COMBI-GROUP(): Eine relationaler Operator zur Unterstützung von Data Mining Anwendungen

Dirk Habich
MLU Halle

Betreuer
Dr. A. Hinneburg
MLU Halle

Betreuer
Prof. Dr. W. Lehner
TU Dresden

Zusammenfassung

Die Anwendung von Data Mining Algorithmen werden durch die Zunahme der Daten in Datenbanksystemen immer interessanter. Die Implementierung der Data Mining Algorithmen kann entweder ausserhalb oder innerhalb des Datenbanksystem vorgenommen werden. Client Programme müssen die notwendigen Daten aus dem Datenbanksystem laden und degradieren dieses damit zu einem reinen Speichersystem ohne Berücksichtigung und Benutzung der effizienten Operatoren innerhalb des Datenbanksystemes. Die Implementierung der Data Mining Algorithmen innerhalb des Datenbanksystemes erfordert komplexe Mechanismen. Als dritte Alternative bietet sich die Implementierung eines spezifischen Anwendungsoperators im Datenbanksystem an, der für besonders datenintensive Aufgaben durch die Data Mining Anwendung benutzt werden kann. Dieser Ansatz ist bereits erfolgreich auf dem Gebiet des Online Analytical Processing (OLAP) angewendet worden, dabei wurden *grouping sets(), cube()* und *rollup()*-Konstrukte als Erweiterung des group by Ausdruckes in Datenbanksystemen implementiert. Bisher wurde aber kein spezifischer Data Mining Operator entwickelt, der für verschiedene Data Mining Anwendungen genutzt werden kann. Im folgenden Abschnitt wird die Motivation für unseren Ansatz kurz erläutert. Abschnitt 2 betrachtet die vorhandenen Möglichkeiten und im Abschnitt 3 wird der Ansatz für einen neuen Gruppierungs- und Diskretisierungsoperator dargestellt.

1 Einleitung

Basismethoden für Data Mining zur Analyse großer Datenbanken ist das Lernen von Klassifikatoren (z.B. Klassfikationsbäume [1]) und Cluster-Analyse (projected clustering [3]). Diese Methoden arbeiten auf großen d-dimensionalen Vektordatenmengen, die in Tabellen mit d Attributen gespeichert werden können. Gemeinsame Teilaufgabe der zitierten Verfahren ist die Analyse vieler Projektionen des d-dimensionalen Vektorraumes. Bei der Analyse werden einzelne Datenpunkte zu Gruppen zusammengefaßt, die in den Algorithmen weiterverwendet werden. Beispielsweise ist es für das Finden von Clustern in Projektionen notwendig die Wahrscheinlichkeitsdichte in 1-dimensionalen Projektionen zu schätzen. Dies kann mit Hilfe eines equi-distanten Histogrammes geschehen, das kontinuierliche Datenwerte in gleich große Intervalle gruppiert und zählt, wie häufig einzelne Intervalle belegt sind. Diese Art von Dichteschätzung wird auf alle 1- oder mehr-dimensionale Projektionen angewendet.

2 Vorhandene Möglichkeiten

Um alle Projektionen mit einer Dimensionalität von k aus einer gegebenen Menge von d Attributen berechnen zu können, müssen $\binom{d}{k}$ SQL-Anweisungen ausgeführt werden, wobei jede SQL-Anweisung in einer Projektion gruppiert. Sind zum Beispiel für die vier Attribute a_1, a_2, a_3, a_4 in allen drei-dimensionalen Projektionen Histogramme zu berechnen, so benötigt man $\binom{4}{3} = 4$ group by-Anweisungen. Jede SQL-Anweisung gruppiert in einer der folgenden Kombinationen

