Wie eine Datenbank-Indexstruktur für Data-Mining angewendet werden kann

Johannes Kirchmaier

20. November 2017

Zusammenfassung

Bei der Verwendung von Data-Mining Algorithmen auf große Datenmengen bieten Indexstrukturen, wie der R*-Baum beim Clusteringverfahren DBSCAN,kommt es zu einer beschleunigten Bearbeitungszeit durch Reduktion der Befehlsanforderungen, aber auch zu einer zwangshaften Erhöhung der benötigten Speicherkapazitäten durch Mehrfachabspeicherung von Daten und/oder dem Hinzufügen neuer Daten, die die Verwaltung erleichtern.

Inhaltsverzeichnis

1 Einleitung

Data-Mining beschreibt den Prozess, große Datenbestände (Big Data), mithilfe von statistischen Methoden, nach noch unbekannten Zusammenhängen zu durchsuchen. Darunter fallen:

- Clusteranalyse: suche nach Daten die sich ähnlich sind und bilde daraus Gruppen
- Ausreißer-Erkennung: Erkennung von Daten die abweichen vom Rest
- Klassifikation: ähnlich wie Cluster, aber in der Regel vordefinierte Gruppen (Länder, Beruf, etc.)
- Regressionsanalyse: Auswertung von statistischen Zusammenhängen unterschiedlicher Attribute
- Assoziationsanalyse: Suche nach häufigen Zusammenhängen in Datensätzen

Diese Durchsuchung der Datenbestände wird ergänzt durch den Begriff der Knowledge Discovery in Databases, welcher das Data-Mining um die Vorbereitung der Daten und die Bewertung der Ergebnisse vervollständigt. Daher Datenbanken meist große Datenmengen beinhalten, spielt, neben der Korrektheit der Ergebnisse, die Bearbeitungszeit der angewandten Methoden eine wichtige Rolle. Hier kann eine Indexstruktur helfen.

Eine Indexstruktur sorgt dafür, dass Daten schneller abgerufen werden können, unter der Bedingung, dass mehr Speicherplatz in der Datenbank benötigt wird und es muss öfter auf die Datenbank geschrieben werden, um die Indexstruktur zu erhalten. Der erhöhte Speicherbedarf lässt sich dadurch erklären, dass bei einem Datenbankindex jede Reihe um mindestens einen Index. Solch ein Index enthält Leaf Nodes, mit den Informationen der Reihe aus der Datenbank, und sortiert diese gegebenenfalls nach einem vorher angegebenen Kriterium. Damit diese Sortierung innerhalb des Index nicht zerstört wird, muss, jedes mal wenn neue Daten in die Datenbank übertragen werden, der Index an diese Daten angepasst werden.

Im Folgenden wird der Density-Based Spatial Clustering of Application with Noise Algorithmus (kurz: DBSCAN), welcher Teil der Clusteranalyse ist, und der R*-Baum, eine Indexstruktur die vorwiegend für Geodatenbanken mit mehrdimensionalen Daten eine Rolle spielt, erläutert, zusammengefÃ¼hrt und veranschaulicht.

2 DBSCAN zur Erkennung von Clustern

Bevor man sich näher mit DBSCAN befassen kann, muss die Clusteranalyse erklärt werden. Ein Cluster ist eine Gruppierung von Daten, die eine gewisse Ähnlichkeit untereinander zeigen. Diese Cluster können entweder exklusiv oder nicht exklusiv sein, das heißt sie überlappen sich nicht oder sie überlappen sich, jeweils respektiv zur Exklusivität. Ein anderes wichtiges Merkmal für Cluster ist ob sie scharf oder unscharf sind, das bedeutet, ob die Clusterzugehörigkeit der Daten binär beantwortet werden kann, oder ob ihnen schwammig zwischen 0 und 1 Werte zugewiesen werden, meist basierend auf der Entfernung zum Clusterzentrum.

