Universität Paderborn
Fachbereich 17
Mathematik – Informatik

Diplomarbeit

# Verfahren zum Clustering von Dokumenten

Robin Sommer
Matrikelnr.: 3487484

Vorgelegt bei

Prof. Dr. F. J. Rammig
und
Prof. Dr. F. Meyer auf der Heide

Juli 2001

"I don't know what you mean by 'glory,'" Alice said.

Humpty Dumpty smiled contemptuously. "Of course you don't – till I tell you. I meant 'there's a nice knock-down argument for you!'"

"But glory doesn't mean 'a nice knock-down argument,'" Alice objected.

"When I use a word," Humpty Dumpty said, in a rather scornful tone, "it means just what I choose it to mean – neither more nor less."

"The question is," said Alice, "whether you can make words mean so many different things."

"The question is," said Humpty Dumpty, "which is to be master – that's all."

– *Lewis Carroll, "Through the Looking Glass"*

# Inhaltsverzeichnis

**1 Einleitung** **1**

**2 Grundlagen** **3**

2.1 Clustering . . . 3

2.1.1 Cluster-Hypothese . . . 3

2.1.2 Partitionierendes Clustering . . . 4

2.1.3 Hierarchisches Clustering . . . 6

2.2 Vektorraummodell . . . 7

2.2.1 Vorverarbeitung . . . 7

2.2.2 Konstruktion der Vektoren . . . 8

2.2.3 Wahl eines Distanzmaßes . . . 10

2.2.4 Dimensionsreduktion . . . 10

2.2.5 Textuelle Zusammenfassungen . . . 13

2.3 Andere Ansätze . . . 14

**3 Clustering-Verfahren** **15**

3.1 Anforderungen . . . 15

3.2 Algorithmen . . . 17

3.2.1 Simple Single Pass K-Means . . . 17

3.2.2 Spektrales Clustering . . . 20

3.2.3 DC-Bäume . . . 22

3.2.4 K-Bäume . . . . . 26

**4 Implementierung** **31**

4.1 Testumgebung . . . . . 31

4.1.1 Anforderungen . . . . . 31

4.1.2 Struktur . . . . . 32

4.1.3 Bibliothek . . . . . 34

4.1.4 Skript-Anbindung . . . . . 40

4.1.5 Programme . . . . . 40

4.2 Algorithmen . . . . . 44

4.2.1 Simple Single Pass K-Means . . . . . 45

4.2.2 Spektrales Clustering . . . . . 46

4.2.3 DC-Bäume . . . . . 48

4.2.4 K-Bäume . . . . . 49

**5 Evaluierung** **51**

5.1 Methode . . . . . 51

5.1.1 Qualitätsmaße . . . . . 53

5.1.2 Testdaten . . . . . 57

5.2 Einzelbewertung . . . . . 58

5.2.1 Simple Single Pass K-Means . . . . . 61

5.2.2 Spektrales Clustering . . . . . 65

5.2.3 DC-Bäume . . . . . 68

5.2.4 K-Bäume . . . . . 72

5.3 Vergleich und Folgerung . . . . . 76

**6 Zusammenfassung** **82**

**Literaturverzeichnis** **84**

# Tabellenverzeichnis


5.1 Themen der Testkollektion . . . . . . . . . . . . . . . . . . . . . . . . . . 59
5.2 Dimensionen der Vektoren . . . . . . . . . . . . . . . . . . . . . . . . . . 60
5.3 K-Means: Dimensionsreduktion mittels *Content Bearing Words* . . . . . 62
5.4 K-Means: Dimensionsreduktion mittels *Latent Semantic Indexing* . . . . 63
5.5 K-Means: Einzelergebnisse der ausgewählten Parametrisierung . . . . . . 63
5.6 K-Means: Wurzel der Hierarchie (Ausschnitt) . . . . . . . . . . . . . . . . 64
5.7 K-Means: Ein bißchen Philosophie (Ausschnitt) . . . . . . . . . . . . . . 64
5.8 Spektrales Clustering: Ohne vorherige Dimensionsreduktion . . . . . . . 65
5.9 Spektrales Clustering: Dimensionsreduktion mittels *CBW* . . . . . . . . 66
5.10 Spektrales Clustering: Wurzel der Hierarchie . . . . . . . . . . . . . . . . 67
5.11 Spektrales Clustering: Gut gefunden, aber schlecht bewertet (Ausschnitt) 68
5.12 DC-Bäume: Dimensionsreduktion mittels *CBW* und Cosinus-Maß . . . . 69
5.13 DC-Bäume: Degenerierter Baum bei $t_1 = 0,9$ . . . . . . . . . . . . . . . . 70
5.14 DC-Bäume: Dimensionsreduktion mittels *LSI* und Cosinus-Maß . . . . . 70
5.15 DC-Bäume: Wurzel der Hierarchie . . . . . . . . . . . . . . . . . . . . . . 71
5.16 DC-Bäume: Kategorie 2 der Wurzel (Ausschnitt) . . . . . . . . . . . . . 71
5.17 K-Bäume: Dimensionsreduktion mittels *Content Bearing Words* . . . . . 73
5.18 K-Bäume: Dimensionsreduktion mittels *Latent Semantic Indexing* . . . 74
5.19 K-Bäume: Einzelergebnisse der ausgewählten Parametrisierung . . . . . 74
5.20 K-Bäume: Wurzel der Hierarchie (Ausschnitt) . . . . . . . . . . . . . . . . 75
5.21 K-Bäume: Kategorie „Film“ (Ausschnitt) . . . . . . . . . . . . . . . . . . . 75

# Abbildungsverzeichnis


2.1 Unregelmäßig geformte Cluster . . . 5

2.2 Dendrogramm als Ergebnis eines hierarchischen Clustering-Verfahrens . 6

2.3 Singularwert-Dekomposition (nach [BDO94]) . . . 13

3.1 Spektrales Clustering mit $n = 6$ und $k = 3$ . . . 21

3.2 *DC-Baum* vor (a) und nach (b) Aufteilung eines vollen Knotens . . . 25

3.3 *K-Baum* vor (a) und nach (b) Aufteilung eines vollen Knotens . . . 28

4.1 Ablaufdiagramm der Testumgebung . . . 33

4.2 Interaktion von `ClusterTree`, `Cluster` und `ClusterP` . . . 39

4.3 Ausgabe von `cluster2html`: Innerer Knoten . . . 45

4.4 Ausgabe von `cluster2html`: Blatt . . . 46

4.5 Clustering-Verfahren eingefügt in die Testumgebung . . . 47

4.6 Abbildung *DC-Baum* (a) auf `ClusterTree` (b) . . . 48

5.1 Ausschnitt aus dem XML-Katalog des *ODPs* . . . 58

5.2 Vergleich der Qualitätswerte bei *CBW*-Reduktion . . . 76

5.3 Vergleich der Qualitätswerte bei *LSI*-Reduktion . . . 79

5.4 Laufzeiten bei *CBW*-Reduktion . . . 80

5.5 Laufzeiten bei *LSI*-Reduktion . . . 80

# 1 Einleitung

Die großen Informationsmengen, mit denen der Mensch inzwischen alltäglich umzugehen hat, machen es ihm oft schwer, die für ihn relevanten Dinge daraus zu extrahieren. Ohne weitere Vorarbeit ist es nicht nur im Internet – als der wohl größten und dabei inhaltlich gleichzeitig unstrukturiertesten Datensammlung schlechthin – kaum möglich, auch nur einen Überblick über die dargebotenen Informationen zu erhalten. Auch in anderen Bereichen wie der aktuellen Nachrichtenlage, in großen firmeninternen Datenbanken oder sonstigen Archiven zeigen sich ähnliche Probleme. Aus der Sicht der Informatik stellt sich hier die Aufgabe, die Benutzerin oder den Benutzer bei der Bewältigung dieser Datenflut möglichst weitgehend zu unterstützen. Ein möglicher Ansatz liegt dabei in einer automatisierten Vorgruppierung und Kategorisierung der Daten, um auf diese Art einen schnellen Zugang zu den im Einzelfall interessierenden Teilen zu ermöglichen.

Dieser Ansatz soll im Rahmen dieser Arbeit näher untersucht werden. Das Ziel ist die Vorstellung und Beurteilung von Verfahren, die, ausgehend von einer unstrukturierten Sammlung einer großen Zahl von Textdokumenten, versuchen, daraus eine thematische Hierarchie zu erstellen und die Dokumente an geeigneter Stelle einzugliedern. Das Ergebnis ist dann von der Form her zum Beispiel vergleichbar mit den thematischen Katalogen, die sich – manuell erstellt – in einer Vielzahl im Internet finden. Sie besitzen die Form eines Baumes, bei dem die Verfolgung eines Astes einer immer stärken thematischen Spezialisierung entspricht. Entsprechend den Kategoriebeschreibungen dieser Kataloge sollen auch die untersuchten Verfahren die Zweige der Hierarchie jeweils mit einer kurzen Zusammenfassung ausstatten, aus der sich auf den Inhalt der einsortierten Dokumente schließen läßt.

Zur automatisierten Erstellung einer solchen thematischen Hierarchie sind verschiedene Ansätze denkbar. In dieser Arbeit soll untersucht werden, inwieweit sich Verfahren zum *Clustering* für diese Aufgabe eignen. *Clustering* ist dabei erstmal ein recht allgemeiner Begriff für den Prozeß, in einer Datenmenge Gruppen von zusammengehörigen Elementen zu identifizieren. Die meisten Clustering-Algorithmen arbeiten dabei weitgehend unabhängig von der konkreten Struktur ihrer Eingabedaten und sollten sich daher, bei geeigneter Modellierung, auch für die Bearbeitung von Dokumenten eignen.

Um dies zu überprüfen werden drei in der Literatur beschriebene Verfahren ausgewählt, charakterisiert und bewertet. Aus der Kombination der dabei erkannten Vorteile von zwei dieser Verfahren ist im Rahmen der Arbeit ein neuer Clustering-Algorithmus entstanden, der zusätzlich in die Untersuchung miteinbezogen wird. Experimentell überprüft werden die Ergebnisse mit Hilfe einer realistischen Testkollektion von etwa 40,000 , aus dem thematischen Katalog des *Open Directory Projects* [Odp] stammenden Webseiten. Neben der grundsätzlichen Frage, inwieweit der vorgestellte Ansatz und die betrachteten Verfahren zu sinnvollen Ergebnissen führen, werden als zusätzliche Ziele die Eignung der Algorithmen für große Datenmengen und die Möglichkeit einer einfachen Navigation in den entstehenden Hierarchien aufgestellt.

Nach dieser Einleitung beginnt das *Kapitel 2 „Grundlagen“* mit der Vorstellung einiger Begriffe zum Thema Clustering. Anschließend wird mit dem Vektorraummodell eine mathematische Modellierung vorgestellt, die es erlaubt, Standardverfahren zum Clustering auf Textdokumente anzuwenden.

*Kapitel 3 „Clustering-Verfahren“* formuliert zunächst einige Anforderungen, die Algorithmen erfüllen müssen, wenn sie sich für die gegebene Aufgabenstellung eignen sollen. Anschließend werden drei nach diesen Kriterien aus der Literatur ausgewählte Verfahren im Detail vorgestellt. Das Kapitel schließt mit der Beschreibung einer vierten, neuen Clustering-Methode.

Da die Ergebnisse der vorgestellten Verfahren miteinander verglichen werden sollen, beginnt *Kapitel 4 „Implementierung“* zunächst mit der Konzeption und Implementierung einer Testumgebung. In diese Umgebung werden die vier Verfahren anschließend integriert.

Das *Kapitel 5 „Evaluierung“* stellt, nach der Definition einer möglichst objektiven Bewertungsbasis, zunächst die Ergebnisse der vier Clustering-Verfahren jeweils für sich vor und endet dann mit einem abschließenden Vergleich und einer Beurteilung der Resultate.

In *Kapitel 6 „Zusammenfassung“* findet sich dann nochmal ein zusammenfassender Überblick über die Ergebnisse dieser Arbeit.

# 2 Grundlagen

Der erste der beiden folgenden Abschnitte führt zunächst einige Begriffe zum Thema Clustering ein und erläutert die wesentlichen Ansätze, die in der Literatur zu diesem Thema zu finden sind. Aus dem dabei erkannten Bedarf für ein Maß, das die Ähnlichkeit von Dokumenten bestimmt, ergibt sich der zweite Abschnitt mit einer mathematischen Modellierung der Eingabedaten.

## 2.1 Clustering

Der Begriff *Clustering* wird in [JD83] recht informal als „*a type of a classification imposed on a finite set of objects*" eingeführt. Es stellt sich damit die Aufgabe, in einer Eingabemenge Gruppen von zusammengehörigen Elementen zu bestimmen. Die Definition der Zusammengehörigkeit kann dabei auf vielfältige Weise vorgenommen werden und hängt stark vom jeweiligen Einsatzgebiet ab. Clustering-Aufgaben stellen sich in vielfältigen Varianten auf so unterschiedlichen Arbeitsgebieten wie der Bildverarbeitung, der DNA-Analyse, der Marktforschung oder – dem Thema dieser Arbeit – dem *Information Retrieval*. Aus diesem Grund hat sich auch trotz langjähriger Forschung kein Standardverfahren etablieren können. Im Gegenteil läßt sich als Ergebnis festhalten, daß viele Clustering-Probleme maßgeschneiderte Lösungen benötigen [JD83].

Der Schwerpunkt dieser Arbeit liegt auf dem Clustering von Dokumenten. Gesucht sind dabei Gruppen von Dokumenten, die einander thematisch ähnlich sind. Trotz teilweise sehr unterschiedlicher Ansätze ist dafür eine gemeinsame Basis nötig, die als *Cluster-Hypothese* im folgenden zuerst vorgestellt werden soll. Anschließend folgt die Beschreibung zweier Klassen von Algorithmen, in die sich bei einer groben Einteilung alle Clustering-Verfahren einsortieren lassen.

### 2.1.1 Cluster-Hypothese

Alle Clustering-Verfahren, die Dokumente thematisch gruppieren sollen, arbeiten auf einer gemeinsamen Basis: Sie setzen die Gültigkeit der sogenannten *Cluster-Hypothese*

[vR79] voraus. Ursprünglich im Zusammenhang von Suchanfragen auf eine Dokumentensammlung formuliert, besagt sie: *Einander ähnliche Dokumente sind relevant für dieselben Anfragen und umgekehrt.* Übertragen auf das Clustering von Dokumenten postuliert diese Hypothese damit, daß zusammengehörige Dokumente einander auch auf eine gewisse Weise ähnlich sind. Dementsprechend muß ein Maß existieren, das mit dem Grad des thematischen Zusammenhangs zweier Dokumente korreliert. Ohne eine sinnvolle Art, die Ähnlichkeit zu messen, ist auch nach [JD83] kein Clustering möglich.

Aus der Gültigkeit der Hypothese folgt aber nur die Existenz eines Maßes. Sie sagt nichts darüber aus, wie es gefunden werden kann. In Abschnitt 2.2.3 wird zwar später die konkrete Wahl von möglichen Distanzmaßen beschrieben, dies ist aber jeweils nur als Versuch zu verstehen, die Ähnlichkeiten möglichst gut zu erfassen. Ob es gelingt, bleibt ebenso experimentell zu überprüfen, wie auch die Gültigkeit der Cluster-Hypothese an sich nur für eine gegebene, feste Dokumentensammlung, nicht aber allgemein festgestellt werden kann (in [vR79] findet sich eine Möglichkeit, eine konkrete Kollektion von Dokumenten daraufhin zu überprüfen). Um ein sinnvolles Clustering überhaupt zu ermöglichen, sei aber im folgenden die Gültigkeit der *Cluster-Hypothese* für alle potentiellen Eingabemengen vorausgesetzt.

### 2.1.2 Partitionierendes Clustering

Der Begriff *partitionierendes Clustering* (*partitional clustering*, manchmal auch *k-clustering*) bezeichnet nach [JD83] allgemein eine Klasse von Algorithmen, die bei Eingabe einer Datenmenge $S$ aus einem $d$-dimensionalen Vektorraum als Ergebnis disjunkte Teilmengen $S_1, \ldots, S_k \subseteq S$ liefern, so daß sich die Elemente einer Menge einander ähnlicher sind als den Elementen der anderen Mengen gegenüber. Dabei ist $k$ je nach Verfahren entweder ein fest vorgegebener Parameter oder selbst ein Ausgabewert.

