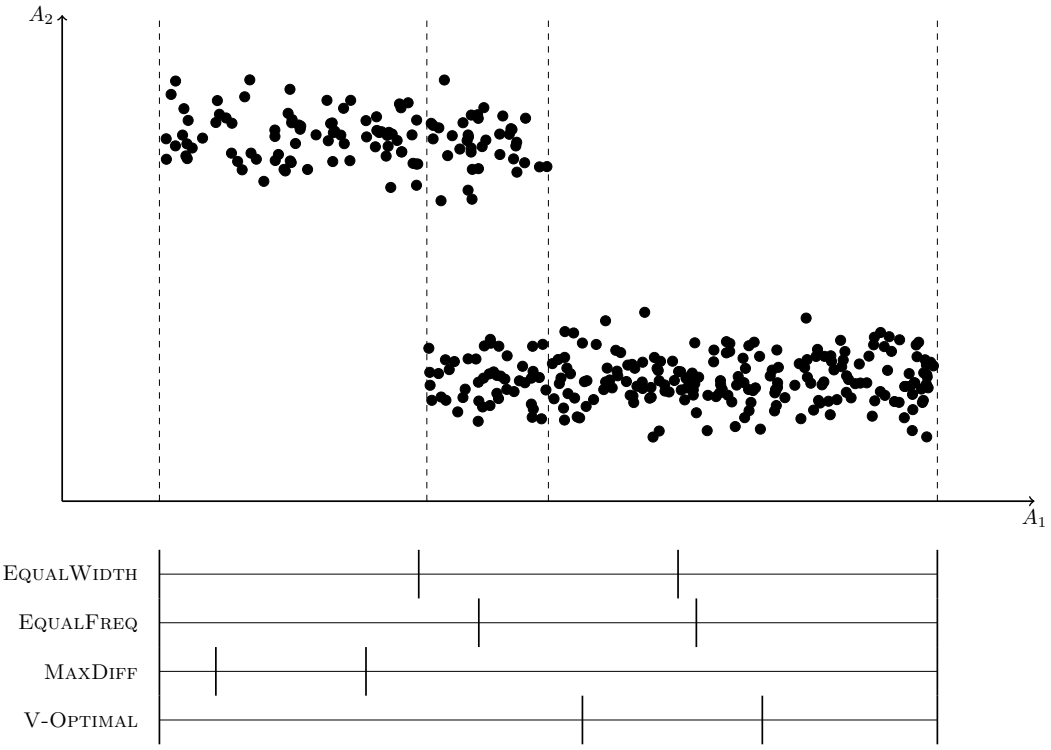


Abbildung 3.10: EQUALWIDTH, EQUALFREQ, V-OPTIMAL bei sich überlappenden Clustern.

3.4.2 Partitionierung über die Häufigkeitsverteilung

In Kapitel 3.4.1.6 zeigten sich die Schwächen der klassischen Methoden bei der Diskretisierung für das Subspace Clustering. Für eine verbesserte Partitionierung sollte die spätere Verwendung der erstellten Einheiten berücksichtigt werden. In Abbildung 3.10 ist eine gewünschte Einteilung zu sehen. Die Vorteile hierbei sind, dass durch jeden gebildeten Schnittpunkt die Grenzen eines höherdimensionalen Clusters markiert werden. Die Projektion auf das Attribut A_1 „vermischt“ diese Grenzen jedoch mit den Grenzen eines zweiten Clusters.
Es wird also eine Methode benötigt, die den Bereich, in dem sich die beiden Cluster bezüglich des Attributes A_1 überschneiden, erkennt. Bei Betrachtung der Häufigkeitsverteilung der Daten in Abbildung 3.11 ist auffällig, dass sich die Verteilung an den gewünschten Schnittpunkten verändert, da sich bei der Überlappung die Anzahl der Beispiele aus zwei Clustern addieren.

3.4.2.1 MAFIA

In [59] wurde mit MAFIA ein Algorithmus vorgestellt, der abhängig von der Häufigkeitsverteilung eine Partitionierung erstellt. Um die Häufigkeitsverteilung zu approximieren wird ein Histogramm mit gleich langen Einheiten erstellt. Dabei sollen die Intervalle

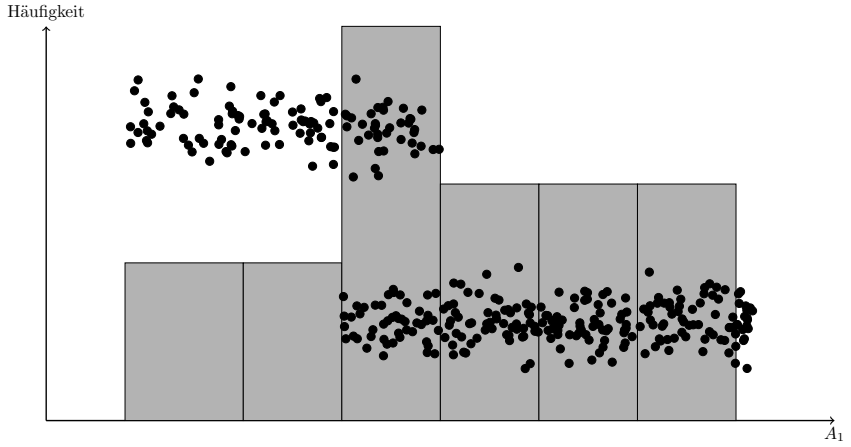


Abbildung 3.11: Häufigkeitsverteilung von überlappenden Clustern.

möglichst klein sein. Für alle so erstellten eindimensionalen Einheiten werden die benachbarten Einheiten zu einer Einheit zusammengefasst, wenn die relative Abweichung ihrer Häufigkeit unter einem Schwellwert β liegt. Bleibt am Ende nur noch eine Einheit für das Attribut, wird das Attribut ähnlich zu CLIQUE mit der EQUALWIDTH-Methode aufgeteilt (siehe Algorithmus 3.2).

Experimentell wurde so in [59] eine Beschleunigung um den Faktor 44 gegenüber CLIQUE erreicht. Durch die Verschmelzung kann also eine deutliche Verringerung der Einheiten erreicht werden. Gleichzeitig werden durch die Beachtung von Sprüngen in der Häufigkeitsverteilung die wichtigen Schnittpunkte gefunden, die zur Erstellung von qualitativ hochwertigen Clustern benötigt werden. Die während der Erstellung der Histogramme erhaltene Information zur Häufigkeit kann nicht dichte Einheiten schon bei der Erstellung der Transaktionsdatenbank ausschließen.

Ein weiterhin bestehendes Problem ist die Bestimmung der anfänglichen Partitionierung, die weiterhin die Qualität und Laufzeit beeinflusst. In Abbildung 3.11 wurde die Partitionierung demonstrativ an die Daten angepasst, um die Unterschiede in der Häufigkeitsverteilung hervorzuheben. Bei einer zu groben Einteilung ist die Wahrscheinlichkeit, dass die erzeugten Intervallgrenzen nicht in der Nähe der gewünschten sind, recht hoch. Ein Extremfall für das obige Beispiel wäre eine Anfangspartitionierung in zwei Intervalle, so dass das Verfahren gar nicht in der Lage ist, drei Intervalle zu bilden.

Eine zu feine Unterteilung führt aber zu einer zu sensitiven Verteilung. In Abbildung 3.12 ist ein feinere Unterteilung als in Abbildung 3.11 dargestellt. Es ist zu erkennen, dass es durch die kleinen Intervalle in manchen Bereichen starke Abweichungen in der Häufigkeitsdichte gibt, die zu unnötigen Einteilungen in Einheiten führen. Es entstehen innerhalb der Cluster sogar Lücken, so dass undichte Einheiten entstehen können, die

Algorithmus 3.2 Partitionierung bei MAFIA

Eingabe: Datensatz $\boldsymbol{X}$ mit den Attributen $\boldsymbol{A}$, Schwellwert β, Intervalllänge x_d und x_b
Ausgabe: $\forall A_i \in \boldsymbol{A}$ eine Partitionierung $\boldsymbol{P}_i$ für A_i

```
for A_i ∈ A do
    P_i = {u^(1), ..., u^(d)} mit ∀u^(k), u^(k+1) : r^(k) = l^(k+1) und ∀u^(k) : r^(k) − l^(k) = x_d
    Berechne Support s(u^(k)) für jede Einheit u^(k) ∈ P_i
    for i = 1 → (d − 1) do
        c_1 = s(u^(i))
        c_2 = s(u^(i+1))
        if |c_1 − c_2| / max(c_1, c_2) ≤ β then
            u^(i+1) = [l^(i), r^(i+1))
            P_i = P_i \ {u^(i)}
        end if
    end for
    if |P_i| == 1 then
        P_i = {u^(1), ..., u^(b)} und ∀u^(k) : r^(k) − l^(k) = x_b
    end if
end for
```

einen Cluster in zwei Teile teilen. Bei einer gröberen Partitionierung wäre dies nicht der Fall (vergleiche Abbildung 3.11 und 3.12).
Trotzdem führt eine zu grobe Einteilung zu potentiell schlechteren Ergebnissen als eine zu feine, da wenige Intervallgrenzen zu einer ungenauen Einteilung führen. Empfehlenswert wäre es also eine möglichst feine Auflösung zu wählen und durch die Wahl des Schwellwertes β die Qualität der Cluster zu steuern. Ein hohes β bewirkt, dass viele Einheiten verschmolzen werden, was zu qualitativ schlechteren Ergebnissen führt, jedoch deutlich schneller ist, da weniger Einheiten erstellt werden. Durch ein kleines β können deutlich komplexere Cluster genauer beschrieben werden, was sich jedoch direkt auf die Laufzeit auswirkt. Im Vergleich zu EQUALWIDTH ist bei gleich bleibender Qualität der Ergebnisse – also der gleichen Anzahl an Intervallen (EQUALWIDTH) im Vergleich zu der Anfangspartitionierung (MAFIA) – eine Verbesserung der Laufzeit zu erwarten, solange $\beta > 0$ eine Verschmelzung von Intervallen erlaubt.

3.4.2.2 Häufigkeit der Nachbarschaft

Die Wahl der Anfangspartitionierung ist selbst bei einen Blick auf die Daten, wie beispielsweise in Abbildung 3.11 schwer zu bestimmen. Entscheidend ist, eine Änderung der Häufigkeit an den richtigen Stellen zu erfassen, also direkt an den Stellen an denen ein Veränderung der Häufigkeit eine gewissen Schwellwert überschreitet. Bei MAFIA ist

dies abhängig von der anfänglichen Partitionierung. Die Wahrscheinlichkeit diese zu treffen steigt bei einer feinen Partitionierung, doch die Aussagekraft der Intervalle sinkt bei einer zu sensitiven Verteilung. Um dieses Verhalten zu verbessern wird in dieser Arbeit folgendes Verfahren vorgeschlagen.

Die wichtigen Stellen an denen sich die Häufigkeit der Beispiele ändert sind immer die Beispiele selbst. Die Problematik bei MAFIA entsteht dadurch, dass unabhängig von diesen Stellen die potentiellen Schnittpunkte durch die anfängliche EQUALWIDTH-Methode gesetzt werden. Alternativ hierzu kann also auch für jedes einzelne Beispiel überprüft werden, ob sich die Dichte der Beispiele innerhalb einer ϵ-Umgebung unterscheidet. Konkret bedeutet das, dass für jedes Beispiel links und rechts von diesem Beispiel ein Intervall erzeugt wird und wenn der Unterschied der Häufigkeiten beider Intervalle prozentual höher ist als β, dann wird das Beispiel als Schnittpunkt für die Intervalle hinzugefügt.

In Abbildung 3.13 ist dieses Vorgehen dargestellt. Der Algorithmus würde für ein Attribut A die Beispiele von links nach rechts durchlaufen und für jedes $x_A^{(i)}$ die Anzahl der Beispiele im Intervall $I_l^{(i)} = [x_A^{(i)} - \epsilon, x_A^{(i)})$ und $I_r^{(i)} = (x_A^{(i)}, x_A^{(i)} + \epsilon]$ ermitteln. Weichen diese Häufigkeiten prozentual um mehr als einen Schwellwert β ab, dann wird ein Schnittpunkt für ein Intervall erzeugt. Beim ersten Beispiel $x_A^{(1)}$ in Abbildung 3.13 gibt es keine weiteren Beispiele in I_l^1, so dass hier der erste Schnittpunkt erzeugt wird. Für den Fall, dass $x_A^{(1)}$ ein Ausreißer ist und damit ebenfalls in I_r^1 keine oder nur sehr wenige weitere Beispiele enthalten sind, kann dieser durch die Angabe eines weiteren, sehr geringen Schwellwertes angegeben werden, so dass dieses Beispiel ignoriert wird, wenn sich zu wenig weitere Beispiele in der Nachbarschaft befinden. An den entscheidenen Stellen,

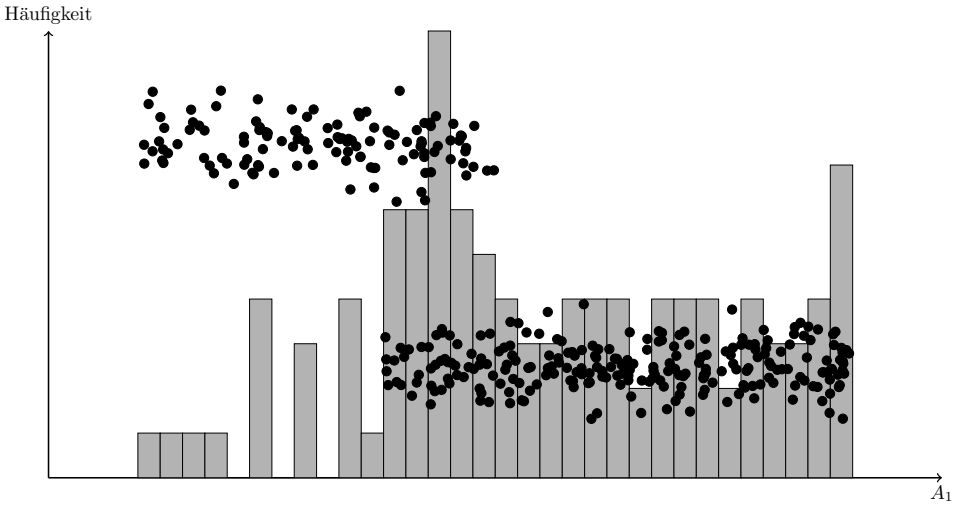


Abbildung 3.12: Sensitivität durch zu feine anfängliche Partitionierung

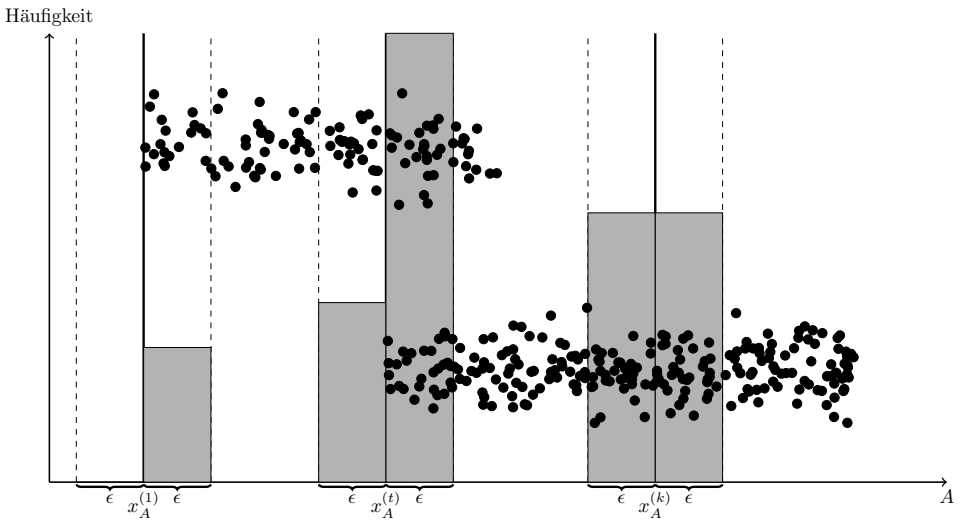


Abbildung 3.13: Partitionierung durch Häufigkeitsdichte in der Nachbarschaft veranschaulicht an drei Beispielen $x_A^{(1)}, x_A^{(t)}$ und $x_A^{(k)}$.

wie beispielsweise $x_A^{(t)}$ haben I_l^t und I_r^t stark unterschiedliche Häufigkeiten, die ab einem bestimmten β zu einem weiteren Schnittpunkt führen.

An gefundenen Schnittpunkten muss noch vermieden werden, dass beim nächsten Beispiel $x_A^{(t+1)}$ ebenfalls ein Schnittpunkt erzeugt wird. Es ist davon auszugehen, dass sich das Ungleichgewicht bei $x_A^{(t+1)}$ nicht grundlegend von $x_A^{(t)}$ unterscheidet. Beide Beispiele als Schnittpunkt zu verwenden würde damit nur unnötige Intervalle erzeugen. Ausgehend davon, dass I_r^t die „neue“ Häufigkeitsdichte beschreibt, wird nun $I_l^{t'} = I_r^t$ für ein neues $x_A^{(t')}$ gesetzt, so dass in diesem Bereich keine weiteren Unterteilungen vorgenommen werden. Damit legt ϵ eine Mindestgröße der Intervalle vor. Das gesamte Vorgehen ist in Algorithmus 3.3 beschrieben.

Im Vergleich zu MAFIA muss ebenfalls ein β als Parameter angegeben werden, der sich identisch verhält. Der Parameter ϵ kann mit der Anfangspartitionierung verglichen werden. Er bestimmt die Größe der Intervalle, die zur Berechnung der Häufigkeitsunterschiede verwendet wird. Dennoch hat dieser Parameter einen Vorteil gegenüber MAFIA, da ϵ deutlich höher gesetzt werden kann und somit nicht die Nachteile einer feinen Auflösung entstehen. Gleichzeitig werden die entscheidenden Schnittpunkte erkannt, da dort die Änderung der Häufigkeit gemessen wird. Da ϵ die Mindestgröße der Intervalle bestimmt und damit auch die Mindestgröße der gefundenen Cluster, ist diese Parameterwahl deutlich einfacher. Insbesondere ist dies der Fall, wenn bestimmte Kennzahlen oder Visualisierung von Teilen der Daten vorliegen an denen man die Größe der Gruppe abschätzen kann.

Die Laufzeit dieser Methode entspricht dabei der von MAFIA, wo für jedes Attribut die Beispiele zunächst sortiert werden müssen, um dann nochmals über jedes Beispiel zur

Algorithmus 3.3 Partitionierung durch die Nachbarschafts-Methode

Eingabe: Datensatz $\boldsymbol{X}$ mit den Attributen $\boldsymbol{A}$, Schwellwert β, Nachbarschaftsradius ϵ, Ausreißerschwellwert σ

Ausgabe: $\forall A_i \in \boldsymbol{A}$ eine Partitionierung $\boldsymbol{P}_i$ für A_i

```
for ∀A_i ∈ A do
    Sortiere X bzgl. A_i
    for x_{A_i} ∈ X do
        c_1 = s([x_{A_i} − ε, x_{A_i}))
        c_2 = s((x_{A_i}, x_{A_i} + ε])
        if (c_1 + c_2)/|X| < σ then
            Überspringe x_{A_i}
        end if
        if |c_1 − c_2| / max(c_1, c_2) > β then
            C_i = C_i ∪ {x_{A_i}}
            Fahre fort mit einem x_{A_j} > x_{A_i} + ε
        end if
    end for
    Erzeuge P_i mit den Schnittpunkten aus C_i
end for
```

Berechnung der Häufigkeiten der Intervalle zu iterieren. Durch eine Implementierung eines laufenden Fensters anhand zweier Warteschlangen, die die beiden Intervalle repräsentieren, benötigt die Nachbarschaftsmethode auch nur eine Iteration über die sortierten Daten.

3.5 Von häufigen Mengen zu dichten Einheiten

Die Verbindung zwischen den dichten Einheiten und den häufigen Mengen wurde in Kapitel 3.2.2 erläutert. In der Orginalliteratur [4] wurden zwar Implementierungen aus den APRIORI Veröffentlichungen verwendet, jedoch für die Erfordernisse von CLIQUE angepasst.

In der vorliegenden Arbeit wird die Suche nach den dichten Einheiten vollständig in das „Häufige Mengen“-Problem transformiert. Dies hat den Vorteil, dass die Implementierung so generisch bleibt und prinzipiell jede Implementierung aus einem gut und lang erforschten Gebiet verwendet werden kann. Somit kann dieser Algorithmus auch innerhalb eines Stream-Kontextes [34] verwendet werden, wenn Algorithmen zum Finden von häufigen Mengen in Streams (wie beispielsweise LOSSYCOUNTING [54]) verwendet werden. Aber auch eine Parallelisierung (siehe Kapitel 3.6) ist so einfacher möglich.