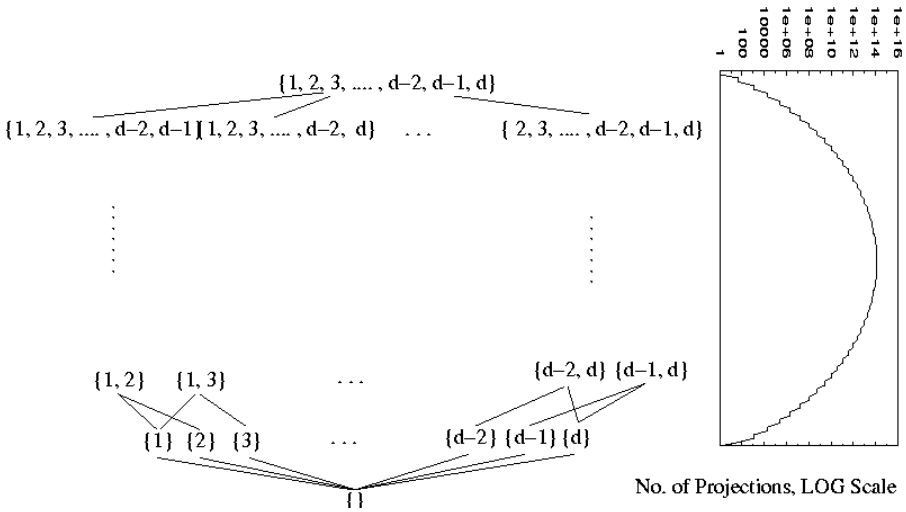

Abbildung 1: Suchraum für Projektioen

(a_1, a_2, a_3), (a_1, a_3, a_4), (a_1, a_2, a_4), (a_2, a_3, a_4). Da diese einzelnen SQL-Anweisungen die gesamte Tabelle immer lesen müssen, ist diese Methode sehr ineffizient und es kann keine Optimierung durch das Datenbanksystem über alle SQL-Anweisungen erfolgen, da Datenbanksysteme bisher einzelne SQL-Anweisung optimieren.

Eine neue Möglichkeit zur Berechnung der Histogramme in den Projektionen bieten die Gruppierungserweiterungen, die im Rahmen des Online Analytical Processing (OLAP) entstanden sind. Mittels *grouping set()* oder *cube()* können alle geforderten Histogramme in den Projektionen mittels einer einzelnen SQL-Anweisung angefordert werden. Alle drei-dimensionalen Projektionen aus vier Attributen können zum Beispiel mittels group by grouping set $((a_1, a_2, a_3)$, (a_1, a_3, a_4), (a_1, a_2, a_4), $(a_2, a_3, a_4))$ beschrieben werden. Der cube()-Operator berechnet alle Kombinationen der aufgeführten Attribute [2]. Deshalb müssen bei der Verwendung des cube()-Operators mittels *having* alle Projektionen herausgefiltert werden, die nicht im Ergebnis enthalten seien sollen. Um alle zweier Projektionen des durch die drei Attribute a_1, a_2, a_3 aufgespannten Datenraumes zu untersuchen, müssen die Kombinationen (a_1, a_2, a_3), (a_1), (a_2), (a_3), () mittels having-Bedingung ausgeschlossen werden. Die linke Grafik der Abbildung 1 zeigt den Suchraum für Projektionen in einer Menge mit d Attributen. Die rechte Grafik zeigt die Anzahl der Projektionen mit Dimensionalität k und einer fixen Anzahl von $d = 50$ Attributen. Mit dem grouping set()-Konstrukt können einzelne Teile der linken Grafik aus der Abbildung 1 spezifiziert werden. Der cube()-Operator mit d Gruppierungattributen berechnet den kompleten Graphen, mittels having-Klausel können bestimmte Teile herausgenommen werden. Somit baut der cube()-Operator den Graphen von oben und der grouping set()-Operator von unten auf.