Die Mitglieder einer großen Gruppe unter den partitionierenden Clustering-Methoden arbeiteten als Optimierungsverfahren. Sie definieren eine Kostenfunktion $c : \mathcal{P}(S) \to \mathbb{R}^+$ und versuchen, zu vorgegebenem $k$ die Kostensumme $\sum_{i=1}^{k} c(S_i)$ unter allen möglichen Partitionen von $S$ zu minimieren.[1] Viele dieser Optimierungsprobleme sind allerdings NP-schwer (siehe z. B. [KPR98]) und daher in der Praxis nur approximativ zu lösen.

Die möglichen Kostenmaße sind vielfältig. Eines der am häufigsten benutzten ist die Summe der quadrierten paarweisen Elementabstände innerhalb einer Menge $S_i$. Die traditionelle Approximationsmethode für dieses Problem ist der *K-Means* -Algorithmus [JD83]. Er wird in einer Variante in Abschnitt 3.2.1 vorgestellt.

Die Optimierungsverfahren besitzen typischerweise zwei Probleme:

[1] $\mathcal{P}(S)$ sei die Potenzmenge von S.

1. Konstruktionsgemäß muß die Anzahl der Teilmengen vorgegeben werden. Je nach Anwendungsbereich mag aber die Anzahl an sinnvollen Clustern eine Information sein, die gerade als *Ergebnis* erwartet wird.
2. Viele dieser Algorithmen tendieren dazu, regelmäßig geformte (etwa sphärische) Teilmengen zu bevorzugen [Eve93]. Dies müssen aber nicht immer die besten sein (siehe Abbildung 2.1).

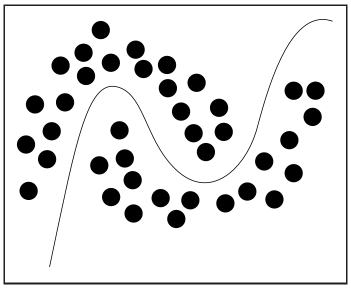


Abbildung 2.1: Unregelmäßig geformte Cluster

Neben der großen Gruppe der Optimierungsalgorithmen gibt es eine Reihe weiterer partitionierender Ansätze in jeweils diversen Varianten. Darunter sind etwa (zum Teil nach [JD83])

*Clustering mit statistischen Ansätzen:*
: Dem Clustering-Prozeß wird die Annahme zugrunde gelegt, daß alle Elemente aus $k$ verschiedenen Populationen mit bekannten Wahrscheinlichkeitsverteilungen stammen. Zu jedem Element läßt sich dann die für es wahrscheinlichste Ursprungspopulation ermitteln.

*Graphenbasiertes Clustering:*
: Aus der Eingabemenge wird ein Graph konstruiert, in dem die Elemente von $S$ gerade die Knoten darstellen und die Kanten Relationen zwischen ihnen repräsentieren. Im einfachsten Fall kann dies der vollständige Graph sein, bei dem jeder Knoten mit jedem anderen verbunden ist. Die Kantengewichte sind dann etwa die Distanzen zwischen den zu einem Knotenpaar gehörenden Elementen. Mittels typischer Graphstrukturen wie etwa minimaler Spannbäume lassen sich dann Partitionen bestimmen.

*Clustering mittels nächster Nachbarn:*
: Dieses geht davon aus, daß zwei Elemente ähnlich sind, wenn sie dieselben Nachbarn besitzen. Dementsprechend wird die Partition iterativ mit Algorithmen zur Suche von nächsten Nachbarn erstellt.

*Spektrales Clustering:*
: Der Clustering-Prozeß projiziert die Elemente zunächst in einen anderen Vektor-

raum mit speziellen Eigenschaften und ermittelt dann dort eine Partition. Dies wird in Abschnitt 3.2.2 genauer vorgestellt.

Diese Einteilung ist nur beispielhaft und sicher nicht eindeutig. Die Grenze zwischen den Verfahren ist teilweise fließend.

### 2.1.3 Hierarchisches Clustering

Neben den partitionierenden Algorithmen gibt es als zweite Klasse die hierarchischen Verfahren [JD83]. Im Gegensatz zu disjunkten Ergebnismengen erzeugen sie eine Hierarchie von Mengen (*Dendrogramm*). Darin steht die Gesamtmenge $S$ an höchster Stelle und wird dann rekursiv auf mehrere (oft zwei) Untermengen aufgeteilt bis schließlich auf unterster Ebene nur noch einelementige Mengen übrigbleiben (siehe Abbildung 2.2). Auf diese Art ist jedes Element der Eingabemenge schließlich ein Teil mehrerer ineinander geschachtelter Gruppen. Folgt man der Hierarchie von oben nach unten entspricht dies einer immer stärkeren Spezialisierung der Mengen.

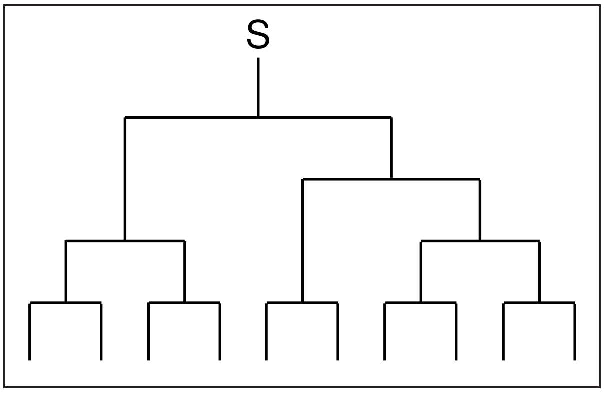



Abbildung 2.2: Dendrogramm als Ergebnis eines hierarchischen Clustering-Verfahrens

Die hierarchischen Algorithmen lassen sich entsprechend ihrer Arbeitsweise weiter unterteilen:

*Teilende Verfahren (divisive):*
Ausgehend von der Gesamtmenge wird das Dendrogramm von oben nach unten aufgebaut (*top-down*). Die Mengen einer Stufe werden jeweils auf Teilmengen aufgeteilt, die dann die nächste Ebene bilden.

Aus den in Kapitel 2.1.2 beschriebenen partitionierenden Algorithmen lassen sich derart teilende Verfahren konstruieren, indem der Ursprungsalgorithmus seine Ergebnisse rekursiv verfeinert.

*Agglomerierende Verfahren (agglomerative):*
Ausgehend von Mengen, die jeweils nur ein einzelnes Element enthalten, wird das Dendrogramm von unten nach oben aufgebaut (*bottom-up*). Jeweils zwei oder mehr Mengen werden zu einer neuen Obermenge zusammengefaßt.

Bei diesen Verfahren gibt es eine Reihe von Varianten, die sich im wesentlichen durch die Auswahl der zu kombinierenden Mengen unterscheiden. Eine Übersicht findet sich z. B. in [Eve93].

*Inkrementelle Verfahren:*
Bei diesen Methoden wird das Dendrogramm als ein Baum interpretiert, der iterativ aufgebaut wird. Jedes Element wird an geeigneter Stelle des ggf. zu vergrößernden Baums eingefügt. Die inkrementellen Verfahren werden in der traditionellen Einteilung der hierarchischen Methoden in der Regel nicht erwähnt; die Algorithmen sind eher neuerer Natur. Beispiele hierfür sind z. B. *BIRCH* [ZRL96] und ein auf *DC-Bäume* [WA00] aufsetzender Algorithmus. Letzterer wird in Abschnitt 3.2.3 vorgestellt.

## 2.2 Vektorraummodell

Wie bei der Betrachtung der Cluster-Hypothese in Abschnitt 2.1.1 festgestellt, benötigt jedes Clustering-Verfahren ein *Distanzmaß*, mit dem sich thematisch zusammengehörige Dokumente erkennen lassen. Auf dem Gebiet des *Information Retrievals* ist die gebräuchlichste Methode, die Ähnlichkeit zweier Dokumente numerisch zu bewerten, ihre Einbettung in einen hochdimensionalen und mit einer Distanzfunktion ausgestatteten Vektorraum [BB99]. Dieses Verfahren soll auch in dieser Arbeit benutzt und daher hier anschließend vorgestellt werden. Dabei sind vor allem drei Dinge zu klären:

1. Welche Teile eines Dokuments sind überhaupt von Interesse?
2. Wie ensteht aus diesen Teilen ein Vektor?
3. Welches Maß beschreibt sinnvoll die Entfernung zweier solcher Vektoren?

### 2.2.1 Vorverarbeitung

Nicht alle Bestandteile eines Dokumentes eignen sich gleichermaßen zur Charakterisierung des Inhalts. Daher extrahieren drei Vorbereitungsschritte zunächst eine Liste von Wörtern, die dann als Basis für die weitere Bearbeitung dienen. Die Schritte sind im einzelnen:

1. Die Entfernung von *Steuerinformationen*: Die Dokumente enthalten in ihrer ursprünglichen Fassung in der Regel viele zusätzliche Bestandteile, die sich nicht auf den Inhalt beziehen. Bei HTML-Seiten helfen beispielsweise weder Hintergrundfarbe noch Navigationselemente bei der thematischen Einordnung. Dementsprechend werden HTML-Auszeichnungen, enthaltende Skripte und ähnliche Dinge entfernt.

2. Die Elimination von *Stoppwörtern*: Wörter, von denen im voraus bekannt ist, daß sie häufig ohne sinntragende Bedeutung vorkommen, lassen sich in einer von Hand gepflegten Liste zusammenstellen und können dann für die weitere Bearbeitung ignoriert werden. Zu diesen Wörtern gehören im Englischen etwa der Artikel `the` oder Präpositionen wie `in` und `on`.

3. Das *Stemming*: Verschiedene Wörter mit gleichem Wortstamm besitzen oft verwandte Bedeutungen. *Stemming*-Algorithmen fassen daher solche Wörter zu Äquivalenzgruppen zusammen. Die Mitglieder einer Gruppe sind damit gleichwertig, und der Wortstamm dient als Vertreter der ganzen Gruppe.

   Der bekannteste *Stemming*-Algorithmus für englischsprachige Texte ist in [Por80] beschrieben. Er faßt beispielsweise `connected`, `connecting` und `connection` zu `connect` zusammen (aber leider auch `universe` und `university` zu `univers`!). Ein Vergleich verschiedener Algorithmen findet sich in [FZ98].

Zur Terminologie: Die Vorverarbeitung liefert als Ergebnis pro Dokument eine Liste von Wortstämmen. Sofern nicht anders erwähnt, sei mit „Dokument“ nachfolgend nun immer diese Wortliste und mit „Wort“ immer jeweils einer dieser Wortstämme bezeichnet.

### 2.2.2 Konstruktion der Vektoren

Das Ziel liegt darin, alle $n$ Dokumente in einen geeigneten Vektorraum einzubetten. In der Literatur findet sich im wesentlichen nur ein einziger Kandidat für diesen Raum: Sei $W$ die Menge aller in mindestens einem Dokument der Eingabemenge vorkommenden Wörter und $d$ die Kardinalität dieser Menge. Wähle dann als Vektorraum $\mathbb{R}^d$.

Als mögliche Interpretation definiert damit jedes Wort $w_i \in W$ $(i \in \{0, \ldots, \text{d-1}\})$ genau eine Dimension des Raums. Dementsprechend liegt die Konstruktion eines Dokumentvektors $v$ nahe: Die $i$-te Komponente von $v$ entspricht gerade der Häufigkeit, mit der das Wort $w_i$ in dem Dokument vorkommt; genannt *lokale Termfrequenz* (*local term frequency*).

Konstruiert man daraus eine Matrix, die die so entstehenden Dokumentvektoren als Spalten enthält, bekommt man eine $d \times n$ *Term-/Dokumentmatrix* . In dieser Matrix entspricht dann eine Zeile $i$ gerade dem Wort $w_i$ und läßt an den von Null verschiedenen Komponenten leicht alle Dokumente erkennen, in denen das Wort mindestens einmal vorkommt. Die Zahl dieser Dokumente ist die *Dokumentfrequenz* $f_i$ (*document frequency*) des Wortes. Ferner entspricht die Summe $g_i$ aller Werte einer Zeile gerade der Anzahl der Vorkommen des Wortes in der Gesamtmenge aller Dokumente und heißt *globale Termfrequenz* (*global term frequency*).

Die Dokumentvektoren können bereits in der bisher beschriebenen Form als Eingabe für das Clustering benutzt werden. Allerdings besteht dabei die Gefahr, daß einzelne sehr häufig in einem Dokument vorkommenden Wörter ein zu großes Gewicht bekommen. Um diesen Effekt zu vermeiden, ist es üblich, eine von mehreren möglichen Gewichtungsfunktionen zu benutzen. Eine Übersicht über eine Reihe solcher Funktionen und ihrer Kombinationsmöglichkeiten findet sich in [BB99]. Zwei später wieder aufgegriffene Beispiele seien hier angeführt:

*Binäre Gewichtung:*
Statt der lokalen Termfrequenz enthält jede Vektorkomponente nur noch 0 oder 1 als Indikator, ob ein Dokument das entsprechende Wort enthält oder nicht. Dies ist eine *lokale* Gewichtungsfunktion, da sie unabhängig von den restlichen Dokumenten arbeitet.

*TFIDF:*
Ist $f_i$ die Dokumentfrequenz des Wortes $w_i$, so wird jede Vektorkomponente $v_i$ multipliziert mit $log\ \frac{n}{f_i}$ („*term frequency times inverse document frequency*“, daher *TFIDF*). Dies ist eine *globale* Gewichtungsfunktion, da sie die Verteilung der Wörter in der Gesamtkollektion miteinbezieht. Trotz aufwendigerer Berechnung ist dies eine der am häufigsten eingesetzen Funktionen [BB99].

Die Komponenten der gefilterten Vektoren sind zwar im allgemeinen nicht mehr ganzzahlig, aber weiterhin nicht negativ.

Zu einer Menge $S \subseteq \mathbb{R}^d$ von Dokumentvektoren läßt sich ferner ein *Zentroid* $\bar{s}$ als Mittelwert definieren:

$$\bar{s} := \frac{1}{|S|} \sum_{s \in S} s$$

Dieser Zentroid dient oft als Stellvertreter für die Menge $S$ und heißt daher auch *repräsentativer Vektor*. Er wird für verschiedene Zwecke verwendet werden.

### 2.2.3 Wahl eines Distanzmaßes

Nachdem jedem Dokument eindeutig ein Vektor in $\mathbb{R}^d$ zugeordnet ist, stellt sich noch die Frage nach einem Distanzmaß. Dieses sei formal definiert als eine Funktion $\delta : \mathbb{R}^d \times \mathbb{R}^d \rightarrow [0,1]$ mit der Interpretation, daß kleinere Funktionswerte größerer Ähnlichkeit entsprechen. Damit werden zwei Dokumentvektoren $x$ und $y$ mit $\delta(x,y) = 0$ als identisch betrachtet.

Für die Wahl eines Distanzmaßes bieten sich eine Reihe von Kandidaten an. In [SGM00] werden verschiedene miteinander verglichen. Demnach ist das häufig verwendete *Cosinus-Maß* gut geeignet, um Ähnlichkeiten zwischen Dokumenten zu erfassen. Für $v,\ w \in \mathbb{R}^d$ ist dabei definiert:

$$\delta(v,w) := 1 - \left( \frac{v \cdot w}{||v||_2\ ||w||_2} \right)$$

Damit entspricht die Distanz zweier Vektoren gerade der Größe des von ihnen eingeschlossenen Winkels.

Ebenfalls angewendet werden die *Minkowski-Metriken*[2] $L_p$ mit den Spezialfällen der *Manhattan*-Metrik ($L_1$) und der *euklidischen* Metrik ($L_2$). Die damit berechneten Abstände können mit Hilfe der Exponentialfunktion in den Wertebereich $[0,1]$ transformiert werden, um ein Distanzmaß nach obiger Definition zu bekommen [SGM00].

### 2.2.4 Dimensionsreduktion

Aus der Wahl des Vektorraums $\mathbb{R}^d$ zeigt sich sofort ein – aus praktischer Sicht – großes Problem: Die Dimension des Raums und damit die Länge der Vektoren ist sehr hoch und bei variabler Dokumentenmenge sogar nach oben unbeschränkt. Dementsprechend sind für die Clustering-Algorithmen sowohl Platz- als auch Laufzeitprobleme zu erwarten („curse of dimensionality“; der Aufwand zur Berechnung des Distanzmaßes ist mindestens linear in der Anzahl an Dimensionen). Allein aus diesem Grund ist es schon geboten, die Dimension des Vektorraums möglichst noch zu reduzieren.

