Christian Böhm, Karin Kailing, Peer Kröger, Hans-Peter Kriegel:

Immer größere und komplexere Datenmengen: Herausforderungen für Clustering-Algorithmen

1 Einleitung

Innovative Technologien und neueste Methoden zur Datengewinnung in verschiedenen Teilbereichen der Wissenschaft wie z.B. Biowissenschaften, Medizin, Astronomie, Geographie, erzeugen eine wahre Flut an Rohdaten. Um dieser Flut Herr werden zu können, werden dringend adäquate – d.h. effiziente und effektive – Methoden zur automatischen Datenanalyse und zur Wissensextraktion (Data Mining) benötigt. Data Mining ist ein Teilbereich des »Knowledge Discovery in Databases« (KDD), dessen Ziel es ist, (semi-)automatisch Wissen aus Datenbanken zu extrahieren, das gültig (im statistischen Sinne), bisher unbekannt und potentiell nützlich ist [FPSS96].

Ein äußerst wichtiges Teilproblem des Data Mining ist das Clustering. Beim Clustering sollen die Objekte einer Datenbank in (a priori unbekannte) Gruppen, auch Cluster genannt, eingeteilt werden, sodass zwei Objekte aus einem gleichen Cluster möglichst ähnlich zueinander, und zwei Objekte aus unterschiedlichen Clustern möglichst unähnlich zueinander sind. Ein deutschsprachiger Überblick über das Thema findet sich zum Beispiel in [ES00].

1.1 Clustering-Verfahren

Die existierenden Clustering-Verfahren lassen sich grob in partitionierende, dichte-basierte und hierarchische Verfahren einteilen. Partitionierende Verfahren zerlegen eine Datenmenge in k Cluster, wobei jeder Cluster mindestens ein Objekt enthält und jedes Objekt genau zu einem Cluster gehört. Bekannte Vertreter sind Verfahren, die auf der Konstruktion zentraler Objekte (z.B. k-Means [McQ67]) oder der Auswahl repräsentativer Objekte basieren (z.B. PAM [KR90], CLARANS [NH94]). K-Means beginnt mit einer »initialen« Zerlegung der Datenmenge, üblicherweise mit einer zufälligen Einteilung der Daten in k Klassen. Dann werden die Zentroide dieser Klassen berechnet. Diese Einteilung ist im Allgemeinen nicht optimal, d.h. für einige Punkte ist der Abstand zum Zentroid ihres Clusters größer als der Abstand zum Zentroid eines anderen Clusters. In einem zweiten Schritt werden diese Punkte dem nächstgelegenen Zentroid zugeordnet und der Zentroid jeweils neu berechnet. Dieser Schritt wird solange wiederholt, bis sich die Cluster nicht mehr ändern. Der Algorithmus konvergiert gegen ein (möglicherweise nur lokales) Optimum und hat eine Laufzeitkomplexität von O(n) für eine Iteration, wobei die Anzahl der Iterationen üblicherweise klein (ca. 5-10) ist. Ergebnis und Laufzeit hängen jedoch stark von der initialen Zerlegung ab. Cluster können auch als Gebiete im d-dimensionalen Raum angesehen werden, in denen die Objekte dicht beieinander liegen, getrennt durch Gebiete, in denen die Objekte weniger dicht liegen. Dichte-basierte Clustering-Verfahren (z.B. DBSCAN [EKSX96], OPTICS [ABKS99]) versuchen, solche dichten Gebiete im Raum zu identifizieren. Grundidee für einen dichte-basierten Cluster ist, dass die lokale Punktdichte bei Objekten innerhalb eines Clusters einen gegebenen Grenzwert überschreitet. Die lokale Punktdichte eines Objekts o ist dabei gegeben durch die Anzahl der Objekte in einer festgelegten Umgebung um das Objekt o. Zusätzlich ist die Menge von Objekten eines Clusters räumlich zusammenhängend. DBSCAN [EKSX96] findet in einem Durchlauf durch die Datenbank alle dichte-basierten Cluster. Durch diese Formalisierung werden auch willkürlich geformte Cluster gefunden und Rauschen wird explizit definiert als die Menge aller Objekte, die zu keinem dichte-basierten Cluster gehören. Im Gegensatz zu partitionierenden Verfahren erzeugen hierarchische Verfahren keine einfache Zerlegung der Daten, sondern eine hierarchische Repräsentation der Daten, aus der man eine Clusterstruktur ableiten kann, beispielsweise Single-Link [JD88] und deren Varianten sowie OPTICS [ABKS99]. Eines der bekanntesten hierarchischen Clustering-Verfahren ist das so genannte Single-Link-Clustering. Dieses Verfahren erfordert keinerlei Eingabeparameter wie die Anzahl der zu findenden Cluster oder eine Grenzdichte. Dafür benötigt das Verfahren zusätzlich eine Distanzfunktion für Mengen von Objekten. Bei Single-Link ist der Abstand von zwei Objektmengen gegeben durch die minimale Distanz zweier beliebiger Objekte aus den beiden Mengen. Die hierarchische Clusterstruktur wird dann durch ein so genanntes Dendrogramm repräsentiert. Die Wurzel repräsentiert einen einzigen Cluster, der die gesamte Datenmenge enthält; die Blätter des Baumes repräsentieren Cluster, in denen sich je ein einzelnes Objekt der Datenmenge befindet. Ein innerer Knoten repräsentiert die Vereinigung aller Sohnknoten und die Kante zwischen einem Knoten und seinem Sohnknoten hat als Attribut noch die Distanz zwischen den beiden repräsentierten Mengen von Objekten. OPTICS ist eine hierarchische Version des dichte-basierten Verfahrens DBSCAN. Der Algorithmus basiert auf dem dichte-verbundenen Clustermodell und erzeugt in einem Datenbankdurchlauf ein DBSCAN-Clustering für alle möglichen Dichteparameter. Im Gegensatz zu DBSCAN weist OPTICS allerdings nicht jedem Objekt der Datenbank eine Clusterzugehörigkeit zu, sondern aggregiert für jedes Objekt weitere Informationen, die für die Darstellung der hierarchischen Clusterstruktur verwendet werden können. OPTICS erzeugt kein Dendrogramm, sondern eine auf den aggregierten Informationen basierende Clusterordnung, die auch noch bei sehr großen Datenmengen eine übersichtliche Darstellung der Clusterhierarchie ermöglicht.

