Bauhaus-Universität Weimar
Fakultät Medien
Studiengang Medieninformatik

Approximative metrische Suche mit Hashfunktionen in hochdimensionalen Datensätzen

Bachelorarbeit

Jonas Köhler
geb. am: 01.06.1989 in Kassel

Matrikelnummer 112373

1. Gutachter: Prof. Dr. Benno Stein
2. Gutachter: Prof. Dr. Volker Rodehorst

Datum der Abgabe: 4. Juli 2016

Erklärung

Hiermit versichere ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Weimar, 4. Juli 2016

. . . . . .
Jonas Köhler

Zusammenfassung

Die Suche nach dem nächsten Nachbarn in einer großen Menge von Daten ist ein klassisches Problem der Informatik. Eine besondere Relevanz hat es für die Suche nach Beinahe-Duplikaten in großen Dokumentensammlungen. Da diese Suche jedoch schwer für hochdimensionale Daten ist, gibt es approximative Suchverfahren, um das Problem in der Praxis lösen zu können. Eines dieser Verfahren besteht aus Hashfunktionen, die ähnliche Objekte auf gleiche Zielwerte abbilden. In dieser Arbeit wird nach einer Motivierung für das Problem im Hinblick auf die Suche nach Beinahe-Duplikaten dieses hashbasierte Suchverfahren erörtert. Fordert man eine bestimmte Qualität an die Approximation, werden weiterhin für eine Teilmenge dieser Verfahren Ober- und Untergrenzen der Suchkomplexität bewiesen. Dieses Ergebnis wird abschließend auf das ursprüngliche Problem bezogen, gute Hashfunktionen für die approximative Suche zu finden.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Suche nach ähnlichen Dokumenten in einer Kollektion von Dokumenten ist spätestens seit der flächendeckenden Verbreitung von Suchmaschinen wie Google einem breiten Publikum bekannt. Eine Grundprimitive von solchen Suchmaschinen ist das Problem, zu einer Suchanfrage diejenigen Dokumente herauszufinden, die zur Anfrage bezüglich eines Ähnlichkeitsbegriffs besonders ähnlich sind. Der naive Ansatz, alle Dokumente nach und nach mit der Anfrage zu vergleichen und dann diejenigen zurückzugeben, die ähnlich genug sind, scheitert gerade bei vielen praxisrelevanten Dokumentenkollektionen an der schieren Menge der Dokumente. Selbst bei verhältnismäßig kleinen Kollektionen gibt es Szenarien, in denen dieser Ansatz der *linearen Suche* aufgrund der Suchzeit nicht machbar ist. Ein Beispiel ist die automatisierte Plagiatserkennung. Um in einer größeren Kollektion wissenschaftlicher Aufsätze potentielle Plagiate zu finden, müssen zunächst Kandidatenmengen hochähnlicher Dokumente gefunden werden, die dann in einem zweiten Schritt von menschlichen Domänenexperten detailliert untersucht werden. Würde man hierzu die lineare Suche nutzen, bliebe einem nichts anderes übrig, als alle n Dokumente untereinander zu vergleichen. Der dabei entstehende Zeitaufwand von $\mathcal{O}\left(n^2\right)$ ist selbst für moderat große Kollektionen nicht mehr leistbar. Für niedrigdimensionale Datensätze, wie sie z.B. in der Computergrafik vorkommen, gibt es für die Suche nach ähnlichen Objekten zu einem Anfrageobjekt sehr effektive Datenstrukturen, die sublineare Suchzeiten ermöglichen. Die Repräsentierung von Dokumenten, z.B. durch Wortvektoren, führt aber oft zu Datensätzen mit mehr als hundertausend Dimensionen, in denen o.g. Datenstrukturen versagen und selten bessere Suchzeiten als die lineare Suche bieten.

Hashfunktionen als Ausweg Um dennoch Kandidatenmengen von hochähnlichen Objekten in solch hochdimensionalen Kollektionen in sublinearer Zeit zu finden, gibt es seit Ende der 1990er-Jahre das Konzept des *locality-*

sensitive Hashing (LSH). Hier werden Hashfunktionen konstruiert, die ähnliche Objekte mit hoher Wahrscheinlichkeit auf die selben Hashwerte abbilden. Speichert man die Objekte nun in einer Hashtabelle unter dem gegebenen Hashwert, liefert das Evaluieren der Hashfunktion und das Nachschauen an der entsprechenden Stelle der Hashtabelle Kandidaten, die mit hoher Wahrscheinlichkeit hochähnlich sind. Sie bieten zwar keine Garantie dafür, alle hochähnlichen Nachbarn eines Objekts zurückzugeben, bzw. dafür dass alle zurückgegebenen Objekte tatsächlich hochähnlich sind. Für die Praxis bilden sie aber dennoch eine gute Approximation und oft die einzige Möglichkeit, das Problem in sehr großen und hochdimensionalen Datensätzen, wie sie z.B. bei der Websuche vorkommen, zu lösen.

Fragestellungen der vorliegenden Arbeit Seit der wegweisenden Arbeit von Indyk und Motwani (1998) haben sich sehr unterschiedliche Weiterentwicklungen des Verfahrens herausgebildet. In Kapitel 2 dieser Arbeit wird das Prinzip der Suche mit Hashfunktionen zusammenfassend genauer erläutert und eine grobe Einordnung der verschiedenen Verfahren vorgenommen. Da mit einer Hashfunktion in den meisten Fällen nur eine unzureichende Menge von Kandidaten zurückgegeben werden kann, gibt es Erweiterungen des Hashing-Ansatzes, um damit umzugehen. Eine davon ist das Hashen der Daten in mehrere Tabellen mit verschiedenen, sich bestenfalls komplementär ergänzenden Hashfunktionen. Die Anzahl der verwendeten Tabellen ist entscheidend. Benutzt man zu wenige Tabellen, werden weiterhin zu wenige Kandidaten zurückgegeben. Benutzt man jedoch mehr Tabellen als notwendig, erhöht sich die Suchzeit signifikant. Die genaue Anzahl der Tabellen, die für einen Datensatz minimal ausreichend wären, ist bisher nicht bekannt. In Kapitel 3 wird daher zunächst ein diskretes Modell der hashbasierten Suche im Mehrtabellen-Ansatz entworfen und mit seiner Hilfe diese Anzahl durch eine Ober- und eine Untergrenze abgeschätzt. Diese Abschätzungen eröffnen nun die Perspektive zur Bestimmung der gesuchten Minimalanzahl abhängig vom gegebenen Datensatz. In Kapitel 4 werden die theoretischen Ergebnisse schließlich hinsichtlich ihrer praktischen Relevanz anhand von Experimenten auf Testdaten evaluiert. In den folgenden Abschnitten der Einleitung wird nun die Motivation des Problems etwas konkretisiert und die Definition einiger Begriffe vorgenommen, die für eine tiefere Diskussion in den folgenden Kapiteln notwendig sind.

1.1 Motivation: Near Duplicate Detection

Als Motivation für das Problem, hochähnliche Nachbarn in einer Dokumentenkollektion zu finden, bietet sich die Suche nach Beinahe-Duplikaten (*Near-*

Duplicate-Detection) an. Um für eine Kollektion von Dokumenten zu entscheiden, ob zu einer Anfrage exakte Duplikate existieren, reicht es, mit einer klassischen Hashfunktion (z.B. dem Message-Digest Algorithmus *MD5* (Rivest, 1992)) einen binären Fingerabdruck der Anfrage zu erzeugen. Nun schaut man, ob sich dieser in der Menge von Fingerabdrücken befindet, die sich aus allen Dokumenten der Kollektion bilden lässt. Das Anlegen der Menge von Fingerabdrücken für die Kollektion hätte einen einmaligen Zeit- und Platzbedarf von $\mathcal{O}(n)$. Die Suchanfrage ließe sich bei einer guten Hashfunktion in $\mathcal{O}(1)$ entscheiden. Es kann aber vorkommen, dass zwei Dokumente semantisch und sogar syntaktisch hochähnlich sind, sich aber nur in einem kleinen Detail unterscheiden. Dies kann zufällig (z.B. könnten bei einer Webseite durch einen Formatierungsfehler deutsche Umlaute mit anderen Symbolen ersetzt worden sein) oder gewollt geschehen (z.B. bei der gezielten Verfremdung einer Passage in einem Plagiat). Da klassische Hashfunktionen konstruiert wurden, um Kollisionen unterschiedlicher Eingangswerte sehr unwahrscheinlich zu machen, wären die Fingerabdrücke mit höchster Wahrscheinlichkeit unterschiedlich, obwohl ein Mensch die Dokumente als ähnlich ansehen würde. In diesem Fall ist es also nötig, in der Kollektion zu suchen und die Kandidaten hinsichtlich ihrer Ähnlichkeit zum Anfragedokument zu evaluieren. Zwei Anwendungsszenarien, in denen die Erkennung von Beinahe-Duplikaten wesentlich sind, sind die Untersuchung, ob Textpassagen wiederverwendet wurden und die Optimierung des Suchindex einer Dokumentensuchmaschine.

Automatisierte Plagiaterkennung Ein großes Problem im wissenschaftlichen und universitären Umfeld ist das zu eigen machen und Publizieren fremder Gedankeninhalte, ohne diese explizit zu kennzeichnen. Ein solcher Diebstahl von Ideen ist innerhalb der wissenschaftlichen Gemeinschaft ein schweres Vergehen und wird in den meisten erkannten Fällen mit dem Entzug der Anerkennung jeder weiteren wissenschaftlichen Arbeit bestraft. Sollte das Vergehen bei der Erlangung eines Titels, wie etwa einer Promotion, begangen worden sein, kann dies zu seiner Aberkennung führen. Trotz strenger Konsequenz bei der Durchsetzung solcher Bestrafungen, sind solche Plagiate weiterhin ein akutes Problem, wie die Diskussion um die Aberkennung akademischer Titel von deutschen Politikern in den letzten Jahren zeigte. Eine Studie von McCabe, die 2005 veröffentlicht wurde, offenbarte, dass 50% der 18.000 untersuchten Studenten zugaben, Plagiate in ihren Arbeiten zu verwenden (McCabe, 2005). Da Plagiate sich gerade dadurch auszeichnen, dass bei ihnen Referenzen nicht angegeben werden, ist ihre Identifizierung oft schwierig, zeitaufwändig und benötigt die Mithilfe vieler fachkundiger Gutachter. Diese müssen in der Lage sein, die plagiierten Quellen in den betreffenden Stellen eines Dokuments zu erkennen. Dies ist in Anbetracht der Anzahl der Publikationen in den meisten

Wissenschaftsdomänen selten mit hinreichender Qualität durchführbar. Bei einer automatisierten Plagiatserkennung wird daher algorithmisch versucht ,in einer Kollektion z.B. wissenschaftlicher Aufsätze Kandidatenmengen hochähnlicher Passagen zu bilden, die dann von der begrenzten Anzahl an Experten des jeweiligen Fachgebiets genauer untersucht werden können. Die Experten müssen nun für eine gegebene Arbeit nicht mehr selbstständig die gesamte potentielle Referenzliteratur durchsuchen, sondern können sich auf die vorgeschlagenen Kandidaten konzentrieren (Maurer, Kappe und Zaka, 2006).

Suchindex-Optimierung Ein anderer Anwendungsfall für die Erkennung von Beinahe-Duplikaten ist die Optimierung des Suchindex einer Websuchmaschine. Nach heutigen Erkenntnissen geht man davon aus, dass 30–40% aller Webseiten im Internet zu einem überwiegenden Anteil Duplikate anderer Seiten sind (Broder u. a., 1997; Shivakumar und Garcia-Molina, 1999). Damit bei einer Suchmaschine nicht unnötig Duplikate durchsucht und zurückgegeben werden, ist es nötig, diese bei der Anlegung des Suchindex herauszufiltern. In der Praxis sind diese Duplikate jedoch syntaktisch nicht identisch. So können unterschiedliche String-Formatierungen, zusätzliche Informationen auf der Webseite, oder einfach unterschiedliche HTML-Repräsentierungen denselben Inhalt in unterschiedlicher Form darstellen. Auch hier hilft die o.g. Suche nach exakter Identität nicht weiter: es müssen mehr oder weniger hochähnliche Dokumente identifiziert werden können.

1.2 Metrische Räume und Ähnlichkeiten

Um über die Begriffe *Abstand* und *Ähnlichkeit* präzise sprechen zu können, ist es wichtig, sie im Vorfeld der weiteren Diskussion zu definieren. Im mathematischen Feld der Analysis gibt es dazu das Konzept eines metrischen Raums, der die natürliche Verallgemeinerung des intuitiven Abstandsbegriffs darstellt. Aus seiner Definition lässt sich durch weitere Verallgemeinerung auch der Begriff einer Ähnlichkeit präzise fassen. Um die Definition nicht leer vorzustellen, werden im Anschluss einige Beispiele von metrischen Räumen gegeben, die im Zuge dieser Arbeit bzw. für die Informatik insgesamt eine Relevanz besitzen.

Definition von metrischen Räumen und Ähnlichkeiten Ein *metrischer Raum* (M, d) ist eine Menge M zusammen mit einer *Metrik* $d\colon M \times M \to \mathbb{R}_{\geq 0}$. Eine Metrik verallgemeinert den Begriff eines Abstands zwischen zwei Punkten und muss lediglich erfüllen:

1. Gleiche Objekte $x, y \in M$ haben keinen Abstand:

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

2. Der Abstand zwischen Objekten $x, y \in M$ ist unabhängig vom Standort der Betrachtung:
$$d(x, y) = d(y, x)$$

3. Der kürzeste Weg zwischen zwei Objekten $x, y \in M$ ist immer der direkte und nicht ein Umweg über ein drittes Objekt $z \in M$:
$$d(x, z) + d(z, y) \geq d(x, y)$$

Das Axiom 3 ist eine sehr geometrische Forderung und wird auch *Dreiecksungleichung* genannt. Interessiert es einen nur, ob Objekte x und y mehr oder weniger ähnlich zueinander sind, wird durch Weglassen des Axioms die Funktion d zu einer *Ähnlichkeit*. Für semantische Konsistenz sagt man hierbei im Widerspruch zur vorigen Definition, dass $d(x, y) = 0$ minimale und $d(x, y) = 1$ maximale Ähnlichkeit (Identität) bedeutet. Hierbei ist anzumerken, dass sich jede Metrik d durch die Transformation

$$s(x, y) = \frac{1}{1 + d(x, y)}$$

in eine Ähnlichkeit s umwandeln lässt.

Euklidische Vektorräume Das einfachste Beispiel für einen metrischen Raum ist der euklidische Vektorraum $\mathbb{R}^n$ zusammen mit der Metrik

$$d(x, y) = \left(\sum_{i=0}^{n} (x_i - y_i)^2 \right)^{\frac{1}{2}}$$

Euklidische Vektorräume entsprechen den geometrischen Räumen unserer Anschauung. Die Semantik ihrer Metrik ist daher unmittelbar klar.

Kosinus-Ähnlichkeit und Winkelabstand Im Information-Retrival wird zum Vergleich zweier Dokumente d_1, d_2 in Vektorraum-Repräsentation die *Kosinus-Ähnlichkeit* zwischen ihren tf-idf Vektoren $v(d_1)$, $v(d_2) \in \mathbb{R}^D$ berechnet (vgl. folgenden Abschnitt 1.3):

$$sim(d_1, d_2) := \frac{\langle v(d_1), v(d_2) \rangle}{|v(d_1)| \cdot |v(d_2)|} = 1 - \cos \phi$$

wobei ϕ den Winkel zwischen den Vektoren $v(d_1)$ und $v(d_2)$ bezeichne. Es gilt

$$sim(d_1, d_2) = 1$$

für Dokumente mit identischer Wortverteilung und

$$sim(d_1, d_2) = 0$$

für Dokumente, die gar keine Worte miteinander teilen. Gegenüber dem euklidischen Abstand bietet das Maß den Vorteil, dass es unabhängig von der Länge der Vektoren ist und nur die relative Wortverteilung der beiden Dokumente berücksichtigt (Manning, Raghavan und Schütze, 2008, p. 120-123). Die Kosinus-Ähnlichkeit ist keine Metrik, da die Dreiecksungleichung (Axiom 3) nicht erfüllt ist. Sie lässt sich aber leicht in den *Winkelabstand* überführen, der wiederum eine Metrik ist:

$$d_{\sphericalangle}(u, v) := \frac{\phi}{\pi} = 1 - \frac{\arccos\big(sim(u, v)\big)}{\pi}$$

Hamming-Metrik In der Informationstheorie wird die Ähnlichkeit zweier Binärcodes $x = x_1 \cdots x_l$, $y = y_1 \cdots y_l$ der Länge l durch ihren *Hamming-Abstand* gemessen

$$d_H(x, y) = \sum_{i=0}^{l} x_i \oplus y_i$$

Die Menge aller Codes der Länge l zusammen mit dieser Metrik bildet den *Hamming-Raum*. Die Hamming-Metrik findet ihre Anwendung z.B. in der Codierungstheorie von digitalen Signalen (MacKay, 2002).

Jaccard-Metrik Um den Unterschied zwischen Mengen zu messen, wurde die *Jaccard-Metrik* (Levandowsky und Winter, 1971) entwickelt. Für zwei endliche Mengen X und Y ist sie definiert als:

$$d_J(X, Y) = \frac{\big|X \cap Y\big|}{\big|X \cup Y\big|}$$

Sie misst die Anzahl gemeinsamer Elemente der Mengen im Verhältnis zur Gesamtzahl in beiden Mengen enthaltenen Elemente. Die Jaccard-Metrik wird z.B. in der Near-Duplicate Erkennung eingesetzt, um die Ähnlichkeit zweier Textabschnitte zu quantifizieren (Manning, Raghavan und Schütze, 2008, S. 438). Eine hohe Jaccard-Ähnlichkeit der Mengen aller n-Gramme[1] in beiden Abschnitten liefert hierbei Kandidaten, bei denen es sich mit hoher Wahrscheinlichkeit um Duplikate handelt.

1 z.B. alle Sequenzen von n aufeinanderfolgenden Wörtern des Textabschnitts

Levenshtein-Abstand Eine sehr andere Metrik von großer Relevanz für die Informatik ist die *Levenshtein-* oder auch *Edit*-Distanz (Levenshtein, 1966). Sie misst den Unterschied zwischen zwei Zeichenketten a und b durch die minimal möglich Anzahl an Änderungen, die erforderlich sind, um a in b umzuwandeln. Eine erlaubte Änderung kann sein:

- Das Einfügen von Zeichen an beliebiger Stelle
- Das Löschen von Zeichen an beliebiger Stelle
- Das Ersetzen von Zeichen an beliebiger Stelle

Der wichtigste Unterschied zu den ersten beiden zuvor genannten Beispielen ist, dass diese Metrik nicht auf einem Vektorraum definiert ist und eine geometrische Interpretierung daher schwer fällt. Die Metrik spielt eine große Rolle im approximativen String-Matching (Navarro, 2001).

Abbildung 1.1: Die Levenshtein Distanz der Wörter *kitten* und *sitting* ist 3. (Wikipedia, 2016)

kitten → **s**itten
sitt**e**n → sitt**i**n
sittin → sittin**g**

1.3 Ähnlichkeitsräume im Information-Retrieval

Dokumente lassen sich auf unterschiedliche Weise in einen metrischen Raum (bzw. einen Ähnlichkeitsraum) einbetten. Eine weitverbreitete Variante bettet Dokumente zusammen mit der im vorigen Abschnitt eingeführten Kosinusähnlichkeit in einen hochdimensionalen reellen Vektorraum ein.

Dokumentenkollektionen Eine Dokumentenkollektion wird dabei als eine Menge von Dokumenten

$$C = \{d_1 \dots d_l\}$$

begriffen, die wiederum aus einer Multimenge von einzelnen Termen

$$d_i = \{t_1 \dots t_k\}$$

bestehen. Die Terme können z.B. die enthaltenen Wörter, oder aber auch durch Stammformreduktion (*stemming*) reduzierte Token sein. Da d_i Multimengen sind, wird die Sortierung der Terme im Dokument nicht beachtet. Gleiche Terme können aber mehrfach in d_i auftreten, daher bleibt die Information über ihre Häufigkeit in diesem *Bag-of-Word*-Modell der Dokumente enthalten.

