Analyse dichtebasierter Clusteralgorithmen am Beispiel von DBSCAN und MajorClust

Michael Busch
Studienarbeit
WS 2004/2005
Universität Paderborn

22. März 2005

Analyse dichtebasierter Clusteralgorithmen am Beispiel von DBSCAN und MajorClust

Für Marion.

Inhaltsverzeichnis

Motivation

Die Clusteranalyse (kurz: *Clustering*) hat als eine Technik des *Data Mining* das Ziel, in beliebigen Datenmengen Strukturen zu finden und Zusammenhänge aufzudecken. Zu diesem Zweck ist bereits eine Vielzahl an Clusteralgorithmen entwickelt worden, welche sich durch unterschiedliche Vorgehensweisen und Strategien auszeichnen. Aufgrund dieser Strategien lassen sich die Algorithmen in eine Taxonomie verschiedener Paradigmen einordnen.
Der Algorithmus MajorClust ist ein solcher Clusteralgorithmus, welcher der Kern der erfolgreichen Suchmaschine AISearch [17] ist. Obwohl die sehr gute Leistung von MajorClust bekannt ist, wurde die Funktionsweise dieses Algorithmus noch nicht tiefgehend analysiert, weshalb er bislang keiner Kategorie in der Clusteralgorithmen-Taxonomie sicher zugeordnet werden konnte.
Hauptsächliches Ziel dieser Arbeit ist deshalb die Evaluierung von MajorClust im Hinblick auf Funktionsweise und Performance. Da MajorClust Ähnlichkeit mit *dichtebasierten* Verfahren hat, wird er mit dem dichtebasierten Algorithmus DBSCAN verglichen. Dabei liegt der Schwerpunkt insbesondere auf der Beantwortung der folgenden Fragen:

- Wie verhält sich das Clusterergebnis und die Performance von MajorClust im Vergleich zum dichtebasierten Clusteralgorithmus DBSCAN?

- Deuten die Clusterergebnisse darauf hin, dass MajorClust ebenfalls dichtebasiert arbeitet?

MajorClust bietet gegenüber anderen dichtebasierten Clusteralgorithmen den Vorteil, dass er sowohl metrische Daten, als auch Ähnlichkeitsdaten verarbeiten und clustern kann. Da Ähnlichkeitsdaten jedoch z. B. durch die Multidimensionale Skalierung (MDS) in den euklidischen Raum eingebettet

werden können, ist es fraglich, wie praktikabel dieser Vorteil tatsächlich ist. Zusätzlich soll daher die Frage beantwortet werden:

- Wie stark beeinflusst die Einbettung der ursprünglichen Ähnlichkeitsdaten in den euklidischen Raum das Ergebnis der Clusteranalyse?

Gliederung der Arbeit

Kapitel 1 enthält eine allgemeine Einführung in die Zielsetzung und Funktionsweise der Clusteranalyse und stellt praktische Einsatzgebiete vor. In Kapitel 2 werden die Begriffe Dichte und dichtebasiertes Clustering definiert und erläutert. Kapitel 3 und 4 stellen die Algorithmen DBSCAN und MajorClust vor, wobei jeweils auf die Strategie, die Funktionsweise und die Performance eingegangen wird. Kapitel 5 präsentiert eine Testumgebung zur Evaluierung der Algorithmen und erläutert wesentliche Implementierungsdetails. Kapitel 6 stellt Messungen und Ergebnisse zum Vergleich der Algorithmen dar und interpretiert diese. In Kapitel 7 schließlich werden die Ergebnisse dieser Arbeit zusammengefasst und es wird ein Ausblick auf verwandte Forschungsaufgaben gegeben.

Danksagung

Ich möchte mich an dieser Stelle bei all denen bedanken, die diese Arbeit ermöglichten.

Benno Stein danke ich für die Betreuung, die oft durch große Entfernungen erschwert wurde. Meiner Familie danke ich für viel Liebe, Unterstützung und ihr Vertrauen in mich. Zu Dank verpflichtet bin ich ebenso Sven Schade und Dirk Haubenreisser für viele Tipps und gewissenhaftes Korrekturlesen.

Ganz besonders bedanke ich mich bei meiner Freundin Anastasia, die mir liebevollen Ausgleich zur Arbeit gab und mir in stressvollen Zeiten ans Herz legte, dass die Welt nicht nur aus Clusterproblemen besteht.

Erklärung

Hiermit versichere ich, dass diese Studienarbeit eigenständig und ohne fremde Hilfe angefertigt wurde. Ebenso wurden keine anderen, als die angegebenen und zitierten Quellen und Hilfmittel, verwendet.

Michael Busch, Olsberg, den 22. März 2005

Teil I

Theoretische Vorüberlegungen

Kapitel 1

Einführung in die Clusteranalyse

Ein Teilbereich des Forschungsgebiets "Knowledge Discovery in Databases" (KDD) ist das sogenannte Data Mining, welches das Ziel hat, Wissen und Strukturen aus bestimmten Datenmengen zu extrahieren. Dabei wird angenommen, dass sich das Wissen bereits in den Datenmengen befindet; mit Hilfe des Data Minings soll es aufgedeckt werden.

Eine Technik des Data Minings ist die sogenannte Clusteranalyse, auch numerische Taxonomie oder kurz Clustering genannt.

Dieses Kapitel bietet eine Einführung in die Clusteranalyse anhand einer beispielhaften Problemstellung in Abschnitt 1.1. Nach der formalen Definition von Clustering und Cluster in Abschnitt 1.2 und der Vorstellung wichtiger Formate der Problemdaten in Abschnitt 1.3 wird in Abschnitt 1.4 ein erstes Beispiel einer Clusteranalyse gegeben. Nachdem in Abschnitt 1.5 Überlegungen zur Komplexität des Clusterproblems angestellt werden schließt Abschnitt 1.6 mit einer Taxonomie gängiger Clusterstrategien.

1.1 Problemstellung

Der Ausgangspunkt bei der Problemstellung ist eine Menge von Daten (Objekten), welche verschiedene Eigenschaften aufweisen. Ziel der Clusteranalyse ist es, diese Objekte in Gruppen, die sogenannten Cluster, aufzuteilen. Diese Gruppierung soll so gewählt werden, dass Objekte innerhalb desselben

Clusters untereinander möglichst ähnlich sind (Homogenität innerhalb der Cluster), Objekte aus unterschiedlichen Clustern jedoch möglichst unähnlich (Heterogenität bei verschiedenen Clustern).

Die Ähnlichkeit zweier Objekte ergibt sich dabei durch Berücksichtigung der Eigenschaften, die die einzelnen Objekte aufweisen. Wie unterschiedlich diese Eigenschaften sein können, sollen die folgenden Beispiele zeigen.

Problem 1: Umzugkartons

Eine Familie ist im Begriff umzuziehen. Nach dem Aussortieren der nicht mehr benötigten Gegenstände gilt es nun die verbliebenen Gegenstände möglichst zweckmäßig auf Umzugkartons zu verteilen.

Um dieses Problem zu lösen, lassen sich verschiedene Eigenschaften der Objekte (hier: Gegenstände) erfassen:

- **Gewicht** Beispiel: Gewichte nicht größer als Belastbarkeit des Kartons
- **Form** Beispiel: Gegenstände müssen in den Karton passen
- **Art des Gegenstands** Beispiel: Sonntagskleidung nicht zu Werkzeugkiste
- **Zerbrechlichkeit** Beispiel: Hanteln nicht auf Porzellangeschirr
- etc.

Problem 2: Automobilkonstruktion

Heutzutage enthalten moderne Autos mehr als 100 unterschiedliche Elektroniksysteme mit über 2500 Kabelverbindungen. Für eine möglichst kostengünstige Produktion ist daher die Planung einer effektiven Anordnung der einzelnen Elektronikbauteile zur Minimierung der Kabellängen notwendig.

Auch bei diesem Gruppierungsproblem lassen sich bestimmte Eigenschaften der Objekte (hier: Elektronikbauteile) finden:

- **Kommunikationspartner** Beispiel: Radio und CD-Wechsler
- **Ergonomie** Beispiel: Mikrofon der Freisprecheinrichtung auf Mundhöhe des Fahrers
- **Funktionsweise** Beispiel: Klimaanlage in der Nähe vom Motor

- **Ortsgebundenheit** Beispiel: Servolenkung in der Nähe der Lenkanlage
- **Sicherheit** Beispiel: Tank mit Sicherheitsabstand zum Unterboden
- etc.

Wie zu sehen ist, eignet sich die Clusteranalyse für sehr unterschiedliche Einsatzgebiete, bei denen das Grundproblem, die Gruppierung von Objekten zum Finden von Strukturen, gegeben ist.

Vergleich mit der Klassifikation

Im Gegensatz zur Vorgehensweise der Klassifikation, bei der die jeweiligen Gruppen a priori feststehen und die einzelnen Objekte diesen Gruppen zugeordnet werden sollen, sind die Cluster beim Clustering nicht vorher bekannt und werden implizit vom jeweiligen Algorithmus generiert. Dabei gibt es Algorithmen, die als Eingabeparameter die Anzahl der Cluster, die erzeugt werden sollen, benötigen; andere Algorithmen erzeugen eine bezüglich ihrer Strategie optimale Anzahl von Clustern.

1.2 Definitionen

Definition 1 *(Clustering, Cluster) Sei D ein Menge von Objekten. Ein Clustering $\mathcal{C} \subseteq \{C \mid C \subseteq D\}$ von D ist eine Aufteilung von D in Teilmengen, welche die folgenden Bedingungen erfüllen: $\bigcup_{C_i \in C} C_i = D$ und $\forall C_i, C_j \in C : C_i \cap C_{j \neq i} = \emptyset$. Diese Teilmengen C_i werden als Cluster bezeichnet.*

Die Definition fordert, dass ein Clustering eine Aufteilung der Objekte in *disjunkte* Teilmengen darstellt. Zusätzlich wird gefordert, dass jedes Objekt genau einem Cluster zugeordnet sein muss.

Eine Besonderheit stellen Clusteralgorithmen (z. B. DBSCAN) dar, die neben den Clustern noch Rauschen (Noise) identifizieren. Das Rauschen kann in der Definition als eine spezielle Gruppe angesehen werden, die Objekte beinhaltet, welche sinnvollerweise keinem Cluster zugeordnet werden können und somit als Störstellen in den Ausgangsdaten klassifiziert werden.

1.3 Datenformat der Problemdaten

Die Daten eines Clusterproblems können in verschiedenen Formaten vorliegen. In dieser Arbeit werden zwei Formate verwendet: Geometrische Daten und Ähnlichkeitsdaten.

Geometrische Daten

Bei geometrischen Daten stellen die Daten eine Menge X dar, deren Objekte sich im euklidischen Raum befinden.

Definition 2 *Ein euklidischer Raum ist ein k-dimensionaler Raum R^k, in dem die Gesetze der euklidischen Geometrie gelten.*

Um den Abstand zwischen zwei Objekten im euklidischen Raum bestimmen zu können, wird eine Metrik benötigt.

Definition 3 *Sei X eine beliebige Menge. Eine Abbildung $d : X \times X \to R$ heißt Metrik, wenn für beliebige Elemente x, y und z von X die folgenden axiomatischen Bedingungen erfüllt sind:*

- $d(x, x) = 0$
- $d(x, y) = 0 \Rightarrow x = y$ *(Definitheit)*
- $d(x, y) = d(y, x)$ *(Symmetrie)*
- $d(x, y) \leq d(x, z) + d(z, y)$ *(Dreiecksungleichung)*

Die in dieser Arbeit verwendete Metrik ist die euklidische Metrik

$$d(x, y) = \sqrt{\sum_{i=1}^{k}(x_i - y_i)^2},$$

die ein Spezialfall der Minkowski-Metrik

$$d(x, y) = \left(\sum_{i=1}^{k}(x_i - y_i)^r\right)^{1/r}, r \geq 1$$

ist. Die Daten werden als k-dimensionale Vektoren der Form

$$x = (x_1, x_2, \ldots, x_k) \in R^k$$

dargestellt.

Ähnlichkeitsdaten

Bei Ähnlichkeitsdaten werden die Daten durch eine Menge X von Objekten dargestellt. Auf dieser Menge X ist ein Ähnlichkeitsmaß d definiert:

$$d : d(x, y) \rightarrow R,$$

mit $x, y \in X$. Für alle Objektpaare aus X können so deren Ähnlichkeits- bzw. Unähnlichkeitswerte für die Clusteranalyse verwendet werden.

Datenumwandlung

Während geometrische Daten verlustfrei in Ähnlichkeitsdaten umgewandelt werden können, indem z. B. die euklidische Metrik als Ähnlichkeitsmaß verwendet wird, ist die umgekehrte Richtung, die Einbettung von Ähnlichkeitsdaten in den euklidischen Raum R^k bei begrenztem k, meist nicht ohne Informationsverlust durchzuführen. Eine Technik zur näherungsweisen Einbettung ist die *Multidimensionale Skalierung* (siehe Anhang B), die in dieser Arbeit verwendet wird.

1.4 Beispiel einer Clusteranalyse

Abb. 1.1 zeigt eine Karte der karibischen Inseln. Die Aufgabe, die mittels Clusteranalyse hier gelöst werden soll, ist die Entdeckung der einzelnen Inseln und deren Zuordnung zu eigenständigen Clustern. Um das Bild auf die Clusteranalyse vorzubereiten werden die Beschriftungen entfernt und das Bild mit Hilfe des Floyd-Steinberg-Dithering-Algorithmus in den monochromen Farbraum transformiert. Die 14075 schwarzen Pixel im euklidischen Raum des Bildes (Auflösung: 344*226 Pixel) werden als die Objekte betrachtet, die es zu clustern gilt. Abb. 1.2 zeigt das umgewandelte Bild, welches als Eingabe für den Clusteralgorithmus DBSCAN dienen soll.

Abb. 1.3 zeigt das Ergebnis (Clustering), welches DBSCAN berechnet hat. Es wurden 24 unterschiedlichen Cluster gefunden, die durch verschiedene Farben dargestellt sind. Es ist deutlich zu erkennen, dass alle Inseln als eigenständige Cluster gefunden wurden. Bemerkenswert bei diesem Beispiel sind die sehr unterschiedlichen Formen der Cluster, die DBSCAN identifiziert hat.

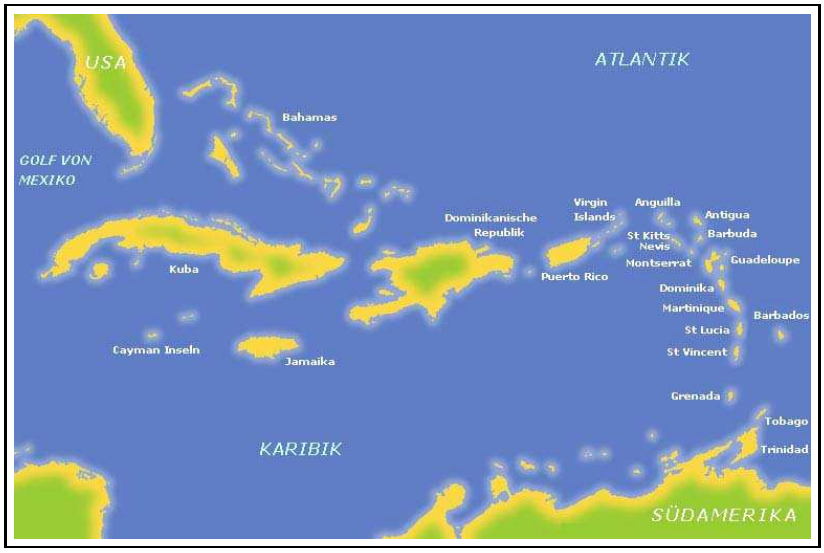

Abbildung 1.1: Übersichtskarte der karibischen Inseln.

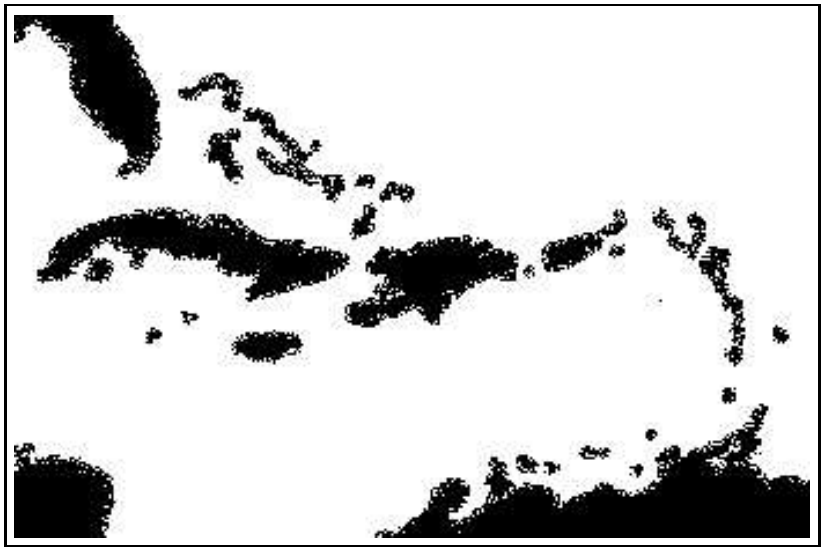

Abbildung 1.2: Monochrome Darstellung als Eingabedaten für die Clusteranalyse.

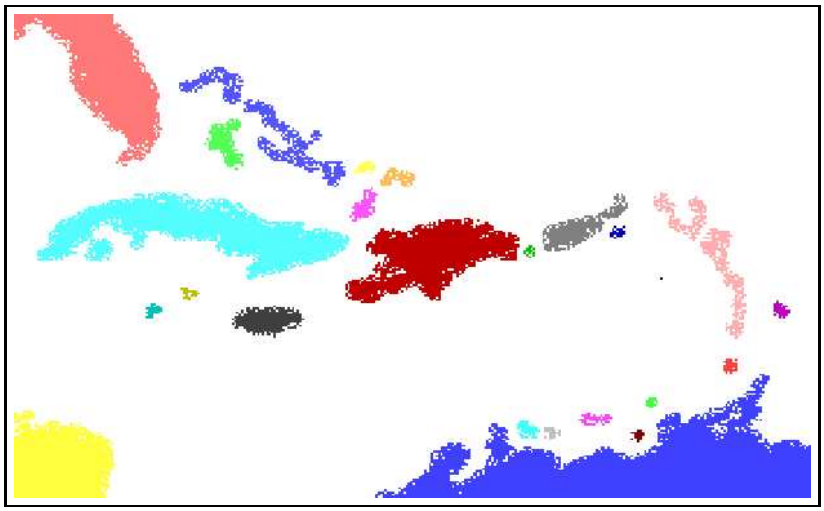

Abbildung 1.3: Clusterergebnis unter Verwendung des Algorithmus DBSCAN.

1.5 Komplexitätsüberlegungen

Um die Komplexität des Clusteringproblems zu verdeutlichen, lassen sich einige Überlegungen anstellen.

Als Beispiel seien 25 verschiedene Objekte gegeben, die in 5 Cluster aufgeteilt werden sollen. Ein naiver Algorithmus könnte so vorgehen, dass er sämtliche möglichen Clusterings berechnet und anschließend das, bezüglich eines Optimierungskriteriums, Geeignetste auswählt. Die Anzahl aller möglichen Clusterings, also alle Variationen, um n Objekte auf k verschiedene Gruppen beliebiger Größe aufzuteilen, lässt sich mit der Stirlingschen Zahl zweiter Ordnung berechnen:

$$S_n^{(k)} = \frac{1}{k!} \sum_{j=0}^{j=k} (-1)^{k-j} \binom{k}{j} j^n$$

Mit den Werten des obigen Beispiels, also $n = 25$ und $k = 5$, ergeben sich $S_{25}^{(5)} = 2.436.684.974.110.751$ mögliche Gruppierungen.
Wird nun noch zusätzlich gefordert, dass die optimale Anzahl der Cluster auch von dem Clusteralgorithmus ermittelt werden soll, steigt die Anzahl

der möglichen Gruppierungen auf

$$\sum_{i=1}^{i=25} S_{25}^{(i)} > 4 \cdot 10^{18}$$

an, da die Clusteranzahl mindestens eins und höchstens 25 betragen kann. Sogar auf einem Teraflop-Rechner, der eine Billion Rechenoperationen pro Sekunde durchführen kann, würde diese vermeintlich simple Aufgabe schon über 11 Tage in Anspruch nehmen, wobei mehrere Millionen Terabyte Speicherplatz notwendig wären; dabei wäre allerdings das Problem, das geeignetste aller Clusterings auszuwählen, noch nicht gelöst.

Ein unüberwachtes Problem

Eine weitere Schwierigkeit, die die Clusteranalyse noch komplexer macht, ist die Tatsache, dass es sich um ein unüberwachtes Problem handelt. Dies bedeuted, dass es außer den Problemdaten selbst keine weiteren Eingaben gibt, die zur Lösung des Problems beitragen. Da jedes Clusterproblem ein neues, einzigartiges ist, können die Algorithmen nicht aus vorherigen Problemlösungsprozeduren Wissen wiederverwenden, sie sind also nicht lernfähig.

1.6 Clusteralgorithmen

Wie der letzte Abschnitt gezeigt hat, ist das naive Konstruieren aller möglichen Clusterings und anschließendes Auswählen der besten Lösung nicht realisierbar, was den Bedarf an leistungsfähigen, heuristischen Clusterstrategien bekräftigt.

Im nächsten Abschnitt werden wesentliche Strategien vorgestellt.