A_1	A_2	$\cdots$	A_p
$I^3_{A_1}$	$I^8_{A_2}$	$\cdots$	$I^5_{A_p}$
$I^4_{A_1}$	$I^2_{A_2}$	$\cdots$	$I^2_{A_p}$
$I^3_{A_1}$	$I^2_{A_2}$	$\cdots$	$I^9_{A_p}$
$I^9_{A_1}$	$I^4_{A_2}$	$\cdots$	$I^1_{A_p}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
$I^4_{A_1}$	$I^8_{A_2}$	$\cdots$	$I^9_{A_p}$

(a) Diskretisierte Datenbank – für jedes Beispiel wurde ein nominales Attribut erzeugt, das angibt in welchem Intervall das Beispiel für dieses Attribut liegt.

$I^3_{A_1}$	$I^4_{A_1}$	$\cdots$	$I^4_{A_2}$	$\cdots$	$I^8_{A_2}$	$\cdots$	$I^9_{A_p}$
1	0	$\cdots$	0	$\cdots$	1	$\cdots$	0
0	1	$\cdots$	0	$\cdots$	0	$\cdots$	0
1	0	$\cdots$	0	$\cdots$	0	$\cdots$	1
0	0	$\cdots$	1	$\cdots$	0	$\cdots$	0
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
0	1	$\cdots$	0	$\cdots$	1	$\cdots$	1

(b) Binäre Transaktionsdatenbank - jedes Intervall für jedes Attribut ist nun ein eigenes Attribut indem entweder ein Beispiel enthalten ist (Wert 1) oder nicht (Wert 0)

Tabelle 3.1: Transformation eines diskretisierten Datensatzes zu einer Transaktionsdatenbank

3.5.1 Transformation in Transaktionsdatenbank

Bisher wurde nur der numerische Raum diskretisiert, um Einheiten zu bilden (siehe Kapitel 3.4). Somit liegt ein Datensatz vor, der die selbe Dimensionalität aufweist, nun aber aus nominalen Attributen besteht, wie in Tabelle 3.1a zu sehen.
Wie in Grundlagenkapitel 2.4 erläutert, braucht ein Algorithmus zum Finden von häufigen Mengen jedoch eine Transaktionsdatenbank als Eingabe. Diese Transformation lässt sich jedoch einfach durchführen.

Definition 3.5.1 (Transformation in eine Transaktionsdatenbank). Sei $\vec{x} \in \boldsymbol{X} \subseteq \mathbb{A}^p$ ein Beispiel aus einem Datensatz $\boldsymbol{X}$, $\boldsymbol{P}$ eine Partitionierung von $\mathbb{A}^p$ und $u^{(i)} \in \boldsymbol{P}$ eine Einheit für die $\vec{x} \sqsubset u^{(i)}$ gilt, dann ist T eine Transaktionsdatenbank mit

$$\boldsymbol{R} = \bigcup_{u^{(i)}} \bigcup_{j=1}^{p} \{u_j^{(i)}\}$$

und eine Transformation von $\boldsymbol{X}$ nach T definiert durch $t : \boldsymbol{X} \to T$ mit

$$t(\vec{x}) = \{u_1^{(i)}, \ldots, u_p^{(i)}\}$$

Die erzeugte Menge aller Items $\boldsymbol{R}$ ist also jedes Intervall, das für jedes Attribut erstellt wurde. Ein Beispiel ist dann eine Transaktion und beinhaltet die Items, die die Intervalle repräsentieren in denen das Beispiel beinhaltet war. Der Support einer Menge von Items ist damit die absolute Häufigkeit der Beispiele, die in den von den Items repräsentierten Intervallen liegen.

Proposition 3.5.2. *Sei T eine durch eine Transformation $t : \boldsymbol{X} \rightarrow T$ aus einem Datensatz $\boldsymbol{X}$ erzeugte Transaktionsdatenbank und $S(T, \xi_{rel})$ die gefundenen häufigen Mengen für einen relativen Schwellwert $\xi_{rel} \in [0, 1]$, dann sind alle Einheiten $w \in S(T, \xi_{rel})$, wobei $\exists u^{(i)} : w \subseteq u^{(i)}$ gilt, dichte Einheiten bezüglich des Schwellwertes $\tau = \xi_{rel}$.*

Beweis. Die Aussage folgt direkt aus der Transformation der Daten in Definition 3.5.1. Für jede gefundene häufige Menge w gibt es mindestens $|\boldsymbol{X}| \cdot \xi_{\text{rel}}$ Transaktionen, die diese Menge beinhalten. Eine Transaktion repräsentiert nach der Definition 3.5.1 ein Beispiel und jedes Item ein Intervall, in dem das Beispiel enthalten ist. Folglich existieren mindestens $|\boldsymbol{X}| \cdot \xi_{\text{rel}}$ Beispiele innerhalb der Einheit w, so dass w bezüglich des Schwellwertes ξ_{rel} dicht ist. □

3.6 Parallele häufige Mengen

Durch die gegebene Transformation und die Möglichkeit vorhandene Implementierungen von Algorithmen zu verwenden, können performante Algorithmen wie beispielsweise FPGROWTH verwendet werden. Dies überträgt die Vorteile dieses Algorithmus, die in Kapitel 2.4.4 besprochen wurden, ebenfalls auf das Subspace Clustering.
Wie in Kapitel 2.5 angedeutet, ist in Zukunft aber eine große Leistungssteigerung eher durch den Einsatz von parallelen Architekturen zu erwarten. Im folgenden werden drei unterschiedliche Ansätze zur Parallelisierung aufgelistet und bezüglich des zu lösenden Problems bewertet.

Cluster Die Verwendung eines Clusters ist insbesondere bei sehr rechenintensiven Problemen mit vielen Daten sinnvoll, um den Mehraufwand der Koordination und Kommunikation zu kompensieren. In Kapitel 2.5.1 wurde hierfür das MAPREDUCE [29] Konzept vorgestellt. Eine Open Source Implementierung ist HADOOP [17], ein Framework, das darauf ausgelegt ist in einem heterogenen Verband von verschiedenen Rechnern, Terrabytes an Daten zu verarbeiten. MAHOUT [60] ist eine auf HADOOP aufbauende Bibliothek, die Algorithmen des maschinellen Lernens implementiert und auch eine FPGROWTH-Implementierung beinhaltet.
In der vorliegenden Arbeit werden jedoch nur Daten mit weniger als 200 Megabyte verarbeitet, so dass eine Fokussierung auf andere parallele Architekturen für Aufgaben dieser Größenordnung sinnvoll erscheinen lässt. Insbesondere ist die Entwicklung einer Schnittstelle zwischen den hier verwendeten Systemen RAPIDMINER (siehe Kapitel 3.9) und HADOOP nicht trivial. RADOOP [64] ist ein Projekt, das an dieser Schnittstelle arbeitet, und trotz mehrerer Monate Entwicklung durch verschiedene Entwickler keine stabile Version zum Zeitpunkt dieser Arbeit veröffentlicht hat. Dank der modularen Im-

plementierung innerhalb dieser Arbeit, ist die spätere Verwendung dieses Projekts jedoch möglich, so dass auch größere Datenmengen verarbeitet werden können.

Grafikkarten Durch die Spieleindustrie angetrieben, wurde viel bei der Entwicklung von hochspezialisierten Grafikkarten geforscht. Auffällig ist dabei die starke Parallelisierung. Das liegt insbesondere daran, dass sich typische Berechnungen einer Grafikkarte sehr gut parallelisieren lassen und jeder dieser Prozessoren für diesen Verwendungszweck optimiert ist. So ist die Hardware darauf ausgelegt, dass eine Gruppe von Kernen dieselbe Berechnung auf verschiedenen Daten durchführt, so dass eine Grafikkarte zum Teil die Eigenschaften einer **SIMD**-Architektur (siehe Kapitel 2.5) erfüllt.
Die Verwendung von Grafikkarten außerhalb von Grafikanwendung wird als *General Purpose Computation on Graphics Processing Unit* (GPGPU) bezeichnet. Auch im Bereich des maschinellen Lernens wird versucht diese Geschwindigkeitsvorteile, die durch die massive Parallelisierung entstehen, zu nutzen. Um jedoch das Potential einer Grafikkarte auszunutzen, muss bei der Implementierung auf die spezialisierten Operationen der Grafikkarten geachtet werden.
Beim Finden von häufigen Mengen gibt es jedoch noch nicht viele Veröffentlichungen. In [32] wird für die CUDA-Schnittstelle der NVidia-Grafikkarten ein paralleler APRIORI Algorithmus implementiert, der jedoch um den Faktor 4 bis 16 langsamer ist im Vergleich zu einer FPGROWTH-Implementierung auf der CPU. Die Autoren arbeiten nach eigenen Angaben an einer FPGROWTH Variante, die bis zum jetzigen Zeitpunkt nicht erschienen ist. Eine weitere Arbeit ist [79], die nach eigenen Angaben auf Datensätzen mit typischer Größe um einen Faktor 4 bis 10 schneller ist, als aktuelle APRIORI Implementierungen. Vergleichende Experimente mit Algorithmen wie FPGROWTH werden jedoch nicht durchgeführt.
Weitere Arbeiten zu der Berechnung von häufigen Mengen auf Grafikkarten sind nicht bekannt. Da die bisherigen Ergebnisse nicht besonders vielversprechend sind und die Entwicklung von einer effizienten GPGPU-FPGROWTH Variante aufgrund der speziellen Gegebenheiten einer Grafikkarte noch nicht erfolgt ist, erscheinen die Erfolgsaussichten bei dieser Art der Parallelisierung eher moderat und würden den Rahmen dieser Arbeit sprengen.

Multikern- und Multiprozessorsysteme Neue Computersysteme besitzen oftmals mehrere Prozessoren, die wiederum mehrere Prozessorkerne besitzen und damit mehrere Berechnungen parallel ausführen können. Erste Arbeiten zur Parallelisierung der Suche nach häufigen Mengen auf solchen Systemen begannen mit einer modifizierten Version von APRIORI in [77]. Neuere Veröffentlichungen konzentrieren sich dagegen auf aktuellere und schnellere Verfahren wie beispielsweise FPGROWTH (siehe [23, 52, 76]), auf die im Folgenden näher eingegangen wird.

FPGROWTH

Grundsätzlich können die Ansätze zur Parallelisierung von FPGROWTH auf Multikern- und Multiprozessorsystemen in zwei Kategorien eingeteilt werden. Wie in Kapitel 2.4.4 beschrieben, erstellt FPGROWTH eine Datenstruktur namens FPTREE, um daraus die häufigen Mengen zu erstellen. Je nachdem, ob ein FPTREE parallel bearbeitet wird oder mehrere FPTREEs zur parallelen Verarbeitung erstellt werden, können die Algorithmen in *Single-Tree-* oder *Multi-Tree-*Ansätze aufgeteilt werden.

Single-Tree-FPGROWTH Nach der initialen Erstellung des FPTREEs werden *Conditional Pattern Base*s und die dazugehörigen *Conditional* FPTREEs erzeugt und rekursiv weiter verarbeitet. Dies ist ein aus der Informatik bekanntes Teile-und-Herrsche Vorgehen. Da die *Conditional Pattern Base*s voneinander unabhängig sind, können sie dann auch prinzipiell parallel bearbeitet werden. Damit bearbeiten diese Algorithmen [52, 42, 24] einen einzelnen anfangs aufgebauten FPTREE, so dass dies als ein *Single-Tree-*Ansatz bezeichnet werden kann.
Laut [52] hat [42] gezeigt, dass ein solcher Ansatz nicht gut skaliert. [24, 52] versuchen diese Nachteile durch verschiedene Ansätze bezüglich guter Nutzung des Prozessor-Caches und Lastenverteilung zu beheben. Die Komplexität der Implementierung ist jedoch recht hoch und sehr systemnah durch Verwendung von verschiedenen Cache-Strategien.

Multi-Tree-FPGROWTH Auf jeder ausführenden Einheiten kann auch ein eigener Baum erstellt und verarbeitet werden. Diese werden anschließend zu einem Endergebnis aggregiert. Das Verfahren, das auch bei MAHOUT verwendet wird, ist in [48] beschrieben. Dabei werden verschiedene Gruppen von Items gebildet, um daraus eine gruppenabhängige Transaktionsdatenbank zu erstellen, für die mehrere FPTREEs erstellt werden, die parallel bearbeitet werden. Bei [63] wird auf den lokalen Daten ein FPTREE erstellt, wobei die *Conditional Pattern Base*s bezüglich der globalen Daten erstellt werden, was Kommunikation zwischen den verarbeitenden Einheiten benötigt.
Es gibt jedoch auch einen allgemeinen Ansatz, der sogar vom eigentlichen Algorithmus unabhängig ist. Folglich können beliebige serielle Verfahren, wie auch APRIORI oder ECLAT [78], verwendet werden. Da das Ziel der Arbeit eine modulare Implementierung ist, bei der einzelne Module leicht ausgewechselt werden können, wird dieser Ansatz in dieser Arbeit verfolgt. Da das Framework innerhalb von RAPIDMINER entworfen wird (siehe Kapitel 3.9), ist ein weiterer Vorteil, dass bei diesem Vorgehen die bewährte und effiziente Implementierung von FPGROWTH, die schon in RAPIDMINER integriert ist, verwendet werden kann.
Schon kurz nach der Veröffentlichung von APRIORI wurde [6] veröffentlicht. Diese Arbeit greift die Idee auf, die Daten in gleich große Teile zu zerteilen und auf diesen un-

abhängig voneinander die häufigen Mengen zu berechnen. Es wurden jedoch weiterhin Nachrichten zwischen den verarbeitenden Einheiten verschickt, um die Zähler bei der Kandidatengenerierung zu aktualisieren. Ein ähnliches Konzept verwendet auch [47].
Die wichtige Idee bei diesem Ansatz ist, dass wenn die gesamte Transaktionsdatenbank in gleich große Teile unterteilt wird, dann auch jede global häufige Menge in mindestens einer Teilmenge der Datenbank häufig sein muss.

Proposition 3.6.1. *Sei T eine Transaktionsdatenbank, die in k gleich große Mengen $P = \{P_1, \ldots, P_k\}$ partitioniert ist. Der Support einer Menge von Items I in einer Partitionen P_i wird berechnet durch*

$$s(I; P_i) = |\{t \in P_i : I \subseteq t\}|$$

so dass gilt

$$\frac{s(I)}{|T|} \geq \xi_{rel} \Rightarrow \exists P_i : \frac{s(I; P_i)}{|P_i|} \geq \xi_{rel}$$

.

Beweis. Angenommen es sei $s(I) \geq \xi_{\text{rel}}|T|$ und $P = \{P_1, \ldots, P_k\}$ eine Partition, so dass $\forall P_i : s(I; P_i) < \xi_{\text{rel}}|P_i|$. Da eine P eine Partition von T ist, muss für den Support gelten, dass $s(I) = \sum_{i=1}^{k} s(I; P_i)$. Damit gilt dann

$$\xi_{\text{rel}}|T| \leq s(I) = \sum_{i=1}^{k} s(I; P_i) < \sum_{i=1}^{k} \xi_{\text{rel}}|P_i| = \xi_{\text{rel}} \sum_{i=1}^{k} |P_i|$$

also

$$\xi_{\text{rel}}|T| < \xi_{\text{rel}} \sum_{i=1}^{k} |P_i| \Leftrightarrow |T| < \sum_{i=1}^{k} |P_i|$$

Damit kann P aber keine Partition von T sein, da die Summe der Transaktionen der einzelnen Mengen der Partition gleich der Anzahl der Transaktionen von T sein muss. □

Wenn nun alle häufigen Mengen $S(T, \xi_{\text{rel}})$ einer Transaktionsdatenbank T gefunden werden sollen, dann kann T in die Datenbanken $P_1, \ldots, P_k$ partitioniert werden, so dass $S(T, \xi_{\text{rel}}) \subseteq \bigcup_{P_i} S(P_i, \xi_{\text{rel}})$. Alle häufigen Mengen sind also in $\bigcup_{P_i} S(P_i, \xi_{\text{rel}})$ enthalten, doch heißt das nicht, dass die beiden Mengen äquivalent sind. Eine Menge kann durchaus innerhalb einer Datenbank P_i häufig sein, aber in allen restlichen $P_j \neq P_i$ nicht mehr vorkommen und insgesamt nicht mehr häufig sein in T. Das sind die sogenannten *False Positives*, die ungerechtfertigterweise ausgegeben werden, falls $\bigcup_{P_i} S(P_i, \xi_{\text{rel}})$ als Ergebnis verwendet wird.
Im schlimmsten Fall ist eine Menge I nur in einer Menge der Partition enthalten und erreicht dort genau den benötigten Schwellwert. Dann ist die globale Häufigkeit genau $\frac{\xi_{\text{rel}}}{k}|\boldsymbol{X}|$. Es wird also eine Menge als häufig ausgegeben, die jedoch $(k-1)\xi_{\text{rel}}|\boldsymbol{X}|$ mal weniger vorkommt, als sie es müsste, um ausgegeben zu werden. Es ist also von der Anzahl der Teildatenbanken abhängig, wie groß der maximale Fehler werden kann. Die

Anzahl der Datenbanken wiederum bestimmt den Grad der Parallelität, der erreicht werden kann.
Um den Fehler zu korrigieren können mit einem zusätzlichen Lauf über die Daten die *False Positives* ermittelt und aus dem Ergebnis entfernt werden. Welchen Anteil dies an der Gesamtlaufzeit hat, wird in Kapitel 4.2 experimentell ermittelt.
Um die Anzahl der erstellten *False Positives* zu reduzieren und damit auch die Laufzeit der einzelnen Instanzen auf den Teildatenbanken zu verbessern, können in einem Vorverarbeitungsschritt die einelementigen, nicht häufigen Mengen ermittelt werden. Nach der Monotonieeigenschaft (siehe 2.4.4) sind diese Mengen für die mehrelementigen, häufigen Mengen nicht von Bedeutung. Folglich können die Items aus der Transaktionsdatenbank T entfernt werden. Bei FPGROWTH werden diese zwar schon vor der Erstellung des initialen FPTREEs entfernt, doch durch die Partitionierung von T kann ein global gesehen nicht häufiges Item innerhalb von P_i häufig sein und somit bei der Erstellung des FPTREEs beteiligt sein.
Diese Parallelisierung lässt sich sehr gut auf das MAPREDUCE Konzept abbilden (siehe auch Kapitel 2.5.1). Nachdem die Daten von den nicht häufigen Items bereinigt wurden, werden sie in gleich große Teile aufgeteilt.Für jede Teilmenge wird eine Map-Funktion gestartet, die die häufigen Mengen auf diesen Daten berechnet. Dies kann mit jedem beliebigen Verfahren durchgeführt werden, wobei sich diese Arbeit auf FPGROWTH bezieht. Die Reduce-Funktion sammelt anschließend die Ergebnisse, entfernt Duplikate und entfernt die Mengen, die bezüglich der Gesamtdaten nicht häufig sind.

3.7 Pruning

Agrawal et al. [4] bewerten die Vorgehensweise von CLIQUE als eine „drastische Reduzierung der Anzahl der Einheiten, die auf Dichte getestet werden müssen“. Trotzdem sei die „Berechnung für hochdimensionale Daten kaum handhabbar“. Dadurch motiviert wird die Kandidatengenerierungsphase genutzt, um nach einer Heuristik ganze Subspaces zu entfernen, um die Anzahl der Einheiten weiter zu reduzieren. Dies wird *Pruning* genannt.
Die Heuristik nach der bewertet wird, wann ein Subspace nicht mehr weiterverwendet wird, bezieht sich bei [4] auf „interessante“ Subspaces. Hierfür wird das Prinzip der **Minimum Description Length** (MDL) verwendet, das im Kontext der Informationstheorie entwickelt wurde [66].
In jeder Iteration von CLIQUE wird, nachdem die k-dimensionalen, dichten Einheiten erstellt wurden, für alle k-dimensionalen Subspaces der prozentuale Anteil der Beispiele berechnet, die von den dichten Einheiten in diesem Subspace abgedeckt werden. Anschließend werden die Subspaces bezüglich dieser Abdeckung absteigend sortiert und ein Schnittpunkt gesucht, so dass alle Subspaces, die eine kleinere Abdeckung haben,

aus den weiteren Iterationen ausgeschlossen werden. Folglich werden also auch keine Kandidaten aus den dichten Einheiten in diesem Unterraum erstellt.
Für jeden der potentiellen Schnittpunkte werden dann zwei Durchschnitte von den Abdeckungen der beiden so getrennten Mengen von Subspaces berechnet, um daraus die benötigte Kodierungslänge zu ermitteln. Dieses Maß gibt an, wie viele Bits benötigt werden, um den Abstand einer jeden Abdeckung zu den jeweiligen Durchschnitt zu speichern. Der Schnittpunkt, der die kleinste Kodierungslänge hat, wird zum *Pruning* gewählt.
ENCLUS [25] baut auf diesem Prinzip auf und stellt weitere Kriterien neben der Abdeckung vor, die durch eine Entropie-basierte Methode angewandt werden.
Zu beachten ist jedoch, dass die Verwendung einer solchen Pruning-Strategie, den Algorithmus zwar schneller macht, dies aber auf Kosten von nicht gefundenen Clustern. Darauf wird auch ausdrücklich von [4] hingewiesen. In anderen Arbeiten[59] wird bewusst auf dieses *Pruning* verzichtet und die Nachteile werden durch eine geschicktere Partitionierung kompensiert.
In dieser Arbeit wird ebenfalls auf das *Pruning* verzichtet. Ein Grund hierfür ist zusätzlich, dass in dieser Arbeit das Subspace Clustering vollständig in das Problem des Findens von häufigen Mengen transformiert wird, um vorhandene und schnellere Implementierungen als APRIORI zu verwenden. In Verfahren, wie beispielsweise FPGROWTH, ist ein *Pruning* aufgrund der mangelnden Kandidatengenerierung auch gar nicht möglich.