Termfrequenz und inverse Dokumentfrequenz Zu jedem Term t lässt sich nun ermitteln, wie oft dieser Term im Dokument auftritt, dies ist die sogenannte *Termfrequenz*

$$tf_d(t) = \frac{\left|\{t_i \mid t_i = t \wedge t_i \in d\}\right|}{|d|}$$

Sie ist ein Gewichtungsfaktor dafür, wie relevant dieser Term für das gegebene Dokument ist. Kommt ein Term sehr oft im Dokument vor, wird er als wichtig angesehen und hat damit eine hohe Termfrequenz. Kommt er dagegen selten bis gar nicht vor, ist die Termfrequenz gering und der Term wird als irrelevant für das Dokument betrachtet. Die reine Termfrequenz allein wäre wenig aussagekräftig, um die Bedeutung zweier Dokumente untereinander bzw. um Dokumente nach ihrer Relevanz bezüglich einer Anfrage zu unterscheiden. Ist sie für einen Term z.B. aufgrund der Domäne der Dokumente über alle Dokumente hinweg sehr hoch, liefert sie kein Diskriminierungsmerkmal. Daher wird zusätzlich betrachtet, wie oft der Term insgesamt in der gesamten Kollektion vorkommt. Definiert man die *Dokumentfrequenz* df_t als die Anzahl der Dokumente, die den Term t enthalten, lässt sich nun die *inverse Dokumentfrequenz* eines Terms t definieren als

$$idf_t = \log \frac{l}{df_t}$$

Dieser Wert ist groß, wenn der Term t in der Gesamtkollektion relativ selten auftritt und niedrig, wenn er über alle Dokumente hinweg häufig erscheint. Kombiniert man nun beide Gewichtungen, ergibt sich als Produkt der sogenannte *tf-idf* Wert eines Terms t in einem Dokument d

$$\textit{tf-idf}_t = tf_d(t) \cdot idf_t$$

Ist dieser Wert hoch, bedeutet dies zum einen, dass der Term t häufig in d vorkommt und darüber hinaus eine hohe Diskriminanz zwischen den Dokumenten ermöglicht, da er in anderen Dokumenten kaum vorkommt. Ist der Wert dagegen niedrig, bedeutet dies entweder, dass der Term kaum oder gar nicht im Dokument erscheint, oder dass er nicht hilfreich dafür ist, das Dokument von anderen zu unterscheiden.

***tf-idf*-Vektoren und semantische Ähnlichkeit** Sind $t_1, \ldots, t_D$ nun alle in C vorkommende (unterschiedliche) Terme, lässt sich ein Dokument $d \in C$ in den D-dimensionalen Vektorraum $\mathbb{R}^D$ einbetten. Dazu trägt man in der i-ten Komponente des zum Dokument gehörenden Vektors $v(d) \in \mathbb{R}^D$ den *tf-idf* Wert des Terms t_i ein:

$$v(d) = \left(\textit{tf-idf}_{t_1}, \ldots, \textit{tf-idf}_{t_D}\right)^T \in \mathbb{R}^D$$

In einer größeren Dokumentenkollektion treten sehr viele unterschiedliche Terme auf. Folglich ist D in der Praxis sehr groß ($D > 100000$). Die Komponenten der *tf-idf*-Vektoren haben in den meisten Kollektionen fast überall den Wert 0, da das Verhältnis der Anzahl der im Dokument vorkommenden Terme zur Anzahl global vorkommender Terme klein ist. Die Dimensionen, die signifikant von 0 abweichen, bilden somit die für das Dokument relevanten Terme. Motiviert durch die Annahme, dass semantisch ähnliche Dokumente eine ähnliche Verteilung für sie relevanter Terme besitzen, lassen sich zwei Dokumente nun im mathematischen Sinne als ähnlich beschreiben, wenn sie in eine ähnliche Richtung des $\mathbb{R}^D$ zeigen. Die im vorigen Abschnitt eingeführte Kosinus-Ähnlichkeit misst genau dieses Verhältnis der Vektoren zueinander. Damit ergibt sich ein Ähnlichkeitsraum, in dem die mathematisch modellierte Ähnlichkeit mehr oder weniger der semantischen Ähnlichkeit der Dokumente entspricht (Manning, Raghavan und Schütze, 2008, Kap. 6).

1.4 Suche nach nächsten Nachbarn

In diesem Abschnitt wird das algorithmische Problem der metrischen Suche präzise definiert und es werden die klassischen Ansätze zu seiner Lösung vorgestellt und im Bezug auf hochdimensionale Datensätze problematisiert.

Nächste-Nachbarn-Anfragen Für einen metrischen Raum M und einen Datensatz $X \subset M$ versucht man bei der *Suche nach dem nächsten Nachbarn* (NN-Suche) zu einem *Anfragepunkt* $q \in M$ denjenigen *Datenpunkt* $x \in X$ zu finden, der den Abstand zu q minimiert (Chávez u. a., 2001). Also:

$$x = \arg\min_{x \in X} d(q, x)$$

Ein Beispiel für dieses Suchproblem ist die Suche nach dem ähnlichsten Dokument in einer Datenbank zu einem gegebenen Suchstring.

r-Umgebungsanfragen Bei der *r-Umgebungssuche* dagegen gibt man einen Abstand $r > 0$ vor und definiert Punkte x, y mit $d(x, y) < r$ als benachbart und Punkte mit $d(x, y) \geq r$ als unbenachbart. Zum angefragten Punkt $q \in M$ sollen nun alle Datenpunkte $x \in X$ zurückgegeben werden, die zu q benachbart sind (Chávez u. a., 2001). Ein Beispiel für dieses Suchproblem ist die Plagiatssuche, bei der man alle Dokumente genauer untersuchen möchte, die einen Ähnlichkeitsschwellwert übersteigen.

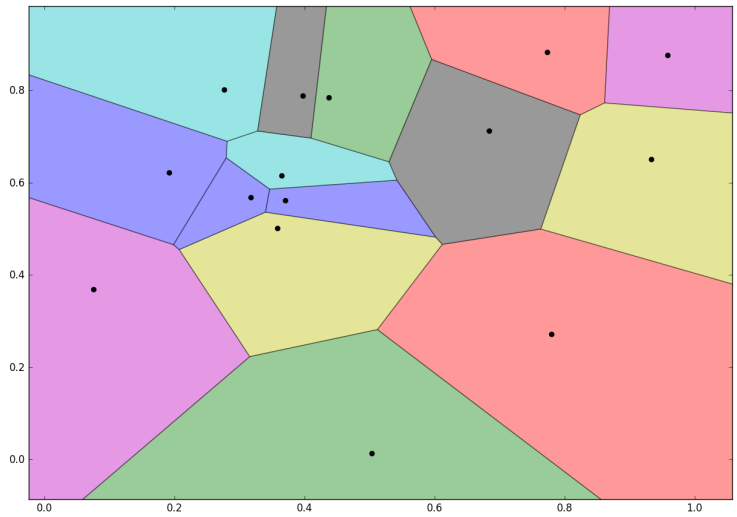


Abbildung 1.2: *Suche nach dem nächsten Nachbarn*: Ein *Voronoi Diagramm* illustriert, in welchen Bereichen des metrischen Raums (hier der $\mathbb{R}^2$) welcher Datenpunkt zurückgegeben werden muss. Für alle Punkte der zusammenhängenden farbigen Gebiete muss der enthaltene Datenpunkt (schwarzer Kreis) zurückgegeben werden.

Lösung der Suchprobleme Der naive Ansatz zum Lösen der Suchprobleme ist die *lineare Suche*. Bei ihr werden alle Datenpunkte nacheinander durchgegangen und evaluiert, ob sie zurückgegeben werden sollen. Wird eine r-Umgebungsanfrage beantwortet, muss dafür lediglich überprüft werden, ob der Abstand kleiner als r ist. Bei einer Suche nach dem nächsten Nachbarn wird wiederum geschaut, ob der aktuelle Kandidat einen geringeren Abstand besitzt als der vorige Kandidat mit kleinstem Abstand. Die Suchzeit wächst so linear mit der Anzahl der Datenpunkte und beträgt daher $\mathcal{O}(n)$. Wie in der Einleitung erwähnt, ist diese lineare Suchzeit für viele praktische Anwendungen und Datensätze zu hoch. In der Vergangenheit wurden daher viele Verfahren entwickelt, die in Anwendungen (z.B. in der Computergrafik, in räumlichen Datenbanken) garantierte Suchzeiten von $\mathcal{O}(\log n)$ erreichen können. Das prominenteste Beispiel hierfür ist ein *kd-Baum* (Bentley, 1975). Eine vollständige Erörterung dieser Verfahren würde den Rahmen dieser Arbeit übersteigen. Eine umfassende Übersicht liefern hier Chávez u. a. (2001) bzw. Samet (1990).

Probleme mit hochdimensionalen Vektorräumen Alle bekannten Verfahren für die exakte metrische Suche besitzen ihre sublinearen Garantien nur

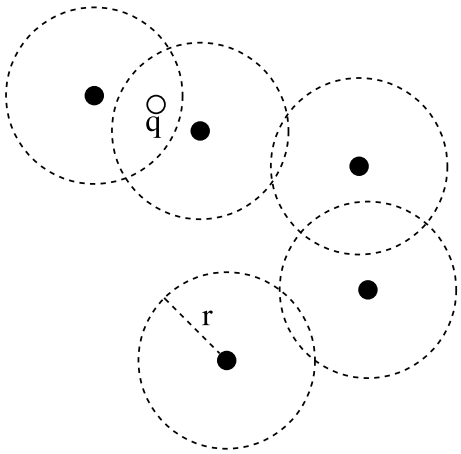


Abbildung 1.3: *r-Umgebungssuche*: Äquivalent zur Definition lässt sich der Datensatz als eine Überdeckung von n verschiedenen r-Bällen begreifen, deren Zentren die Datenpunkte aus X sind. Die Suche soll nun die Zentren aller Bälle zurückgeben in denen q enthalten ist.

für Datensätze mit einer nicht allzu großen *Dimension*[2] (Weber, Schek und Blott, 1998). Für Datensätze mit höherer Dimension konvergiert die Suchzeit sehr schnell gegen die einer linearen Suche. Die unterschiedlichen Gründe für diesen *curse of dimensionality* werden in Weber, Schek und Blott (1998), Aggarwal, Hinneburg und Keim (2001) oder Beyer u. a. (1999) diskutiert. Eine Dimensionszahl beschreibt die Anzahl an Freiheitsgraden des Datensatzes und ist je nach Kontext unterschiedlich definiert. Ein naiver Begriff der Dimension ist die klassische Vektorraum-Dimension[3] D des Raums $\mathbb{R}^D$, in den die Daten eingebettet sind. Die Praxis zeigt jedoch, dass Dimensionsbegriffe existieren, die im Bezug auf einen Datensatz eine deutlich präzisere Beschreibung der Komplexität liefern. Beispielsweise ließen sich zufällig gewählte Punkte aus der reellen Zahlengerade $\mathbb{R}$ auch auf einer Gerade in den $\mathbb{R}^{1000}$ einbetten. Kennt man diese Einbettung, lässt sich ein Anfragepunkt q orthogonal auf diese Gerade projizieren, womit sich eine Nachbarschaftssuche auf eine simple Binärsuche in der Zahlengerade reduziert. Werden die Punkte dagegen aus einer beliebigen Verteilung des $\mathbb{R}^{1000}$ gewählt, ist dies im Allgemeinen nicht mehr möglich. Das Beispiel illustriert die sogenannte *Mannigfaltigkeits-Hypothese*

2 Die genaue Größe ist Gegenstand der Diskussion. Verschiedene Autoren sprechen von $D < 10$, andere dagegen von $D < 30$, bzw. sogar $D < 100$. Einig ist man sich bei $D > 1000$. Da die meisten Anwendungen im Information Retrieval Einbettungsräume mit 100000 Dimensionen und mehr haben, ist hier das Problem besonders relevant.

3 die maximale Anzahl linear unabhängiger Vektoren in einem Erzeugendensystem

für hochdimensionale Datensätze. Sie besagt, dass in der Praxis vorkommende hochdimensionale Datensätze in der Nähe einer niedrigdimensionalen Untermannigfaltigkeit angesiedelt sind (Burges, 2009). Motiviert dadurch existieren verschiedene Begriffe einer *intrinsischen Dimension* des Datensatzes (Camastra, 2003), die versuchen die metrische Komplexität der Daten, unabhängig von ihrer Einbettung als Untermannigfaltigkeit des hochdimensionalen Raums, zu beschreiben. Eine jüngst eingeführte Größe, bei der gezielt die Schwierigkeit der NN-Suche untersucht wird, ist der *relative Kontrast* des Datensatzes (He, Kumar und S.-F. Chang, 2012). Er wird für den Anfragepunkt q und den Datensatz X definiert als

$$C^q = \frac{d^q_{mean}}{d^q_{min}} = \frac{\mathbb{E}\left[d(x,q)\right]}{\min_{x \in X} d(x,q)}$$

Werden die Anfragepunkte nun zufällig aus einer Verteilung gesamplet ergibt sich der relative Kontrast des gesamten Datensatzes im Bezug auf diese Anfrageverteilung:

$$C = \frac{\mathbb{E}\left[d^q_{mean}\right]}{\mathbb{E}\left[d^q_{min}\right]}$$

He, Kumar und S.-F. Chang konnten in ihrem Artikel zeigen, dass diese Größe ein gutes Maß für die zu erwartende Suchzeit ist. Ein hoher relativer Kontrast bedeutet dabei, dass die Datenverteilung günstige Eigenschaften für eine metrische Suche besitzt. Ein niedriger relativer Kontrast dagegen indiziert einen zunehmend bedeutungsloseren Abstandsbegriff. Beim Verzicht auf eine exakte Suche gibt es dennoch Möglichkeiten, das Problem der Suche nach dem nächsten Nachbarn auch in hohen Dimensionen mit einer für die Praxis hinreichenden Qualität zu approximieren. Eine Möglichkeit, dies zu erreichen, ist die geschickte Anwendung von Hashfunktionen. Dies soll im folgenden Kapitel erläutert werden.

Kapitel 2

Hashfunktionen für die approximative NN-Suche in hohen Dimensionen

In diesem Kapitel werden Hashfunktionen für die approximative Lösung des zuvor erwähnten Suchproblems in hohen Dimensionen vorgestellt. Im ersten Abschnitt werden der Begriff der Approximation präzisiert, die Grenzen der approximativen Suche mithilfe von Hashfunktionen aufgezeigt und der Unterschied zwischen datenabhängigen und datenunabhängigen Hashfunktionen erläutert. Im zweiten Abschnitt des Kapitels werden verschiedene Konstruktionsprinzipien für Hashfunktionen untersucht, anhand derer im dritten Abschnitt die bestehenden Verfahren eingeordnet werden. Im letzten Abschnitt werden auf den Hashfunktionen aufbauende Suchverfahren, das Recall-Problem dieser Hashverfahren und Möglichkeiten zu dessen Lösung vorgestellt.

2.1 Approximative Suche mit Hashfunktionen

Approximierte, randomisierte Suche nach nächsten Nachbarn Um das im letzten Abschnitt eingeführte Problem mit hochdimensionalen Daten zu umgehen, lässt sich die Suche nach dem nächsten Nachbarn randomisiert approximieren (ϵ-NN). Hierzu wählt man die abgeschwächte Problemformulierung, dass man mit Wahrscheinlichkeit $\delta > 0$ (möglichst maximal) zu einem gegebenen Anfragepunkt $q \in M$ und einem Toleranzradius $\epsilon > 0$ einen Nachbarn $x' \in X$ (bzw. entsprechend k Nachbarn $x_1, \ldots, x_k$) finden möchte, dessen Abstand zum echten nächsten Nachbarn $x \in X$ maximal um den Faktor $1 + \epsilon$ abweicht (Andoni, 2009, p. 20) (vgl. Abb. 2.1). Es soll also gelten, dass

$$\mathbb{P}\left[d(q, x') \leq (1 + \epsilon) \cdot d(q, x)\right] \geq \delta.$$

Reduktion auf ϵ-PLEB und randomisierte r-Nachbarsuche Eine äquivalente Formulierung ist das approximierte *point location in equal balls* (ϵ-PLEB) Problem. Hierbei soll für einen Radius $r > 0$, einen Approximationskonstante $\epsilon > 0$ und einen Anfragepunkt q ein Punkt x' mit $d(q, x') < (1 + \epsilon)r$ zurückgegeben werden, wenn der nächste Nachbar x von q den Abstand $d(q, x) < r$ besitzt. Hat der nächste Nachbar x einen Abstand von $d(q, x) \geq (1 + \epsilon)r$, soll nichts zurückgegeben werden. Hat der nächste Nachbar einen Abstand

$$r < d(q, x) < (1 + \epsilon)r$$

soll mit gleicher Wahrscheinlichkeit einer dieser Punkte oder gar kein Punkt zurückgegeben werden (vgl. Abb. 2.2). Es lässt sich zeigen, dass das ϵ-NN Problem auf dieses Problem durch eine binäre Suche mit logarithmischem Aufwand reduziert werden kann (Har-Peled, 2001; Indyk und Motwani, 1998). Eine ähnliche Formulierung, auf die das ϵ-NN in logarithmischer Zeit reduziert werden kann, ist die *randomisierte r-Nachbarsuche*. Hier soll für einen Anfragepunkt q, eine Approximationskonstante $c > 0$ und einen Radius $r > 0$ mit einer (möglichst maximalen) Wahrscheinlichkeit $\delta > 0$ ein Punkt x mit Abstand $d(x, q) < cr$ zurückgegeben werden (vgl. Abb. 2.3) (Andoni und Indyk, 2006; Har-Peled, 2001).

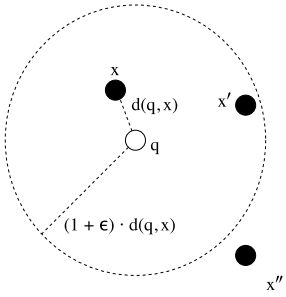


Abbildung 2.1: *ϵ-NN Suche*: Mit Wahrscheinlichkeit δ werden x oder x' zurückgeben. Die Wahrscheinlichkeit x'' zu erhalten beträgt höchstens $1 - \delta$.

Lokal-sensitive Hashfunktionen Eines der ersten und bekanntesten Verfahren zur effizienten Lösung dieses abgeschwächten Problems, auch für hochdimensionale Daten, ist *locality-sensitive hashing* (LSH) (Indyk und Motwani, 1998). Die Grundidee ist, eine Hashfunktion h zu konstruieren, die jedem Punkt y in M einen Hashwert $h(y)$ (z.B. eine ganze Zahl, oder einen binären Code) zuordnet und diesen Hashwert direkt auf eine Speicheradresse abbildet. An dieser Speicheradresse legt man nun die Liste aller Datenpunkte (bzw. die auf sie verweisenden Indizes) ab, die diesen Hashwert erhalten haben. Analog

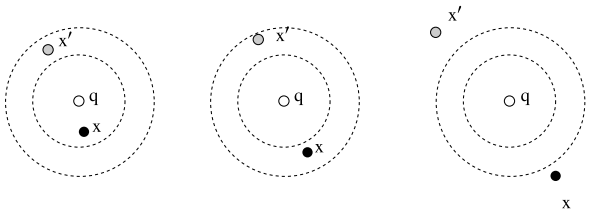


Abbildung 2.2: ϵ-*PLEB*: Links: x oder x' wird zurückgegeben. Rechts: kein Wert wird zurückgegeben. Mitte: eine der beiden vorigen Möglichkeiten wird zufällig gewählt.

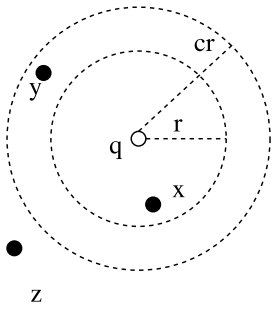


Abbildung 2.3: *Randomisierte r-Nachbarsuche*: Die Punkte x und y werden mit Wahrscheinlichkeit δ zurückgegeben werden - der Punkt z dagegen nur mit Wahrscheinlichkeit $1 - \delta$.

zu klassischen Hashtabellen (Knuth, 1998, p.513-558), bei denen ein beliebiger Datenwert unter einem Schlüssel (z.B. einem String) abgespeichert werden und in konstanter Zeit[1] assoziativ wiedergefunden werden soll, lassen sich nun unter Anfrage des Hashwerts alle darunter gespeicherten Datenpunkte zurückgeben. Bei klassischen assoziativen Containern ist man bemüht, h so zu konstruieren, dass Kollisionen vermieden werden. Das heißt, dass für verschiedene Suchschlüssel a und b mit hoher Wahrscheinlichkeit unterschiedliche Hashwerte $h(a)$ und $h(b)$ erzeugt werden. Damit soll verhindert werden, dass innerhalb der unter einem Hashwert gespeicherten Datenwerte gesucht werden muss. Für die metrische Suche ist es dagegen von Vorteil, Kollisionen bewusst zu erzwingen, wenn a und b einen geringen Abstand (bzw. eine hohe Ähnlichkeit) besitzen. In diesem Fall liefert das Auswerten des Hashwerts $h(q)$ einer Suchanfrage q

1 Sind ein Großteil der möglichen Hashwerte vergeben, wird die Suche approximativ logarithmisch. In einem festen Addressraum müssen die unter dem Hashwert abgelegten Listen durchsucht werden. In der offenen Addressierung muss z.B. durch Mehrfach-Hashing im Adressraum gesucht werden.

in konstanter Zeit eine Adresse, unter welcher überwiegend nahe Datenpunkte gespeichert sind. Präzisiert bedeutet dies, dass h so konstruiert sein muss, dass für einen gegebenen Abstand $r > 0$, eine Approximationskonstante $c > 0$, eine (möglichst hohe) positive Kollisionswahrscheinlichkeit $p_1 > 0$ und eine (möglichst niedrige) negative Kollisionswahrscheinlichkeit $p_2 > 0$ für zwei Punkte x, y aus M gelten soll, dass

$$\mathbb{P}\left[h(x) = h(y) | d(x, y) < r\right] > p_1$$

und

$$\mathbb{P}\left[h(x) = h(y) | d(x, y) > cr\right] < p_2$$

Eine solche Hashfunktion nennen Indyk und Motwani (r, c, p_1, p_2)-sensitiv.

