Ausarbeitung zum Proseminar

„Datenreduktion“

Thema

„Clustering Techniken“

Tom Crecelius

1. Einführung

1.1 Was ist Clustering?

1.2 Anwendungsgebiete

2. Clustering großer räumlicher Datenmengen

2.1 Was ist ein „räumliches" Datenbanksystem?

2.2 Clustering-Algorithmen

2.2.1 Partitionierende Algorithmen

2.2.2 Hierarchische Algorithmen

2.2.3 Single-Scan-Algorithmen

2.3 Räumliche Indexstrukturen

2.3.1 Eigenschaften

2.3.2 Birch*

3. Clustering in metrischen Räumen

3.1 Metrischer Raum vs. n-dimensionaler Vektorraum

3.2 Probleme des Clustering in metrischen Räumen

3.3 Lösungsansätze

3.3.1 Map-First-Methode

3.3.2 Pre-Clustering-Algorithmen

4. Zusammenfassung

1.Einführung

1.1 Was ist Clustering ?

Clustering bedeutet, Objekte einer Datenbank in sinnvolle Untermengen zu gruppieren. Sinnvoll heißt in diesem Zusammenhang, daß ähnliche Objekte in gleichen Gruppen, in sogenannten Clustern, gespeichert werden. Ähnlichkeit zwischen Objekten wird durch eine Abstandsfunktion definiert, welche die Dreiecksungleichung[1] erfüllen muß.

1.2 Anwendungsgebiete

In den vergangenen Jahrzehnten wurde Clustering in Gebieten der Mustererkennung, Datenanalyse und Bildbe- und Verarbeitung verwendet. In neuester Zeit bildet Clustering einen Teilbereich der Suche nach brauchbaren, verständlichen und repräsentativen Mustern in Daten räumlichen Datenbanken (Knowledge Discovery).

2. Clustering großer räumlicher Datenmengen

2.1 Was ist ein „räumliches" Datenbanksystem?

Ein räumliches Datenbanksystem (Spatial Database System (SDBS)) verwaltet geometrische oder mehrdimensionale Objekte, z. B. Punkte, Linien, Polygone usw., bzw. n-dimensionale Merkmalsvektoren, die eine Art Beschreibung von Objekten sind.

Ein SDBS bietet in seinem Datenmodell und seiner Anfragesprache neben den räumlichen Datentypen auch die effiziente Implementierung der Operationen und Anfragen auf diesen Datentypen an.

Typische Operationen eines SDBS sind Abstandsberechnungen oder Berechnungen von Überschneidungen.

Wichtige Anfragen sind *Region-Queries*, also die Suche nach allen Objekten, die eine spezifizierte (Anfrage-) Region schneiden und *Nearest-Neighbor-Queries*, die alle Objekte, die am nächsten zu einem spezifizierten (Anfrage-) Objekt liegen, zurückliefern.

[1] Dreiecksungleichung: Für 3 Punkte x,y,z und Abstandsfunktion d gilt: $d(x,z) \leq d(x,y) + d(y,z)$

2.2 Clustering-Algorithmen

2.2.1 Partitionierende Algorithmen

Partitionierende Algorithmen [EKSX98] erzeugen eine Partition einer Datenbank D mit n Objekten in eine Menge von k Cluster, wobei k ein Eingabeparameter ist. Jeder Cluster wird entweder durch den Schwerpunkt (k-means) oder einem Objekt im Cluster, welches am nächstem zum Mittelpunkt liegt (k-medoid[2]), repräsentiert. Jedes Datenobjekt wird dem Cluster mit dem nächstliegenden Repräsentanten zugeordnet. Dies impliziert, daß das entstehende Clustering äquivalent zu einem Voronoi-Diagramm[3] ist. Die Form aller Cluster, die durch den Algorithmus gefunden werden ist konvex.

Beispiel: Eine Clustering mit n=23 Objekten und k=3 Cluster

Partitionierende Algorithmen beginnen typischerweise mit einer initialen Partition von D und benutzen anschließend eine iterative Kontrollstrategie, um die Clusterqualität zu optimieren. In diesem Fall heißt Clusterqualität optimieren, die durchschnittliche Distanz eines Objekts zu dem Repräsentanten des Clusters minimieren. Daher wird das Clustering-Problem durch partitionierende Algorithmen als Optimierungsproblem angesehen und kann durch das Problem des lokalen Minimums erschwert werden.

Ein Beispiel für einen partitionierenden Algorithmus ist CLARANS (Clustering Large Applications based on RANdomized Search). [NH94]

CLARANS ist ein k-medoid-Algorithmus mit einer zufallsbasierten und begrenzten Suchstrategie zur Verbesserung der initialen Partition. Im Gegensatz zu früheren Algorithmen ist es der erste effiziente Algorithmus. Im Hinblick auf sehr großen Datenmengen ist CLARANS aber immer noch sehr ineffizient, da angenommen wird, daß alle Objekte in den Hauptspeicher passen. Außerdem ist CLARANS sehr empfindlich bzgl. der Eingabeordnung der Daten.