1.6.1 Kategorien

Heutzutage gibt es eine Vielzahl an Clusteralgorithmen, die sich durch verschiedene Strategien und Heuristiken unterscheiden. Abb. 1.4 zeigt eine Kategorisierung bekannter Algorithmen. In den folgenden Abschnitten werden die wichtigsten Kategorien kurz vorgestellt.

Hierarchische Algorithmen

Agglomerative hierarchische Algorithmen beginnen mit einem Anfangszustand, in dem jedes Objekt der Problemdaten einem eigenen Cluster zu-

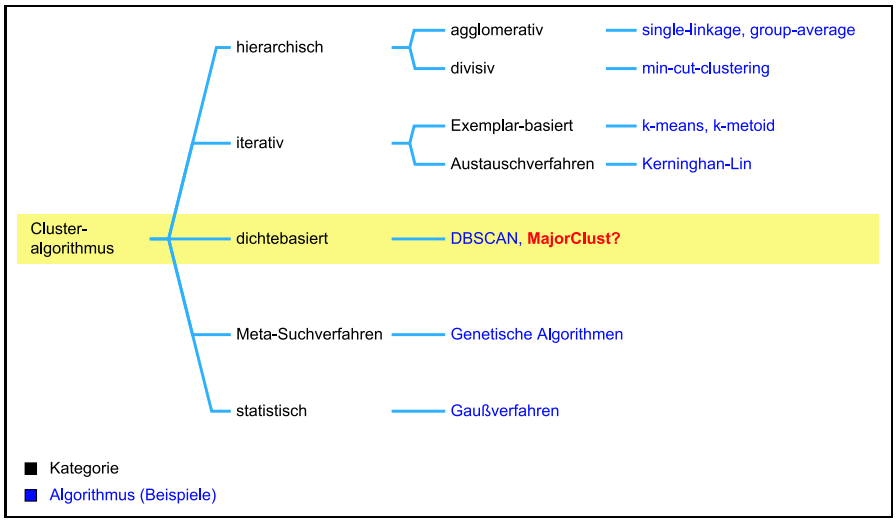

Abbildung 1.4: Eine Taxonomie von wichtigen Clusteralgorithmen.

geordnet ist. Im weiteren Verlauf der Clusteranalyse werden dann die bezüglich des verwendeten Ähnlichkeitsmaßes ähnlichsten Cluster miteinander verschmolzen. Dieser Schritt wird solange wiederholt, bis eine bestimmte Abbruchbedingung erreicht ist.

Divisive hierarchische Algorithmen hingegen gehen von einem einzelnen Cluster aus, in dem sich zu Beginn alle Objekte befinden. Im weiteren Verlauf werden die Cluster sukzessive in kleinere Cluster aufgeteilt, bis der Algorithmus aufgrund des Abbruchkriteriums terminiert.

Graphisch lässt sich die Funktionsweise hierarchischer Algorithmen als Dendogramm darstellen (siehe Abb. 1.5).

Iterative Algorithmen

Iterative Algorithmen versuchen, ausgehend von einem existierenden Clustering, durch Verlagerung der Objekte in andere Cluster zu einer besseren Lösung zu gelangen. Iterative Algorithmen benötigen als Eingabeparameter die Anzahl der Cluster, die erzeugt werden sollen.

Bei dem *Exemplar-basierten iterativen Verfahren* wird für jedes Cluster ein Repräsentant errechnet, zu dem die Objekte aufgrund ihrer Ähnlichkeit zugeordnet werden.

Bei dem *Austauschverfahren* hingegen werden immer zwei Objekte aus

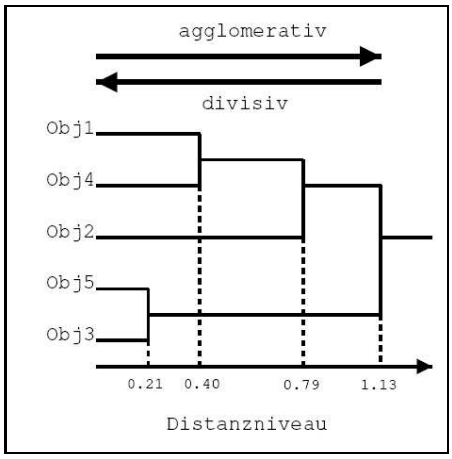

Abbildung 1.5: Dendogramm hierarchischer Clusteralgorithmen.

unterschiedlichen Clustern vertauscht, um die Lösung zu verbessern.

Dichtebasierte Algorithmen

Algorithmen dieser Kategorie suchen nach Regionen hoher Dichte im Merkmalsraum. Für diese Regionen werden verschiedene Cluster angelegt und alle darin befindlichen Objekte dem Cluster hinzugefügt. Während der Begriff der Dichte eindeutig definiert werden kann (siehe Kapitel 2), verfolgen verschiedene Algorithmen recht unterschiedliche Strategien, um dieses Ziel zu erreichen.

Ob die Strategie von MajorClust dem dichte-basierten Ansatz ähnlich ist, und die Einordnung von MajorClust somit sinnvoll ist, soll im weiteren Verlauf dieser Arbeit untersucht werden.

Vergleich der Clusterstrategien

Anhand einiger Eigenschaften lassen sich die Qualitäten und bevorzugten Einsatzgebiete der verschiedenen Strategien vergleichen.

Dichte-basierte Algorithmen weisen den großen Vorteil auf, dass sie Cluster beliebiger Formen entdecken können, während andere Algorithmen mit nicht-konvexen Clustern erhebliche Probleme haben. Des Weiteren lassen sich diese Algorithmen mit speziellen Datenstrukturen sehr effizient implementieren, so dass sie auch für größere Datenmengen geeignet sind. Der

größte Nachteil dichtebasierter Verfahren ist jedoch die starke Abhängigkeit des Clusteringergebnisses von ihren Eingabeparametern. Iterative Verfahren weisen als ähnlichen Nachteil die Notwendigkeit auf, dass die gewünschte Clusteranzahl a priori bekannt sein muss.

Tabelle 1.1 fasst den qualitativen Vergleich zusammen.

	iterativ	hierarchisch	dichte-basiert
Skalierbarkeit	-	-	+
Nicht-konvexe Cluster	-	-	+
Eingabeparameter	-	+	-

Tabelle 1.1: Vergleich der Clusterstrategien

1.7 Zusammenfassung

Dieses Kapitel hat die Clusteranalyse als Technik zur unüberwachten Gruppierung von Daten vorgestellt und die Begriffe Clustering und Cluster formal definiert. Des Weiteren wurden euklidische Daten und Ähnlichkeitsdaten als wichtige Formate für Clusterprobleme vorgestellt.

Es wurden Beispiele zu Einsatzgebieten der Clusteranalyse gegeben und deren Funktionsweise demonstriert.

Des Weiteren wurden Überlegungen zur Komplexität des Clusterproblems angestellt und somit der Bedarf an leistungsfähigen Clusteralgorithmen begründet.

Zuletzt wurde eine Taxonomie zur Zeit existierender Algorithmen vorgestellt und die wichtigsten Kategorien erläutert.

Kapitel 2

Dichte

Die Intention dieses Kapitels ist es, dem Leser ein Gefühl für den Begriff der Dichte zu vermitteln.

Nach einer kurzen Einführung in die dichtebasierte Clusteranalyse in Abschnitt 2.1 wird in Abschnitt 2.2 der Unterschied zwischen parametrischen und nicht-parametrischen Methoden in der Statistik dargelegt.

Als eines der einfachsten Mittel zum Schätzen der Dichte wird in Abschnitt 2.3 das Histogramm anhand einer eindimensionalen Zufallsstichprobe vorgestellt. Mit diesem Grundwissen wird die Definition des Histogramms auf den mehrdimensionalen Raum ausgeweitet.

Die Simplizität des Histogramms wird mit einigen Nachteilen erkauft, welche der Kerndichteschätzer nicht mit sich bringt. Abschnitt 2.4 stellt deshalb den Kerndichteschätzer als verbessertes Werkzeug zur Dichteschätzung vor.

Abschließend wird in Abschnitt 2.5 ein anschaulicher Bezug zur dichtebasierten Clusteranalyse genommen und mit verschiedenen Beispielen demonstriert.

2.1 Einführung

Die Idee der dichtebasierten Clusteranalyse ist es, Regionen im Merkmalsraum zu bestimmen, die eine hohe Anzahl von Objekten (also eine hohe Dichte) aufweisen. Gleichzeitig sollten diese Regionen jedoch durch Bereiche mit einer kleinen Anzahl von Objekten (geringe Dichte) abgegrenzt sein.

Der erste Schritt der dichtebasierten Clusteranalyse ist deshalb eine Bestimmung der verschiedenen Dichten im Merkmalsraum. Natürlich ist vor

Durchführung der Clusteranalyse unbekannt, wo sich Clusterkandidaten befinden werden. Es lassen sich deshalb keine abgegrenzten Bereiche festlegen, für die Dichtewerte im Sinne der Formel

$$\text{Dichte} = \frac{\text{Anzahl Objekte}}{\text{Einnehmender Flächeninhalt}}$$

berechnet werden sollten.

Die Lösung für diese Konflikt ist die Bestimmung der Dichte im statistischen Sinn, also die Bestimmung der *Dichteverteilung* der Problemdaten. Diese Bestimmung kann sowohl parametrisch als auch nicht-parametrisch durchgeführt werden. Im nächsten Abschnitt werden beide Methoden verglichen.

2.2 Parametische und nicht-parametrische Dichteschätzung

Gegeben sei eine Zufallsvariable X, die einer unbekannten, stetigen Verteilungsfunktion $F(x) = P(X \leq x)$ genügt und für die die Dichte f mit $F(x) = P(X \leq x) = \int_{-\infty}^{x} f(t)dt$ geschätzt werden soll. Dabei gilt für die Dichte:

$$f(x) \geq 0, \int_{-\infty}^{\infty} f(x)dx = 1$$

Parametrische Dichteschätzung

Bei der parametrischen Dichteschätzung wird vorausgesetzt, dass bekannt ist, *welcher* Verteilungsfunktion X genügt, jedoch nicht, *wie* die Daten verteilt sind.

Ist beispielsweise bekannt, dass die unbekannte Zufallsvariable X der Normalverteilung

$$F(x) = P(X \leq x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{(t-\mu)^2}{2\sigma^2}} dt$$

genügt, so reicht es aus, den Erwartungswert μ sowie die Streuung σ^2 zu bestimmen, um die unbekannte Dichte

$$f(t) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(t-\mu)^2}{2\sigma^2}}$$

schätzen zu können.

Diese Methode birgt jedoch den entscheidenden Nachteil, dass schon geringe Abweichungen zwischen der erwarteten und der tatsächlichen Verteilungsfunktion zu sehr schlechten Ergebnissen führen können.

Nicht-parametrische Dichteschätzung

Diesen Nachteil haben die nicht-parametrische Methoden hingegen nicht, da sie keine Annahmen über die Verteilungsfunktion von X treffen und somit wesentlich flexibler als parametrische Methoden sind. Alle Informationen zur Bestimmung der Dichte wird aus den Daten selber gewonnen.

Nicht-parametrische Methoden eignen sich daher besonders für Probleme, für die a priori keine Vorhersagen über die Verteilungsfunktion gemacht werden können, sie sind also prädestiniert für die variantenreichen Probleme der Clusteranalyse.

Im nächsten Abschnitt wird das Histogramm als eine der einfachsten Methoden der parameterlosen Dichteschätzung vorgestellt.

2.3 Das Histogramm als einfacher Dichteschätzer

Die Aufgabe eines Histogramms ist es, die unbekannte Dichte f einer stetigen Verteilung X zu bestimmen. Dabei wird nicht vorausgesetzt, dass d eine bestimmte parametrische Gestalt, wie etwa die Normalverteilung, hat.

Der Einfachheit halber wird das Histogramm zunächst im eindimensionalen Raum definiert; im weiteren Verlauf wird die Definition auf den mehrdimensionalen Raum erweitert.

Um ein Histogramm zur Schätzung von f zu berechnen, wird eine Zufallsstichprobe $X_1, \dots X_n$ aus X benötigt. Der Kerngedanke des Histogramms ist es nun, das Intervall $[X_{min}, \dots, X_{max}]$ dieser Daten in k disjunkte Teilintervalle, auch Klassen oder Bins genannt, der Breite h zu zerlegen. Ausgehend von einem Startpunkt x_0 ist das Intervall der m-ten Klasse

$$\Delta_m(h) = [x_0 + (m-1)h, x_0 + mh],$$

mit $1 \leq m \leq k$ (siehe Abb. xx).

Die Idee des Histogramms ist es nun zu zählen, wieviele Datenpunkte sich in den verschiedenen Klassen befinden. Die Daten werden also diskretisiert. Die Klassenhäufigkeit n_i, also die Anzahl der Datenpunkte in Klasse

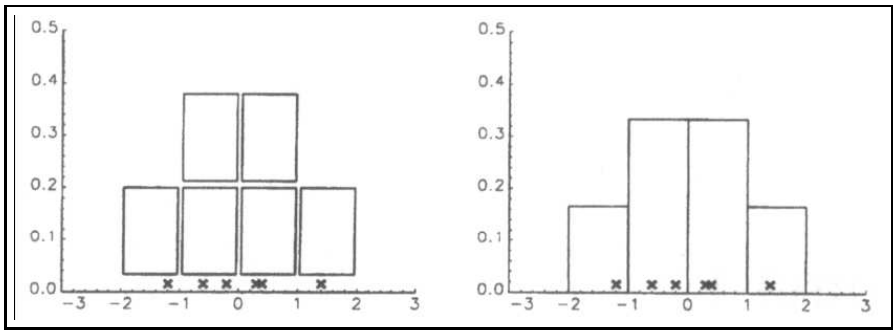

Abbildung 2.1: Das Histogramm als Summe gestapelter Blöcke auf den Klassenmitten der Klassen der jeweiligen Beobachtungen. (Quelle: [19])

i, lässt sich dann wie folgt berechnen:

$$n_m = \sum_{i=1}^{n} 1_{X_i \in \Delta_m(h)}.$$

Mit Hilfe dieser Klassenhäufigkeiten lässt sich nun an Stelle x die Dichte schätzen:

$$f(x) = \frac{1}{nh} \cdot (n_m \mid x \in \Delta_m(h)).$$

Die Normierung, also das Dividieren durch nh, muss durchgeführt werden, damit die Dichte die Anforderungen

$$f(x) \geq 0, \int_{X_{min}}^{X_{max}} f(x)dx = 1$$

erfüllt.

Grafisch betrachtet wird für jeden Datenpunkt ein Block mit der Fläche $1/n$ und der Breite h auf der Klassenmitte gestapelt, in der der Punkt liegt. Die Kreuze an der Abszisse der Diagramme in Abb. 2.1 stellen die Datenpunkte dar. Je mehr Datenpunkte in einer Klasse liegen, desto mehr Blöcke werden aufeinander gestapelt. Die Fläche der Rechtecke der einzelnen Klassen, die sich als die Summe der Flächen der übereinandergestapelten Blöcke ergeben, repräsentieren dann die Klassenhäufigkeit (vgl. [10]). Wie ein Histogramm die zugrunde liegenden Daten darstellt, hängt von zwei Parametern ab: Klassenbreite h und Ursprung x_0. Abb. 2.2 und 2.3 zeigen, wie die beiden Parameter das Histogramm beeinflussen. Es gibt verschiedene

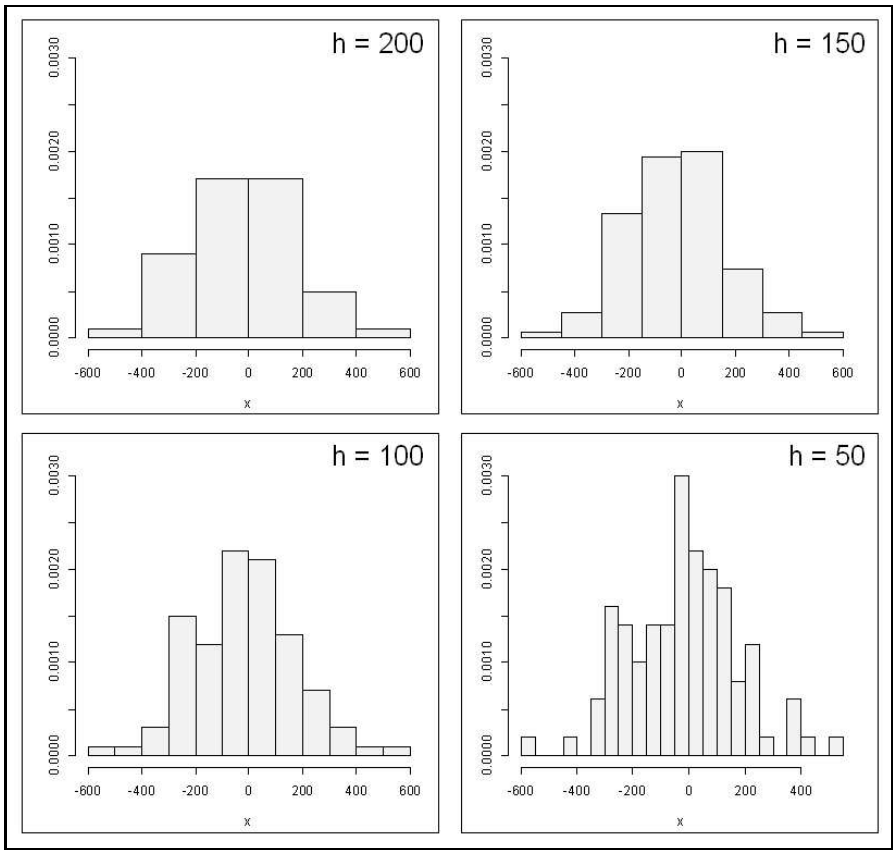

Abbildung 2.2: Einfluss der Bandbreite: Histogramme für denselben Datensatz mit $h = 50, 100, 150, 200$

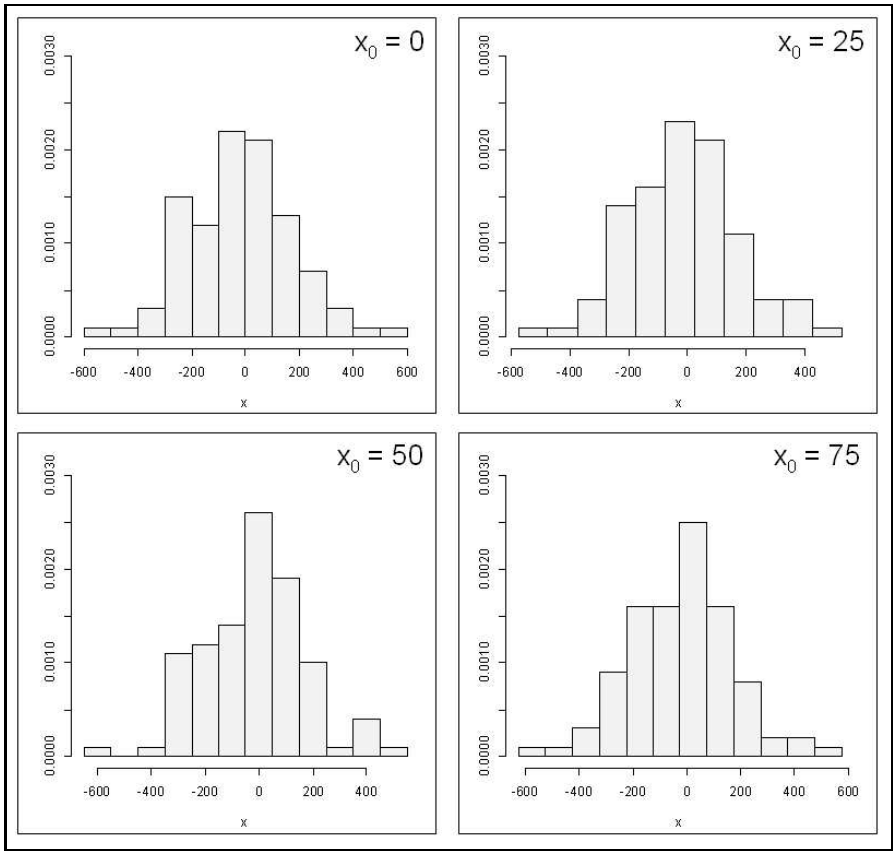

Abbildung 2.3: Einfluss des Ursprungs: Histogramme für denselben Datensatz mit $x_0 = 0, 25, 50, 75$ und $h = 100$

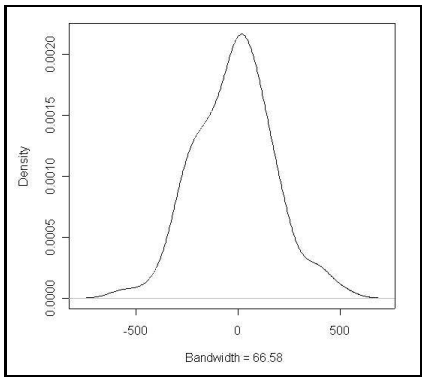

Abbildung 2.4: Kerndichteschätzer mit Kernfunktion: Dichte der Standardnormalverteilung. Quelle: [19]

Verfahren zur Bestimmung von h und zur Minimierung des Einflusses von x_0 auf das Histogramm. Dazu sei auf entsprechende Literatur verwiesen ([10],[19],[9]).