3.8 Erstellung der Ausgabe

Das Ergebnis der letzten drei Module ist eine Menge von dichten Einheiten in unterschiedlichen Subspaces. Der nächste und letzte Schritt ist nach [4] das Finden von zusammenhängenden dichten Einheiten innerhalb des selben Unterraumes. Diese bilden die Subspace Cluster. Anschließend wird für diese Cluster eine einfache und intuitive Repräsentation in Form der minimalen Beschreibung erstellt und ausgegeben.

3.8.1 Von dichten Einheiten zu Subspace Clustern

Nach der Definition 3.2.11 besteht ein Subspace Cluster aus einer Menge von dichten Einheiten, die zusammenhängend sind und sich im selben Subspace befinden. Die Aufgabe ist nun also aus einer Menge von dichten Einheiten die Teilmengen zu bestimmen, so dass innerhalb jeder Teilmenge diese Eigenschaften gelten. Dabei ist zu beachten, dass die Teilmengen maximal sind, d.h. dass keine Einheit einer Teilmenge auch Element einer anderen Teilmenge sein kann ohne diese Eigenschaften zu verletzen.

Zunächst wird die Menge der Einheiten partitioniert bzgl. der Zugehörigkeit zu den jeweiligen Subspaces. Jede dieser Teilmengen der Partitionierung kann nun unabhängig von den anderen Teilmengen weiter in die zusammenhängenden Einheiten partitioniert werden. Durch die Unabhängigkeit lässt sich dieser Schritt leicht parallelisieren. Die einzelnen Teilprobleme sind dann äquivalent zu dem Finden von Zusammenhangskomponenten eines ungerichteten Graphen in der Graphentheorie. Dabei stellt jede dichte Einheit einen Knoten dar und zwei Knoten sind miteinander verbunden, wenn sie eine gemeinsame Oberfläche haben. Die Einheiten in einer solchen Zusammenhangskomponente sind dann nach Definition zusammenhängende Einheiten und bilden somit einen Subspace Cluster.

Algorithmus 3.4 Tiefensuche zum Finden von zusammenhängenden Einheiten

Eingabe: $\boldsymbol{D} = \{u^{(1)}, \ldots, u^{(m)}\}$
Ausgabe: Partitionierung von $\boldsymbol{D}$ in zusammenhängende Einheiten
  while $u^{(i)} \in \boldsymbol{D} \wedge u^{(i)}$ noch nicht besucht **do**
    return DFS$(u^{(i)})$ als eine Menge der Partition
  end while
  procedure DFS$(u^{(i)})$
    $\boldsymbol{U} = \{u^{(i)}\}$
    Markiere $u^{(i)}$ als besucht
    for $u^{(j)} \in \boldsymbol{D} \wedge u^{(i)} \boxminus u^{(j)}$ und $u^{(j)}$ noch nicht besucht **do**
      $\boldsymbol{U} = \boldsymbol{U} \cup$ DFS$(u^{(j)})$
    end for
    return $\boldsymbol{U}$
  end procedure

Die Zusammenhangskomponenten können einfach mittels Breiten- oder Tiefensuche [27] gefunden werden. Bei der Tiefensuche wird eine beliebige, jedoch noch nicht besuchte Einheit ausgewählt, als besucht markiert und alle Einheiten, die eine gemeinsame Oberfläche mit dieser Einheit haben und ebenfalls noch nicht besucht wurden, gesucht. Auf diesen gefundenen Einheiten wird diese Suche erneut rekursiv aufgerufen bis keine nicht besuchten Einheiten gefunden werden. Alle so gefundenen Einheiten sind zusammenhängend und bilden einen Cluster. Auf die restlichen Einheiten, die so nicht gefunden wurden, wird diese Methode ebenfalls angewandt, um weitere Cluster zu finden. Der Algorithmus ist in 3.4 angegeben.

Da jede Einheit für jedes Attribut genau 2 Nachbarn haben kann, müssen bei k Attributen $2k$ Überprüfungen durchgeführt werden, so dass für die Laufzeit für m dichte Einheiten $\mathcal{O}(km)$ gilt.

3.8.2 Erstellung der minimalen Beschreibung

Nun wurden zwar die Subspace Cluster identifiziert, doch eine Repräsentation als eine Menge von Einheiten ist nicht so leicht zu interpretieren. In [4] wurde hierfür die minimale Beschreibung vorgeschlagen (siehe Definition 3.2.15). Es stellt sich dabei heraus, dass das Problem selbst für den 2-dimensionalen Fall NP-schwer ist. Bekannte Approximationen sind nach [4] nicht für den hochdimensionalen Raum geeignet, so dass ein *Greedy*-Algorithmus vorgeschlagen wird.

Jeder gefundene Subspace Cluster wird dabei unabhängig von den weiteren Clustern als Eingabe verarbeitet, was auch hier eine Parallelisierung ermöglicht. Der Algorithmus geht dabei folgendermaßen vor. Aus den zusammenhängenden, dichten Einheiten wird eine zufällige Einheit gewählt und einer neu erstellten Region, die zunächst nur aus dieser Einheit besteht, zugewiesen. Diese Region wird für jedes Attribut erweitert.

Für ein Attribut A_i bedeutet das, dass zunächst alle linken Nachbareinheiten aller Einheiten einer Region überprüft werden, ob sie ebenfalls im Subspace Cluster enthalten sind. Ist dies der Fall werden diese Nachbarn ebenfalls der Region zugewiesen und der Vorgang solange wiederholt bis die Nachbarn diese Bedingung nicht mehr erfüllen. Anschließend wird dies mit den rechten Nachbarn wiederholt.

Angewandt auf alle Attribute expandiert die Region von einer Einheit solange im Raum bis die entstehende Region nicht mehr im Cluster vorhanden ist. Nach Definition ist diese Region maximal, da sie in keiner Dimension erweitert werden kann, ohne nicht mehr im Cluster enthalten zu sein. Ist dies der Fall, wird die nächste dichte Einheit im Subspace Cluster gewählt, die noch keiner Region zugewiesen worden ist und die Prozedur wiederholt bis alle dichten Einheit in einer Region enthalten sind, so dass am Ende der gesamte Custer mit Regionen abgedeckt ist. Die Definitionen in Kapitel 3.2.3 beschreiben, wie aus diesen Regionen die minimale Beschreibung erstellt wird.

Für jeden gefundenen Subspace Cluster beträgt dabei die Laufzeit bei diesem Vorgehen $\mathcal{O}(m^{\frac{2(k-1)}{k}})$ (siehe [4]).

3.9 Implementierung

Die in den vorherigen Kapiteln vorgestellten Module wurden zur experimentellen Analyse in Kapitel 4 implementiert. Das folgende Kapitel stellt diese Implementierung innerhalb eines Frameworks vor und erläutert wie die einzelnen Module im Zusammenspiel ein Subspace Clustering als Ergebnis erzeugen.

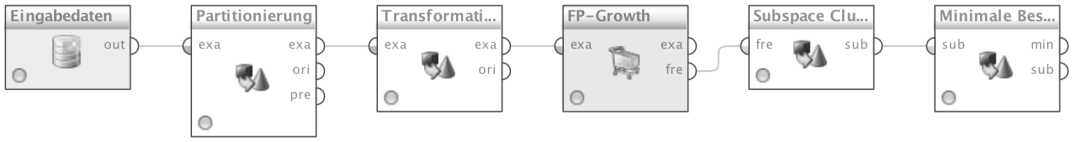


Abbildung 3.14: Beispiel für einen Subspace Clustering Prozess in RAPIDMINER

3.9.1 RAPIDMINER

Im Bereich des maschinellen Lernens wiederholen sich oftmals viele Aufgaben. Ein Datensatz muss eingelesen, vorverarbeitet, bereinigt und normalisiert werden, bevor ein Lernverfahren gestartet wird und die Ergebnisse evaluiert und gespeichert werden. Damit die benötigten Operationen in anderen Kontexten wiederverwendet werden können, wurde RAPIDMINER [56] entwickelt.
In einer Art Baukastenprinzip gibt es verschiedene Operatoren, die Daten erhalten, verarbeiten und weiterschicken. Ein Prozess kann so aus einer Vielzahl von schon vorhandenen Operatoren zusammengestellt und gestartet werden. Um das System mit eigenen Operatoren zu erweitern, existiert ein Plugin-Mechanismus.
Dieser Mechanismus soll in dieser Arbeit verwendet werden, um die fehlenden Operatoren zu implementieren und den modularen Ansatz von RAPIDMINER zu nutzen, um die verschiedenen Schritte des Algorithmus zu implementieren. Dies erlaubt es schon vorhandene Operatoren zu nutzen und das Verfahren durch das Hinzufügen von weiteren Operatoren zur Partitionierung oder zum Suchen von häufigen Mengen zu erweitern.
RAPIDMINER ist dabei in Java implementiert und wird in der Version 5.1 verwendet.

3.9.2 Implementierte Operatoren

In Abbildung 3.14 ist ein Beispiel aus einem RAPIDMINER-Prozess dargestellt, der ein Subspace Clustering erstellt. Zunächst werden die Daten eingeladen und eine Partition erstellt. Es folgt eine Transformation dieser Daten in eine Transaktionsdatenbank, so dass ein FPGROWTH-Operator die häufigen Mengen finden kann. Dieses Ergebnis ist die Eingabe eines Operators, der aus den häufigen Mengen in ein Subspace Clustering transformiert. In einem letzten Schritt wird dieses Clustering in eine minimale Beschreibung überführt.
Abgesehen vom Einladen der Daten und dem FPGROWTH-Operator mussten die restlichen Operatoren implementiert und in einem Plugin eingebunden werden. Wie in Kapitel 3.4 erläutert, ist die Partitionierung ein wichtiger Aspekt, so dass hier verschiedene Operatoren verwendet werden können. Eine EQUALWIDTH- und EQUALFREQ-Methode

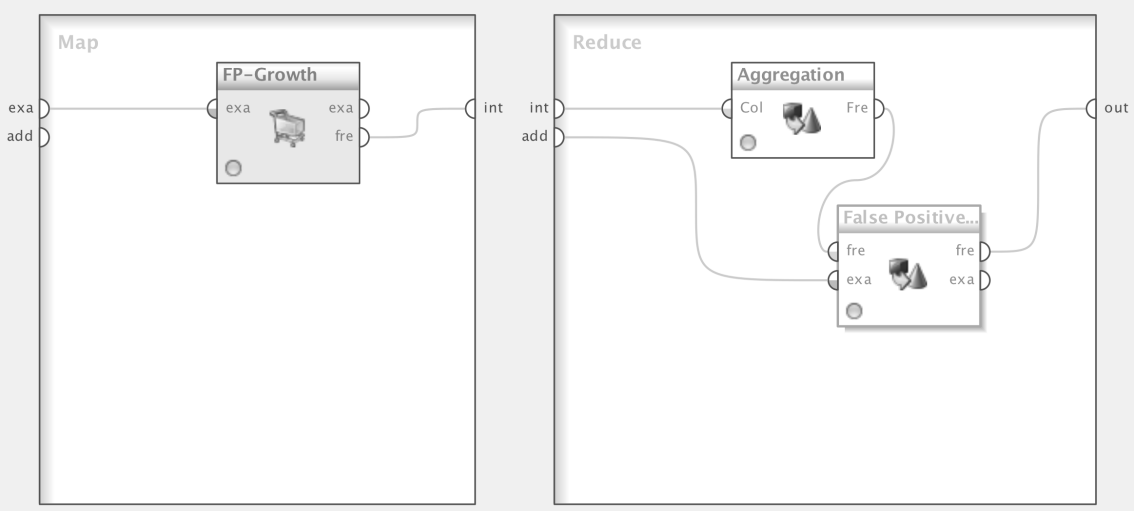


Abbildung 3.15: Ausführende Einheiten innerhalb des MAPREDUCE-Operators zum parallelen Finden von häufigen Mengen

ist in RAPIDMINER schon vorhanden. Die restlichen Operatoren zur Diskretisierung von Merkmalen sind jedoch überwachte Methoden und sind für das Clustering nicht relevant. Zusätzlich wurde die MAFIA- und Nachbarschafts-Partitionierung eingebunden. Weitere Methoden können leicht als Operatoren eingesetzt werden.
Für eine weitere Auswertung und zur experimentellen Analyse wurden auch weitere Operatoren implementiert, die beispielsweise die Abdeckung der Subspace Cluster berechnet. Insgesamt wurden in dieser Arbeit 45 Klassen implementiert, die innerhalb von 20 Operatoren im Subspace Clustering Plugin zur Verfügung stehen. Der Quellcode steht auf der CD-ROM im Anhang C zur Verfügung.

3.9.3 MAPREDUCE

Um durch Parallelisierung eine Verbesserung der Laufzeit zu erreichen, ist auch ein lokales, auf Threads basierendes MAPREDUCE-Framework implementiert und steht im Plugin zur Verfügung. Dabei existieren zunächst keinerlei Abhängigkeiten zwischen dem Framework und RAPIDMINER. Es werden lediglich Schnittstellen zu den Map- und Reduce-Funktionen bereitgestellt, die implementiert werden müssen. Dies gewährleistet, dass die Algorithmen nur auf die beiden Funktionen abgebildet werden und damit auch eine einfachere Portierung auf nicht lokale Verfahren möglich ist.

Parallele häufige Mengen Das implementierte Framework wird verwendet, um einen MAPREDUCE-Operator zu erstellen. In RAPIDMINER können Operatoren weitere Operatoren beinhalten, die wiederum in mehreren ausführenden Einheiten gruppiert sind. Abbildung 3.15 zeigt das Innere des MAPREDUCE-Operators, das zwei ausführende Ein-

heiten beinhaltet, die der Map- und der Reduce-Funktion entsprechen. Der eingehende Datensatz wird automatisch in so viele Teildatensätze aufgeteilt, wie Threads beim Operator als Parameter angegeben sind. Somit können auch andere seriell implementierte Operatoren in einem parallelem Kontext ausgeführt werden.
In Abbildung 3.15 wird der Operator verwendet, um für jeden Teildatensatz die häufigen Mengen anhand des FPGROWTH-Operators zu ermittleln. In der Reduce-Phase werden zwei Operatoren verwendet, um zunächst alle häufigen Mengen zu aggregieren bzw. Duplikate zu entfernen, und dann die *False Positives* zu entfernen. Der gesamte MAPREDUCE-Operator kann nun den FPGROWTH-Operator in Abbildung 3.14 ersetzen.

Parallele Partitionierung Das implementierte MAPREDUCE-Framework wird auch an weiteren Operatoren eines Subspace Clustering Prozesses eingesetzt. Auch die Partitionierung der einzelnen Attribute wird mit diesem Verfahren beschleunigt. Die Daten werden hierbei nicht horizontal in mehrere Teilmengen von Beispielen, sondern vertikal geteilt. Das bedeutet, dass jede Map-Funktion eine Menge von Attributen erhält, die sie unabhängig von den anderen Attributen bearbeiten und partitionieren kann. Hierfür wurde die Oberklasse des Diskretisierungsoperators in RAPIDMINER erweitert, so dass dies gleich für alle Operatoren, also auch für die nicht verwendeten überwachten Diskretisierungstechniken, verwendet werden kann. Die Reduce-Funktion muss nur noch die einzelnen Attribute zu einem Datensatz zusammenführen.

Parallele Erstellung der Ausgabe Neben der Partitionierung kann das Framework auch bei der Erstellung der Ausgabe eingesetzt werden. Hierbei wird in jedem Subspace unabhängig von den anderen Unterräumen in der Map-Funktion eine Tiefensuche ausgeführt, um zusammenhängende, dichte Einheiten zu finden. Für die so gefundenen Subspace Cluster können anschließend ebenfalls unabhängig voneinander in einer Map-Funktion des Frameworks die minimale Beschreibung durch Regionen berechnet werden.

4

EXPERIMENTELLE ANALYSE

Im folgenden Kapitel wird der in Kapitel 3 beschriebene Algorithmus experimentell untersucht. Nachdem die für die Analyse zugrunde liegende Hard- und Software beschrieben wird, folgt die Beschreibung und Bewertung der parallelen Implementierungen. Anschließend werden Untersuchungen zu den Auswirkungen der verschiedenen Partitionierungstechniken auf die Qualität der gefundenen Cluster und die Laufzeit des Verfahrens durchgeführt.
Zum Schluß wird auf die in Kapitel 2.1.4 vorgestellten Daten das Verfahren des Subspace Clustering angewendet, um hilfreiche Informationen zu erlangen.

4.1 Setup

Hardware Bei der experimentellen Analyse der Algorithmen wird ein Multiprozessorsystem mit 4 Prozessoren verwendet. Dabei handelt es sich um Intel Xeon X7550 Prozessoren, die jeweils 8 Kerne bei einer Taktrate von 2 GHz besitzen, so dass das gesamte System 32 Recheneinheiten besitzt, die parallel Berechnungen durchführen können.
Durch Hyperthreading kann die Anzahl der für das Betriebssystem virtuell zur Verfügung stehenden Prozessoren verdoppelt werden. Dabei handelt es sich um eine spezielle Technologie von Intel, bei der jeder Rechenkern bestimmte Einheiten innerhalb des Chips dupliziert besitzt, für die Berechnungen aber nur eine ausführende Einheit zur Verfügung steht. Dadurch ist es möglich, dass sobald ein Prozess oder Thread einige Zyklen, beispielsweise aufgrund eines Cache-Miss, warten muss, in dieser Zeit ein anderer Prozess ausgeführt wird und die Prozessorkerne so besser ausgelastet sind.

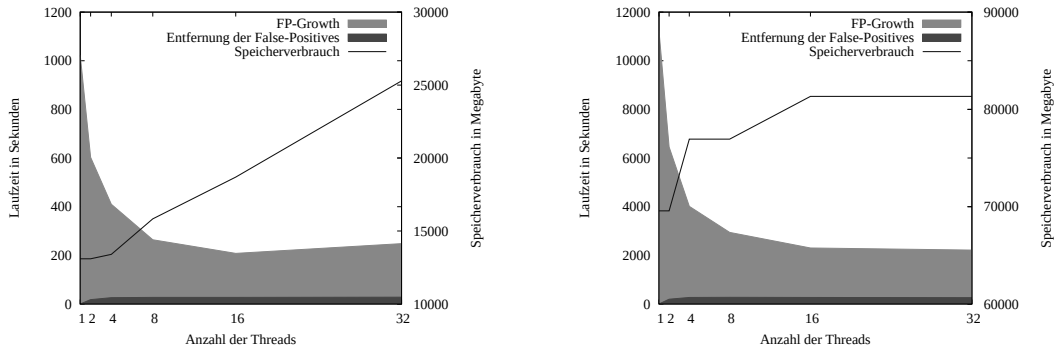
Das System verfügt über 128 GB Arbeitsspeicher, so dass jedem Prozessorkern 4 GB Speicher zur Verfügung stehen.

Software Alle Implementierungen und Experimente basieren auf der Version 5.1.015 von RAPIDMINER. Zur Analyse der verschiedenen Diskretisierungstechniken und der Laufzeit des parallelen Algorithmus werden synthetische Daten verwendet, die von einem Datengenerator in RAPIDMINER erzeugt werden. Dieser erstellt für jedes Attribut zwei normalverteilte Cluster, so dass 2^p viele Cluster erzeugt werden. Wie viele Attribute im Einzelnen erstellt werden, ist bei den jeweiligen Experimenten angegeben. Als Java-Version wird die Referenzimplementierung von Java 6 verwendet.