Um eine Clusteranalyse anzuwenden gibt es mehrere Verfahren:

- Partitionierende Clusterverfahren: Zuweisung in Cluster durch vorher bestimmte Anzahl der Cluster und Veränderung deren Zentren, bis sich die Zuordnung nicht mehr verändert (Beispiel K-Means)

- Hierarchische Clusterverfahren: Zuweisung der Objekte zu einer Hierachie, auf der einen Seite ein Cluster mit allen Objekten, auf der anderen fÃ¼r jedes Objekt ein Cluster (Beispiel Divisive Analysis Clustering)
- Dichtebasierte Verfahren: Betrachtung von Objekten, welche dicht beieinander liegen, getrennt von Objektmengen mit geringer Dichte (Beispiel DBSCAN)
- Kombinierte Verfahren: Kombination aus vorherig genannten Verfahren (Beispiel Spectral Clustering)
- Biclustering: gleichzeitiges Clustern von Zeilen und Spalten (Überwiegend Verwendung in der Bioinformatik)

DBSCAN zählt zu den dichtebasierten Verfahren, denn er arbeitet mit einem vorgegebenen Abstand ϵ zwischen den Objekten und macht diese, basierend darauf, dass sie mindestens $minPts$ Nachbarn innerhalb des Clusters haben, zu Kernobjekten oder Randobjekten.

2.1 Anwendung ohne Indexstruktur

Die wichtigsten Merkmale von DBSCAN sind, dass bei diesem Verfahren die Dichte berücksichtigt wird, dass die Anzahl der Cluster vorher nicht bekannt sein muss ,dass Rauschpunkte bei der Clusterzuweisung ignoriert und separat zurückgeliefert werden und dass es keine vorgegebene geometrischer Raum des Clusters gibt. Um diese Eigenschaften zu gewährleisten verfügt der Algorithmus Ã¼ber zwei Parameter. ϵ und $minPts$. ϵ beschreibt den maximalen Abstand zwischen zwei Objekten, man nennt ihn auch *Nachbarschaftslänge*. Mit dem Parameter $minPts$ wird festgelegt wann ein Objekt dicht und wann es dichte-erreichbar ist, denn wenn die Anzahl an ϵ-erreichbaren Nachbarn $>= minPts$ ist, so nennt man das Objekt dicht oder Kernobjekt.Wenn ein Objekt ϵ-erreichbar ist aber nicht $minPts$ Nachbarn hat, so nennt man es dichte-erreichbar oder Randobjekt. Der letzte zu nennende Fall, der Objekte die weder dichte-erreichbar noch dicht sind, nennt man Rauschpunkte (im englischen noise).

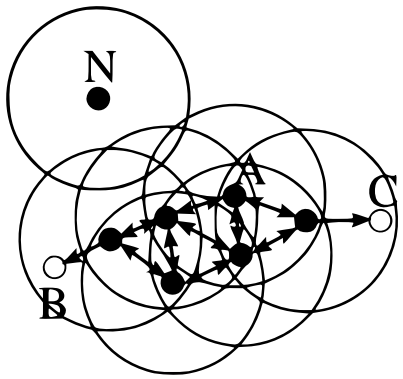

Abbildung 1: DBSCAN angewendet auf neun Objekte

In Abbildung 1 sieht man ϵ dargestellt durch die Pfeile und dadurch auch alle ϵ-erreichbaren Objekte. Alle dichten Objekte des Clusters A sind schwarz markiert und mit Pfeilen verbunden. B und C sind zwar mit den dichten Objekten aus A verbunden, haben aber nicht $minPts$ Nachbarn (in diesem Fall ist $minPts = 3$), deshalb sind die Objekte B und C dichte-erreichbar und gehören deswegen auch

zum Cluster A. N hingegen ist nicht dichte-erreichbar und gehört deswegen zur Rauschmenge.

Ein Problem, welches mit DBSCAN auftauchen kann ist, dass der Algorithmus keine unterschiedlichen Dichten der Cluster verarbeiten oder erkennen kann. Dieses Problem wird von dem Algorithmus Ordering Points To Identify the Clustering Structure (OPTICS) gelöst, welcher auf DBSCAN basiert, indem er die Rauschpunkte auf eine Vorrangwarteschlange legt und vergleicht von welchem Cluster er am besten zu erreichen ist, dadurch werden Rauschpunkte in die Cluster integriert und dadurch ist diese Analyse rauschfrei.

2.2 Integration eines R*-Baumes

Der R*-Baum ist eine Indexstruktur die speziell für räumliche Informationen erstellt wurde. Er sind eine Abwandlung des R-Baumes. Das R im Namen steht für Rechteck, denn Ziel des Baumes ist es, möglichst effektiv Objekte in rechteckige Begrenzungskasten (im englischen bounding box) einzusortieren. Die Besonderheit des R*-Baumes, im Vergleich mit dem R-Baum, ist, dass eine weiterentwickelte Teilungs-Strategie für eine Minimierung des Überlappens sorgt. Diese Reduktion sorgt dafür, dass meistens weniger Teilbäume durchsucht werden müssen und darum die Suchanfragen effizienter werden. Außerdem bewirkt ein erhöhter Füllgrad des R*-Baumes ebenfalls effizienteres Verhalten.