Ferner kann die Beschränkung auf vergleichsweise wenige Dimensionen auch die Qualität der Ergebnisse verbessern: Sicherlich eignen sich nicht alle Wörter gleichermaßen zur Erkennung von thematischen Unterschieden zwischen Dokumenten. Zwar entfernt die Stoppwort-Elimination (siehe Abschnitt 2.2.1) schon einige eher unwichtige Wörter,

[2] Sind $s, t \in \mathbb{R}^d$, so sind die Minkowski-Metriken $L_p$ definiert als $L_p(s,t) := \left( \sum_{i=0}^{d-1} |s_i - t_i|^p \right)^{1/p}$

aber bei vielen Begriffen entscheidet erst der Kontext über den Grad der Bedeutung (ein Wort wie `Web` wird in einer allgemeinen Sammlung von Webseiten keine große Aussagekraft haben; in einer zoologischen Kollektion dürfte es aber ein deutlicher Hinweis auf Spinnen sein). Wäre es also möglich, nur semantisch wichtige Teile der Vektoren zu erkennen, dürfte das deutlich bessere Ergebnisse ermöglichen.

Zwei sehr unterschiedliche Methoden zur Dimensionsreduktion sollen hier vorgestellt werden.

### Content Bearing Words (CBW)

Der erste Ansatz liegt in dem Versuch, aus der Gesamtmenge aller Wörter einige wenige, dafür aber – relativ zu der gegebenen Dokumentensammlung – inhaltstragende (*content bearing words*) zu identifizieren. Alle anderen Terme werden für die weitere Bearbeitung schlicht ignoriert, d. h. die entsprechenden Zeilen der Term-/Dokumentmatrix gestrichen.

In [BKR98] werden mehrere Maße vorgestellt, mit denen sich in einem Text die inhaltstragenden Wörter identifizieren lassen sollen. Alle arbeiten auf Basis derselben Grundthese: Bei einem Wort, das für den Inhalt von Bedeutung ist, werden die Stellen seines Auftretens häufiger in einer textuellen Nachbarschaft zueinander liegen, als dies bei nicht inhaltsrelevanten Termen der Fall ist (*Clumping-Hypothese*). Zur Messung der Stärke dieser Tendenz werden mehrere statistische Ansätze vorgestellt. Zumindest einer von diesen (*condensation clustering*) läßt sich sinnvoll auf die Situation einer Dokumentensammlung (im Gegensatz zu einem einzigen Text) übertragen und gut in die vorgestellte Modellierung eingliedern.

Jedes Wort $w_i$ besitzt, wie in Abschnitt 2.2.2 beschrieben, eine globale Termfrequenz $g_i$, die angibt, wie oft dieses Wort insgesamt in allen Dokumenten auftritt. Unter der Annahme, daß jedes dieser Auftreten unabhängig und zufällig auf die $n$ Dokumente verteilt ist, läßt sich die erwartete Anzahl $e_i$ von Dokumenten berechnen, die diesen Term enthalten:

$$e_i = n * \left(1 - \left(1 - \frac{1}{n}\right)^{g_i}\right)$$

Vergleicht man nun diesen Erwartungswert mit der ermittelten Dokumentfrequenz $f_i$, erhält man ein Maß, wie stark die tatsächliche Verteilung des Terms von einer zufälligen abweicht:

$$m_i = \frac{f_i}{e_i}$$

$m_i$ wird *Kondensationsstärke* (*condensation strength*) genannt. Aus der *Clumping*-Hypothese folgt damit die Interpretation: Um so kleiner der Wert $m_i$ ist, um so wichtiger ist das Wort für den Inhalt. Dies läßt sich nun einfach zur Dimensionsreduktion nutzen, indem nur noch Wörter mit Kondensationsstärke $m_i$ kleiner als ein vorgegebener Schwellwert $c$ beachtet werden.

Besonders betont werden soll hier, daß, soweit bekannt, diese Methode zur Identifikation der inhaltstragenden Wörter in der Literatur bisher noch nicht zur Dimensionsreduktion beim Clustering von Dokumenten benutzt worden ist. Im Rahmen dieser Arbeit muß sich daher erst noch herausstellen, ob das Verfahren für diese Anwendung praktikabel ist.

### Latent Semantic Indexing (LSI)

Ein gänzlich anderes Verfahren zur Dimensionsreduktion stellt *Latent Semantic Indexing* [DDF+90] dar. Im Gegensatz zur Identifizierung einzelner wichtiger Wörter geht es von der These aus, daß den Dokumenten eine verborgene und von der genauen Wortwahl unabhängige semantische Struktur zugrundeliegt. Durch eine Projektion in einen niedrigdimensionalen Unterraum soll diese Struktur erhalten bleiben, gleichzeitig aber das „Rauschen", das allein durch die Ausdrucksweise entsteht, eliminiert werden. Das Ziel ist ein Vektorraum, in dem thematisch ähnliche Dokumente nah beieinander liegen.

Das Hilfsmittel für diese Projektion ist die *Singularwert-Dekomposition* der Term-/Dokumentmatrix (siehe z. B. [BB99]). Jede $m \times n$ Matrix $M$ kann als Produkt dreier (im wesentlichen) eindeutiger Matrizen geschrieben werden:

$$M = U \Sigma V^T$$

Dabei ist $U$ eine orthogonale $m \times m$ Matrix, $\Sigma$ eine $m \times n$ Diagonalmatrix und $v$ eine orthogonale $n \times n$ Matrix (siehe Abbildung 2.3). Die Diagonaleinträge von $\Sigma$ sind die *Singularwerte* von $M$ und ihrer Größe nach entlang der Diagonale absteigend geordnet. Die Spalten von $U$ bilden die *linken Singularvektoren* (auch: *Termvektoren*) und die Spalten von $V$ die *rechten Singularvektoren* (auch: *Dokumentvektoren* [3]).

Aus dieser Dekomposition läßt sich eine Projektion in einen Raum der Dimension $k$ für alle $k \leq min(m, n)$ konstruieren. Sei dazu $U_k$ die aus den ersten $k$ Spalten von $U$ gebildete Matrix, $V_k$ bestehe aus den ersten $k$ Zeilen von $V^T$ und $\Sigma_k$ sei die aus den ersten $k$ Singularwerten gebildete Diagonalmatrix. Dann ist die gesuchte Projektion eines Vektors $v$ definiert durch:

[3] Um Verwechslungen mit den Eingabedokumenten zu vermeiden, wird diese Bezeichnung im folgenden nicht benutzt.

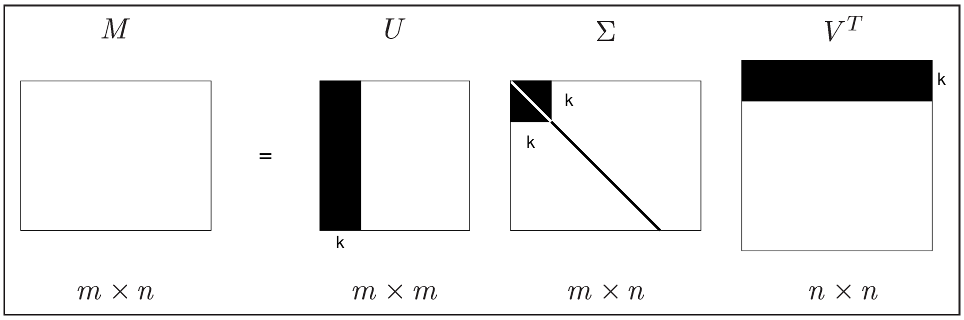



Abbildung 2.3: Singularwert-Dekomposition (nach [BDO94])

$$v' := v^T U_k \Sigma_k^{-1}$$

Die Matrix $M_k := U_k \Sigma_k V_k^T$ hat eine spezielle Eigenschaft: $M_k$ ist die beste Approximation von $M$ mit Rang $k$ bzgl. der Frobenius-Norm[4] (*Satz von Eckart und Young* [BDO94]).

In diesem Sinne ist auch die gewählte Projektion die bestmögliche.

### 2.2.5 Textuelle Zusammenfassungen

Alle Clustering-Verfahren erzeugen Mengen mit untereinander ähnlichen Dokumenten. Damit eine Anwenderin oder ein Anwender einen schnellen Überblick über das (hoffentlich vorhandene) gemeinsame Thema dieser Dokumente bekommen kann, bietet es sich an, eine kurze textuelle Zusammenfassung zu erzeugen.

Ein in [CKPT92] vorgestellter Ansatz für die Generierung einer solchen Beschreibung (*cluster digest*) bestimmt die am häufigsten in der ganzen Gruppe auftretenden Wörter. Diese stellen dann direkt die gesuchte Zusammenfassung dar. Ausgehend von den Dokumentvektoren sind diese Wörter leicht zu bestimmen: Es genügt, ihren Zentroiden zu berechnen und darin die größten Komponenten zu suchen. Auch hierbei kann sich der Einsatz von Gewichtungsfunktionen positiv bemerkbar machen, um das Gewicht einzelner Wörter ggf. etwas zu relativieren.

Besonders gute Zusammenfassungen sind auf diese Art zu erwarten, wenn die Dokumentvektoren zuvor mittels der Methode der *Content Bearing Words* in der Dimension

[4]Die Frobenius-Norm $||A||_F$ einer $m \times n$-Matrix $A = (a_{ij})$ ist definiert als

$$||A||_F := \left( \sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2 \right)^{1/2}$$

reduziert worden sind. Unter der Annahme, daß es dadurch gelingt, die inhaltstragenden Wörter zu erkennen, sollten die am häufigsten unter ihnen auftretenden Terme eine gute Charakterisierung des Inhalts leisten können.

Die beschriebene Erzeugung von Zusammenfassungen über Zentroide ist nicht möglich, wenn die Dokumentvektoren mittels *Latent Semantic Indexing* in der Dimension reduziert werden, da dann die Komponenten der entstehenden Vektoren keinen direkten Zusammenhang zu den Termen mehr haben. Als ein Ausweg läßt sich die Struktur des erzeugten Raums ausnutzen (siehe Näheres dazu in Abschnitt 3.2.2 über spektrales Clustering). Eine andere Alternative ist es, noch einmal die ursprünglichen, nichtreduzierten Dokumentvektoren zu betrachten und daraus Zentroide zu generieren.

## 2.3 Andere Ansätze

Neben dem Vektorraummodell finden sich in der Literatur nur wenige grundsätzlich andere Ansätze zum Clustering von Dokumenten. Zwei Beispiele, die versuchen, die Strukturen der natürlichen Sprache genauer zu erkennen, liefern [ZE98] und [BGMZ97]. In [ZE98] wird eine Methode vorgestellt, die mittels *Suffixbäumen* versucht, gemeinsame Phrasen innerhalb der Dokumente zu erkennen (im Gegensatz zu einzelnen gemeinsamen Wörtern). Allerdings werden die Bäume bei einer großen Zahl von Dokumenten sehr umfangreich und eignen sich daher nicht für größere Datenmengen. Dementsprechend wird dieser Ansatz in dem Artikel auch nur dafür benutzt, die Ergebnisse einer Suchmaschine thematisch zu gruppieren. In [BGMZ97] findet sich ein Clustering-Verfahren, das pro Dokument jeweils alle Folgen von aufeinanderfolgenden Wörter einer bestimmten Länge betrachtet (*shingles*) und daraus einen kurzen, charakteristischen Teil extrahiert (*sketch*), der dann zum Clustering benutzt wird.

Weitere semantische Ansätze finden sich in etwa in [Kow97]. Ferner liefert [Cha00] eine generelle Übersicht über verschiedene Formen der Erkennung von Strukturen innerhalb von Dokumenten.

# 3 Clustering-Verfahren

Das folgende Kapitel stellt die in dieser Arbeit untersuchten Algorithmen zum Clustering von Dokumenten vor. Da – wie bereits in Abschnitt 2.1 erwähnt – jedes Clusteringproblem seine eigenen Lösungen benötigt, werden zunächst die Kriterien erläutert, anhand derer drei in der Literatur zu findende Algorithmen ausgewählt worden sind. Bei diesen anschließend im Detail beschriebenen Verfahren handelt es sich um:

*Simple Single Pass K-Means:*
: Eine Variante des bereits in Abschnitt 2.1.2 erwähnten traditionellen *K-Means*.

*Spektrales Clustering:*
: Clustering mit einem dem *Latent Semantic Indexing* verwandten Ansatz.

*DC-Bäume:*
: Ein inkrementelles Verfahren.

Nach der Vorstellung dieser drei Verfahren folgt dann die Beschreibung einer neuen, im Rahmen dieser Arbeit entstandenen Methode. Sie übernimmt Teile aus dem ersten und dem dritten der genannten Algorithmen und trägt daher den Namen *K-Baum*.

## 3.1 Anforderungen

In der Literatur zum Thema Clustering gibt es eine derartige Vielzahl von unterschiedlichen Verfahren und ihrer jeweiligen Varianten, daß kaum ein umfassender Überblick gewonnen werden kann. In [JD83] heißt es etwa bezeichnenderweise allein zu der Wahl der Kostenfunktion beim partitionierenden Clustering: „*The literature of cluster analysis is spread so widely and over so many areas of science that a single criterion is rediscovered repeatedly.*“

Aus dem großen Angebot gilt es für diese Arbeit einige wenige, vielversprechende Verfahren auszuwählen. Dabei soll ein Kriterium für ihre Zusammenstellung darin liegen, möglichst unterschiedliche Ansätze zu betrachten, da nicht im voraus klar ist, welche Herangehensweise die vielversprechendste ist. Ferner sollen die einzelnen Verfahren die folgenden Kriterien erfüllen, um die zu Beginn der Arbeit genannten Ziele erreichen zu können:

*Erstellung einer Hierarchie:*
Ziel ist es, die Dokumente in einer Baumstruktur anzuordnen. Dementsprechend wird ein hierarchischer Algorithmus benötigt.

*Skalierbarkeit:*
Der Algorithmus soll auch mit großen Datenmengen zurecht kommen können. Einzelne in der Literatur zu findende Algorithmen sind nur mit wenigen hundert Dokumenten praktikabel (etwa [ZE98]). Insbesondere sollte zumindest die mittlere asymptotische Laufzeit besser als quadratisch in der Anzahl der Dokumente sein; ein Problem etwa für die große Gruppe der hierarchischen agglomerierenden Verfahren, die dieses Ziel typischerweise (konstruktionsbedingt) nicht erreichen können.

*I/O-Effizienz:*
Als Folge der Forderung nach Skalierbarkeit muß der Zugriff auf die Dokumente möglichst sequentiell erfolgen. Da große Datenmengen nicht im Hauptspeicher gehalten werden können, ist die I/O-Zeit nicht vernachlässigbar. Zumindest während der Erzeugung eines Knoten der Hierarchie muß ein Algorithmus daher auf die Dokumente in linearer Folge zugreifen (und damit insbesondere nur einmal auf jedes Dokument). Optimalerweise benötigt er für die Erzeugung der gesamten Hierarchie genau einen Durchlauf durch die Eingabemenge.

Die I/O-Effizienz soll hier allerdings im wesentlichen unter praktischen Gesichtspunkten betrachtet werden. Eine formale Analyse (siehe etwa [Vit99]) wird im Rahmen dieser Arbeit nicht durchgeführt.

*Navigierbarkeit:*
Die erzeugte Hierarchie soll eine gute Navigation möglichst direkt erlauben, damit eine zeitaufwendige Nachbearbeitung des Ergebnisses nicht nötig ist. Dazu gehört insbesondere die Generierung von deutlich mehr als zwei Kategorien pro Ebene und eine einigermaßen balancierte Hierarchie.

Vorteilhaft, wenn auch keine zwingende Voraussetzung für die Auswahl, sind zusätzlich folgende Eigenschaften:

*Generierung von Zusammenfassungen:*
Die textuellen Zusammenfassungen einer Kategorie sollten möglichst direkt während des Clustering-Prozesses erzeugt werden. Zwar ist es immer möglich, im nachhinein Zusammenfassungen zu erzeugen, aber dafür ist dann ein weiterer, zeitaufwendiger Durchlauf durch die Hierarchie und die Dokumentvektoren nötig.

*Erweiterbarkeit:*
Vorteilhaft für die praktikable Anwendung ist es, wenn neue Dokumente in eine

bestehende Hierarchie eingefügt werden können, ohne sie komplett neu aufbauen zu müssen. Optimalerweise ergibt sich auf diese Art sogar das gleiche Ergebnis, das der Algorithmus geliefert hätte, wenn die zusätzlichen Daten bereits in der Ausgangsmenge enthalten gewesen wären.