Es gibt zwei Techniken, um CLARANS in ein SDBS durch räumliche Indexstrukturen zu integrieren.
1. **R*-Baum basiertes Samplen (siehe Abschnitt 2.3.1)**

[2] Sei O = {O_1, .. O_m} eine Menge von Objekten, d eine Abstandsfunktion. Der Medoid O_k ist das Objekt, der die Summe $\sum_{i=1}^{n} d(O_i, O)$ minimiert

[3] Beispiel: Ein Voronoi-Diagramm einer Ebene mit n Punkten besteht aus n konvexen Polygonen, die die Ebene vollständig überdecken und genau einen der n Punkte enthalten. Jede Punkt innerhalb eines Polygons liegt am nächsten zum Mittelpunkt des Polygons. Siehe zur Demonstration unter: „http://simon.cs.cornell.edu/Info/People/chew/Delaunay.html“

2. Fokussieren auf relevante Cluster

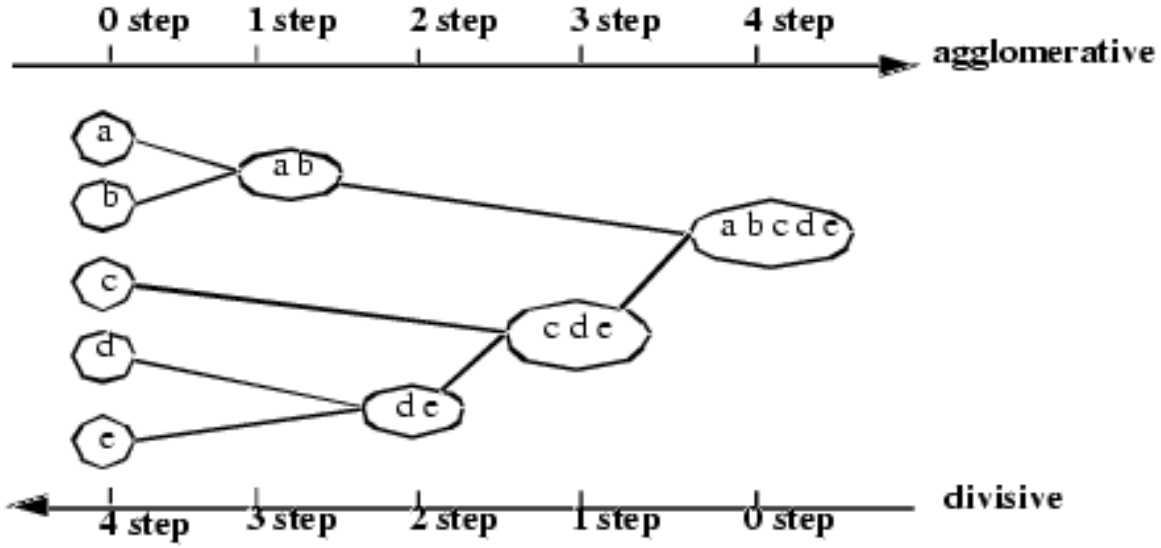
CLARANS kann große, festplattenresidente Datenmengen durch die Benutzung von R*-Bäumen handhaben, indem nur relevante Teile der Datenbank betrachtet werden.

Typischerweise versuchen k-medoid Algorithmen das bestehende Clustering dadurch zu verbessern, indem einer der Medoide in der Partitionierung durch einen Nicht-Medoid ausgetauscht wird, um dann die Clusterqualität des alten und neuen Clustering zu vergleichen. In CLARANS ist die Berechnung der Clusterqualität, der am meisten Zeit verbrauchende Vorgang, weil ein Scan durch die gesamte Datenbank vollzogen werden muß. Doch eigentlich tragen nur Objekte des alten bzw. neuen Clusters zur Clusterqualität bei. Daher müssen nur Objekte aus 2 der k Cluster von der Festplatte gelesen werden.

Partitionierende Algorithmen sind sehr nützlich, wenn die Cluster von konvexer Form sind , alle Cluster fast gleich groß sind und die Anzahl k der Cluster vernünftig bestimmt werden kann. Wenn k nicht bekannt ist, können verschiedene Parameter ausprobiert werden und von den entstandenen Cluster der "Silhouette Coefficient" [KR90] - eine numerische Maßeinheit für die Zweckmäßigkeit der Anzahl der Cluster - bestimmt werden.

2.2.2 Hierarchische Algorithmen

Im Gegensatz zu partitionierenden Algorithmen entsteht durch hierarchische Algorithmen [EKSX98] kein Ein-Ebenen-Clustering, sondern es wird eine Datenbank D mit n Objekten in mehreren Ebenen von Partitionen (Clustern) zerlegt. Diese hierarchische Zerlegung wird durch ein Dendrogramm aufgezeigt. Ein Dendrogramm ist ein Baum, der D iterativ in kleinere Sub-Mengen aufteilt, bis jeder Knoten des Baums aus nur noch einem Objekt besteht. In so einer Hierarchie repräsentiert jeder Knoten des Baumes einen Cluster.



Beispiel: Hierarchisches Clustering einer Menge {a, b, c, d, e}

Das Dendrogramm kann entweder von den Blättern zur Wurzel aufwärts (agglomerativer Ansatz) oder von der Wurzel zu den Blättern abwärts (divisiver Ansatz) durch Mischen bzw. Teilen der Cluster in jeder Stufe erstellt werden.