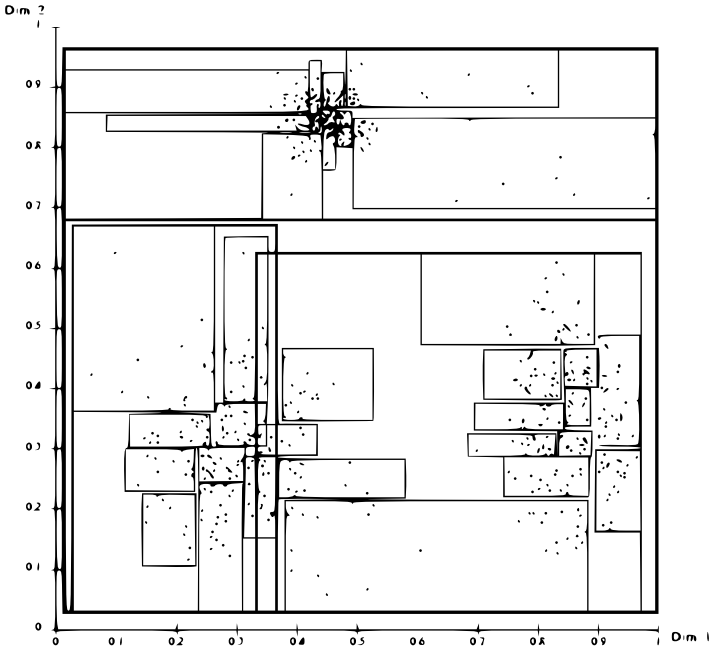

Abbildung 2: Einteilung einer Objektmenge in einen 3-stufigen R*-Baum

Auf der Abbildung 2 ist das Endprodukt der Aufteilung einer Objektmenge in einen R*-Baum mit drei Ebenen. Der alles umfassende Baum ist hier mit einem Rechteck um die gesamte Menge dargestellt, das ist die erste Ebene des R*-Baumes und alle weiteren sind Teilbäume davon. Die zweite Ebene teilt sich in drei Teilbäume auf, einer ist am oberen Bildrand zu erkennen, ein anderer ist nahezu quadratisch unten rechts und der letzte nimmt die linke untere Ecke des Bildes ein, schneidet den Teilbaum der unteren linken Ecke aber ein wenig. Durch diesen Schnitt sieht man, dass eine Überlappung nicht immer vermieden werden kann, aber trotzdem minimal

gehalten wird um den Effizienzverlust zu verringern.Die bisherigen Rechtecke zeigen Indexseiten. Die kleinsten sichtbaren Rechtecke stellen die Blätter der Struktur dar und bilden die dritte Ebene. In diesen Blättern werden die Objekte zu Gruppen zusammengefasst.

2.2.1 Einordnung des R*-Baumes

Um einen R*-Baum mit DBSCAN verwenden zu können, muss zuerst der Baum auf die angewandte Datenmenge angewendet werden. Ein R*-Baum speichert in seinen Nicht-Blättern eine Möglichkeit, Kindknoten zu erkennen, und die Begrenzungskästen aller Einträge seiner Kindknoten. Für die Anwendung ist besonders dies wichtig, damit die Koordinaten der Knoten abgefragt werden können. Die Blätter des Baumes speichern die Daten der Kinder, oft in Form eines Begrenzungskastens und einer externen Kennung.

Beim Einfügen neuer Objekte in den Baum wird eine gemischte Strategie verwendet, für die inneren Knoten wird die Erweiterung und der Bereich minimiert, während für die Blätter die Überlappung minimiert wird. Um Teilungen zu vermeiden, fügt man bei dem R*-Baum Objekte und Teilbäume neu ein, ähnlich dem System einen B-Baum auszugleichen. Sollte eine Teilung unvermeidbar sein, wählt man eine Teilungsachse aus dem Umfeld und reduziert danach die Überlappungen. Wenn vor dem Erstellen eines R*-Baumes bereits eine gefüllte Datenbank vorliegt, kann der Sort-Tile-Recursive (SRT) Algorithmus angewendet werden, um einen noch effizienteren Baum zu bekommen. Bei dem SRT wird vor der Anwendung festgelegt wie viele Objekte in einem Kindknoten enthalten sein dürfen. Danach wird die Objektmenge in Schnitte zerlegt und die Objekte der einzelnen Schnitte rekursiv durchlaufen, bis die maximale Anzahl an Objekten pro Kindknoten erreicht ist und ein neuer begonnen wird. Nachdem jedes Objekt verwaltet wurde, bleibt ein Baum mit geringer Schnittmenge und hoher Effizienz.