2 Leistungssteigerung für große Datenmengen

2.1 Indexstrukturen

Zahlreiche Clustering-Methoden nutzen Distanzprädikate auf Objekten in ihren Algorithmen. So wird beispielsweise bei k-Means jeder Punkt der Datenbank demjenigen Cluster-Repräsentanten zu-

geordnet, zu dem er die geringste Distanz aufweist. Bei Single-Link und verwandten Verfahren werden in jedem Schritt die beiden Objekte bzw. (Teil-) Cluster zu einem neuen Cluster verschmolzen, die die geringste Distanz zueinander aufweisen. Dichtebasiertes Clustering wie DBSCAN oder OPTICS beruht u.a. auf der Zusammenfassung von Punkten, die einen vorgegebenen Höchstabstand nicht überschreiten. Bei allen diesen Verfahren ist es nahe liegend, die Berechnung des Distanzprädikates nicht auf allen Paaren von Punkten durchzuführen, sondern mittels geeigneter Datenbank-Grundoperationen wie der Ähnlichkeitsanfrage, einem Distanz-Ranking oder dem Similarity Join (Ähnlichkeits-Verbund), auf eine möglichst geringe Anzahl passender Kandidaten einzuschränken. Wichtigste Grundlage für eine effiziente Auswertung aller genannten Datenbank-Grundoperationen ist eine geeignete Indexstruktur für hochdimensionale Vektorräume. Die meisten dieser Zugriffsstrukturen organisieren die Daten hierarchisch in balancierten Bäumen und gründen somit auf den Prinzipien des B-Baums [BM72] (eindimensional) und des R-Baums [Gut84] (mehrdimensional, aber aufgrund seines Leistungsverhaltens beschränkt auf niedrigdimensionale Vektorräume). Strukturen wie der TV-tree [LJF95], der X-tree [BKK96], der SS-tree [WJ96] usw. ordnen jedem Knoten des Baums eine Seite des Hintergrundspeichers sowie eine Region des Datenraums zu. Diese Region schließt die im jeweiligen Teilbaum gespeicherten Vektoren sowie die Regionen untergeordneter Knoten ein. Die einzelnen Ansätze unterscheiden sich sowohl in der Geometrie der Regionen als auch in der algorithmischen Vorgehensweise, wie Knoten bei einem Überlauf aufgeteilt werden. So verwendet etwa der X-tree als Region einen minimal umgebenden Hyperquader um die approximierte Vektormenge. Der SS-tree dagegen nutzt (hyper-) kugelförmige Regionen.

Anfragebearbeitung

Bei einfachen Ähnlichkeitsanfragen unterscheidet man insbesondere Bereichsanfragen (range query) und Nächste-Nachbarn-Anfragen (nearest neighbor query). Bereichsanfragen, bei denen mit Hilfe eines Mittelpunkts und eines Radius ε eine Anfragekugel spezifiziert wird, werden meist mittels einer Tiefensuche im Baum bearbeitet: alle Knoten des Baumes werden traversiert, deren zugeordnete Region einen nicht leeren Schnitt mit der Anfragekugel aufweist. Bei einer Nächste-Nachbarn-Anfrage ist ein Anfragepunkt und eine Anzahl k von gesuchten Punkten spezifiziert. Sie kann ebenfalls mittels einer Tiefensuche im Baum implementiert werden [Hen94, RKV95] oder mithilfe einer Prioritäts-Warteschlange, in der die Kind-Knoten der bereits bearbeiteten Knoten nach dem Abstand zum Anfragepunkt sortiert werden [HS95]. Von letzterem Algorithmus wurde gezeigt, dass er bei gegebenem Index eine minimale Anzahl von Knotenzugriffen benötigt [BBKK97]. Die Kosten der Anfragebearbeitung können für beide Anfragetypen weiter reduziert werden, wenn man berücksichtigt, dass Knoten, die auf benachbarten Seiten auf dem Externspeicher abgelegt sind, kostengünstiger in einer gemeinsamen Leseoperation geladen werden. Dieser Effekt kann für einen Fast Index Scan ausgenutzt werden, lohnt sich aber nur bei sorgfältiger Planung der Reihenfolge der Zugriffe – eine relativ einfache Aufgabe bei Bereichsanfragen. Bei Nächste-Nachbarn-Anfragen hingegen wird für diesen Planungsvorgang ein Wahrscheinlichkeitsmodell benötigt, das situationsbedingt die Wahrscheinlichkeit schätzt, dass ein Knoten im späteren Verlauf des Algorithmus benötigt wird [BBJ+00].

Hochdimensionale Räume

Die hierarchische Strukturierung der Daten in einer Baumstruktur führt im Idealfall dazu, dass bei Bereichs- und Nächste-Nachbarn-Anfragen nur wenige Pfade von der Wurzel bis zu einem Blattknoten besucht werden müssen, und dass sich in der Folge die Komplexität einer solchen Anfrage von linear (bei einer sequenziellen Anfragebearbeitung) auf logarithmisch reduziert. Bei kritischer Betrachtung kann dies aber nur bei Datenräumen gelingen, die entweder bereits aufgrund der Anzahl der Merkmale niedrigdimensional sind, oder zumindest so dünn besetzt sind, dass infolge von Merkmals-Abhängigkeiten die eigentliche Dimensionalität des besetzten Datenraums niedrig genug ist. Diese eigentliche Dimensionalität lässt sich z.B. mit dem Konzept der fraktalen Dimension formalisieren [PKF00]. Bei hochdimensionalen, vollbesetzten Datenräumen (z.B. bei unabhängig gleichverteilten Merkmalen) gibt es den Effekt, dass sich im Allgemeinen die Anfragekugel in vielen Dimensionen mit mindestens zwei im Vektorraum benachbarten Regionen schneidet, wodurch sich insgesamt ein Schnitt mit annähernd 2^d Regionen ergibt (vgl. Abb. 1 in 3D). Da auf jede Seite maximal einmal zugegriffen wird, führt dies nicht zu einem tatsächlich exponentiellem Wachstum, aber der Effekt einer Reduktion zu logarithmischer Komplexität geht ab Dimensionen von ca. 10-15 wieder verloren. Genauer werden diese Effekte in [BBKK97] und [Böh00] beschrieben, wo ein Kostenmodell für Bereichs- und Nächste-Nachbarn-Anfragen für den X-tree vorgeschlagen wird. Dieses Kostenmodell basiert auf dem Prinzip der Minkowski-Summe (vgl. Abb. 2) und wurde zur Optimierung zahlreicher Parameter wie etwa der logischen Seitengröße