Für mehr als 100 Gruppierungsattribute ist der Schreibaufwand für die SQL-Anweisung viel zu groß und die Berechnung der Projektionen mittels dieser OLAP-Funktionen verschafft keinen Vorteil. Bei der Auswertung der erweiterten Gruppierungsanweisungen wird als erstes die gemeinsame Aggregationsbasis ausgerechnet und danach gruppiert und das Ergebnis in eine temporäre Tabelle geschrieben. Anschließend werden die jeweils lokalen Gruppierungskombinationen auf der temporären Tabelle berechnet. Diese Vorgehensweise ist sehr effizient, wenn die Gruppierung über die Aggregationsbasis die Tabelle wesentlich verkleinert, was in unserem Fall nicht geschieht. Die temporäre Tabelle entspricht der Ausgangstabelle.

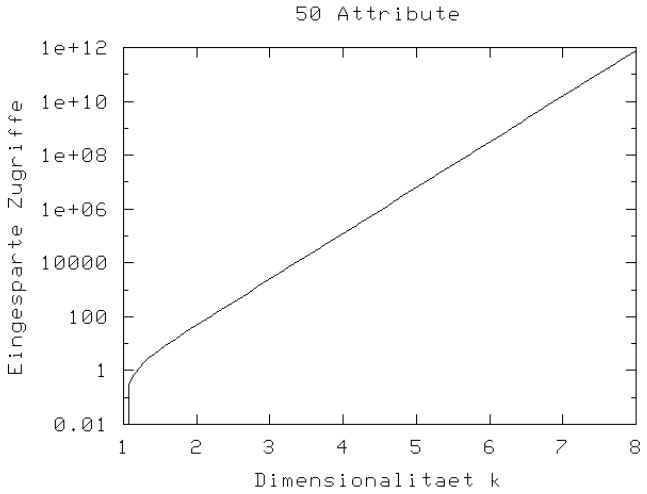

Abbildung 2: Zugriffseinsparungen

3 Der COMBI-GROUP Operator

In beiden vorgeschlagen Methoden, für die Berechnung aller Projektionen mit Dimensionalität k aus d Attributen, wird die Tabelle mindestens $\binom{d}{k}$-mal gescannt. Auf Grund dieser Beobachtung war es uns Motivation genug einen neuen Operator zu entwickeln, der alle Projektionen mit Dimensionalität k aus einer gegeben Menge von d Attributen bestimmt. Die Syntax orientiert sich dabei an den OLAP-Funktionen. Der neue Operator

$$\text{group by grouping combination } (\ (a_1, a_2, \ldots, a_d)\ ,\ k\ )$$

berechnet aus den Gruppierungsattributen $(a_1, a_2, \ldots, a_d)$ alle Projektionen mit der geforderten Dimensionalität k.

Folgenden Ansatz zur effizienteren Berechnung von Histogrammen in Projektionen, soll in dieser Studienarbeit verfolgt werden. Da niedrig-dimensionale Histogramme nur wenig Speicherplatz benötigen, können mehrere Histogramme in einem Tabellendurchlauf berechnet werden. Besonders bei mehr-dimensionalen Projektionen wird so das mehrfache Lesen einzelner Dimensionen vermieden. Konkret werden dadurch $d^{k-1} - 1$ Zugriffe auf den selben Datenwert eines Attributes eingespart. Die Abbildung 2 zeigt dies am Beispiel eines 50-dimensionalen Vektorraumes.

Literatur

[1] Johannes Gehrke, Raghu Ramakrishnan, and Venkatesh Ganti. Rainforest - a framework for fast decision tree construction of large datasets. In *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 416–427, 1998.

[2] Jim Gray, Adam Bosworth, Andrew Layman, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. In *Proceedings of the Twelfth International Conference on Data Engineering, February 26 - March 1, 1996, New Orleans, Louisiana*, pages 152–159, 1996.

[3] Cecilia M. Procopiuc, Michael Jones, Pankaj K. Agarwal, and T. M. Murali. A monte carlo algorithm for fast projective clustering. In *Proceedings of the ACM SIGMOD international conference on Management of data*, pages 418–427. ACM Press, 2002.