Mehrdimensionale Histogramme

Analog zum eindimensionalen Fall, in dem der Merkmalsraum in k Klassen der Breite h aufgeteilt wird, wird der d-dimensionale Merkmalsraum in $d \cdot k$ Bereiche aufgeteilt. Für $d = 2$ kann man sich ein zweidimensionales Bild vorstellen über das zur Klasseneinteilung ein Gitter gelegt wird. Um die Klassenhäufigkeiten zu zählen, würden dann über jedem Gitterfeld Würfel gestapelt.

2.4 Kerndichteschätzer

Ein Histogramm hat zwei entscheidende Nachteile. Zum Einen findet ein Informationsverlust durch die Diskretisierung der ursprünglichen Daten statt. Des Weiteren ist das Histogramm ein Werkzeug zur Schätzung von stetigen Dichtefunktionen, es ist selbst jedoch unstetig. Um diese Schwächen auszugleichen, werden Kerndichteschätzer eingesetzt.

Im Gegensatz zum Histogramm verwendet ein Kerndichteschätzer keine Blöcke, die übereinander gestapelt werden. Die Verbesserung besteht in der Benutzung von Kernfunktionen, die über jede Beobachtung gestapelt

werden. Diese müssen die Eigenschaften einer Dichtefunktion

$$f(x) \geq 0, \int_{-\infty}^{\infty} f(x)dx = 1$$

erfüllen. Abb. 2.4 zeigt als Beispiel einen Kerndichteschätzer, der als Kernfunktion die Dichte der Standardnormalverteilung verwendet:

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}.$$

Es ist deutlich zu erkennen, dass es sich um eine stetige Dichteschätzung handelt und kein Treppeneffekt wie beim Histogramm zu erkennen ist.

Im nächsten Abschnitt soll unter Verwendung einer Dichteschätzung von Problemdaten mit Hilfe eines Kerndichteschätzers die Idee der dichtebasierten Clusteranalyse dargelegt werden.

2.5 Dichtebasierte Clusteranalyse

Um ein Gefühl dafür zu bekommen, wie die Kenntnis der Dichteverteilung einer Datenmenge für die Clusteranalyse ausgenutzt werden kann, wird im Folgenden das Beispiel der karibischen Inseln aus Abschnitt 1.4 weitergeführt.

Abb. 2.5a) zeigt die Dichteschätzung der Karte, welche die karibischen Inseln zeigt (vgl. Abb. 1.1), die durch Einsatz eines Kerndichteschätzers (Kernfunktion: Dichte der Standardnormalverteilung) aus den entsprechenden Daten (vgl. Abb. 1.2) ermittelt wurde.

Um sich nun die Vorgehensweise der Clusteranalyse anhand dieser Dichteverteilung verständlich zu machen, ist es hilfreich sich vorzustellen, diese Dichteverteilung würde mit Wasser befüllt. Wird Wasser abgelassen, treten zuerst die höchsten Spitzen der Dichteverteilung aus der Wasseroberfläche hervor wie in Abb. 2.5b zu sehen ist.

Ein Algorithmus könnte bildlich gesprochen so vorgehen, dass er das Wasser zu einem gewissen Teil ablässt und die Spitzen, die separat, dh. getrennt voneinander durch Wasser, aus der Wasseroberflache hervortreten, in einzelne Cluster klassifiziert. Abb. 2.5c beispielsweise zeigt einen Wasserstand, bei dem die Dominikanische Republik und Kuba in einzelne Cluster angeordnet würden - eine durchaus sinnvolle Klassifikation.

Abb. 2.5c jedoch macht eine Problematik des dichte-basierten Clusterings erkennbar. Um die Insel Puerto Rico entdecken zu können, muss der Wasserspiegel sehr weit abgesenkt werden. Dann aber sind Kuba und die

Dominikanische Republik nicht mehr durch Wasser voneinander getrennt und werden einem gemeinsamen Cluster zugeordnet.

Betrachtet man die Höhe, auf die der Wasserspiegel abgelassen wird, als Eingabeparameter für einen dichte-basierten Clusteringalgorithmus wird deutlich, dass das Clusteringergebnis sehr stark von diesem Parameter abhängt.

In dem nächsten Kapitel wird der Algorithmus DBSCAN vorgestellt, der diesen Vorgang des dichte-basierten Clusterings auf einem heuristischen Weg nachzubilden versucht.

2.6 Zusammenfassung

Dieses Kapitel hat den Begriff der Dichte im statistischen Sinne erläutert und das Histogramm als einfachstes Werkzeug zum Schätzen der Dichteverteilung einer Zufallsvariablen vorgestellt. Danach wurde der Kerndichteschätzer als Verbesserung des Histogramms eingeführt, der Kernfunktionen, welche den Bedingungen einer Dichtverteilung genügen müssen, verwendet. Abschließend wurde die Idee der dichtebasierten Clusteranalyse beispielhaft erläutert.

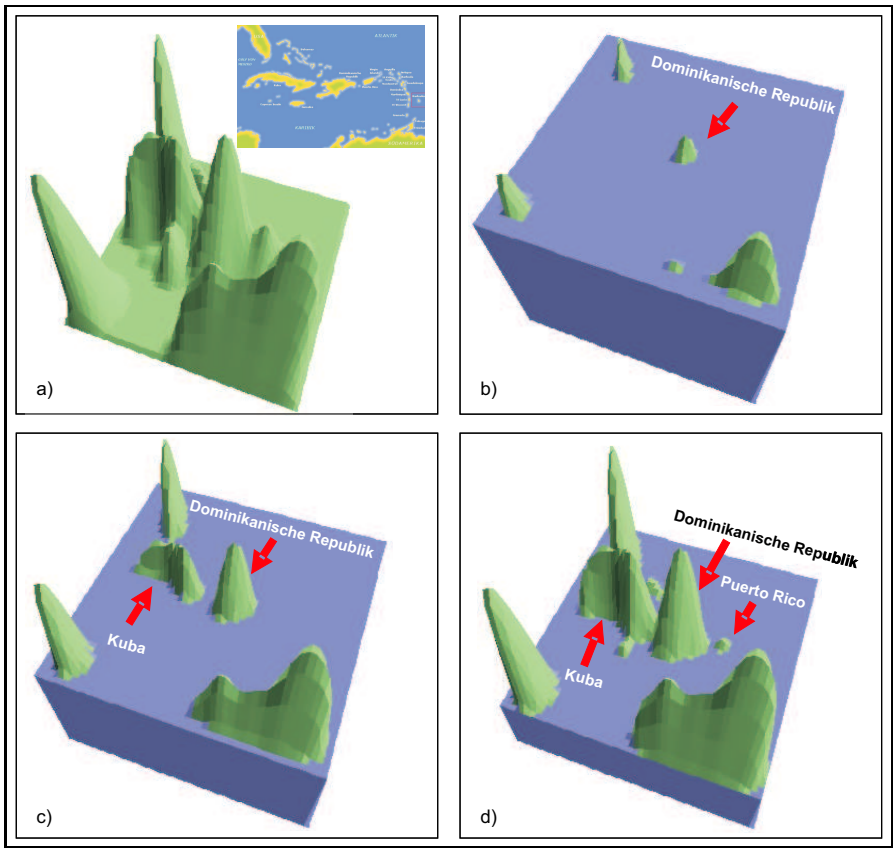

Abbildung 2.5: a) Dichteschätzung der karibischen Inseln mittels Kerndichteschätzer (Kernfunktion: Standardnormalverteilung). b) Wasserspiegel ist gerade soweit abgesenkt, dass die Dominikanische Republik auftaucht. c) Die Dominikanische Republik und Kuba sind durch Wasser klar voneinander getrennt. d) Der Wasserspiegel wird weiter abgesenkt, so dass Puerto Rico auftaucht. Die Dominikanische Republik und Kuba bilden jetzt jedoch ein Cluster.

Kapitel 3

DBSCAN

Der Name DBSCAN ist eine Abkürzung für "**D**ensity-**B**ased **S**patial **C**lustering of **A**pplications with **N**oise."[1]

Die wesentlichen Zielsetzungen von DBSCAN sind dreigeteilt:

- Minimales Dömanenwissen erforderlich, um die Eingabeparameter zu bestimmen
- Entdeckung von Clustern beliebiger Form
- Gute Performance bei großen Datenmengen (Datenbanken)

Dieses Kapitel stellt den Algorithmus DBSCAN vor. Abschnitt 3.1 erläutert die Strategie, die DBSCAN zur Clusteranalyse verfolgt. In Abschnitt 3.2 wird der Algorithmus als Pseudocode vorgestellt und die Funktionsweise erläutert. Abschnitt 3.3 verdeutlicht die theoretischen Überlegungen mit einem praktischen Beispiel für eine Clusteranalyse mit DBSCAN.

3.1 Clusterstrategie

3.1.1 Problemdaten

Es sei angenommen, dass die Problemdaten eine Menge von Punkten darstellen, die in einer Datenbank D enthalten sind. Diese Punkte befinden sich im euklidischen Raum mit beliebiger Dimension k und einer darauf definierten Metrik $d : D \times D \to \mathbb{R}$.

[1] etwa: Dichte-basierte räumliche Clusteranalyse für Anwendungen mit Rauschen.

In den folgenden Abbildungen und Beispielen wird die Funktionsweise von DBSCAN im 2-dimensionalen Raum mit euklidischer Metrik demonstriert.

3.1.2 Strategie

In Kapitel 2 wurde der Kerngedanke der dichtebasierten Clusteranalyse vorgestellt: die Punktdichte innerhalb eines Clusters sollte bedeutend höher sein, als außerhalb des Clusters. Des Weiteren sollten Bildpunkte, die sinnvollerweise keinem Cluster zugeordnet werden können, als Rauschen[2] identifiziert werden. Diese Idee verfolgt DBSCAN durch die Analyse der Umgebungen der einzelnen Punkte. Die folgenden Definitionen bereiten die Einführung der Begriffe DBSCANCluster[3] und Noise vor, so dass die Strategie von DBSCAN formalisiert werden kann.

Definition 4 *(ϵ-Umgebung) Die ϵ-Umgebung eines Punktes p , bezeichnet als $N_\epsilon(p)$, sei wie folgt definiert: $N_\epsilon(p) = \{q \in D \mid d(p,q) \leq \epsilon\}$.*

Die verwendete Metrik d legt die Form der ϵ-Umgebung fest. Wird die Manhatten-Distanz benutzt, hat die ϵ-Umgebung quadratische Umrisse; die euklidische Distanz führt zu einer Hypersphäre.

DBSCAN untersucht die ϵ-Umgebung aller Punkte aus D. Ein naiver Algorithmus könnte lediglich fordern, dass sich in der ϵ-Umgebung jeden Punktes in einem Cluster eine Mindestanzahl von Punkten ($MinPts$) befinden müssen. Dies ist jedoch eine zu verallgemeinernde Vorgehensweise, denn es kann zwischen zwei Arten von Punkten eines Clusters unterschieden werden:

- Punkte Innerhalb des Clusters (Kernpunkte),
- Punkte am Rand des Clusters (Randpunkte).

Definition 5 *(Kernpunkt, core(D)) Ein Kernpunkt $p \in D$ ist ein Punkt, für den die Bedingung $|N_\epsilon(p)| \geq MinPts$ erfüllt ist. Weiterhin enthalte die Menge $core(D) = \{p \mid p \in D \wedge |N_\epsilon(p)| \geq MinPts\}$ alle Kernpunkte aus D.*

Üblicherweise befinden sich in der ϵ-Umgebung eines Kernpunktes weit mehr Punkte als in der ϵ-Umgebung eines Randpunktes. Um trotzdem zu

[2] engl.: Noise.

[3] Da in Definition 1 der Begriff Cluster schon allgemein definiert wurde, soll hier der Begriff DBSCANCluster als Interpretation des Algorithmus DBSCAN verwendet werden, um Mehrdeutigkeiten vorzubeugen.

gewährleisten, dass alle Punkte dem Cluster hinzugefügt werden, müsste der $MinPts$-Parameter sehr klein gewählt werden, so dass er nicht mehr charakteristisch für das Cluster wäre.

Die folgenden Definitionen bereiten eine sinnvollere Definition eines DBSCANClusters vor.

Definition 6 *(Direkte Erreichbarkeit) Ein Punkt p ist direkt erreichbar von einem Punkt q bezüglich ϵ und MinPts, wenn*
1) $p \in N_\epsilon(q)$ und
2) q ein Kernpunkt ist.

Offensichtlich ist direkte Erreichbarkeit symmetrisch für zwei Kernpunkte, für einen Kern- und einen Randpunkt hingegen ist sie asymmetrisch.

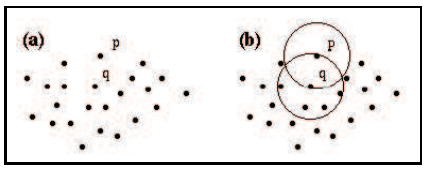

Abbildung 3.1: a) Randpunkt p, Kernpunkt q b) Asymmetrische Fall der direkten Erreichbarkeit: p ist direkt erreichbar von q, jedoch ist q nicht direkt erreichbar von p. Quelle: [6]

Definition 7 *(Erreichbarkeit) Ein Punkt p ist erreichbar von einem Punkt q bezüglich ϵ und MinPts, wenn es einen Weg von Punkten $p_1, \ldots, p_n \wedge p_1 = q \wedge p_n = p$ gibt, so dass p_{i+1} direkt-erreichbar von p_i ist.*

Erreichbarkeit ist eine asymmetrische, transitive Erweiterung der direkten Erreichbarkeit. Abb. 3.1 zeigt einige Beispiele zur Erreichbarkeit.

Zwei Punkte desselben Clusters C, die keine Kernpunkte und deshalb Randpunkte sind, sind gegenseitig nicht erreichbar. Allerdings muss es einen Kernpunkt in C geben, von dem aus beide Randpunkte erreichbar sind. Für diese Fälle wird die Verbundenheit eingeführt.

Definition 8 *(Verbundenheit) Ein Punkt p ist verbunden mit einem Punkt q bezüglich ϵ und MinPts, wenn es einen Punkt o gibt, so dass sowohl p als auch q von o bezüglich ϵ und MinPts erreichbar sind.*

Verbundenheit ist eine symmetrische Relation. Für erreichbare Punkte ist sie darüberhinaus reflexiv (vgl. Abb. 3.2).

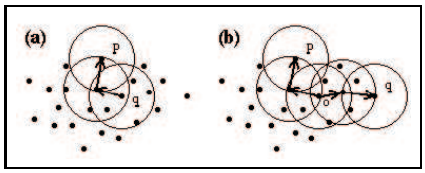

Abbildung 3.2: a) Asymmetrischer Fall der Erreichbarkeit: p ist erreichbar von q; q ist jedoch nicht erreichbar von p. b) Die Randpunkte p und q sind miteinander über den Kernpunkt o verbunden. Quelle: [6]

Mit den Begriffen Erreichbarkeit und Verbundenheit als Grundlage ist es nun möglich, die Interpretation des Algorithmus DBSCAN von Clustern und Rauschen zu formalisieren.

Definition 9 *(DBSCANCluster) Sei D eine Datenbank mit Punkten. Ein DBSCANCluster C bezüglich ϵ und MinPts ist eine nicht-leere Teilmenge von D, welche die folgenden Bedingungen erfüllt:*
1) $\forall p, q : p \in C \wedge q$ ist erreichbar von p bezüglich $\epsilon, MinPts \Rightarrow q \in C$ (Maximalität)
2) $\forall p, q \in C : p$ ist verbunden mit q bezüglich $\epsilon, MinPts$. (Konnektivität)

Definition 10 *(Noise) Seien $C_1, \ldots, C_k$ DBSCANCluster der Datenbank D bezüglich $\epsilon_i, MinPts_i, i = 1, \ldots, k$. Dann sei noise $:= \{p \in D \mid \forall i : p \notin C_i\}$.*

Ein DBSCANCluster stellt also eine Menge von bezüglich ϵ und $MinPts$ verbundenen Punkten dar. Noise hingegen beinhaltet alle übrigen Punkte aus D, die aufgrund dieser Definition keinem Cluster zugeordnet werden können.

Es ist anzumerken, dass ein DBSCANCluster mindestens $MinPts$ Punkte enthält. Diese Behauptung kann wie folgt bewiesen werden. Ein DBSCANCluster C enthält mindestens einen Punkt p. Per Definition muss p mit sich selbst über einen beliebigen Punkt $o \in C$ verbunden sein (wobei $p = o$ gelten kann), weshalb o ein Kernpunkt ist. Da sich in der ϵ-Umgebung von o mindestens $MinPts$ Punkte befinden, folgt $|C| \geq MinPts$.

Im nächsten Abschnitt wird DBSCAN als Pseudocode vorgestellt. Um die Korrektheit des Algorithmus zu zeigen, werden zuvor noch einige Überlegungen angestellt.

Sind die Parameter ϵ und $MinPts$ gegeben, kann ein Cluster C in zwei Schritten entdeckt werden. Zuerst wird ein beliebiger Punkt $p \in core(D)$, zu

C hinzugefügt. Alle weiteren Punkte aus D werden ebenfalls C zugeordnet, falls sie von p aus erreichbar sind. Diese Vorgehensweise setzt voraus, dass jedes Cluster eindeutig mittels eines beliebigen, diesem Cluster zugehörigen Kernpunktes ermittelt werden kann.

Lemma 1 *Sei $p \in core(D)$. Dann bildet die Menge $O = \{o \mid o \in D \wedge o$ ist erreichbar von p bezüglich ϵ und $MinPts\}$ ein Cluster bezüglich ϵ und $MinPts$.*

Jeder Punkt in C ist erreichbar von jedem der Kernpunkte $p \in core(C)$. Deshalb enthält C genau die Punkte, die von einem beliebigen Kernpunkt $p \in core(C)$ erreichbar sind.

Lemma 2 *Sei C ein Cluster bezüglich ϵ und $MinPts$ und sei $p \in core(C)$ ein beliebiger Kernpunkt aus C. Dann ist C gleich der Menge $O = \{o \mid o \in D \wedge o$ ist erreichbar von p bezüglich ϵ und $MinPts\}$*

3.2 Algorithmus

Im Folgenden wird DBSCAN als Pseudocode vorgestellt. Dabei wird nicht auf Implementierungsdetails, wie verwendete Datenstrukturen, eingangen.

Algorithmus DBSCAN.
Eingabe: Punktmenge P, ϵ, MinPts.
Ausgabe: Eine Funktion $c : P \rightarrow \mathbb{N}$, die jedem Punkt eine Clusternummer zuordnet.

---

(01) $ClusterId := 1$
(02) $UNCLASSIFIED := -1, NOISE := -2$
(03) $\forall p \in P$ **do** $c(p) := UNCLASSIFIED$ **enddo**
(04) $\forall p \in P$ **do**
(05) **if** $c(p) = UNCLASSIFIED$ **then**
(06) **if** $ExpandCluster(P, p, ClusterId, \epsilon, MinPts) = true$ **then**
(07) $ClusterId := ClusterId + 1$
(08) **endif**
(09) **endif**
(10) **enddo**

Zunächst wird jeder Punkt der Datenbank D als unklassifiziert gekennzeichnet (03). In der Hauptschleife besucht DBSCAN jeden Punkt, der noch als unklassifiziert markiert ist, genau einmal. Für jeden dieser Punkte wird die folgende Funktion ExpandCluster aufgerufen (06).

Funktion ExpandCluster.
Eingabe: $P, CurrentPoint, ClusterId, \epsilon, MinPts$.
Ausgabe: $\{true, false\}$.

```
(01)  seeds := regionQuery(P, CurrentPoint, ε)
(02)  if |seeds| < MinPts then
(03)      c(CurrentPoint) := NOISE
(04)      return false
(05)  else
(06)      ∀p ∈ P do c(p) := ClusterId enddo
(07)      seeds := seeds \ CurrentPoint
(08)      while seeds ≠ ∅ do
(09)          p := seeds.first()
(10)          result := regionQuery(P, p, ε)
(11)          if |result| ≥ MinPts then
(12)              ∀ResPoint ∈ P do
(13)                  if c(ResPoint) ∈ {UNCLASSIFIED, NOISE} then
(14)                      if c(ResPoint) = UNCLASSIFIED then
(15)                          seeds := seeds ∪ ResPoint
(16)                      endif
(17)                      c(ResPoint) := ClusterId
(18)                  endif
(19)              enddo
(20)          endif
(21)          seeds := seeds \ p
(22)      enddo
(23)      return true
(24)  endif
```

In Zeile (01) wird mit dem Aufruf regionQuery die ϵ-Umgebung des aktuellen Punktes $CurrentPoint$ untersucht. Handelt es sich bei $CurrentPoint$

um einen Kernpunkt, dann werden die Punkte in dieser Umgebung der seeds-Liste hinzugefügt, andernfalls wird *CurrentPoint* als Noise klassifiziert. Sollte es sich bei diesem Punkt jedoch nicht um Rauschen sondern um einen Randpunkt handeln, dann wird seine Klassifizierung im weiteren Verlauf noch einmal geändert.

Die ϵ-Umgebungen aller Punkte in seeds werden mit dem Aufruf regionQuery untersucht (10); alle neu gefundenen Punkte, welche noch unklassifiziert sind, werden seeds ebenfalls hinzugefügt (15). Sollte ein Punkt gefunden werden, der bereits als Noise klassifiziert wurde, so wird er statt dessen dem aktuellen Cluster zugeordnet (17). Auf diese Weise sucht der else-Teil der if-Anweisung (05)-(23) alle von *CurrentPoint* erreichbaren Punkte. Da dieser else-Teil nur ausgeführt wird, sofern es sich bei *CurrentPoint* um einen Kernpunkt handelt, wird immer ein vollständiges Cluster gefunden, wie Lemma 2 beweist.