Im folgenden sollen nun vier Clustering-Verfahren vorgestellt werden, die zumindest hoffen lassen, die genannten Kriterien erfüllen zu können. Zum Teil wird allerdings erst in der Praxis zu sehen sein, inwieweit diese Hoffnung berechtigt ist.

## 3.2 Algorithmen

### 3.2.1 Simple Single Pass K-Means

Eines der bekanntesten Clustering-Verfahren ist der *K-Means*-Algorithmus. In der kanonischen Form, die im Anschluß zunächst vorgestellt werden soll, handelt es sich um ein partitionierendes Verfahren, das mehrere Iterationen über die Eingabedaten benötigt. Damit widerspricht es zweien der zuvor genannten Kriterien. In [FLE00] wird jedoch eine Variante vorgestellt, die pro Partitionierung nur einmal linear über alle Dokumente läuft und dabei mindestens vergleichbare Ergebnisse liefert. Mittels Rekursion läßt sich daraus schließlich ein hierarchisches Verfahren generieren, das allen Kriterien entspricht.

#### Kanonischer K-Means

Der klassische *K-Means*-Algorithmus [JD83] ist ein Optimierungsverfahren, das bei Eingabe von $S \subseteq \mathbb{R}^d$ und $k \in \mathbb{N}$ versucht, unter allen Partitionen $S_1, \ldots, S_k \subseteq S$ diejenige zu finden, die $\sum_{i=1}^{k} c(S_i)$ minimiert. Ist $d : \mathbb{R}^d \times \mathbb{R}^d \to [0, 1]$ ein Distanzmaß und $\bar{s_i}$ der Zentroid der Menge $S_i$, so gilt dabei:

$$c(S_i) := \sum_{u \in S_i} \delta(\bar{s_i}, u)^2$$

Dieses Optimierungsproblem entspricht einer Minimierung der Summe über die paarweisen, quadrierten Elementabstände innerhalb der Mengen.

Der *K-Means*-Algorithmus beginnt das Clustering mit heuristisch gewählten Startzentroiden und ordnet dann jeden Vektor derjenigen Menge $S_i$ zu, deren Zentroid $\bar{s_i}$ ihm am nächsten liegt. Aus dieser Aufteilung der Vektoren werden neue Zentroide berechnet, und es wird von vorne begonnen. Der Prozeß stoppt, sobald eine gewisse Konvergenz erreicht ist.

**Algorithmus 3.1** (`Kanonischer_K-Means`).

| | |
|---|---|
| EINGABE: | Vektoren $S = \{v_1, \ldots, v_n\} \subseteq \mathbb{R}^d$ <br> Partitionsgröße $k \in \mathbb{N}_{\geq 2}$ |
| AUSGABE: | Disjunkte $S_1, \ldots, S_k \subseteq S$ <br> Zentroide $\bar{s}_1, \ldots, \bar{s}_k \in \mathbb{R}^d$ |

*Wähle* $\bar{s}_i \in \mathbb{R}^d$ *für* $i \in \{1, \ldots, k\}$
*Wiederhole*
    $S_i := \emptyset$ *für alle* $i \in \{1, \ldots, k\}$
    *Für alle* $j \in \{1, \ldots, n\}$*:*
        *Bestimme* $i \in \{1, \ldots, k\}$ *mit* $\delta(\bar{s}_i, v_j)$ *minimal*
        $S_i := S_i \cup \{v_j\}$
    *Berechne neue Zentroide* $\bar{s}_i$ *für alle* $i \in \{1, \ldots, k\}$
*bis ein gewisses Stopp-Kriterium erreicht ist*

Platz für vielfältige Variationen lassen sowohl die erste als auch die letzte Zeile:

- Für die Wahl der Startzentroide $\bar{s}_i$ sind diverse Varianten vorgeschlagen worden. Möglich sind z. B. zufällig aus der Eingabemenge ausgewählte Vektoren oder Zentroide einer zufälligen Partitionierung.
- Ein notwendiges Stopp-Kriterium ist erfüllt, wenn sich bei einer Iteration keine neuen Zentroide mehr ergeben haben. Ferner kann man z. B. auch bereits abbrechen, wenn die Änderungen der Zentroide unterhalb eines gewissen Schwellwertes liegen. Zusätzlich wird die Schleife in der Regel beendet, wenn eine vorgegebene maximale Zahl von Iterationen durchlaufen worden ist. Dies garantiert zum einen die Terminierung auch dann, wenn eine Konvergenz gar nicht erreicht werden kann – was nicht garantiert ist – und begrenzt zum zweiten die Laufzeit.

In der Praxis benötigt der Algorithmus typischerweise nur einige wenige Iterationen, um eine stabile Lösung zu erreichen [JD83]; allerdings handelt es sich dabei dann im allgemeinen nur um ein *lokales* Minimum. Da der *K-Means* neben der Partitionierung der Eingabemenge auch die dazugehörigen Zentroide liefert, ist eine einfache Generierung von textuellen Zusammenfassungen möglich.

### Single Pass Variante

In [FLE00] wird nun eine Variante dieses Verfahrens vorgestellt, die auch für Eingabemengen, die nicht im Hauptspeicher gehalten werden können, praktikabel ist. Sie bearbeitet die Vektoren blockweise mit dem kanonischen *K-Means*, ohne dabei allerdings die Zentroide zwischenzeitlich neu zu initialisieren. Entstanden ist dieses Verfahren aus einer deutlichen Vereinfachung der in [BFR98] vorgestellten Methode (daher der Name „*Simple* Single Pass K-Means“).

**Algorithmus 3.2** (`Simple_Single_Pass_K-Means`).

EINGABE: Vektoren $S = \{v_1, \ldots, v_n\} \in \mathbb{R}^d$
Partitionsgröße $k \in \mathbb{N}_{\geq 2}$
Blockgröße $b \in \mathbb{N}$
Höchstzahl an Iterationen $t \in \mathbb{N}$

AUSGABE: Disjunkte $S_1, \ldots, S_k \subseteq S$
Zentroide $\bar{s_1}, \ldots, \bar{s_k} \in \mathbb{R}^d$

1 *Wähle* $\bar{s_i} \in \mathbb{R}^d$ *für* $i \in \{1, \ldots, k\}$
2 $S_i := \emptyset$ *für alle* $i \in \{1, \ldots, k\}$
3 *Solange Datenmenge noch nicht erschöpft*
4     *Lese* $b$ *neue Vektoren* $v_{j_1}, ..., v_{j_b}$
5     *Wiederhole* // Kanonischer *K-Means*
6         $W_i := \emptyset$ *für alle* $i \in \{1, \ldots, k\}$
7         *Für alle* $l \in \{1, \ldots, b\}$ :
8             *Bestimme* $i \in \{1, \ldots, k\}$ *mit* $\delta(\bar{s_i}, v_{j_l})$ *minimal*
9             $W_i := W_i \cup \{v_{j_l}\}$
10         *Berechne neue* $\bar{s_i}$ *für alle* $i \in \{1, \ldots, k\}$ // Siehe Text
11     *bis keine Änderung der Zentroide mehr oder* $t$ *Iterationen erreicht*
12     $S_i := S_i \cup W_i$

Für die Initialisierung der Zentroide in der ersten Zeile sind wieder mehrere Varianten denkbar. Ebenso ließen sich in Zeile 11 andere Konvergenzkriterien verwenden.

Etwas modifiziert erfolgt die Berechnung der neuen Zentroide in Zeile 10. Anstatt sie ausschließlich aus den den Mengen $W_i$ zugewiesenen Vektoren zu bestimmen, werden auch die Dokumentvektoren miteinbezogen, die schon in den vorherigen Durchläufen der äußeren Schleife bearbeitet worden sind. Diese sind bereits einer Menge $S_i$ zugewiesen worden. Um nun den Zentroid $\bar{s_i}$ zu berechnen, werden alle bereits in $S_i$ liegenden Vektoren aufaddiert, gemittelt und das Ergebnis als ein zusätzlicher Dokumentvektor betrachtet. Dieser geht mit der Kardinalität von $S_i$ als Gewicht in die Berechnung der neuen Zentroide ein:

$$\bar{s_i} := \frac{1}{|W_i| + |S_i|} \left( \sum_{u \in W_i} u + \sum_{v \in S_i} v \right) \quad \text{für alle } i \in \{1, \ldots, k\}$$

Die Berechnung der neuen Zentroide läßt sich technisch etwas schneller gestalten, wenn man mit jeder Menge $S_i$ jeweils ein Tupel $D_i := ( \sum_{v \in S_i} v,\ |S_i| )$ speichert. Dieses läßt sich für neu in $S_i$ aufgenommene Vektoren leicht aktualisieren. Da sich ferner die benötigten Mengenoperationen in asymptotischer Laufzeit $O(1)$ realisieren lassen (etwa mittels verketteter Listen), ergibt sich als Gesamtlaufzeit $O(n)$ – bei als konstant angenommenen Parametern $k$, $b$ und $t$ sowie fester Dimension $d$.

Wendet man diesen Algorithmus nun rekursiv auf die entstehenden Mengen $S_i$ an, ergibt sich ein hierarchisches Verfahren mit den in Abschnitt 3.1 geforderten Eigenschaften. Geht man jeweils von gleichmäßigen Partitionierungen aus, wird die Hierarchie eine logarithmische Tiefe haben. Die asymptotische Laufzeit beträgt dann $O(n\ log(n))$. Bei jeder Partitionierung werden die beteiligten Vektoren genau einmal eingelesen und bei der Implementierung läßt sich sicherstellen, daß dies in linearer Folge geschieht. Der Parameter $k$ gibt den Grad der Verzweigung innerhalb der Hierarchie an und die entstehenden Zentroide lassen sich zur Erzeugung von textuellen Zusammenfassungen verwenden. Aufgrund des rekursiven Ansatzes ist es allerdings nicht möglich, neue Vektoren nachträglich einzufügen.

### 3.2.2 Spektrales Clustering

In [KVV98] wird ein spektrales Clustering-Verfahren vorgestellt, das eng verwandt zur Dimensionsreduktion mittels *Latent Semantic Indexing* (siehe Abschnitt 2.2.4) ist. Es handelt sich dabei um ein partitionierendes Verfahren, das zum Erzeugen einer Hierarchie rekursiv angewendet werden muß.

Der Ansatz ist exakt derselbe, den *LSI* verwendet: Mittels Singularwert-Dekomposition der Term-/Dokumentmatrix wird eine Projektion in einen niedrigdimensionalen Vektorraum definiert. In der Interpretation soll dieser die semantischen Zusammenhänge der Dokumentvektoren erfassen, ohne dabei von der genauen Wortwahl abhängig zu sein. Folgt man dieser Interpretation weiter, repräsentieren die zu den $k$ größten Singularwerten gehörenden linken Singularvektoren $\bar{s_i}$ die durch die Dimensionsreduktion gefundenen semantischen *Konzepte* und seien daher hier auch *Konzeptvektoren* genannt. Die Bilder $b_i$ der Konzeptvektoren unter der Projektion bilden eine Basis des $k$-dimensionalen Raums [BB99].

Der Clustering-Prozeß erfolgt nun gemäß dieser Konzeptvektoren. Jeder Vektor $\bar{s_i}$ entspricht einer der zu erzeugenden Teilmengen $S_1, \ldots, S_k$. Ein Dokumentvektor wird zunächst in den niedrigdimensionalen Raum projiziert und der ihm – im Sinne des Cosinus-Maßes – am nächsten gelegene Basisvektor $b_i$ bestimmt. Er wird dann der Menge $S_i$

zugewiesen. In der Interpretation läßt sich sagen: Das Dokument paßt im Vergleich aller Konzepte am besten zu dem von $\bar{s_i}$ repräsentierten.

Der zu einem Dokumentvektor $v$ nächstliegende Basisvektor $b_i$ ist leicht zu finden: Ist $v'$ die euklidisch normierte Projektion des Vektors $v$, so entspricht der Wert der $j$-ten Komponente von $v'$ gerade dem Cosinus des Winkels, den $v'$ mit dem Basisvektor $b_j$ einschließt. Dementsprechend genügt es, die betragsmäßig größte Komponente in der Projektion von $v$ zu suchen (siehe Abbildung 3.1). Da die exakte Größe der Winkel nicht benötigt wird, braucht die Projektion dabei nicht wirklich normiert zu sein.

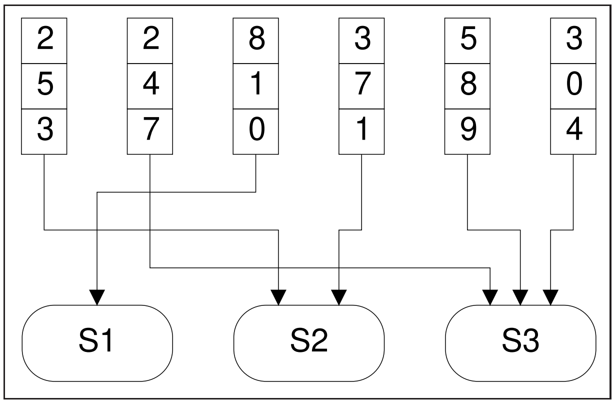



Abbildung 3.1: Spektrales Clustering mit $n = 6$ und $k = 3$

Insgesamt ergibt sich der folgende Algorithmus:

**Algorithmus 3.3** (`Spektrales Clustering`).

EINGABE: Vektoren $S = \{v_1, \ldots, v_n\} \in \mathbb{R}^d$
Partitionsgröße $k \in \mathbb{N}_{\geq 2}$

AUSGABE: Disjunkte $S_1, \ldots, S_k \subseteq S$
Konzeptvektoren $\bar{s_1}, \ldots, \bar{s_k} \in \mathbb{R}^d$

*Bilde Matrix $M$ mit Spalten $v_1, \ldots, v_n$*
*Berechne Singularwert-Dekomposition $M_k = U_k \Sigma_k V_k^T$* // Abbildung 2.3
$S_i := \emptyset$ *für alle* $i \in \{1, \ldots, k\}$
*Für alle* $v \in S$:
    *Berechne Projektion* $v' = v^T U_k \Sigma_k^{-1}$
    *Bestimme Index* $i \in \{1, \ldots, k\}$ *der betragsmäßig größten Komponente von* $v'$
    $S_i := S_i \cup \{v\}$

Sind sowohl $k$ als auch die Dimension der Vektoren konstant, benötigt die Schleife in den Zeilen $4-7$ asymptotische Laufzeit $O(n)$. Der weitaus größte Zeitbedarf liegt aber in der Berechnung der Singularwert-Dekomposition in Zeile 2. Auch wenn nur die ersten $k$ Singularwerte mit den dazugehörigen Singularvektoren benötigt werden, sind die dafür bekannten Algorithmen [Ber92] bei großen Matrizen in der Praxis sehr aufwendig.

Ein Ausweg wird in [DFK$^+$99] beschrieben: Statt die Dekomposition der gesamten Matrix zu berechnen, wird eine feste Anzahl von Spalten nach einer geeignet gewählten Wahrscheinlichkeitsverteilung zufällig ausgewählt und zu einer neuen Matrix zusammengefaßt. Die Singularwert-Dekomposition dieser Matrix läßt sich als gute Approximation der Dekomposition der Gesamtmatrix betrachten. Auf diese Art entsteht ein randomisierter Algorithmus mit asymptotischer (worst-case) Laufzeit $O(n)$. Experimente in [DFK$^+$99] zeigen auch in der Praxis sowohl in der Laufzeit als auch im Grad der Approximation sehr vielversprechende Ergebnisse. Demnach sollte der oben vorgestellte Algorithmus die geforderte Skalierbarkeitseigenschaft erfüllen können, denn auch der nur sequentielle Zugriff auf die Dokumentvektoren ist damit (weitgehend) gewährleistet. Genauer und etwas abgeschwächt: Auf die wenigen für die Singularwert-Dekomposition ausgewählten Vektoren wird zweimal zugegriffen; aufgrund der geringen Zahl sollte dies aber vertretbar sein.

Dafür erfüllt dieser Algorithmus die beiden in Abschnitt 3.1 genannten optionalen Kriterien: Zum einen kann zumindest eine gewisse Anzahl neuer Dokumentvektoren nach demselben Prinzip in eine bestehende Hierarchie eingefügt werden, wenn die Singularwert-Dekompositionen gespeichert worden sind. Zum anderen ergibt sich die Möglichkeit für textuelle Zusammenfassungen sogar ohne explizite Berechnung von Zentroiden: Die *Konzeptvektoren* stehen in der Interpretation gerade für die Gemeinsamkeiten aller ihnen zugeordneten Dokumente. Daher sollten sie auch eine Beschreibung dieser semantischen Konzepte liefern können. Sie werden dafür analog der Verwendung von Zentroiden eingesetzt, d. h. die jeweils größten Komponenten bestimmen die ausgewählten Terme.