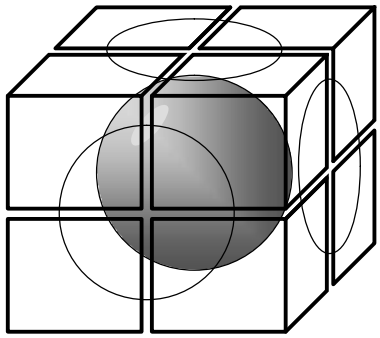

Abb. 1: Schnitt einer Anfragekugel

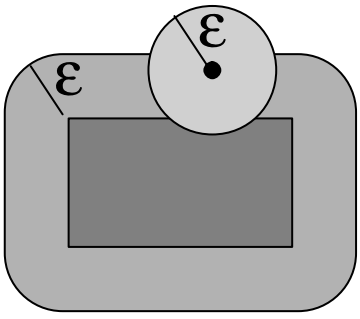

Abb. 2: Minkowski-Summe

[BK00] oder der Dimension [BBK+00] benutzt. Die Minkowski-Summe vergrößert das Volumen der Seitenregion in der dargestellten Weise um die Anfrageregion. Das resultierende Volumen entspricht der Wahrscheinlichkeit eines Zugriffs auf die Seite. Basierend auf der Überlegung, dass hierarchische Indexstrukturen ab einer gewissen Dimension ohnehin keine Effizienzsteigerungen mehr bewirken können, wurde als Alternative eine Verbesserung der sequenziellen Anfragebearbeitung mit Hilfe einer Datenkompressionsmethode vorgeschlagen [WSB98]. Idee ist, den gesamten Datenraum mit einem auf Quantilen basierenden Quantisierungsgitter zu überziehen. Dieses Gitter erlaubt eine effiziente Codierung der ungefähren Lage der Datenpunkte (Vector Approximation, VA-file), was in einem Filterschritt genutzt werden kann. Das VA-file unterliegt zwar kaum dem Curse of Dimensionality, die Zeitkomplexität bei der Anfragebearbeitung ist jedoch schon in niedrigdimensionalen oder dünn besetzten Datenräumen linear. Der IQ-tree [BBJ+00] integriert eine ähnliche Idee zur Vektor-Quantisierung in eine hierarchische Indexstruktur, wobei die Auflösung des Quantisierungsgitters – ein sehr kritischer Parameter – während der Einfügeoperationen automatisch angepasst und gemäß Kostenmodell optimiert wird. Zur Optimierung der Anfragebearbeitung für Bereichs- und Nächste-Nachbarn-Anfragen dient der bereits oben diskutierte Fast Index Scan. Auch parallele und verteilte Architekturen für Indexstrukturen wurden vorgeschlagen [BBB+97, DS82, FB93].

2.2 Der Similarity Join

Data-Mining-Algorithmen stellen während ihres Laufs typischerweise eine sehr hohe Anzahl an Ähnlichkeitsanfragen. So wird etwa bei DBSCAN für jeden Punkt der Datenbank je eine Bereichsanfrage mit dem Radius ε ausgewertet. Zunächst war die Idee, die Auswertung zu verzögern, Ähnlichkeitsanfragen zu sammeln, und schließlich in einem gemeinsamen Durchlauf durch den Index auszuwerten [BEKS00]. Die konsequente Weiterentwicklung dieser Idee führt zum Similarity-Join, einer Datenbank-Grundoperation, bei der die Unterscheidung zwischen Anfrage-Objekten und Datenbank-Objekten fallen gelassen wird. In seiner einfachsten Form ermittelt der Similarity Join die Menge aller Punkt-Paare aus zwei Datenmengen (bzw. einer einzigen Datenmenge beim Similarity Self Join), die voneinander einen Abstand von höchstens ε aufweisen. Fortgeschrittene Algorithmen für den Similarity Join können jedoch auch auf verschiedenen Nächste-Nachbarn-Prädikaten beruhen. Möchte man einen Clustering-Algorithmus mit Hilfe eines Similarity Joins implementieren, so ist dies häufig mit einem hohen Aufwand verbunden, da die Algorithmen für diesen Zweck meist deutlich umformuliert werden müssen. Diesem höheren Aufwand stehen dann aber als Vorteile sehr hohe Effizienzgewinne, ein großes Optimierungspotenzial seitens des Datenbanksystems und eine einfache Parallelisierbarkeit gegenüber.

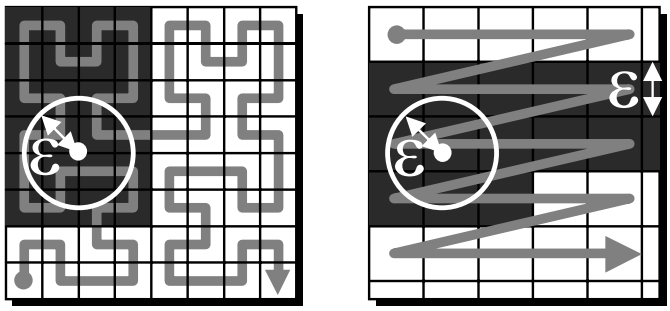

Abb. 3: Raumfüllende Kurven

In [BBBK00] wurden die beiden dichte-basierten Clustering-Algorithmen DBSCAN und OPTICS so reformuliert, dass die Bereichsanfragen, die im ursprünglichen Algorithmus einzeln für jeden Datenpunkt gestellt werden, durch einen einzigen Aufruf (bei DBSCAN durch zwei Aufrufe) des Similarity Join ersetzt werden. Abgesehen von einem deutlich reduzierten Overhead, der bei den einzelnen Bereichsanfragen mit dem häufigen Kontextwechsel zwischen der Anwendung und dem Datenbank-Managementsystem verbunden ist, ergibt sich ein besonders hohes Leistungspotenzial durch verbesserte Index Traversierungsstrategien für den Similarity Join. Bereits mit einer relativ einfachen Depth-First-Strategie für den X-tree konnte eine Leistungssteigerung um den Faktor 7 bei OPTICS sowie um den Faktor 33 bei DBSCAN erreicht werden. Darüber hinaus wurden in der Literatur zahlreiche Verfahren und Varianten für die Bearbeitung eines Similarity Joins vorgeschlagen. Darunter