4.2 Parallele häufige Mengen

Bei der Evaluation des parallelen FPGROWTH werden synthetische Datensätze generiert (siehe Kapitel 4.1). Nach einem Testlauf mit einer seriellen Variante von FPGROWTH werden verschiedene weitere Läufe mit je 2, 4, 8, 16 und 32 Prozessorkernen gestartet.

Beobachtungen Bei einem Datensatz mit 10 000 Beispielen und 1 000 Attributen, die mittels EQUALWIDTH in jeweils 10 Intervalle unterteilt werden, werden für einen Schwellwert von $\xi_{\text{rel}} = 0.1$ die häufigen Mengen gesucht. Die Ergebnisse der Laufzeit und des Speicherbedarfs sind in Abbildung 4.1a dargestellt, wobei nur die reine Laufzeit von FPGROWTH inklusive der notwendigen Aggregation gemessen wird und nicht das Generieren, Diskretisieren und Transformieren in eine Transaktionsdatenbank der Daten.
Bei der Laufzeit ist zu Beginn eine Beschleunigung von 1 023 auf 603 Sekunden zu beobachten (Faktor 1.7). Diese stellt auch die stärkste Beschleunigung dar. Bereits bei Verwendung von 4 Kernen sinkt die Laufzeit auf nur 410 Sekunden (Faktor 2.5). Die geringste Laufzeit wird bei der Verwendung von 16 Kernen erreicht und liegt bei 208 Sekunden (Faktor 4.9). Bei 32 Kernen steigt die Laufzeit wieder auf 248 Sekunden an.
Die Aggregation der Ergebnisse benötigt bei zwei Kernen 18 Sekunden der gemessenen 603 Sekunden und stellt damit 3% der Gesamtlaufzeit dar. Der Wert steigt langsam bis auf 28 Sekunden bei 32 Kernen an und benötigt damit ungefähr 11% der Gesamtlaufzeit.
Während sich der Unterschied im Speicherverbrauch zwischen 1, 2 und 4 Kernen auf ungefähr 300 Megabyte beläuft, wobei ein serieller FPGROWTH 13 097 Megabyte benötigt, steigt dieser bei wachsender Anzahl von verwendeten Kernen stärker an, so dass bei 32 Kernen 25 286 Megabyte benötigt werden.
Ähnliches wurde auch für einen Datensatz von 100 000 Beispielen beobachtet, wobei eine Beschleunigung um den Faktor 5 erreicht wurde (siehe Abbildung 4.1b). Im Vergleich zu dem kleineren Datensatz verschlechterte sich die Laufzeit bei der Transition von 16 auf 32 Kerne nicht . Es kann aber auch keine Verbesserung gemessen werden. Bei einer



(a) 10 000 Beispiele mit 1 000 Attributen

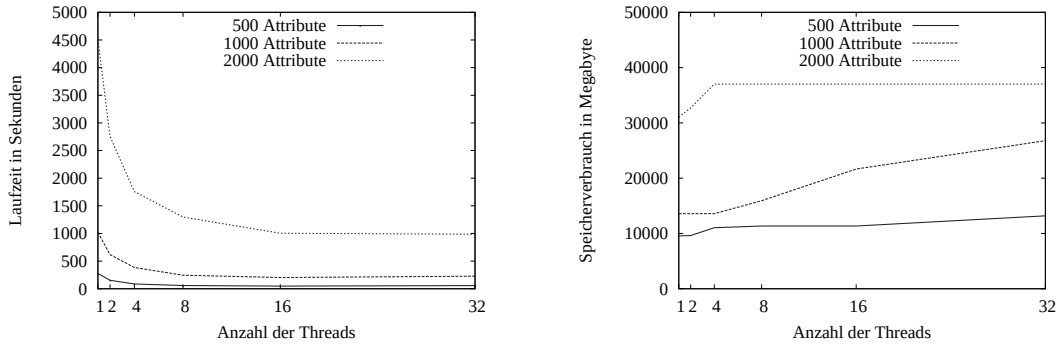
(b) 100 000 Beispiele mit 1 000 Attributen

Abbildung 4.1: Laufzeit von FPGROWTH parallelisiert mit MAPREDUCE auf einen synthetischen Datensatz mit einem Schwellwert von $\xi_{\text{rel}} = 0.1$

Verringerung des Schwellwertes auf $\xi_{\text{rel}} = 0.05$ und 100 000 Beispielen erhöht sich der Faktor der Beschleunigung auf 6.5. Der Anteil der Aggregation der einzelnen häufigen Mengen liegt zwischen 0.3% und 1% der Gesamtlaufzeit.

In Abbildung 4.2 ist der Einfluss der Attributanzahl auf die Laufzeit und den Speicherbedarf abgebildet. Bei gleicher Anzahl von Beispielen und unterschiedlicher Anzahl von Attributen zeigt sich, dass der Beschleunigungsfaktor recht konstant bei 4.5 liegt.

Bewertung Mit der Verwendung des MAPREDUCE-Verfahrens auf FPGROWTH kann mit bis zu 16 Kernen eine Verbesserung bis zum Faktor 6.5 gemessen werden. Eine weitere Erhöhung der Threads bringt jedoch keine weitere Beschleunigung. In den meisten Fällen erhöht sich auch der Speicherbedarf. Dies ist darin begründet, dass für jede Teildatenbank auch ein eigener FPTREE erstellt wird. Die Komprimierung eines FPTREEs basiert darauf, dass häufige Items für jede Transaktion mit wenigen Knoten dargestellt werden. Durch die Verwendung von disjunkten FPTREEs kann dieser Effekt nicht erreicht werden. Gleichzeitig bleibt der Bedarf an Speicher in einigen Fällen aber auch konstant bzw. erhöht sich nur geringfügig (siehe Abbildungen 4.1b und 4.2b), was darauf schließen lässt, dass die Komprimierung stark von den Daten abhängt und eine weitere Aufteilung nicht zwangsläufig zu schlechteren Ergebnissen führt. Insgesamt ist der absolute Speicherbedarf für hochdimensionale Daten jedoch sehr hoch. Schon bei nur einem Rechenkern benötigte das Experiment mit 100 000 Beispielen und 1 000 Attributen ca. 70 GB Arbeitsspeicher, so dass für noch höher dimensionale Daten die Verwendung von Streaming-Algorithmen immer wichtiger wird.



(a) Laufzeit

(b) Speicherbedarf

Abbildung 4.2: Einfluss der Anzahl der Attribute auf die Laufzeit und den Speicherbedarf auf einen synthetischen Datensatz mit 10 000 Beispielen und einem Schwellwert von $\xi_{\text{rel}} = 0.1$

In allen Messungen zeigt sich, dass die Laufzeitverbesserung zwischen der Verwendung von 8 und 16 Kernen nur sehr gering ist. Dafür steigt in den meisten Fällen der Speicherbedarf deutlich an, so dass der beste Kompromiss zwischen Speicherbedarf und Laufzeit bei der Verwendung von 8 Kernen zu erwarten ist.

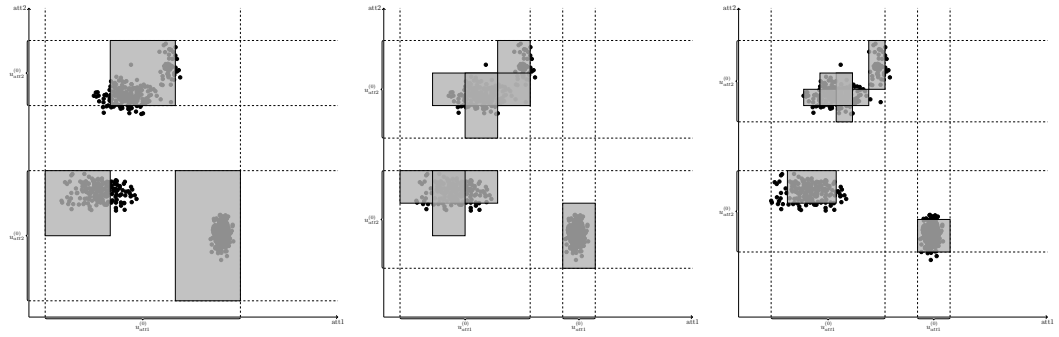
Es zeigt sich auch, dass der Mehraufwand für die Aggregation der Ergebnisse und dem Entfernen der *False Positives* kaum zur Laufzeit beiträgt. Dies ist auch unabhängig von der Anzahl der zu aggregierenden häufigen Mengen. Bei sinkendem Schwellwert schwindet der Anteil deutlich und kann vernachlässigt werden.

4.3 Partitionierung

In Kapitel 3.4 zur Partitionierung des Raumes zeigte sich, dass obwohl die Partitionierung ein wichtiger Schritt im Prozess des Subspace Clustering mit Hilfe von häufigen Mengen ist, viele klassische Diskretisierungstechniken nicht verwendet werden können. Es zeigte sich, dass lediglich drei Methoden vielversprechend sind: EQUALWIDTH, MAFIA und die selbst entwickelte Nachbarschaftsmethode. Im folgenden werden diese drei Methoden auf die Qualität der Cluster und den Einfluss auf die Laufzeit untersucht.

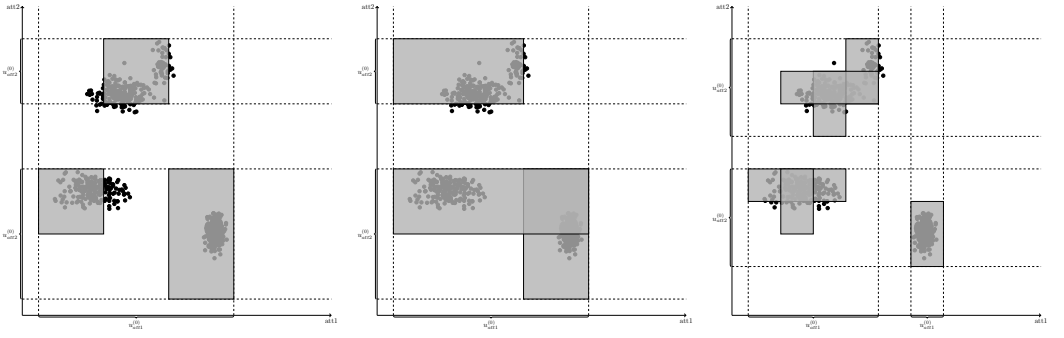
Für die Untersuchungen wird ein Datengenerator verwendet, der verschiedene, normalverteilte Cluster mit unterschiedlichen Varianzen erzeugt, die sich in den verschiedenen Dimensionen überlappen.

Qualität der Ergebnisse Um zunächst eine visuelle Einschätzung der Cluster zu bekommen, werden mit dem Datengenerator mehrere Datensätze mit zufälligen Parametern für die Normalverteilung (Erwartungswert und Varianz) erzeugt. Jeder Datensatz hat nur zwei Attribute mit 4 Clustern, damit diese anhand mehrerer Plots analysiert werden können.

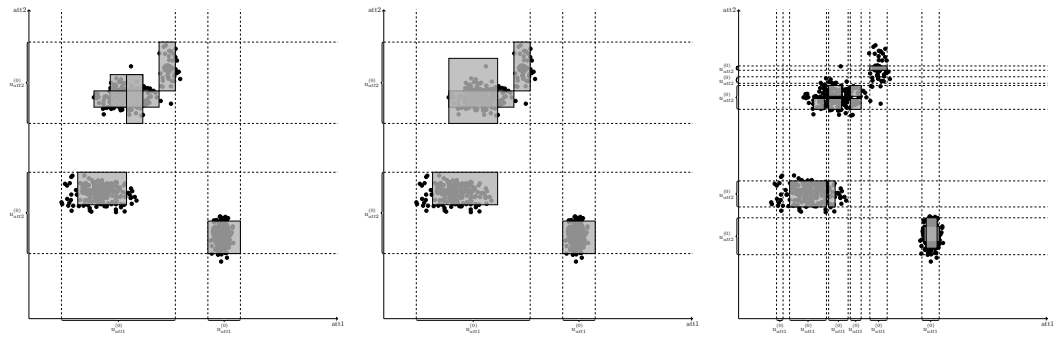


(a) $|I_A| = 5, \xi_{\text{rel}} = 0.1$ **(b)** $|I_A| = 10, \xi_{\text{rel}} = 0.01$ **(c)** $|I_A| = 20, \xi_{\text{rel}} = 0.01$

Abbildung 4.3: Partitionierung mit EQUALWIDTH auf synthetischen Datensatz 1

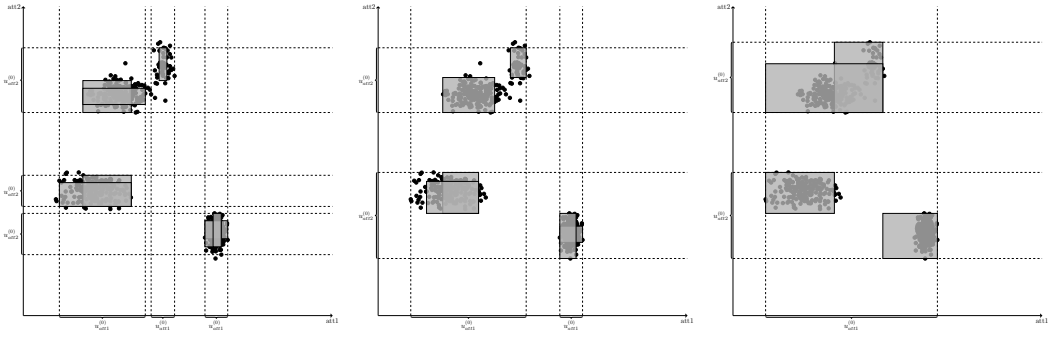


(a) $|I_A| = 5, \beta = 0.3, \xi_{\text{rel}} = 0.1$ **(b)** $|I_A| = 5, \beta = 0.5, \xi_{\text{rel}} = 0.1$ **(c)** $|I_A| = 10, \beta = 0.1, \xi_{\text{rel}} = 0.01$



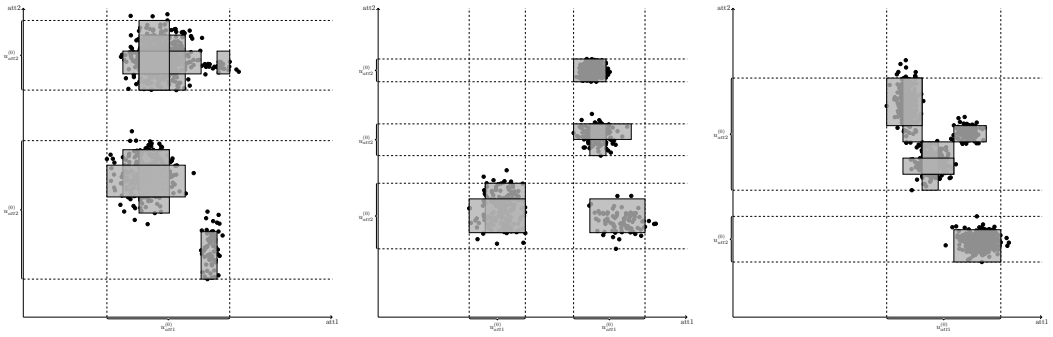
(d) $|I_A| = 20, \beta = 0.1, \xi_{\text{rel}} = 0.01$ **(e)** $|I_A| = 20, \beta = 0.5, \xi_{\text{rel}} = 0.01$ **(f)** $|I_A| = 150, \beta = 0.5, \xi_{\text{rel}} = 0.01$

Abbildung 4.4: Partitionierung mit MAFIA auf synthetischen Datensatz 1



(a) $\epsilon = 0.5, \beta = 0.5, \xi_{\text{rel}} = 0.01$ **(b)** $\epsilon = 1.0, \beta = 0.3, \xi_{\text{rel}} = 0.01$ **(c)** $\epsilon = 3.0, \beta = 0.5, \xi_{\text{rel}} = 0.01$

Abbildung 4.5: Partitionierung mit der Nachbarschaftsmethode auf Datensatz 1



(a) $\epsilon = 1.0, \beta = 0.5, \xi_{\text{rel}} = 0.01$ **(b)** $\epsilon = 1.0, \beta = 0.5, \xi_{\text{rel}} = 0.01$ **(c)** $\epsilon = 1.0, \beta = 0.1, \xi_{\text{rel}} = 0.01$

Abbildung 4.6: Partitionierung mit der Nachbarschaftsmethode auf Datensätzen 2, 3 und 4

Die Abbildungen 4.3 zeigen die EQUALWIDTH-Partitionierung und die damit gefundenen Subspace Cluster. Dabei stellen die grauen Rechtecke in der Abbildung die maximalen Regionen der Subspace Cluster dar. Die Anzahl der Intervalle, die erzeugt werden, ist mit $|I_A|$ angegeben. Eine zu niedrige Anzahl an Intervallen (Abbildung 4.3a) führt zu sehr groben Clustern, die unter anderem viel leeren Raum umfassen (Cluster unten rechts), nur die Hälfte eines Clusters (unten links) oder zwei eng beieinander liegende Cluster zusammenfasst (oben). Erst eine Erhöhung der Anzahl der Intervalle (Abbildung 4.3b) liefert bessere Ergebnisse für die beiden unteren Cluster und erst ab 20 Intervallen (Abbildung 4.3c) werden die beiden oberen Cluster richtig erkannt. Folglich entsteht für ein gutes Clustering auf nur zwei Attributen ein Suchraum von 2^{40} Mengen, was bei vielen Attributen schnell zu Laufzeitproblemen führt. Auffällig ist auch, dass ein simpler Cluster durch vier Regionen beschrieben wird, was zu unnötig komplexen minimalen Beschreibungen führt.

Die Ergebnisse auf dem selben Datensatz für die MAFIA-Methode zeigen die Abbildungen 4.4. Hier muss ebenfalls eine anfängliche EQUALWIDTH-Partitionierung gewählt wer-

den. Bei zu wenig Intervallen zeigen sich die selben Probleme (siehe 4.4a). Hinzu kommt, dass eine ungünstige Wahl des Parameters β zu weniger Clustern und einem schlechteren Clustering führt (siehe 4.4b). Erst eine Vergrößerung der Anzahl der Intervalle führt zu einem Clustering, wie in Abbildung 4.4c bzw. 4.4d. Den Einfluss von β zeigen die Abbildungen 4.4d und 4.4e. Ein kleines β führt zu fein beschriebenen Clustern, die unter anderem viele Regionen beinhalten (oben links in 4.4d) und ein großes β zu sehr groben Gruppen, was dazu führt, dass nahe liegende Cluster zusammengefasst werden.
Auffällig ist die Ähnlichkeit der entstandenen Cluster zwischen der EQUALWIDTH- und der MAFIA-Methode, die durch die Festlegung der anfänglichen Intervalle bei MAFIA entsteht. Dies macht die Wahl von der Anzahl der Intervalle zu einem wichtigen und schwer zu wählenden Parameter. Viele kleine Intervalle führen dazu, dass die Unterschiede in der Häufigkeitsverteilung stark schwanken (siehe auch Kapitel 3.4.2.1), so dass sehr viele Intervalle erstellt werden, die dazu führen, dass ein Cluster in viele kleine Cluster aufgeteilt wird. Dies ist beispielsweise in Abbildung 4.4f zu beobachten.
Zu nicht identischen Clustern führt die Nachbarschaftsmethode, wie in Abbildung 4.5 zu sehen. Statt der Einteilung in Intervallen, wird die Umgebung jedes Beispiels nach Änderung in der Häufigkeitsverteilung untersucht. Als Parameter für die Größe der Nachbarschaft wird ϵ gewählt und entspricht in etwa der minimalen Größe eines Cluster für ein Attribut. Ein niedriges ϵ von 0.5 (Abbildung 4.5a) führt zu einer feinen Einteilung und durch eine Erhöhung des Wertes entstehen breitere Cluster (siehe Abbildung 4.5). Ein Wert von $\epsilon = 3.0$ entspricht in diesem Datenbeispiel in etwa der Einteilung in 6 Intervalle, ist also mit Abbildung 4.4a zu vergleichen. Obwohl β dort niedriger ist, sind die so entstandenen Cluster deutlich ungenauer. Wenn β ebenfalls auf 0.5 gesetzt wird, erstellt MAFIA aus allen Beispielen ein großes Cluster.
Als guter Richtwert für die Festsetzung von ϵ erweist sich ein Wert von 1.0. Abbildung 4.4*b* zeigt eine gute Beschreibung der 4 erzeugten Cluster. Auch für weitere Datensätze erzeugt die Nachbarschaftsmethode für diesen Wert gute Beschreibungen der Cluster (siehe Abbildung 4.6).
Für die qualitativ vergleichbaren Ergebnisse, wie zum Beispiel in Abbildung 4.3c, 4.4d und 4.5b, werden 40 (EQUALWIDTH), 33 (MAFIA) und 24 (Nachbarschaftsmethode) Intervalle erzeugt.
Die Tabellen 4.1 stellen die berechneten Jaccard- und Rand-Indizes (siehe Kapitel 2.3.5) für die unterschiedlichen Verfahren auf den Datensatz 1 (siehe Abbildungen 4.3, 4.4 und 4.5) mit dem Schwellwert $\xi_{\text{rel}} = 0.01$ dar. Dabei werden auch die 1-dimensionalen Subspaces mit einbezogen und für alle Räume der Durchschnitt für die jeweilige Parameterkombination berechnet.
Dabei zeigt sich, dass die beste Parameterkombination bei der Nachbarschaftsmethode bessere Ergebnisse erzeugt, als die besten Parameter bei der EQUALWIDTH- oder MAFIA-Methode. Aber auch bei anderen Parametern sind die Werte bei der Nachbarschafts-

$\lvert I_A \rvert$	5	10	20	30
	0.400	**0.879**	0.871	0.863

(a) EQUALWIDTH – Rand-Index ∅

$\lvert I_A \rvert$	5	10	20	30
	0.396	**0.786**	0.766	0.744

(b) EQUALWIDTH – Jaccard-Index ∅

β \ $\lvert I_A \rvert$	5	10	20	30
0.1	0.400	0.879	0.871	0.866
0.3	0.400	0.880	0.873	0.870
0.5	0.399	0.880	0.873	0.872
0.7	0.394	**0.880**	0.878	0.869

(c) MAFIA – Rand-Index ∅

β \ $\lvert I_A \rvert$	5	10	20	30
0.1	0.396	0.786	0.766	0.751
0.3	0.396	0.788	0.772	0.765
0.5	0.396	0.789	0.772	0.771
0.7	0.394	**0.790**	0.784	0.763

(d) MAFIA – Jaccard-Index ∅

β \ ϵ	0.5	1.0	1.5	2.0
0.1	0.934	0.870	0.877	0.879
0.3	0.931	0.873	0.877	0.879
0.5	**0.951**	0.872	0.877	0.879
0.7	0.861	0.871	0.877	0.876

(e) Nachbarschaft – Rand-Index ∅

β \ ϵ	0.5	1.0	1.5	2.0
0.1	0.842	0.764	0.783	0.787
0.3	0.845	0.772	0.783	0.787
0.5	**0.885**	0.771	0.783	0.787
0.7	0.749	0.769	0.784	0.780

(f) Nachbarschaft – Jaccard-Index ∅

Tabelle 4.1: Durchschnittliche Jaccard- und Rand-Indizes der Partitionierungstechniken auf dem Datensatz 1

methode im Durchschnitt besser. Weitere Experimente mit den Datensätzen 2, 3 und 4 zeigen ähnliche Ergebnisse (siehe Anhang A).