Komplexität

Die Laufzeit von DBSCAN hängt im Wesentlichen von der Umsetzung der Funktion regionQuery ab. Eine naive Implementierung etwa, die für jeden Punkt aus D überprüfen muss, ob er in der ϵ-Umgebung des zu untersuchenden Punktes liegt, benötigt n Operationen für eine query. Da für jeden Punkt aus D höchstens einmal regionQuery aufgerufen wird, ergibt sich eine Gesamtlaufzeit von $O(n^2)$.

Eine effizientere Implementierung von regionQuery lässt sich mit komplexeren Datenstrukturen wie etwa dem R^*-Tree (vgl. [4], [8]) realisieren, der lediglich $O(\log n)$ Schritte zur Ermittlung der ϵ-Umgebung benötigt, so dass die Gesamtlaufzeit auf $O(n \log n)$ verbessert werden kann.

3.3 Beispiel

Abb. 3.3 und 3.4 zeigen ein Beispiel für eine Clusteranalyse mit DBSCAN.

3.4 Zusammenfassung

In diesem Kapitel wurde der Algorithmus DBSCAN vorgestellt. Dazu wurde die Strategie erläutert, welche DBSCAN verfolgt, und wichtige Begriffe wie ϵ-Umgebung, Kernpunkt, DBSCANCluster und Noise einführt. Der Algorithmus wurde als Pseudocode dargestellt und die Funktionsweise anhand eines Beispiels demonstriert.

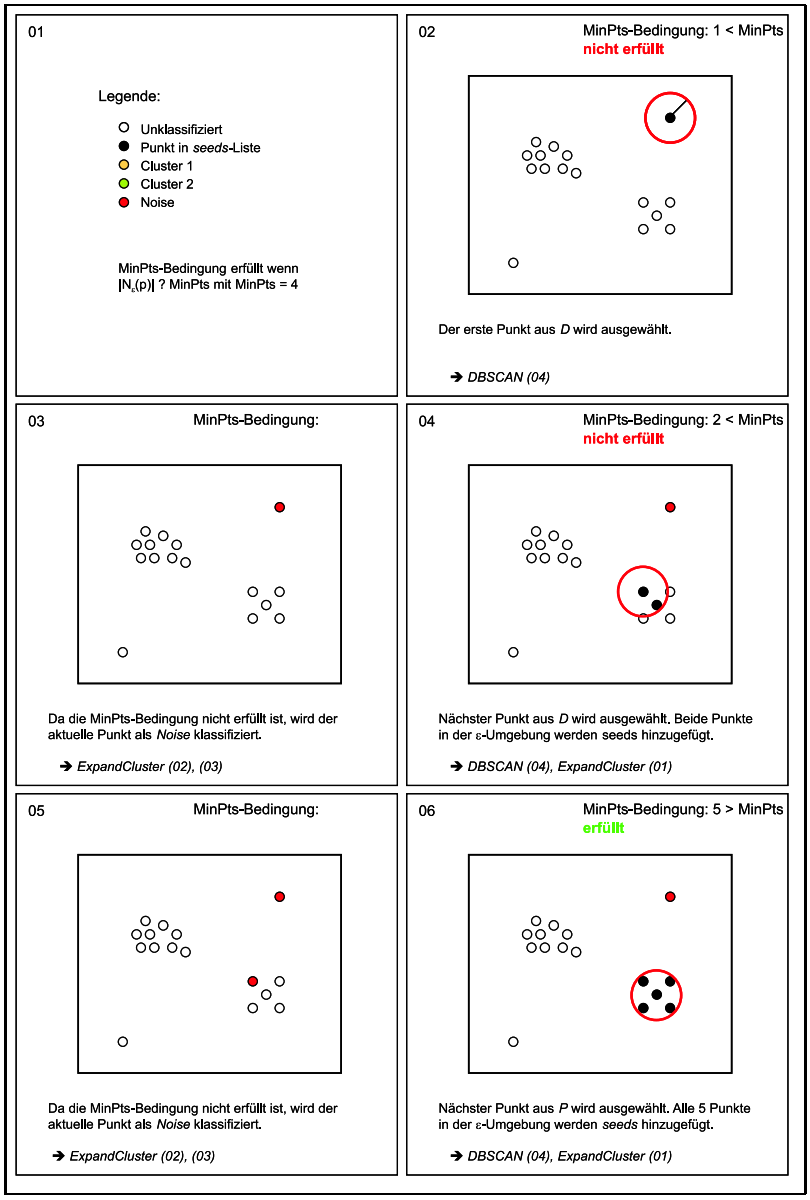

Abbildung 3.3: Beispiel für den Algorithmus DBSCAN 1/2

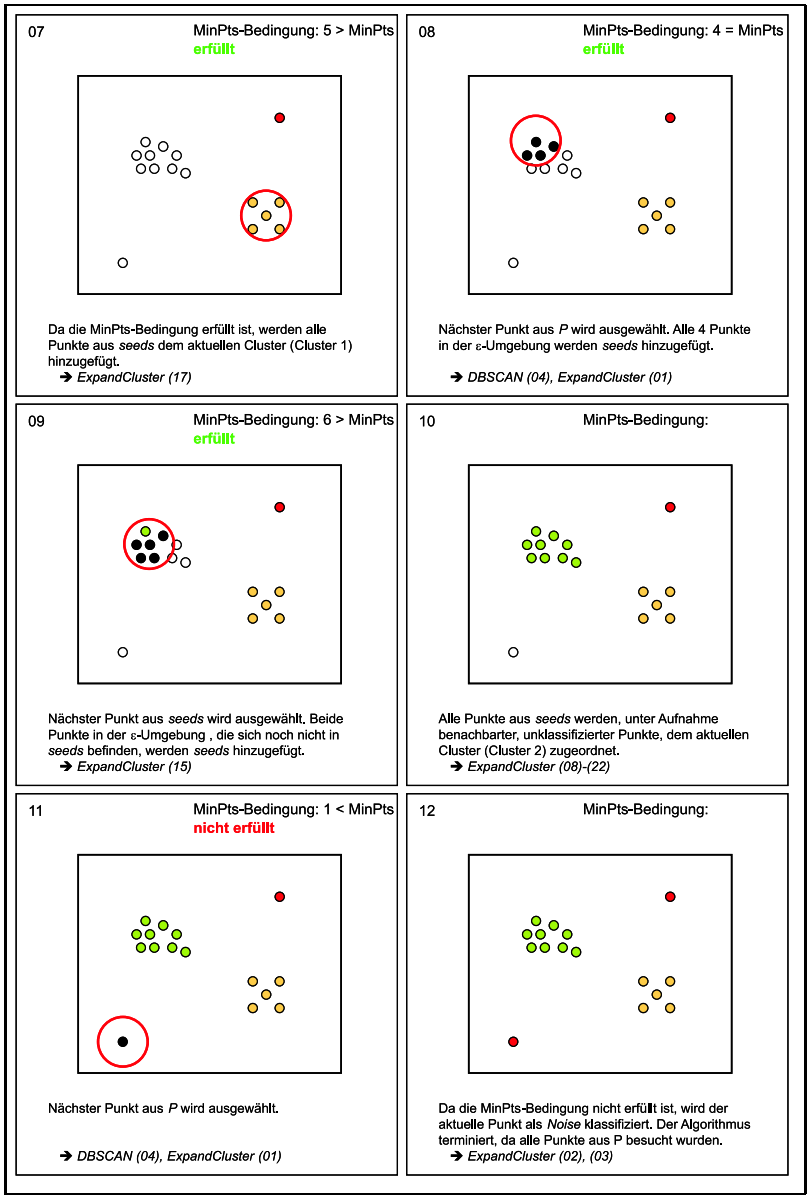

Abbildung 3.4: Beispiel für den Algorithmus DBSCAN 2/2

Kapitel 4

MajorClust

MajorClust ist ein Clusteralgorithmus, der an der Universität Paderborn entwickelt wurde. Dieser Algorithmus betrachtet die Objekte, die klassifiziert werden sollen, als geschlossenes System, in dem eine Struktur aufgedeckt werden soll. Durch eine lokale Heuristik approximiert MajorClust eine Maximierung des Gütemaßes "gewichteter partieller Kantenzusammenhang" Λ, welches die Qualität eines Clusterings, also einer Strukturierung des geschlossenen Systems, angibt.

In diesem Kapitel wird der Algorithmus MajorClust angelehnt an [16] vorgestellt:

- Abschnitt 4.1 veranschaulicht die Betrachtung der Problemdaten als geschlossenes System mit einer praktischen Problemstellung. Danach wird die Strategie von MajorClust erläutert, wobei insbesonders auf den "gewichteten partiellen Kantenzusammenhang" Λ eingegangen wird.

- In Abschnitt 4.2 wird MajorClust als Pseudocode vorgestellt. Dabei werden Besonderheiten des Algorithmus sowie dessen Komplexität beleuchtet.

- Abschnitt 4.3 schließt das Kapitel mit einer beispielhaften Clusteranalyse mit MajorClust ab.

4.1 Clusterstrategie

Im Gegensatz zur Vorgehensweise von DBSCAN betrachtet der Algorithmus MajorClust die Objekte, die geclustert werden sollen, als geschlossenes System. Ein solches System wird als ungerichteter Graph dargestellt, der gewichtete oder ungewichtete Kanten haben kann. Ziel von MajorClust ist es, Strukturen in diesem Graph und somit in dem geschlossenen System zu identifizieren. Die Komponenten der gefundenen Struktur bilden dann die Cluster.

Im Folgenden werden Vorbereitungen getroffen, um den Begriff Struktur formalisieren zu können.

4.1.1 Problemdaten

Es sei angenommen, dass die Problemdaten eine Menge X von Objekten darstellen. Auf dieser Menge X sei ein Ähnlichkeitsmaß d definiert (vgl. 1.3). Die Funktionen f und g bilden die Daten auf einen Graphen $G = \langle V, E, w \rangle$ ab:

1. $f : X \to V$ ist eine bijektive Funktion welche jedes Objekt aus $x \in X$ auf einen Knoten $v \in V$ abbildet.

2. $g : X \times X \times \mathbb{R} \to V \times V \times \mathbb{R}$ ist eine surjektive Funktion, welche die paarweisen Ähnlichkeiten der Objekte auf die Kanten zwischen den entsprechenden Knoten abbildet. Dabei wird nicht gefordert, dass der Graph vollständig ist, d. h. $(\forall \langle (x, y), d(x, y) \rangle$ mit $x, y \in X$ und $d(x, y) \in \mathbb{R}) \Rightarrow (\exists \langle e, w(e) \rangle$ mit $e = (f(x), f(y)) \in E$ und $w(e) \in \mathbb{R})$ muss nicht erfüllt sein.

Anwendungsbeispiel: Dokumentenmodelle

Im Bereich des Information Retrieval werden Dokumentenmodelle verwendet, um Dokumente (Texte) zu abstrahieren. Ein häufig verwendetes Dokumentenmodell ist das Vektorraummodell, welches n Dokumente d_j als m-dimensionale Vektoren $\vec{d_j} = (w_{1,j}, w_{2,j}, \ldots, w_{m,j})$ darstellt, wobei m die Anzahl der verschiedenen Worte im Vokabular bezeichnet, also die Worte, die in den Dokumenten vorkommen.

Bei der einfachsten Vektordarstellung, der binären Gewichtung, bezeichnet der i-te Eintrag im Vektor des Dokuments, ob das i-te Wort des Vokabulars in diesem Dokument vorkommt oder nicht. Eine Erweiterung ist

die tfidf-Gewichtung, bei der die Einträge der Vektoren eine Kombination aus den normalisierten Worthäufigkeiten tf[1] und den inversen Dokumentenhäufigkeiten idf[2] darstellen.

Mit Hilfe einer Funktion $\varphi : (\vec{d_1}, \vec{d_2}) \mapsto [0; 1]$ kann dann die Ähnlichkeit zweier Dokumente d_1 und d_2 berechnet werden. Häufig wird das wie folgt definierte Kosinusmaß verwendet:

$$\varphi(d_1, d_2) = \frac{\langle \vec{d_1}, \vec{d_2} \rangle}{||\vec{d_1}|| \cdot ||\vec{d_2}||},$$

wobei $\langle \vec{d_1}, \vec{d_2} \rangle = \vec{d_1}^T \vec{d_2}$ das Skalarprodukt und $||\vec{d}||$ die euklidische Distanz bezeichnen. Die Ähnlichkeit zweier Dokumente aus einer Kollektion mit n Dokumenten und einem Vokabular mit m Wörten berechnet sich also mit

$$\varphi(d_1, d_2) = \frac{\sum_{i=1}^{m} w_{i,1} \cdot w_{i,2}}{\left(\sum_{i=1}^{m} w_{i,1}^2\right)^{1/2} \cdot \left(\sum_{i=1}^{m} w_{i,2}^2\right)^{1/2}}.$$

Die Aufgabe, welche die Clusteranalyse in diesem Anwendungsbeispiel zu lösen hat, ist eine automatische Kategorisierung der Dokumente. Dabei werden keine Kategorien vorgegeben, sondern die Dokumente sollen anhand ihrer paarweisen Ähnlichkeiten gruppiert werden. Dazu wird ein ungerichteter, gewichteter Graph G konstruiert, welcher für jedes Dokument d_i einen entsprechenden Knoten v_i beinhaltet. Dabei ist G vollständig, d. h. zwischen je zwei Knoten v_i, v_j aus G existiert eine Kante, deren Gewicht der Ähnlichkeit $\varphi(d_i, d_j)$ entspricht. Dass MajorClust, angewendet auf einen solchen Graphen, tatsächlich eine Kategorisierung der Dokumente erzeugt, wird in Abschnitt 4.3 demonstriert.

4.1.2 Strategie

Zunächst wird die Strategie von MajorClust anhand von ungewichteten Graphen $G = \langle V, E \rangle$ beschrieben. Im weiteren Verlauf werden die Definitionen auf gewichtete Graphen $G = \langle V, E, w \rangle$ erweitert.

Das Verständnis des Begriffs Struktur eines Systems basiert auf den folgenden Merkmalen:

- Modularität. Das System (der Graph $G = \langle V, E \rangle$) kann in einzelne Module (Cluster) zerlegt werden, so dass jedes Element (jeder Knoten

[1] tf (term frequency): Normalisierte Anzahl der Vorkommnisse des Wortes im durch den Vektor repräsentierten Dokument.

[2] idf (inversed document frequency): Anzahl der Dokumente, in denen das Wort vorkommt.

$v \in V$) zu genau einem Cluster gehört. Eine solche Modularisierung wird analog zu Definition 1 als Clustering bezeichnet[3].

- Konnektivität. Die Cluster werden implizit durch Ausnutzung der Konnektivität zwischen den Knoten definiert. Dabei soll die Konnektivität zweier Knoten, die dem gleichen Cluster zugeordnet werden, größer als die Konnektivität zweier Knoten sein, welche unterschiedlichen Clustern zugeordnet sind.

- Kontraktion. Die Kontraktion von G, d. h. die Substituierung von Knoten die ein gemeinsames Cluster bilden durch einen einzelnen Knoten, wird als die Struktur von G angesehen.

Gewichteter partieller Kantenzusammenhang Λ

Nachdem der Begriff Struktur informell beschrieben wurde soll im Folgenden das Maß "Gewichteter partieller Kantenzusammenhang" Λ eingeführt werden, welches die Güte einer Modularisierung des Graphen G, also eines Clusterings $C(G)$, berechnet. Analog zu Definition 1.2 wird das Clustering eines Graphen $C(G)$ definiert:

Definition 11 *Sei $G = \langle V, E\rangle$ ein Graph. Ein Clustering $C(G) = (C_1, \ldots, C_n)$ bezeichne die Aufteilung von G in n Teilgraphen $G(C_i)$ induziert durch die Teilmengen C_i mit $\bigcup_{C_i \in C} C_i = V$ und $\forall C_i, C_j \in C : C_i \cap C_{j \neq i} = \emptyset$. Diese Teilmengen C_i werden als Cluster bezeichnet.*

Das Maß Λ basiert auf dem graphen-theoretischen Konzept des Kantenzusammenhangs.

Definition 12 *(Kantenzusammenhang) Der Kantenzusammenhang $\lambda(G)$ eines Graphen G gibt die minimale Anzahl der Kanten von G an, die entfernt werden müssen, damit G nicht zusammenhängend ist: $\lambda(G) = min\{|E^{'}| : E^{'} \subset E$ und $G^{'} = \langle V, E \setminus E^{'}\rangle$ ist nicht zusammenhängend$\}$.*

Definition 13 *(Gewichteter partieller Kantenzusammenhang Λ) Sei $G = \langle V, E\rangle$ ein Graph und sei $C = (C_1 \ldots C_n)$ ein Clustering der Knoten von G. Der gewichtete partielle Kantenzusammenhang von C, $\Lambda(C)$, sei definiert als*

$$\Lambda(C) := \sum_{i=1}^{n} |C_i| \cdot \lambda(C_i),$$

[3]Die Menge D von Objekten wird durch die Menge V der Knoten des Graphen G dargestellt.

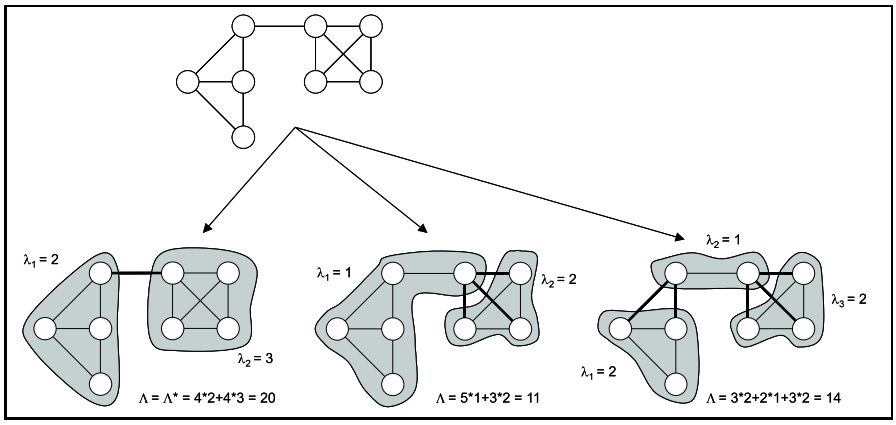

Abbildung 4.1: Verschiedene Clusterings eines Graphen und die zugehörigen Λ-Werte. Quelle: [16]

wobei $\lambda(C_i)$ den Kantenzusammenhang des von der Menge C_i induzierten Teilgraphen $G(C_i)$ bezeichnet.

Abbildung 4.1 demonstriert den gewichteten partiellen Kantenzusammenhand Λ.

Definition 14 *(Struktur) Sei $G = \langle V, E \rangle$ ein Graph und sei C^* ein Clustering der Knoten von G, welches Λ maximiert:*

$$\Lambda(C^*) \equiv \Lambda^* := max\{\Lambda(C) \mid C \text{ ist ein Clustering von } G\}$$

Die Kontraktion $H = \langle C^*(G), E_{C^*} \rangle$ wird als die Struktur des durch G repräsentierten Systems bezeichnet.

Abbildung 4.2 zeigt, wie durch Maximierung von Λ eine Struktur im zugrunde liegenden Graph gefunden wird. Es ist anzumerken, dass die Anzahl der Cluster einer Struktur implizit in der Definition enthalten ist.

Erweiterung für gewichtete Graphen

Λ kann leicht zur Verwendung mit gewichteten Graphen erweitert werden. Dabei wird nicht nur die Existenz einer Kante sondern deren Gewicht zur Ermittlung von Λ berücksichtigt. Als Voraussetzung wird der gewichtete Kantenzusammenhang $\tilde{\lambda}$ eingeführt: $\tilde{\lambda}(G) = min\{\sum_{e \in E'} w(e) \mid E' \subset E$ und $G' = \langle V, E \setminus E' \rangle$ ist nicht zusammenhängend $\}$.

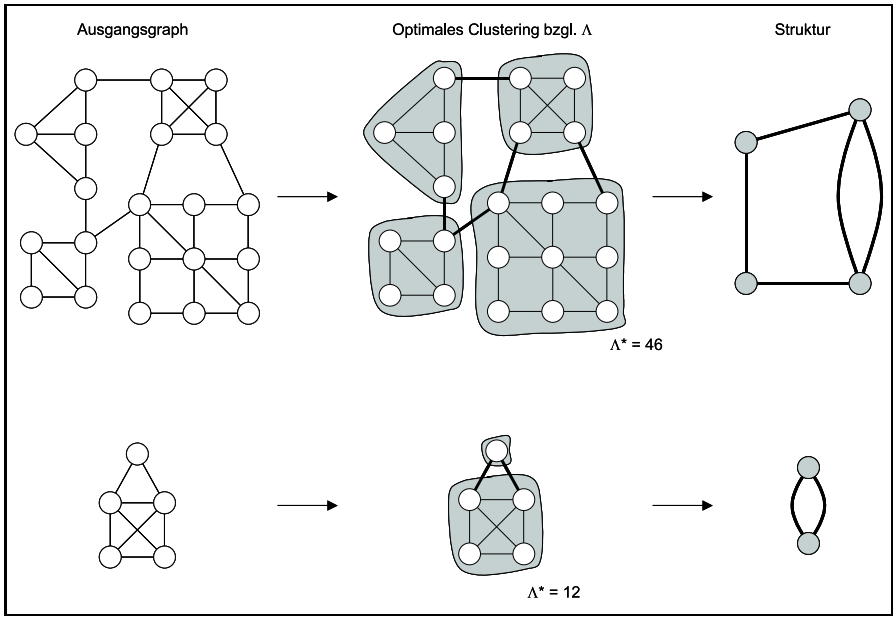

Abbildung 4.2: Beispiele für die Strukturierung eines Graphen. Quelle: [16]

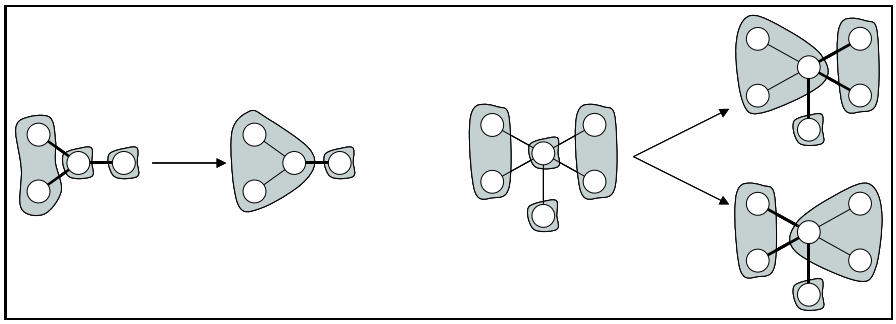

Abbildung 4.3: Links: Eindeutige Entscheidung von MajorClust, Rechts: Zwei mögliche Entscheidungen, die zum gleichen Λ-Wert führen. Quelle: [16]

Mit Hilfe des gewichteten Kantenzusammenhangs können die Definitionen aus diesem Abschnitt auf gewichtete Graphen erweitert werden.