2.2.2 Anwendung des Baumes

Um den Baum jetzt effektiv anwenden zu können muss der bekannte DBSCAN Algorithmus angepasst werden. Daher man nun in der Lage ist, die Koordinaten der Objekte abzufragen, muss nicht für jedes Objekt alle Objekte durchlaufen werden, sondern man kann mithilfe des R*-Baumes eine Bereichsanfrage verarbeiten, die die Gesamtzahl der abzufragenden Objekte, gerade bei großen Datenbanken, um ein vielfaches reduziert. Diese Bereichsanfrage funktioniert so, dass die Wurzel abgefragt wird, welchem Pfad zu folgen ist um das Suchrechteck (im englischen Query box) zu finden. Das wird dadurch ermöglicht, dass jeder Knoten des R*-Baumes die Bereichsrechtecke seiner Kindknoten kennt. Auch wenn Überlappungen weitestgehend vermieden werden, treten diese nicht selten auf. Daher diese Überlappungen eine Suchanfrage stören können, werden von alle Kindknoten nach dem Suchrechteck durchsucht und bei einer gegebenen Überlappung, werden diese rekursiv abgearbeitet, bis alle betroffenen Kindknoten berücksichtigt wurden.

Eine Möglichkeit die abzufragenden Objekte von der Index-Liste des R*-Baumes mithilfe SQLite in einem Zwei-Dimensionalen Koordinatenystems wäre:

```
SELECT id FROM Baum-Index
WHERE minX >= x - ϵ AND maxX <= x + ϵ
AND minY >= y - ϵ AND maxY <= y + ϵ;
```

Mit dieser Anfrage werden alle ids, der kleinstmöglichen Teilbereiche des Baumes, die den angefragten Bereich enthalten, aus Baum-Index zurückgegeben. Diese

Liste an ids kann nun verwendet werden, um alle Objekte, innerhalb dieser Bereiche, abzufragen, ob diese erreichbar sind. Daher bei einem Begrenzungsquadrat, des erwünschten Kreisbereiches, noch weitere Objekte enthalten sein können, ist diese weitere Abfrage unablässig, solange man Wert auf genaue Daten legt.

2.2.3 Auswirkung auf den Algorithmus

Nachdem man den R*-Baum in die Datenbank integriert hat und sich, seine Effizienz bei Bereichsanfragen, für DBSCAN nutzbar gemacht hat, ist eine Verbesserung des Laufzeitverhaltens festzustellen.

	einzelne Bereichsanfrage	gesamter Algorithmus
ohne R*-Baum	O(n)	$O(n^2)$
mit R*-Baum	O(log n)	O(n log n)

Diese Effizienzsteigerung ist erklärbar dadurch, dass der R*-Baum eine Laufzeitkomplexität von O(log n) für eine einzelne Bereichsanfrage hat, ohne diese Indexstruktur muss DBSCAN jedes Objekt abfragen, ob dieses in ϵ-Nachbarschaft besitzt oder nicht. Um den gesamten Algorithmus durchzuarbeiten, braucht DBSCAN mit und ohne R*-Baum n Aufrufe der einzelnen Bereichsanfragen.

Der R*-Baum bringt allerdings nicht nur Vorteile, denn auch der Erstellvorgang benötigt Zeit, nämlich O(n log n). Diese Laufzeit kann jedoch meist vernachlässigt werden, denn der Baum wird nur einmal erstellt, DBSCAN hingegen kann beliebig oft angewendet werden und gleicht dadurch diesen Zeitverlust leicht aus. Auch bei dem Einfügen von neuen Objekten in die Datenbank ist eine Effizienzverlust feststellbar, denn das Enfügen benötigt O(log n) Zeit für jedes neu einzufügende Objekt.

Der einzige nennenswerte Nachteil ist der erhöhte Speicherbedarf, denn für jeden Knoten des R*-Baumes, müssen sowohl die Informationen der Kindknoten gespeichert werden, als auch deren Begrenzungskasten.