### 3.2.3 DC-Bäume

Das Clustering mit Hilfe von *DC-Bäumen* [WA00] stellt ein inkrementelles Verfahren dar. Zu jedem Zeitpunkt existiert ein Baum, der die aktuelle Hierarchie aller schon betrachteten Dokumentvektoren darstellt. Der Clustering-Algorithmus fügt die Vektoren sukzessive den Blättern des Baums hinzu. Dafür bestimmt er jeweils zunächst die Zielposition des Vektors, fügt ihn dann dort ein und modifiziert anschließend, wenn nötig, den Baum, um die invarianten Eigenschaften eines *DC-Baums* zu erhalten.

Ein *DC-Baum* ist ein Baum mit vier Parametern: Einem Verzweigungsfaktor $b$, zwei Distanzschwellwerten $t_1, t_2 \in [0,1]$ und einem Sollwert $m$ für die minimale Anzahl von

Nachfolgern eines Knotens. Die in den Knoten gespeicherten Daten unterscheiden sich nach der Lage:

*Blätter:*
: Ein Blatt $Q$ enthält bis zu $b$ Einträge der Form $(Q_i, \bar{q_i})$. Dabei ist jeweils $Q_i$ die Menge der Dokumentvektoren, die durch dieses Blatt repräsentiert werden und $\bar{q_i}$ ihr Zentroid.

*Innere Knoten:*
: Ein innerer Knoten $P$ besitzt bis zu $b$ Einträge der Form $(P_i,\ n_i,\ \bar{p_i})$. Dabei ist jeweils $P_i$ ein Nachfolgeknoten, $n_i$ die Anzahl der Dokumentvektoren, die in einem Blatt unterhalb dieses Nachfolgeknotens liegen, und $\bar{p_i}$ der Zentroid dieser Vektoren.

Die wesentliche Operation auf diesem Baum ist das Einfügen eines neuen Dokumentvektors $v$. Um dafür die richtige Position zu finden, beginnt jeweils an der Wurzel eine rekursive Suche. An jedem inneren Knoten wird geprüft, ob Nachfolger existieren, bei denen die Distanz ihrer Zentroiden zu dem neuen Vektor kleiner als $t_1$ ist. Falls nicht, wird ein neuer Nachfolger des Knotens erzeugt: ein Blatt mit einem Eintrag der Form $(\{v\}, v)$. Falls dagegen entsprechende Nachfolger vorhanden sind, wird $v$ an den nächstgelegenen unter ihnen rekursiv weitergereicht. Wird auf diese Art ein Blatt erreicht, sucht der Einfügealgorithmus unter allen Blatteinträgen $(Q_i, \bar{q_i})$ denjenigen, dessen Zentroid dem Vektor $v$ am nächsten liegt. Falls die Distanz höchstens $t_2$ beträgt, wird $v$ zu dem Eintrag hinzugefügt. Ansonsten bekommt das Blatt einen neuen Eintrag $(\{v\}, v)$.

Damit übernehmen die beiden Schwellwerte die folgenden Funktionen:

$t_1$: Verhindert, daß ein Dokumentvektor einen Ast hinuntergereicht wird, der kaum Ähnlichkeit zu ihm besitzt. Dadurch wird das Ergebnis weniger abhängig von der Reihenfolge der Einfügeoperationen.

$t_2$: Erlaubt die direkte Kombination von sehr ähnlichen Dokumenten und spart auf diese Art Zeit und Platz – verglichen mit der alternativen Variante, in jedem Eintrag eines Blattes immer nur einen Dokumentvektor zu speichern.

Das bisher beschriebene Verfahren zum Einfügen ist noch nicht vollständig: In den Momenten, in denen ein neuer Eintrag in einem Knoten erzeugt wird, kann es passieren, daß der erlaubte Verzweigungsfaktor $b$ überschritten wird. In diesem Fall wird der volle Knoten auf zwei neue aufgeteilt (siehe Abbildung 3.2).

Lagert man die Aufteilung eines vollen Knotens zunächst in eine Funktion namens `DCTree_split` aus, sieht der Einfügealgorithmus im Detail folgendermaßen aus:

**Algorithmus 3.4** (`DCTree_insert`).

EINGABE: Vektor $v \in \mathbb{R}^d$
Knoten $P$

```
Falls P ein innerer Knoten:            // Mit r Einträgen der Form (P_i, n_i, p̄_i)
    δ := min_i{ δ(v, p̄_i) }
    Falls δ < t_1:                              // Rekursiv weiterreichen
        Bestimme Eintrag i mit δ = δ(v, p̄_i)
        DCTree_insert(v, P_i)
        Aktualisiere Eintrag i
        Falls r = b + 1:           // Der rekursive Aufruf kann r vergrößern
            DCTree_split(P)
    Sonst:                                       // Neues Blatt einfügen
        Erzeuge neues Blatt Q mit einem Eintrag ({v}, v)
        Erzeuge neuen Eintrag (Q, 1, v)                       // r := r + 1
        Falls r = b + 1:
            DCTree_split(P)
Sonst:                             // Blatt mit r Einträgen der Form (Q_i, q̄_i)
    δ := min_i{ δ(v, q̄_i) }
    Falls δ < t_2:                    // Zu bestehendem Eintrag hinzufügen
        Bestimme Eintrag i mit δ = δ(v, q̄_i)
        Q_i := Q_i ∪ {v}
        Aktualisiere q̄_i
    Sonst:                                      // Neuen Eintrag erzeugen
        Erzeuge neuen Eintrag ({v}, v)                        // r := r + 1
        Falls r = b + 1:
            DCTree_split(P)
```

Die Aufteilung eines vollen Knotens mittels `DCTree_split` geschieht nach einer Heuristik, die versucht, die $b + 1$ Nachfolger so auf zwei neue Knoten zu verteilen, daß

1. die Ähnlichkeit zwischen den Zentroiden der neuen Knoten minimiert wird,
2. die Ähnlichkeit zwischen den Zentroiden der Einträge innerhalb der beiden Knoten maximiert wird,
3. keiner der beiden weniger Nachfolger bekommt als der Wert $m$ vorgibt.

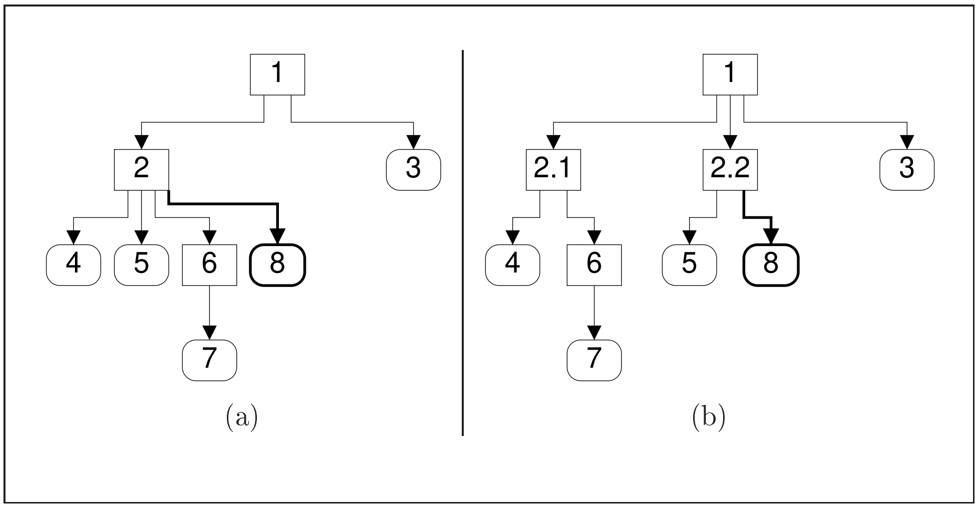



Abbildung 3.2: *DC-Baum* vor (a) und nach (b) Aufteilung eines vollen Knotens

**Algorithmus 3.5** (`DCTree_split`).

EINGABE: Knoten $P$ mit Einträgen $e_1, \ldots, e_{b+1}$ und $\bar{e_i}$ Zentroid in Eintrag $e_i$

AUSGABE: Knoten $A$ mit Einträgen $a_1, \ldots, a_{n_A}$ $(m \leq n_A \leq b)$
Knoten $B$ mit Einträgen $b_1, \ldots, a_{n_B}$ $(m \leq n_B \leq b)$

*Finde Paar* $(\bar{e_i}, \bar{e_j})$ *mit maximaler Distanz zwischen* $\bar{e_i}$ *und* $\bar{e_j}$

$seed_A := \bar{e_i}$

$seed_B := \bar{e_j}$

$E := \{e_1, \ldots, e_{b+1}\}$

*Solange* $E \neq \emptyset$

    *Falls* $A$ *oder* $B$ *weniger als* $m - |E| + 1$ *Einträge hat:*

        *Weise ganz* $E$ *dem entsprechenden Knoten zu*

        *Stoppe*

    *Bestimme* $e_i \in E$ *mit minimaler Distanz von* $\bar{e_i}$ *zu* $seed_A$ *oder* $seed_B$

    *Weise* $e_i$ *dem Knoten mit entsprechend näherem seed zu*

    $E := E \setminus \{e_i\}$

Bei der Aufteilung eines Knotens erhält sein Vorgänger einen zusätzlichen Nachfolger. Daher ist es möglich, daß nun dieser wiederum den Verzweigungsfaktor überschreitet und seinerseits geteilt werden muß (Zeile 8 in `DCTree_insert`). Der Prozeß kann sich rekursiv bis zur Wurzel fortsetzen.

Der eigentliche Clustering-Algorithmus beginnt nun mit einem nur aus einem leeren Wurzelknoten bestehenden *DC-Baum* und führt für jeden Dokumentvektor $v_i \in S$ der Eingabemenge einmal `DCTree_insert`($v_i$, `wurzel`) aus. Der entstehende Baum stellt dann direkt die gesuchte Dokumentenhierarchie dar.

Damit ist in optimaler Weise die Forderung nach I/O-Effizienz auf den Eingabedaten erfüllt: Jeder Dokumentvektor wird genau einmal betrachtet und das Einlesen kann in streng linearer Folge geschehen. Etwas problematisch ist dennoch die Beurteilung der Laufzeit, da sich im allgemeinen keine Aussage über die Form des entstehenden Baumes machen läßt. In [WA00] wird explizit betont, daß es sich bei einem *DC-Baum* nicht generell um einen höhenbalancierten Baum handelt. Damit läßt sich die asymptotische Laufzeit einer Einfügeoperation allgemein nur mit $O(n)$ angeben, wodurch der Gesamtalgorithmus quadratisch und damit für große Datenmengen unpraktikabel wird.

Andererseits hängt auch die Navigierbarkeit in der entstehenden Hierarchie stark von der Balance des Baumes ab. Ein degenerierter Baum wäre auch unabhängig von der Laufzeit ein schlechtes Ergebnis. Die Hoffnung ist daher, daß der Algorithmus in der Praxis einigermaßen balancierte Bäume erzeugt, wodurch sich der mittlere Aufwand für eine Einfügeoperation auf etwa $O(log(n))$ verringern würde. Dies bleibt experimentell zu überprüfen.

Bei der Betrachtung der beiden in Abschnitt 3.1 optionalen Kriterien zeigen sich positive Eigenschaften des *DC-Baum*-Verfahrens: Die in den Knoten gespeicherten Zentroide liefern textuelle Zusammenfassungen und neue Dokumente lassen sich ohne Probleme mit weiteren Einfügeoperationen aufnehmen.

Abschließend soll noch eine Bemerkung zur Einordnung der *DC-Bäume* gemacht werden. Die Datenstruktur wird in [WA00] speziell für das Clustering von Webseiten eingeführt. Zusammen mit einer eigenen Methode der Dimensionsreduktion dient sie in dem Artikel dazu, eine thematische *Partition* der Dokumente zu erzeugen. Da es sich bei der Datenstruktur aber um einen Baum handelt, ist dadurch gleichzeitig ein hierarchisches Verfahren gegeben. Im Gegensatz zum Originalartikel wird im Rahmen dieser Arbeit die entstehende Hierarchie als Ergebnis betrachtet. Ferner kommen zur Dimensionsreduktion nur die beiden in Abschnitt 2.2.4 beschriebenen Verfahren zum Einsatz.

### 3.2.4 K-Bäume

Im Rahmen dieser Arbeit ist eine neue Datenstruktur zum Clustering, genannt *K-Baum*, entstanden, die nun vorgestellt werden soll. Das Verfahren übernimmt Teile aus zwei der zuvor beschriebenen Methoden:

- Aus dem *DC-Baum*-Verfahren wird der inkrementelle Ansatz übernommen: Dokumentvektoren werden möglichst weit den Baum hinuntergereicht, dort eingefügt und zu volle Knoten anschließend aufgeteilt.

- Für die Aufteilung eines Knotens wird das (kanonische) *K-Means*-Verfahren benutzt.

Ein *K-Baum* hat vier Parameter: Einen Verzweigungsfaktor $b$, einen Distanzschwellwert $t$, eine Höchstzahl $m$ von Nachfolgern eines Knotens und eine maximale Anzahl von *K-Means*-Iterationen $c$. Dabei gilt $m > b$.

Ein *K-Baum* besitzt zwei Typen von Knoten: Dynamische Knoten und statische Knoten. Jeder neu erzeugte Knoten ist zunächst ein dynamischer und wird evtl. zu einem späteren Zeitpunkt in einen statischen Knoten umgewandelt. An statischen Knoten werden keine Änderungen bezüglich ihrer Position im Baum oder ihrer Nachfolger mehr vorgenommen. Es wird nicht explizit zwischen inneren Knoten und Blättern unterschieden.

Jeder Knoten P besitzt Nachfolger $P_1, \ldots, P_j$, eine Dokumentenmenge $D_P$ und einen Vektor $\bar{p}$. Dabei ist $\bar{p}$ der Zentroid der Vereinigung von $D_P$ mit allen Dokumentvektoren, die durch einen der Nachfolger repräsentiert werden. Ein dynamischer Knoten besitzt bis zu $m$ Nachfolger, ein statischer bis zu $b$.

Das Einfügen eines Dokumentvektors $v$ in einen bestehenden Baum beginnt ähnlich dem Vorgehen bei *DC-Bäumen*. Ausgehend von der Wurzel wird die Einfügestelle rekursiv bestimmt. An jedem Knoten wird dafür zunächst der Nachfolger $Q$ ermittelt, dessen Zentroid den geringsten Abstand zu dem Dokumentvektor $v$ besitzt. Ist der aktuelle Knoten ein dynamischer, wird geprüft, ob dieser Abstand unterhalb des Schwellwerts $t$ liegt. Falls ja, wird $v$ der Dokumentenmenge $D_Q$ des Nachfolgers hinzugefügt. Falls der Schwellwert überschritten ist, bekommt der aktuelle Knoten einen neuen Nachfolger, dessen Dokumentenmenge nur aus dem neuen Vektor $v$ besteht. Überschreitet dabei die Gesamtzahl der Nachfolger die Grenze $m$, wird der Knoten in einen statischen mit bis zu $b$ Nachfolgern umgewandelt (siehe Abbildung 3.3).

Trifft der rekursive Einfügealgorithmus auf einen statischen Knoten, wird der neue Vektor $v$ unabhängig von dem Schwellwert $t$ an den Nachfolger weitergereicht, dessen Zentroid ihm am nächsten liegt.