Im Gegensatz zu partitionierenden Algorithmen brauchen hierarchische Algorithmen keinen Eingabeparameter k. Trotzdem muß eine Abbruchbedingung angegeben werden, wann das Mischen bzw. Teilen der Cluster enden soll. Ein Beispiel für eine Abbruchbedingung für den agglomerativen Ansatz ist die kritische Distanz D_{min} zwischen allen Clustern. Wenn kein Abstand zweier Cluster kleiner als D_{min} ist, dann terminiert der Algorithmus.

Da agglomerative hierarchische Algorithmen die nächstliegenden Cluster zusammenmischen, hängt der Algorithmus stark von der Nearest-Neighbor-Suche und der Distanzfunktion innerhalb der Cluster ab. Räumliche Indexstrukturen machen Gebrauch von der Tatsache, daß die Nearest-Neighbor-Suche ein lokales Problem ist. Dies führt zu einer wesentlichen Verbesserung der Kosten.

Durch entsprechende festplattenbasierende räumliche Indexstrukturen können hierarchische Algorithmen in ein SDBS integriert werden.

2.2.3 Single-Scan-Algorithmen

Die Grundidee eines Single-Scan-Algorithmus [EKSX98] ist es, benachbarte Objekte in der Datenbank, basierend auf lokalen Clusterbedingungen, in Cluster zu gruppieren, so daß nur ein einziger Scan über alle Objekte der Datenbank erfolgt. Das Ergebnis ist wie im Fall der partitionierenden Algorithmen ein Ein-Ebenen-Clustering. Single-Scan-Algorithmen sind sehr effizient, wenn die Suche nach benachbarten Objekten effizient durch das SDBS unterstützt wird.

Die Laufzeit beträgt O(n * Datenbank-Laufzeit für NN-Suche)

Schema für einen Single-Scan-Algorithmus:

```
SingleScanClustering(Database DB)
    FOR each object o in DB DO
      IF o is not yet member of some cluser THEN
        create a new cluster C;
        WHILE neighboring objects satisfy the cluster condition DO
              add them to C
        ENDWHILE
      ENDIF
    ENDFOR
```

Unterschiedliche Clusterbedingungen zielen in unterschiedliche Clusterdefinitionen und Algorithmen.

Ein Beispielalgorithmus ist DBSCAN (Density Based Spatial Clustering of Applications with Noise) [EKSX 96]. DBSCAN ist entwickelt worden, um Cluster jeglicher Form in räumlichen Datenbanken mit Streuung zu finden. Die Schlüsselidee ist die, daß für jeden Punkt in einem Cluster, die Umgebung innerhalb eines bestimmten Radius (Eps), ein Minimum an Punkten enthalten muß (MinPts), d.h. also die Dichte des Clusters muß einen gewissen Schwellwert überschreiten. DBSCAN braucht zwei Eingabeparameter: MinPts und Eps. Es existieren einfache Heuristiken, um diese Parameter für den „dünnsten" Cluster zu bestimmen.

Es ist möglich, gewisse Single-Scan-Algorithmen zu hierarchischen Clustering-Algorithmen zu erweitern (z.B. DBSCAN zu STING (Statistical INformation Grid based Method) [WYM97]), ohne dabei große Effiziensverluste tolerieren zu müssen.

2.3 Räumliche Indexstrukturen

2.3.1 Eigenschaften

Räumliche Datenstrukturen organisieren die Datenpunkte bzw. den Datenraum in der Art, daß Punkte, die nahe zueinander liegen, zusammen auf der Festplatte gespeichert werden. Deshalb enthalten Indexstrukturen sehr viele Informationen über die Verteilung der Punkte und ihr Clustering. Diese räumliche Indexstrukturen können in Clustering-Algorithmen integriert werden.

Beispiele für die Integration einer räumlichen Indexstruktur in einen Clustering-Algorithmus:

2.3.2 R-Baum basiertes Samplen [KR90]*

Clustering eines SDBS mit wenig Hauptspeicher kann dadurch erreicht werden, daß man eine begrenzte, relativ kleine Anzahl von Repräsentanten aus der Datenbank wählt und einen Clustering-Algorithmus auf diesen Repräsentanten anwendet. Dies ist eine Art des Data Sampling. Der Nachteil ist, daß sich die Clusterqualität verschlechtert, wenn nur eine kleine Teilmenge der Daten betrachtet werden.

Traditionelles Data Sampling funktioniert nur im Hauptspeicher. Eine neue Methode wurde durch Einbeziehung eines R*-Baums entwickelt. Von jeder Datenseite des R*-Baums wird ein oder wenige Repräsentanten gewählt. Die Clusterstrategie des R*-Baums, der die Überlappung zwischen Direktorierechtecken minimiert, erzielt eine gut verteilte Menge von Repräsentanten.
Experimente zeigen, daß sich die Effizienz, um den Faktor 48 bis 158 verbessert, wohingegen sich die Clusterqualität nur um 1.5 % bis 3.2 % verschlechtert im Vergleich zu CLARANS mit und ohne R*-Baum basiertes Sampling.