- Algorithmen, die auf vorkonstruierten Indexstrukturen für hochdimensionale Räume aufsetzen [BKS93, HJR97]
- Algorithmen, die spezialisierte Indexstrukturen für den Similarity Join konstruieren [LR94, vdBSS00, SSA97, BK01]
- Algorithmen, die auf speziellen Sortierordnungen wie etwa raumfüllenden Kurven [Ore89, KS98] oder einer gitter-basierten Ordnung [BBKK01] beruhen (vgl. Abb. 3).

Besonders interessant sind hierbei gerade auch parallele und verteilte Algorithmen wie etwa [BKS96] oder [SA97], da sich hierdurch implizit eine Parallelisierung des Clustering-Algorithmus ergibt. Der zusätzliche Implementierungsaufwand bei der Parallelisierung ist hier also mit der Umformulierung des Algorithmus auf der Basis des Similarity Join bereits antizipiert.

Im Gegensatz zu dichte-basiertem Clustering bauen Algorithmen wie k-Means oder Single-Link eher auf Nächsten-Nachbarn-Prädikaten als auf Bereichs-Anfragen auf. Eine entsprechende Join-Operation wurde in [BK03] vorgeschlagen. Algorithmen für den Nearest Neighbor Join sind etwa [HS98] oder [BK02].

2.3 Datenkompression

Eine beliebte, effizienzsteigernde Maß-

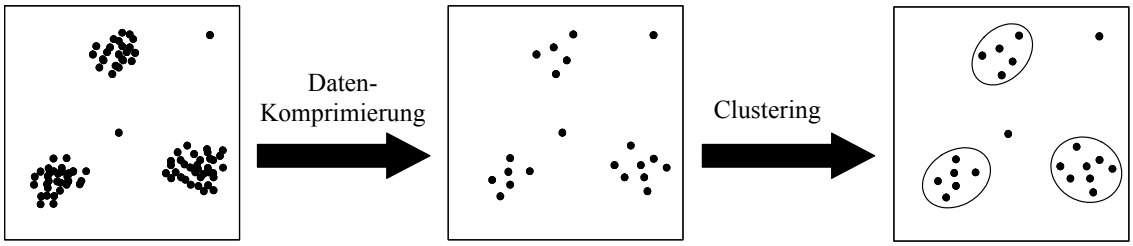

Abb. 3: Effizientes Clustern durch Datenkompression

nahme bei großen Datenmengen sind Techniken zur Datenkompression. Grundidee ist hier, das Clustering nicht für die gesamte Datenbank zu berechnen, sondern für eine deutlich kleinere Menge an repräsentativen Objekten (siehe Abb. 4). In einem Nachbearbeitungsschritt muss dann das Clustering der Repräsentanten auf die Gesamtmenge übertragen werden. Da der Schritt des Clusterings auf der Gesamtmenge üblicherweise teuerer ist als Sampling, Clustering auf dem Sample und Nachbearbeitung zusammen, kann man mit dieser Strategie enorme Laufzeitverbesserungen erzielen. In den letzten Jahren haben sich verschiedene Techniken zur Datenkompression etabliert. Im folgenden werden beispielhaft drei Techniken vorgestellt.

Sampling

Beim »Sampling« wird die Anzahl der Objekte einer Datenbank verringert, indem man nach geeigneten Kriterien bestimmte Objekte auswählt. Das einfachste Auswahlkriterium ist dabei der Zufall (»random sampling«). Nachteil der zufälligen Auswahl ist, dass die Clusterstruktur in den Daten nicht berücksichtigt wird. In [EKX95] wird daher ein indexbasiertes Sampling vorgeschlagen: Mit den zu clusternden Daten wird eine räumliche Indexstruktur (z.B. ein R*-Baum [BKSS90]) erstellt. Von den Datenseiten des Index werden ein oder mehrere Repräsentanten extrahiert, und das Clustering-Verfahren wird auf diese Repräsentanten angewendet. Da die Indexstruktur eine gewisse gruppierende Eigenschaft besitzt, berücksichtigt diese Auswahlstrategie die Clusterstruktur der Daten deutlich besser. Sampling-Techniken sind meist problemlos auf alle Arten von Clustering-Verfahren anwendbar.

BIRCH

Bei BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) [ZRM96] werden zunächst kompakte Beschreibungen von Objektmengen durch sog. »Clustering Features (CF)« generiert. Diese CFs erlauben die effiziente Berechnung allgemeiner Informationen wie Zentroid, Radius und Durchmesser der repräsentierten Teilmenge von Objekten. Zudem werden die CFs hierarchisch in einem höhenbalancierten Baum organisiert. Ein innerer Knoten repräsentiert dabei einen (Sub-)Cluster, der wiederum aus den Sub-Clustern zusammengesetzt ist, die durch seine Einträge (Teilbäume) repräsentiert sind. Der Baum der CFs stellt eine komprimierte, hierarchische Repräsentation der Daten dar. Die Einfügealgorithmen zum Aufbau des Baumes stellen sicher, dass die Clusterstruktur der Daten berücksichtigt bleibt. Nach Erstellung des Baumes kann ein Clustering-Verfahren auf die Blätter des Baumes angewendet werden. Laufzeit und Qualität von BIRCH sind mit dem indexbasierten Sampling vergleichbar.