Qualität der Ergebnisse und der Tradeoff zwischen Präzision und Recall Mit der Konstruktion einer Hashtabelle unter Nutzung einer solchen Hashfunktion erhält man ein approximatives Verfahren für die Suche nach nächsten Nachbarn, das unabhängig von der Dimension[2] des Datenraums mehr oder weniger gute Kandidaten für nahe Nachbarn liefert. Die Qualität des Verfahrens im Bezug auf die zurückgegebenen Ergebnisse lässt sich mit der klassischen Signalerkennungstheorie (Fawcett, 2006) analysieren. Nachbarn, die korrekterweise mit der Anfrage auf denselben Hashcode abgebildet werden, bezeichnet man als *true positives*. Unbenachbarte Datenpunkte mit dem selben Hashcode sind *false positives* bzw. der *Fehler erster Art*. Nachbarn, die irrtümlicherweise einen anderen Hashcode erhalten werden *false negatives* bzw. der *Fehler zweiter Art* genannt. Unbenachbarte Punkte, die korrekterweise auf einen anderen Hashcode abgebildet werden, heißen *true negatives*. Setzt man den Fehler erster Art zur Grundgesamtheit aller benachbarten Punkte, bzw. den Fehler zweiter Art zur Grundgesamtheit aller nicht-benachbarten Punkte in Relation, erhält man eine frequentistische Wahrscheinlichkeit für den Eintritt des jeweiligen Fehlers. In der Betrachtung von Hashfunktionen bietet sich aber auch die Bayessche Sicht an: Die Wahrscheinlichkeit dafür, dass der Fehler erster Art nicht eintritt, ist die *Präzision*

$$\mathbb{P}\left[d(x, y) < r | h(x) = h(y)\right]$$

der Hashfunktion. Die Wahrscheinlichkeit für den Nichteintritt des Fehlers zweiter Art ist der *Recall*

$$\mathbb{P}\left[h(x) = h(y) | d(x, y) < r\right]$$

2 Unter der Voraussetzung, dass eine gute Funktion h unabhängig von der Dimension gewählt werden kann.

Verteilt die Hashfunktion nur wenige Codes auf die Datenpunkte, steigt die Wahrscheinlichkeit, dass zwei Datenpunkte denselben Code erhalten. Dadurch werden Kollisionen wahrscheinlicher. Dies hat im Allgemeinen zur Folge, dass im Verhältnis mehr entfernte Punkte zusammen zurückgegeben werden (die Präzision sinkt), aber auch insgesamt mehr benachbarte Datenpunkte (der Recall steigt). Umgekehrt sorgt eine Verteilung von vielen Codes auf die Datenpunkte dafür, dass weniger Datenpunkte unter einem Code zusammengefasst werden und damit die Wahrscheinlichkeit für Kollisionen sinkt. Dies hat im Allgemeinen ein Sinken der Präzision und ein Steigen des Recalls zur Folge. Lassen sich stochastisch unabhängige Hashfunktionen

$$h_1, \ldots, h_l$$

aus einer Familie mit identischen Eigenschaften hinsichtlich Präzision und Recall erzeugen, lässt sich die Präzision (unter Inkaufnahme eines reduzierten Recalls) erhöhen, indem man für einen Punkt y statt des Hashwerts $h(y)$ den Code

$$\big(h_1(y), \ldots, h_l(y)\big)^T \in \mathbb{Z}^l$$

vergibt. Ist

$$p_{f1} = \mathbb{P}\left[d(x, y) < r | h(x) \neq h(y)\right]$$

die Wahrscheinlichkeit für den Eintritt des ersten Fehlers bei einer einzelnen Hashfunktion, reduziert sich diese auf $p_{f1}^l < p_{f1}$. Hat man umgekehrt stochastisch unabhängige Hashfunktionen

$$h_1, \ldots, h_k$$

aus einer entsprechenden Familie mit jeweils hoher Präzision, lassen sich die Datenpunkte redundant in entsprechend k verschiedenen Hashtabellen speichern. Bei der Suche nach den Nachbarn von q werden nun alle Einzelergebnisse aus den k Tabellen zurückgegeben. Dies erhöht den Recall, hat aber zwangsläufig auch eine Reduktion der Präzision zur Folge (vgl. Abb. 2.4). Dieses Prinzip wird in Stein (2007) zusammengefasst als:

> Code length controls precision. Code mulitiplicity controls recall.

LSH zum Lösen des approximierten Suchproblems Indyk und Motwani (1998) zeigten nun, dass mit der o.g. Methode für eine gegebene Familie von (r, c, p_1, p_2)-sensitiven Hashfunktion und dem Wert

$$\rho = -\frac{\log p_1}{\log p_1 / p_2}$$

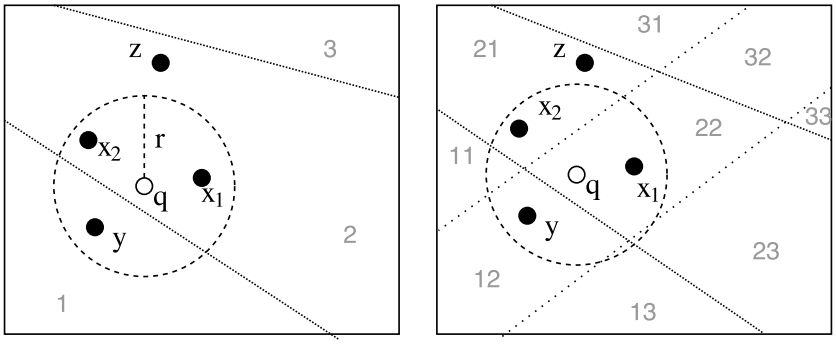


Abbildung 2.4: *Precision-Recall-Tradeoff*: Links: Die Hashfunktion vergibt für x_1 und x_2 korrekterweise dieselben Codes wie für q. Dem Punkt z wird irrtürmlicherweise ebenfalls derselbe Code zugewiesen - dies ist ein Präzisions-Fehler. Der Punkt y wird irrtürmlicherweise einem anderen Code zugeordnet - dies ist ein Recall-Fehler. Rechts: wenn man eine weitere Hashfunktion hinzunimmt, vergibt diese nun statt einem einzelnen Code einen konkatenierten Code und erhöht die Länge der erzeugten Codes auf 2. Damit wird z nun korrekterweise von q getrennt. Hierbei geht aber auch unkorrekterweise die Beziehung zu x_2 verloren – die Präzision steigt – der Recall sinkt.

das ϵ-PLEB Problem in der Zeit $\mathcal{O}(n^\rho)$ gelöst werden kann. Dazu konstruieren sie $k = n^\rho$ verschiedene konkatenierte Hashfunktionen

$$h_i = \left(h_{1,i}, \ldots, h_{l,i}\right)^T, \; i = 1 \ldots l$$

aus $l = \log_{p_1/p_2} 2n$ einzelnen Funktionen der Familie. Die für die Laufzeit entscheidende Größe ρ und die Approximationskonstante c sind dabei nicht beliebig frei voneinander wählbar. Für die klassische euklidische Metrik gilt z.B. grundsätzlich, dass $\rho \leq \frac{0.462}{c^2}$ (Motwani, Naor und Panigrahy, 2005). Für andere Metriken existieren ähnliche Schranken, aber auch Algorithmen, die das Problem innerhalb dieser Schranken nahezu optimal lösen (Andoni, 2009; Andoni und Indyk, 2006). Aus dieser erzwungenen Schranke des Verhältnisses von ρ und c folgt $p_1 \cdot p_2 < K$ für eine feste Konstante $K < 1$. Es existiert somit für den allgemeinen Fall eine Art Unschärferelation, die den Tradeoff zwischen Präzision und Recall, unabhängig von der Wahl der Hashfamilie, festlegt und damit eine fundamentale Grenze für die Möglichkeit zur approximierten Suche darstellt (vgl. Abb. 2.5).

Datenabhängige und Datenunabhängige Hashfunktionen Die Kerneigenschaft der oben eingeführten Hashfunktionen hängt nur vom betrachteten metrischen Raum ab und ignoriert die zu speichernden Daten komplett. In der Praxis hat sich gezeigt, dass Hashfunktionen, die die Datenverteilung in ihrer

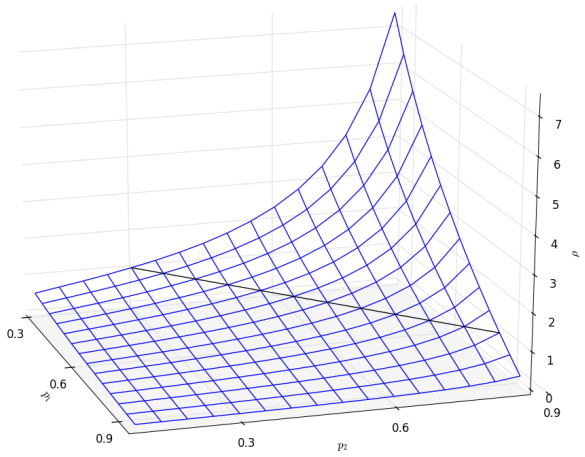


Abbildung 2.5: *Das Verhalten von ρ in Abhängigkeit von p_1 und p_2.*: Die Wahrscheinlichkeiten p_1 und p_2 legen den führenden Exponenten für die Suchzeit ρ fest. Es ist deutlich zu sehen, dass ein hoher Recall bei einer geringen Präzision sehr schnell zu Overhead gegenüber der linearen Suche führt. Umgekehrt garantiert die Inkaufnahme von moderater Präzision bzw. moderatem Recall eine feste sublineare Suchzeit.

Konstruktion miteinbeziehen, für viele Anwendungen deutlich bessere Ergebnisse erzielen (wie z.B. die Verfahren von Salakhutdinov und Hinton (2009) und Weiss, Torralba und Fergus (2008)). Während bei der ersten Methode der gesamte metrische Raum in diskrete Gebiete mit demselben Hashwert unterteilt wird und im Anschluss Datenpunkte in die entsprechenden Gebiete fallen, wird bei der zweiten Methode die Hashfunktion möglichst passgenau für die Datenpunkte konstruiert, so dass die Gebiete mit selbem Hashwert von der Datenverteilung abhängen (vgl. Abb. 2.6). Andoni und Razenshteyn (2015b) führen daher einen *datenabhängigen* Begriff einer lokal-sensitiven Hashfunktion ein. Sie definieren eine Hashfamilie (c, r, p_1, p_2) sensitiv *in Bezug auf einen Datensatz* $X \subset M$, wenn für jede Suchanfrage $q \in M$ und jeden Datenpunkt $x \in X$ erfüllt ist, dass

$$\mathbb{P}\left[h(x) = h(q) | d(x, q) < r\right] > p_1$$

bzw.

$$\mathbb{P}\left[h(x) = h(q) | d(x, q) > cr\right] < p_2$$

Der wichtige Unterschied zu der Definition von Indyk und Motwani (1998) ist, dass nur q aus dem ganzen metrischen Raum gewählt werden muss, wohingegen x nur aus den Daten gewählt werden muss. Diese schwächere Anforderung hat zur Folge, dass sich auch die im letzten Abschnitt eingeführten Untergrenzen

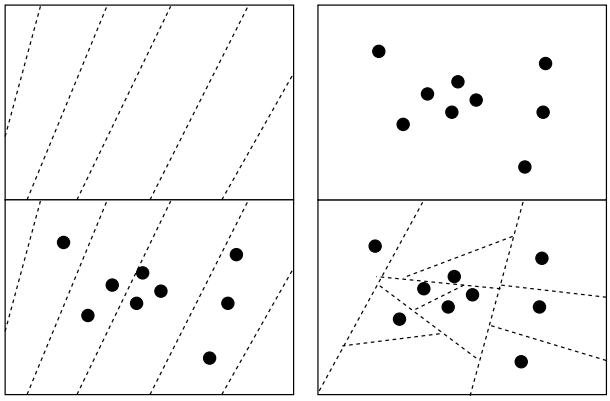


Abbildung 2.6: *Raumpartitionierung durch Hashfunktionen*: Links: Die Hashfunktion unterteilt den metrischen Raum unabhängig von Daten in Gebiete. Rechts: Die Hashfunktion wird auf Basis der zu speichernden Daten erzeugt.

verändern. So ist es – selbst ohne die konkrete Verteilung der Daten zu kennen – möglich, einen Algorithmus anzugeben, der $\rho = \frac{1}{2c^2-1}$ erreicht und damit unterhalb der theoretisch möglichen Grenze des vorigen Abschnitts liegt (Andoni und Razenshteyn, 2015a). Andoni und Razenshteyn (2015b) zeigt jedoch auch, dass dies – zumindest ohne die Datenverteilung einzuschränken – die theoretisch mögliche Untergrenze für die datenabhängige Konstruktion ist. Damit existiert auch hier eine vergleichbare Unschärferelation für die approximative Suche.

2.2 Konstruktion von Hashfunktionen für die approximative Suche

Nach der theoretischen Einführung stellt sich nun die natürliche Frage, wie solche lokal-sensitiven Hashfunktionen konstruiert werden können. Im folgenden Abschnitt soll dies genauer erörtert werden.

Anatomie einer lokal-sensitiven Hashfunktion Eine Hashfunktion ist eine Abbildung

$$h\colon M \to \mathbb{D}$$

eines metrischen Raums (meist der $\mathbb{R}^D$ mit einer üblichen Metrik) in einen diskreten Raum $\mathbb{D}$. Üblicherweise ist $\mathbb{D}$ der l-dimensionale Raum $\mathbb{Z}^l$ bzw. je nach Codierung der Hamming-Hyperwürfel $\{0,1\}^l$. Im Allgemeinen lässt sich diese Funktion aus zwei Komponenten zusammensetzen: einer *Raumeinbettung*

$$E\colon M \to M'$$

und einer *Quantisierung*

$$Q\colon M' \to \mathbb{D}$$

Die entstehende Hashfunktion ergibt sich also als Komposition dieser zwei Funktionen

$$h(x) = Q\big(E(x)\big)$$

Die Wahl der Einbettung Wie im vorigen Kapitel im Kontext der intrinsischen Dimension eingeführt, liegt den meisten hochdimensionalen Datensätzen die Mannigfaltigkeits-Hypothese zugrunde: die Datensätze sind im hochdimensionalen Raum nahe einer deutlich niedrigdimensionaleren Untermannigfaltigkeit angeordnet. Kennt man nun eine Kartierung dieser Untermannigfaltigkeit, die die Metrik bzw. Topologie des Datensatzes erhält, lässt sich das metrische Problem auf diesen niedrigdimensionalen Datensatz vereinfachen. Dieses Prinzip wird von vielen Verfahren in der Analyse hochdimensionaler Daten ausgenutzt (Burges, 2009). Im Einbettungsschritt der Hashfunktion wird folg-

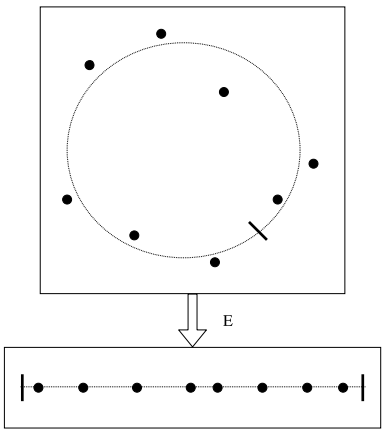


Abbildung 2.7: *Mannigfaltigkeits-Hypothese*: Die Daten sind im zweidimensionalen Raum in der Nähe eines Kreises angeordnet. Durchschneiden des Kreises und Entfernen des Rauschens projiziert die Daten auf die reelle Linie. Diese stereografische Projektion erhält die Metrik der Punkte zueinander relativ gut. Die Wahl der Karte sorgt aber für maximal starke Verzerrungen an der Stelle, an der der Kreis zerschnitten wird.

lich versucht, eine Einbettung E zu finden, die das Verhältnis der Abstände der hochdimensionalen Datendarstellung in einem niedrigdimensionalen Raum erhält. Ist d die Metrik im Ursprungsraum und d' die Metrik im Einbettungsraum, wird also versucht, für ein möglichst kleines $\epsilon > 0$ die Wahrscheinlichkeit

$$\mathbb{P}\left[|d(x,y) - d'(E(x),E(y))| < \epsilon\right]$$

minimal werden zu lassen. Im Kontext einer r-Umgebungssuche kann auch versucht werden, benachbarte Punkte in M auf benachbarte Punkte in M' bzw. unbenachbarte Punkte in M auf unbenachbarte Punkte in M' abzubilden. Ist $r > 0$ ein Schwellwert für die Ähnlichkeit, ab der zwei Punkte als benachbart angesehen werden, wird versucht die Wahrscheinlichkeiten

$$\mathbb{P}\left[d'(E(x), E(y)) < r \mid d(x, y) > r\right]$$

und

$$\mathbb{P}\left[d'(E(x), E(y)) > r \mid d(x, y) < r\right]$$

zu minimieren. Die erste Minimierung ist analog der Forderung für gute Präzision lokal-sensitiven Hashfunktion. Die zweite entspricht der Forderung nach gutem Recall einer lokal-sensitiven Hashfunktion. In Stein (2007) wird für die r-Umgebungssuche ein Fehlermaß für Einbettungen vorgestellt, das in diesem Kontext deutlich präziser die Qualität einer Einbettung evaluiert, als es die Forderung nach globaler metrischer Konsistenz tut. Dazu wird zunächst die Menge

$$R_r = \left\{\{x, y\} \mid d(x, y) < r\right\}$$

aller Nachbarn im Ursprungsraum bzw. die Menge

$$R'_r = \left\{\{x, y\} \mid d'(E(x), E(y)) < r\right\}$$

aller Nachbarn im Einbettungsraum bestimmt. Für die Einbettung E wird nun der *Präzisions-Stress* definiert als

$$e_{P,r} = \frac{\sum_{x,y \in R'_r} \left(d(x, y) - d'\big(E(x), E(y)\big)\right)^2}{\sum_{x,y \in R'_r} \big(d(x, y)\big)^2}$$

bzw. der *Recall-Stress* als

$$e_{R,r} = \frac{\sum_{x,y \in R_r} \left(d(x, y) - d'\big(E(x), E(y)\big)\right)^2}{\sum_{x,y \in R_r} \big(d'(x, y)\big)^2}$$

Diese beiden Statistiken messen, wieviele Datenpunkte nun nach der Einbettung inkorrekterweise benachbart bzw. nicht mehr benachbart sind. Stein konnte in seinem Artikel zeigen, dass lineare Einbettungen, die nach diesem Maß optimiert sind, im Kontext der approximativen r-Umgebungssuche bessere Ergebnisse erzielen, als jene, die versuchen, die Ähnlichkeiten global zu erhalten. Interessanterweise existieren auch für Datensätze, die nicht auf Un-

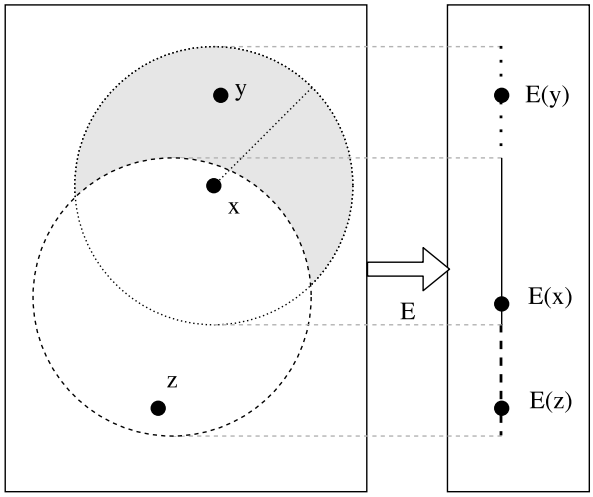


Abbildung 2.8: *Einbettungsstress*: Bei der Einbettung E von $\mathbb{R}^2$ nach $\mathbb{R}$ werden Nachbarschaften von x nicht korrekt abgebildet. Ein ursprünglich benachbarter Punkt y ist nun nicht mehr benachbart. Sein nun erhöhter Abstand zu x in $\mathbb{R}$ erhöht den Recall-Stress. Ein ursprünglich nicht benachbarter Punkt z wird dagegen in den r'-Ball um $E(x)$ abgebildet. Sein höherer Abstand in $\mathbb{R}^2$ trägt positiv zum Präzisions-Stress bei.

termannigfaltigkeiten eingeschränkt sein müssen, gute Einbettungen in niedrige Dimensionen. So garantiert z.B. das Johnson-Lindenstrauss-Lemma (Johnson, Lindenstrauss und Schechtman, 1986), dass für eine beliebige Punktmenge $X \subset \mathbb{R}^D$ des euklidischen Raums eine lineare Einbettung E in einen n-dimensionalen Unterraum $\mathbb{R}^n$ möglich ist, für die

$$n = 8\frac{\log |X|}{\epsilon^2}$$

und

$$(1-\epsilon)d(x,y) \leq d'\big(E(x), E(y)\big) \leq (1+\epsilon)d(x,y)$$

gilt. Dies ist die Grundlage für Einbettungsverfahren, die auf Zufallsprojektionen basieren, ohne die Verteilung der eigentlichen Daten zu berücksichtigen. Da es sehr viele Verfahren gibt, um hochdimensionale Daten metrisch konsistent in niedrigdimensionale Räume einzubetten (vgl. Burges (2009) für eine Übersicht), unterscheiden sich die in den letzten Jahren entwickelten Hashverfahren an dieser Stelle sehr stark. In Jingdong Wang, H. T. Shen, Song u. a. (2014) wird grob zwischen Verfahren unterschieden, die *lineare* bzw. *nichtlineare* Einbettungen nutzen. *Lineare* Verfahren projizieren die Daten mithilfe einer Matrix in einen Unterraum. Die Matrix kann dabei zufällig aus einer

Verteilung mit günstigen Eigenschaften[3] gewählt werden, wie z.B. in Charikar (2002) oder Datar u. a. (2004), oder hinsichtlich einer Fehlerfunktion optimiert werden, die die Anforderungen an metrische Konsistenz enthält, wie z.B. in L. Zhang u. a. (2013) oder in Strecha u. a. (2012). *Nichtlineare* Verfahren nutzen komplexere Einbettungen, basierend auf tiefen neuronalen Netzen (Salakhutdinov und Hinton, 2009; Zhao u. a., 2015), Zerlegungen in Eigenfunktionen (Weiss, Torralba und Fergus, 2008) oder nichtlineare Kernelfunktionen (Joly und Buisson, 2011; Kulis und Grauman, 2009; W. Liu, Jun Wang, Ji u. a., 2012). Gegenüber den linearen Verfahren haben diese Modelle meist eine höhere Beschreibungskomplexität und ermöglichen daher präzisere Einbettungen.