3 Versuche

Nachdem die Theorie nun verdeutlicht wurde, gilt zu zeigen, wie sich der Algorithmus in der Praxis verhält, dafür wird zuerst der Unterschied zwischen dem DBSCAN ohne Indexstruktur und mit gezeigt. Außerdem sollen die Möglichkeiten des Algorithmus, mithilfe einem konkreten Beispiel, dargestellt werden.

3.1 Vergleich von DBSCAN mit und ohne Indexstruktur

Dass DBSCAN mit einem R*-Baum performanter ist als ohne, wurde in der Theorie bereits erklärt. Um diesen Unterschied besser zu veranschaulichen, stelle man sich nun eine Datenbank mit 10 geographischen Daten vor und ein R*-Baum ist zu dieser bereits gegeben. Der R*-Baum wurde nicht mit dem SRT Algorithmus erstellt, sondern parallel zu der Datenbank gefüllt. Im Anschluss soll DBSCAN auf die Datenbank angewendet werden, um mögliche Cluster in den Daten zu entdecken. Wenn die räumlichen Daten, dieser Datenbank, Familienwohnpositionen auf einer Karte darstellen, können diese Cluster als Reiehnhäuseroder Mehrfamilienhäuser gedeutet werden, je nach strenge der Dichteanfragen und die Rauschpunkte wären Einzelhäuser.

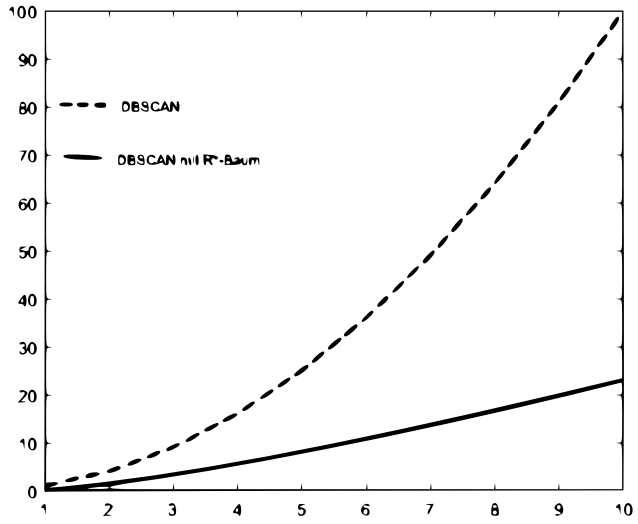

Abbildung 3: Darstellung der Performanz der beiden Algorithmen

Wie man auf dem Graphen eindeutig erkennen kann, ist bereits bei einer Menge von zehn Objekten ein großer Unterschied, zwischen dem klassischen DBSCAN und dem angepassten Algorithmus mit einem R*-Baum, sichtbar.Während die Quadratische Funktion 100 Befehlsaufrufe benötigt, werden von dem verbesserten Algorithmus nur 23 Befehlsaufrufe verwendet. Sollte eine größere Datenbank gegeben sein, ist der klassische DBSCAN-Algorithmus, performanztechnisch, keine Konkurrenz.

3.2 Darstellung anhand eines Beispieles

Um zu veranschaulichen an welchen Datenmengen der R*-Baum Anwendung findet, stelle man sich eine Datenbank mit den Koordinaten von Stadtzentren vor. Die Datenbank kann natürlich noch weitere Spalten mit Informationen über diese Städte enthalten, diese spielen bei der Clusterfindung aber vorerst keine Rolle. Die IDs und die Bereichskoordinaten der Städte werden in den R*-Baum eingetragen, hierbei ist zu beachten, dass bei der Definition des Baumes zwischen 3-11 Spalten, nur ungerade, angegeben werden müssen, darunter die ID und die jeweiligen von-bis Bereiche der Koordinaten, aber nicht über fünf Dimensionen.