Data Bubbles

Eine Weiterentwicklung der Daten-Kompressionstechnik BIRCH stellen die sogenannten Data Bubbles [BKKS01] dar. Es werden weitere Informationen wie z.B. die durchschnittliche k-Nächste-Nachbarn-Distanz aller repräsentierten Objekte aggregiert. Außerdem definieren die Autoren in [BKKS01] eine Distanzfunktion auf Data Bubbles. Diese zusätzlichen Informationen und Konzepte sind speziell für hierarchische Clustering-Verfahren wichtig. Daher eignet sich diese Datenkompressionstechnik speziell für diese Klasse von Cluster-Algorithmen. Data Bubbles können auf zwei unterschiedliche Arten erzeugt werden: (1) Man kann aus den Daten einen CF-Baum (siehe vorangehender Abschnitt »BIRCH«) aufbauen, diesen an einem beliebigen Level abschneiden und aus den Blättern relativ einfach die Bubbles erzeugen, indem man die zusätzlichen Informationen aggregiert. (2) Man kann eine bestimmte Anzahl an Objekten zufällig aus der Datenbank wählen (bei k Objekten erhält man k Data Bubbles), alle übrigen Objekte demnächstgelegenen zuordnen, und die entsprechenden Informationen generieren. Eine Datenkompression mit Hilfe der Technik Data Bubbles bringt speziell in Verbindung mit dem dichte-basierten, hierarchischen Clustering-Verfahren OPTICS enorme Performanzsteigerungen bei sehr geringen Qualitätsverlusten.

2.4 Inkrementelles Clustering

Wenn es kontinuierliche Updates der Datenbank gibt, d.h. Einfügungen oder Löschungen, dann verändert sich im Allgemeinen auch das Clustering. Dabei ist es wünschenswert, dass die Aktualisierung der Clusterstruktur inkrementell erfolgt. Ein erneutes Clustering der gesamten Datenmenge nach jedem Update ist in der Regel zu aufwendig. Die inkrementelle Version des dichtebasierten Clustering-Verfahrens DBSCAN (Incremental DBSCAN) [EKS+98] nützt aus, dass sich Änderungen der Datenbank nur lokal auf die Clusterstruktur auswirken. Die meisten Objekte behalten ihren Status bei. Die Effizienzsteigerung durch Incremental DBSCAN bewegen sich im Bereich um den Faktor 20 bei 25.000 Update-Operationen [EKS+98]. Bei der inkrementellen Version des dichte-basierten hierarchischen Verfahrens OPTICS (Incremental OPTICS) [KKG03], können die Ände-

rungen in der hierarchischen Clusterstruktur im schlimmsten Fall Auswirkungen auf alle Objekte der Datenbank haben. Dennoch ist es üblicherweise deutlich effizienter, die hierarchische Clusterstruktur inkrementell zu aktualisieren, statt von Grund auf neu zu berechnen.

3 Hochdimensionale Daten

Zusätzlich zur steigenden Anzahl an Daten, nimmt auch die Komplexität der gespeicherten Daten immer weiter zu. Dank moderner Datenbanktechnologien (z.B. objekt-relationale Datenbanken) lassen sich heute beliebig komplexe Objekte in großen Mengen verwalten. Die Modellierung solch komplexer Objekte führt häufig zu sehr hochdimensionalen Merkmalsvektoren. Ein Beispiel hierfür sind Microarray-Daten: abhängig vom Ziel des Clusterings kann sich die Dimensionalität hier zwischen 10^2 (Clustering der Gene) und 10^3 bis 10^4 (Clustering der Testpersonen) bewegen. Solche hochdimensionalen Merkmalsräume stellen meist große Probleme für Clustering-Verfahren dar.

Hochdimensionale Merkmalsräume verursachen im Kontext Clustering eine Menge von grundsätzlichen Problemen, die mit dem Ausdruck »Curse of Dimensionality« (Fluch der Dimensionalität) zusammengefasst werden. Die Folge ist, dass die Objekte in hochdimensionalen Räumen stark streuen, und nicht mehr sinnvoll zu clustern sind (siehe z.B. [BGRS99]). Dennoch gibt es meistens Cluster in verschiedenen Teilräumen niedrigerer Dimensionalität, d.h. die Daten clustern, wenn man gewisse (teilweise unterschiedliche) Attribute ausblendet. Erschwerend kommt hinzu, dass Objekte in verschiedenen Teilräumen unterschiedlich clustern können, z.B. Objekt A clustert in einem Teilraum mit den Objekt B und C, in einem anderen Teilraum aber nicht mit B und C, sondern mit den Objekten D und E. Offensichtlich kann man traditionelle Clustering-Verfahren für diese Probleme nicht verwenden. Daher wurden in den letzten Jahren spezielle Verfahren entwickelt, die auf das Problem des Clusterings hochdimensionaler Daten zugeschnitten sind.

Projected Clustering

Beim Projected Clustering ist das Ziel, die Datenmenge wie gehabt in Cluster zu partitionieren, sodass jedem Objekt eine Cluster-ID zugeordnet wird. Zusätzlich darf aber jeder Cluster in einem unterschiedlichen Teilraum des Merkmalsraumes existieren, d.h. die Cluster müssen typischerweise ein Clusterkriterium optimieren, unterschiedliche Cluster dürfen dies aber in unterschiedlichen Teilräumen. Einer der ersten Ansätze zum Projected Clustering ist das Verfahren PROCLUS [AP99]. Das Ziel von PROCLUS ist, die Datenobjekte in k Cluster (k ist ein Eingabeparameter) ähnlich wie k-Means zu partitionieren. Jeder Cluster optimiert dabei ein Clusterkriterium in einer achsenparallelen Projektion (Teilraum) des Merkmalsraumes. Eine Verbesserung von PROCLUS stellt ORCLUS [AY00] dar: Cluster optimieren ein Clusterkriterium nun auch in nicht achsenparallelen Projektionen des Datenraumes. Sowohl PROCLUS als auch ORCLUS haben Probleme mit Rauschen (Ausreißer-Objekte, die eigentlich zu keinem Cluster gehören), da alle Objekte einem der k Cluster zugeordnet werden müssen. Ein weiterer Nachteil des Projected Clustering Ansatzes ist, dass die Informationen über Objekte, die in verschiedenen Teilräumen unterschiedlich clustern, nicht generiert werden können. Dem steht die Eindeutigkeit der Clusterzugehörigkeit als möglicher Vorteil gegenüber.