4.2 Algorithmus

Da angenommen wird, dass das Problem der Maximierung von Λ NP-vollständig ist (siehe Anhang A), wurde der Algorithmus MajorClust entworfen, um die Optimierung von Λ möglichst effizient zu approximieren.

Anfangs weist der Algorithmus jeden Knoten von G einem eigenen Cluster zu. Es folgen Iterationen, in denen das Clustering optimiert wird. In jeder Iteration werden alle Knoten besucht. Für jeden Knoten wird überprüft, welches Cluster im bishering Clustering die größte Attraktivität aufweist. Falls der aktuelle Knoten v_i einem anderen Cluster als dem Attraktivsten zugeordnet ist, wird diese Zuordnung entsprechend geändert. Die Attraktivität eines Clusters C_i ergibt sich aus der Summe der Gewichte der Kanten, die von v_i zu Knoten aus C_i führen. Formal ausgedrückt berechnet sich die Attraktivität von Cluster C_i für Knoten v_i mit $\sum \{w(e) \mid e = \{u, v_i\} \in E \wedge u \in C_i\}$. Optional kann zusätzlich ein Schwellwert t angegeben werden, so dass nur Kanten e berücksichtigt werden, deren Gewichte $w(e) \geq t$ sind. (vgl. Zeile 06).

Sollten mehr als ein Cluster die gleiche Attraktivität für den aktuellen Knoten aufweisen, wird eines dieser Cluster zufallsig ausgewählt (vgl. Abb. 4.3). Nachdem ein stabiles Clustering gefunden wurde, also sobald in einer Iteration von MajorClust keine Clusterwechsel mehr stattgefunden haben,

terminiert der Algorithmus.

Im Folgenden wird MajorClust als Pseudocode vorgestellt.

Algorithmus MajorClust.
Eingabe: Ein gewichteter Graph $G = \langle V, E, w \rangle$, Schwellwert $t \in \mathbb{R}$
Ausgabe: Eine Funktion $c : V \to \mathbb{N}$, die jedem Knoten eine Clusternummer zuordnet.

---

(01) $n := 0, ready := false$
(02) $\forall v \in V$ **do** $n := n + 1, c(v) := n$ **enddo**
(03) **while** $ready = false$ **do**
(04) $\quad ready := true$
(05) $\quad \forall v \in V$ **do**
(06) $\quad\quad c^* := i$ **if** $\sum \{w(e) \mid e = \{u, v\} \in E \wedge w(e) \geq t \wedge c(u) = i\}$ max.
(07) $\quad\quad$ **if** $c(v) \neq c^*$ **then** $c(v) := c^*, ready := false$
(08) $\quad$ **enddo**
(09) **enddo**

Die sehr gute Laufzeit von MajorClust wird mit einer Suboptimalität erkauft. Da der Algorithmus auf lokale Entscheidungen beschränkt ist, und global Eigenschaften, wie etwa den Kantenzusammenhang, außer Acht lässt, wird nicht immer die optimale Lösung gefunden. Ein Beispiel für diese Suboptimalität ist in Abb. 4.4 zu sehen.

Komplexität

Die Laufzeit von MajorClust beträgt $\Theta(|E| \cdot |C_{max}|)$, wobei $C_{max} \subseteq V$ ein maximales Cluster darstellt. In der while-Schleife (03-08) wird jede Kante von G zweimal untersucht. In jeder Iteration wird ein Cluster um mindestens einen Knoten vergrößert bis keine Veränderungen mehr stattfinden und der Algorithmus terminiert. Es kann vorkommen, dass der Algorithmus zwischen zwei oder mehreren Clusterings oszilliert. Das ist jedoch kein schwerwiegender Nachteil, da diese Situationen leicht entdeckt werden können.

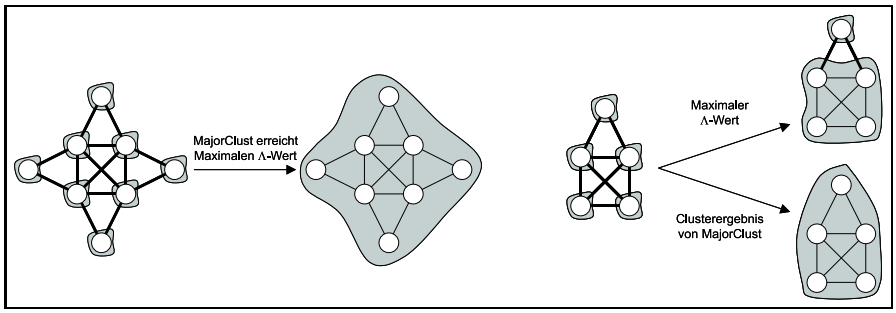

Abbildung 4.4: Die Beschränkung auf lokale Entscheidungen von MajorClust kann zu suboptimalen Λ-Werten führen. Quelle: [16]

4.3 Beispiel

Dieser Abschnitt illustriert eine Clusteranalyse mit MajorClust anhand des praktischen Beispiels der Kategorisierung von Dokumenten, welches bereits in Abschnitt 4.1 eingeführt wurde.

Die hier verwendete Dokumentenkollektion $D = \{d_1, d_2, \ldots, d_8\}$ besteht aus acht Texten, welche verschiedene Themen behandeln:

- Autos
  - d_1: “Audi A6 Avant“ (Quelle: www.autobild.de)
  - d_2: “Die Autos von morgen“ (Quelle: www.autobild.de)
  - d_3: “BMW 1er“ (Quelle: www.autobild.de)
- Fussball
  - d_4: “Borussia Dortmund“ (Quelle: www.kicker.de)
  - d_5: “Werder Bremen“ (Quelle: www.kicker.de)
  - d_6: “Bayern München“ (Quelle: www.kicker.de)
- Tiere
  - d_7: “Löwe“ (Quelle: www.tierenzyklopaedie.de)
  - d_8: “Tiger“ (Quelle: www.tierenzyklopaedie.de)

Im Folgenden soll demonstriert werden, wie MajorClust die entsprechenden Kategorien *Autos, Fussball* und *Tiere* entdeckt.

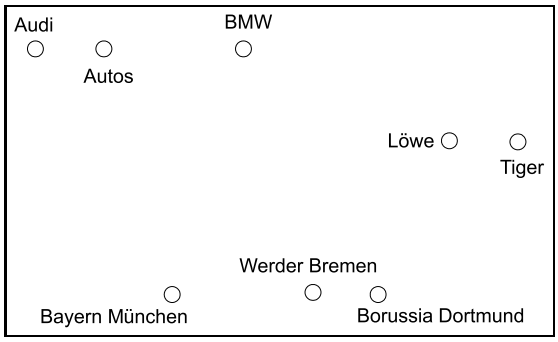

Abbildung 4.5: Darstellung der Ähnlichkeiten der Dokumente eingebettet mittels MDS im 2-dimensionalen Raum

Zunächst werden die Dokumente mittels tf-idf-Gewichtung vektorisiert. Mit Hilfe des Kosinusmaßes werden dann alle paarweisen Dokumentenähnlichkeit berechnet, die in Tabelle 4.1 dargestellt sind.

	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8
d_1	1.000							
d_2	0.337	1.000						
d_3	0.253	0.210	1.000					
d_4	0.065	0.017	0.039	1.000				
d_5	0.127	0.028	0.078	0.298	1.000			
d_6	0.073	0.025	0.218	0.227	0.249	1.000		
d_7	0.109	0.063	0.136	0.084	0.123	0.101	1.000	
d_8	0.080	0.051	0.077	0.034	0.059	0.059	0.478	1.000

Tabelle 4.1: Matrix der paarweisen Dokumentenähnlichkeiten

In diesem Beispiel sollen die Ähnlichkeiten visualisiert werden. Dazu wird die Multidimensionale Skalierung verwendet, um die Werte in den 2-dimensionalen Raum einzubetten. Abbildung 4.5 zeigt das Ergebnis der Einbettung.

Abbildung 4.6 demonstriert die Funktionsweise von MajorClust. Zu Beginn ist jedem Knoten ein eigenes Cluster zugeordnet. In jedem Schritt ist der aktuell von MajorClust besuchte Knoten gelb markiert. Darüberhinaus sind die Attraktivitäten der benachbarten Cluster angegeben. Die Attraktivität eines Clusters errechnet sich aus Summe der Kanten vom aktuellen Knoten zu den Knoten des jeweiligen Clusters. In jedem Schritt wird der

aktuell besuchte Knoten dem Cluster zugeordnet, welcher die größte Attraktivität aufweist.

Nach dem letzten Schritt der Clusteranalyse zeigt Abbildung 4.6 die Λ-Werte der einzelnen Cluster. Es ergibt sich eine Summe für das gesammte Clustering von $\Lambda(C) = \Lambda^* = 1,380 + 0,956 + 1,428 = 3,764$, was eine Maximierung des "gewichteten partiellen Kantenzusammenhangs" Λ bedeutet. MajorClust hat also ein bezüglich Λ optimales Clustering gefunden. Ein Vergleich mit den Dokumenten zeigt außerdem, dass das gefundene Clustering das gewünschte Ergebnis zeigt.

4.4 Zusammenfassung

In diesem Kapitel wurde der Algorithmus MajorClust vorgestellt. Dazu wurde zunächst der Begriff Struktur beschrieben und diesbezüglich das Optimierungskriterium "Gewichteter partieller Kantenzusammenhang" Λ definiert. Nachdem MajorClust als Pseudocode dargestellt wurde, demonstrierte das praktische Beispiel "Dokumentenkategorisierung" dessen Funktionsweise.

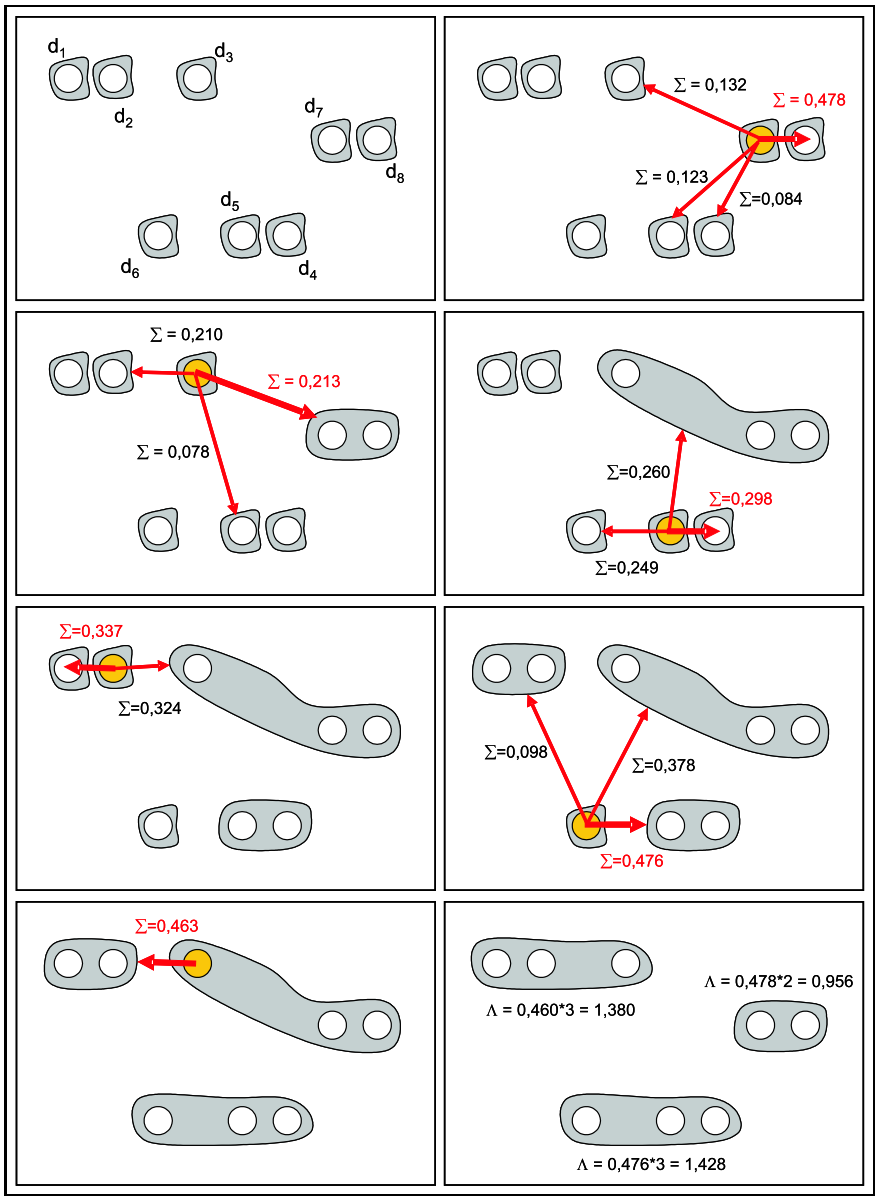

Abbildung 4.6: Beispiel für eine Clusteranalyse mit dem Algorithmus MajorClust

Teil II

Vergleich der Algorithmen

Kapitel 5

Testumgebung

Um Experimente mit den Algorithmen DBSCAN und MajorClust durchführen und miteinander vergleichen zu können, wurde eine Testumgebung in der Programmiersprache Java entwickelt, welche verschiedene Testszenarien ermöglicht. Dieses Kapitel stellt die Testumgebung anhand von Klassendiagrammen[1] vor und erläutert einzelne Komponenten:

- Abschnitt 5.1 veranschaulicht die verwendeten Formate, in denen die Problemdaten abgespeichert werden.
- In Abschnitt 5.2 werden die Implementierungen von DBSCAN und MajorClust erläutert.
- Abschnitt 5.3 stellt das F-Measure vor, ein Clustervaliditätsmaß, welches die Güte eines Clusterings bezüglich eines Referenzclusterings berechnet.

5.1 Datenformate

In Abschnitt 1.3 wurden bereits die beiden Datenformate vorgestellt, die für die Experimente verwendet werden sollen: geometrische Daten und Ähnlichkeitsdaten. Die Testumgebung bietet zwei Importierungsmöglichkeiten, um die Daten einzulesen. Diese werden in den nächsten Abschnitten beschrieben.

[1] Der Übersichtlichkeit halber zeigen die Klassendiagramme in diesem Kapitel nur die zum Verständnis der einzelnen Komponenten notwendigen Methoden und Variablen.

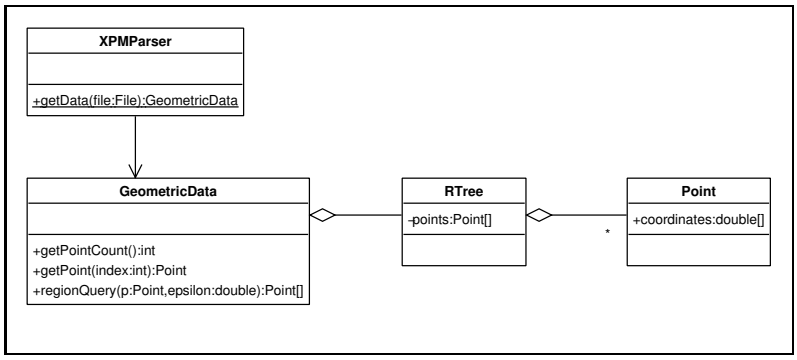

Abbildung 5.1: Klassendiagramm mit den an der Speicherung von geometrischen Daten beteiligten Klassen.

5.1.1 Geometrische Daten

Die Klasse `XPMParser` liest XPM-Bilddateien ein und speichert diese als geometrische Daten ab. Eine XPM-Datei ist eine Text(ASCII)-Datei, welche eine $m \times n$ Matrix beinhaltet. Jeder Eintrag dieser Matrix entspricht einem Pixel des kodierten Bildes. `XPMParser` speichert für jeden schwarzen Pixel mit Koordinaten $(i, j), 1 \leq i \leq m, 1 \leq j \leq n$ einen zweidimensionalen Datenpunkt $P = (x, y)$ ab, wobei $x = i$ und $y = j$ gesetzt werden. Es wird eine neues Objekt `GeometricData` erzeugt, welches die Datenpunkte enthält.

Abbildung 5.1 zeigt ein Klassendiagramm, welches die an der Speicherung von geometrischen Daten beteiligten Klassen darstellt.

Die einzelnen Datenpunkte werden in einer Datenstruktur namens R*-Tree abgespeichert ([4], [8]). Eine Suchanfrage, die als Ergebnis die Punkte zurückliefern soll, die sich in einer bestimmten Region im geometrischen Raum befinden, kann der R*-Tree in Zeit $O(\log n)$ berechnen (n = Anzahl Datenpunkte). Eine naive Datenstruktur würde hierfür $O(n)$ benötigen. In dieser Testumgebung wird die Open-Source Java-Implementierung vom R*-Tree namens “spatialindex“ verwendet [15].

5.1.2 Ähnlichkeitsdaten

Die Klasse `TfIdf` liest Textdateien ein, und wandelt diese mit Hilfe der tfidf Gewichtung in Vektoren um (vgl. Abbschnitt 4.1.1). Dazu wird ein Vokabular mit m Wörtern verwendet, so dass für jedes Dokument ein

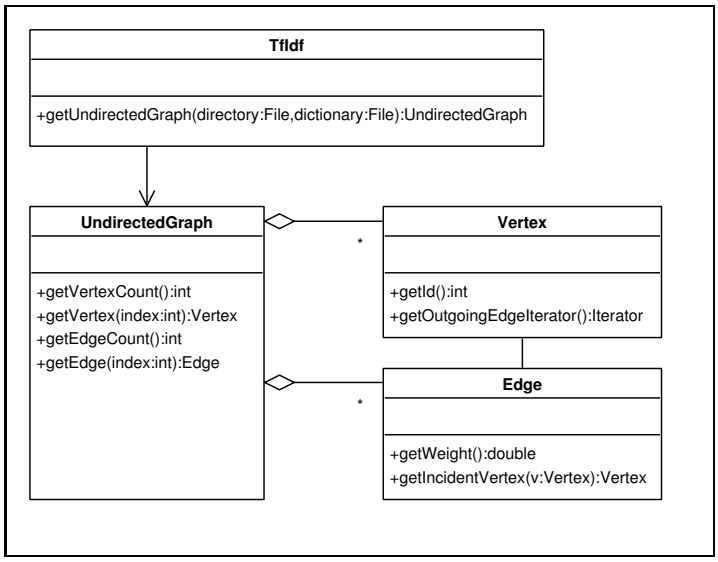

Abbildung 5.2: Klassendiagramm mit den an der Speicherung von Ähnlichkeitsdaten beteiligten Klassen.

m-dimensionaler Vektor erzeugt wird. Nach der Vektorisierung wird ein vollständiger ungerichteter Graph (Objekt `UndirectedGraph`) erstellt. Dieser enthält für jedes Dokument d_i einen Knoten v_i. Zwischen je zwei Knoten v_i und v_j existiert eine gewichtete Kante e_{ij}, deren Gewicht der Ähnlichkeit der Dokumente d_i und d_j entspricht, welche mit Hilfe des Kosinusmaßes berechnet wird. Formal: $w(e_{ij}) = \varphi(d_i, d_j)$. Abbildung 5.2 zeigt das entsprechende Klassendiagramm.

5.1.3 Datenkonvertierung

Ähnlichkeitsdaten $\Longrightarrow$ geometrische Daten

Um Ähnlichkeitsdaten in geometrische Daten zu konvertieren, wird die Multidimensionale Skalierung verwendet (siehe Anhang B). Dazu besitzt die Klasse `MultiDimensionalScaling` eine Anbindung an das "R-Tool" [14], ein Statistikprogramm, welches Implementierungen für verschiedenste statistische Verfahren beinhaltet. Der eingesetzte Befehl des "R-Tools" ist *isoMDS* aus dem Paket *MASS*.