Die Wahl der Quantisierung Im Quantisierungsschritt müssen die kontinuierlichen Werte diskretisiert werden, um eine Unterteilung des Raums in die verschiedenen Hashgebiete zu ermöglichen. Hier gilt, dass benachbarte Werte aus M' mit hoher Wahrscheinlichkeit auf denselben Wert abgebildet werden sollen und nicht benachbarte mit hoher Wahrscheinlichkeit auf verschiedene Werte. Die Anforderungen an eine Quantisierung Q müssen also dieselben sein, wie an eine lokal-sensitive Hashfunktion selbst. Dies lässt sich durch *Minimierung des Einbettungsabstands*, durch *Minimierung des Quantisierungsfehlers*, durch *Quantisierung auf eine Stellvertretermenge* oder *implizit* erreichen. Bei der Minimierung des Einbettungsabstands wird die Quantisierung selbst als eine Einbettung in den diskreten Raum, versehen mit einer Metrik, aufgefasst[4]. Ist $\mathbb{D} = \mathbb{Z}^l$ wird dabei oft die euklidische Metrik genutzt. Ist $\mathbb{D} = \{0,1\}^k$, wird in den meisten Fällen der Hamming-Abstand verwendet. Die Anforderungen an eine gute Quantisierung sind nun analog zu denen für eine gute Einbettung. Da es sich hierbei aber um eine diskrete Einbettung handelt, können nicht die üblichen Methoden für das Finden von kontinuierlichen Einbettungen genutzt werden. Einige aktuelle Verfahren, wie z.B. Zhao u. a. (2015), umgehen dieses Problem, indem sie die diskreten Sprünge zwischen den quantisierten Zielwerten mit einer Sigmoide, wie z.B.

$$\sigma(x) = \frac{1}{1 + e^{-a \cdot x}},\ a > 1$$

3 Damit ist eine Verteilung gemeint, für man zeigen kann, dass aus ihr gewählte Matrizen Zufallsprojektionen erzeugen, die eine gute metrische Konsistenz gewährleisten.

4 z.B. von $\mathbb{R}^D$ nach $\mathbb{Z}^D$ durch komponentenweises Runden zusammen mit der auf die Teilmenge induzierten Metrik

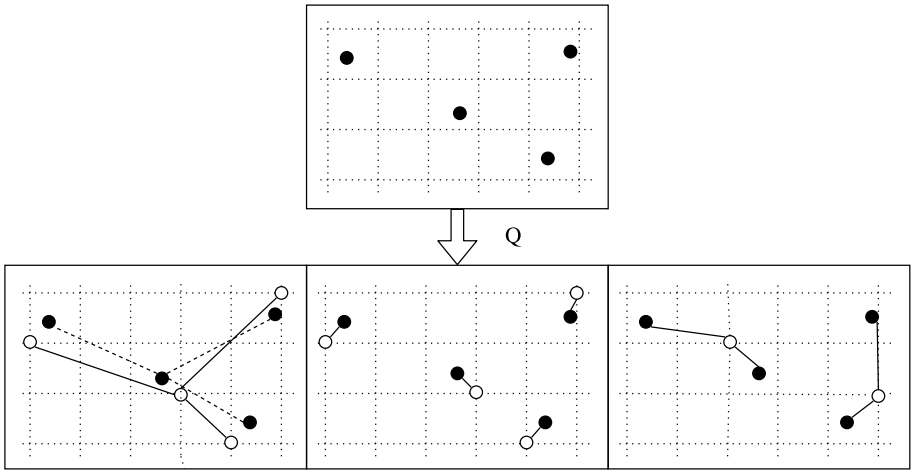


Abbildung 2.9: *Links*: Beim Minimieren des Einbettungsabstands müssen alle paarweisen Abstände zwischen den Datenpunkten und ihren entsprechenden Einbettungen angeglichen werden. *Mitte*: Minimieren des Quantisierungsfehlers optimiert dieselbe Größe, benötigt aber nur die Abstände der Punkte zur ihren quantisierten Darstellungen. *Rechts*: Bei der Quantisierung durch Stellvertreter, werden diese (z.B. durch Clustern der Daten) vorher gewählt. Danach werden Punkte auf ihren nächsten Stellvertreter abgebildet.

approximieren und dann kontinuierliche Einbettungen anwenden. Andere Verfahren versuchen direkt das diskrete Optimierungsproblem

$$\min \sum_{x,y \in X} d'(x,y) \cdot d''\big(Q(x), Q(y)\big)$$

zu lösen. Dieses Problem ist durch Hinzunahme einfacher Zusatzforderungen (z.B. eine gleiche Wahrscheinlichkeit dafür, dass Hashbits den Wert 0 oder 1 annehmen) NP-schwer (Weiss, Torralba und Fergus, 2008). Verfahren, die diesen Ansatz verfolgen, wie z.B. He, W. Liu und S.-F. Chang (2010), Kulis und Darrell (2009) und Strecha u. a. (2012), müssen also Vereinfachungen des Problems (z.B. durch Abschwächungen der Optimierungsrandbedingungen) vornehmen, um es approximativ zu lösen. Es kann gezeigt werden, dass durch Minimieren des Quantisierungsfehlers

$$e_Q(x) = d'\big(x, Q(x)\big)$$

auch der o.g. diskrete Einbettungsabstand minimiert wird (Jingdong Wang, H. T. Shen und T. Zhang, 2014). Bei der Quantisierung auf eine Stellvertretermenge wird zum Datensatz eine Stellvertretermenge $Z \subset \mathbb{Z}^{D'}$ konstruiert und danach jeder zu quantisierende Punkt dem Index seines nächstgelegenen Stellvertreterpunkts zugeordnet. Da sich dieses Problem leichter modellieren und

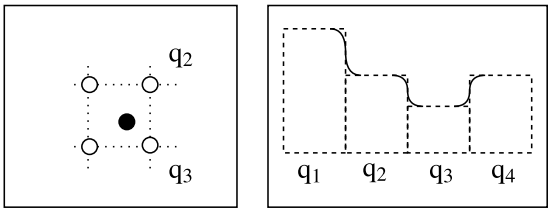


Abbildung 2.10: *Approximation der diskreten Optimierung mit einer Sigmoide*: Das ursprüngliche diskrete Optimierungsproblem, den Punkt zu quantisieren, lässt sich durch Annäherung der Abstände mit einer Sigmoide approximieren. Dadurch wird das Optimierungsproblem beliebig oft differenzierbar und klassischen kontinuierlichen Optimierungsverfahren zugänglich.

optimieren lässt als die Optimierung der paarweisen Abstände von Punkten, wird es z.B. von Jegou, Douze und Schmid (2011) und T. Zhang, Jingdong Wang und Com (2014) als Grundlage zum Finden einer guten Quantisierung verwendet. Bei Verfahren, die eine Einbettung in den Vordergrund der Optimierung stellen, wird meist eine implizite Quantisierung vorgenommen. Damit ist gemeint, die Datenpunkte nahe ihrer quantisierten Repräsentanten einzubetten und neue Datenpunkte uninformiert mit simplen Methoden, z.B. durch Schwellwerte, zu quantisieren. Für binäre Vektoren nimmt man dafür komponentenweise die Signumsfunktion

$$\operatorname{sign}(x) = \begin{cases} 0 & x < 0 \\ 1 & \mathit{sonst} \end{cases}$$

Soll eine Quantisierung auf Werte in $\mathbb{Z}^k$ erfolgen, nutzt man dagegen komponentenweise die Gaußklammer

$$\lfloor x \rfloor = \arg\min_{z \in \mathbb{Z}} |z - x|$$

Konstruktion durch Wahl von Parametern Die meisten in der Praxis berechenbaren Familien $\mathcal{F}$ von Funktionen

$$f \colon X \to Y$$

lassen sich *parametrisiert* beschreiben. Damit ist gemeint, dass eine Parametermenge Φ und eine surjektive Auswahlfunktion

$$\chi \colon \Phi \to \mathcal{F}$$

existiert, so dass jedes $f \in \mathcal{F}$ durch einen Parameter $\phi \in \Phi$ als $f = \chi(\phi)$ dargestellt werden kann. Ein Beispiel sind lineare Funktionen

$$f : \mathbb{R}^n \to \mathbb{R}^k$$

Jede dieser Funktionen lässt sich durch eine Matrix $A \in \mathbb{R}^{n \times k}$ darstellen, als

$$f(x) = Ax$$

Damit ist f durch den Parameter $\phi = A$ aus der Parametermenge $\Phi = \mathbb{R}^{n \times k}$ und die Auswahlfunktion

$$\chi \colon A \mapsto \big(x \mapsto Ax\big)$$

bestimmt. Die Familie $\mathcal{F}$, die Parametermenge Φ und die Auswahlfunktion χ bilden ein *Modell* $(\mathcal{F}, \Phi, \chi)$ für Funktionen $f \in \mathcal{F}$. Hat man nun ein Modell $(\mathcal{H}, \Phi, \chi)$ für Hashfunktionen gegeben, impliziert dies Modelle $(\mathcal{E}, \Phi_E, \chi_E)$ und $(\mathcal{Q}, \Phi_Q, \chi_Q)$ für die Einbettungen und Quantisierungen. Die Suche nach guten Hashfunktionen, bzw. guten zugrundeliegenden Einbettungen bzw. Quantisierungen, lässt sich also als Suche nach den richtigen Parametern innerhalb eines gewählten Modells beschreiben.

Parameterwahl im datenunabhängigen Fall Bei datenunabhängigen Hashverfahren überwiegt die randomisierte Wahl von ϕ aus einer Verteilung, die eine gute Qualität der durch ϕ erzeugten Hashfunktionen garantiert. In Datar u. a. (2004) wurde z.B. gezeigt, dass sich durch zufälliges Samplen eines Vektors a aus einer multivariaten Normalverteilung zusammen mit einer gleichverteilten Zufallszahl b aus dem Interval $[0, r]$ die (im Bezug auf ρ) optimale Hashfunktion

$$h(x) = \left\lfloor \frac{x \cdot a + b}{r} \right\rfloor$$

erzeugen lässt. Die quantisierten Werte in $\mathbb{Z}$ lassen sich entweder direkt als binäre Codes nutzen, oder bei Nutzung mehrerer solcher Funktionen zu konkatenierten Codes erweitern. Für die o.g. Jaccard-Distanz bzw. die Kosinus-Ähnlichkeit wird eine entsprechende Variante in Charikar (2002) gegeben. Andere Verfahren, wie Andoni und Indyk (2006), versuchen die Hashfunktion durch die Wahl von zufälligen Gittern zu konstruieren, die gemeinsam den Raum überdecken.

Parameterwahl im datenabhängigen Fall Im datenabhängigen Fall bietet sich die Formulierung einer *Fehlerfunktion*

$$L \colon \Phi \to \mathbb{R}$$

an. Ist eine Hashfunktion $h = \chi(\phi)$ (bzw. Einbettung oder Quantisierung) mit Parameter ϕ ungeeignet, so soll $L(\phi)$ groß sein. Ist sie dagegen geeignet, soll $L(\phi)$ klein werden. Viele Fehlerfunktionen, die sich auf eine möglichst fehlerfreie Einbettung bzw. eine Quantisierung durch Minimierung des Einbettungsabstands konzentrieren, haben (vereinfacht) die Form

$$L(\phi) = \sum_{x,y \in X} \Big(d(x,y) - d'\big(\chi(\phi)(x), \chi(\phi)(y)\big) \Big)^2$$

Wird dagegen versucht, eine Quantisierung der Datenpunkte auf diskrete Stellvertreter zu optimieren, stellt sie sich (vereinfacht) als

$$L(\phi) = \sum_{x \in X} d\big(x, \chi(\phi)(x)\big)$$

dar. Verfahren, die dagegen bemüht sind durch die Hashcodes eine niedrigdimensionale Einbettung zu finden, aus der sich der hochdimensionale Datensatz rekonstruieren lässt[5], liefern zusätzlich zu einer Hashfunktion $h_\phi = \chi(\phi)$ eine Quasi-Umkehrfunktion

$$g_\phi \colon \mathbb{D} \to M$$

für die

$$g_\phi \circ h_\phi \sim id_M$$

gelten soll. Für den Datensatz gilt es also, die Funktion

$$L(\phi) = \sum_{x \in X} d\Big(x, g_\phi\big(h_\phi(x)\big)\Big)$$

zu minimieren. Dieses Prinzip der Reduktion auf eine diskrete, niedrigdimensionale Datenmenge durch einen *information-bottleneck* ist vor allem durch Salakhutdinov und Hinton (2009) populär geworden.

Die Wahl der Fehlerfunktion L ist nun entscheidend dafür, welche Anforderungen die durch Parameteroptimierung gefundene Funktion erfüllen soll. Neben den bereits erwähnten Anforderungen an die metrische Konsistenz der Einbettung bzw. den geringen Fehlers der Quantisierung fügen viele Verfahren in der Praxis zusätzliche Anforderungen hinzu: Um die Hashgebiete in dichten Regionen des Raums kleiner und damit präziser werden zu lassen, fordern einige Autoren, wie z.B. Paulevé, Hervé Jégou und Amsaleg (2010), dass entstehende Gebiete möglichst gleich viele Datenpunkte beinhalten. Andere, wie Weiss, Torralba und Fergus (2008), schlagen zudem vor, dass die einzelnen

5 Das Prinzip ist in den letzten Jahren unter dem Stichwort *deep learning* populär geworden

Hashfunktionen aus der Familie möglichst unabhängig voneinander sind, um die Datenmenge durch mehrere Hashgitter mit minimaler Redundanz abzudecken. Unabhängig heißt hierbei, dass für zwei Hashfunktionen h_1, h_2 aus einer Familie gelten soll, dass

$$\mathbb{P}\left[h_i(x) = b | h_j(x) = b\right] = \mathbb{P}\left[h_i(x) = b\right]$$

Unüberwachte und Überwachte Optimierung Um einen Minimierer für L zu finden, gibt es *überwachte* und *unüberwachte* Verfahren (Jun Wang, W. Liu u. a., 2016).[6] Im *überwachten* Fall werden Kandidaten für die Hashfunktion generiert, ihr Fehler auf dem Datensatz evaluiert und anschließend die Parameter so korrigiert, dass dabei der Fehler verringert wird. Dieses Verfahren wird solange wiederholt, bis sich der Fehler nicht weiter minimieren lässt. Hat man eine differenzierbare Version der Fehlerfunktion, kann dies z.B. über einen Gradientenabstieg im Parameterraum erfolgen. Bei der *unüberwachten* Optimierung existiert ein Modell für die Hashfunktion, für das sich ein Minimierer der Fehlerfunktion direkt aus dem Datensatz errechnen lässt. Für lineare Modelle lässt sich ein Minimierer z.B. meist durch direktes Lösen einer linearen Gleichung bestimmen. Der Vorteil des ersten Ansatzes liegt in seiner Flexibilität, da er auch für komplexe Modelle funktioniert und damit in der Praxis sehr präzise Ergebnisse zu ermöglicht. Der Nachteil ist aber die u.U. sehr hohe Anzahl an Iterationen, die für eine gute Optimierung notwendig sind, und dass häufig nur ein lokales und kein globales Minimum für das gegebene Modell erreicht werden kann. Der Vorteil des zweiten Ansatzes ist dagegen, dass er zum gegebenen Modell beweisbar einen globalen Minimierer liefern kann und eine feste Laufzeit besitzt. Der Nachteil ist die Einschränkung auf einfachere, gut verstandene Modelle, und damit eine geringere Flexibilität.

6 Die Begriffe *überwacht* bzw. *unüberwacht* dürfen nicht mit entsprechenden Begriffen aus dem Kontext des maschinellen Lernens verwechselt werden.

2.3 Existierende Verfahren

Bestehende Verfahren lassen sich nun hinsichtlich o.g. Ordnungsaxen in verschiedene Gruppen einteilen. Für eine vollständige und umfassende Diskussion der Verfahren seit Indyk und Motwani (1998) wird hierbei auf die umfangreichen Survey-Artikel Jingdong Wang, H. T. Shen, Song u. a. (2014), Jingdong Wang, H. T. Shen und T. Zhang (2014) und Jun Wang, W. Liu u. a. (2016) verwiesen.