Subspace Clustering

Bei einem Merkmalsraum mit d Merkmalen gibt es 2^d mögliche Teilräume, in denen Cluster verborgen sein können. Die Aufgabenstellung beim Subspace Clustering ist, automatisch alle Cluster in allen Teilräumen zu ermitteln. Aus Performanzgründen ist dabei natürlich wichtig, möglichst wenig Teilräume zu betrachten, optimalerweise nur die, in denen auch tatsächlich Cluster vorhanden sind. Die erste Arbeit zum Subspace Clustering ist das Verfahren CLIQUE [AGGR98]. Der d-dimensionale Datenraum wird von CLIQUE mit Hilfe eines achsenparallelen, equidistanten Gitters in Zellen zerlegt. Eine Zelle ist dicht, wenn eine Mindestanzahl an Objekten in der Zelle liegt. Benachbart gelegene dichte Zellen werden zu Clusternverschmolzen. CLIQUE startet mit der Erzeugung aller dichten Zellen in allen 1-dimensionalen Teilräumen und generiert nun schrittweise aus allen k-dimensionalen dichten Zellen die potentiellen (k + 1)-dimensionalen dichten Zellen (Kandidaten). Am Ende jeder Iteration müssen diese Kandidaten noch überprüft werden, ob sie tatsächlich dicht sind. CLIQUE terminiert, wenn keine neuen Kandidaten erzeugt werden können. Um möglichst wenig Kandidaten zu erzeugen (d.h. wenige Kandidaten überprüfen zu müssen, und nach möglichst wenigen Iterationen abzubrechen), benutzt CLIQUE die sog. Monotonie-Eigenschaft für dichte Zellen: Ist eine Zelle in einem (k + 1)-dimensionalen Merkmalsraum T dicht, so ist die Zelle auch in allen k-dimensionalen Teilräumen von T dicht. Die umgekehrte Folgerung besagt, dass (k + 1)-dimensionale Zellen, die aus mindestens einer nicht dichten k-dimensionalen Zelle entstanden sind, nicht dicht sein können, und daher nicht mehr überprüft werden müssen. Anschließend an die Berechnung aller dichten Zellen, müssen benachbarte Zellen im selben Teilraum in einer aufwendigen Prozedur zu Clustern verschmolzen werden.

Eine mathematische Definition »optimaler« Subspace Cluster bieten die Autoren in [PJAM02]. Zudem wird ein heuristischer Algorithmus namens DOC angegeben, um Näherungen solcher »optimalen« Subspace Cluster zu berechnen. DOC ist nicht gitterbasiert, allerdings werden die »optimalen« Subspace Cluster mit Hilfe von Hyper-Würfeln fester Kantenlänge, in denen eine gewisse Anzahl an Punkten liegen muss, definiert. Da wiederum keine Angaben über die Punkteverteilung innerhalb der Hyper-Würfel gemacht werden, hat DOC ähnliche Nachteile wie CLIQUE: Cluster, die größer sind als die Hyper-Würfel werden nur unvollständig erkannt. Rauschpunkte, die in einem dichten Hyper-Würfel liegen, werden fälschlicherweise zu einem Cluster gezählt.

Eine etwas andere Strategie verfolgt RIS [KKKW03]. Anstatt direkt die Subspace Cluster zu erzeugen, berechnet RIS

alle Teilräume des Merkmalsraumes, in denen Cluster enthalten sind und ordnet sie mit Hilfe eines Qualitätsmaßes. Dieses Qualitätskriterium basiert auf dem dichte-basierten Clustermodell von DBSCAN [EKSX96]. Die Auswahl der Teilräume anhand des Qualitätskriteriums unterliegt einem ähnlichen Monotonieverhalten wie im Falle von CLIQUE, sodass RIS ebenfalls Teilräume effizient ausschließen kann. Die eigentlichen Cluster kann der Anwender in einem Nachbearbeitungsschritt mit jedem beliebigen Clustering-Verfahren auf den gefundenen Teilräumen erzeugen. RIS ist daher kein Subspace Clustering-Verfahren im engeren Sinne, sondern vielmehr ein Verfahren zur Merkmalsauswahl (»Feature Selection«), das die Informationen über Objekte, die in verschiedenen Teilräumen unterschiedlich clustern, konserviert. RIS vermeidet die Nachteile der gitterbasierten Ansätze, da das Qualitätskriterium auf dem dichte-basierten Clustermodell von DBSCAN beruht, das Cluster unterschiedlicher Form erkennt und nicht anfällig gegen Ausreißer ist (Rauschen wird explizit behandelt). Nachteil aller Subspace-Clustering-Verfahren ist die Sensitivität gegenüber der Parameterwahl. Alle Verfahren arbeiten mit einem globalen Dichtegrenzwert. Verschieden dichte Cluster in einem Teilraum können ggf. nicht mit einer Parametereinstellung erkannt werden. Hinzu kommt, dass niedriger dimensionale Teilräume typischerweise bevorzugt werden, da Cluster in höher-dimensionalen Teilräumen naturgemäß weniger dicht sind und daher den globalen Dichtegrenzwert nicht mehr erfüllen.

4 Diskussion und Ausblick

Aufgrund der immer größer werdenden Datenberge ist für reale Anwendungen die Skalierbarkeit von Clustering-Verfahren in Bezug auf die Datenbankgröße eines der Kernprobleme. Da die durch das Data Mining gewonnenen Informationen immer öfter Firmen oder Wissenschaftlern einen signifikanten Wettbewerbsvorteil verschaffen, ist die Effizienz von Clustering-Verfahren auch ein wichtiger finanzieller Aspekt. Ein weiteres Kernproblem für das Clustering ist die Tatsache, dass die zu betrachtenden Objekte immer komplexer werden. Als Folge werden die Daten oft als sehr hochdimensionale Merkmalsvektoren modelliert. Hochdimensionale Merkmalsräume bergen aber große Probleme für Clustering-Verfahren. In diesem Artikel wurden verschiedene Methoden dargestellt, die speziell für dieses Kernproblem entwickelt wurden.