4.4 Analyse von Microarray-Daten

In der Literatur zum Subspace Clustering wird häufig die Analyse von Microarray-Daten (siehe auch Kapitel 2.1) als ein Anwendungsbeispiel hervorgehoben. Im Übersichtsartikel [46] wird beschrieben, dass Biologen an bestimmten Mustern in den Daten interessiert sind. Subspace Clustering kann dabei auf zwei verschiedenen Wegen versuchen Muster in den Daten zu finden, je nachdem was als Attribut und was als Beispiel angesehen wird.

4.4.1 Gruppierung von Genen

Die Expression von verschiedenen Genen wird durch die Microarray-Technologie auf verschiedenen Gewebeproben untersucht. Für ein bestimmtes Gen kann also sein Expressionslevel für eine bestimmte Probe gemessen werden. Nun kann die Datenmatrix so erstellt werden, dass jedes Gen einem Beispiel entspricht und die Gewebeproben die Attribute darstellen. Somit wäre ein Subspace Cluster eine Menge von Genen, die sich für eine Teilmenge der Attribute ähnlich verhalten, d.h. eine ähnliche Expression vorweisen. Nach [46] haben Gene je nach der zellularen Umgebung unterschiedliche Funktionen und sind deswegen unterschiedlich ausgeprägt. Werden die getesteten Proben aus unterschiedlichen Umgebungen oder unter unterschiedlichen Laborbedingungen oder Zeitpunkten entnommen, können diese Subspace Cluster einen Hinweis auf die Funktionen von bestimmten Genen liefern.
Anders ausgedrückt: Eine Menge von Genen verhält sich unter bestimmten Bedingungen (oder in unterschiedlichen Gewebetypen) ähnlich, so dass angenommen werden kann, dass sie für die Funktionsweise unter diesen Bedingungen relevant sind, während andere Gruppen von Genen sich unter anderen Bedingungen gleich verhalten.
In der Literatur wird das Clustering bzw. Subspace Clustering von Microarray-Daten oftmals in diesem Kontext durchgeführt (siehe zum Beispiel [46, 41, 22]). Die bei dieser Arbeit vorhandenen Daten (siehe Kapitel 2.1.4) repräsentieren jedoch nicht Gewebeproben, die unter unterschiedlichen experimentellen Bedingungen entnommen wurden, sondern vergleichbares Gewebe von unterschiedlichen Patienten, die an der selben Art von Krebs erkrankt sind (vergleiche Kapitel 2.1.2). Damit hätte ein gefundener Subspace Cluster folgende Bedeutung.

Definition 4.4.1. Sei $\boldsymbol{X}$ ein Datensatz der Kardinalität $|\boldsymbol{X}| = 17\,882$ im Raum $\mathbb{A}^{113} = A_1 \times \ldots \times A_{113} \subseteq [0, 15]^{113}$ mit den Attributen $\boldsymbol{A} = \{A_1, \ldots, A_{113}\}$. Dabei repräsentiert jedes Beispiel $\vec{x}^{(i)} \in \boldsymbol{X}$ ein Gen, ein Attribut $A_j \in \boldsymbol{A}$ einen Patienten und $x_j^{(i)}$ ist die Expression des Gens für das Merkmal A_j. Jeder gefundene Subspace Cluster $\mathbf{C}_k = (\boldsymbol{X}_k, \boldsymbol{S}_k)$ mit $\boldsymbol{X}_k \subseteq \boldsymbol{X}$ und $\boldsymbol{S}_k \subseteq \boldsymbol{A}$ entspricht einer Menge von Genen $\boldsymbol{X}_k$, die für eine Menge von Patienten $\boldsymbol{S}_k$ ähnlich ausgeprägt sind.

Nun gibt es bei Menschen eine genetische Diversität, die für die individuelle Vielfalt sorgt. Genetische Fingerabdrücke nutzen beispielsweise den Umstand, dass das DNA-Profil eines jeden Menschen einzigartig ist. Bemerkenswert ist jedoch, dass dieser Unterschiede nur einen kleinen Teil der Gene ausmacht. In [26] wird der Umstand, dass 99% der Gene sich ähnlich verhalten, verwendet, um Gendaten so stark zu komprimieren, dass man sie als E-Mail Anhang verschicken kann.
Tatsächlich zeigt eine Berechnung der Korrelationskoeffizienten, dass jeder Patient stark mit jeden anderen Patienten korreliert. Der Koeffizient liegt bei jedem Paar zwischen

0.9 und 1. Das bedeutet, dass die meisten Gene für alle Patienten sehr ähnlich ausgeprägt sind. Folglich wird als Ergebnis eines Subspace Clusterings nach Definition 4.4.1 für jeden Unterraum eine einzige Gruppe erwartet.
Um zu untersuchen, ob nur eine Gruppe in jedem Subspace gefunden wird, werden zunächst die 1-dimensionalen Cluster untersucht. Dabei ist die Annahme, dass, falls es kein Merkmal gibt, für das mehr als ein Cluster gefunden wird, dann auch in den höherdimensionalen Räumen nicht mehr als ein Cluster gefunden wird.

Proposition 4.4.2. *Sei $\boldsymbol{X}$ ein Datensatz im Raum $\mathbb{A}^p = A_1 \times \ldots \times A_p \subseteq \mathbb{R}^p$ und $\mathcal{C} = \{(\boldsymbol{X}_1, \{A_1\}), \ldots, (\boldsymbol{X}_p, \{A_p\})\}$ die Menge aller 1-dimensionalen Subspace Cluster mit $\boldsymbol{X}_i \subseteq \boldsymbol{X}$. Dann gibt es keine zwei unterschiedlichen höherdimensionalen Subspace Cluster $\mathbf{C}_v = (\boldsymbol{X}_v, \boldsymbol{S}_v)$ und $\mathbf{C}_w = (\boldsymbol{X}_w, \boldsymbol{S}_w)$ mit $X_v \subseteq \boldsymbol{X}$ bzw. $\boldsymbol{X}_w \subseteq \boldsymbol{X}$ und $|\boldsymbol{S}_v| > 1$ bzw. $|\boldsymbol{S}_w| > 1$, so dass $\boldsymbol{S}_v = \boldsymbol{S}_w$.*

Beweis. Wenn $\{(\boldsymbol{X}_1, A_1), \ldots, (\boldsymbol{X}_p, A_p)\}$ alle 1-dimensionalen Subspace Cluster sind, dann gibt es für jedes Attribut A_i nur ein Intervall $[l_i, r_i)$, also eine Einheit $u^{(i)} = \{[l_i, r_i)\}$, so dass $s(\{[l_i, r_i)\}) > \tau$. Durch die Monotonieeigenschaft der dichten Einheiten (siehe Proposition 3.2.12) kann jede k-dimensionale dichte Einheit $u^{(j)}$ nur aus den 1-dimensionalen dichten Einheiten erstellt werden, so dass $u^{(j)} \subseteq \{[l_1, r_1), \ldots, [l_p, r_p)\}$. Da es für jedes Attribut A_i nur genau ein Intervall $[l_i, r_i)$ gibt, ist jede Teilmenge von $\{[l_1, r_1), \ldots, [l_p, r_p)\}$ in genau einem der 2^p Unterräumen von $\mathbb{A}^p$ und damit gibt es in jedem Subspace $\mathbb{S}^k$ genau einen Kandidaten, der dicht sein kann. Folglich gibt es in jedem Unterraum höchstens einen Subspace Cluster. □

Um dies zu überprüfen, wird ein Experiment gestartet, bei dem das Verfahren zum Finden von häufigen Mengen nur Mengen mit einem Item ausgibt. Dadurch können im weiteren Verlauf nur 1-dimensionale Subspace Cluster erzeugt werden. Das Ergebnis wird daraufhin überprüft, ob es in jedem Unterraum genau einen Cluster gibt. Als Partitionierungsverfahren wird die Nachbarschaftsmethode mit dem Parameter $\epsilon = 1$ gewählt. Nach [20, S. 82] sind zwei Gene gleich exprimiert, wenn ihre Differenz kleiner < 1 ist (diese Arbeit behandelt den selben Datensatz). Da ϵ sozusagen die Mindestintervallgröße darstellt, ist dies eine sinnvolle Parameterwahl. Als Mindestschwellwert für eine dichte Einheit wird $\tau = 0.1$ gewählt
Bei den Experimenten zeigt sich, dass für den Parameter $\beta \in \{0.3, 0.5, 0.7\}$ für jedes Merkmal nur ein Cluster existiert. Somit wird nach Proposition 4.4.2 für jede Teilmenge der Patienten nur eine Gruppe von Genen gefunden. In Abbildung 4.7 sind exemplarisch 2-dimensionale Plots von für 5 zufällig gewählte Patienten zu sehen. Es ist deutlich zu erkennen, dass die Gene eine Gruppe bilden, die für jeden der abgebildeten Subspaces eine sehr ähnliche Form aufweist. Die weiteren 2-dimensionalen Plots zeigen ein ähnliches Verhalten, was die hohe Korrelation auch vermuten lässt.
An der Abbildung 4.7 ist auch zu erkennen, dass sich sehr kompakte und dichte Gruppen bilden, die nur wenige Ausreißer beinhalten. Somit sind die erzeugten Intervalle sehr

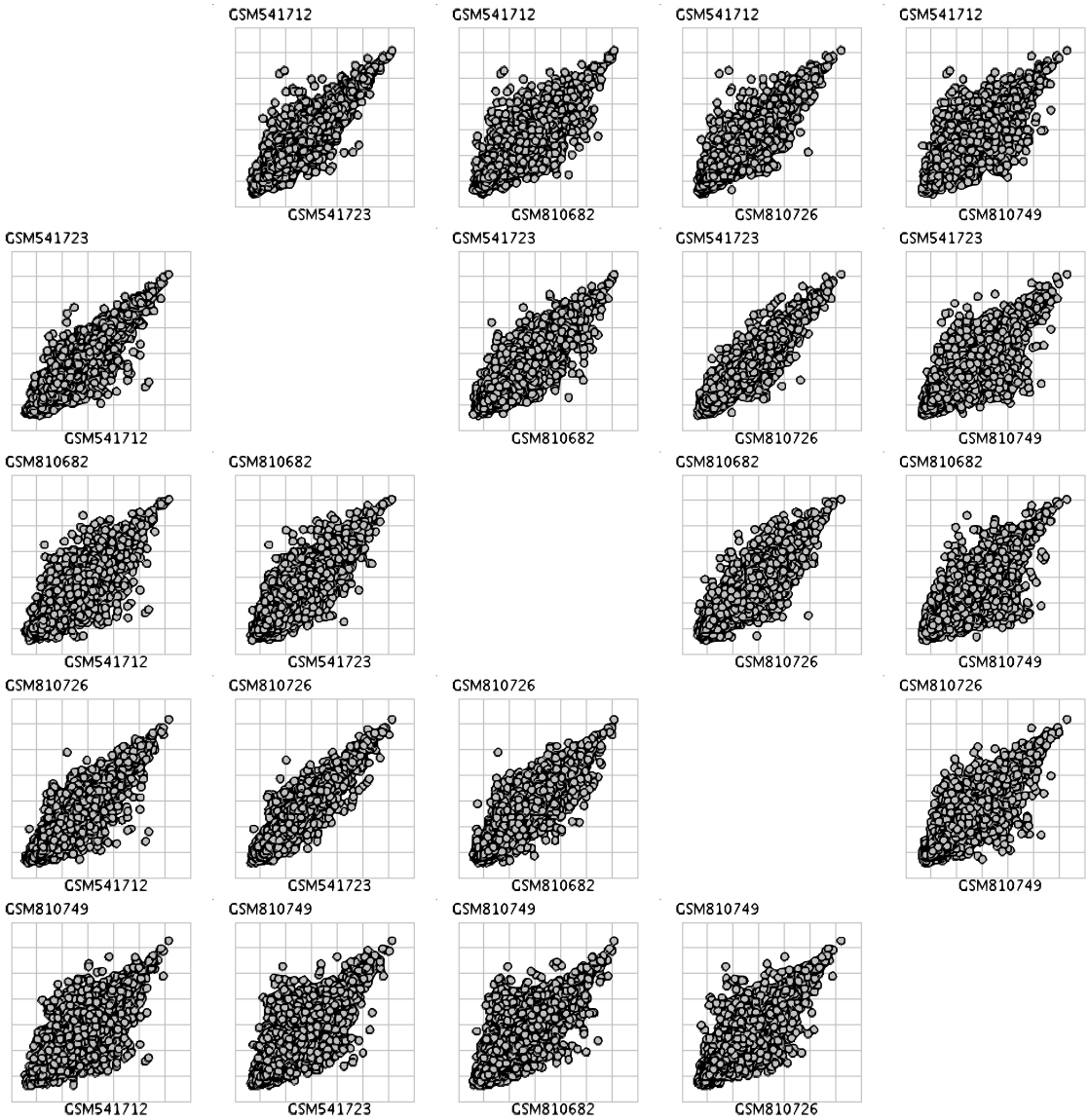


Abbildung 4.7: Alle Gene bilden eine Gruppe im 2-dimensionalen Raum für 5 zufällig gewählte Patienten.

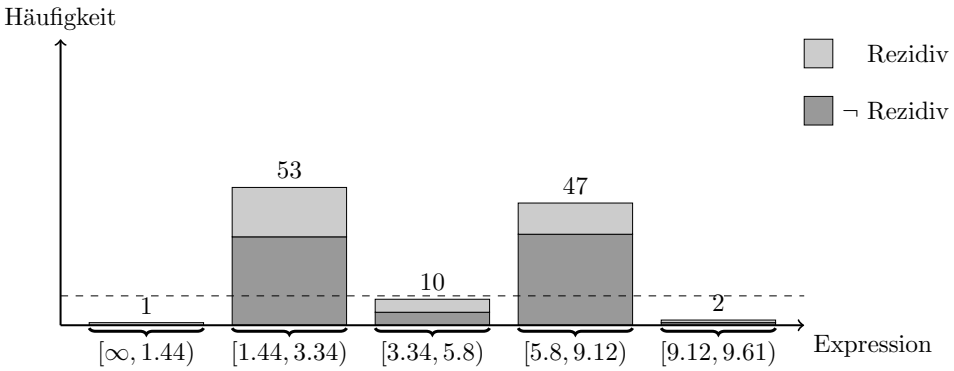
dicht und führen zu sehr dichten, höherdimensionalen Clustern. Das Ergebnis eines Subspace Clustering ist dann selbst bei höheren Schwellwerten ein Cluster je Subspace und das in einem großen Teil der 2^{113} Subspaces. Dies führt also zu einem Verfahren, das sehr hohe Anforderung an Laufzeit und Speicher hat und keine hilfreichen Ergebnisse liefert, außer, dass Menschen sich genetisch gesehen sehr ähnlich sind.

4.4.2 Gruppierung von Gewebeproben

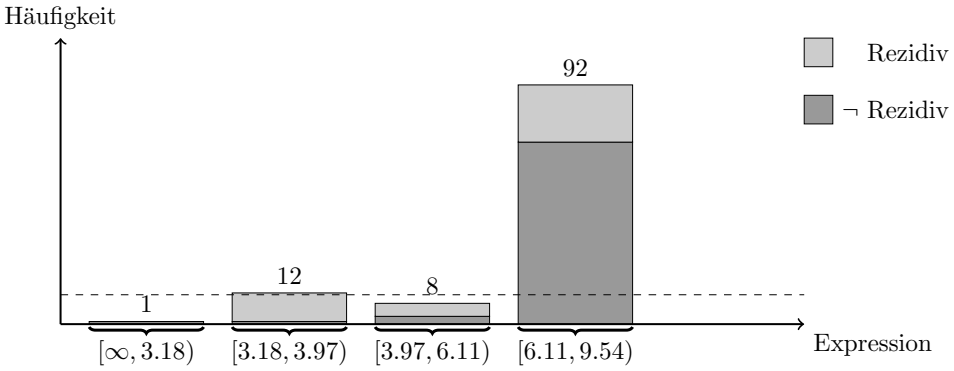
Eine weitere Möglichkeit der Analyse entsteht durch das Transponieren der Datenmatrix. Damit würde jede Probe einem Beispiel entsprechen, das für unterschiedliche Gene, also den Attributen, unterschiedliche Expressionswerte aufweist. Ein gefundener Subspace Cluster wäre bei den vorliegenden Daten damit eine Gruppe von Patienten, die sich bezüglich bestimmter Gene sehr ähnlich ist.

Definition 4.4.3. Sei $\boldsymbol{X}$ ein Datensatz der Kardinalität $|\boldsymbol{X}| = 113$ im Raum $\mathbb{A}^{17\,882} = A_1 \times \ldots \times A_{17\,882} \subseteq [0, 15]^{17\,882}$ mit den Attributen $\boldsymbol{A} = \{A_1, \ldots, A_{17\,882}\}$. Dabei repräsentiert jedes Beispiel $\vec{x}^{(i)} \in \boldsymbol{X}$ einen Patienten, ein Attribut $A_j \in \boldsymbol{A}$ ein Gen und $x_j^{(i)}$ ist die Expression des Gens A_j von einem Patienten $x^{(i)}$. Jeder gefundene Subspace Cluster $\mathbf{C}_k = (\boldsymbol{X}_k, \boldsymbol{S}_k)$ mit $\boldsymbol{X}_k \subseteq \boldsymbol{X}$ und $\boldsymbol{S}_k \subseteq \boldsymbol{A}$ entspricht einer Gruppe von Patienten $\boldsymbol{X}_k$, die sich bezüglich bestimmter Gene $\boldsymbol{S}_k$ ähnlich sind.