Einordnung nach Einbettung

Lineare Einbettungen	(Charikar, 2002; Datar u. a., 2004; Gong u. a., 2013; Herve Jégou u. a., 2008; Strecha u. a., 2012; Jianfeng Wang u. a., 2013; D. Zhang u. a., 2010; L. Zhang u. a., 2013)
Nichtlineare Einbettungen	(He, S. F. Chang u. a., 2011; He, W. Liu und S.-F. Chang, 2010; Heo u. a., 2012; Jegou, Douze und Schmid, 2011; Joly und Buisson, 2011; Kulis und Grauman, 2009; W. Liu, Jun Wang, Ji u. a., 2012; W. Liu, Jun Wang, Kumar u. a., 2011; Norouzi, Fleet und Salakhutdinov, 2012; Salakhutdinov und Hinton, 2009; F. Shen u. a., 2013; Weiss, Torralba und Fergus, 2008; Wu u. a., 2013; Yang u. a., 2014; T. Zhang, Jingdong Wang und Com, 2014; Zhao u. a., 2015)

Einordnung nach Quantisierung

Implizit	(Charikar, 2002; Datar u. a., 2004; He, W. Liu und S.-F. Chang, 2010; Heo u. a., 2012; Joly und Buisson, 2011; Kulis und Grauman, 2009; W. Liu, Jun Wang, Ji u. a., 2012; Norouzi, Fleet und Salakhutdinov, 2012; Salakhutdinov und Hinton, 2009; Strecha u. a., 2012; Jianfeng Wang u. a., 2013; Weiss, Torralba und Fergus, 2008; Wu u. a., 2013; Zhao u. a., 2015)
Minimierung des Quantisierungsfehlers	(Gong u. a., 2013; He, S. F. Chang u. a., 2011; Herve Jégou u. a., 2008; W. Liu, Jun Wang, Kumar u. a., 2011; F. Shen u. a., 2013; Yang u. a., 2014; D. Zhang u. a., 2010; L. Zhang u. a., 2013)
Abbildung auf diskrete Stellvertreter	(Jegou, Douze und Schmid, 2011; T. Zhang, Jingdong Wang und Com, 2014)

Einordnung nach Datenabhängigkeit

Datenabhängig	(Gong u. a., 2013; He, S. F. Chang u. a., 2011; He, W. Liu und S.-F. Chang, 2010; Heo u. a., 2012; Jegou, Douze und Schmid, 2011; Joly und Buisson, 2011; Kulis und Grauman, 2009; W. Liu, Jun Wang, Ji u. a., 2012; Norouzi, Fleet und Salakhutdinov, 2012; Salakhutdinov und Hinton, 2009; F. Shen u. a., 2013; Strecha u. a., 2012; Jianfeng Wang u. a., 2013; Weiss, Torralba und Fergus, 2008; Wu u. a., 2013; Yang u. a., 2014; D. Zhang u. a., 2010; L. Zhang u. a., 2013; T. Zhang, Jingdong Wang und Com, 2014; Zhao u. a., 2015)
Datenunabhängig	(Charikar, 2002; Datar u. a., 2004; Herve Jégou u. a., 2008; W. Liu, Jun Wang, Kumar u. a., 2011)

Einordnung nach Überwachtheit

Überwacht	(Heo u. a., 2012; W. Liu, Jun Wang, Ji u. a., 2012; Norouzi, Fleet und Salakhutdinov, 2012; Jianfeng Wang u. a., 2013; Wu u. a., 2013; Yang u. a., 2014; D. Zhang u. a., 2010; L. Zhang u. a., 2013)
Unüberwacht	(Charikar, 2002; Datar u. a., 2004; Gong u. a., 2013; He, S. F. Chang u. a., 2011; He, W. Liu und S.-F. Chang, 2010; Jegou, Douze und Schmid, 2011; Herve Jégou u. a., 2008; Joly und Buisson, 2011; Kulis und Grauman, 2009; W. Liu, Jun Wang, Kumar u. a., 2011; Salakhutdinov und Hinton, 2009; F. Shen u. a., 2013; Strecha u. a., 2012; Weiss, Torralba und Fergus, 2008; T. Zhang, Jingdong Wang und Com, 2014; Zhao u. a., 2015)

2.4 Retrieval mit Hashfunktionen

Basierend auf den Konzepten der approximativen Suche und lokal-sensitiven Hashfunktionen, haben sich verschiedene Strategien entwickelt, um die Suche praktisch durchzuführen.

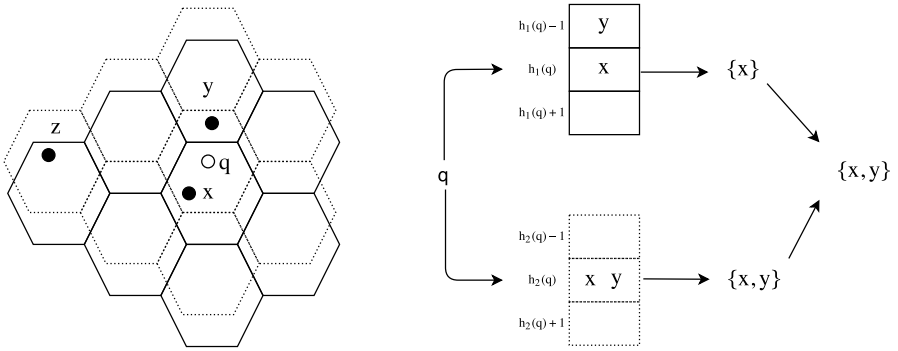


Abbildung 2.11: *Multi-Table-Hashing*: Durch redundantes Speichern der Punkte in zwei Tabellen können mit einer Auswertung der Hashfunktionen mehr Nachbarn gefunden werden, als es durch eine einzige Tabelle ginge.

Table-Lookup und Hash-Code-Ranking Den Taxonomien von Jun Wang, W. Liu u. a. (2016), Jingdong Wang, H. T. Shen, Song u. a. (2014) und Jingdong Wang, H. T. Shen und T. Zhang (2014) folgend, lassen sich diese Strategien grob in zwei Gruppen unterteilen. *Table-Lookup*-Verfahren versuchen weiterhin die vollständige Suche durch das heuristische Auswerten der Hashtabelle(n) (basierend auf dem Hashcode der Suchanfrage) zu vermeiden. Andere Verfahren nutzen ein *Hash-Code-Ranking*, bei dem die vollständige NN-Suche direkt im Hamming-Raum der durch die Hashfunktion erzeugten Binärcodes durchgeführt wird. Ist der Code-Raum niedrigdimensional, die Anzahl der Datenpunkte moderat groß und die Abbildung der Ursprungsmetrik auf die Hammingmetrik des Code-Raums von geringem Fehler, ist die Suche dort effizient und unterscheidet sich im Bezug auf die Suchzeit nicht signifikant von den anderen Verfahren (Jingdong Wang, H. T. Shen, Song u. a., 2014). Die Table-Lookup-Verfahren lassen sich weiterhin unterteilen in *Multi-Table*- und *Multi-Probe*-Verfahren. Die ersten versuchen, wie bei LSH, über das Anlegen und Auswerten von mehreren, unabhängigen Hashtabellen die Recall-Schwäche einer einzigen Funktion auszugleichen. Die zweiten Verfahren setzen, wie die o.g. Ranking-Verfahren, eine Hashfunktion voraus, deren Fehler bei der Einbet-

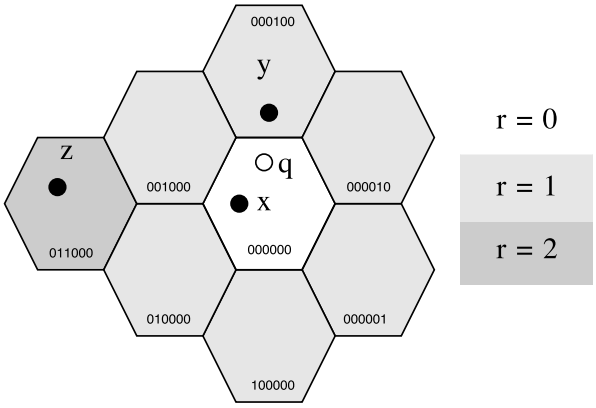


Abbildung 2.12: *Multi-Probe-Hashing*: Adjazente Hashgebiete bekommen mit hoher Wahrscheinlichkeit Codes, die einen geringen Abstand im Hamming-Raum besitzen. Durchsuchen der Gebiete bis zu einem festen Abstand r liefert mehr potentielle Nachbarn von q.

tung vom metrischen Raum in den Hamming-Raum der Codes gering ist. Ist dies möglich, können nun benachbarte Code-Gebiete, bis zu einem festgelegten Hamming-Abstand vom Code der Anfrage, durchsucht werden.

Das Recall-Problem Es ist offensichtlich, dass der Hammingraum der Codes im Ranking-Verfahren eine niedrige Dimension haben muss. Andernfalls hätte man durch die Einbettung in einen Coderaum keinen Vorteil gegenüber einer klassischen linearen Suche gewonnen. Grobe Codes haben aber wiederum den Nachteil, dass die Präzision der Suche darunter leidet. Weiterhin ist die Methode für sehr große Datensätze ungeeignet, da sich hier die lineare Laufzeit deutlich von den anderen Verfahren abhebt (Jingdong Wang, H. T. Shen, Song u. a., 2014). Vorteil ist aber, dass die Verfahren deutlich mehr echte Nachbarn zurückliefert, als die Table-Lookup-Verfahren. Table-Lookup Verfahren ermöglichen längere und damit feinere Codes. Sie liefern aber keine Garantie, dass die gesuchten Nachbarn tatsächlich gefunden werden. In der Praxis wird tatsächlich nur ein Bruchteil der Kandidaten zurückgegeben. Um diese Schwäche auszugleichen, muss bei beiden Verfahren gesucht werden. Die Multi-Probe-Verfahren werten dabei bis zu einem Hamming-Abstand r alle benachbarten Code-Gebiete aus. Hat ein Code also die Länge l, müssen dabei l^r Gebiete durchsucht werden. Für Codelängen, die eine hinreichende Präzision in der Praxis liefern ($l > 100$ bit), entstehen schon für kleine r ($r > 3$) schnell Suchzeiten, die keine Vorteile gegenüber der linearen Suche mehr bieten (Grauman und Fergus, 2013; Masci u. a., 2013). Die Multi-Table-Verfahren haben hier den Vorteil, dass durch die Überlappung der Codegebiete mehrerer Hashfunktionen feine Unterteilungen möglich sind, wobei die Anzahl der Auswertungen

pro weiterer Unterteilung nur linear steigt. Nachteil ist hier jedoch, dass für jede weitere Tabelle zusätzlicher Speicher benötigt wird und dass die Anzahl der benötigten Tabellen, um zu einem gegebenen Datensatz einen guten Recall zu erreichen, im Vorfeld unbekannt ist (Jingdong Wang, H. T. Shen, Song u. a., 2014).

Kapitel 3

Das Recall-Problem im Multi-Table-Hashing

Nachdem im letzten Kapitel Hashfunktionen für die approximative Lösung des Suchproblems in hohen Dimensionen vorgestellt wurden, soll in diesem Kapitel auf das erwähnte Recall-Problem genauer eingegangen werden. Wie im letzten Kapitel erläutert, versuchen Multi-Table Hashverfahren durch redundantes Speichern der Datenpunkte in mehrere Tabellen den Recall zu erhöhen. Dabei ist es in der Praxis oft unbekannt, wieviele Tabellen für einen gegebenen Datensatz für *guten* Recall ausreichen. Um diesen Begriff etwas zu schärfen, wird im ersten Abschnitt eine Definition einer *idealen Hashfunktion* vorgeschlagen, die im Bezug auf Präzision und Recall perfekt ist. Es wird gezeigt, dass solche Funktionen, außer für triviale Datensätze, unmöglich sind. Das Modell wird daher auf eine *Familie von idealen Hashfunktionen* erweitert. Eine solche lässt sich beweisbar immer konstruieren – die Anzahl der benötigten Hashfunktionen ist aber entscheidend für die Suchzeit. Um das Problem besser greifen zu können, wird im zweiten Abschnitt ein kombinatorisches Modell der r-Umgebungssuche vorgestellt und seine Relation zu dem Modell einer idealen Hashfamilie hergeleitet. Im dritten Abschnitt wird wiederum mithilfe dieses kombinatorischen Modells eine Ober- und eine Untergrenze für die Anzahl benötigter Hashfunktionen im Modell einer idealen Hashfamilie hergeleitet. Deren praktische Anwendbarkeit wird im schließenden Abschnitt diskutiert.

3.1 Ideale Hashfunktionen

Betrachten wir den Fall einer *idealen* Hashfunktion h. Die Anforderungen an eine solche sind einfach zu formulieren. Sie sollte jedem Datenpunkt $x \in X$ und jedem Anfragepunkt $q \in M$ genau dann denselben Hashwert zuweisen,

wenn diese nahe genug beieinander liegen. Stein (2007) folgend, bedeutet dies:

$$\forall x \in X \; \forall q \in M \; : \; d(q,x) < r \Leftrightarrow h(x) = h(q)$$

In dieser Aussage sind zwei Teilforderungen an h verknüpft, die sich in beiden Richtungen des Implikationspfeils wiederfinden lassen.

Definition 3.1.1. Sei h eine Hashfunktion $h\colon M \to \mathbb{Z}$.

- h ist *präzise* über (M, X), wenn

$$\forall q \in M \; \forall x \in X \; : \big(h(x) = h(q) \Rightarrow d(x,q) < r\big)$$

- h ist *erschöpfend* über (M, X), wenn

$$\forall q \in M \; \forall x \in X \; : \big(d(q,x) < r \Rightarrow h(q) = h(x)\big)$$

- h ist *ideal* über (M, X), wenn sie präzise und erschöpfend ist.

Das Problem Offensichtlich lässt sich dieser Wunsch schon in sehr einfachen Suchkonstellationen nicht mehr erfüllen. Sei z.B. $M = \big\{0, \frac{1}{2}, 1\big\}$, $X = \{0, 1\} \subset M$ und $r = 1/2$, dann ergibt sich für $q = 1/2$ das unlösbare Problem, dass zum einen $h(0) \neq h(1)$ und zum anderen aber $h(1/2) = h(0)$ und $h(1/2) = h(1)$ gelten muss. Hierin liegt das im vorigen Abschnitt eingeführte Recall-Problem begründet.

Nutzung mehrerer Tabellen Multi-Table-Verfahren umgehen das Problem unter der Nutzung mehrerer Tabellen. Hierzu erlauben wir zwei Hashfunktionen h_1 und h_2, die nun nicht nicht mehr jedem Anfragepunkt q und jedem Datenpunkt x denselben Hashwert zuweisen, sollten diese benachbart sein. Stattdessen fordern wir nur, dass nicht benachbarte Paare (q, x) in keinem Fall denselben Hashwert erhalten sollen. Um dennoch alle Nachbarschaften in einem gemeinsamen Hashwert abzubilden, schließen wir das Modell mit der Forderung ab, jede mögliche Nachbarschaft eines q und eines x mit mindestens einer Hashfunktion h_i abzudecken. Die obengenannte Konstellation wäre dann mithilfe der Hashfunktionen

$$h_1(0) = 0, \; h_1\left(\frac{1}{2}\right) = 0, \; h_1(1) = 1$$

$$h_2(0) = 0, \; h_2\left(\frac{1}{2}\right) = 1, \; h_2(1) = 1$$

lösbar. Diese Forderung für mehrere Hashfunktionen lässt sich nun allgemein formulieren als

Definition 3.1.2. Sei $\mathcal{H}$ eine Familie von Hashfunktionen.

1. $\mathcal{H}$ ist *präzise* über (M, X), wenn $\forall h \in \mathcal{H} : (h \text{ ist präzise})$.
2. $\mathcal{H}$ ist *erschöpfend* über (M, X), wenn
$$\forall q \in M \; \forall x \in X \; : \; d(x, q) < r \Rightarrow \big(\exists h \in \mathcal{H} \; : \; h(x) = h(q)\big)$$
3. $\mathcal{H}$ ist *ideal* über (M, X), wenn sie präzise und erschöpfend ist.

Eine solche Familie lässt sich für beliebige metrische Räume M und endliche Datenmengen $X = \{x_1, \ldots, x_n\}$ auf naive Art und Weise immer konstruieren (vgl. Abb. 3.1).

Lemma 1. *Sei M ein metrischer Raum, $X \subset M$, $|X| < \infty$. Dann existiert eine ideale Familie von Hashfunktionen $\mathcal{H}$ über (M, X).*

Beweis. Definiere die Menge aller Schnitte von r-Bällen, die sich um Datenpunkte $x \in X$ bilden lassen:
$$\mathcal{S} = \bigcup_{I \subset X} \left\{ \bigcap_{x \in I} B_r(x) \right\}$$
Sei $\phi \colon \mathcal{S} \to \mathbb{N}_{\geq 1}$ eine (injektive) Abzählung dieser Menge. Für jeden Punkt $x \in X$ definiere nun eine Hashfunktion h_x wie folgt:
$$h_x(y) = \begin{cases} 0 & y \in B_r(x) \\ z(x, y) & y \notin B_r(x) \end{cases}$$
mit
$$z(x, y) = \min \big\{ \phi(A) \mid x \in A \wedge A \in \mathcal{S} \wedge A \not\subset B_r(y) \big\}$$
Sei nun q ein möglicher Anfragepunkt. Gilt $d(q, x) < r$, dann ist $q \in B_r(x)$ und es gilt für mindestens die Funktion h_x, dass $h_x(q) = h_x(x) = 0$. Damit ist die Erschöpfungseigenschaft erfüllt. Gilt umgekehrt $d(q, x) > r$, dann kann es nach Definition von z keine Menge $S \in \mathcal{S}$ geben, für die sowohl $q \in S$ als auch $x \in S$ gilt. Aufgrund der Injektivität von ϕ folgt damit, dass $h_x(x) \neq h_x(q)$. □

Die obengenannte Familie ist natürlich viel zu groß, um eine praktische Relevanz zu haben. Da ihre Anwendung das Auswerten der n Hashfunktionen erfordern würde, gäbe es keinen Vorteil im Vergleich zur linearen Suche. Dies motiviert die Frage, wie klein eine Familie von Hashfunktionen $\mathcal{H}$ für einen metrischen Raum M und einen Datensatz X sein kann, um noch ideal sein. Das erste Beispiel zeigte, dass eine zu kleine Familie im Allgemeinen nicht ausreicht. Das zweite Beispiel zeigt eine ausreichende, aber viel zu große Anzahl von Tabellen. In der weiteren Diskussion bezeichnen wir die minimal notwendige Größe von $\mathcal{H}$ als ν.

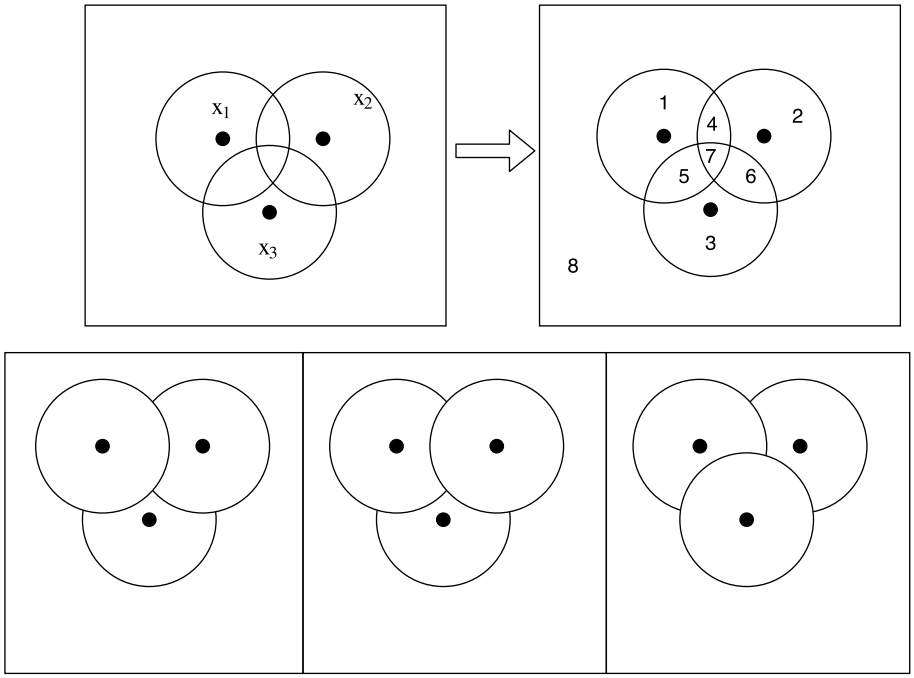


Abbildung 3.1: *Die ideale Familie von Hashfunktionen aus dem Beweis von Lemma* 1: Zu gegebenem Raum und Datenmenge werden alle Schnitte der die Datenpunkte umgebenden r-Bälle gebildet und durchnummeriert. Mithilfe der Nummerierung kann so die ideale Familie $\{h_1, h_2, h_3\}$ konstruiert werden.

3.2 Daten-Anfrage-Graphen zur Modellierung des Suchproblems

Um die Recall-Problematik besser beschreiben zu können, führen wir ein kombinatorisches Modell ein. Für einen metrischen Raum (M, d) und eine Datenmenge $X \subset M$ sei der *Daten-Anfrage-Graph* (DAG) der Graph $G(V, E)$ mit der Knotenmenge $V = M$ und der Kantenmenge

$$E = \big\{\{u, v\} \mid d(u, v) < r \wedge u \in M \wedge v \in X\big\}$$

Für die weitere Diskussion bezeichne alle Knoten in X als *Datenpunkte* und alle Knoten in $X^C = M \setminus X$ als *Anfragepunkte*. G hat nach Definition von E die Eigenschaft, dass Anfragepunkte keine Kanten teilen und nur Kanten zwischen einem Datenpunkt und einem Anfragepunkt bzw. zwei Anfragepunkten existieren. Inzidente Kanten (q, x) an einem Anfragepunkt $q \in X^C$ zeigen, welche Datenpunkte $x \in X$ zu diesem Anfragepunkt durch eine exakte r-Umgebungssuche zurückgegeben werden müssen. Gibt es eine Kante zwischen zwei Datenpunkten x_1, x_2, sind diese gegenseitig in ihrer jeweiligen

r-Umgebung enthalten. Es ist zu beachten, dass in G nur endlich viele Datenknoten existieren und jeder Anfragepunkt $q \in X^C$ damit nur endlich viele adjazente Kanten (und damit endlichen Grad) besitzt. Jeder Anfragepunkt $q \in X^C$ hat weiterhin genau eine Menge von Nachbarn $\mathcal{N}(q) \subset X$. Insgesamt gibt es maximal $2^{|X|} < \infty$ verschiedene Möglichkeiten, solche Mengen von Nachbarn zu bilden. Mit der Äquivalenzrelation

$$q_1 \sim q_2 \;\Leftrightarrow \mathcal{N}(q_1) = \mathcal{N}(q_2)$$

lassen sich so endlich viele Äquivalenzklassen $[q]$ konstruieren. Identifiziert man nun alle Knoten von G, die in der selben Äquivalenzklasse liegen, wird G ein *reduzierter Daten-Anfrage-Graph* von Zusammenhangsklassen von Anfragepunkten und den zurückzugebenden Datenpunkten für jede dieser Zusammenhangsklassen.