*2.3.3 BIRCH** (Balanced Iterative Reducing and Clustering using Hierarchies)

BIRCH* [ZRL96] ist ein Algorithmus, der eine hochspezialisierte Baumstruktur zum Clustering großer Datenmengen in n-dimensionalen Vektorräumen verwendet. Während eines sequentiellen Scans über alle Objekte einer Datenbank werden inkrementell kompakte Beschreibungen von Sub-Clustern, sogenannte Cluster Feature (CF*) berechnet. Cluster Feature sind zusammenfassende Repräsentationen der Objekte eines Clusters. Sie enthalten Informationen über die Anzahl der Objekte, den Centroid[4] und den Radius. Die CF*-Werte werden in einem speziell dazu konstruierten CF*-Baum gespeichert.

[4] Der Centroid eines Clusters ist der Schwerpunkt des Clusters. Formal: Seien $v_i ; 1 \le i \le n$ k-dimensionale Objekte. Der Centroid $\overline{C}$ ist definiert als $\overline{C} = \sum_{i=1}^{n} \frac{v_i}{n}$;

2.3.3.1 CF-Baum*

Ein CF*-Baum ist eine balancierte Baumstruktur, ähnlich einem R*-Baum. Die Anzahl der Knoten im Baum wird durch eine vorspezifizierte Zahl M beschränkt. Jeder Nicht-Blattknoten (NL) besitzt maximal B Einträge der Form (CF_i^*,$child_i$), wobei $child_i$ ein Zeiger auf den i-ten Sohnknoten des Nicht-Blattknotens ist und CF_i^* ist das CF* der Menge der Objekte in dem Teilbaum, dessen Wurzel am i-ten Sohnknoten endet. Ein Blattknoten (L) enthält höchstens B Einträge. Jeder Eintrag ist ein CF* eines Clusters und jeder Cluster in der Blattebene erfüllt einen Schwellwert T, der die Qualität des Clusters kontrolliert.
In einem CF*-Baum unterscheiden sich die CF* und somit die Blatt- und Nicht-Blattknoten wesentlich. Nicht-Blattknoteneinträge dirigieren lediglich ein Objekt zum nächstliegendem Cluster, d. h. ein Objekt wird durch die CF_i weiter zur Blattebene gereicht, wogegen Blattknoteneinträge dynamisch entstehende Cluster repräsentieren.

Der Aufbau des Baumes geschieht inkrementell während eines Datenscans über die Objekte der Datenbank. Zuerst wird das nächstliegende Blatt zum einzufügendem Objekt gesucht. Wenn dort ein Eintrag existiert, der mit dem neuen Objekt noch die Schwellwertbedingung erfüllt, wird das Objekt in den entsprechenden Cluster eingefügt und die CF*-Werte aktualisiert. Existiert kein solcher Eintrag, wird ein neuer Eintrag in dem Blattknoten erstellt. Falls der Blattknoten keinen Eintrag mehr aufnehmen kann, wird dieser gespalten und die Einträge in beiden entstehenden Knoten umverteilt. Solch eine Spaltung kann potentiell bis zu Wurzel reichen.

Durch jedes Spalten eines Knotens wächst der Baum um einen Knoten. Während des Datenscans und der damit verbundenen Einfügeoperationen, kann der Baum über die Beschränkung der Größe durch die vorspezifizierte Zahl M steigen. Dann ist es notwendig den Platzverbrauch und somit wieder die Knotenanzahl zu verringern. Dies geschieht, indem der CF*-Baum neu aufgebaut und dabei nahe zueinanderliegende Cluster zusammengemischt werden. Um dies zu erreichen, wird der Schwellwert T erhöht und der Baum nur noch mit den CF*s aufgebaut. Anschließend setzt der Datenscan dort wieder an, wo er aufgehört hat und die Objekte werden wie gewohnt in den Baum eingefügt.

In einem CF*-Baum gibt es also zwei Typen von Einfügeoperationen.

Als Typ-I-Einfügung wird das Einfügen eines einzelnen Objekts oder äquivalent dazu das Einfügen eines Clusters mit nur einem Objekt in den CF*-Baum bezeichnet.
Als Typ-II-Einfügung wird das Einfügen eines Clusters mit mehr als einem Objekt in den CF*-Baum bezeichnet.
Daraus folgt, daß es Abstandsmessungen zwischen einem Objekt und einem CF* und zwischen zwei CF* geben muß.

3. Clustering in metrischen Räumen

3.1 Metrischer Raum vs. n-dimensionaler Vektorraum

Ein Metrischer Raum[5] wird informal durch eine Abstandsfunktion mit zugehöriger Menge von Objekten beschrieben. In einem metrischen Raum ist die einzig gegebene Operation auf Objekten die Berechnung des Abstandes zwischen Objekten.

Ein n-dimensionaler Vektorraum ist ein spezieller Typ des metrischen Raums. Er kann durch Vektoren repräsentiert werden und erlaubt daher verschiedenartige Vektoroperationen (z. B. Addition oder Subtraktion von Vektoren), um dichtere Repräsentationen von Cluster zu formen und um Zeit und Platzverbrauch des Clusterproblems zu verringern.