Ein aktuelles, offenes Forschungsgebiet ist die effiziente Unterstützung partieller Ähnlichkeitsanfragen für Subspace und Projected Clustering, siehe Abschnitt 3.1. Traditionelle räumliche Indexstrukturen (Abschnitt 2.1) sind nicht dafür entwickelt, Anfragen in verschiedenen Teilräumen des Merkmalsraumes effizient zu beantworten. Des weiteren macht es keinen Sinn, für jeden Teilraum, der betrachtet wird, einen eigenen Index zu erzeugen. Um die Performanz von Subspace und Projected Clustering-Verfahren zu erhöhen, werden daher Indexstrukturen gebraucht, die Ähnlichkeitsanfragen wie z.B. Bereichsanfragen oder k-Nächste-Nachbarn-Anfragen in unterschiedlichen Teilräumen eines Merkmalsraumes effizient unterstützen können.

Oft reicht die traditionelle Modellierung der Daten als Merkmalsvektoren nicht mehr aus, um die intuitive Ähnlichkeit der Objekte adäquat auszudrücken. Komplexere Modellierungswerkzeuge wie Bäume oder Graphen sind dann notwendig, um die Semantik der Objekte sinnvoll zu modellieren. Die Entwicklung von Clusteringalgorithmen für komplex modellierte Objekte ist ein weiteres hoch relevantes Forschungsgebiet.

Referenzen

[ABKS99] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. OPTICS: Ordering Points to Identify the Clustering Structure. ACM SIGMOD Int. Conf. on Management of Data, 1999.

[AGGR98] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications. ACM SIGMOD Int. Conf. on Management of Data, 1998.

[AP99] C. C. Aggarwal and C. Procopiuc. Fast Algorithms for Projected Clustering. ACM SIGMOD Int. Conf. on Management of Data, 1999.

[AY00] C.C. Aggarwal and P.S. Yu. Finding Generalized Projected Clusters in High Dimensional Space. ACM SIGMOD Int. Conf. on Management of Data, 2000.

[BBB+97] S. Berchtold, C. Böhm, B. Braunmüller, D.A. Keim, and H.-P. Kriegel. Parallel Similarity Search in Multimedia Databases. ACM SIGMOD Int. Conf. on Management of Data, 1997.

[BBBK00] C. Böhm, B. Braunmüller, M. Breunig, and H.P. Kriegel. High Performance Clustering Based on the Similarity Join. In Int. Conf. on Information and Knowledge Management (CIKM), 2000.

[BBJ+00] S. Berchtold, C. Böhm, H.V. Jagadish, H.-P. Kriegel, and J. Sander. Independent quantization: An index compression technique for high-dimensional data spaces. Int. Conf. on Data Engineering (ICDE), 2000.

[BBK+00] S. Berchtold, C. Böhm, D.A. Keim, H.-P. Kriegel, and X. Xu. Optimal Multidimensional Query Processing Using Tree Striping. Int. Conf. on Data Warehousing and Knowledge Discovery (DaWaK), 2000.

[BBKK97] S. Berchtold, C. Böhm, D.A Keim, and H.-P. Kriegel. A Cost Model For Nearest Neighbor Search in High-Dimensional Data Space. ACM PODS Symp. on Principles of Database Systems, 1997.

[BBKK01] C. Böhm, B. Braunmüller, F. Krebs, and H.-P. Kriegel. Epsilon Grid Order: An Algorithm for the Similarity Join on Massive High-Dimensional Data. ACM SIGMOD Int. Conf. on Management of Data, 2001.

[BEKS00] B. Braunmüller, M. Ester, H.-P. Kriegel, and J. Sander. Efficiently Supporting Multiple Similarity Queries for Mining in Metric Databases. Int. Conf. on Data Engineering (ICDE), 2000.

[BGRS99] K.S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When Is Nearest Neighbor Meaningful?. Int. Conf. on Database Theory (ICDT), 1999.

[BK00] C. Böhm and H.-P. Kriegel. Dynamically Optimizing High-Dimensional Index Structures. Int. Conf. on Extending Database Technology (EDBT), 2000.

[BK01] C. Böhm and H.-P. Kriegel. A Cost Model and Index Architecture for the Similarity Join. Int. Conf. on Data Engineering (ICDE), 2001.

[BK02] C. Böhm and F. Krebs. High Performance Data Mining Using the Nearest Neighbor Join. Int. Conf. on Data Mining (ICDM), 2002.

[BK03] C. Böhm and F. Krebs. Supporting KDD Applications by the k-Nearest Neighbor Join. Int. Conf on Database and Expert Systems Applications (DEXA), 2003.

[BKK96] S. Berchtold, D.A. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. Int. Conf. on Very Large Databases (VLDB), 1996.

[BKKS01] M.M. Breunig, H.-P. Kriegel, P. Kröger, and J. Sander. Data Bubbles: Quality Preserving Performance Boosting for Hierarchical Clustering. ACM SIGMOD Int. Conf. on Management of Data, 2001.

[BKS93] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Efficient Processing of Spatial Joins Using R-trees. ACM SIGMOD Int. Conf. on