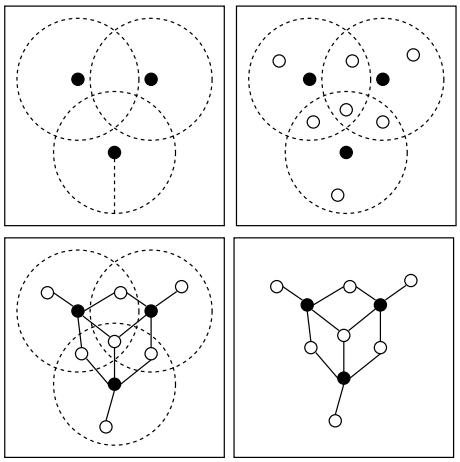


Abbildung 3.2: *Konstruktion eines reduzierten DAG aus einer dreielementigen Datenmenge*: *Oben links*: drei Punkte aus X und ihr umgebender r-Ball. *Oben rechts*: die weißen Punkte sind Repräsentanten der Zusammenhangsklassen. *Unten links*: Kanten verbinden Datenpunkte und Zusammenhangsklassen, in denen sie enthalten sind. *Unten rechts*: Reduktion des metrischen Suchproblems auf ein rein kombinatorisches.

Hashfunktionen und DAG Partitionierungen Wie im vorigen Kapitel diskutiert, zerlegt eine Hashfunktion h den Raum M in Gebiete mit gleichem Hashwert. Fasst man nun Knoten mit selbem Hashwert zusammen, ergibt sich eine Partitionierung der Knotenmenge

$$V = V_1 \cup \cdots \cup V_k,\ i \neq j \Rightarrow V_i \cap V_j = \emptyset$$

und damit induzierte Teilgraphen

$$G \supset G_1 \cup \cdots \cup G_k$$

Sind die erzeugten Knotenmengen durch keine Kante verbunden, gilt sogar

$$G = G_1 \cup \cdots \cup G_k$$

Umgekehrt existiert für je so eine Zerlegung genau eine Hashfunktion. Für alle Punkte $y \in M$ existiert genau eine Zerlegungsmenge V_i, sodass $y \in V_i$. Damit ergibt sich die eindeutige Hashfunktion $h\colon y \mapsto i$. Zwischen präzisen

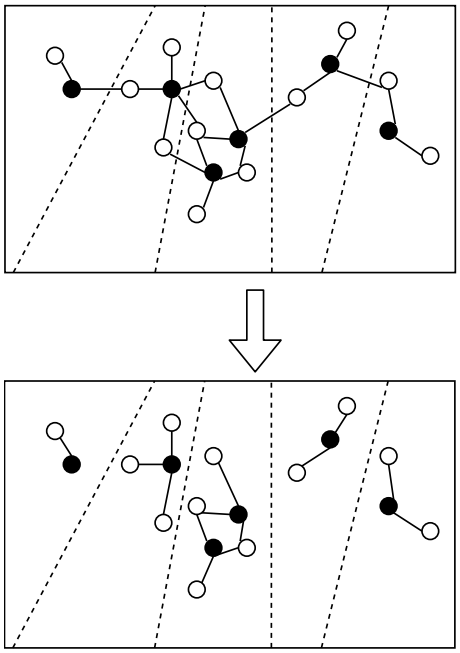


Abbildung 3.3: *Hashfunktionen und DAGs*: Die Hashfunktion zerlegt den Raum in Gebiete mit gleichem Hashwert. Kanten, die über zwei Gebiete hinweg verlaufen, gehen verloren. Haben innerhalb eines Gebiets ein weißer Anfragepunkt und ein schwarzer Datenpunkt keine gemeinsame Kante, entsteht ein Präzisionsfehler (mittleres Gebiet). Innerhalb der Gebiete befinden sich Teilgraphen des ursprünglichen (reduzierten) DAG.

Hashfunktionen und Partitionierungen der Knotenmenge des DAG besteht nun folgender Zusammenhang:

Lemma 2. *Sei $V_1 \ldots V_k$ eine Partitionierung der Knoten eines Daten-Anfrage-Graphen G und h die eindeutig daraus zu konstruierende Hashfunktion. Dann ist h genau dann präzise, wenn*

$$\forall i\ \forall x \in X \cap V_i\ \forall y \in V_i \setminus \{x\}\ :\ \{x, y\} \in E|_{V_i}$$

Beweis. Seien $x \in X \cap V_i$, $y \in V_i$.
$\Rightarrow$:
Ist h eine präzise Hashfunktion, so gilt

$$h(x) = h(y) \Rightarrow d(x, y) < r$$

und damit auch, dass

$$\{x, y\} \in E \Rightarrow \{x, y\} \in E|_{V_i}$$

$\Leftarrow$:
Da h die aus der Zerlegung konstruierte Hashfunktion ist, gilt $h(x) = h(y)$. $\{x, y\} \in E|_{V_i}$ impliziert weiterhin, dass $d(x, y) < r$. Damit ist h präzise. □

3.3 Abschätzungen der minimal benötigten Anzahl von Hashfunktionen für perfekten Recall im Multi-Table-Hashing

Mit den im vorigen Abschnitt eingeführten Daten-Anfrage-Graphen lässt sich die oben eingeführte Zahl ν nun durch eine obere und eine untere Schranke mit kombinatorischen bzw. algorithmischen Überlegungen abschätzen.

Abschätzung mit einer unteren Schranke Für die untere Schranke reicht es zu zeigen, dass für einen Anfragepunkt durch eine Hashfunktion nie mehr als eine Untergrenze von Nachbarschaften im DAG modelliert werden können. Will man aber auf einen perfekten Recall nicht verzichten, benötigt man mehr Hashfunktionen.

Definition 3.3.1. Für einen gegebenen Graphen $G = (V, E)$, definiere die *Schnittzahl* $cc(G)$ als die minimale Anzahl von Cliquen in G, die notwendig sind, um alle Kanten von G zu überdecken.

Satz 1. *Sei (M, d) ein metrischer Raum, $X \subset M$ eine endliche Datenmenge und G der Daten-Anfrage-Graph von $((M, d), X)$. Sei weiterhin*

$$cc_{\max} = \max_{q \in X^C} cc\left(G|_{\mathcal{N}(q)}\right)$$

die größtmögliche Schnittzahl, die eine Nachbarschaft eines Anfrageknoten q besitzen kann. Dann gilt

$$cc_{\max} \leq \nu$$

Beweis. Sei q ein o.g. Maximierer und $\mathcal{H}$ eine ideale Familie von Hashfunktionen. Seien weiterhin

$$V_1, \ldots, V_k$$

die von $\mathcal{H}$ erzeugten Partitionierungsmengen der Knoten von G, die q enthalten. Da jede Hashfunktion $h \in \mathcal{H}$ präzise ist, sind diese Mengen Cliquen in G und damit insbesondere Cliquen in $G|_{\mathcal{N}(q)}$. Da $\mathcal{H}$ weiterhin erschöpfend ist, ist jede Kante zwischen q und seinen benachbarten Datenpunkten in einer dieser Cliquen enthalten. Damit überdecken diese Cliquen $G|_{\mathcal{N}(q)}$ vollständig. Mit der Definition der Schnittzahl folgt

$$cc_{\max} = cc\left(G|_{\mathcal{N}(q)}\right) \leq k$$

□

Abschätzung mit einer oberen Schranke Um die obere Schranke abzuschätzen, wird auf den reduzierten Daten-Anfrage-Graphen ein Algorithmus angewandt, der eine Menge von Zerlegungen des Graphen erzeugt und eine beweisbare Anzahl an Zerlegungen nicht überschreitet. Für jede dieser Zerlegungen wird gezeigt, dass sie einer präzisen Hashfunktion entsprechen und gemeinsam jede Nachbarschaft im Graphen modellieren.

Satz 2. *Sei*

$$\deg_{\max} = \max_{q \in X^C} \deg q$$

der Maximalgrad eines Anfragepunkts $q \in X^C$ in G. Dann gilt

$$\nu \leq \deg_{\max}$$

Der o.g. Maximalgrad lässt sich geometrisch beschreiben als die maximale Zahl von r-Bällen, in denen ein $q \in M$ enthalten ist. Dies ist äquivalent zur maximalen Anzahl von Datenpunkten x, die sich in einen Ball mit Radius r unterbringen lassen. Um Satz 2 zu beweisen, reicht es, zu zeigen, dass man mit $\deg_{\max}$ Hashfunktionen perfekten Recall erzielen kann. Dies lässt sich konstruktiv durch Anwendung des Algorithmus 3.1 auf die reduzierte Version von G zeigen.

Lemma 3. *Die innere Schleife von Algorithmus 3.1 terminiert. Daher befindet sich jeder Knoten $v \in V$ in einer der Partitionierungsmengen S.*

Beweis. Sei o.B.d.A $y \in X^C$ ein Anfrageknoten mit $\deg(y) > 0$. Dann sei für Durchlauf t der inneren Schleife

$$A_t = \left\{y' \in X^C \setminus \{y\} \mid \deg(y') \geq \deg(y)\right\}$$

Algorithmus 3.1 : Konstruktionsalgorithmus für eine hinreichende Partitionierung von G

Input : Daten-Anfrage-Graph $G = (V, E)$
Output : Eine Menge $\mathcal{C}$ von Knotenpartitionierungen von G
$\mathcal{C} \leftarrow \emptyset$
while $E \neq \emptyset$ **do**
  $W \leftarrow \{y \in X^C \mid \deg(y) = 0\}$
  $\mathcal{V} \leftarrow \{W\}$
  $V' \leftarrow V \setminus W$
  while $V' \neq \emptyset$ **do**
    if $\max_{y \in X^C} \deg(y) > 0$ **then**
      $y \leftarrow argmax_{y \in X^C} \deg(y)$ beliebig
      $x \leftarrow argmax_{x \in \mathcal{N}(y)} \deg(x)$ beliebig
      $S \leftarrow \mathcal{N}(x) \cap X^C \cup \{x\}$
      $\mathcal{V} \leftarrow \mathcal{V} \cup \{S\}$
      $V' \leftarrow V' \setminus S$
      $E \leftarrow E \setminus E|_S$
    else
      Wähle $x \in X$ beliebig.
      $V' \leftarrow V' \setminus \{x\}$
      $\mathcal{V} \leftarrow \mathcal{V} \cup \{\{x\}\}$
    end
  end
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathcal{V}\}$
end

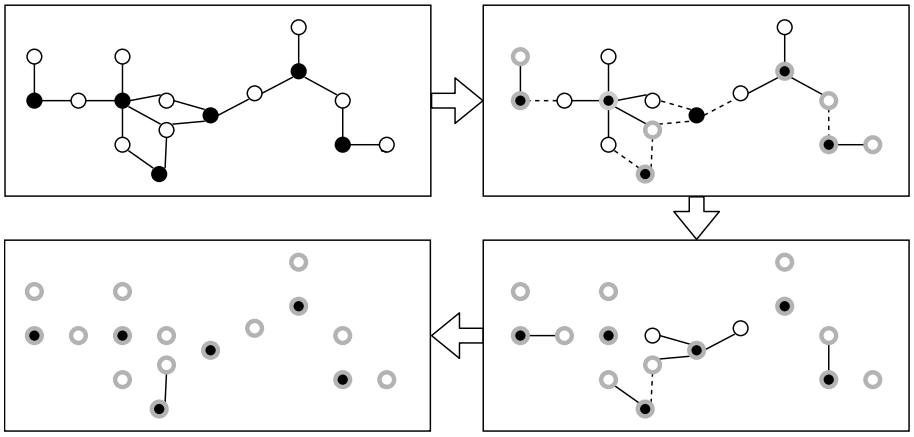


Abbildung 3.4: *Die Anwendung von 3.1 Algorithmus auf einen Beispiel-DAG*: In drei Schritten wird der DAG oben links in drei verschiedene Zerlegungen aufgeteilt, die alle Kanten des Graphen überdecken. Jedes Einzelbild beschreibt einen Durchlauf der äußeren Schleife. Die in den Durchläufen der inneren Schleife ausgewählten Anfrage- bzw. Datenpunkte sind grau umrandet.

die Menge aller Anfragepunkte, deren Grad höher oder gleich dem Grads von y ist. Sei $B_t = A_t \cap S_t$ die Teilmenge dieser Anfragepunkte, die innerhalb des *If*-Blocks gewählt werden. Ist y in B_t enthalten, wird er in diesem Schritt aus der Knotenmenge entfernt. Andernfalls gilt $|A_{t+1}| \leq |A_t| - |B_t| < |A_t|$. Damit ist y spätestens nach $|A_0|$ Durchläufen der einzige Anfragepunkt mit maximalem Grad $\deg(y) > 0$. Nun wird ein zu y benachbarter Datenpunkt x gewählt und dessen gesamte Nachbarschaft an Anfragepunkten zusammen mit y entfernt. □

Lemma 4. *Nach Durchlaufen der inneren Schleife des Algorithmus 3.1 gilt im entstehenden Residualgraphen G':*

$$\deg_{\max}^{(G')} < \deg_{\max}^{(G)} \ \vee \ \deg_{\max}^{(G)} = 0$$

Beweis. Sei $y \in X^C$ ein Anfrageknoten mit Maximalgrad $\deg_{\max}^{(G)} > 0$. Nach Lemma 3 gibt es einen Durchlauf, in dem y aus V' als Teil der Nachbarschaft $\mathcal{N}(x)$ eines Datenpunkts $x \in \mathcal{N}(y) \cap X$ entfernt wird. In diesem Schritt wird die Kante $\{x, y\}$ aus E entfernt und im Residualgraphen gilt

$$\deg(y)^{(G')} \leq \deg(y)^{(G)} - 1$$

□

Eine direkte Folge von Lemma 4 ist, dass auch die äußere Schleife von Algorithmus 3.1 in $l \leq \deg_{\max}$ Schritten terminiert. Dies impliziert insbesondere, dass jede Kante $\{u, v\} \in E$ in einer der entstandenen Zerlegungsmengen enthalten ist.

Lemma 5. *Jede Zerlegung $\mathcal{V}$, die beim Durchlaufen der inneren Schleife des Algorithmus' 3.1 entsteht, erfüllt die Axiome von Lemma 2.*

Beweis. Sei $V_i \in \mathcal{V}$ eine Zerlegungsmenge. In V_i ist maximal der eine Datenpunkt $x \in X$ enthalten, dessen Nachbarschaft ausgewählt und zu $\mathcal{V}$ hinzugefügt wurde. Alle anderen in V_i enthaltenen Knoten sind Anfragepunkte. Da nur Anfragepunkte $q \in X^C$ aus der Nachbarschaft von x zu V_i hinzugefügt wurden, gilt für alle diese $(x, q) \in E|_{V_i}$. □

Mit diesen Lemmata ausgestattet, beweist die Anwendung von Algorithmus 3.1 konstruktiv Satz 2.

Beweis von Satz 2. Algorithmus 3.1 liefert $l \leq \deg_{\max}$ Zerlegungen $\mathcal{V}_1 \dots \mathcal{V}_l$ der Knotenmenge von G. Aus jeder dieser Zerlegungen $\mathcal{V}_i$ lässt sich nach Lemma 5 eine präzise Hashfunktion h_i konstruieren. Da weiterhin nach Lemma 4 jede Kante $(x, y) \in E$ in einer der Mengen einer der konstruierten Zerlegungen enthalten sein muss, gilt für mindestens eine dieser präzisen Hashfunktionen h_i, dass:

$$h_i(x) = h_i(y)$$

Zu jedem Anfragepunkt $y \in M^C$ können also alle benachbarten Datenpunkte $x \in B_\epsilon(y) \cap X$ durch Auswerten von $h_1^{-1}(y) \dots h_l^{-1}(y)$ gefunden werden. Damit ist die so konstruierte Familie ideal im Sinne der o.g. Definition. □

3.4 Praktische Relevanz des Ergebnisses

Nachdem beide Abschätzungen bewiesen wurden, soll nun auf ihre praktische Relevanz eingegangen werden. Hierzu soll zunächst die Nützlichkeit des Algorithmus' 3.1 erörtert werden, bevor auf die Qualität der Abschätzungen und ihre Bedeutung für praktische Berechnungen auf einem gegebenen Datensatz eingegangen wird. Abschließend soll der Bezug zum im Abschnitt 2.2 eingeführten Lernproblem von guten Hashfunktionen hergestellt werden.

Nützlichkeit von Algorithmus 3.1 Das Ergebnis ist in dieser Form rein theoretischer Natur. Der obengenannte Algorithmus eignet sich nur als Zwischenschritt des Beweises. Für reale Datensätze würde er an der exponentiellen Größe $\mathcal{O}\left(2^{|X|}\right)$ des reduzierten DAG scheitern. Auch ist die Komplexität der Zerlegungsmengen bei vielen Überlappungen derart hoch, dass es unwahrscheinlich scheint ein Modell einer Hashfunktion zu finden, das in seiner Auswertung nicht bereits eine Zeitkomplexität von $\mathcal{O}(|X|)$[1] besitzt. Eine direkte

1 z.B. durch Traversierung aller Datenpunkte

Verwendung des Algorithmus zur Konstruktion einer idealen Familie scheint für echte Datensätze nicht infrage zu kommen.

Exakte Bestimmung von ν Die abgeschätzte Zahl ν wiederum ist von Wert, wenn man zu einem gegebenen Datensatz die minimal benötigte Anzahl von Hashtabellen im Multi-Table-Hashing ermitteln möchte. Leider sind die Abschätzungen nach oben bzw. nach unten echte Ungleichungen (vgl. Abbildung 3.5). Eine exaktere Bestimmung von ν wäre Gegenstand zukünftiger Forschung.

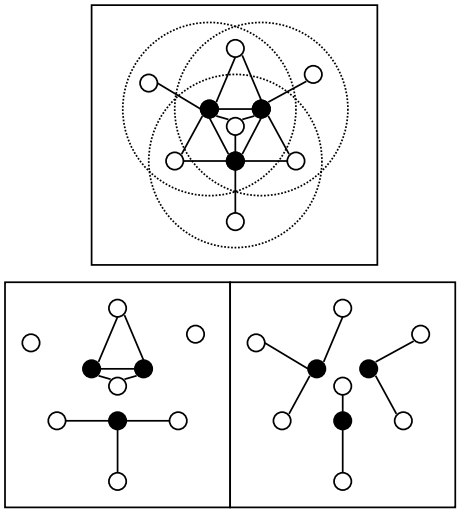


Abbildung 3.5: *Ein Beispiel für die obere und die untere Schranke*: Für den DAG, der sich aus der oberen Punktkonstellation ergibt, wäre 1 die in dieser Arbeit errechnete untere Schranke und 3 die obere Schranke für ν. Dennoch gibt es eine Möglichkeit mit nur 2 Knotenzerlegungen alle Kanten zwischen Anfrage- und Datenpunkten abzubilden. Weniger Zerlegungen sind nicht möglich, ohne Präzision zu verlieren. Damit ist $\nu = 2$.