Viele Clustering-Algorithmen hängen von Vektoroperationen ab (z.B. bei der Berechnung des Centroiden, der einen Cluster repräsentiert und die Berechnungszeit somit verbessert.) Solche Algorithmen können keine Daten in metrischen Räumen clustern.

3.2 Probleme von Clustering in metrischen Räumen

Ein Problem des Clusterings in metrischen Räumen ist es, daß das Konzept eines Centroiden nicht definiert ist. Ein weiteres Problem liegt der Tatsache zu Grunde, daß Abstandsfunktionen in metrischen Räumen potentiell sehr teuer sind. Wenn man z. B. den Domain aller Zeichenketten betrachtet und zur Ähnlichkeitsbestimmung die Edit-Distance[6] Funktion annimmt, werden zwischen zwei Zeichenketten der Länge m und n, O(mn) einfachen Zeichenvergleiche benötigt, um den Abstand der beiden Zeichenketten voneinander zu berechnen.
Ein drittes Problem ist, daß die domainspezifische Natur anwendungsspezifischem Clustering Anforderungen stellt, die kaum durch einen einzelnen Algorithmus zu erfüllen sind.

3.3 Lösungsansätze

3.3.1 Map-First-Methode

Ein Lösungsansatz ist es, alle Objekte in einen k-dimensionalen Koordinaten Raum (Bildraum) abzubilden und dabei die Abstände zwischen den jeweiligen Objekten beizubehalten und anschließend die entstandenen Vektoren (Bildvektoren) zu clustern. k wird als Bilddimensionalität bezeichnet.

---

[5] Sei $X \neq \emptyset$ ein Menge. Dann heißen $d:X^2 \rightarrow \mathbb{R}$ Metrik auf X und (X,d) metrischer Raum falls $\forall x,y,z \in X$ gilt:

i) $d(x,y) \geq 0$

ii) $d(x,y) = d(y,x)$

iii) $d(x,z) \leq d(x,y) + d(y,z)$

iv) $d(x,y) = 0 \Leftrightarrow x=y$

[6] Die Edit-Distance Funktion mißt die Anzahl der einfachen Zeichenoperationen, um eine Zeichenkette in eine zweite zu überführen

Definition

Sei $O = \{O_1,...,O_m\}$ eine Menge von Objekten, k > 0. Eine Funktion $f : O \to IR^k$ wird als abstandsbewahrende Transformation bezeichnet, wenn für je zwei Objekte im metrischen Raum gilt, daß die Abstände zwischen den Objekten, gleich den Abständen zwischen ihren Bildvektoren ist, d. h.

$$\forall i, j \in \{1,...,n\} : d(O_i, O_j) = \left\| f(O_i) - f(O_j) \right\|$$

Das folgende Lemma besagt, daß es eine solche abstandsbewahrende Transformation immer gibt.

Lemma 1

Für jede Menge $O = \{O_1,...,O_m\}$ von Objekten in metrischen Räumen, gibt es eine Funktion f und ein $k < |O|$, so daß $f : O \to IR^k$ eine Abstandsbewahrende Funktion ist.

Beispiel für eine abstandsbewahrende Transformation:

Objekte x, y, z mit Abstandsfunktion d und Abständen:

d(x,y)=3, d(y,z)=4, d(z,x)=5

$\Rightarrow$ x=(3,0), y=(0,0), z=(0,4)

Ein Beispielalgorithmus für die Map-First-Methode ist *FastMap.*

***FastMap* ist eine skalierbare, approximative, abstandsbewahrende Transformation. Die Abstände zwischen den Objekten werden durch FastMap nur näherungsweise beibehalten. Als Eingabe erhält der Algorithmus n Objekte, eine Abstandsfunktion d und eine nicht-negative Zahl k. Als Ausgabe liefert der Algorithmus n Bildvektoren und einen k-dimensionalen Bildraum.**
Um n Objekte in den Bildraum abzubilden benötigt FastMap 3nkc Aufrufe von d, wobei c ein Parameter ist (typischer Weise c=1 oder c=2)

Um den k-dimensionalen Bildraum zu erstellen, wird jeder der k Achsen durch eine Linie zwischen jeweils zwei Objekten definiert. Diese 2k Objekte werden als Pivot Objekte bezeichnet.

FastMap besitzt die Fähigkeit Objekte inkrementell und schnell in den Bildraum abzubilden. FastMap projiziert ein neues Objekt auf die k Koordinatenachsen, um einen k-dimensionalen Vektor davon zu erzeugen. Der Algorithmus braucht dafür nur 2k Aufrufe von d. Abstände können jetzt durch die euklidische Abstandsfunktion berechnet werden.

Experimente zeigen aber, daß die Map-First-Methode entweder keine gute Clusterqualität erzeugt (z. B. FastMap) oder aber nicht auf große Datenmengen anwendbar ist, da die meisten Algorithmen annehmen, daß alle Objekte in den Hauptspeicher passen und Abstände paarweise zwischen allen Objekten berechnen.

3.3.2 Pre-Clustering-Algorithmen

Ein anderer Lösungsansatz wird durch Pre-Clustering-Algorithmen realisiert. Sie erstellen in einer ersten Phase eine datenabhängige Zusammenfassung großer Datenmengen in Sub-Cluster und repräsentieren diese in kompakter Form. Anschließend werden diese Repräsentationen durch ein domainspezifisches Clustering weiter analysiert.