Management of Data, 1993.
[BKS96] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Parallel Processing of Spatial Joins Using Rtrees. Int. Conf. on Data Engineering (ICDE), 1996.
[BKSS90] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. ACM SIGMOD Int. Conf. on Managment of Data, 1990.
[BM72] R. Bayer and E.M. McCreight. Organization and maintenance of large ordered indices. Acta Informatica, 1(3), 1972.
[Böh00] C. Böhm. A Cost Model for Query Processing in High-Dimensional Data Spaces. ACM Transactions on Database Systems (TODS), 25(2), 2000.
[DS82] H.C. Du and J.S. Sobolewski. Disk allocation for cartesian product files on multiple Disk systems. ACM Transactions on Database Systems (TODS), 1982.
[EKS+98] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu. Incremental Clustering for Mining in a Data Warehousing Environment. Int. Conf. on Very Large Databases (VLDB), 1998.
[EKSX96] 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.
[EKX95] M. Ester, H.-P. Kriegel, and X. Xu. A Database Interface for Clustering in Large Spatial Databases. Int. Conf. on Knowledge Discovery and Data Mining (KDD), 1995.
[ES00] M. Ester and J. Sander. Knowledge Discovery in Databases. Springer, 2000.
[FB93] C. Faloutsos and P. Bhagwat. Declustering Using Fractals. Journal of Parallel and Distributed Information Systems (PDIS), 1993.
[FPSS96] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. Knowledge Discovery and Data Mining: Towards a Unifying Framework. Int. Conf. on Knowledge Discovery and Data Mining (KDD), 1996.
[Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. ACM SIGMOD Int. Conf. on Managment of Data, 1984.
[Hen94] A. Henrich. A distance-scan algorithm for spatial access structures. ACM Workshop on Advances in Geographic Information Systems, 1994.
[HJR97] Y.-W. Huang, N. Jing, and E.A.. Rundensteiner. Spatial Joins Using R-trees: BreadthFirst Traversal with Global Optimizations. Int. Conf. on Very Large Databases (VLDB), 1997.
[HS95] G.R. Hjaltason and H. Samet. Ranking in spatial databases. In Max J. Egenhofer and John R. Herring, editors, Advances in Spatial Databases (SSD), 1995.
[HS98] G.R. Hjaltason and H. Samet. Incremental Distance Join Algorithms for Spatial Databases. ACM SIGMOD Int. Conf. on Management of Data, 1998.
[JD88] A.K. Jain and R.C. Dubes. Algorithms for Clustering Data. Prentice Hall, 1988.
[KKG03] H.-P.. Kriegel, P. Kröger, and I. Gotlibovich. Incremental OPTICS: Efficient Computation of Updates in a Hierarchical Cluster Ordering. Int. Conf. on Data Warehousing and Knowledge Discovery (DaWaK), 2003.
[KKKW03] K. Kailing, H.-P. Kriegel, P. Kröger, and S. Wanka. Ranking Interesting Subspaces for Clustering High Dimensional Data. Eur. Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD), 2003.
[KR90] L. Kaufman and P.J. Rousseeuw. Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley & Sons, 1990.
[KS98] N. Koudas and C. Sevcik. High Dimensional Similarity Joins: Algorithms and Performance Evaluation. Int. Conf on Data Engineering (ICDE), 1998.
[LJF95] K. Lin, H.V. Jagadish, and C. Faloutos. The TV-Tree: An Index Structure for HighDimensional Data. VLDB Journal, 3, 1995.
[LR94] M.-L. Lo and C.V. Ravishankar. Spatial Joins Using Seeded Trees. ACM SIGMOD Int. Conf. on Management of Data, 1994.
[McQ67] J. McQueen. Some Methods for Classification and Analysis of Multivariate Observations. Symp. Math. Statist. Prob., 1, 1967.
[NH94] R.T. Ng and J. Han. Efficient and Effective Clustering Methods for Spatial Data Mining. Int. Conf. on Very Large Data Bases (VLDB), 1994.
[Ore89] J.A. Orenstein. Redundancy in Spatial Databases. ACM SIGMOD Int. Conf. on Management of Data, 1989.
[PJAM02] C.M. Procopiuc, M. Jones, P.K. Agarwal, and T.M. Murali. A Monte Carlo Algorithm for Fast Projective Clustering. ACM SIGMOD Int. Conf. on Management of Data, 2002.
[PKF00] B.-U. Pagel, F. Korn, and C. Faloutsos. Deflating the Dimensionality Curse Using Multiple Fractal Dimensions. Int. Conf. on Data Engineering (ICDE), 2000.
[RKV95] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. ACM SIGMOD Int. Conf. on Management of Data, 1995.
[SA97] J. C. Shafer and R. Agrawal. Parallel Algorithms for High-dimensional Proximity Joins. Int. Conf. on Very Large Data Bases (VLDB), 1997.
[SSA97] K. Shim, R. Srikant, and R. Agrawal. High-dimensional Similarity Joins. Int. Conf. on Data Engineering (ICDE), 1997.
[vdBSS00] J. van den Bercken, M. Schneider, and B. Seeger. Plug&Join: An Easy-to-Use Generic Algorithm for Efficiently Processing Equi and Non-equi Joins. Int. Conf. on Extending Database Technology (EDBT), 2000.
[WJ96] D.A. White and R. Jain. Similarity indexing with the SS-tree. Int. Conf. on Data Engineering (ICDE), 1996.
[WSB98] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. Int. Conf. on Very Large Databases (VLDB), 1998.
[ZRM96] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. ACM SIGMOD Int. Conf. on Management of Data, 1996.

Christian Böhm ist Professor für Praktische Informatik an der Ludwig-Maximilians-Universität München und erforscht Indexstrukturen für die Ähnlichkeitssuche sowie Data-Mining-Algorithmen. Er ist Autor von ca. 40 wissenschaftlichen Publikationen.

Karin Kailing arbeitet als wissenschaftliche Mitarbeiterin in der Gruppe von Prof. Hans-Peter Kriegel im Bereich Knowledge Discovery in Databases. Ihre Dissertation über Cluster-Verfahren für komplexe Objekte wird im Sommer eingereicht.

Peer Kröger arbeitet ebenfalls als wissenschaftlicher Mitarbeiter in der Gruppe von Prof. H.-P. Kriegel im Bereich Knowledge Discovery in Datenbanken. Seine Dissertation mit dem Titel "Coping with New Challenges for Density-Based Clustering" wird im Sommer eingereicht.

Hans-Peter Kriegel ist seit 1991 Inhaber des Lehrstuhls für Datenbanksysteme an der Universität München und seit 2003 Direktor des Departments „Institut für Informatik". Er hat über 200 Veröffentlichungen auf begutachteten Tagungen sowie in Fachzeitschriften. Diplom (1973) und Promotion (1976) in Informatik erfolgten an der Universität Karlsruhe. Seine Forschungsschwerpunkte liegen in den Bereichen Datenbanksysteme für komplexe Objekte (Molekularbiologie, Medizin, Multimedia, CAD usw.), insbesondere Anfragebearbeitung, Ähnlichkeitssuche, hochdimensionale Indexstrukturen sowie im Bereich Knowledge Discovery in Databases und Data Mining.