Exakte Bestimmung der Untergrenze Die Bestimmung der Untergrenze impliziert die Berechnung der Schnittzahl für die Nachbarschaftsgraphen in G. Das Problem eine solche minimale Überdeckung von Cliquen für alle Kanten eines gegebenen Graphen zu finden, ist auch als *Clique-Edge-Cover*-Problem bekannt und NP-vollständig (Orlin, 1977). Die Anzahl der Cliquen dieser Minimalzahl kann jedoch mit einer beliebigen Cliquen-Überdeckung von oben abgeschätzt werden. Dies kann z.B. mit einer Greedy-Strategie geschehen. Hierzu wählt man im gegebenen Graphen einen Knoten v mit maximalem Kantengrad. Ausgehend von diesem konstruiert man eine maximale Clique in der v enthalten ist und entfernt alle ihre Kanten aus G. Das Verfahren wird

nun auf dem entstehenden Residualgraphen solange wiederholt, bis keine Kanten mehr in ihm enthalten sind. Für eine solche Approximation $cc(G)_{\text{greedy}}$ gilt also:

$$cc(G) \leq cc(G)_{\text{greedy}}$$

Exakte Bestimmung der Obergrenze Auch die Bestimmung von $\deg_{\max}$ ist nicht einfach. Wie erläutert entspricht $\deg_{\max}$ der maximalen Anzahl von Überlappungen die alle möglichen r-Bälle um Datenpunkte bilden können, bzw. der maximalen Anzahl an r-Bällen um Datenpunkte in denen ein Punkt in M enthalten ist. Äquivalent formuliert bedeutet dies

$$\deg_{\max} = \max_{y \in M} \left| X \cap B_r(y) \right|$$

algorithmisch bestimmen zu müssen. Dieses Optimierungsproblem ist als *Maximum-Enclosing-Ball*-Problem bekannt und in der reellen, euklidischen Ebene mit Zeitkomplexität $\mathcal{O}\left(|X|^2\right)$ lösbar (Chazelle und Lee, 1986). In Figueiredo und Fonseca (2009) wurde gezeigt, dass es für den D-dimensionalen euklidischen Raum nicht schneller als in $\mathcal{O}\left(|X|^D\right)$ zu lösen ist. Sie zeigen darüber hinaus, dass auch jede Approximation von oben, in dem man leicht größere Bälle bis zu einem Radius von $(1+\epsilon)r$ akzeptiert, immer eine minimale Zeitkomplexität von $\mathcal{O}\left(|X|/\epsilon^{d-1}\right)$ besitzt. Einen derart optimalen Approximationsalgorithmus geben Figueiredo und Fonseca abschließend an. Für die dadurch bestimmte Größe $\deg_{\max}^{approx}$ gilt:

$$\deg_{\max} \leq \deg_{\max}^{approx}$$

Allgemeinere Lösungen sind dem Autor dieser Arbeit bisher nicht bekannt. Dennoch erscheint auch eine heuristische Lösung immer mithilfe von klassischen Optimierungsverfahren (z.B. durch genetische Programmierung (Banzhaf, 1998) oder Simulated Annealing (Hwang, 1988)) möglich zu sein. Eine solche Annäherung $\deg_{\max}^{heur}$ versucht, Kandidaten für einen optimalen Ball iterativ, in Bezug auf die Anzahl, der von ihnen überdeckten Datenpunkte, zu maximieren. Da die so erzielte Anzahl maximal das o.g. Optimum erreichen kann, gilt

$$\deg_{\max}^{heur} \leq \deg_{\max}$$

Umgekehrt haben alle Punkte innerhalb eines r-Balles den maximalen Abstand $2r$. Damit bildet die maximale Anzahl an $2r$-Nachbarn aus X eines Datenpunkts $x \in X$ eine simple obere Grenze für $\deg_{\max}$. Diese lässt sich durch paarweise Vergleiche mit dem Zeitaufwand $\mathcal{O}\left(|X|^2\right)$ ermitteln.

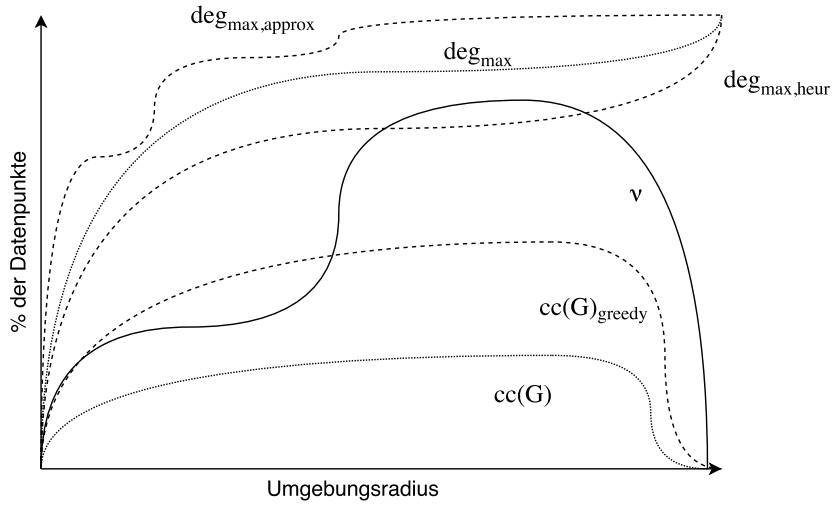


Abbildung 3.6: *Relation der Größen*: Die beiden angegegebenen Schranken sperren ν vollständig ein. Für die hier angegebene heuristische Obergrenze bzw. approximative Untergrenze ist dies u.U. nicht möglich. Die approximative Obergrenze überschätzt die obere Schranke echt.

Implikationen für das Lernproblem von Hashfunktionen Die in diesem Kapitel gewonnenen Erkenntnisse zeigen, dass im Szenario des Multi-Table-Hashings eine Minimalanzahl verschiedener Hashfunktionen für guten Recall und gute Präzision notwendig sind. Die Analyse mithilfe der Daten-Anfrage-Graphen legt zudem nahe, dass diese verschiedenen Hashfunktionen aufeinander abgestimmt sein sollten, um die jeweiligen Schwächen im Bezug auf das Recall-Problem – gerade in Gebieten mit hohem Überlappungsgrad – auszugleichen. Für das Lernproblem, eine datenabhängige Familie Hashfunktionen zu einem gegebenen Datensatz zu finden, folgt daraus, dass die zu lernenden Hashfunktionen nicht unabhängig voneinander, sondern im Bezug aufeinander gelernt werden sollten. In neueren Veröffentlichungen wie Jin u. a. (2013), X. Liu, He und B. Lang (2013), Jun Wang, Kumar und S.-F. Chang (2010) und Xu u. a. (2011) wird dieser Ansatz bereits verfolgt. In keiner dieser Publikationen wird jedoch eine Untergrenze von Tabellen angegeben, die für guten Recall im Multi-Table-Hashing notwendig sind. In folgenden Untersuchungen sollte also analysiert werden, ob sich aus den o.g. Erkenntnissen neue gute Ansätze ableiten lassen, um Familien von Hashfunktionen zu finden, die sich komplementär ergänzen.

Kapitel 4

Experimente

In diesem Kapitel, wird eine Approximation der zuvor abgeschätzen Grenzen auf einem bekannten Testdatensatz des Information-Retrieval ausgerechnet, um darauf aufbauend eine Idee der Größenordnung von ν für reale Daten zu gewinnen.

4.1 Versuchsaufbau

Datensatz Der Datensatz, der in diesem Experiment untersucht wird, ist die tf-idf-Einbettung des *twenty-newsgroups*-Korpus (K. Lang, 1995), die von der quelloffenen Machine-Learning-Bibliothek *scikit-learn* bereitgestellt wird (Pedregosa u. a., 2011). Dieser Korpus besteht aus 11.314 Beiträgen, die zu 20 verschiedenen Themengebieten in unterschiedlichen Newsgroups verfasst wurden. In der unverarbeiteten Form, beträgt seine Einbettungsdimension 130.107. Um das Verhalten der Grenzen in Abhängigkeit der Dimension des Datensatzes zu vergleichen, werden die Daten in diesem Experiment jedoch mit Zufallsprojektionen in Unterräume mit Dimension 10, 100, bzw. 1.000 projiziert. Die Dimensionsreduzierung per Zufallsprojektion bietet bei vergleichbarer Qualität den Vorteil einer deutlich schnelleren Berechnungszeit, verglichen mit anderen Ansätzen wie zum Beispiel einer Hauptkomponentenanalyse (PCA) (Bingham und Mannila, 2001). Da weiterhin die ausgeführten Approximationen in den begrenztem Zeitrahmen dieser Arbeit nicht auf dem vollständigen Datensatz ausgeführt werden konnten, wurde das Experiment auf 10 Testgruppen von jeweils 200 zufällig gewählten Vektoren der Einbettung eingeschränkt, deren errechnete Ergebnisse gemittelt wurden.

Approximation der Untergrenze Um die Untergrenze zu approximieren, wird in dieser Arbeit eine Greedy-Strategie gewählt. Dazu wird zunächst zum gegebenen Abstand r der Abstandsgraph G_r gebildet, in dem zwei Punkte

x, y adjazent sind, wenn sie einen Abstand $d(x, y) < r$ besitzen. Nun wird für jeden Knoten x dieses Abstandsgraphs der Untergraph $G_r|_{\mathcal{N}(x)}$ gebildet, der durch Einschränkung von G_r auf die Nachbarschaft von x ensteht. Auf jedem dieser Nachbarschaftsgraphen wird nun mit der im letzten Kapitel erklärten Greedy-Strategie eine Kanten-Überdeckung mit maximalen Cliquen konstruiert. Unter der Annahme, dass jeder Datenpunkt x als Anfragepunkt im Bezug auf die Datenmenge $X \setminus \{x\}$ aufgefasst werden kann, liefert dies (ein sehr grobes) Sampling von Anfragepunkten und ihren Schnittzahlen. Das Maximum dieser errechneten Schnittzahlen wird nun als Approximation der Untergrenze verwendet.

Approximation der Obergrenze Um die Obergrenze zu berechnen, wird versucht einen r-Ball zu finden, der maximal viele Datenpunkte beinhaltet. Dieser wird in dieser Arbeit durch einen genetischen Algorithmus approximiert (Banzhaf, 1998). Das Genom ist hierbei der Vektor, der das Zentrum des Balls festlegt (vgl. Abbildung 4.1). Die Fitness-Funktion

$$F(x) = |B_r(x) \cap X|$$

misst, wieviele Punkte in einem aktuellen Ball enthalten sind. Für eine feste Anzahl von 1.000 Schritten werden nun in jedem Schritt 100 Kandidaten auf ihre Fitness überprüft. Die besten 50 werden per Crossover entlang einer zufällig gewählten Dimension gekreuzt und mit einem normalverteilten Rauschen mutiert. Anschließend werden diese besten 50 zusammen mit 50 auf diese Art neu erzeugten Kandidaten in eine neue Generation von 100 Kandidaten für den nächsten Schritt zusammengefasst (vgl. Abbildung 4.2). Da die maximale Fitness der Kandidaten nur zunehmen kann, nähert sie sich dem gesuchten Maximalgrad an.

Parameter der Experimente Um die Veränderung der Grenzen in Abhängigkeit von verschiedenen Einflussfaktoren zu untersuchen, werden in dieser Arbeit drei Parameter variiert. Um die *Abhängigkeit von der Dimension* des Datensatzes zu beurteilen, werden beide Approximationen auf jeweils auf den 10, 100, bzw. 1.000 dimensionalen Unterraumprojektionen durchgeführt. Um die *Abhängigkeit vom gewählten Radius* r zu beurteilen, wird dieser linear von $r = 0$ bis $r = \max_{x,y \in X} d(x, y)$ variiert. Und um schließlich die *Abhängigkeit von der zugrundeliegenden Metrik* zu verstehen, werden diese Experimente sowohl für die Kosinus- als auch für die euklidische Distanz berechnet.

Darstellung der Ergebnisse Um die Ergebnisse besser vergleichbar zu machen, definiere für den Radius r und den durch r entstehenden Abstandsgra-

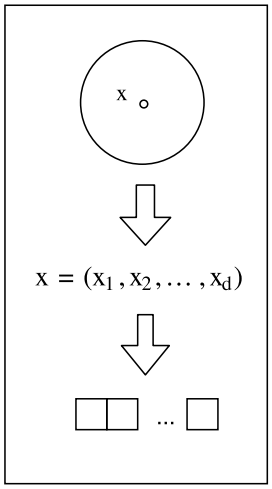


Abbildung 4.1: *Codierung der Bälle auf ein Genom*: Jede Komponente des Zentrums eines Balles definiert ein Gen seines Genoms.

phen G_r das durchschnittliche Verhältnis der Nachbarschaftsgrößen zur Gesamtzahl aller Datenpunkte

$$N_{G_r} = \mathbb{E}\left[\frac{\deg(x)}{|X|}\right]$$

Diese Größe skaliert die horizontale Achse. Die errechneten Werte werden dagegen auf der vertikalen Achse in Relation zur Gesamtgröße des Datensatzes skaliert.

4.2 Ergebnisse

Interpretation der Ergebnisse Unter der Annahme, dass die Approximationen keine zu geringe Qualität haben, wird zum einen ersichtlich, dass ν schon für geringe Umgebungsradien r einen großen Anteil an der Gesamtgröße des Datensatzes ausmacht. Damit eine perfekte Familie von Hashfunktionen also effizienter als eine lineare Suche sein kann, darf die Komplexität ihrer Auswertung nur um einen geringen Faktor größer sein als ein Vergleich zweier Punkte bei einer linearen Suche. Weiterhin lässt sich aus den Ergebnissen ableiten, dass die hier angegebenen Grenzen ν für kleine Umgebungsradien relativ präzise einschränken. Für große r bieten sie dagegen kaum Information über den Verlauf von ν. Bei der euklidischen Suche fällt darüber hinaus auf, dass die Obergrenze sehr stark unter dem Fluch der Dimensionalität leidet. Schon für kleine Umgebungsradien hilft sie kaum, ν von oben zu beschränken.

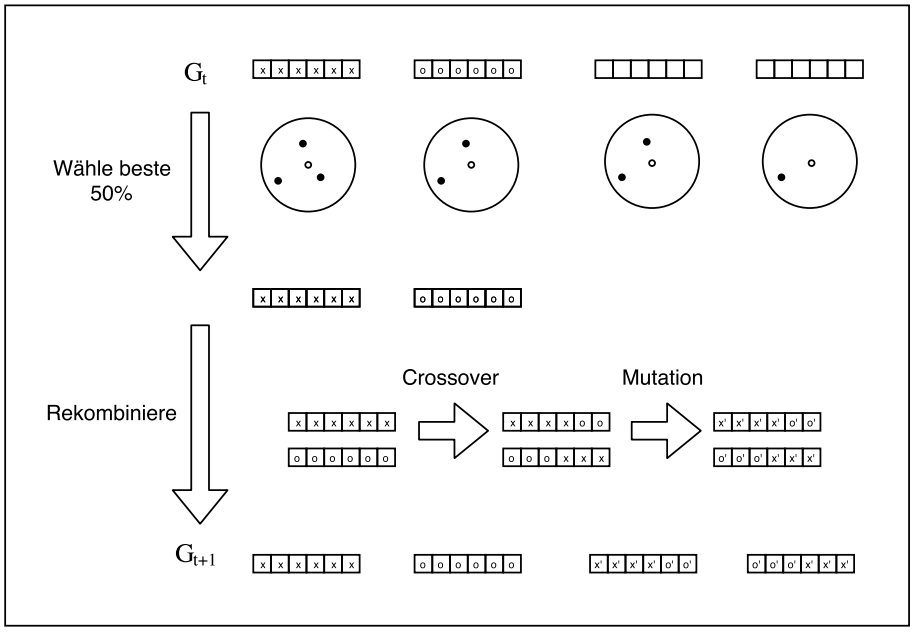
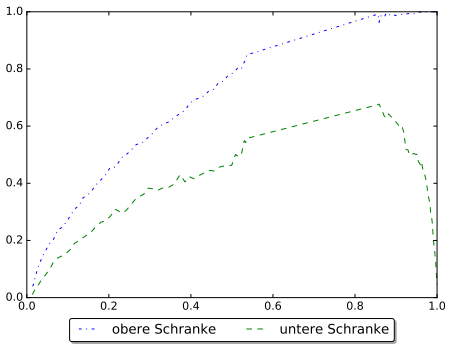


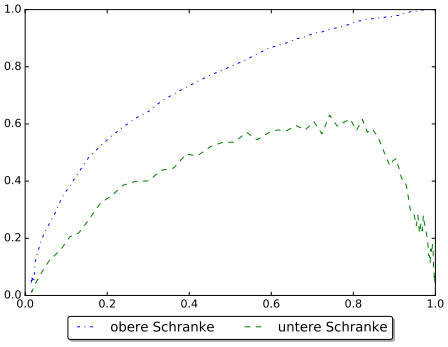
Abbildung 4.2: *Iterationsschritt der Optimierung*: Innerhalb einer Generation G_t von Bällen, werden die besten 50% ausgewählt. Die zufällige Rekombination dieser erzeugt die andere Hälfte der neuen Generation G_{t+1} von Kandidaten.

Bewertung der Qualität der Ergebnisse Da im Rahmen dieser Arbeit nicht genügend Zeit zur Verfügung stand, die Qualität der o.g. Approximationen zu evaluieren, können die berechneten Werte u.U. sehr stark von den exakten Grenzen abweichen. Zusätzlich gibt es wie bereits erläutert die Möglichkeit, dass sich ν oberhalb der oberen bzw. unterhalb der unteren hier berechneten Grenzen bewegt. Beides versieht die hier berechneten Grenzen mit einer Unsicherheit. Darüber hinaus wird nur eine Datenverteilung gewählt und aus dieser nur sehr wenige Samples in relativ niedrigen Dimensionen gewählt. Dies erschwert die Generalisierbarkeit der Ergebnisse auf größere und anders verteilte Datensätze. Trotz dieser Einschränkungen liefern die errechneten Ergebnisse einen Anhaltspunkt, um einen Verlauf von ν einschätzen zu können.

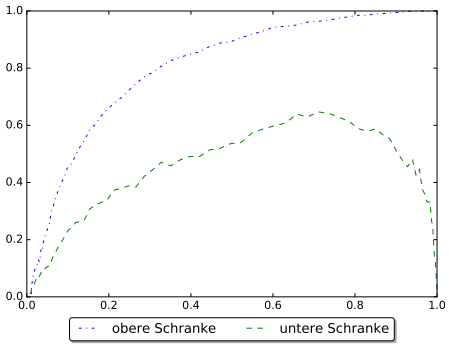
Metrik: Kosinus-Ähnlichkeit, Dimension: 10



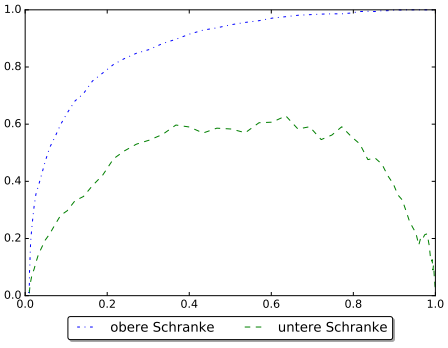
Metrik: euklidisch, Dimension: 10



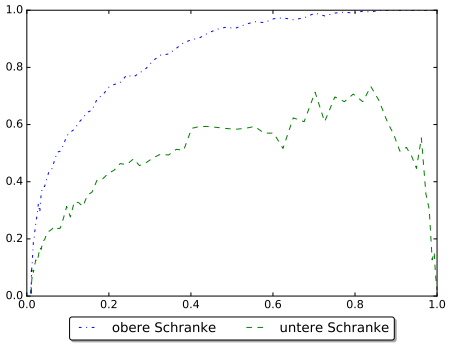
Metrik: Kosinus-Ähnlichkeit, Dimension: 100



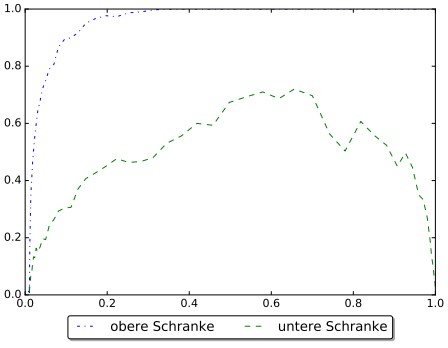
Metrik: euklidisch, Dimension: 100



Metrik: Kosinus-Ähnlichkeit, Dimension: 1000



Metrik: euklidisch, Dimension: 1000



Kapitel 5

Zusammenfassung und Ausblick

5.1 Zusammenfassung

In dieser Arbeit wurde das Problem, nächste Nachbarn in einem metrischen Datensatz zu finden, durch Anwendungen im Information Retrieval motiviert und danach im Hinblick auf hochdimensionale Datensätze problematisiert. Im Anschluss wurde eine approximative Lösung mithilfe von lokal-sensitiven Hashfunktionen erörtert, deren Konstruktionsprinzipien diskutiert und darauf aufbauend eine Einordnung zeitgenössischer Verfahren vorgenommen. Diese Hashfunktionen bilden Grundprimitive für darauf aufbauende approximative Retrieval-Verfahren, die sämtlich unter dem Problem eines schlechten Recalls von echten Nachbarn leiden. Für eines dieser Retrieval-Verfahren – dem Multi-Table-Hashing – wurde daraufhin untersucht, unter welchem Umständen ein ideales Retrieval möglich ist. Es wurde gezeigt, dass für perfektes Retrieval mehrere Tabellen zwingend notwendig sind, und mit den Daten-Anfrage-Graphen ein diskretes Modell angegeben, mit dessen Hilfe man die Anzahl dieser Tabellen durch eine obere und eine untere Schranke abschätzen kann. Da diese Schranken schwer zu berechnen sind, wurden einige Ansätze vorgestellt, um diese zu approximieren. Um weiterhin eine Einschätzung davon zu erhalten, wie sich diese Anzahl für verschiedene Metriken, Umgebungsabstände und Dimensionen auf Dokumentendatensätzen verhält, wurden Experimente zur Abschätzung dieser Schranken durchgeführt. Diese haben momentan noch illustrativen Charakter, motivieren aber eine tiefergehende empirische Untersuchung.