Hervorzuheben ist, dass dieser Fall zumindest algorithmisch interessanter ist. Es handelt sich dabei um einen 17 882-dimensionalen Datenraum bei nur 113 Beispielen, so dass die Probleme von hochdimensionalen Daten, die in Kapitel 3.1.1 besprochen wurden, hier relevant sind.
Es ist auch zu erwarten, dass die Gene nicht alle miteinander korrelieren. Letztendlich ist eine unterschiedliche Expression zwischen verschiedenen Genen für die Kodierung zuständig. Eine Berechnung auf einer zufälligen Teilmenge bestätigt dies. Weitere Analysen zu der Korrelation der Gene auf diesen Datensatz sind in [20] zu finden.
Auch bei diesem Vorgehen ist es interessant, eine Teilmenge der Gene zu finden, bei der sich mehr als eine Gruppe von Patienten bilden lassen. Eine Gruppe, die alle Patienten enthält, erzeugt keinen Informationsgewinn. Deswegen werden auch hier Experimente durchgeführt, die nach Proposition 4.4.2 zeigen sollen, ob mehrere Cluster in einem Subspace möglich sind. Dabei wird die Nachbarschaftsmethode mit denselben Parametern zur Partitionierung gewählt.
Es zeigt sich, dass für den größten Teil der Gene weiterhin nur eine Gruppe gefunden wird. Dennoch existieren Gene, die mehr als einen Cluster beinhalten. Bei einem hohen Wert von $\beta = 0.7$, der dazu führt, dass relativ hohe Häufigkeitsunterschiede zur Bildung von Intervallen verwendet werden, zeigen sich 2 von 17 882 Genen mit jeweils zwei Clustern. Abbildung 4.8 zeigt zwei Histogramme mit der verwendeten Partitionierung für diese beiden Gene. In 4.8a werden zwei etwa gleich große Gruppen deutlich getrennt, wohingegen in 4.8b eine Gruppe nur sehr wenige Elemente enthält und fast nicht mehr dicht ist.
Auffällig ist jedoch, dass der Anteil eines Rezidivs in der kleinen Gruppe sehr hoch ist. Während nur 38 aus der Grundgesamtheit aller untersuchten 113 Patienten einen Rückfall erleiden (ca. 33%), sind dies in der Gruppe von 12 Patienten, deren Gen G30 deutlich



(a) Gen G14



(b) Gen G30

Abbildung 4.8: Histogramme von Gen G14 und G30, die für $\beta = 0.7$ zwei Cluster bilden

niedriger exprimiert ist, noch 11, was einen Anteil von 92% entspricht. Es wurde also ein Gen gefunden, bei dem bezüglich des Expressionslevels sich die Patienten in zwei Gruppen unterteilen lassen, wobei eine Gruppe eine deutlich höheres Risiko für einen Rückfall der Erkrankung hat. Dieses Wissen kann nun dazu verwendet werden, Hochrisikopatienten zu ermitteln, aber auch, um weitere Forschung an den Genen zu betreiben, die solch eine Gruppierung ermöglichen.

Eine Verringerung von β führt zu einer größeren Anzahl an Merkmalen mit zwei Clustern (für $\beta = 0.5$ sind es 17 und für $\beta = 0.3$ gleich 34). Ein kleineres β, wie beispielsweise 0.1, führt nicht mehr zu weiteren Clustern, sondern zu mehreren Intervallen, die jedoch weiterhin einen Cluster bilden.

Diese Ergebnisse bedeuten, dass für über 17 800 Gene nur eine dichte Einheit erzeugt wird. In Tabelle 4.2 ist die Anzahl der Merkmale angegeben, für die nur ein Cluster gefunden wird. Weiterhin interessant ist die durchschnittliche prozentuale Abdeckung der einzelnen Cluster, also der Anteil der Patienten, die in diesem Cluster enthalten sind. Diese Abdeckung ist für alle diese Gruppen sehr hoch und liegt im Durchschnitt über 90%, was direkte Auswirkung auf das Clustering hat.

	Mehr als ein Cluster	Genau ein Cluster	
β	Anzahl	Anzahl	Abdeckung (∅)
0.7	2	17 880	96.0%
0.5	17	17 865	95.4%
0.3	34	17 848	95.0%

Tabelle 4.2: Anzahl der Merkmale, in der ein bzw. mehr als ein Cluster gefunden werden. Für die einzelnen Cluster ist zusätzlich die durchschnittliche, prozentuale Abdeckung der Cluster angegeben.

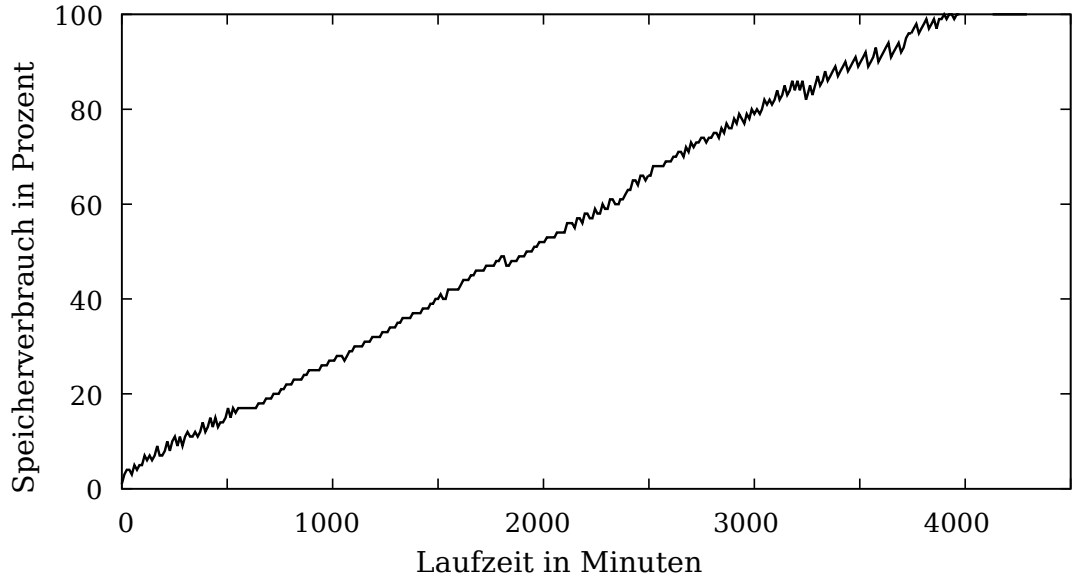


Abbildung 4.9: Speicherverbrauch in Prozent (von 128 GB Arbeitsspeicher) beim Subspace Clustering von allen 17 882 Genen, die nur ein Cluster enthalten, bei $\beta = 0.3$

Für die über 17 000 sehr dichten, einzelnen Cluster entsteht ein ähnliches Problem, wie bei der Gruppierung von Genen (siehe Kapitel 4.4.1). Sie bilden untereinander ebenfalls höherdimensionale Mengen, die aufgrund der hohen Dichte zu sehr vielen, sehr großen häufigen Mengen führen. Damit benötigt das Verfahren zum Finden von häufigen Mengen eine hohe Laufzeit bei sehr großem Speicherbedarf. Abbildung 4.9 zeigt die Laufzeit und den Speicherbedarf des Subspace Clusterings auf dem vollständigen Datensatz. Bei einem Schwellwert von $\xi_{\text{rel}} = 0.1$ bricht das Verfahren nach über drei Tagen Laufzeit aufgrund mangelnden Speichers ab. Die Erhöhung des Schwellwertes dagegen würde dazu führen, dass potentiell interessante Cluster, wie in Abbildung 4.8b, nicht mehr gefunden werden, da sie wegen des niedrigen Supports nicht ausgegeben werden.
Folglich ist eine Verwendung aller Attribute dieses Datensatzes beim Subspace Clustering aufgrund der vielen einzelnen und dichten Cluster nicht durchführbar. Die folgenden Experimente beschränken sich deswegen auf die 34 Attribute, die bei der Verwendung von $\beta = 0.3$ mehr als einen eindimensionalen Cluster je Attribut erzeugt haben. Der Datensatz wird also auf die 34 Attribute projiziert und das Subspace Clustering gestartet. Das Ergebnis dieses Experiments sind 7 078 verschiedene Subspace Cluster in 6 765 verschiedenen Unterräumen (aus insgesamt $2^{34} = 17\,179\,869\,184$ Unterräumen). Aus der Relation zwischen gefundenen Clustern und Subspaces kann eine durchschnittliche Anzahl von ca. 1.05 Cluster pro Subspace berechnet werden. In 299 von 6 765 Unterräumen existieren mehrere Cluster (insgesamt 612). Die höchste Dimensionalität eines Subspace Clusters ist 9 und es wurden bis zu 4 Cluster für einen (2-dimensionalen) Subspace gefunden. Im Folgenden werden die gefundenen Cluster behandelt. Dabei werden mehrere Aspekte betrachtet, um die interessante Gruppen von Patienten herauszufiltern.

Unterräume mit zwei Clustern Für jeden Patienten existiert ein binominales Attribut „Rezidiv“, das für jeden Patienten angibt, ob es einen Rückfall nach einer Behandlung des Neuroblastoms gab. Dieses Attribut kann als Label verwendet werden, um mit externen Indizes zu ermitteln, ob es ein gutes Clustering in einem Subspace bezüglich dieses Labels gibt.
Hierfür werden alle Unterräume untersucht für die genau zwei Cluster gefunden wurden, da dies die Anzahl der unterschiedlichen Klassen für das Label ist. Für diese Cluster wird der Jaccard- bzw. Rand-Index berechnet. Hohe Werte bei diesen Maßen deuten daraufhin, dass eine Gruppierung gefunden wurde, die dem Label der Daten entspricht. Die verwendeten Gene, die zu diesem Clustering führen, ermöglichen also eine gute Einteilung der Patienten bezüglich ihres Krankheitsverlaufes.
Die höchsten erreichten Indizes sind 0.56 (Jaccard) und 0.66 (Rand). Generell werden die besten Werte in 1-dimensionalen Unterräumen gefunden, lassen sich also anhand eines Gens gut hinsichtlich des Jaccard- bzw. Rand-Indexes gruppieren. In Tabelle 4.3 sind die zehn besten Gene und die dazugehörigen Cluster aufgelistet. Für jedes Gen existiert ein

Gen	Jaccard	Rand	Niedrig exprimierter Cluster				Hoch exprimierter Cluster			
				Rezidiv				Rezidiv		
			Interval	Nein	Ja	Ja (∅)	Interval	Nein	Ja	Ja (∅)
G24	0.52	0.64	[6.00, 8.27)	67	18	21%	[9.34, 10.73)	1	12	**92%**
G10	0.51	0.64	[3.14, 4.27)	4	15	**79%**	[5.10, 8.07)	65	17	21%
G01	0.56	0.65	[5.14, 6.62)	3	10	**77%**	[7.56, 11.13)	71	20	22%
G28	0.51	0.65	[4.22, 5.65)	6	15	**71%**	[6.64, 9.32)	65	15	19%
G31	0.55	0.65	[4.64, 6.07)	2	14	**88%**	[7.07, 9.70)	70	20	22%
G13	0.54	0.65	[4.32, 6.35)	69	19	21%	[7.45, 8.45)	1	12	**92%**
G32	0.53	0.66	[2.69, 3.64)	4	12	**75%**	[4.64, 8.04)	67	16	19%
G30	0.49	0.62	[3.18, 3.98)	1	11	**92%**	[6.71, 9.54)	65	18	22%
G03	0.42	0.60	[3.61, 4.96)	5	19	**79%**	[5.88, 7.81)	56	14	20%
G05	0.46	0.60	[2.14, 4.06)	4	16	**80%**	[5.10, 8.71)	61	19	24%

Tabelle 4.3: Die besten Indizes für zwei Cluster

Cluster, bei dem der Anteil der Patienten mit Rezidiv deutlich über dem Anteil von ca. 33% für den gesamten Datensatz liegt. Dabei ist der Anteil besonders für die Gene G24, G31, G13 und G30 sehr hoch.

Bei Betrachtung des Expressionslevels ist weiterhin auffällig, dass der hohe Anteil von Rezidiven überwiegend bei dem niedriger exprimierten Cluster auftritt. Nur in zwei von zehn Fällen ist das höher exprimierte Cluster auffällig. Dies bedeutet, dass für Patienten, die einen Rückfall erleiden, diese Gene weniger aktiv sind, als bei einem anderen Krankheitsverlauf.

Abweichende Anzahl an Rezidiven Unabhängig von externen Indizes werden nun die Cluster gesucht, die einen hohen, abweichenden Anteil von Patienten mit bzw. ohne einem Rückfall beinhalten. Wie bereits erwähnt, ist das Verhältnis im gesamten Datensatz 38 (Rezidiv) zu 75 (kein Rezidiv), was einen Anteil von ca. 33% entspricht. Gesucht werden nun alle Cluster, in denen deutlich mehr (> 60%) oder sehr wenige (<5%) Rückfälle beinhaltet sind. Diese Cluster werden im Folgenden *abweichende Subspace Cluster* genannt. Insgesamt existieren 633 abweichende Cluster in 623 Subspaces.

Von diesen 623 Unterräumen gibt es insgesamt acht, in denen genau zwei abweichende Cluster vorkommen und einen Unterraum, der drei abweichende Cluster beinhaltet. In Tabelle 4.4 sind die minimalen Beschreibungen dieser Cluster aufgelistet. Interessant ist dabei der Subspace $\{G12, G03\}$, bei dem in beiden Clustern nur jeweils ein Patient einen abweichenden Krankheitsverlauf hat. Beide Cluster beinhalten jedoch nur 34% aller Pati-

Subspace	Minimale Beschreibung	Rezidiv			Abd.[1] (∅)	
		Nein	Ja	Ja (∅)		
$\{G21, G03\}$	$(5.24 \leq G21 < 6.20) \wedge (6.83 \leq G03 < 7.82)$	17	0	0.00	15%	26%
	$(1.67 \leq G21 < 2.65) \wedge (3.61 \leq G03 < 4.96)$	3	9	0.75	11%	
$\underline{\{G12, G03\}}$	$(4.36 \leq G12 < 5.92) \wedge (6.83 \leq G03 < 7.82)$	24	1	0.04	22%	$\underline{34\%}$
	$(2.45 \leq G12 < 3.42) \wedge (3.61 \leq G03 < 4.96)$	1	13	0.93	12%	
$\{G12, G01\}$	$(4.36 \leq G12 < 5.92) \wedge (9.56 \leq G01 < 10.56)$	22	0	0.00	19%	30%
	$(2.45 \leq G12 < 3.42) \wedge (5.14 \leq G01 < 6.63)$	3	9	0.75	11%	
$\{G32, G28\}$	$(5.53 \leq G32 < 7.05) \wedge (7.64 \leq G28 < 8.63)$	25	1	0.04	23%	33%
	$(2.70 \leq G32 < 3.65) \wedge (4.22 \leq G28 < 5.65)$	4	7	0.64	10%	
$\{G22, G28\}$	$(1.17 \leq G22 < 2.23) \wedge (4.22 \leq G28 < 5.65)$	2	12	0.86	12%	25%
	$(5.18 \leq G22 < 6.44) \wedge (7.64 \leq G28 < 8.63)$	15	0	0.00	13%	
$\{G10, G28\}$	$(3.14 \leq G10 < 4.27) \wedge (4.22 \leq G28 < 5.65)$	3	10	0.77	12%	32%
	$(6.10 \leq G10 < 7.08) \wedge (7.64 \leq G28 < 8.63)$	22	1	0.04	20%	
$\{G12, G26\}$	$(4.36 \leq G12 < 5.92) \wedge (6.53 \leq G26 < 7.53)$	26	1	0.04	24%	37%
	$(2.45 \leq G12 < 3.42) \wedge (5.54 \leq G26 < 6.53)$	4	11	0.73	13%	
$\{G12, G28\}$	$(4.36 \leq G12 < 5.92) \wedge (7.64 \leq G28 < 8.63)$	23	1	0.04	21%	37%
	$(2.45 \leq G12 < 3.42) \wedge (4.22 \leq G28 < 5.65)$	4	14	0.78	16%	
	$(4.36 \leq G12 < 5.92) \wedge (5.49 \leq G29 < 6.27)$	12	0	0.00	11%	
$\underline{\{G12, G29\}}$	$(2.45 \leq G12 < 3.42) \wedge (2.18 \leq G29 < 3.17)$	4	14	0.78	16%	$\underline{46\%}$
	$(4.36 \leq G12 < 5.92) \wedge (2.18 \leq G29 < 3.17)$	20	1	0.05	19%	

Tabelle 4.4: Subspaces mit mehreren abweichenden Clustern ([1] Abdeckung)

Subspace	Minimale Beschreibung	Rezidiv			Abd.[1] (∅)
		Nein	Ja	Ja (∅)	
$\underline{\{G12\}}$	$(2.45 \leq G12 < 3.42)$	14	24	$\underline{0.63}$	34%
$\{G26, G03, G24\}$	$(6.53 \leq G26 < 7.53) \wedge (6.83 \leq G03 < 7.82)$ $\wedge (6.00 \leq G24 < 8.27)$	31	1	0.03	28%
{G26, G13, G04}	$(6.53 \leq G26 < 7.53) \wedge (4.32 \leq G13 < 6.35)$ $\wedge (6.83 \leq G03 < 7.82)$	30	1	0.03	27%
{G33, G28}	$(4.90 \leq G33 < 6.82) \wedge (7.64 \leq G28 < 9.32)$	36	1	0.03	33%

Tabelle 4.5: Abweichende Cluster mit hoher Abdeckung ([1] Abdeckung)

enten. Je größer die Abdeckung ist, desto aussagekräftiger ist der Subspace und die darin enthaltenen Cluster. Die Ergebnisse in Tabelle 4.3 zeigen jedoch, dass es zumindest keine vollständige Abdeckung von zwei Clustern gibt, da der Jaccard- und Rand-Index hierfür den Wert 1 erreichen würde.

Die höchste Abdeckung erreichen die Gene G12 und G29, für die fast die Hälfte der Patienten einem der drei abweichenden Cluster zugewiesen wird. Zwei der Cluster beinhalten bis auf eine Ausnahme nur Patienten ohne Rezidiv, während im Falle von niedrigem Expressionslevel beider Gene ein rezidiver Krankheitsverlauf beobachtet werden kann.

Subspace	Minimale Beschreibung	Rezidiv			Abd.[1] (∅)
		Nein	Ja	Ja (∅)	
$\{G18, G23,$ $G17, G29$ $G21\}$	$(2.63 \leq G18 < 3.56) \wedge (2.99 \leq G23 < 4.22) \wedge$ $(1.94 \leq G17 < 2.92) \wedge (2.18 \leq G29 < 3.17) \wedge$ $(1.67 \leq G21 < 2.65)$	8	13	0.62	19%
$\{G22, G18,$ $G17, G29$ $G21\}$	$(1.17 \leq G22 < 2.23) \wedge (2.63 \leq G18 < 3.56) \wedge$ $(1.94 \leq G17 < 2.92) \wedge (2.18 \leq G29 < 3.17)$ $(1.67 \leq G21 < 2.65)$	3	11	0.79	12%
$\{G18, G08,$ $G23, G17$ $G29\}$	$(2.63 \leq G18 < 3.56) \wedge (2.22 \leq G08 < 3.19) \wedge$ $(2.99 \leq G23 < 4.22) \wedge (1.94 \leq G17 < 2.92)$ $(2.18 \leq G29 < 3.17)$	5	10	0.67	13%
$\{G18, G08,$ $G23, G17$ $G21\}$	$(2.63 \leq G18 < 3.56) \wedge (2.22 \leq G08 < 3.19) \wedge$ $(2.99 \leq G23 < 4.22) \wedge (1.94 \leq G17 < 2.92)$ $(1.67 \leq G21 < 2.65)$	5	9	0.64	12%
$\{G18, G08,$ $G23, G29$ $G21\}$	$(2.63 \leq G18 < 3.56) \wedge (2.22 \leq G08 < 3.19) \wedge$ $(2.99 \leq G23 < 4.22) \wedge (2.18 \leq G29 < 3.17)$ $(1.67 \leq G21 < 2.65)$	5	9	0.64	12%
$\{G18, G12,$ $G17, G29$ $G21\}$	$(2.63 \leq G18 < 3.56) \wedge (2.45 \leq G12 < 3.42) \wedge$ $(1.94 \leq G17 < 2.92) \wedge (2.18 \leq G29 < 3.17)$ $(1.67 \leq G21 < 2.65)$	3	12	0.80	13%
$\{G08, G23,$ $G17, G29$ $G21\}$	$(2.22 \leq G08 < 3.19) \wedge (2.99 \leq G23 < 4.22) \wedge$ $(1.94 \leq G17 < 2.92) \wedge (2.18 \leq G29 < 3.17)$ $(1.67 \leq G21 < 2.65)$	5	10	0.67	13%
$\{G18, G08,$ $G23, G17$ $G29, G21\}$	$(2.63 \leq G18 < 3.56) \wedge (2.22 \leq G08 < 3.19) \wedge$ $(2.99 \leq G23 < 4.22) \wedge (1.94 \leq G17 < 2.92)$ $(2.18 \leq G29 < 3.17) \wedge (1.67 \leq G21 < 2.65)$	5	9	0.64	12%

Tabelle 4.6: Abweichende Cluster mit hoher Dimensionalität ([1] Abdeckung)

Einzelne abweichende Cluster mit einer hohen Abdeckung sind in Tabelle 4.5 abgebildet. Als einziges Gen hat G12 eine hohe Abdeckung von Rezidiven (bei niedrigem Expressionslevel), während die anderen Cluster überwiegend ereignisfreie Behandlungen abdecken.