Als Algorithmus ergibt sich damit:

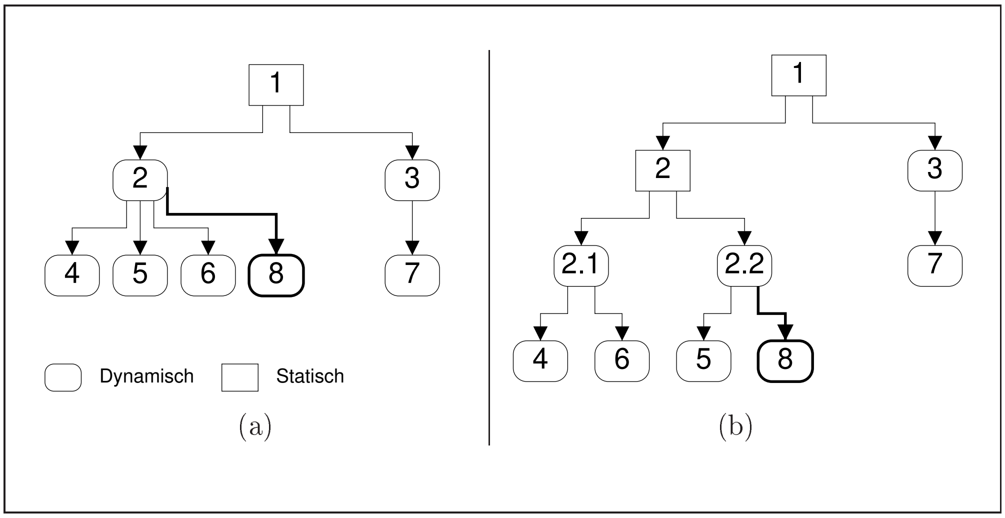



Abbildung 3.3: *K-Baum* vor (a) und nach (b) Aufteilung eines vollen Knotens

**Algorithmus 3.6 (`KTree_insert`).**

EINGABE: Vektor $v \in \mathbb{R}^d$
Knoten $P$

$\delta := min_i\{\ \delta(v, \bar{p_i})\ \}$
*Setze* $Q$ *auf Nachfolger* $P_i$ *mit* $\delta = \delta(v, \bar{p_i})$
*Falls* $P$ *ein dynamischer Knoten:*
    *Falls* $\delta < t$*:*
        $D_Q = D_Q \cup \{v\}$
        *Aktualisiere* $\bar{q}$ *und* $\bar{p}$
    *Sonst:*
        *Erzeuge neuen Nachfolger* $P_j$ *von* $P$
          *mit* $D_{P_j} := \{v\}$ *und* $\bar{p_j} := v$
        *Aktualisiere* $\bar{p}$
        *Falls* $P$ *nun* $m + 1$ *Nachfolger hat:*
            `KTree_convertNode`$(P)$
*Sonst:* // Statischer Knoten
    `KTree_insert`$(v, Q)$
    *Aktualisiere* $\bar{p}$

Die Konvertierung eines dynamischen Knotens in einen statischen geschieht mit Hilfe des kanonischen *K-Means* (siehe 3.2.1). Dazu partitioniert der Algorithmus die Zentroiden der $m+1$ Nachfolger in Teilmengen $S_1, \ldots, S_b$. Aus jeder (nicht leeren) Teilmenge $S_i$ entsteht dann ein neuer Nachfolger des Knotens, in dem jeweils alle $S_i$ zugewiesenen alten Nachfolger zusammengefaßt werden.

**Algorithmus 3.7 (`KTree_convertNode`).**

EINGABE: Dynamischer Knoten $P$ mit genau $m+1$ Nachfolgern

AUSGABE: Statischer Knoten $P$ mit höchstens $b$ Nachfolgern

$S := \{\bar{p}_i \mid i \in \{1, \ldots, m+1\}\}$ // O.B.d.A. $\bar{p}_i$ paarweise verschieden
$S_1, \ldots, S_b :=$ `Kanonischer_K-Means`$(S, b)$
*Für alle* $j \in \{1, \ldots, b\}$ *mit* $S_j \neq \emptyset$*:*
    *Erzeuge neuen Nachfolger* $P'_j$ *von* $P$
    *Für alle* $\bar{p}_i \in S_j$*:*
        *Verschiebe den Knoten* $P_i$ *so, daß er Nachfolger von* $P'_j$ *wird*
*Markiere* $P$ *als statisch*

Eine sich aus der Definition von `KTree_insert` und `KTree_convertNode` ergebene Eigenschaft der Knotentypen sei noch festgestellt: Dynamische Knoten sind immer Blätter oder direkte Vorgänger eines Blattes. Andersherum ist ein statischer Knoten niemals ein Blatt oder ein direkter Vorgänger eines Blattes.

Das auf *K-Bäumen* basierende Clustering-Verfahren arbeitet wieder in Übereinstimmung mit dem *DC-Baum*-Algorithmus: Beginnend mit einem leeren Baum fügt es sukzessive alle Dokumentvektoren ein.

Ebenfalls völlig analog zu dem *DC-Baum*-Verfahren verläuft die Überprüfung der Kriterien aus Abschnitt 3.1: Die I/O-Effizienz ist auf optimale Weise garantiert, aber die Laufzeit ist dennoch schwer zu beurteilen, da ohne weiteres keine Aussagen über die Balance des entstehenden Baumes zu treffen sind. Damit ergibt sich im schlechtesten Fall zwar wieder ein asymptotischer Aufwand von $O(n^2)$, aber die Hoffnung liegt auch hier darin, in der Praxis eine mittlere Laufzeit von $O(n\ log(n))$ zu erhalten. Ferner liefern die in den Knoten gespeicherten Zentroide textuelle Zusammenfassungen und neue Dokumente lassen sich leicht hinzufügen.

Aus der theoretischen Analyse läßt sich noch nicht abschätzen, ob eines der beiden inkrementellen Verfahren, *DC-Baum* und *K-Baum*, in der Praxis Vorteile gegenüber dem anderen bietet. Dies wird erst die Evaluierung in Kapitel 5 zeigen. Zuvor beschreibt das nächste Kapitel die Implementierung der Algorithmen innerhalb einer geeignet konzipierten Testumgebung.

# 4 Implementierung

Damit die im vorherigen Kapitel vorgestellten Algorithmen in der Praxis bewertet und miteinander verglichen werden können, sind sie in eine Testumgebung integriert worden. Deren Konzeption und Implementierung soll im folgenden zunächst vorgestellt werden. Anschließend wird erläutert, wie sich die Clustering-Verfahren darin einfügen.

## 4.1 Testumgebung

### 4.1.1 Anforderungen

Die Testumgebung soll es auf einfache Art und Weise ermöglichen, mit verschiedenen Clustering-Verfahren zu experimentieren und sie miteinander zu vergleichen. Um dies zu ermöglichen, sind die folgenden Ziele formuliert:

*Flexibles Vektorraummodell:*
Das in Abschnitt 2.2 beschriebene Vektorraummodell soll vollständig implementiert werden. Dabei übernimmt der entsprechende Teil der Implementierung den gesamten Prozeß, ausgehend von den unbearbeiteten Eingabedaten bis zu den endgültigen Dokumentvektoren, die dann schließlich als Eingabe der Clustering-Algorithmen dienen. Für das Vektorraummodell sind möglichst viele der wählbaren Parameter (etwa Distanzmaße und Art der Dimensionsreduktion) variabel und frei kombinierbar zu gestalten.

*Bereitstellung der benötigten Datenstrukturen:*
Sowohl die Testumgebung als auch die Clustering-Algorithmen benötigen eine Reihe, sich teilweise überschneidender Datenstrukturen. Diese sollen mit wohldefinierten Schnittstellen bereitgestellt werden, so daß insbesondere die Clustering-Verfahren auf einem relativ hohen Abstraktionsniveau implementiert werden können.

*Standardisierte Schnittstelle zu den Clustering-Verfahren:*
Mit der Hilfe der bereitgestellten Datenstrukturen ist eine Ein- und Ausgabeschnittstelle zu den Clustering-Verfahren zu definieren, die es erlaubt, verschiedene Verfahren direkt gegeneinander auszutauschen.

*Skript-Anbindung für „Rapid Prototyping“:*
Um ein schnelles und unkompliziertes Testen von Verfahren, Ideen und Variationen zu ermöglichen, sollen die bereitgestellten Datenstrukturen eine Anbindung an eine Skriptsprache besitzen, die die gesamte Funktionalität zur Verfügung stellt.

*Interaktive Benutzerschnittstelle:*
Die erzeugten Hierarchien sollen für die Benutzerin oder den Benutzer optisch aufbereitet werden und interaktiv zu betrachten sein.

*Objektive Qualitätsbewertung:*
Eine Hierarchie soll objektiv bewertet werden können, um verschiedene Clustering-Verfahren und Variationen der Parameter miteinander vergleichen zu können.

*Beherrschung großer Datenmengen:*
Der Umfang der Datenmengen, die bearbeitet werden können, soll nur durch die Kapazität der externen Datenträger begrenzt sein. Insbesondere dürfen die Teile der Testumgebung nicht darauf angewiesen sein, komplett im Hauptspeicher arbeiten zu müssen.

Der Schwerpunkt bei der Implementierung der Testumgebung liegt dabei deutlich auf der Flexibilität. Immer wenn es gilt, Flexibilität gegen Geschwindigkeit abzuwiegen, wird ersterer der Vorzug gegeben. Der Grund dafür ist, daß im Rahmen dieser Arbeit gerade verschiedene Ansätze und Kombinationen von Ansätzen verglichen werden sollen. Sollte sich dann eine Kombination aus Vektorraum-Parametrisierung und Clustering-Verfahren als besonders gut geeignet herausstellen, läßt sich dieser Spezialfall sicher mit vielen Optimierungen deutlich effizienter implementieren.

### 4.1.2 Struktur

Um die angesprochene Flexibilität zu erreichen, ist die Testumgebung in zwei Bereiche geteilt: eine Bibliothek der implementierten Datenstrukturen und Basisalgorithmen sowie eine Sammlung von mehreren, relativ kleinen Programmen, die möglichst frei kombinierbar sind.[1]

Das Ablaufdiagramm in Abbildung 4.1 zeigt, wie sich beide Teile zusammenfügen. Zu Beginn müssen aus den unbearbeiteten Dokumenten die Steuerinformationen entfernt werden. Für die im Rahmen dieser Arbeit betrachteten HTML-Seiten erledigt diese Aufgabe das Programm `coll2store`. Besitzen die Eingabedaten eine andere Form (etwa XML), läßt sich an dieser Stelle leicht ein anderes Programm einsetzen. Die so von Ballast befreiten Dokumente fügt `coll2store` dann in eine Instanz der Bibliotheksklasse `DocStore` ein, die dabei Stoppwörter eliminiert und das *Stemming* durchgeführt.

[1] Dies entspricht ein wenig der Unix-Philosophie „*Write programs that do one thing and do it well. Write programs to work together.*“ (McIlroy, zitiert nach [Sal94])

`DocStore` speichert die Dokumente dann als Folge von Wortstämmen. Damit ist die in Abschnitt 2.2.1 beschriebene Vorverarbeitungsphase abgeschlossen.

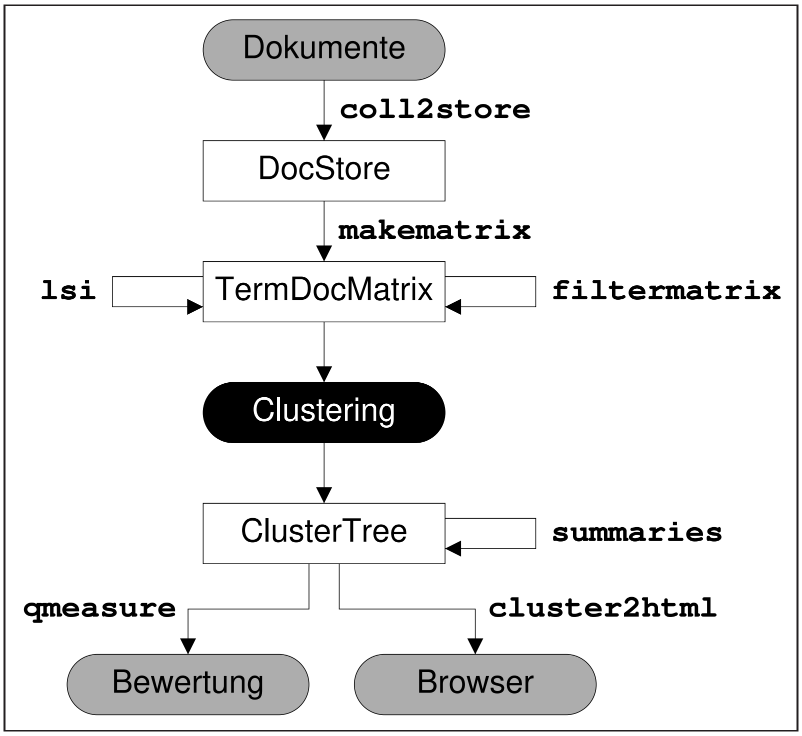



Abbildung 4.1: Ablaufdiagramm der Testumgebung

Aus dieser aufbereiteten Form der Dokumente erzeugt dann das Programm `makematrix` ungewichtete Dokumentvektoren (Abschnitt 2.2.2) und bildet damit eine Term-/Dokumentmatrix, die mittels der Klasse `TermDocMatrix` gespeichert wird.

Wahlweise bearbeiten anschließend die Programme `filtermatrix` oder `lsi` die so entstandene Matrix. Ersteres beherrscht die Gewichtung der Spalten, die Normierung nach mehreren Distanzmaßen (Abschnitt 2.2.3), die Dimensionsreduktion (Abschnitt 2.2.4) mit *Content Bearing Words* sowie die zufällige Auswahl von Dokumentvektoren. `lsi` implementiert die Dimensionsreduktion mittels *Latent Semantic Indexing*. Beide liefern ihre Ergebnisse wieder als `TermDocMatrix` und sind damit auch in Kombination anwendbar.

Die so vorbereitete `TermDocMatrix` dient nun als Eingabe für einen der Clustering-Algorithmen, der dann seine Ergebnishierarchie in der zur Verfügung gestellten Datenstruktur `ClusterTree` speichert. Solch ein Baum enthält für jeden Knoten des Hierarchiebaums die Dokumente, die in ihm liegen, einen Zentroiden sowie eine textuelle Zusammenfassung. Diese Datenfelder sind von den Clustering-Algorithmen entsprechend zu füllen. Falls ein Clustering-Verfahren keine Zusammenfassungen erzeugen kann (etwa weil die Vektoren mittels *LSI* reduziert worden sind; siehe Abschnitt 2.2.5), so hilft das Programm `summaries` weiter: Es erzeugt zu einem gegegeben `ClusterTree` mit Hilfe der *unbearbeiteten* Term-/Dokumentmatrix nachträglich textuelle Zusammenfassungen.

Ein `ClusterTree` läßt sich nun auf zwei Arten weiterverwenden: Zum einen berechnet `qmeasure` drei Qualitätsfunktionen zur Bewertung der Ergebnisse. Zum anderen bereitet `cluster2html` einen beliebigen Knoten visuell mittels einer HTML-Darstellung auf. `cluster2html` beherrscht die standardisierte `CGI`-Schnittstelle, so daß es sich direkt durch einen Webserver ausführen läßt. Damit kann eine Benutzerin oder ein Benutzer die Hierarchie interaktiv mit einem beliebigen Browser erforschen.

Die Bibliothek und die entsprechenden Programme sollen nun genauer vorgestellt werden.

### 4.1.3 Bibliothek

Als Implementierungssprache der Bibliothek ist *C++* gewählt worden, da es eine gute Kombination aus den Vorteilen der objektorientierten Programmierung und der zu erreichenden Effizienz bietet. Entwickelt und getestet ist die gesamte Testumgebung unter dem Betriebssystem *Linux* auf einer *Intel Pentium*-kompatiblen Plattform. Sie ist aber so portabel wie möglich gehalten und sollte zumindest auch auf anderen Unix-Derivaten benutzbar sein.

Bei der folgenden Beschreibung der durch die Bibliothek bereitgestellen Klassen sollen vor allem die Besonderheiten der jeweiligen Implementierung erläutert sowie ggf. dargestellt werden, inwiefern zu treffende Entscheidungen durch die in Abschnitt 4.1.1 erwähnten Ziele begründet sind.

#### DocStore

Die Klasse `DocStore` speichert alle wesentlichen Informationen zu den einzelnen Dokumenten. Neben einigen Zusatzinformationen ist dies natürlich vor allem ihr Inhalt, der als Folge von Wortstämmen gesichert wird. Da jeder in der Dokumentensammlung auftretende Wortstamm eine eindeutige Identitätsnummer (`Wort-ID`) erhält, läßt sich der Inhalt als Folge von ganzen Zahlen behandeln. Die Methode zum Einfügen eines neuen Dokumentes in eine Instanz der Klasse `DocStore` führt im einzelnen folgende Schritte durch:

1. Das Dokument bekommt eine eindeutige Identitätsnummer (`Dokument-ID`).
2. Die Stoppwörter werden entfernt.

3. Das *Stemming* wird ausgeführt. Hierfür wird Porters Algorithmus [Por80] in einer Implementierung von B. Frakes und C. Cox [FC] benutzt.