5.2 Ausblick

Exaktere Bestimmung von ν und Abschätzung engerer und leichter zu berechnender Grenzen Die ursprüngliche Forschungsfrage nach einer exakten Anzahl von benötigten Hashfunktionen für perfektes Multi-Table-Hashing konnte in dieser Arbeit nur mit einer Über- und einer Unterschätzung beantwortet werden. Beide Grenzen sind im Allgemeinen für einen gegebenen Datensatz schwer zu berechnen. Eine weitere Eingrenzung von ν und eine Bestimmung von leichter zu berechnenden Grenzen ist in aufbauenden theoretischen Studien erforderlich, um das Ergebnis praktisch nutzen zu können. Die, in dieser Arbeit vorgestellten, Daten-Anfrage-Graphen scheinen hierbei nützlich zu sein. Ihre Eigenschaften sollten – gerade im Bezug auf verschiedene Metriken, Umgebungsradien etc. – in weiteren Untersuchungen analysiert werden.

Weitere Experimente Die Ergebnisse der Experimente legen zudem nahe, dass die Obergrenze unter der euklidischen Metrik für hochdimensionale Datensätze wenig Information über den exakten Verlauf von ν liefert. Für die Kosinus-Distanz schränkt sie ν dagegen signifikant stärker ein. Dieses Ergebnis motiviert weitere Untersuchungen für andere Metriken. Nimmt man an, dass die Qualität der durchgeführten Approximationen nicht zu gering ist, scheint es für hochdimensionale Datensätze mithilfe des Multi-Table-Ansatzes praktisch unmöglich zu sein, perfekten Recall zu erhalten, ohne durch die Anzahl der Tabellen eine superlineare Suchzeit in kauf nehmen zu müssen. Dies motiviert, in weiteren Experimenten die berechneten Unter- und Obergrenzen bei existierenden Multi-Table-Verfahren einzusetzen und die Suchzeit mit der linearen Suche zu vergleichen. Die Experimente wurden weiterhin bisher nur auf sehr eingeschränkten Datensätzen durchgeführt. Die Datenverteilung entspricht zwar der eines Datensatzes, wie er üblicherweise im Information Retrieval vorkommt, die Menge der Daten ist jedoch noch zu gering und ihre Repräsentierung zu niedrigdimensional. Zusätzliche Experimente sollten ein breiteres Spektrum von Korpora untersuchen. Dabei sollte vor allem ein Fokus auf diejenigen gelegt werden, die für die Near-Duplicate-Erkennung besonders relevant sind. Darüber hinaus sollten sie auf auf einer größeren Teilmenge der Daten in ihrer ursprünglich sehr hochdimensionalen Repräsentierung durchgeführt werden.

Danksagung

Ich möchte mich bei allen bedanken, die mich mit viel Zeit und Geduld beim Anfertigen dieser Arbeit formal und inhaltlich beraten, unterstützt und ergänzt haben, die mir immer wieder geholfen haben das ein oder andere Motiviationstief zu überwinden, und die mir geholfen haben, mein Studium an der Bauhaus-Universität eine erfolgreiche und schöne Zeit werden zu lassen. Ein herzliches Danke an: Prof. Dr. Benno Stein, Michael Völske, Mama, Papa, Anne, Joni, Martin und alle anderen Freunde, Verwandten, Dozenten und Kommilitonen, die mich in meinen zweieinhalb Jahren in der Medieninformatik begleitet haben.

Literatur

Aggarwal, Charu C, Alexander Hinneburg und Daniel A Keim (2001). „On the surprising behavior of distance metrics in high dimensional space“. In: *International Conference on Database Theory*. Springer, S. 420–434 (siehe S. 11).

Andoni, Alexandr (2009). „Nearest Neighbor Search : the Old, the New, and the Impossible“. Diss. Massachusetts Institute of Technology (siehe S. 13, 18).

Andoni, Alexandr und Piotr Indyk (2006). „Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions“. In: *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*. IEEE, S. 459–468. ISBN: 0769527205. DOI: 10.1109/FOCS.2006.49 (siehe S. 14, 18, 27).

Andoni, Alexandr und Ilya Razenshteyn (2015a). „Optimal Data-Dependent Hashing for Approximate Near Neighbors“. In: *arXiv preprint arXiv:1501.01062v3*. arXiv: 1501.01062v3 [cs] (siehe S. 20).

— (2015b). „Tight Lower Bounds for Data-Dependent Locality-Sensitive Hashing“. In: *arXiv preprint arXiv:1507.04299*. arXiv: 1507.04299 [cs] (siehe S. 19, 20).

Banzhaf, Wolfgang (1998). *Genetic programming : an introduction ; on the automatic evolution of computer programs and its applications*. XIX, 470 S : graph. Darst; Literaturverz. S. 399 - 443. San Francisco, Calif.: Kaufmann [u.a.], 1998. ISBN: 155860510X; 3920993586 (siehe S. 49, 52).

Bentley, Jon Louis (1975). „Multidimensional binary search trees used for associative searching“. In: *Communications of the ACM* 18.9, S. 509–517. ISSN: 00010782. DOI: 10.1145/361002.361007 (siehe S. 10).

Beyer, Kevin u. a. (1999). „When Is “Nearest Neighbor” Meaningful?“ In: *Database Theory — ICDT'99: 7th International Conference Jerusalem, Israel, January 10–12, 1999 Proceedings*. Hrsg. von Catriel Beeri und Peter Buneman. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 217–235. ISBN: 978-3-540-49257-3. DOI: 10.1007/3-540-49257-7_15 (siehe S. 11).

Bingham, Ella und Heikki Mannila (2001). „Random Projection in Dimensionality Reduction: Applications to Image and Text Data“. In: *Proceedings of*

the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '01. San Francisco, California: ACM, S. 245–250. ISBN: 1-58113-391-X. DOI: 10.1145/502512.502546 (siehe S. 51).

Broder, Andrei Z. u. a. (1997). „Syntactic Clustering of the Web". In: *Comput. Netw. ISDN Syst.* 29.8-13, S. 1157–1166. ISSN: 0169-7552. DOI: 10.1016/S0169-7552(97)00031-7 (siehe S. 4).

Burges, Christopher J. C. (2009). „Dimension reduction: A guided tour". In: *Machine Learning* 2.4, S. 275–365. ISSN: 1935-8237. DOI: 10.1561/2200000002 (siehe S. 12, 21, 23).

Camastra, Francesco (2003). „Data dimensionality estimation methods: a survey". In: *Pattern Recognition* 36.12, S. 2945–2954. ISSN: 0031-3203. DOI: http://dx.doi.org/10.1016/S0031-3203(03)00176-6 (siehe S. 12).

Charikar, Moses S. (2002). „Similarity estimation techniques from rounding algorithms". In: *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing - STOC '02*. New York, NY, USA: ACM Press, S. 380–388. ISBN: 1581134959. DOI: 10.1145/509907.509965 (siehe S. 24, 27, 30–33).

Chávez, Edgar u. a. (2001). „Searching in metric spaces". In: *ACM Computing Surveys* 33.3, S. 273–321. ISSN: 03600300. DOI: 10.1145/502807.502808 (siehe S. 9, 10).

Chazelle, B. M. und D. T. Lee (1986). „On a circle placement problem". In: *Computing* 36.1, S. 1–16. ISSN: 1436-5057. DOI: 10.1007/BF02238188 (siehe S. 49).

Datar, Mayur u. a. (2004). „Locality-Sensitive Hashing Scheme Based on P-stable Distributions". In: *Proceedings of the twentieth annual symposium on Computational geometry - SCG '04*. ACM Press, S. 253. ISBN: 1581138857. DOI: 10.1145/997817.997857 (siehe S. 24, 27, 30–33).

Fawcett, Tom (2006). „An Introduction to ROC Analysis". In: *Pattern Recogn. Lett.* 27.8, S. 861–874. ISSN: 0167-8655. DOI: 10.1016/j.patrec.2005.10.010 (siehe S. 16).

Figueiredo, Celina M. H. de und Guilherme D. da Fonseca (2009). „Enclosing Weighted Points with an Almost-unit Ball". In: *Inf. Process. Lett.* 109.21-22, S. 1216–1221. ISSN: 0020-0190. DOI: 10.1016/j.ipl.2009.09.001 (siehe S. 49).

Gong, Yunchao u. a. (2013). „Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Bd. 35. 12. IEEE, S. 2916–2929. ISBN: 9781457703942. DOI: 10.1109/TPAMI.2012.193 (siehe S. 30–33).

Grauman, Kristen und Rob Fergus (2013). „Learning binary hash codes for large-scale image search". In: *Studies in Computational Intelligence*. Bd. 411.

Springer, S. 49–87. ISBN: 9783642286605. DOI: 10.1007/978-3-642-28661-2-3 (siehe S. 35).

Har-Peled, S. (2001). „A replacement for Voronoi diagrams of near linear size“. In: *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, S. 94–103. DOI: 10.1109/SFCS.2001.959884 (siehe S. 14).

He, Junfeng, Shih Fu Chang u. a. (2011). „Compact hashing with joint optimization of search accuracy and time“. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, S. 753–760. ISBN: 9781457703942. DOI: 10.1109/CVPR.2011.5995518 (siehe S. 30–33).

He, Junfeng, Sanjiv Kumar und Shih-Fu Chang (2012). „On the Difficulty of Nearest Neighbor Search“. In: *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, S. 1127–1134 (siehe S. 12).

He, Junfeng, Wei Liu und Shih-Fu Chang (2010). „Scalable Similarity Search with Optimized Kernel Hashing“. In: *Kdd 2010*. ACM, S. 1129–1138. ISBN: 9781450300551. DOI: 10.1145/1835804.1835946 (siehe S. 25, 30–33).

Heo, Jae Pil u. a. (2012). „Spherical hashing“. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, S. 2957–2964. ISBN: 9781467312264. DOI: 10.1109/CVPR.2012.6248024 (siehe S. 30–33).

Hwang, Chii-Ruey (1988). „Simulated annealing: Theory and applications“. In: *Acta Applicandae Mathematica* 12.1, S. 108–111. ISSN: 1572-9036. DOI: 10.1007/BF00047572 (siehe S. 49).

Indyk, Piotr und Rajeev Motwani (1998). „Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality“. In: *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*. Bd. 8, S. 604–613 (siehe S. 2, 14, 16, 17, 19, 30).

Jegou, Herve, Matthijs Douze und Cordelia Schmid (2011). „Product Quantization for Nearest Neighbor Search“. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on 33.1*, S. 117–128. DOI: 10.1109/CVPR.2013.379 (siehe S. 26, 30–33).

Jégou, Herve u. a. (2008). „Query-Adaptative Locality Sensitive Hashing“. In: *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on. IEEE* (siehe S. 30–33).

Jin, Z. u. a. (2013). „Complementary Projection Hashing“. In: *2013 IEEE International Conference on Computer Vision*, S. 257–264. DOI: 10.1109/ICCV.2013.39 (siehe S. 50).

Johnson, William B., Joram Lindenstrauss und Gideon Schechtman (1986). „Extensions of lipschitz maps into Banach spaces“. In: *Israel Journal of Mathematics* 54.2, S. 129–138. ISSN: 1565-8511. DOI: 10.1007/BF02764938 (siehe S. 23).

Joly, Alexis und Olivier Buisson (2011). „Random maximum margin hashing“. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, S. 873–880. ISBN: 9781457703942. DOI: 10.1109/CVPR.2011.5995709 (siehe S. 24, 30–33).

Knuth, Donald E. (1998). *The Art of Computer Programming, Volume 3: (2Nd Ed.) Sorting and Searching*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc. ISBN: 0-201-89685-0 (siehe S. 15).

Kulis, Brian und Trevor Darrell (2009). „Learning to hash with binary reconstructive embeddings“. In: *Nips*, S. 1–9. ISBN: 9781615679119 (siehe S. 25).

Kulis, Brian und Kristen Grauman (2009). „Kernelized locality-sensitive hashing for scalable image search“. In: *Proceedings of the IEEE International Conference on Computer Vision* Iccv, S. 2130–2137. DOI: 10.1109/ICCV.2009.5459466 (siehe S. 24, 30–33).

Lang, Ken (1995). „Newsweeder: Learning to filter netnews“. In: *Proceedings of the Twelfth International Conference on Machine Learning*, S. 331–339 (siehe S. 51).

Levandowsky, Michael und David Winter (1971). „Distance between sets“. In: *Nature* 234.5323, S. 34–35 (siehe S. 6).

Levenshtein, Vladimir I (1966). „Binary codes capable of correcting deletions, insertions, and reversals“. In: *Soviet physics doklady*. Bd. 10. 8, S. 707–710 (siehe S. 7).

Liu, Wei, Jun Wang, Rongrong Ji u. a. (2012). „Supervised hashing with kernels“. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, S. 2074–2081. ISBN: 9781467312264. DOI: 10.1109/CVPR.2012.6247912 (siehe S. 24, 30–33).

Liu, Wei, Jun Wang, Sanjiv Kumar u. a. (2011). „Hashing with Graphs“. In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, S. 1–8 (siehe S. 30–33).

Liu, Xianglong, Junfeng He und Bo Lang (2013). „Reciprocal Hash Tables for Nearest Neighbor Search“. In: *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*. AAAI'13. Bellevue, Washington: AAAI Press, S. 626–632 (siehe S. 50).

MacKay, David J. C. (2002). *Information Theory, Inference & Learning Algorithms*. New York, NY, USA: Cambridge University Press. ISBN: 0521642981 (siehe S. 6).

Manning, Christopher D., Prabhakar Raghavan und Hinrich Schütze (2008). *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press. ISBN: 0521865719, 9780521865715 (siehe S. 6, 9).

Masci, Jonathan u. a. (2013). „Sparse similarity-preserving hashing“. In: *arXiv preprint arXiv:1312.5479*. arXiv: 1312.5479 [cs] (siehe S. 35).

Maurer, Hermann A., Frank Kappe und Bilal Zaka (2006). „Plagiarism - A Survey“. In: *Journal of Universal Computer Science* 12 (8), S. 1050–1084 (siehe S. 4).

McCabe, D (2005). *Research Report of the Center for Academic Integrity*. `http://www.academicintegrity.org` (siehe S. 3).

Motwani, Rajeev, Assaf Naor und Rina Panigrahy (2005). „Lower bounds on Locality Sensitive Hashing“. In: *SIAM Journal on Discrete Mathematics*, S. 930–935. ISSN: 0895-4801. DOI: `10.1137/050646858`. arXiv: `0510088 [cs]` (siehe S. 18).

Navarro, Gonzalo (2001). „A Guided Tour to Approximate String Matching“. In: *ACM Comput. Surv.* 33.1, S. 31–88. ISSN: 0360-0300. DOI: `10.1145/375360.375365` (siehe S. 7).

Norouzi, Mohammad, David J Fleet und Ruslan Salakhutdinov (2012). „Hamming Distance Metric Learning“. In: *Advances in neural information processing systems*. S. 1061–1069 (siehe S. 30–33).

Orlin, James (1977). „Contentment in graph theory: Covering graphs with cliques“. In: *Indagationes Mathematicae (Proceedings)* 80.5, S. 406–424. ISSN: 1385-7258. DOI: `http://dx.doi.org/10.1016/1385-7258(77)90055-5` (siehe S. 48).

Paulevé, Loïc, Hervé Jégou und Laurent Amsaleg (2010). „Locality sensitive hashing: A comparison of hash function types and querying mechanisms“. In: *Pattern Recognition Letters* 31.11, S. 1348–1358. ISSN: 01678655. DOI: `10.1016/j.patrec.2010.04.004` (siehe S. 28).

Pedregosa, F. u. a. (2011). „Scikit-learn: Machine Learning in Python“. In: *Journal of Machine Learning Research* 12, S. 2825–2830 (siehe S. 51).

Rivest, Ronald (1992). *The MD5 message-digest algorithm*. `http://tools.ietf.org/html/rfc1321`. [Online; accessed 15-June-2016] (siehe S. 3).

Salakhutdinov, Ruslan und Geoffrey Hinton (2009). „Semantic Hashing“. In: *International Journal of Approximate Reasoning* 50.7, S. 969–978 (siehe S. 19, 24, 28, 30–33).

Samet, Hanan (1990). *The Design and Analysis of Spatial Data Structures*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0-201-50255-0 (siehe S. 10).

Shen, Fumin u. a. (2013). „Inductive hashing on manifolds“. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, S. 1562–1569. ISBN: 978-0-7695-4989-7. DOI: `10.1109/CVPR.2013.205`. arXiv: `1303.7043` (siehe S. 30–33).

Shivakumar, Narayanan und Hector Garcia-Molina (1999). „Finding Near-Replicas of Documents on the Web“. In: *The World Wide Web and Databases: International Workshop WebDB'98, Valencia, Spain, March 27-28, 1998. Selected Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg,

S. 204–212. ISBN: 978-3-540-48909-2. DOI: 10.1007/10704656_13 (siehe S. 4).

Stein, Benno (2007). „Principles of Hash-based Text Retrieval“. In: *30th International ACM Conference on Research and Development in Information Retrieval (SIGIR 07)*, S. 527–534. DOI: 10.1145/1277741.1277832 (siehe S. 17, 22, 38).

Strecha, Christoph u. a. (2012). „LDAHash: Improved matching with smaller descriptors“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.1, S. 66–78. ISSN: 01628828. DOI: 10.1109/TPAMI.2011.103 (siehe S. 24, 25, 30–33).

Wang, Jianfeng u. a. (2013). „Order Preserving Hashing for Approximate Nearest Neighbor Search“. In: *MM: ACM International Conference on Multimedia*. ACM, S. 133–142. ISBN: 9781450324045. DOI: 10.1145/2502081.2502100 (siehe S. 30–33).

Wang, Jingdong, Heng Tao Shen, Jingkuan Song u. a. (2014). „Hashing for Similarity Search: A Survey“. In: *arXiv preprint arXiv:1408.2927*. arXiv: 1408.2927 (siehe S. 23, 30, 34–36).

Wang, Jingdong, Heng Tao Shen und Ting Zhang (2014). „A Survey on Learning to Hash“. In: *arXiv preprint arXiv:1606.00185*. arXiv: 1606.00185 (siehe S. 25, 30, 34).

Wang, Jun, Sanjiv Kumar und Shih-Fu Chang (2010). „Semi-supervised hashing for scalable image retrieval“. In: *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, S. 3424–3431 (siehe S. 50).

Wang, Jun, Wei Liu u. a. (2016). „Learning to Hash for Indexing Big Data—A Survey“. In: *Proceedings of the IEEE* 104.1, S. 34–57 (siehe S. 29, 30, 34).

Weber, Roger, Hans-Jörg Schek und Stephen Blott (1998). „A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces“. In: *Proceedings of the 24rd International Conference on Very Large Data Bases*. VLDB '98. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., S. 194–205. ISBN: 1-55860-566-5 (siehe S. 11).

Weiss, Yair, Antonio Torralba und Rob Fergus (2008). „Spectral Hashing“. In: *Advances in Neural Information Processing Systems*. 1, S. 1–8. ISBN: 9781605609492. DOI: 10.1.1.141.7577 (siehe S. 19, 24, 25, 28, 30–33).

Wikipedia (2016). *Levenshtein distance — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Levenshtein_distance&oldid=720409646. [Online; accessed 15-June-2016] (siehe S. 7).

Wu, Chenxia u. a. (2013). „Semi-supervised nonlinear hashing using bootstrap sequential projection learning“. In: *IEEE Transactions on Knowledge and Data Engineering* 25.6, S. 1380–1393. ISSN: 10414347. DOI: 10.1109/TKDE.2012.76 (siehe S. 30–33).

Xu, Hao u. a. (2011). „Complementary hashing for approximate nearest neighbor search“. In: *2011 International Conference on Computer Vision*, S. 1631–1638. DOI: 10.1109/ICCV.2011.6126424 (siehe S. 50).

Yang, Haichuan u. a. (2014). „Adaptive Object Retrieval with Kernel Reconstructive Hashing“. In: *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*. IEEE, S. 1955–1962. ISBN: 978-1-4799-5118-5. DOI: 10.1109/CVPR.2014.251 (siehe S. 30–33).

Zhang, Dell u. a. (2010). „Self-Taught Hashing for Fast Similarity Search“. In: *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*. ACM, S. 18–25. ISBN: 9781605588964. DOI: 10.1145/1835449.1835455. arXiv: 1004.5370 (siehe S. 30–33).

Zhang, Lei u. a. (2013). „Topology preserving hashing for similarity search“. In: *MM, ACM International Conference on Multimedia*. ACM, S. 123–132. ISBN: 9781450324045. DOI: 10.1145/2502081.2502091 (siehe S. 24, 30–33).

Zhang, Ting, Jingdong Wang und Jingdw Microsoft Com (2014). „Composite Quantization for Approximate Nearest Neighbor Search“. In: *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*. Bd. 32, S. 838–846 (siehe S. 26, 30–33).

Zhao, Fang u. a. (2015). „Deep Semantic Ranking Based Hashing for Multi-Label Image Retrieval“. In: *arXiv preprint arXiv:1501.06272*. arXiv: 1501.06272 (siehe S. 24, 30–33).