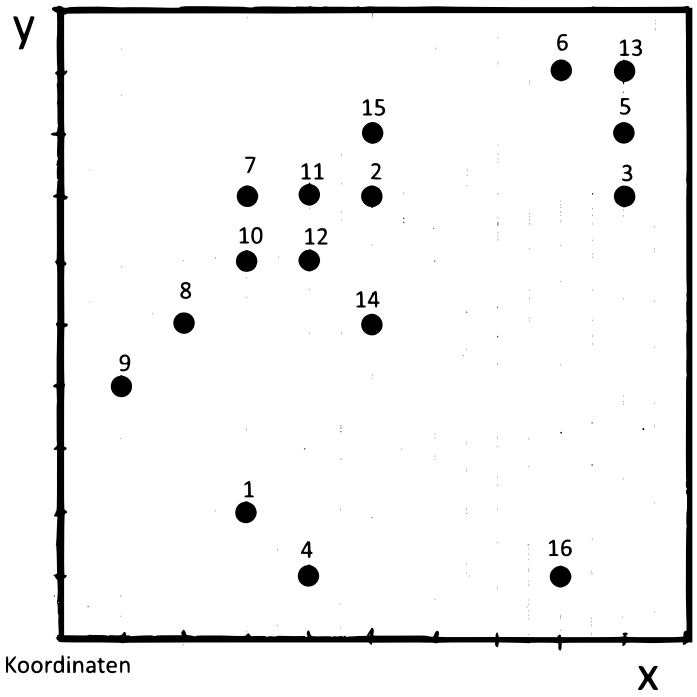

Abbildung 4: Kartenausschnitt mit Einträgen aus der Datenbank

Für die weitere Veranschaulichung wird der dargestellte Kartenausschnitt betrachtet. Die Zahlen stehen für die Indizes der Städte. Als ϵ wird der Abstand 1 und für $minPts$ wird 2 gewählt. Der veränderte DBSCAN-Algorithmus bearbeitet nun jede Stadt nacheinander, indem zuerst alle Rückgabewerte des R*-Baumes, für den Bereich ϵ um die aktuelle Stadt, nach ϵ-erreichbaren Städten abgesucht wird. All diese Nachbarstädte werden dann verarbeitet um festzulegen, dass sie in demselben Cluster sind.

Nachdem DBSCAN abgeschlossen ist, können alle Cluster mit den zugehörigen Städte-IDs ausgegeben werden. Für den abgebildeten Kartenausschnitt wären das:

- Cluster 1 enthält IDs: 1, 4
- Cluster 2 enthält IDs: 2, 7, 8, 9, 10, 11, 12, 14, 15
- Cluster 3 enthält IDs: 3, 5, 6, 13
- Rauschpunkte sind IDs: 16

Diese Clusterinformationen können anschließend weiterverarbeitet werden, um beispielsweise zu erkennen, dass Cluster 2 das größte Städtegebiet ist, oder wenn diese Auswertung mit einer Älteren verglichen wird, können Entwicklungen festgestellt werden.

4 Schlussfolgerung

Nachdem die Vorzüge der Verwendung eines R*-Baumes für DBSCAN nahegelegt wurden, ist zu sehen, dass Indexstrukturen mit der Größe der Datenmenge, die eingespeist werden, an Effektivität, gegenüber den klassischen Algorithmen, dazugewinnen. Sollten große Datenbanken verwaltet werden, gilt es zu entscheiden, ob die Zunahme in Speicherdaten, durch die Indexstrukturen, hinnehmbar ist, oder ob ein geringerer Speicherbedarf, der schnelleren Verarbeitungszeit zu bevorzugen ist.

Literatur

[1] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, Bernhard Seeger. The R*-tree:an efficient and robust acces method for points and rectangles. ACM SIGMOD, 1990.

[2] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Siaowei Xu. A density-based algorithm for discovering clusters in large spatial database with noise. KDD.96 Proceedings, 1996.

[3] Scott T. Leutenegger, Jeffrey M. Edgington, Mario A. Lopez. STR: A SIMPLE AND EFFICIENT ALGORITHM FOR R-TREE PACKING. NASA, 1997.

[4] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, Jörg Sander. OPTICS: Ordering Points To Identify the Clustering Structure. ACM SIGMOD, 1999.

[5] L. Kaufman, P.J. Roussew. Finding Groups in Data - An Introduction to Cluster Analysis. John Wiley and Sons, 1990.

[6] Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth. From Data Mining to Knowledge Discovery in Databases. AI Magazine, 1996.

[7] T. Sellis, R. Roussopoulos, C. Faloutsos. The R*-tree: A dynamic index for multidimensional objects. In Proceedings of Very Large Data Bases, 1987.

[8] Martin Ester, Jörg Sander. Knowledge Discovery in Databases: Techniken und Anwendungen. Springer-Verlag, 2013.

[9] Erich Schubert, Arthur Zimek, Hans-Peter Kriegel. Generalized Outlier Detection with Flexible Kernel Density Estimates. SDM, 2014.

[10] Erich Schubert, Martin Ester, Hans-Peter Kriegel, Xiaowei Xu. DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN. ACM Transactions on Database Systems. 2017.