4. Die Wortstämme werden in eine Folge von `Wort-ID`s konvertiert und gespeichert.

Zu jedem Dokument werden ferner optional einige zusätzliche Dinge gesichert, die bei Bedarf später wieder abgefragt werden können:

- Ein *Thema* als Vorklassifizierung. Dies wird zur Qualitätsmessung benötigt und in Kapitel 5 näher erläutert. Die Identifizierung eines Themas erfolgt auch wieder über eine Identitätsnummer (`Themen-ID`).

- Eine *Adresse*, die die Herkunft des Dokumentes angibt (etwa eine `URL`). Die Adresse wird als Zeichenkette gespeichert.

- Eine *Zusammenfassung*, die den Inhalt kurz beschreibt. Dabei handelt es sich ebenfalls um eine Zeichenkette.

Um später wieder an die Daten eines Dokumentes zu kommen, wird nur die `Dokument-ID` benötigt. Alternativ läßt sich durch alle Dokumente iterieren.

Eine Instanz von `DocStore` bietet zusätzlich noch einige weitere Informationen:

- Eine Abbildung von `Wort-ID`s auf die Wörter selber und umgekehrt.

- Eine Abbildung von `Themen-ID`s auf eine textuelle Beschreibung der Themen.

- Die globale Termfrequenz zu jedem Wort (gebildet aus allen aktuell gespeicherten Dokumenten).

- Die Dokumentfrequenz zu jedem Wort.

`DocStore` hält die Dokumentendaten nicht komplett im Hauptspeicher, sondern schreibt sie beim Einfügen direkt auf einen externen Datenträger. Um dennoch effiziente Operationen zu ermöglichen, wird jeweils in größeren Blöcken auf die Daten zugegriffen. Die genaue Verwaltung wird dabei dem Betriebssystem überlassen: Linux bietet, wie fast alle modernen Unix-Derivate, die Möglichkeit, einen Teil des Inhalts einer Datei in den Adreßraum eines Prozesses zu blenden (*memory mapped I/O* [Ste93]). Das laufende Programm kann dann darauf mittels normaler Hauptspeicheradressierung zugreifen, und das Betriebssystem sorgt dafür, daß die Daten dort auch wirklich mit denen in der Datei übereinstimmen. Neben dieser Automatik bekommt man so auch einen sehr effizienten Zugriff: Die Daten brauchen nicht zwischen Betriebssystem- und Prozeßadressraum umkopiert zu werden wie es bei anderen Lese-/Schreibmethoden nötig wäre, und das System kann zur Steigerung der Effizienz gut Caching benutzen.

### Vector

Ein `Vector` speichert $n \in \mathbb{N}$ Fließkommazahlen und bietet als Methoden die typischen Vektor-Operationen: Neben Addition und Skalarmultiplikation sind dies vor allem die Berechnung einer Vektornorm und die Distanzmessung zu einem zweiten `Vector`. Bei Normierung und Distanzberechnung läßt sich jeweils angeben, mit welchem Maß gearbeitet werden soll (implementiert sind die $L_1$-, $L_2$- und $L_\infty$-Metriken sowie das Cosinus-Maß).

Aus Effizienzgründen arbeitet die Klasse `Vector` mit Referenzzählung [Str91], d. h. beim Kopieren eines `Vector`s werden nicht die $n$ Fließkommazahlen kopiert, sondern nur ein Zeiger darauf. Gleichzeitig wird ein interner Zähler erhöht, der anzeigt, daß es nun einen weiteren `Vector` für diese Werte gibt und diese damit nicht ohne weiteres gelöscht werden können.

### TermDocMatrix

Die Klasse `TermDocMatrix` nimmt die Term-/Dokumentmatrix auf. Sie speichert zu jeder Spalte eine `Dokument-ID`, damit nachvollziehbar bleibt, zu welchem Dokument der jeweilige Vektor gehört. Ebenso merkt sich `TermDocMatrix` zu jeder Zeile eine `Wort-ID`. Dadurch wird es möglich, textuelle Zusammenfassungen aus einer beliebig ausgewählten Teilmenge von Dokumentvektoren zu erzeugen. Eine entsprechende Methode steht zur Verfügung.

Zur eigentlichen Speicherung der Matrixdaten benutzt `TermDocMatrix` wahlweise eine von zwei Hilfsklassen: `DenseMatrix` oder `SparseMatrix`. Während erstere die Einträge der Matrix direkt Element für Element speichert (und damit bei der gewählten Genauigkeit konstant vier Bytes pro Element benötigt), benutzt `SparseMatrix` ein komprimiertes Format für dünnbesetzte Matrizen. Dabei werden nur die von Null verschiedenen Elemente sowie eine Verzeigerung zum Ermitteln ihrer Indizes gesichert (*compressed column storage format* [BB99]). Dies sorgt bei den Term-/Dokumentmatrizen für eine enorme Platzersparnis: Da die Dimension der Dokumentvektoren der Anzahl an Wörtern in der Gesamtkollektion entspricht, ein einzelnes Dokument aber typischerweise nur sehr wenige dieser Wörter auch wirklich enthält, entstehen sehr dünnbesetzte Matrizen. Eine Ausnahme sind die Matrizen, die bei der Dimensionreduktion mittels *LSI* entstehen. Diese sind dichtbesetzt und werden daher besser mit `DenseMatrix` behandelt.

Alle drei Klassen, `TermDocMatrix`, `DenseMatrix` und `SparseMatrix`, sind von einer gemeinsamen, im *C++*-Jargon *abstrakten*, Oberklasse `Matrix` abgeleitet, die die gemeinsamen Operationen definiert: Ermitteln der Dimensionen sowie Spalten lesen, modifizieren und hinzufügen. Eine Matrixspalte wird dabei durch einen `Vector` repräsentiert. Durch diese gemeinsame Schnittstelle ist es möglich, auch Matrizen zu erstellen, die die zusätzlichen Informationen der `TermDocMatrix` nicht enthalten, und mit den gleichen

Methoden darauf zuzugreifen. Andersherum erlaubt es diese Abstraktion, eine gegebene Matrix unabhängig von ihrem konkreten Typ zu lesen oder zu modifizieren.

Die Klasse `SparseMatrix` bietet zusätzlich noch eine weitere Methode, die die Matrixdaten in dem portablen *Harwell-Boeing Sparse Matrix Format* [DGL92] speichert. Damit ist bei Bedarf auch ein Datenaustausch mit anderen, kompatiblen Programmen möglich.

Sowohl `DenseMatrix` als auch `SparseMatrix` greifen wieder mit *memory mapped I/O* (siehe Beschreibung der Klasse `DocStore`) auf die Matrixdaten zu. Beschränkt man sich auf einen linearen Zugriff auf die Spalten der Matrix, so sind damit sehr effiziente Operationen möglich. Etwas problematisch dagegen sind unregelmäßige Spaltenzugriffe, da dabei für das Betriebssystem keine Möglichkeit besteht, die benötigten Daten effizient einzulesen. (Im Gegenteil: Es wird in der Regel Daten vorauslesen, die gar nicht benutzt werden.) Dieses Problem besitzt aber jede externe Datenhaltung.

### ClusterTree

Die Klasse `ClusterTree` implementiert im wesentlichen eine Baumstruktur. Sie bietet Methoden zum Einfügen, Entfernen und Verschieben von Knoten. Ein Knoten wird dabei durch eine weitere Klasse `Cluster` beschrieben, deren Instanzen die jeweils benötigten Daten speichern. Jeder Knoten bekommt zur Identifizierung eine eindeutige Nummer (`Knoten-ID`) und besitzt als weitere Attribute eine Dokumentenmenge (in Form einer Menge von `Dokument-ID`s), einen Zentroiden (als `Vector`), eine textuelle Zusammenfassung (als Zeichenkette) sowie drei Fließkommazahlen für die in Abschnitt 5.1.1 beschriebenen Qualitätsmaße. Es stehen Methoden zur Manipulation dieser Attribute in `Cluster` zur Verfügung. Ferner bietet die Klasse Navigationsmethoden für die Baumstruktur (Vorgänger-, Nachfolger- und Nachbarknoten). `ClusterTree` enthält ferner eine Methode zum direkten Zugriff auf einen Knoten anhand seiner `Knoten-ID`.

Als etwas schwierig hat sich die Speicherung eines `ClusterTree`s erwiesen. Der erste Ansatz hat den Baum komplett im Hauptspeicher gehalten. Das hat sich aber schnell als unpraktikabel erwiesen, da die Clustering-Algorithmen unter Umständen sehr degenerierte und damit große Hierarchien erzeugen können. Insbesondere wegen der Größe der in jedem Knoten gespeicherten Zentroide ist es daher nicht möglich, nur im Hauptspeicher zu arbeiten.

In einer zweiten Implementierung ist daher auf externe Speicherung zurückgegriffen worden. Auch dabei haben sich sich allerdings einige Schwierigkeiten gezeigt: Da zu jeder Zeit Knoten hinzugefügt, entfernt und verschoben werden können, ist es nicht möglich, die Daten in einer Form zu speichern, die dem Aussehen des Baums entspricht und damit zumindest bei einer bestimmten Zugriffsfolge direkt effizient arbeitet. Aus dem gleichen Grund bietet das bei der Klasse `DocStore` beschriebene *memory mapped I/O* hier keine Vorteile. Bei (aus Sicht des Betriebssystems) wahllosem Zugriff ist der

Verwaltungsaufwand höher als der Nutzen. Der Verzicht auf diese Unterstützung des Betriebssystems führt direkt zu der nächsten Schwierigkeit: Die Automatik der Lese-/Schreiboperationen fällt weg, d. h. der `ClusterTree` hat sich selbst darum zu kümmern, immer die gerade benötigten Daten im Hauptspeicher zu halten.

Um diese Probleme gut zu beherrschen und der Anwenderin oder dem Anwender der Bibliothek dennoch eine möglichst einfache Schnittstelle anbieten zu können (z. B. soll nicht sie bzw. er dafür Sorge tragen müssen, daß die richtigen Knoten geladen sind), werden zwei Dinge getan:

- Die Speicherung der Knoten geschieht mittels einer Datenbank. Die `Knoten-ID`s stellen dabei den Schlüssel dar. Die Implementierung benutzt die *Berkeley DB* [Bdb], eine im Quelltext erhältliche, für nicht-kommerzielle Anwendungen frei benutzbare Datenbankbibliothek. Diese Datenbank speichert pro Schlüssel einen variabel großen Block von beliebigen Daten. Die Attribute eines jeden Knotens werden daher in eine Binärform konvertiert und so mit der `Knoten-ID` als Schlüssel an die Bibliothek weitergereicht.

  Statt der *Berkeley DB* ist zunächst die Bibliothek *GNU Database Manager* [Gdb] verwendet worden. Dabei haben sich allerdings wiederholte, wenn auch leider kaum zu reproduzierende Abstürze gezeigt. Da nach dem Austausch der Datenbankbibliothek diese Probleme sofort verschwunden sind (beide Bibliotheken bieten eine fast identische Programmierschnittstelle), bleibt die Vermutung eines schwerwiegenden Fehlers in der aktuellen Version 1.8.0 des *GNU Database Managers*.

- Der Zugriff auf einen Knoten (in Form einer Instanz der Klasse `Cluster`) geschieht mittels des Konzepts der „intelligenten Zeiger" (*smart pointers* in [Str91]). Dabei handelt es sich hier um eine Klasse `ClusterP`, die sich für die Anwenderin oder den Anwender der Bibliothek wie ein Zeiger auf eine Instanz von `Cluster` verhält. Insbesondere kann sie oder er über die normale Zeigerdereferenzierung (in *C++* der Operator „`->`") auf sämtliche Methoden zugreifen (siehe Abbildung 4.2(a)).

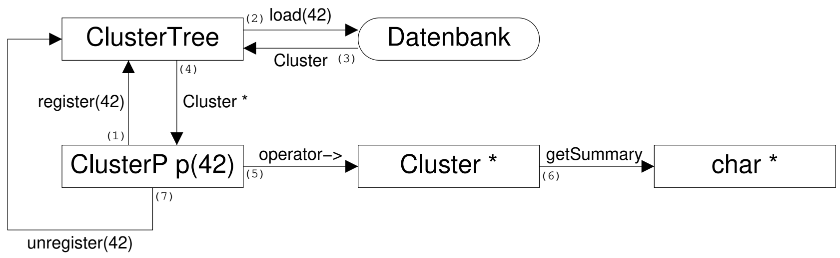
  Bei der Konstruktion einer Instanz der Klasse `ClusterP` wird als Argument die `Knoten-ID` des gewünschten Knotens angegeben. Der Konstruktor weist damit dann den `ClusterTree` an, das entsprechende `Cluster`-Objekt aus der Datenbank zu laden und merkt sich einen (echten) Zeiger auf dessen Position. Bei einer Dereferenzierung liefert `ClusterP` nun immer diesen Zeiger und ermöglicht damit, beliebige Methoden von `Cluster` aufzurufen.

  In dem Moment, in dem der `ClusterTree` für einen `ClusterP` einen Knoten aus der Datenbank lädt, merkt er sich, daß es nun einen Nutzer für diesen Knoten gibt. Dieser darf daher vorläufig nicht mehr aus dem Hauptspeicher entfernt werden. Sobald aber der `ClusterP` seinen eigenen Gültigkeitsbereich verläßt, meldet sein Destruktor dem `ClusterTree`, daß der Knoten nicht mehr benötigt wird. Dementsprechend kann er nun aus dem Speicher gelöscht werden (nachdem etwaige Änderungen der Attribute in die Datenbank übernommen worden sind). Über eine Zählung der aktiven `ClusterP` pro Knoten ist es auch möglich, sichere Zugriffe mehrerer „intelligenter Zeiger“ zu gewährleisten.

Zur Effizienzsteigerung werden Knoten, die von keinem `ClusterP` mehr referenziert werden, nicht sofort aus dem Hauptspeicher entfernt, sondern zunächst in einem Cache gespeichert. Überschreitet dieser eine gewisse vorgegebene Größe, löscht der `ClusterTree` dann zuerst die Knoten, deren letzte Referenzierung am längsten zurückliegt.

```
{
  ClusterP p(42);
  cout << p->getSummary() << endl;
}
```

(a)




(b)

Abbildung 4.2: Interaktion von `ClusterTree`, `Cluster` und `ClusterP`

Zur Verdeutlichung der Zusammenhänge zwischen den Klassen `ClusterTree`, `Cluster` und `ClusterP` ist in Abbildung 4.2(b) dargestellt, welche Schritte intern erfolgen, wenn der Programmtext aus Abbildung 4.2(a) ausgeführt wird (ohne den Caching-Mechanismus).

### 4.1.4 Skript-Anbindung

Für die geforderte Skript-Anbindung ist die Wahl auf die Sprache *Python* [Bea00] gefallen. Diese relativ neue Skriptsprache besitzt gegenüber anderen, derzeit noch weiterverbreiteteren (etwa *Perl*) den Vorteil, von Grund auf objektorientiert konzipiert zu sein. Ferner erlaubt es *Python* der Programmiererin oder dem Programmierer, mit wenig Mühe gut les- und wartbare Skripte zu erstellen und stellt dazu noch eine große Bibliothek an Zusatzmodulen zur Verfügung.

Eine Skriptanbindung per Hand für eine komplette Bibliothek zu erstellen, ist zum einen aufwendig und zum anderen sehr eintönig: aufwendig, weil für jede Bibliotheksfunktion (oder Methode einer Klasse) eine eigene Schnittstellenfunktion benötigt wird, und eintönig, weil diese sich bis auf die übergebenen Parameter kaum unterscheiden. Dazu kommt eine schwierige Wartbarkeit bei Änderungen der Bibliothek, wie sie gerade in der Entwicklungsphase nicht zu vermeiden sind. Als Ausweg ist daher eine automatische Generierung gewählt worden. Der *Simplified Wrapper and Interface Generator (SWIG)* [Swi] ist ein Programm, das aus *C/C++* Deklarationen direkt die benötigten Schnittstellenfunktionen für *Python* (und andere Skriptsprachen) erzeugen kann. Allerdings wird die letzte stabile Version 1.1 von *SWIG* derzeit zugunsten einer weitgehenden Überarbeitung nicht mehr gepflegt. Da sich allerdings die zur Verfügung stehende Entwicklungsversion 1.3 als noch nicht einsatzbereit erwiesen hat, gleichzeitig aber auch keine anderen, vergleichbaren Programme zu existieren scheinen, ist die alte Version 1.1 zum Einsatz gekommen. Diese besitzt leider ein paar Probleme: Sie beherrscht zum einen nicht die vollständige *C++*-Semantik (es fehlen etwa das Überladen von Methoden und die *Template*-Unterstützung). Dadurch sind ein paar kleinere Änderungen an den Schnittstellen nötig gewesen. Zum anderen enthalten die per *SWIG* erzeugten Programmstücke einige wenige Fehler, die nachträglich behoben werden müssen.