Dem Objekt `MultiDimensionalScaling` wird ein `UndirectedGraph`-

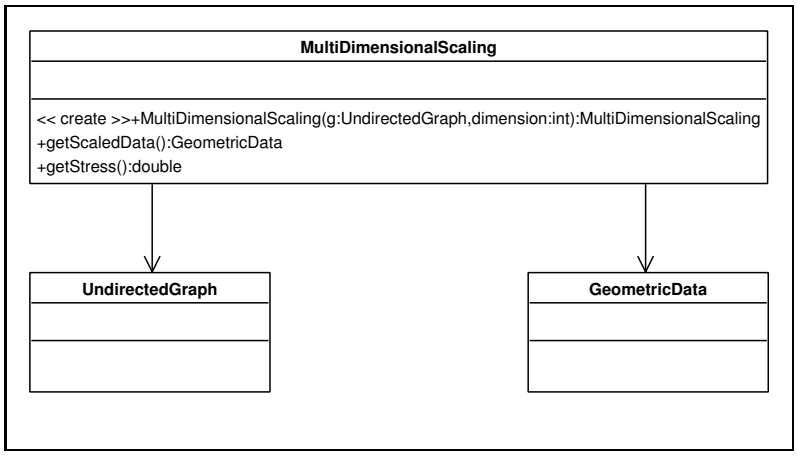

Abbildung 5.3: Klassendiagramm mit den an der multidimensionalen Skalierung beteiligten Klassen.

Objekt übergeben. Zudem wird die Dimension des Raumes übergeben, in den die Daten eingebettet werden sollen. `MultiDimensionalScaling` erstellt daraufhin eine $n \times n$ Matrix mit allen paarweisen Ähnlichkeitswerten der n Objekte des Graphen. Diese Matrix wird an den Befehl *isoMDS* übergeben und die multidimensionale Skalierung ausgeführt. Daraufhin werden die skalierten Daten eingelesen und in einem Objekt `GeometricData` gespeichert. Diese geometrischen Daten sowie der Stress-Wert der eingebetteten Daten stehen dann mit den Methodenaufrufen `getScaledData():GeometricData` und `getStress():Double` zur Verfügung.

Geometrische Daten $\Longrightarrow$ Ähnlichkeitsdaten

Die Umwandlung von geometrischen Daten in Ähnlichkeitsdaten ist nicht von Nöten, da der Algorithmus MajorClust bereits so implementiert wurde, dass er sowohl Objekte des Typs `UndirectedGraph` als auch `GeometricData` verarbeiten kann.

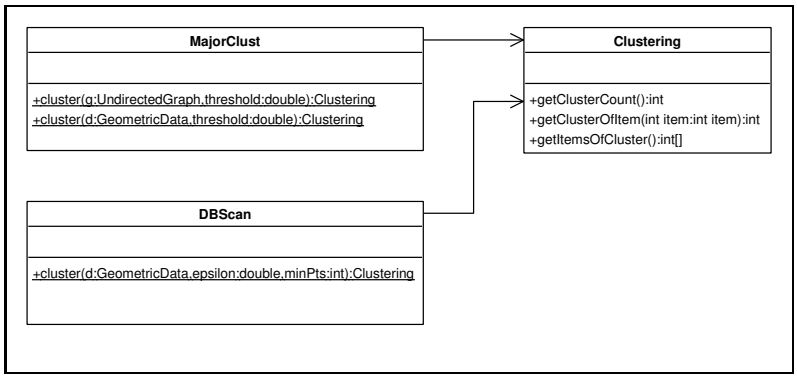

Abbildung 5.4: Klassendiagramm mit den Klassen DBSCAN und MajorClust.

5.2 Implementierung von DBSCAN und MajorClust

DBSCAN

Der Algorithmus DBSCAN wurde im Wesentlichen genauso implementiert, wie es der Pseudocode in Abschnitt 3.2 zeigt. Die Operation `regionQuery` wird von dem R*-Tree ausgeführt, so dass sich eine Laufzeit für DBSCAN von $O(n \log n)$ ergibt.

MajorClust

MajorClust wurde in zwei Varianten implementiert: Variante 1 clustert Ähnlichkeitsdaten, Variante 2 hingegen geometrische Daten. Der wesentliche Unterschied liegt in der Realisierung von Zeile 06 des Pseudocodes: In Variante 2 wird auf die aus DBSCAN bekannte Funktion *regionQuery* zurückgegriffen, welche vom R*-Tree berechnet wird.

Variante 1:

Algorithmus MajorClust1.
Eingabe: Ein gewichteter Graph $G = \langle V, E, w \rangle$, Schwellwert $t \in \mathbb{R}$
Ausgabe: Eine Funktion $c : V \to \mathbb{N}$, die jedem Knoten eine Clusternummer zuordnet.

---

(01) $n := 0, ready := false$
(02) $\forall v \in V$ **do** $n := n + 1, c(v) := n$ **enddo**
(03) **while** $ready = false$ **do**
(04) $ready := true$
(05) $\forall v \in V$ **do**
(06) $c^* := c(v)$
(07) $maxWeight := -\infty$
(08) **for** $i := 1, \ldots, n$ **do** $sum(i) := 0$ **enddo**
(09) $\forall u : e = (v, u) \in E$ **do**
(10) $sum(c(u)) := sum(c(u)) + w(e)$
(11) $value := sum(c(u))$
(12) **if** $value > maxWeight$ **then**
(13) $c^* = c(u)$
(14) $maxWeight := value$
(15) **endif**
(16) **enddo**
(17) **if** $c(v) \neq c^*$ **then** $c(v) := c^*, ready := false$
(18) **enddo**
(19) **enddo**

Variante 2:

Algorithmus MajorClust2.
Eingabe: Punktmenge P, Ähnlichkeitsmaß $d : P \times P \rightarrow \mathbb{R}$, Schwellwert $t \in \mathbb{R}$
Ausgabe: Eine Funktion $c : P \rightarrow \mathbb{N}$, die jedem Punkt eine Clusternummer zuordnet.

---

(01) $n := 0, ready := false$
(02) $\forall p \in P$ **do** $n := n + 1, c(p) := n$ **enddo**
(03) **while** $ready = false$ **do**
(04) $\quad ready := true$
(05) $\quad \forall p \in P$ **do**
(06) $\quad\quad c^* := c(p)$
(07) $\quad\quad maxWeight := -\infty$
(08) $\quad\quad$ **for** $i := 1, \ldots, n$ **do** $sum(i) := 0$ **enddo**
(09) $\quad\quad neighborhood := regionQuery(P, p, t)$
(10) $\quad\quad \forall q \in neighborhood$ **do**
(11) $\quad\quad\quad sum(c(q)) := sum(c(q)) + d(p, q)$
(12) $\quad\quad\quad value := sum(c(q))$
(13) $\quad\quad\quad$ **if** $value > maxWeight$ **then**
(14) $\quad\quad\quad\quad c^* = c(q)$
(15) $\quad\quad\quad\quad maxWeight := value$
(16) $\quad\quad\quad$ **endif**
(17) $\quad\quad$ **enddo**
(18) $\quad\quad$ **if** $c(p) \neq c^*$ **then** $c(p) := c^*, ready := false$
(19) $\quad$ **enddo**
(20) **enddo**

5.3 F-Measure

Das F-Measure ist ein externes Clustervaliditätsmaß. Dieses Maß berechnet die Güte eines Clusterings bezüglich eines Referenzclusterings. Sei D eine Menge von Objekten und sei $\mathcal{C} = \{C_1, \ldots, C_k\}$ ein Clustering von D. Des weiteren sei $\mathcal{C}^* = \{C_1^*, \ldots, C_k^*\}$ ein Referenzclustering, dass z. B. von einem

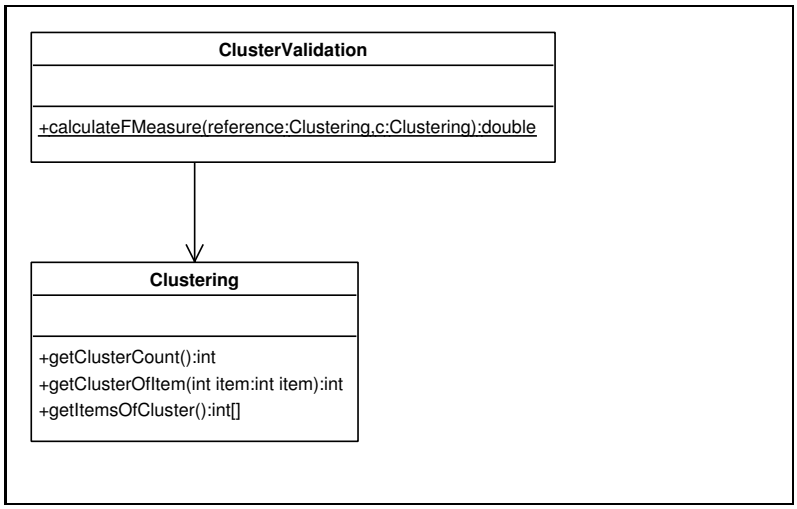

Abbildung 5.5: Klassendiagramm mit der Klasse ClusterValidation zur Berechnung des F-Measures.

Menschen definiert wurde. Dann berechnet das F-Measure $F : \mathcal{C} \times \mathcal{C} \mapsto [0, 1]$ einen Wert zwischen 0 und 1. Ist $F = 1$, dann handelt es sich bei $\mathcal{C}$ um ein perfektes Clustering bezüglich $\mathcal{C}^*$. Geht F hingegen gegen 0, dann handelt es sich bei $\mathcal{C}$ um ein sehr schlechtes Clustering bezüglich $\mathcal{C}^*$.

Das F-Measure kombiniert die aus dem Information Retrieval bekannten Maße *Precision* und *Recall*. Während *Precision* die "Reinheit" eines Clusters bezüglich einer Referenzklasse angibt, berechnet *Recall* den Anteil der Objekte einer Referenzklasse, die in einem bestimmten Cluster enthalten sind.

Definition 15 *(Precision) Der Precision-Wert eines Clusters j bezüglich einer Klasse i ist definiert als*

$$prec(i, j) = \frac{|C_j \cap C_i^*|}{|C_j|}.$$

Definition 16 *(Recall) Der Recall-Wert eines Clusters j bezüglich einer Klasse i ist definiert als*

$$rec(i, j) = \frac{|C_j \cap C_i^*|}{|C^*|}.$$

Das F-Measure kombiniert Precision und Recall durch Bildung des harmonischen Mittels:

$$F_{i,j} = \frac{1}{\frac{1}{2}\left(\frac{1}{prec(i,j)} + \frac{1}{rec(i,j}\right)}$$

Basierend auf dieser Formel kann das F-Measure definiert werden.

Definition 17 *(F-Measure) Das F-Measure eines Clusterings ist definiert als*

$$F = \sum_{i=1}^{l} \frac{|C_i^*|}{|D|} \cdot \max_{j=1,\ldots,k}\{F_{i,j}\}$$

Die Klasse `ClusterValidity` beihaltet eine Implementierung des F-Measures: Der Befehl `calculateFMeasure` liefert den F-Measure-Wert des Clusterings `c` bezüglich des Referenzclusterings `reference` (vgl. Abb. 5.5).

5.4 Zusammenfassung

Dieses Kapitel hat die Testumgebung vorgestellt, mit der die im nächsten Kapitel beschriebenen Experimente durchgeführt werden. Dabei wurden die verwendeten Datenformate sowie die Implementierungsaspekte der beiden Algorithmen anhand von Klassendiagrammen beschrieben. Zuletzt wurde das Clustervaliditätsmaß *F-Measure* eingeführt.

Kapitel 6

Experimente

Dieses Kapitel zeigt verschiedene Experimente, die die Algorithmen DBSCAN und MajorClust miteinander vergleichen. Dabei kommt die in Kapitel 5 vorgestellte Testumgebung zum Einsatz. Es werden drei verschiedene Experimente durchgeführt:

- Abschnitt 6.1 vergleicht die Algorithmen anhand von geometrischen Daten.
- Abschnitt 6.2 vergleicht die Algorithmen anhand von Ähnlichkeitsdaten.
- In Abschnitt 6.3 wird die Performance der Algorithmen gemessen.

6.1 Geometrische Daten

6.1.1 Versuchsbeschreibung

Dieses Experiment vergleicht MajorClust und DBSCAN anhand von geometrischen Daten. Als Testdatensatz wird ein Ausschnitt des Bildes der karibischen Inseln verwendet, der in Abb. 6.1a zu sehen ist. Beide Algorithmen verwenden als Datenstruktur den R*-Tree.

Es wurden zahlreiche Clusteranalysen mit veränderten Eingabeparametern durchgeführt. Abb. 6.1 zeigt eine Auswahl der Ergebnisse, die DBSCAN berechnet hat, während Abb. 6.3 die Ergebnisse von MajorClust präsentiert.

6.1.2 Diskussion der Ergebnisse

DBSCAN

Abb. 6.1b-f zeigen verschiedene Clusterings, die DBSCAN berechnet hat. Dabei wurden unterschiedliche Eingabeparameter ϵ und $MinPts$ verwendet. Wie in den Experimenten festgestellt wurde, beeinflusst vor allem ϵ das Ergebnis der Clusteranalyse. Wird ϵ groß gewählt, d. h. $\epsilon \geq 15$, dann findet DBSCAN keine kleineren Inseln, sondern erzeugt nur einen einzigen Cluster, welcher alle Bildpunkte beinhaltet. Mit kleiner werdenden ϵ werden zunehmend mehr Cluster erzeugt. Dabei ist zu erkennen, dass DBSCAN die einzelnen Inseln sehr präzise findet. Vor allem die Ränder der Inseln werden erkannt. Bei sehr kleinem ϵ hat DBSCAN daher Ähnlichkeit mit dem hierarchisch agglomerativen Single-Linkage-Clusterverfahren, das ebenfalls eine sehr gute Randerkennung aufweist.

Abb. 6.2 soll verdeutlichen, wie unterschiedliche Werte von ϵ das Clusterergebnis beeinflussen. Dafür wird ein Ausschnitt zweier Inseln der Bahamas vergrößert dargestellt. Für einen Randpunkt u, der rot gefärbt ist, sind zwei ϵ-Umgebungen verschiedener Größe eingezeichnet. Im oberen Beispiel, für das $\epsilon = 3$ gilt, ist zu erkennen, dass kein Punkt der grün gefärbten Insel in der ϵ-Umgebung von u liegt. Somit werden die Punkte der beiden Inseln unterschiedlichen Clustern zugeordnet. Im unteren Beispiel ist die ϵ-Umgebung von u mit $\epsilon = 8$ deutlich größer, weshalb sich jetzt der magenta gefärbte Punkt v in der Umgebung befindet. Da u die Kernpunktbedingung erfüllt, wird in Zeile (15) von DBSCAN v in die seed-Liste eingefügt. Auf diese Weise erzeugt DBSCAN nur einen Cluster für die beiden Inseln.

Dieses Experiment zeigt, dass die Wahl der Eingabeparameter großen Einfluss auf das Ergebnis der Clusteranalyse hat. Ester u. a., schlagen in [6] eine Heuristik zur Bestimmung von ϵ und $MinPts$ vor. Diese ist allerdings nur im zweidimensionalen Raum effizient, bei höheren Dimensionen, die z. B. zum Kategorisieren von Dokumenten notwendig sind, ist sie kaum noch anwendbar.

MajorClust

MajorClust wurde in Variante 2 (vgl. Abschnitt 5.2) in Verbindung mit der Datenstruktur R*-Tree angewendet. Abb. 6.3a-f zeigen für verschiedene Schwellwerte t die Clusterings, die MajorClust berechnet hat.

Bei kleinem t betrachtet MajorClust für einen Punkt u, für den der attraktivste Cluster bestimmt werden soll, nur eine kleine Umgebung von u. Daher werden auch zusammenhängende Inseln in eine Vielzahl von Clus-

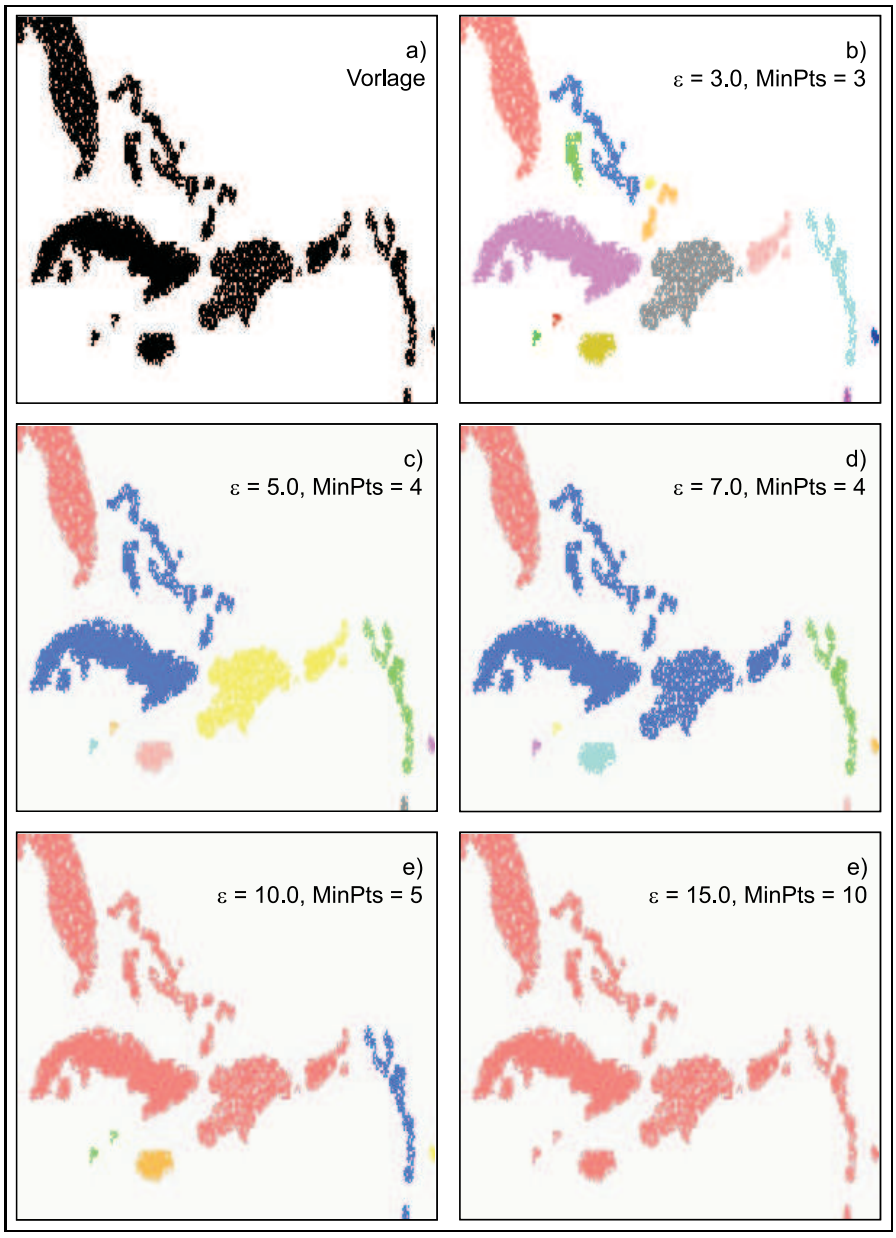

Abbildung 6.1: Clusterergebnisse mit DBSCAN unter Verwendung von geometrischen Daten.

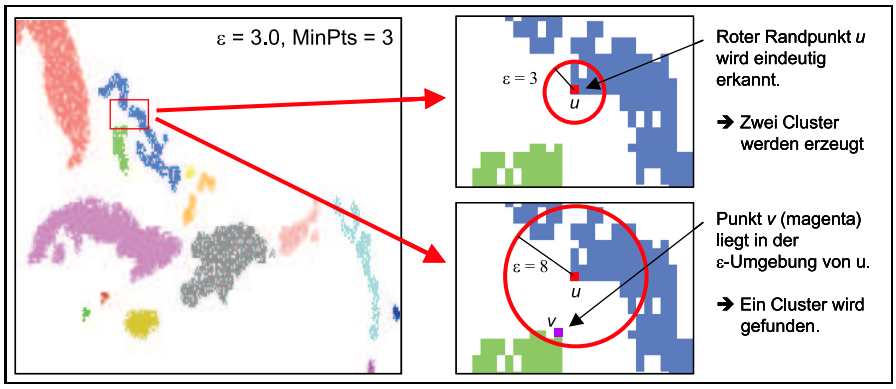

Abbildung 6.2: Eine größere ϵ-Umgebung führt zur Vereinigung nahe gelegener Inseln.

ter aufgeteilt (Abb. 6.3a). Wird t jedoch größer gewählt, dann werden die einzelnen Inseln weitestgehend gefunden (Abb. 6.3e).

Die Randerkennung von MajorClust ist im Vergleich zu DBSCAN jedoch deutlich schlechter. Abb. 6.4 verdeutlicht dieses Phänomen. Für $t = 16$ ist die t-Umgebung des Punktes u dargestellt. Der Cluster der roten Insel weist für u eine größere Attraktivität auf, als der Cluster der blauen Insel, da sich mehr rote Punkte in der Umgebung von u befinden als blaue Punkte. Darüberhinaus sind nur wenige blaue Punkte sehr nahe an u gelegen.

Das Beispiel in Abb. 6.5 zeigt ein Verhalten von MajorClust, dass auf eine dichtebasierte Vorgehensweise schließen lässt. Für den links markierten Bereich sind rechts zwei Dichteschätzungen mit unterschiedlichen "Wasserständen" dargestellt, welche verschiedenen Werten für t entsprechen (vgl. auch Abschnitt 2.5). Bei einem hohen Wasserstand ist die Oberfläche der Dichteschätzung durch Wasser getrennt. Diese Trennung existiert an der gleichen Stelle, an der MajorClust zwei Cluster getrennt hat. Ist das Wasser jedoch weiter abgesenkt (verändertes t), dann erscheint die Oberfläche der Dichteschätzung als zusammenhängend. Auch MajorClust separiert die Insel an der zuvor getrennten Stelle nicht mehr in zwei Cluster, wenn t erhöht wird (vgl. Abb. 6.3d-f).