Weiterhin gibt es noch 61 höherdimensionale Unterräume mit je einem abweichenden Cluster in 5 und 6 Dimensionen. Davon beinhalten 50 Cluster maximal 13 Patienten, für die alle kein Rückfall vermerkt ist. Die restlichen Gruppen haben bis zu 21 Patienten. Tabelle 4.6 zeigt diese Cluster. Die wichtigsten Gene hierbei sind $\{G18, G23, G17, G29, G21\}$, die in fast allen Clustern Verwendung finden, während G08, G22 und G12 nur in vier bzw. einem Cluster vorkommen.

Auffällig ist überdies, dass alle Cluster überdurchschnittlich viele Rezidive beinhalten und die Expressionslevel für alle Gene der Cluster sehr niedrig sind. Dies deckt sich mit den Beobachtungen in Tabelle 4.3, 4.4 und 4.5.

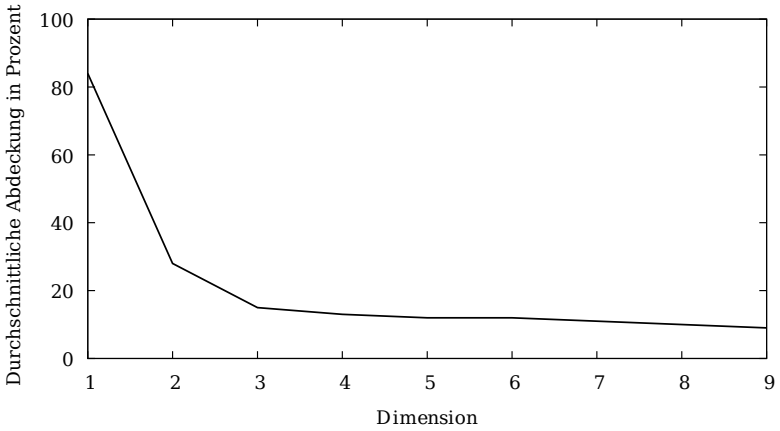


Abbildung 4.10: Durchschnittliche Abdeckung pro Dimension

Subspaces mit hoher Abdeckung Im nächsten Schritt wird untersucht, in welchen Subspaces Cluster gefunden werden, die die Patienten möglichst vollständig in Gruppen aufteilen. Für eine vollständige Partitionierung muss eine hundertprozentige Abdeckung erreicht werden. Abbildung 4.10 zeigt die durchschnittliche Abdeckung der Subspaces pro Dimension. Es ist zu erkennen, dass schon bei 2-dimensionalen Unterräumen die Abdeckung der Patienten auf 30% fällt und diese für die noch höheren Dimensionen 10% bis 15% beträgt.
Bei genauerer Untersuchung stellt sich heraus, dass es nur zwei 2-dimensionale Unterräume gibt, deren Cluster über 80% der Patienten abdecken. Tabelle 4.7 listet diese beiden Unterräume und 11 weitere 1-dimensionale Subspaces auf, die eine hohe Abdeckung haben. Auffällig ist hier, dass 6 der 11 Unterräume einen Cluster beinhalten, der einen hohen Anteil an rezidiven Krankheitsverläufen hat. Dies und die hohe Abdeckung führt dazu, dass diese Subspace Cluster gute Werte beim Jaccard- und Rand-Index erreichen (siehe Tabelle 4.3).

Silhouetten-Koeffizient In Kapitel 2.3.5 zur Evaluation von Clustern wurde der Silhouetten-Koeffizient als interner Index vorgestellt. Dabei wird die Silhouette, die zwischen -1 und +1 liegt, für ein Beispiel eines Cluster berechnet und der Silhouetten-Koeffizient ist der Durchschnitt der Silhouetten der im Cluster liegenden Beispiele. Je höher der Koeffizient eines Clusters, desto besser definiert und von den restlichen Clustern abgegrenzt ist dieser. Der Cluster mit dem höchsten Wert von 0.92 ist der 4-dimensionaler Subspace Cluster

$$(1.44 \leq G14 < 2.36) \wedge (2.63 \leq G18 < 3.56) \wedge (1.94 \leq G17 < 2.92) \wedge (1.67 \leq G21 < 2.65)$$

und beinhaltet 38 Patienten.

Subspace	Minimale Beschreibung	Rezidiv			Abd.[1] (∅)	
		Nein	Ja	Ja (∅)		
$\{G21, G29\}$	$(1.67 \leq G21 < 2.65) \wedge (2.18 \leq G29 < 3.17)$	33	18	0.35	45%	80%
	$(4.15 \leq G29 < 6.27) \wedge (5.24 \leq G21 < 6.20)$ $\vee$ $(4.15 \leq G29 < 5.49) \wedge (4.25 \leq G21 < 6.20)$	29	11	0.28	35%	
$\{G31\}$	$(4.64 \leq G31 < 6.07)$	2	14	0.88	14%	94%
	$(7.07 \leq G31 < 9.70)$	70	20	0.22	80%	
$\{G05\}$	$(2.14 \leq G05 < 4.06)$	4	16	0.80	18%	89%
	$(5.10 \leq G05 < 8.71)$	61	19	0.24	71%	
$\{G09\}$	$(3.13 \leq G09 < 4.85)$	37	17	0.31	48%	92%
	$(5.80 \leq G09 < 7.54)$	30	20	0.40	44%	
$\{G01\}$	$(5.14 \leq G01 < 6.63)$	3	10	0.77	12%	93%
	$(7.59 \leq G01 < 11.13)$	71	20	0.22	81%	
$\{G29\}$	$(2.18 \leq G29 < 3.17)$	35	19	0.35	48%	91%
	$(4.15 \leq G29 < 6.27)$	35	14	0.29	43%	
$\{G28\}$	$(4.22 \leq G28 < 5.65)$	6	15	0.71	19%	90%
	$(6.64 \leq G28 < 9.32)$	65	15	0.19	71%	
$\{G26\}$	$(5.54 \leq G26 < 8.35)$	65	29	0.31	83%	94%
	$(3.73 \leq G26 < 4.72)$	5	7	0.58	11%	
$\{G19\}$	$(4.63 \leq G19 < 6.73)$	32	12	0.27	39%	93%
	$(1.73 \leq G19 < 3.65)$	37	24	0.39	54%	
$\{G13\}$	$(7.45 \leq G13 < 8.45)$	1	12	0.92	12%	90%
	$(4.32 \leq G13 < 6.35)$	69	19	0.22	78%	
$\{G15\}$	$(2.99 \leq G15 < 4.67)$	26	6	0.19	28%	93%
	$(5.64 \leq G15 < 10.20)$	45	29	0.39	65%	
$\{G10\}$	$(3.14 \leq G10 < 4.27)$	4	15	0.79	17%	90%
	$(5.10 \leq G10 < 8.07)$	65	17	0.21	73%	

Tabelle 4.7: Subspace Cluster mit hoher Abdeckung ([1] Abdeckung)

Subspace	Minimale Beschreibung	Rezidiv			Sil.[1] (∅)
		Nein	Ja	Ja (∅)	
$\{G22, G29\}$	$(1.17 \leq G22 < 2.23) \wedge (2.18 \leq G29 < 3.17)$	3	12	0.80	0.89%
$\{G22, G17\}$	$(1.17 \leq G22 < 2.23) \wedge (1.94 \leq G17 < 2.92)$	3	12	0.80	0.90%
$\{G22, G18\}$	$(1.17 \leq G22 < 2.23) \wedge (2.63 \leq G18 < 3.56)$	3	11	0.79	0.90%

Tabelle 4.8: Abweichende Subspace Cluster mit hohem Silhouetten-Koeffizient ([1] Silhouetten-Koeffizient)

Überdies haben auch alle 2- und 3-dimensionalen Projektionen dieses Clusters einen hohen Silhouettenwert von über 0.9. Der Anteil an Rezidiven ist mit 0.32 jedoch nicht unter- oder überdurchschnittlich. In Tabelle 4.8 sind die abweichenden Subspace Cluster mit hohem Koeffizienten (0.89 - 0.90) aufgelistet. Auffällig ist das Gen G22, dass in allen drei Clustern vorkommt und sehr niedrig exprimiert ist.

Zusammenfassung der Ergebnisse Aus dem gesamten Suchraum von 2^{34} Unterräumen werden Subspace Cluster überwiegend in Unterräumen mit niedriger Dimensionalität gefunden. Bei der Berechnung von Jaccard- und Rand-Indizes zeigen sich nur 1-dimensionale Cluster mit hohen Ergebnissen. Der Silhouetten-Koeffizient dagegen findet auch einen 4-dimensionalen Cluster, der zwar auch viele Patienten umfasst, aber keine besondere Verteilung von Patienten mit bzw. ohne Rezidiv vorweist.
Insgesamt zeigt sich, dass Subspace Cluster, die besonders viele Rezidive beinhalten, oftmals nur im 1- oder 2-dimensionalen Subspaces zu finden sind. Bei Betrachtung der Expressionslevel der Gene dieser Unterräume ist zu erkennen, dass diese im niedrig exprimierten Bereich zu finden sind. Für einige wenig aktive Gene lassen sich also Gruppen von Patienten bilden, die vermehrt einen Rückfall der Krankheit erleiden.

5

Zusammenfassung und Fazit

Aufgrund der steigenden Dimensionalität von Datensätzen in aktuellen Anwendungsgebieten, werden Verfahren, die den Fluch der Dimensionalität berücksichtigen, immer wichtiger. Im Bereich der Clusterverfahren werden beispielsweise Algorithmen zum Subspace-Clustering verwendet, um Unterräume für gute Cluster zu finden.
Die vorliegende Arbeit befasste sich mit einer speziellen Variante dieser Verfahren, die eine starke Verbindung zu einem weiteren Themengebiet des maschinellen Lernens aufweist – dem Finden von häufigen Mengen. In Kapitel 3 wurde diese Verbindung hervorgehoben und mit CLIQUE ein Algorithmus vorgestellt, der einen starken Bezug zu APRIORI, einen der ersten Algorithmen zum Finden von häufigen Mengen, hat. Dabei wurde das Verfahren in mehrere Komponenten abstrahiert und unterteilt, die näher untersucht wurden.
Dabei stellte sich heraus, dass die Partionierung des Datenraumes ein wichtiger Schritt im gesamten Prozess ist, der sowohl Laufzeit, als auch Qualität der Ergebnisse beeinflusst. Mehrere Verfahren der Diskretisierung wurden untersucht und die Nachteile erläutert. Die Analyse ergab ein neues Verfahren, das diese Nachteile verbessert, um genauere Ergebnisse zu liefern. Eine experimentelle Analyse in Kapitel 4.3 zeigte nicht nur eine Verbesserung der erzeugten Subspace-Cluster, sondern auch eine einfachere Wahl der benötigten Parameter. Diese sind intuitiver zu wählen und liefern durchschnittlich bessere Ergebnisse.
Bei der Kapselung in einzelne Schritte wurde überdies das Problem vollständig transformiert, so dass es mit aktuellen und leistungsfähigen Verfahren aus dem gut erforschten Gebiet des Findens von häufigen Mengen gelöst werden konnte, wie z.B. FPGROWTH. Dabei konnte eine im ursprünglichen Verfahren verwendete Heuristik nicht übernom-

men werden. Da durch diese Heuristik aber potentiell wichtige Cluster nicht gefunden werden und sie nur zur Verbesserung der Laufzeit eingeführt wurde, kann auf diese Methode verzichtet werden.
Weiterhin wurde eine parallele Version von FPGROWTH mit Hilfe eines hierfür entwickelten MAPREDUCE-Frameworks für RAPIDMINER verwendet. In der experimentellen Analyse in Kapitel 4.2 konnte damit eine Verbesserung der Laufzeit um den Faktor 6.5 gemessen werden.

Anwendung Das so entstandene Verfahren wurde abschließend zur Microarray-Analyse genutzt. Hierbei zeigten sich jedoch einige Schwächen des Verfahrens. Zwar kann die Methode für hochdimensionale Daten eingesetzt werden, doch werden einige Probleme beim Finden von häufigen Mengen auf dieses Verfahren übertragen. Das ist insbesondere dann der Fall, wenn viele hochdimensionale Subspaces existieren, für die es noch dichte Einheiten gibt. Dies ist bei den Gendaten der Fall, wo ein Großteil der Attribute sehr dichte Cluster besitzt, die zu sehr vielen hochdimensionalen Clustern führen. Dadurch steigt der Speicherbedarf exponentiell an, so dass Experimente auf dem vollständigen Datenraum nicht durchgeführt werden konnten.
Projiziert auf die eindimensonalen Subspaces, für die mehrere Cluster gefunden wurden, konnten mehrere Teilmengen der Gene ermittelt werden, die Patienten gruppieren. Bezüglich des Krankheitsverlaufes nach einer Behandlung zeigten sich Gruppen von Patienten, die überdurchschnittlich oft an einem Rückfall des Neuroblastoms litten. Das bedeutet, dass bestimmte Gene ähnlich exprimiert sind, wenn ein Rückfall vorliegt. In den meisten Fällen waren diese Gene weniger aktiv, als für Gruppen von Patienten, die sowohl rezidive als auch nicht rezidive Krankheitsverläufe vorweisen. Dies kann zur Klassifikation und zur weiteren Forschung an diesen Genen genutzt werden.

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

ALGORITHMENVERZEICHNIS

LITERATURVERZEICHNIS

[1] ACHTERT, ELKE, CHRISTIAN BÖHM, HANS-PETER KRIEGEL, PEER KRÖGER und ARTHUR ZIMEK: *Robust, Complete, and Efficient Correlation Clustering*. In: *Proceedings of the Seventh SIAM International Conference on Data Mining, April 26-28, 2007, Minneapolis, Minnesota, USA*. SIAM, 2007.

[2] AGGARWAL, CHARU C., JOEL L. WOLF, PHILIP S. YU, CECILIA PROCOPIUC und JONG SOO PARK: *Fast algorithms for projected clustering*. In: *Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, SIGMOD '99, Seiten 61–72, New York, NY, USA, 1999. ACM.

[3] AGGARWAL, CHARU C. und PHILIP S. YU: *Finding generalized projected clusters in high dimensional spaces*. In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, SIGMOD '00, Seiten 70–81, New York, NY, USA, 2000. ACM.

[4] AGRAWAL, RAKESH, JOHANNES GEHRKE, DIMITRIOS GUNOPULOS und PRABHAKAR RAGHAVAN: *Automatic subspace clustering of high dimensional data for data mining applications*. In: *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, SIGMOD '98, Seiten 94–105, New York, NY, USA, 1998. ACM.

[5] AGRAWAL, RAKESH, TOMASZ IMIELIŃSKI und ARUN SWAMI: *Mining association rules between sets of items in large databases*. SIGMOD Rec., 22:207–216, June 1993.

[6] AGRAWAL, RAKESH und JOHN C. SHAFER: *Parallel Mining of Association Rules*. IEEE Trans. on Knowl. and Data Eng., 8:962–969, December 1996.

[7] AGRAWAL, RAKESH und RAMAKRISHNAN SRIKANT: *Fast Algorithms for Mining Association Rules in Large Databases*. In: *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, Seiten 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[8] 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), Seiten 483–485, New York, NY, USA, 1967. ACM.

[9] AMIR, AMIHOOD, RONEN FELDMAN und REUVEN KASHI: *A new and versatile method for association generation*. Inf. Syst., 22:333–347, September 1997.

[10] BANERJEE, A und R N DAVE: *Validating clusters using the Hopkins statistic*. 2004 IEEE International Conference on Fuzzy Systems, 1:149–153, 2004.

[11] BELLMAN, R.E.: *Adaptive control processes: a guided tour*. Rand Corporation Research studies. Princeton University Press, 1961.

[12] BEYER, KEVIN S., JONATHAN GOLDSTEIN, RAGHU RAMAKRISHNAN und URI SHAFT: *When Is "Nearest Neighbor" Meaningful?* In: *Proceedings of the 7th International Conference on Database Theory*, ICDT '99, Seiten 217–235, London, UK, 1999. Springer-Verlag.

[13] BISHOP, CHRISTOPHER M.: *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[14] BLUM, AVRIM L. und PAT LANGLEY: *Selection of relevant features and examples in machine learning*. Artif. Intell., 97:245–271, December 1997.

[15] BOHM, CHRISTIAN, KARIN KAILING, HANS-PETER KRIEGEL und PEER KROGER: *Density Connected Clustering with Local Subspace Preferences*. In: *Proceedings of the Fourth IEEE International Conference on Data Mining*, ICDM '04, Seiten 27–34, Washington, DC, USA, 2004. IEEE Computer Society.

[16] BÖHM, CHRISTIAN, KARIN KAILING, PEER KRÖGER und ARTHUR ZIMEK: *Computing Clusters of Correlation Connected objects*. In: *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, SIGMOD '04, Seiten 455–466, New York, NY, USA, 2004. ACM.

[17] BORTHAKUR, DHRUBA: *The Hadoop Distributed File System: Architecture and Design*. The Apache Software Foundation, 2007.

[18] BRODEUR, G M, R C SEEGER, M SCHWAB, H E VARMUS und J M BISHOP: *Amplification of N-myc in untreated human neuroblastomas correlates with advanced disease stage.* Science, 224(4653):1121–4, 1984.

[19] BRODEUR, GARRETT M.: *Neuroblastoma: biological insights into a clinical enigma*. Nature Reviews Cancer, 3(3):203–216, 2003.

[20] BÜTZKEN, MIRIAM: *Die Relevanz bestimmter Exons für die Überlebensprognose beim Neuroblastom*. Diplomarbeit, TU Dortmund, 2009.

[21] CAMPBELL, NEIL A. und JANE B. REECE: *Biology*. Benjamin-Cummings Publishing Company, 2005.

[22] CHANG, YE-IN I., JIUN-RUNG R. CHEN und YUEH-CHI C. TSAI: *Mining subspace clusters from DNA microarray data using large itemset techniques.* Journal of computational biology : a journal of computational molecular cell biology, 16(5):745–768, Mai 2009.

[23] CHEN, DEHAO, CHUNRONG LAI, WEI HU, WENGUANG CHEN, YIMIN ZHANG und WEIMIN ZHENG: *Tree partition based parallel frequent pattern mining on shared memory systems*. In: *Proceedings of the 20th international conference on Parallel and distributed processing*, IPDPS'06, Seiten 313–313, Washington, DC, USA, 2006. IEEE Computer Society.

[24] CHEN, DEHAO, CHUNRONG LAI, WEI HU, WENGUANG CHEN, YIMIN ZHANG und WEIMIN ZHENG: *Tree partition based parallel frequent pattern mining on shared memory systems*. In: *Proceedings of the 20th international conference on Parallel and distributed processing*, IPDPS'06, Seiten 313–313, Washington, DC, USA, 2006. IEEE Computer Society.

[25] CHENG, CHUN-HUNG, ADA WAICHEE FU und YI ZHANG: *Entropy-based subspace clustering for mining numerical data*. In: *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '99, Seiten 84–93, New York, NY, USA, 1999. ACM.

[26] CHRISTLEY, SCOTT, YIMING LU, CHEN LI und XIAOHUI XIE: *Human genomes as email attachments*. Bioinformatics, 25:274–275, January 2009.

[27] CORMEN, THOMAS H., CLIFFORD STEIN, RONALD L. RIVEST und CHARLES E. LEISERSON: *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd Auflage, 2001.

[28] CORTES, CORINNA und VLADIMIR VAPNIK: *Support-Vector Networks*. Mach. Learn., 20:273–297, September 1995.

[29] DEAN, JEFFREY und SANJAY GHEMAWAT: *MapReduce: simplified data processing on large clusters*. In: *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*. USENIX Association, 2004.

[30] DOUGHERTY, JAMES, RON KOHAVI und MEHRAN SAHAMI: *Supervised and Unsupervised Discretization of Continuous Features*. In: *MACHINE LEARNING: PROCEEDINGS OF THE TWELFTH INTERNATIONAL CONFERENCE*, Seiten 194–202. Morgan Kaufmann, 1995.

[31] EVERITT, B., S. LANDAU und M. LEESE: *Cluster analysis*. A Hodder Arnold Publication. Arnold, 2001.