Beispiele für Pre-Clustering-Algorithmen sind Bubble und Bubble-FM. Sie bauen direkt auf BIRCH* auf und erweitern dazu die Definition der CF* in der Blatt- und Nicht-Blattebene und sind für Clustering in metrischen Räumen ausgelegt.

3.4 Bubble/Bubble-FM

3.4.1 Bubble [GRG99]

Für jeden Cluster, der durch den Algorithmus gefunden wird, werden Anzahl der Objekte, Zentrales Objekt und der Radius des Clusters zurückgeliefert, um in weiteren Prozeßschritten verwendet werden zu können. Weil in einem metrischen Raum allgemein keine neuen Objekte durch Operationen auf Mengen von Objekten erzeugt werden können, wird ein aktuelles Objekt, der Clustroid im Cluster als Clusterzentrum ausgezeichnet.

3.4.1.1 CF* der L-Ebene

Die Cluster Feature CF* der Blattebene enthalten Informationen über die Anzahl der Objekte, 2p repräsentative Objekte, die RowSum[7]-Werte der 2p Objekte, der Clustroid[8] und den Radius[9].

Um die Schwellwertbedingung T einzuhalten, welche die Clusterqualität bestimmt, wird T durch die Clustroid-Distanz[10] D_0 definiert, d.h. ein neues Objekt O_{new} wird nur dann in einen Cluster eingefügt, wenn gilt:

$$D_0(C, O_{new}) \leq T$$

Durch Neueinfügungen in einen Cluster ändern sich möglicherweise die CF*-Werte des jeweiligen Clusters. Daher ist es wichtig, daß alle CF*-Werte inkrementell berechnet werden können.

---

[7] Die RowSum(O) eines Objektes O ist ein Maß für die relativen Abstände von O zu allen anderen Objekten im Cluster $RowSum(O) = \sum_{i=1}^{n} d^2(O_i, O)$

[8] Sei O = {O_1, .. O_m} eine Menge von Objekten. Der Clustroid $\hat{C} \in O$ minimiert die RowSum(O)

[9] Sei C = {O_1, .. O_m} die Menge der Objekte eines Clusters mit Clustroid $\hat{C}$. Der Radius des Clusters ist $r = \sqrt{\sum_{i=1}^{n} \frac{d^2(O_i, \hat{C})}{n}}$

[10] Sei C_a = {O_1, .. O_m}, C_b = {O_1, .. O_m} die Mengen von Objekten zweier Cluster. Die Clustroid Distanz D_0 zweier Cluster ist definiert als $D_0(C_a, C_b) = d(\hat{C}_a, \hat{C}_b)$

3.2.1.2 Inkrementelles Instandhalten der CF* in der L-Ebene

Der Instandhaltung des Clustroid liegten zwei Heuristiken zu Grunde.

Zur Motivation der Typ-I-Heuristik trifft man die folgende Approximation, daß unter jeder abstandsbewahrende Transformation f, der Bildvektor des Clustroiden gleich dem Centroiden aller Bildvektoren ist.

Folgendes Lemma besagt, daß dies die bestmögliche Approximation ist.

Lemma 2

Sei $C = \{O_1,...,O_m\}$ die Menge der Objekte eines Clusters mit Clustroid $\hat{C}$. Sei $f : C \to IR^k$ eine abstandsbewahrende Transformation und sei $\overline{C}$ der Centroid der Bildvektoren, dann gilt:

$$\forall O \in C : \left\| f(\hat{C}) - \overline{C} \right\| \leq \left\| f(O) - \overline{C} \right\|$$

Auf Grund dieser Approximation kann man beobachten, daß die Berechnung der RowSum(O_{new}) eines neuen Objekts O_{new} unter erheblicher Speichereinsparung approximativ durch den Clustroiden möglich ist.

$$\begin{aligned} \text{RowSum}(O_{new}) &= \sum_{j=1}^{n} d^2(O_{new}, O_j) = \sum_{j=1}^{n} (f(O_{new}) - f(O_j))^2 \\ &= \sum_{j=1}^{n} (f(O_j) - f(\overline{C}))^2 + n(f(\overline{C}) - f(O_{new}))^2 \\ &\approx nr^2(C) + nd^2(\hat{C}, O_{new}) \end{aligned}$$

Eine zweite Beobachtung ist die, daß nach Neueinfügung eines Objekts O_{new} in einen Cluster C, der neue Clustroid $\hat{C}^*$ innerhalb eines Radius ε um den alten Clustroid $\hat{C}$ liegt.

$$d(\hat{C}^*, \hat{C}) \leq \varepsilon \text{ ; mit } \varepsilon = \frac{T}{n+1}$$

Das bedeutet also, daß bei hoher Anzahl von Objekten im Cluster, die Objekte innerhalb des Radius ε um $\hat{C}^*$ fast die gleichen sind, wie um $\hat{C}$.