Dieses Experiment zeigt, dass auch MajorClust brauchbare Ergebnisse beim Clustern von geometrischen Daten liefert. Jedoch ist die Randerkennung schlechter als dies bei DBSCAN der Fall ist. Ebenso wie DBSCAN wird MajorClust stark von der Wahl des Eingabeparameters beeinflusst. Aller-

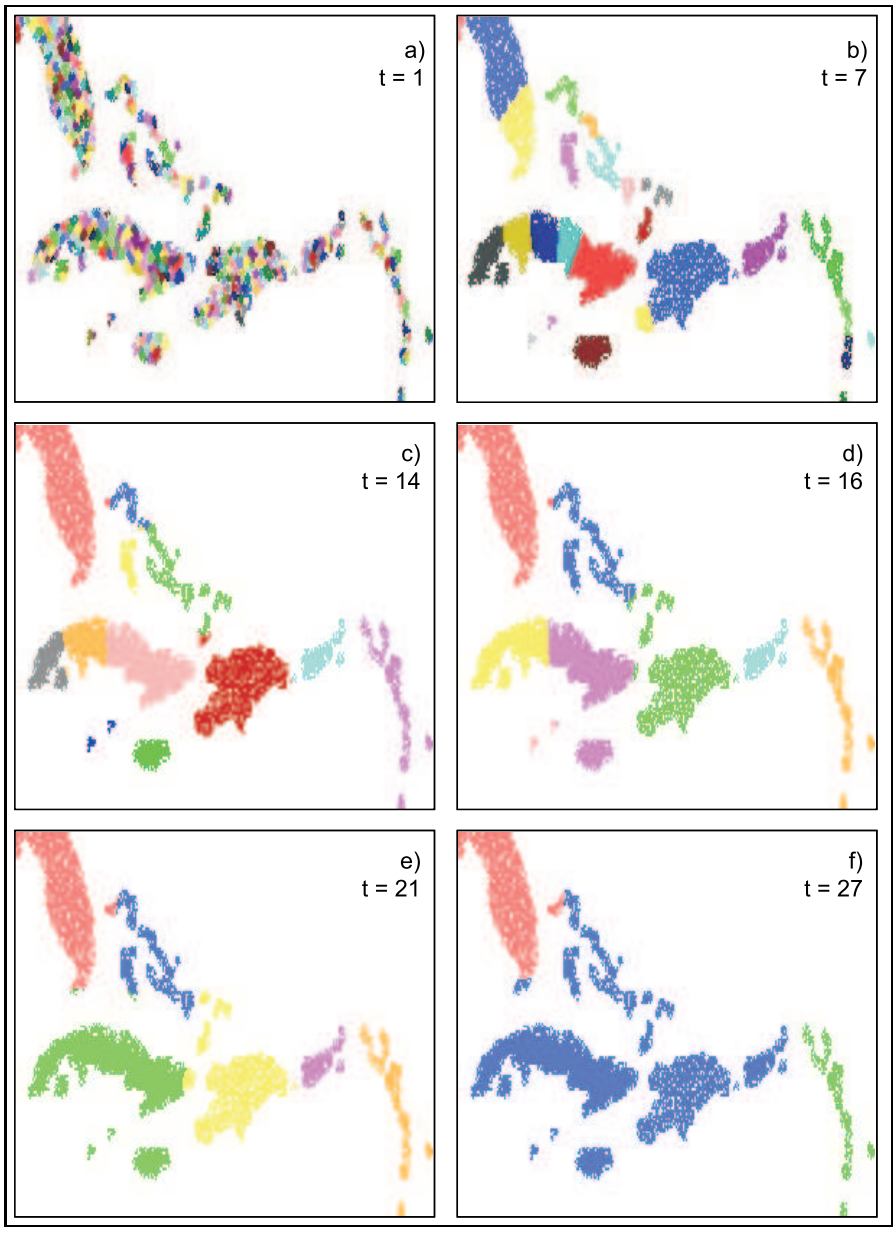

Abbildung 6.3: Clusterergebnisse mit MajorClust unter Verwendung von geometrischen Daten.

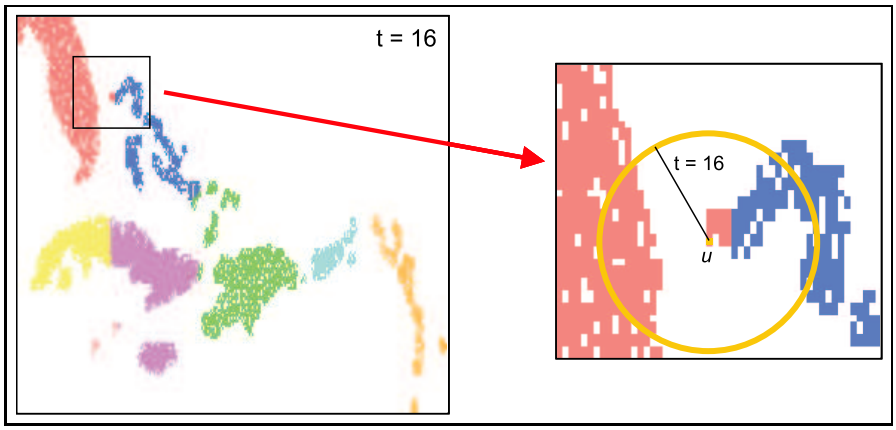

Abbildung 6.4: Schlechte Randerkennung von MajorClust.

dings ist die Ermittlung eines geeigneten Wertes einfacher, da MajorClust mit t nur einen Parameter benötigt.

6.2 Ähnlichkeitsdaten

6.2.1 Versuchsbeschreibung

Dieses Experiment untersucht die Fähigkeit der beiden Algorithmen, Dokumente zu kategorisieren (vgl. Abschnitt 4.3). Dazu wird der Reuters 21578-Korpus verwendet. Dieser beinhaltet mehrere tausend Dokumente, die unterschiedlichen Kategorien zugeordnet sind. Für dieses Experiment werden 1000 Dokumente aus 10 Kategorien des Reuters-Korpus verwendet. Zu jeder Kategorie gehören jeweils 100 Dokumente. Im Korpus existieren verschiedene Dokumente, welche mehreren Kategorien zugeordnet sind. Diese werden für das Experiment nicht verwendet.

Die Dokumente werden tokenisiert und mittels tfidf-Gewichtung vektorisiert. Als Eingabe für MajorClust wird ein vollständiger, ungerichteter Graph erzeugt, welcher für jedes Dokument einen Knoten enthält. Zwischen je zwei Knoten wird eine Kante erzeugt, deren Gewicht der Ähnlichkeit der beiden Dokumente entspricht. Diese Ähnlichkeiten werden mit Hilfe des Kosinusmaßes berechnet. Da DBSCAN keine Ähnlichkeitsdaten verarbeiten kann, werden diese mit Hilfe der multidimensionalen Skalierung (MDS) in den k-dimensionalen Raum eingebettet (vgl. Anhang B). Um den Ein-

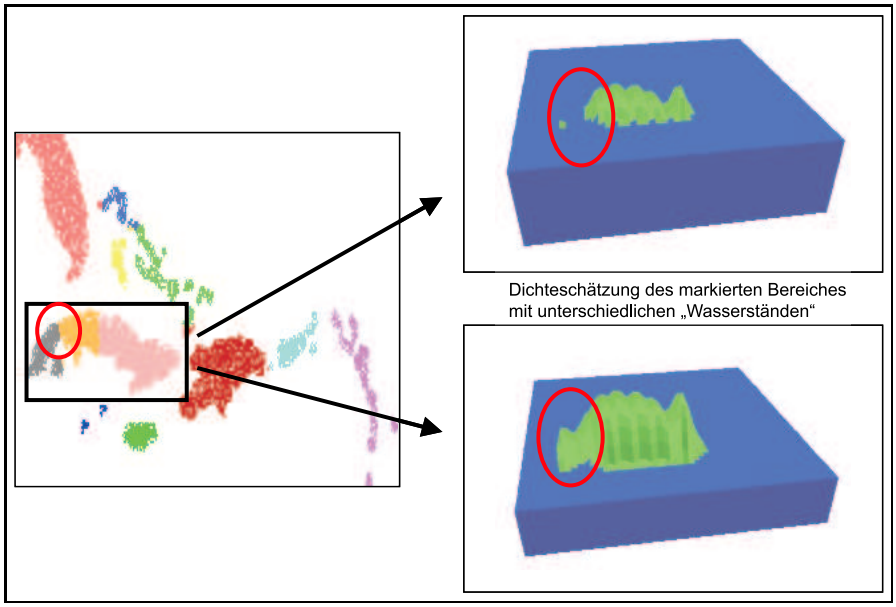

Abbildung 6.5: Cluster werden an Stellen geringer Dichte von MajorClust getrennt.

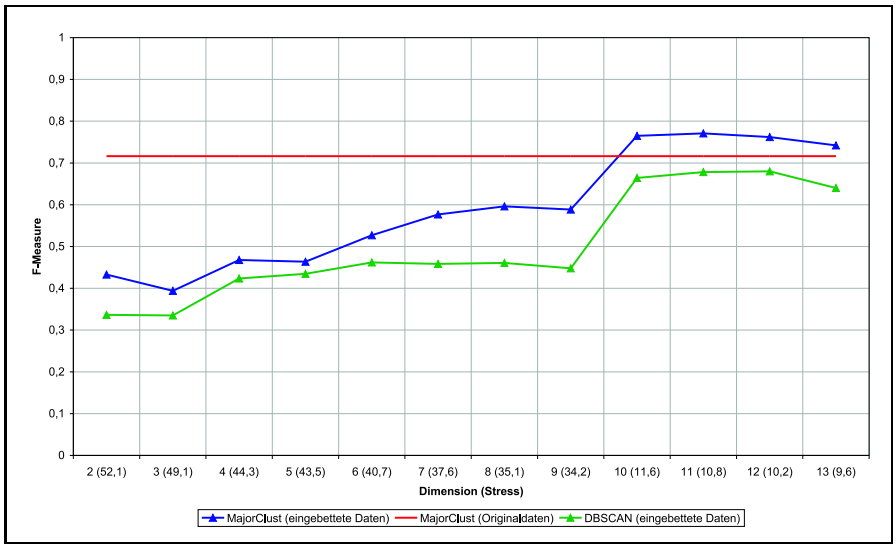

Abbildung 6.6: Clusteringvergleich anhand von Ähnlichkeitsdaten.

fluss der MDS auf das Clusterergebnis zu untersuchen, werden die Daten in Räume unterschiedlicher Dimensionalität eingebettet, d. h. für $2 \leq k \leq 13$.

Die Güte eines Clusterings wird mit Hilfe des F-Measures (vgl. Abschnitt 5.3) berechnet. Dies ist möglich, da als Referenzclustering die Reuters-Kategorisierung bekannt ist. Für jeden Algorithmus wird eine Reihe von Clusteranalysen durchgeführt, wobei jeweils die Eingabeparameter variiert werden. MajorClust wird für jeden Wert von t darüberhinaus dreimal ausgeführt, da dieser Algorithmus in nicht eindeutigen Situationen Zufallswerte verwendet und sich so bei mehrmaligem Ausführen leicht verschiedene Ergebnisse ergeben. Der größte F-Measure-Wert wird als Ergebnis einer Testreihe festgehalten.

Abbildung 6.6 zeigt die Ergebnisse des Experiments.

6.2.2 Diskussion der Ergebnisse

Wie leicht zu sehen ist, dominiert MajorClust diesen Test deutlich. Mit einem F-Measure-Wert von $F = 0.716$ für die Ähnlichkeitsdaten und $F = 0.770$ für die eingebetteten Daten (bei Dimension $k = 11$), liefert MajorClust außerordentlich gute Werte. Auch DBSCAN erzeugt mit einem maximalen $F = 0.680$ (bei Dimension $k = 12$) brauchbare Ergebnisse, fällt jedoch

gegenüber MajorClust deutlich ab.

Zunächst sehr überraschend ist die Tatsache, dass MajorClust bessere Ergebnisse mit den eingebetteten, geometrischen Daten erzielt. Ein Deutung für dieses Phänomen kommt aus einem der multidimensionalen Skalierung verwandten Gebiet, dem *Latend Semantic Indexing*, welches ebenfalls eine Reduktion der Dimensionalität bei möglichst hohem Informationserhalt anstrebt. Wie in [5] erläutert, erzielt die Einbettung der Daten in einen Raum geringerer Dimensionalität eine Verminderung von Rauschen innerhalb der Daten. Wird die Zahl der Dimensionen allerdings noch weiter begrenzt, tritt ein Informationsverlust auf, da nicht mehr genügend Freiheitsgrade verfügbar sind, um alle Informationen unterzubringen. Abb. 6.6 spiegelt diese These genau wieder. Im Bereich von 10 Dimensionen werden die besten Clusterergebnisse erzielt. Dabei sollte der Leser bemerken, dass die Problemdaten aus genau 10 Kategorien stammen. Beim Übergang von 10 zu 9 Dimensionen ist ein großer Abfall der Clusteringqualität zu erkennen, denn 9 Dimensionen reichen nicht mehr aus, um alle Informationen der 10 Kategorien darzustellen. Oberhalb von 11 Dimensionen nimmt der F-Measure-Wert wieder leicht ab. Hier lässt offentsichtlich die Rauschreduktion der multidimensionalen Skalierung nach.

Dieses Experiment hat gezeigt, dass beide Algorithmen sehr gute Ergebnisse bei der Dokumentenkategorisierung liefern. Dabei ist jedoch zu beachten, dass die multidimensionale Skalierung einen erheblichen Rechenaufwand benötigt, vor allem, wenn es sich um eine große Anzahl von Daten und eine hohe Dimensionalität des einzubettenden Raumes handelt. Auf dem verwendeten Testsystem[1] benötigte das “R-Tool“ zur Einbettung von 1000 Objekte in den 13-dimensionalen Raum mehr als 15 Minuten. Somit benötigt die multidimensionale Skalierung mehr Rechenzeit als die eigentliche Clusteranalyse. Deshalb ist DBSCAN kaum für zeitkritsche Anwendungen, wie z. B. die Kategorisierung von Suchergebnissen einer Suchmaschine in Echtzeit, geeignet. Eine Reduktion der Dimensionalität ist für DBSCAN jedoch zwingend erforderlich, da keine bekannte Datenstruktur existiert, die regionQuery-Anfragen im hochdimensionalen Raum effizient berechnen kann.

Da MajorClust Ähnlichkeitsdaten clustern kann und daher keine Einbettung benötigt, ist er prädestiniert für diese Aufgabe, vor allem, da er sogar bei den nicht um Rauschen reduzierten Ähnlichkeitsdaten bessere Ergebnisse liefert, als DBSCAN bei den eingebetteten Daten.

[1] Pentium M Centrino 1,7 GHz mit 1 GB RAM.

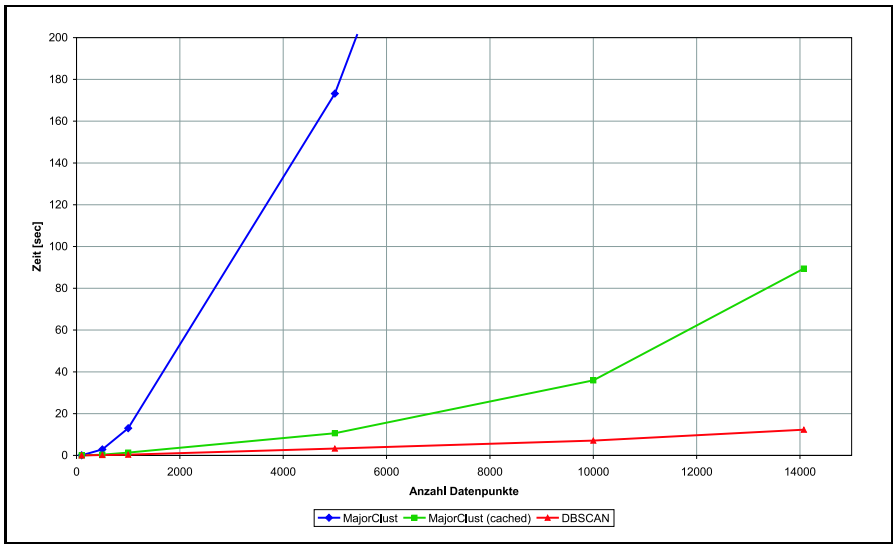

Abbildung 6.7: Performancevergleich.

6.3 Performancevergleich

6.3.1 Versuchsbeschreibung

Im letzten Versuch soll die Performance der beiden Algorithmen verglichen werden. Um Einflüsse der verwendeten Datenstrukturen auszugleichen, werden geometrische Daten verwendet, so dass beide Algorithmen den R*-Tree verwenden können. Es wird $\epsilon = t$ gesetzt, damit beide Algorithmen gleich komplexe Anfragen an den R*-Tree stellen.

Abbildung 6.7 zeigt die benötigte Zeit für eine Clusteranalyse, aufgetragen über der Anzahl der verwendeten Datenpunkte.

6.3.2 Diskussion der Ergebnisse

Abbildung 6.7 zeigt, dass MajorClust (blaue Linie) gegenüber DBSCAN (rote Linie) wesentlich langsamer ist, wenn viele Datenpunkte geclustert werden sollen. Mit einer einfachen Technik kann MajorClust allerdings erheblich beschleunigt werden. Aus der Funktionsweise der Algorithmen erkennt man, dass DBSCAN für jeden Datenpunkt die ϵ-Umgebung nur einmal abfragt. MajorClust hingegen requestiert die t-Umgebung für jeden Datenpunkt un-

ter Umständen mehrmals vom R*-Tree. Eine Beschleunigung kann daher durch das Cachen bereits berechneter t-Umgebungen erzielt werden. Die grüne Linie zeigt, wie stark sich die Verwendung eines solchen Caches auswirkt. Mit dieser Verbesserung ist MajorClust zwar immer noch langsamer als DBSCAN, jedoch in der gleichen Größenordnung.

Diese Egebnisse zeigen, dass DBSCAN schneller als MajorClust arbeitet. Es ist jedoch zu beachten, dass dieses Experiment aufgrund der Verwendung von geometrischen Daten sehr der Funktionsweise von DBSCAN entgegenkommt. Im vorangegangenen Experiment wurde erwähnt, dass DBSCAN für zeitkritische Anwendungen in Verbindung mit Ähnlichkeitsdaten kaum anwendbar ist.

6.4 Zusammenfassung

Dieses Kapitel hat die verschiedenen Experimente zum Vergleich von DBSCAN und MajorClust beschrieben und ausgewertet. Es wurde gezeigt, dass beide Algorithmen unter Verwendung von geometrischen Daten als auch Ähnlichkeitsdaten gute Ergebnisse liefern.

Des Weiteren wurde herausgefunden, dass die multidimensionale Skalierung durch eine Rauschreduktion eine Verbesserung des Clusterergebnisses bei Verwendung von Ähnlichkeitsdaten erzielen kann.

Im letzten Experiment wurde die Performance der Algorithmen verglichen, wobei DBSCAN mit geometrischen Daten etwas schneller war. Es wurde gezeigt, dass MajorClust unter Verwendung eines Caches erheblich beschleunigt werden kann.

Kapitel 7

Ergebnis und Zusammenfassung

An dieser Stelle sollen die in der Motivation gestellten Fragen mit Hilfe der aus den verschiedenen Experimenten gewonnenen Erkenntnissen beantwortet werden.

- Wie verhält sich das Clusterergebnis und die Performance von MajorClust im Vergleich zum dichtebasierten Clusteralgorithmus DBSCAN?

MajorClust liefert gute Ergebnisse beim Clustern von geometrischen Daten, wird jedoch von DBSCAN in der Leistung übertroffen. Letzterer weist eine bessere Randerkennung von Figuren beliebiger Form auf und arbeitet etwas schneller.

Beim Kategorisieren von Dokumenten erreicht MajorClust qualitativ hervorragende Werte, die die sehr gute Leistung von DBSCAN noch deutlich übertreffen. Zudem benötigt MajorClust keine Einbettung der Ähnlichkeitsdaten in einen Raum beschränkter Dimensionalität, weshalb MajorClust hier wesentlich schneller Ergebnisse liefert als DBSCAN.

Zusammenfassend lässt sich daher sagen, dass DBSCAN erste Wahl bei geometrischen Clusteraufgaben ist. Liegen dagegen Problemdaten sehr hoher Dimension vor, sollte MajorClust verwendet werden.

- Deuten die Clusterergebnisse darauf hin, dass MajorClust ebenfalls dichtebasiert arbeitet?

MajorClust weist dichtebasierte Eigenschaften auf: Der Algorithmus liefert gute Ergebnisse beim Clustern von geometrischen Daten, wie z. B. Erkennung von Regionen mit verschieden Formen. Die Eigenschaft, dass MajorClust Regionen an den Stellen in mehrere Cluster aufspaltet, an denen die Dichte der Kerndichteschätzung ein Minimum beschreibt, deutet ebenfalls auf eine dichtebasierte Vorgehensweise hin.

Diese Beobachtungen lassen darauf schließen, dass eine Zuordnung von MajorClust zur Kategorie der dichtebasierten Clusteralgorithmen durchaus sinnvoll ist. Eine theoretische Zurückführung der Funktionsweise von MajorClust auf das dichtebasierte Konzept steht allerdings noch aus.