[32] FANG, WENBIN, MIAN LU, XIANGYE XIAO, BINGSHENG HE und QIONG LUO: *Frequent itemset mining on graphics processors*. In: *Proceedings of the Fifth International Workshop on Data Management on New Hardware*, DaMoN '09, Seiten 34–42, New York, NY, USA, 2009. ACM.

[33] FLYNN, MICHAEL J.: *Some computer organizations and their effectiveness*. IEEE Trans. Comput., 21:948–960, September 1972.

[34] GABER, MOHAMED MEDHAT, ARKADY ZASLAVSKY und SHONALI KRISHNASWAMY: *Mining data streams: a review*. SIGMOD Rec., 34:18–26, June 2005.

[35] GAUTIER, LAURENT, LESLIE COPE, BENJAMIN M. BOLSTAD und RAFAEL A. IRIZARRY: *affy—analysis of Affymetrix GeneChip data at the probe level*. Bioinformatics, 20:307–315, February 2004.

[36] HAN, J. und M. KAMBER: *Data mining: concepts and techniques*. The Morgan Kaufmann series in data management systems. Elsevier, 2006.

[37] HAN, JIAWEI, JIAN PEI und YIWEN YIN: *Mining frequent patterns without candidate generation*. In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, SIGMOD '00, Seiten 1–12, New York, NY, USA, 2000. ACM.

[38] HASTIE, T., R. TIBSHIRANI und J.H. FRIEDMAN: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2008.

[39] IOANNIDIS, YANNIS E. und VISWANATH POOSALA: *Balancing histogram optimality and practicality for query result size estimation*. In: *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, SIGMOD '95, Seiten 233–244, New York, NY, USA, 1995. ACM.

[40] IRIZARRY, RAFAEL A., BRIDGET HOBBS, FRANCOIS COLLIN, YASMIN D. BEAZER BARCLAY, KRISTEN J. ANTONELLIS, UWE SCHERF und TERENCE P. SPEED: *Exploration, normalization, and summaries of high density oligonucleotide array probe level data*. Biostatistics, 4(2):249–264, April 2003.

[41] JIANG, DAXIN, CHUN TANG und AIDONG ZHANG: *Cluster Analysis for Gene Expression Data: A Survey*. IEEE Transactions on Knowledge and Data Engineering, 16:1370–1386, November 2004.

[42] JIN, RUOMING, GE YANG und G. AGRAWAL: *Shared memory parallelization of data mining algorithms: techniques, programming interface, and performance*. Knowledge and Data Engineering, IEEE Transactions on, 17(1):71 – 89, jan. 2005.

[43] KAILING, KARIN, HANS P. KRIEGEL und PEER KROGER: *Density-Connected Subspace Clustering for High-Dimensional Data*. In: *Proc. 4th SIAM International Conference on Data Mining*, April 2004.

[44] KOOI, ROBERT PHILIP: *The optimization of queries in relational databases*. Doktorarbeit, Cleveland, OH, USA, 1980. AAI8109596.

[45] KOTSIANTIS, SOTIRIS und DIMITRIS KANELLOPOULOS: *Discretization techniques: A recent survey*. GESTS International Transactions on Computer Science and Engineering, 32:47–58, 2006.

[46] KRIEGEL, HANS-PETER, PEER KRÖGER und ARTHUR ZIMEK: *Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering*. ACM Trans. Knowl. Discov. Data, 3:1:1–1:58, March 2009.

[47] LAMINE M. AOUAD, N-A. LE-KHAC und M-T. KECHADI: *Distributed Frequent Itemsets Mining in Heterogeneous Platforms*. Journal of Engineering, Computing and Architecture, 1, 2007.

[48] LI, HAOYUAN, YI WANG, DONG ZHANG, MING ZHANG und EDWARD Y. CHANG: *PFP: Parallel FP-Growth for Query Recommendation*. In: *Proceedings of the 2008 ACM conference on Recommender systems*, RecSys '08, New York, NY, USA, 2008. ACM.

[49] LINNÉ, C. und J.J. PLANER: *Gattungen der Pflanzen und ihre natürliche Merkmale, nach der Anzahl, Gestalt, Lage und Verhaltniss aller Blumentheile*. Nummer Bd. 2 in *Gattungen der Pflanzen und ihre natürliche Merkmale, nach der Anzahl, Gestalt, Lage und Verhaltniss aller Blumentheile*. Karl Wilhelm Ettinger, 1775.

[50] LIU, GUIMEI, JINYAN LI, KELVIN SIM und LIMSOON WONG: *Distance Based Subspace Clustering with Flexible Dimension Partitioning*. Data Engineering, International Conference on, 0:1250–1254, 2007.

[51] LIU, HUAN, FARHAD HUSSAIN, CHEW LIM TAN und MANORANJAN DASH: *Discretization: An Enabling Technique*. Data Min. Knowl. Discov., 6:393–423, October 2002.

[52] LIU, LI, ERIC LI, YIMIN ZHANG und ZHIZHONG TANG: *Optimization of frequent itemset mining on multiple-core processor*. In: *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, Seiten 1275–1285. VLDB Endowment, 2007.

[53] MACQUEEN, J. B.: *Some Methods for Classification and Analysis of MultiVariate Observations*. In: CAM, L. M. LE und J. NEYMAN (Herausgeber): *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, Band 1, Seiten 281–297. University of California Press, 1967.

[54] MANKU, GURMEET SINGH und RAJEEV MOTWANI: *Approximate frequency counts over data streams*. In: *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB '02, Seiten 346–357. VLDB Endowment, 2002.

[55] MATTSON, TIMOTHY, BEVERLY SANDERS und BERNA MASSINGILL: *Patterns for parallel programming*. Addison-Wesley Professional, 2004.

[56] MIERSWA, INGO, MICHAEL WURST, RALF KLINKENBERG, MARTIN SCHOLZ und TIMM EULER: *YALE: Rapid Prototyping for Complex Data Mining Tasks*. In: UNGAR, LYLE, MARK CRAVEN, DIMITRIOS GUNOPULOS und TINA ELIASSI-RAD (Herausgeber): *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, Seiten 935–940, New York, NY, USA, August 2006. ACM.

[57] MITCHELL, THOMAS M.: *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 Auflage, 1997.

[58] MOORE, GORDON E.: *Cramming more components onto integrated circuits*. Electronics, 38(8), April 1965.

[59] NAGESH, HARSHA, SANJAY GOIL und ALOK CHOUDHARY: *Adaptive grids for clustering massive data sets*. In: *In 1st SIAM International Conference Proceedings on Data Mining*, 2001.

[60] OWEN, SEAN, ROBIN ANIL, TED DUNNING und ELLEN FRIEDMAN: *Mahout in Action*. Manning Publications, 1 Auflage, Januar 2011.

[61] PARSONS, LANCE, EHTESHAM HAQUE und HUAN LIU: *Subspace clustering for high dimensional data: a review*. SIGKDD Explor. Newsl., 6:90–105, June 2004.

[62] POOSALA, VISWANATH: *Histogram-based estimation techniques in database systems*. Doktorarbeit, Madison, WI, USA, 1997. UMI Order No. GAX97-16074.

[63] PRAMUDIONO, IKO und MASARU KITSUREGAWA: *Parallel FP-Growth on PC Cluster*. In: WHANG, KYU-YOUNG, JONGWOO JEON, KYUSEOK SHIM und JAIDEEP SRIVASTAVA (Herausgeber): *Advances in Knowledge Discovery and Data Mining*, Band 2637

der Reihe *Lecture Notes in Computer Science*, Seiten 570–570. Springer Berlin / Heidelberg, 2003.

[64] PREKOPCSÁK, Z., G. MAKRAI, T. HENK und CS. GÁSPÁR-PAPANEK: *Radoop: Analyzing Big Data with RapidMiner and Hadoop*. In: *RCOMM 2011: RapidMiner Community Meeting And Conference*. Rapid-I, 2011.

[65] RAUBER, THOMAS und GUDULA RÜNGER: *Multicore: Parallele Programmierung*. Springer-Verlag, 2008.

[66] RISSANEN, JORMA: *Modeling by Shortest Data Description*. Automatica, 14:465–471, 1978.

[67] SANDER, JÖRG, MARTIN ESTER, HANS-PETER KRIEGEL und XIAOWEI XU: *Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications*. Data Min. Knowl. Discov., 2:169–194, June 1998.

[68] SCARUFFI, PAOLA, ALEXANDER VALENT, ALEXANDER SCHRAMM, KATHY ASTRAHANTSEFF, ANGELIKA EGGERT und GIAN PAOLO TONINI: *Application of microarray-based technology to neuroblastoma.* Cancer Lett, 228(1-2):13–20, 2005.

[69] SCHOWE, BENJAMIN: *Feature Selection for high-dimensional data in RapidMiner*. In: FISCHER, SIMON und INGO MIERSWA (Herausgeber): *Proceedings of the 2nd RapidMiner Community Meeting And Conference (RCOMM 2011)*, Aachen, 2011. Shaker Verlag.

[70] SEEGER, R. C., G. M. BRODEUR, H. SATHER, A. DALTON, S. E. SIEGEL, K. Y. WONG und D. HAMMOND: *Association of multiple copies of the N-myc oncogene with rapid progression of neuroblastomas.* N Engl J Med, 313(18):1111–1116, Oct 1985.

[71] TAN, PANG-NING, MICHAEL STEINBACH und VIPIN KUMAR: *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[72] THEODORIDIS, SERGIOS und KONSTANTINOS KOUTROUMBAS: *Pattern Recognition, Fourth Edition*. Academic Press, 4th Auflage, 2008.

[73] VREEKEN, J. und A ZIMEK: *When Pattern Met Subspace Cluster - A Relationship Story*. In: *Proceedings of the 2nd Workshop on Discovering, Summarizing and Using Multiple Clusterings (MultiClust'11)*, 2011.

[74] WATSON, JAMES D. und FRANCIS H. C. CRICK: *Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid*. Nature, 171(4356):737–738, 1953.

[75] WEIERSTRASS, KARL: *Über die analytische Darstellbarkeit sogenannter willkürlicher Functionen einer reellen Veränderlichen*. Sitzungsberichte der Akademie zu Berlin, 1885.

[76] ZAÏANE, OSMAR R., MOHAMMAD EL-HAJJ und PAUL LU: *Fast Parallel Association Rule Mining without Candidacy Generation*. In: *Proceedings of the 2001 IEEE International Conference on Data Mining*, ICDM '01, Seiten 665–668, Washington, DC, USA, 2001. IEEE Computer Society.

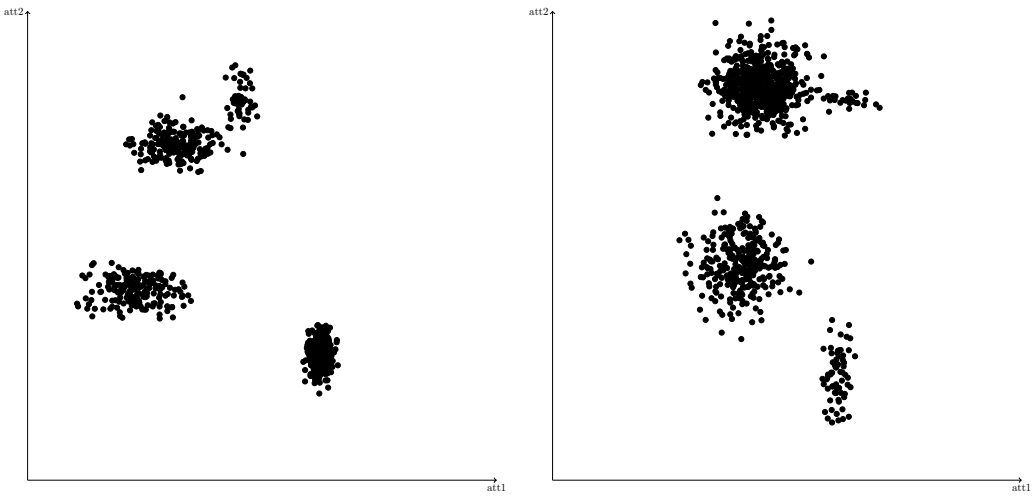
[77] ZAKI, M. J., M. OGIHARA, S. PARTHASARATHY und W. LI: *Parallel data mining for association rules on shared-memory multi-processors*. In: *Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '96, Washington, DC, USA, 1996. IEEE Computer Society.

[78] ZAKI, MOHAMMED J.: *Scalable Algorithms for Association Mining*. IEEE Trans. on Knowl. and Data Eng., 12:372–390, May 2000.

[79] ZHANG, FAN, YAN ZHANG und J. BAKOS: *GPApriori: GPU-Accelerated Frequent Itemset Mining*. In: *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*, Seiten 590 –594, September 2011.

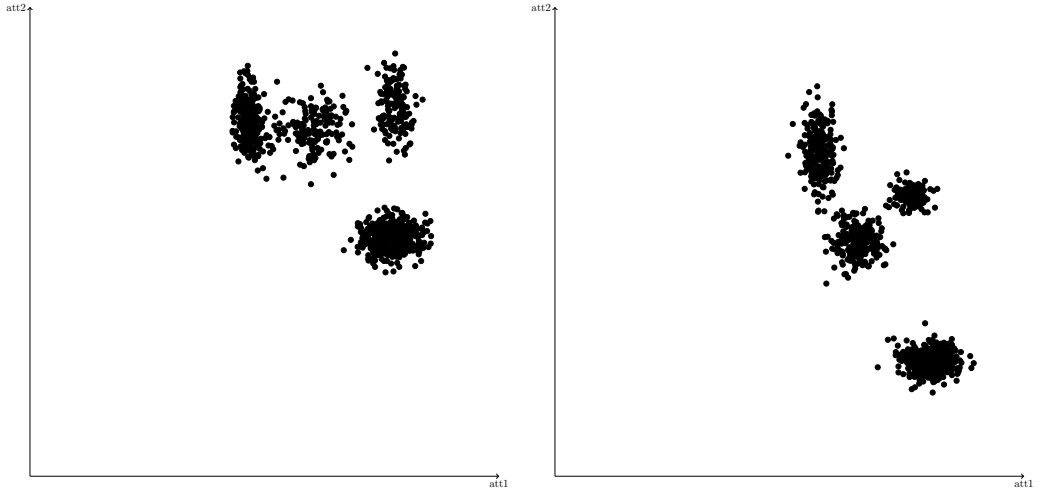
A

PARTITIONIERUNG



(a) Datensatz 1 **(b)** Datensatz 2

Abbildung A.1: Untersuchte Datensätze 1 und 2



(a) Datensatz 3 **(b)** Datensatz 4

Abbildung A.2: Untersuchte Datensätze 3 und 4

$\lvert I_A \rvert$	5	10	20	30
	0.475	**0.786**	0.776	0.764

(a) EQUALWIDTH – Rand-Index ∅

$\lvert I_A \rvert$	5	10	20	30
	0.472	**0.750**	0.727	0.688

(b) EQUALWIDTH – Jaccard-Index ∅

β \ $\lvert I_A \rvert$	5	10	20	30
0.1	0.475	0.786	0.776	0.765
0.3	0.475	0.786	0.782	0.775
0.5	0.475	0.787	0.785	0.784
0.7	0.473	0.777	0.777	**0.791**

(c) MAFIA – Rand-Index ∅

β \ $\lvert I_A \rvert$	5	10	20	30
0.1	0.472	0.750	0.727	0.690
0.3	0.472	0.750	0.740	0.711
0.5	0.472	0.752	0.745	0.734
0.7	0.472	0.735	0.731	**0.756**

(d) MAFIA – Jaccard-Index ∅

β \ ϵ	0.5	1.0	1.5	2.0
0.1	0.771	0.786	0.788	0.783
0.3	0.680	0.797	0.788	0.785
0.5	0.822	0.781	0.788	0.785
0.7	**0.839**	0.780	0.772	0.774

(e) Nachbarschaft – Rand-Index ∅

β \ ϵ	0.5	1.0	1.5	2.0
0.1	0.624	0.745	0.752	0.743
0.3	0.564	0.762	0.751	0.748
0.5	0.742	0.736	0.752	0.749
0.7	**0.779**	0.736	0.726	0.731

(f) Nachbarschaft – Jaccard-Index ∅

Tabelle A.1: Durchschnittliche Jaccard- und Rand-Indizes der Partitionierungstechniken auf dem Datensatz 2

$\lvert I_A \rvert$	5	10	20	30
	0.304	0.467	0.664	**0.679**

(a) EQUALWIDTH – Rand-Index ∅

$\lvert I_A \rvert$	5	10	20	30
	0.302	0.396	0.532	**0.557**

(b) EQUALWIDTH – Jaccard-Index ∅

β \ $\lvert I_A \rvert$	5	10	20	30
0.1	0.304	0.465	0.664	**0.665**
0.3	0.303	0.464	0.664	0.639
0.5	0.303	0.464	0.631	0.636
0.7	0.303	0.466	0.626	0.632

(c) MAFIA – Rand-Index ∅

β \ $\lvert I_A \rvert$	5	10	20	30
0.1	0.302	0.395	**0.532**	0.526
0.3	0.302	0.396	**0.532**	0.487
0.5	0.302	0.396	0.485	0.492
0.7	0.302	0.398	0.486	0.491

(d) MAFIA – Jaccard-Index ∅

β \ ϵ	0.5	1.0	1.5	2.0
0.1	0.618	0.668	0.671	0.307
0.3	**0.824**	0.660	0.633	0.308
0.5	0.816	0.667	0.633	0.305
0.7	0.464	0.628	0.632	0.306

(e) Nachbarschaft – Rand-Index ∅

β \ ϵ	0.5	1.0	1.5	2.0
0.1	0.440	0.545	0.557	0.299
0.3	0.602	0.526	0.496	0.302
0.5	**0.604**	0.544	0.495	0.302
0.7	0.385	0.488	0.495	0.302

(f) Nachbarschaft – Jaccard-Index ∅

Tabelle A.2: Durchschnittliche Jaccard- und Rand-Indizes der Partitionierungstechniken auf dem Datensatz 3

$\lvert I_A \rvert$	5	10	20	30
	0.289	0.455	**0.676**	0.666

(a) EQUALWIDTH – Rand-Index ∅

$\lvert I_A \rvert$	5	10	20	30
	0.289	0.379	**0.570**	0.534

(b) EQUALWIDTH – Jaccard-Index ∅

β \ $\lvert I_A \rvert$	5	10	20	30
0.1	0.289	0.453	**0.679**	0.668
0.3	0.289	0.293	0.611	0.669
0.5	0.289	0.292	0.611	0.676
0.7	0.289	0.292	0.610	0.610

(c) MAFIA – Rand-Index ∅

β \ $\lvert I_A \rvert$	5	10	20	30
0.1	0.289	0.378	**0.579**	0.540
0.3	0.289	0.289	0.459	0.545
0.5	0.289	0.289	0.459	0.568
0.7	0.289	0.289	0.459	0.457

(d) MAFIA – Jaccard-Index ∅

β \ ϵ	0.5	1.0	1.5	2.0
0.1	0.651	**0.678**	0.611	0.609
0.3	0.657	0.643	0.612	0.607
0.5	0.660	0.649	0.611	0.609
0.7	0.447	0.611	0.610	0.290

(e) Nachbarschaft – Rand-Index ∅

β \ ϵ	0.5	1.0	1.5	2.0
0.1	0.489	**0.579**	0.462	0.461
0.3	0.503	0.499	0.465	0.458
0.5	0.518	0.511	0.465	0.461
0.7	0.369	0.463	0.465	0.288

(f) Nachbarschaft – Jaccard-Index ∅

Tabelle A.3: Durchschnittliche Jaccard- und Rand-Indizes der Partitionierungstechniken auf dem Datensatz 4

B

Abkürzungen der Gene

Gen	Gen-Id
G01	3746574
G02	2903219
G03	3797032
G04	3810749
G05	3141589
G06	4048241
G07	4048265
G08	3059393
G09	2350981
G10	2412624
G11	2632778
G12	3452478
G13	2354634
G14	4028512
G15	3954375
G16	3210616
G17	4030063

(a)

Gen	Gen-Id
G18	4030162
G19	4030371
G20	3371114
G21	4031136
G22	2357193
G23	2417362
G24	2470838
G25	2471233
G26	3045570
G27	2505993
G28	2361761
G29	4035017
G30	3266279
G31	3050388
G32	2585476
G33	3026216
G34	3990927

(b)

Tabelle B.1: Abkürzungen der Gene

C

CD-ROM

Hiermit bestätige ich, die vorliegende Diplomarbeit selbständig und nur unter Zuhilfenahme der angegebenen Literatur verfasst zu haben.

Ich bin damit einverstanden, dass Exemplare dieser Arbeit in den Bibliotheken der Universität Dortmund ausgestellt werden.

Dortmund, den 15. März 2012

Marcin Skirzynski