Die Typ-I-Heuristik lautet dann wie folgt:

Wenn die Anzahl der Objekte in einem Cluster kleiner als eine Konstante p ist, halte alle Objekte des Clusters im Hauptspeicher und berechne den Clustroid $\hat{C}^*$ durch Neuberechnung aller RowSum-Werte exakt. Sonst halte eine Teilmenge t von p Objekten innerhalb eines Radius ε um den alten Clustroiden $\hat{C}$ im Hauptspeicher und berechne $\hat{C}^*$ approximativ durch ausschließliches betrachten der RowSum-Werte der p Objekte. Das Objekt mit dem niedrigsten RowSum-Wert ist der neue Clustroid.

Falls die RowSum(O_{new}) eines neu eingefügten Objekts O_{new} kleiner als die RowSum(O_{pmax}) des Objekts O_{pmax} mit dem größten RowSum-Wert der p Objekte in der Teilmenge t ist, ersetze in der Teilmenge t das Objekt O_{pmax} durch O_{new}.

Das Finden der p Objekte ist dadurch einfach, weil es die Objekte mit den niedrigsten RowSum-Werten im Cluster sind. Experimente zeigen, daß diese Heuristik sehr gut funktioniert.

Der Motivation der Typ-II-Heuristik, liegt die Beobachtung zu Grunde, daß der neue Centroid $\overline{X}$ zweier zu mischender Cluster C_1 und C_2 auf der Verbindungslinie zwischen den Centroiden $\overline{X}_1$ und $\overline{X}_2$ des jeweiligen Clusters C_1, C_2 liegt. Die genaue Position hängt von der Anzahl n_1, n_2 der Objekte in den beiden Clustern ab und ist durch den Schwellwert T beschränkt.

$$(\overline{X} - \overline{X}_1)^2 = \frac{n_2^2 (\overline{X}_1 - \overline{X}_2)^2}{(n_1 + n_2)^2} \approx \frac{n_2^2 d^2 (\vec{C}_1, \vec{C}_2)}{(n_1 + n_2)^2} < \frac{n_2^2 T}{(n_1 + n_2)^2}$$

Weiterhin wird erstens angenommen, daß C_1 und C_2 nahe zueinander liegen, sich aber nicht überlappen. Denn da C_1 und C_2 gemischt werden sollen, wird das Schwellwertkriterium T erfüllt, was impliziert, daß C_1 und C_2 nahe zueinander liegen und weil sie zwei unterschiedliche Cluster im ursprünglichen CF*-Baum waren, sind sie nicht überlappend.

Wegen dem Fehlen irgendwelcher vorhergehender Information über die Cluster, wird zweitens angenommen, daß die Objekte in den zusammengemischten Cluster gleichverteilt sind. Daher ist die Anzahl der Objekte in den Clustern nahezu gleich.

Aus der Beobachtung und den beiden Annahmen heraus folgt, daß der neue Clustroid $\vec{C}^*$ der zusammengemischten Cluster C_1 und C_2 auf den Peripherien, also dem Rand der beiden Cluster liegt. Die Objekte der Peripherie, sind die Objekte, die am weitesten vom Clustroid weg liegen und haben daher die jeweils höchsten RowSum-Werten im zugehörigen Cluster.

Die Typ-II-Heuristik ist folgende:

Speichere p Objekte aus den Peripherien der beiden zu mischenden Cluster C_1, C_2 im neuen CF* und berechne den Clustroiden $\vec{C}^*$ approximativ durch ausschließliches Betrachten der RowSum-Werte der p Objekte. Der neue Clustroid $\vec{C}^*$ ist das Objekt mit dem niedrigsten RowSum-Wert der p Objekte.

Die inkrementelle Instandhaltung der restlichen Werte in den CF* der Blattebene ist trivial.

3.4.1.3 CF* der Nicht-Blattebene

Die CF* spiegeln die Verteilung aller Cluster im Sub-Baum des jeweiligen Sohnknoten durch Sample Objekte (S) wider.

Die Sample Objekte jedes Nicht-Blattknotens NL ist die Vereinigung der Sample Objekte der Blattknoteneinträge NL_i.

Die Sample Objekte für jeden Blattknoteneintrag NL_i werden rekursiv ermittelt.

Sei NL ein Nicht-Blattknoten, $child_1$,...,$child_k$ die Sohnknoten von NL mit jeweils n1,...,nk Einträgen.

Wenn der Sohnknoten $child_i$ des Nicht-Blattknoteneintrags NL_i ein Blattknoten ist, werden die Sample Objekte zufällig aus allen Clustroiden der CF^* des gesamten Blattknoten ausgewählt. Sonst werden die Sample Objekten zufällig aus den Sample Objekten des Sohnknoten $child_i$ ausgewählt.

Die Anzahl der Sample Objekte für jeden Nicht-Blattknoten NL ist durch eine Zahl, die sogenannten Sample Size (SS) beschränkt. Die Anzahl der Sample Objekte für einen Nicht-Blattknoten, die durch ein Knoteneintrag NL_i beigetragen werden, ist beschränkt durch $\max\left(\left\lfloor \frac{n_i * SS}{\sum_{j=1}^{k} n_j} \right\rfloor, 1\right)$

Aus jedem Nicht-Blattknoteneintrag NL_i wird mindestens ein Sample Objekt entnommen, damit die Verteilung der Sample Objekte repräsentativ für alle Sohnknoten ist.
Weiterhin wird der Abstand zwischen einem neu eingefügtem Objekt O_{new} und einem Nicht-Blattknoteneintrag NL_i durch die Sample Objekte der Einträge und der Average Inter-Cluster Distanz[11] definiert.

$$d(O_{new}, NL_i) = D_2(\{O_{new}\}, S(NL_i))$$

Die Genauigkeit der Sample Objekte hängt von der Häufigkeit ihrer Aktualisierungen ab. Jede Aktualisierung der Sample Objekte erhöht hingegen die laufenden Kosten. Daher ist es wichtig eine Balance zwischen den Kosten und der Genauigkeit zu finden.