- Wie stark beeinflusst die Einbettung der ursprünglichen Ähnlichkeitsdaten in den euklidischen Raum das Ergebnis der Clusteranalyse?

Ein überraschendes Ergebnis ergab sich bei der Untersuchung dieser Frage. Zunächst wurde angenommen, dass die Einbettung der Ähnlichkeitsdaten in einen Raum beschränkter Dimensionalität das Clusterergebnis verschlechtern würde. Das Gegenteil jedoch wurde festgestellt, denn die multidimensionale Skalierung erreicht eine Rauschreduktion innerhalb der Daten, wenn die Dimensionalität des Zielraumes die Anzahl der zu erwartenden Cluster nicht unterschreitet.

Eine Einschränkung ist jedoch die Tatsache, dass die multidimensionale Skalierung eine erhebliche Rechenzeit beansprucht, die deutlich größer als die der Clusteranalyse ist. Somit ist die Fähigkeit von MajorClust, Daten beliebiger Dimensionalität effizient clustern zu können, ein großer Vorteil bei zeitkritischen Aufgaben gegenüber Algorithmen wie DBSCAN, die Datenstrukturen benötigen, welche nur mit Daten beschränkter Dimensionalität effizient arbeiten.

7.1 Ausblick

Eine wesentliche Schwierigkeit tritt bei der Verwendung der Clusteralgorithmen auf: Die Ermittlung optimaler Eingabeparameter für die Algorithmen ist mühselig und schwer automatisierbar. Es sollten deshalb leistungsfähige Heuristiken gefunden werden, die bei diesem Problem Hilfestellung leisten.

Eine andere Herausforderung für die Zukunft ist die Verbesserung der Datenstrukturen, die Algorithmen wie DBSCAN benötigen. Der R*-Tree

etwa arbeitet bei mehr als 11 Dimensionen nicht mehr effizient. Ein Schwerpunkt sollte daher auf die Erforschung neuer oder verbesserter Datenstrukturen gelegt werden.

Anhang A

Komplexität

In Kapitel 4 wurde der gewichtete partielle Kantenzusammenhang Λ eingeführt. Ziel bei der Clusteranalyse mit MajorClust ist es, Λ zu maximieren. Zur Zeit wird angenommen, dass das Problem der Maximierung von Λ NP-vollständig ist, ein Beweis jedoch steht noch aus.

Für das verwandte Problem, Λ unter Vorgabe einer festen Clusteranzahl $|C| = k$ zu maximieren, kann die NP-Härte nachgewiesen werden. Dieses Problem wird im Folgenden als Ψ bezeichnet.

Sowohl für Λ, als auch für Ψ, läßt sich ein Entscheidungs- und ein Optimierungsproblem formulieren. Es ergeben sich vier Probleme, die im Folgenden formal definiert werden.

1. Λ_{opt} bezeiche das Optimierungsproblem mit der Maximierungsfunktion "gewichteter partieller Kantenzusammenhang Λ".

2. Ψ_{opt} bezeiche das Optimierungsproblem mit der Maximierungsfunktion "gewichteter partieller Kantenzusammenhang Λ" und fester Clusteranzahl k.

3. $\Lambda_{ent} := \{\langle G, \Lambda_{min}\rangle \mid$ es existiert ein Clustering C des Graphen G mit $\Lambda(C) \geq \Lambda_{min}\}$

4. $\Psi_{ent} := \{\langle G, \Lambda_{min}, k\rangle \mid$ es existiert ein Clustering C mit genau k Cluster des Graphen G mit $\Lambda(C) \geq \Lambda_{min}\}$

Komplexität

Die folgende Reduktion zeigt, wie ähnlich das Problem Ψ_{ent} dem Clique-Problem ist.

Satz 1 Ψ_{ent} ist NP-vollständig.

Es folgt der Beweis von Satz 1.

Lemma 1.1 $\Psi_{ent} \in NP$

Beweis:
Um zu zeigen, dass Ψ_{ent} polynomiell verifizierbar ist und somit in NP liegt, wird im Folgenden ein polynomieller Verifizierer für Ψ_{ent} konstruiert.
Sei V_1 mit Eingabe $\langle G, \Lambda_{min}, k, C\rangle$ ein polynomieller Verifizierer, der testet, ob der Graph $G = (V, E)$ ein Clustering C mit $\Lambda(C) \geq \Lambda_{min}$ besitzt:

Verifizierer V_1.
Eingabe: $\langle G, \Lambda_{min}, k, C\rangle$
Ausgabe: "Eingabe akzeptiert" oder "Eingabe nicht akzeptiert"

---

1. Teste, ob $\langle C\rangle$ die Codierung eines Clusterings $(C_1, ..., C_k)$ von G ist.
2. Setze $x = 0$
3. Für alle $i = 1, ..., k$:
   (a) Berechne die Kantenzusammenhangszahl λ_i für das Cluster C_i.
   (b) Berechne $x = x + |C_i| \cdot \lambda_i$
4. Wenn $x \geq \Lambda_{min}$ akzeptiere, sonst lehne ab.

Um diesen Verifizierer polynomiell auf einer deterministischen Turingmaschine umzusetzen, wird die Codierung eines Clusterings C als Bitfolge der Länge $|V| \cdot \lceil \log(|V|) \rceil$ realisiert. Dabei bezeichnen die Bits $(j-1) \cdot \lceil \log(|V|) \rceil + 1 ... j \cdot \lceil \log(|V|) \rceil$ die Binärdarstellung der Nummer i des Clusters C_i, welchem der Knoten V_j zugeordnet ist (vgl. Abb. A.1). Diese Anzahl ($\lceil \log(|V|) \rceil$) der Bits für die Nummer eines Clusters ist ausreichend, denn ein Graph mit $|V|$

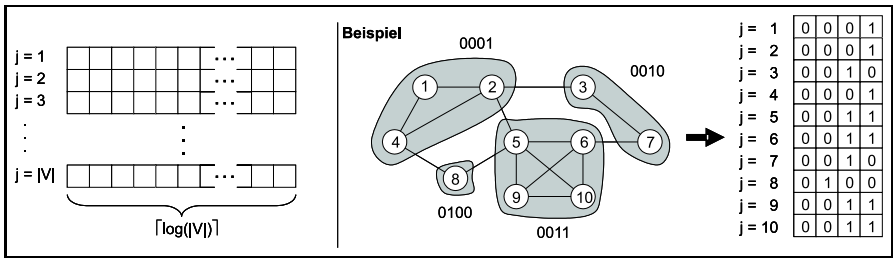

Abbildung A.1: Codierung eines Clusterings.

Knoten kann höchstens in eben soviele Cluster aufgeteilt werden und die Zahl $|V|$ kann binär mit $\lceil \log(|V|) \rceil$ Bits dargestellt werden. Außerdem wird in Schritt 1 überprüft, ob es sich um ein Clustering mit genau k Clustern handelt. Somit kann der erste Schritt des Verifizierers in polynomieller Zeit umgesetzt werden.
Schritt 2 des Verifizierers benötigt nur eine Rechenoperation.
Die Schleife in Schritt 3 wird höchstens $|V|$-mal durchlaufen, nämlich dann, wenn jeder Knoten einem eigenen Cluster zugeordnet ist. Die Kantenzusammenhangszahl λ_i kann etwa mit dem Maximalflussalgorithmus von Ford und Fulkerson in $O(|V|^5)$ berechnet werden (3a). In Schritt 3b der Schleife sind sowohl die Addition als auch die Multiplikation in $O(n^2)$ durchzuführen, wobei n die Anzahl der Stellen der größten der beiden Zahlen bezeichnet. x kann maximal einen Wert von $|V| \cdot (|V| - 1)$ annehmen, und zwar genau dann, wenn der Graph G eine $|V|$-Clique bildet. Somit lässt sich Schritt 3 in $O((|V| \cdot (|V| - 1))^2) = O(|V|^4)$ berechnen.
Der letzte Schritt des Verifizierers benötigt lediglich $O(\log n)$ Rechenoperationen, um zwei Zahlen mit Stellenanzahl von höchstens n miteinander zu vergleichen. Bei einem maximalen Wert von $|V| \cdot (|V| - 1)$ für x (s. o.) benötigt dieser Schritt also höchstens $O(\log(|V| \cdot (|V| - 1))) = O(2 \cdot \log(|V|)$.
Insgesamt ergibt sich also eine Laufzeit für den Verifizierer von
$O(|V| \cdot \lceil \log(|V|) \rceil) + O(1) + O(|V|^6) + O(|V|^5) + O(2 \cdot \log(|V|)$
$= O(|V|^6)$.

Somit ist die Laufzeit polynomiell in $\langle G, \Lambda_{min}, C \rangle$. Daraus folgt, dass V_1 ein polynomieller Verifizierer für Ψ_{ent} ist.

Lemma 1.2 $Clique \leq \Psi_{ent}$

Beweis:

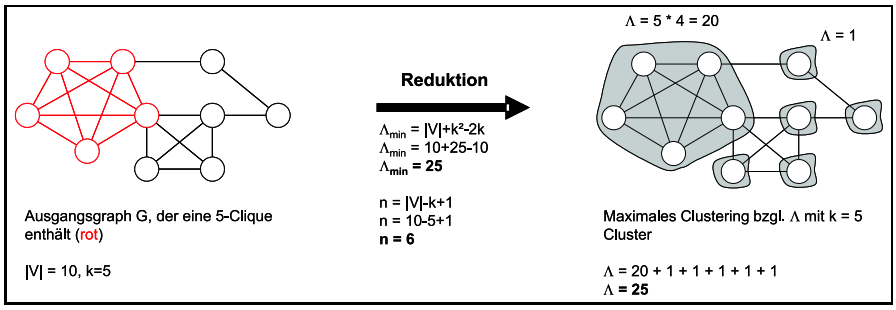

Abbildung A.2: Beispiel für eine Reduktion.

Zunächst wird die Sprache *Clique* definiert.

Definition Clique: Sei $G = (V, E)$ ein ungerichteter Graph. Eine Teilmenge C der Knoten von G heißt *Clique*, wenn für alle $i, j \in C$ gilt $\{i, j\} \in E$. Die Knoten einer Clique sind also alle paarweise durch Kanten im Graphen G verbunden. Eine Clique C heißt k-Clique, wenn C genau k Knoten enthält.
Clique := $\{\langle G, k\rangle |\ G$ ist ein ungerichteter Graph mit einer k-Clique$\}$.

Clique ist NP-vollständig.

Reduktion von Clique auf Ψ_{ent}:
In dieser Reduktion gelte für ein Cluster C mit $|C| = 1 : \Lambda(C) = 1$. Aus der Eingabe $\langle G, k\rangle$ wird im Folgenden die Eingabe $\langle G', \Lambda_{min}, n\rangle$ konstruiert.

- Sei $G' = G$
- Sei $\Lambda_{min} = |V| + k^2 - 2k$
- Sei $n = |V| - k + 1$

Damit ist die Reduktion abgeschlossen.

Zu zeigen:

$$w \in Clique \Leftrightarrow f(w) \in \Psi_{ent}$$

Zunächst "⇒":
Sei G ein Graph, der eine k-Clique enthält, also $(G, k) = w \in Clique$. Dann

muss ein Clustering des Graphen G mit n Cluster gefunden werden mit $\Lambda(G) \geq \Lambda_{min}$. Das erste Cluster C_1 enthalte die k Knoten, die in G eine k-Clique bilden. Dann bleiben noch $|V|-k$ Knoten übrig, die auf die restlichen $n-1$ Cluster verteilt werden. Da $n-1 = |V|-k+1-1 \Rightarrow n-1 = |V|-k$ gibt es ebensoviele nichtleere Cluster wie noch nicht zugeordnete Knoten. Da kein Cluster leer bleiben darf, wird jedem Cluster genau einer dieser Knoten zugeordnet.
Das Cluster C_1 enthält eine k-Clique. Folglich ergibt sich der Wert $\Lambda(C_1) = k \cdot (k-1)$. Für die $n-1$ Cluster, die genau einen Knoten enthalten, gilt $\forall i = 2...n : \Lambda(C_i) = 1$. Insgesamt ergibt sich also $\Lambda(G) = \sum_{i=1}^{n} \Lambda(C_i) = k \cdot (k-1) + \sum_{i=2}^{n} \Lambda(C_i) = k^2 - k + n - 1$. Mit $n = |V| - k + 1$ folgt $\Lambda(G) = k^2 - k + |V| - k + 1 - 1 = |V| + k^2 - 2k = \Lambda_{min}$. Daraus folgt $\Lambda(G) \geq \Lambda_{min} \Rightarrow f(w) \in \Psi_{ent}$.

"$\Leftarrow$":
Sei $(G, \Lambda_{min}, n) \in \Psi_{ent}$. Ein zulässiges Clustering von G muss genau n Cluster enthalten. Da kein Cluster leer bleiben darf, werden n Knoten von G je einem Cluster zugeordnet. Es bleiben $|V| - n = |V| - |V| + k - 1 = k - 1$ Knoten übrig, die noch auf die Cluster verteilt werden müssen. Unter der (im Allgemeinen nicht zutreffenden) Annahme, dass diese $k-1$ Knoten zu allen Knoten aus G eine Kante haben, wird eine Zuordnung der Knoten zu den n Clustern gesucht, die $\Lambda(G)$ maximiert. Das Maxmimum ergibt sich, wenn eine möglichst große Clique gebildet wird. Die $k-1$ Knoten werden daher o.B.d.A. dem Cluster C_1 zugeordnet. Da diesem vorher schon ein Knoten zugeordnet war, enthält es nun k Knoten, die eine k-Clique bilden. Daraus folgt $\Lambda(C_1) = k^2 - k$. Da die anderen $n-1$ Cluster nur einen Knoten enthalten, ergibt sich $\Lambda(G) = k^2 - k + n - 1$. Mit $n = |V| - k + 1$ ergibt sich $\Lambda(G) = |V| + k^2 - 2k = \Lambda_{min}$. Dies ist also der minimale Wert für $\Lambda(G)$ damit $(G, \Lambda_{min}, n) \in \Psi_{ent}$ gilt. Da dieses Clustering von G aber gleichzeitig $\Lambda(G)$ maximiert, ist es das einzig mögliche Clustering. Dann aber muss G eine k-Clique enthalten und $w = (G, k) \in Clique$.

Schließlich muss noch gezeigt werden, dass f polynomiell in $\langle G, k \rangle$ berechenbar ist:
Der Graph G kann in Schritt 1 in $O(|V| + |E|)$ kopiert werden.
Die Berechnungen in den Schritten 2 und 3 können in $O(max(\log(|V|), \log(k)))$ berechnet werden. Da $|V| \geq k$ also in $O(\log(|V|)$.
Somit ergibt sich als Gesamtlaufzeit $O(|V|+|E|)+O(\log(|V|) = O(|V|+|E|)$.

Damit ist f eine polynomielle Reduktion und Lemma 1.2 ist bewiesen.

Aus Lemmata 1.1 und 1.2 folgt Satz 1.

Anhang B

Multidimensionale Skalierung

Die multidimensionale Skalierung (MDS) ist eine Technik zur Einbettung von Daten in den euklidischen Raum, für die lediglich paarweise Ähnlichkeiten in Form eines Ähnlichkeitsmaßes vorliegen (vgl. Abschnitt 1.3). Dabei sollen die Daten so in den euklidischen Raum R^k eingebettet werden, dass die paarweisen Ähnlichkeiten der Objekte möglichst gut durch die paarweisen Distanzen im euklidischen Raum dargestellt werden. Oft wird die MDS auch zur Visualisierung von Daten verwendet (mit $k = 2$ oder $k = 3$).

Erstmals wurde von J. B. Kruskal ein Algorithmus zur Durchführung der MDS vorgeschlagen. Dieser versucht durch sukzessives Repositionieren der Objektentsprechungen $x_1, \ldots, x_n \in R^k$ im euklidischen Raum die Darstellung stetig zu verbessern. Um zu ermitteln, wie gut diese Objektentsprechungen $x_1, \ldots, x_n$ die Ähnlichkeiten $\delta_{i,j}$ der Originaldaten approximieren, werden sog. Stress-Maße der Form

$$S(x_1, \ldots, x_n) = \left(\sum_{i<j} (\delta_{i,j} - d_{i,j})^2 \right)^{1/2}$$

verwendet, wobei $d_{i,j} = ||x_i - x_y||$ eine Metrik im euklidischen Raum zur Bestimmung der Distanz zwischen x_i und x_j bezeichnet. Hohe Stress-Werte kennzeichnen eine schlechte Approximierung durch die Objektentsprechungen.

Kruskal schlug zwei verschiedene Stress-Maße vor, welche einige Verbes-

serungen beinhalten:

$$S_1(x_1, \ldots, x_n) = \left(\frac{\sum_{i<j} (\hat{\delta}_{i,j} - d_{i,j})^2}{\sum_{i<j} d_{i,j}^2} \right)^{1/2}$$

und

$$S_2(x_1, \ldots, x_n) = \left(\frac{\sum_{i<j} (\hat{\delta}_{i,j} - d_{i,j})^2}{\sum_{i<j} (d_{i,j} - \bar{d})^2} \right)^{1/2},$$

wobei $\bar{d}$ den Mittelwert aller Distanzen $d_{i,j}$ bezeichnet.

Es gibt zwei wesentliche Vorteile dieser Stress-Maße. Zum einen werden die Stress-Werte normiert, um sie vergleichbar mit Werten anderer MDS-Daten zu machen. Des Weiteren werden statt Ähnlichkeiten $\delta_{i,j}$ die sogenannten Disparitäten $\hat{\delta}_{i,j} = f(\delta_{i,j})$ zur Berechnung verwendet. Die Funktion f wird aus dem den Daten zugrunde liegenden Modell abgeleitet. Die Verwendung von Disparitäten statt Ähnlichkeitswerten hat den Vorteil, dass die Differenzen $(\delta_{i,j} - d_{i,j})$ nicht absolut berechnet werden müssen.

Um die Güte von mit der MDS skalierten Daten anhand dieser Stress-Maße schätzen zu können hat Kruskal einige Erfahrungswerte zu deren Beurteilung vorgeschlagen. Tabelle B.1 stellt diese Werte dar.

Anpassungsgüte	S_1	S_2
gering	$0,2$	$0,4$
ausreichend	$0,1$	$0,2$
gut	$0,05$	$0,1$
ausgezeichnet	$0,025$	$0,15$
perfekt	0	0

Tabelle B.1: Kruskals Erfahrungswerte zur Beurteilung von Stress-Werten

Literaturverzeichnis

[1] Michael R. Anderberg.
Cluster Analysis for Applications.
Academic Press, 1973

[2] Klaus Backhaus u. a.
Multivariate Analysemethoden. Eine anwendungsorientierte Einführung. Achte Auflage.
Springer Verlag, 1996.

[3] Ricardo Baeza-Yates and Berthier Ribeiro-Neto.
Modern Information Retrieval.
Addison-Wesley, 1999.

[4] Norbert Beckmann, Hans-Peter Kriegel.
An Efficient and Robust Access Method. for Points and Rectangles+.
ACM-SIGMOD, 1990.

[5] S. Deerwester u. a.
Indexing by latent semantic analysis.
Journal of the American Society for Information Science, 1990.

[6] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu.
A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise.
Int. Conf. on Knowledge Discovery and Data Mining (KDD), 1996.

[7] Vladimir Estivill-Castro.
Why so many clustering algorithms - A Position Paper.
ACM SIGKDD Explorations Newsletter Pages 65-75, 2002.

[8] A. Guttman.
R-trees: A dynamic index structure for spatial searching.
Proceedings of ACM SIGMOD International Conference on Management of Data, 1984.

[9] Robert Hafner.
Nichtparametrische Verfahren der Statistik.
Springer-Verlag Wien, 2001.

[10] Steffen Kohler.
Nichtparametrische Dichteschätzung. Seminararbeit.
Universität Tübingen, 2001.

[11] Dr. Hannelore Liero.
Statistische Datenanalyse. Vorlesung.
Universität Potsdam, 2004.

[12] Dirk Loss.
Data Mining: Klassifikations- und Clusteringverfahren. Ausarbeitung im Rahmen des Projektseminars "CRM für Finanzdienstleister".
Westfälische Wilhelms-Universität Münster, 2002.

[13] Stuart Russell and Peter Norvig.
Artificial Intelligence - A Modern Approach. Second Edition.
Prentice Hall, 2003.

[14] The R Project for Statistical Computing.
www.r-project.org.

[15] Spatial Index Library
http://www.cs.ucr.edu/ marioh/spatialindex.

[16] Benno Stein und Oliver Niggemann.
On the Nature of Structure and Its Identification.
Widmayer u. a., LNCS 1665.
Springer-Verlag Heidelberg, 1999.

[17] Benno Stein und Sven Meyer zu Eissen.
AISEARCH: Category Formation of Web Search Results.
Universität Paderborn, 2003.

[18] Benno Stein und Sven Meyer zu Eissen.
Automatic Document Categorization. Interpreting the Performance of Clustering Algorithms.
Universität Paderborn, 2003.

[19] Thorsten Thadewald.
Uni- und bivariate Dichteschätzung. Wissenschaftliche Dissertation.
Freie Universität Berlin, 1998.

[20] Eduard Wiebe.
Dichtebasierte Clustering Verfahren in der industriellen Bildverarbeitung. Diplomarbeit.

Universität Paderborn, 2004.

[21] Wikipedia. Die freie Enzyklopädie.
www.wikipedia.de.