Eine Spaltung eines Sohnknotens $child_i$ verursacht eine Umverteilung der Einträge zwischen $child_i$ und dem neuem Knoten $child_{k+1}$. Daher müssen die Sample Objekte der zugehörigen Einträge NL_i, NL_{k+1} auf jeden Fall aktualisiert werden bzw. erstmals ausgewählt werden.
Um Veränderungen der Verteilung in allen Sohnknoten widerzuspiegeln, werden die Sample Objekte aller Einträge eines Nicht-Blattknotens im Falle einer Spaltung eines Sohnknotens aktualisiert.

Experimente haben gezeigt, daß Bubble bzgl. der Parameterwahl für Sample Size (SS) und Repräsentationszahl (2p) unempfindlich gegenüber kleineren Abweichungen ist. Als gut stellte sich SS=10 heraus. Die Repräsentationszahl hingegen ist abhängig von dem Verzweigungsgrad (BF) des Baumes. Hier haben Experimente ergeben, daß 5*BF für die Repräsentationszahl gute Ergebnisse erzielen.

[11] Sei $C_1=\{0_1,...O_{n1}\}$, $C_2\{O_1,...,O_{n2}\}$ die Menge der Objekte zweier Cluster C_1, C_2. Die Average Inter-Cluster Distanz D_2 ist definiert als $D_2(C_1, C_2) = \sqrt{\frac{\sum_{i=1}^{n1}\sum_{j=1}^{n2} d^2(O_{1i}, O_{2j})}{n_1 n_2}}$

3.5 BUBBLE-FM

Bubble-FM [GRG99] ist eine verbesserte Variante von Bubble, da die Anzahl der Aufrufe der Abstandsfunktion d durch Integrieren von FastMap verringert wurde. Bubble-FM unterscheidet sich von Bubble nur in der Benutzung der Sample Objekte und dadurch in der Definition der CF^* der Nicht-Blattebene.

Bei Einfügung eines neuen Objekts wird mit Hilfe von FastMap die Menge der Sample Objekte in einen approximativen Bildraum transformiert. Anschließend werden die Abstände durch die euklidische Abstandsfunktion berechnet.

Auch beim Aktualisieren der Sample Objekte wird FastMap benutzt, um im Bildraum den Centroiden der Bildvektoren zu bestimmen. Der Centroid der Bildvektoren ist dann der Centroid des Clusters.

Die CF^* der Nicht-Blattebene enthalten Informationen über die Sample Objekte, den Centroid der Bildvektoren der Sample Objekte und die Bildvektoren der 2k Pivot Objekte, die den Bildraum aufspannen.

4. Zusammenfassung

Abschließend ist festzustellen, daß partitionierende Algorithmen gut geeignet sind, wenn die Cluster von konvexer Form sind und ihre Anzahl genau bestimmt werden kann.
Hierarchische Algorithmen sind gut geeignet, wenn die Anzahl der Cluster nicht von vornherein bestimmbar ist.
Single-Scan-Algorithmen sind in ihrer Laufzeit stark von der NN-Suche abhängig.
Räumliche Indexstrukturen sind generell für Clustering-Algorithmen gut geeignet.
Und für Clustering in metrischen Räumen sind Pre-Clustering-Algorithmen unverzichtbar.

[NH94] R.T. Ng and J. Han. Efficient and effective clustering Methods for spatial data mining. VLDB, 1994

KR90] Kaufman L., Rousseeuw P.J.: "Finding Groups in Data: An Introduction to Cluser Analysis", John Wiley&Sons,1990)

[EKSX] Ester M., Kriegel H.-P., Sander J., Xu X.:"Densitz- Connected Sets and their Application for Trend Detection in Spatial Databases". Proc. 3rd Int. Conf. on Knowledge Discovery and Data Mining. Newport Beach, Ca., AAAI Press, 1997

[WYM97] Wang W., Yang J., Muntz R.:"STING: A Statistical Information Grid Approach to Spatial Data Minig", Proc. 23.rd Int. Conf. on Very Large Data Bases, Athens, Greeece, Morgan Kaufmann, 1997, pp.186-195

[ZRL96] Zhang T. Ramakrishnan R., Linvz M.: "BIRCH: An Efficient Data Clustering Method for Verz Large Databases". Proc. ACM SIGMOD Int. Conf. on Managment of Data, ACM Ress, 1996

[GRG99] Ganti V., Ramakrishnan R., Gehrke J.:"Clustering Large Datasets in Arbitrary Metric Spaces"

[EKSX98] Ester M., Kriegel H.-P., Sander J., Xu X.:"Clustering for Mining in Large Spatial Databases"