Trotz dieser Schwierigkeiten bietet die so erzeugte Schnittstelle die volle Funktionalität der Bibliothek und wird mittels eines kleines Shell-Skripts bei Änderungen automatisch aktualisiert. Im Ergebnis lassen sich damit die Klassen der Bibliothek aus *Python* und *C++* heraus fast identisch benutzen.

### 4.1.5 Programme

Im folgenden werden die Programme vorgestellt, die an dem in Abbildung 4.1 dargestellten Prozeß beteiligt sind. Sie setzen alle auf die zuvor vorgestellte Bibliothek auf und sind daher auf relativ hohem Abstraktionsniveau implementiert. Alle arbeiten von der Kommandozeile aus und erwarten ihre Parameter direkt beim Aufruf. Mit einer Ausnahme sind alle Programme wie die Bibliothek in *C++* kodiert.

### coll2store

Das Programm `coll2store` dient dazu, die unbearbeiteten Eingabedokumente in eine Instanz der Klasse `DocStore` zu konvertieren. Es erwartet die Dokumente in einer Datei in der Form:

```
[URL]
[Thema]
[ ... HTML-Daten ... ]
[ ... HTML-Daten ... ]
[ ... HTML-Daten ... ]
<><><>
[URL]
[Thema]
[ ... HTML-Daten ... ]
[ ... HTML-Daten ... ]
[ ... HTML-Daten ... ]
<><><>
[...]
```

`URL` ist dabei die Adresse des Dokumentes und `Thema` eine Zeichenkette, die einen thematischen Kontext beschreibt und für die Evaluierung benutzt wird (siehe Kapitel 5).

`coll2store` liest jeweils ein Dokument ein und entfernt zunächst mit Hilfe der Bibliothek *libxml* [Xml] die HTML-Auszeichnungen. Anschließend werden die übriggebliebenen Wörter an `DocStore` weitergereicht.

### makematrix

Das Programm `makematrix` erzeugt aus den Daten eines `DocStore`s eine ungewichtete Term-/Dokumentmatrix. Zunächst befragt es den `DocStore` nach der Anzahl der gespeicherten, paarweise verschiedenen Wortstämme und kennt damit die Dimension der zu erzeugenden Vektoren. Anschließend iteriert dann `makematrix` durch die Dokumente. Aus der Liste von `Wort-IDs`, die `DocStore` liefert, läßt sich leicht ein Dokumentvektor erzeugen. Dazu wird für jedes Wort seine `ID` als die Nummer der zugehörigen Vektorkomponente betrachtet. Werden zunächst alle Komponenten mit Null initialisiert, genügt es dann, für jeden Term in der Liste die entsprechende Komponente jeweils zu inkrementieren. Es ergibt sich ein ungewichteter Dokumentvektor, der mit der entsprechenden `Dokument-ID` versehen in die `TermDocMatrix` eingefügt wird.

### filtermatrix

Mit `filtermatrix` stehen verschiedene Filteroperationen für die Term-/Dokumentmatrix zur Verfügung. Als Eingabe bekommt das Programm eine Instanz von `TermDocMatrix` und dem zugehörigen `DocStore` und erzeugt damit eine neue `TermDocMatrix` als Ausgabe.

`filtermatrix` kann die Spalten der Matrix mit Hilfe der $L_1$-, $L_2$- und $L_\infty$-Metriken oder des Cosinus-Maßes normieren. Als globale Gewichtungsfunktion beherrscht es *TFIDF*, als lokale die binäre Gewichtung. Ferner ist die Dimensionsreduktion mittels *Content Bearing Words* bei beliebigem Schwellwert $c$ für die Kondensationsstärke implementiert. Zusätzlich können alle Terme (d. h. Zeilen der Matrix) entfernt werden, die in der Dokumentenkollektion in weniger bzw. mehr als einem gewissen Prozentsatz von Dokumenten auftreten. Schließlich ist es möglich, einzelne Spalten der Matrix gezielt auszuwählen. Dazu können Bereiche von Spaltennummern oder `Dokument-IDs` angegeben werden oder eine Wahrscheinlichkeit, mit der jede Spalte in die neue Matrix übernommen werden soll.

Alle diese Filter lassen sich kombinieren. Ihre Implementierung weist keine Besonderheiten auf, da alle nötigen Informationen bereits von den Bibliotheksklassen `DocStore` und `TermDocMatrix` zur Verfügung gestellt werden.

### lsi

Das Programm `lsi` implementiert die Dimensionsreduktion mittels *Latent Semantic Indexing*. Als Eingabe bekommt es eine `TermDocMatrix`, berechnet daraus die reduzierte Singularwert-Dekomposition, transformiert alle Dokumentvektoren in den neuen Vektorraum und erzeugt so eine neue `TermDocMatrix` als Ausgabe.

Für die Singularwert-Dekomposition benutzt `lsi` die Implementierung aus dem *General Text Parser (GTP)* [Gtp] von Michael W. Berry et. al.. *GTP* ist ein Programm, in dem seinerseits bereits ein großer Teil des in 2.2 beschriebenen Vektorraummodells implementiert ist. Es bekommt unbearbeitete Dokumentendaten, erzeugt eine gewichtete Term-/Dokumentmatrix und berechnet anschließend die Dekomposition. Allerdings erlaubt es nur wenige Variationen in dem genauen Ablauf und arbeitet komplett im Hauptspeicher. Daher bietet es nicht die Flexibilität der hier entwickelten Testumgebung. Übernommen werden aus *GTP* nur die Funktionen zur Berechnung der Singularwert-Dekomposition, deren mathematischer Hintergrund in [Ber92] beschrieben wird. Da *GTP* intern recht modular aufgebaut ist, läßt sich der entsprechende Teil – wenn auch mit etwas Aufwand – extrahieren. Als Eingabe erwartet der Dekompositionsalgorithmus eine Matrix im *Harwell-Boeing*-Format [DGL92] und speichert die drei entstehenden Matrizen dann in einem eigenen Binärformat. Leider ist dieses Format kaum dokumentiert. Lediglich ein bei *GTP* mitgeliefertes *Perl*-Skript zur lesbaren Ausgabe gibt etwas Aufschluß. Die aus der Analyse des Skripts entstandene Klasse `GTPParser` stellt eine Schnittstelle zur Verfügung, mit der `lsi` auf die Daten zugreifen kann.

Leider arbeitet auch die Dekompositionsroutine von *GTP* nur im Hauptspeicher, so daß sich bei zu großen Matrizen an dieser Stelle Probleme ergeben können. Dies widerspricht zwar den zu Beginn dieses Kapitels formulierten Zielen, ist aber für diese Arbeit nicht zu vermeiden gewesen. Es ist keine auf externen Daten arbeitende Implementierung bekannt, die sich hätte übernehmen lassen, und eine Neuimplementation hätte den zeitlichen Rahmen gesprengt. Bei Problemen mit mangelndem Speicher bietet sich aber wieder die in [DFK+99] vorgestellte Variante an, die Singularwert-Dekomposition nur für eine Untermatrix zu berechnen (siehe Abschnitt 3.2.2). Durch die Möglichkeit mittels `filtermatrix` einzelne Spalten zufällig auswählen zu können und einer entsprechenden Vorkehrung in `lsi` ist dieser Ausweg gegeben.

### summaries

Mit `summaries` können für einen `ClusterTree` im nachhinein textuelle Zusammenfassungen erzeugt werden (etwa wenn aufgrund einer Dimensionsreduktion per *Latent Semantic Indexing* die Clustering-Algorithmen dies nicht direkt machen konnten; siehe Abschnitt 2.2.5).

Dazu bestimmt `summaries` für jeden Knoten alle Dokumente, die in diesem Knoten oder einem seiner Nachfolger liegen, berechnet aus deren Vektorraumdarstellungen einen Zentroiden und erzeugt daraus eine Zusammenfassung. Allerdings gehen nicht alle Terme in die Betrachtung ein: Auch `summaries` implementiert noch einmal die Dimensionsreduktion mit *Content Bearing Words*. In diesem Fall geschieht das allerdings relativ zu dem Knoten: Die Kondensationsstärke $m_i$ wird ausschließlich aus den betroffenen Dokumenten bestimmt. Ein vorzugebender Schwellwert $c$ gibt an, welche Terme bei der Generierung der Zusammenfassung betrachtet werden sollen.

Um seine Aufgabe ausführen zu können, benötigt `summaries` die ursprüngliche, ungewichtete Term-/Dokumentmatrix, den zugehörigen `DocStore` und den `ClusterTree`. Letzterer wird dann um die generierten textuellen Zusammenfassungen ergänzt.

### qmeasure

Zur – möglichst objektiven – Qualitätsbewertung einer Hierarchie dient das Programm `qmeasure`. Es berechnet für jeden Knoten drei Qualitätswerte (*Purity*, *Entropy* und *F-Measure*) sowie deren Mittelwerte für die gesamte Hierarchie. Für die genaue Definition der Qualitätsmaße sei hier auf Abschnitt 5.1.1 verwiesen. Die Berechnung erfolgt direkt nach den dort vorgestellten Funktionen und die Werte werden in den entsprechenden Attributen der Knoten gespeichert. In den Attributen des Wurzelknotens sichert `qmeasure` die Mittelwerte. Ferner erlaubt `qmeasure` zur späteren Identifikation die Angabe einer Beschreibung der Hierarchie, die es als Zeichenkette in dem Attribut *Zusammenfassung* der Wurzel ablegt.

#### cluster2html

Zur visuellen Aufbereitung einer Hierarchie dient das Programm `cluster2html`. Es liest einen `ClusterTree` und den dazugehörigen `DocStore` und erzeugt aus der Angabe einer `Knoten-ID` eine in HTML kodierte Ansicht des entsprechenden Knotens (siehe Abbildungen 4.3 und 4.4). Aufgeführt werden darin alle Nachfolger mit ihren textuellen Zusammenfassungen und Qualitätswerten sowie alle Dokumente, die sich direkt in diesem Knoten befinden, mit Zusammenfassung und Thema.

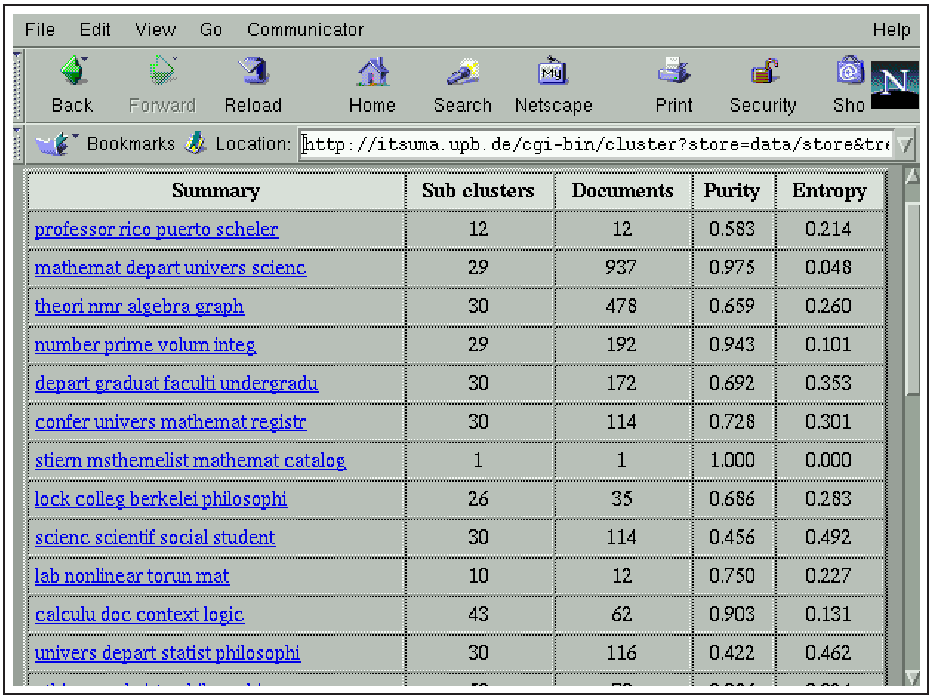
`cluster2html` arbeitet sowohl von der Kommandozeile aus als auch per *CGI*-Schnittstelle. Damit kann es direkt von einem Webserver aus aufgerufen werden. Die ausgegebenen Nachfolger werden als Hyperlinks kodiert, die auf die entsprechenden Knoten verweisen. So kann die Anwenderin oder der Anwender sich komfortabel per Browser durch die gesamte Hierarchie bewegen.

Als einziges der in diesem Abschnitt beschriebenen Programme ist `cluster2html` statt in *C++* in *Python* (siehe Abschnitt 4.1.4) implementiert. Der Grund liegt im wesentlichen in der Erzeugung des Ausgabeformats: Zusammen mit dem *Python*-Modul *HTMLgen* [Hge] läßt sich sehr komfortabel HTML-Text erzeugen. Dank der Skriptanbindug der Bibliothek an *Python* ist `cluster2html` damit sehr einfach implementierbar.

## 4.2 Algorithmen

In die Basis der vorgestellten Bibliothek können die Clustering-Verfahren gut integriert werden werden. Ihre Implementierung kann hierbei sehr direkt der Beschreibung der Algorithmen aus Kapitel 3 folgen, da die Details wie Ein-/Ausgabe oder Speicherverwaltung bereits abgedeckt sind. Dementsprechend wird in den nachfolgenden Abschnitten im wesentlichen auf die Verbindung der Algorithmen mit der Bibliothek eingegangen. Ferner werden zu jedem Verfahren noch einmal alle angebotenen Parametrisierungen zusammengefaßt.

In Abbildung 4.5 ist zu sehen, wie sich die Algorithmen in die Testumgebung einfügen. Jedes Verfahren ist in einem eigenen Programm implementiert, das zumindest eine Term-/Dokumentmatrix als Eingabe bekommt (für die textuellen Zusammenfassungen benötigen die Programme zusätzlich den dazugehörigen `DocStore`). Ausgabe ist jeweils ein `ClusterTree`.




| Summary | Sub clusters | Documents | Purity | Entropy |
|---|---|---|---|---|
| professor rico puerto scheler | 12 | 12 | 0.583 | 0.214 |
| mathemat depart univers scienc | 29 | 937 | 0.975 | 0.048 |
| theori nmr algebra graph | 30 | 478 | 0.659 | 0.260 |
| number prime volum integ | 29 | 192 | 0.943 | 0.101 |
| depart graduat faculti undergradu | 30 | 172 | 0.692 | 0.353 |
| confer univers mathemat registr | 30 | 114 | 0.728 | 0.301 |
| stiern msthemelist mathemat catalog | 1 | 1 | 1.000 | 0.000 |
| lock colleg berkelei philosophi | 26 | 35 | 0.686 | 0.283 |
| scienc scientif social student | 30 | 114 | 0.456 | 0.492 |
| lab nonlinear torun mat | 10 | 12 | 0.750 | 0.227 |
| calculu doc context logic | 43 | 62 | 0.903 | 0.131 |
| univers depart statist philosophi | 30 | 116 | 0.422 | 0.462 |

Abbildung 4.3: Ausgabe von `cluster2html`: Innerer Knoten

### 4.2.1 Simple Single Pass K-Means

Der *K-Means*-Algorithmus ist in der in Abschnitt 3.2.1 vorgestellten Variante in dem Programm `kmeans` implementiert. Neben `TermDocMatrix` und `DocStore` benötigt es folgende Parameter:

- Die Anzahl $k$ von Teilmengen bei jeder Partitionierung.
- Die maximale Größe $m$ einer Dokumentenmenge.
- Die maximale Zahl von Iterationen $t$ pro Partitionierung.
- Die Blockgröße $b$.
- Das zu benutzende Distanzmaß.

`kmeans` erzeugt dann zunächst einen `ClusterTree` mit einem leeren Wurzelknoten und partitioniert alle Dokumentvektoren in $k$ Teilmengen. Für jede (nicht leere) der so entstandenen Mengen erzeugt es dann einen Nachfolgeknoten der Wurzel. Die Zentroiden dieser Knoten werden direkt aus der Ausgabe von *K-Means* übernommen, und auch die Zusammenfassung wird daraus generiert. Die Mengen, die die vorgegebene maximale

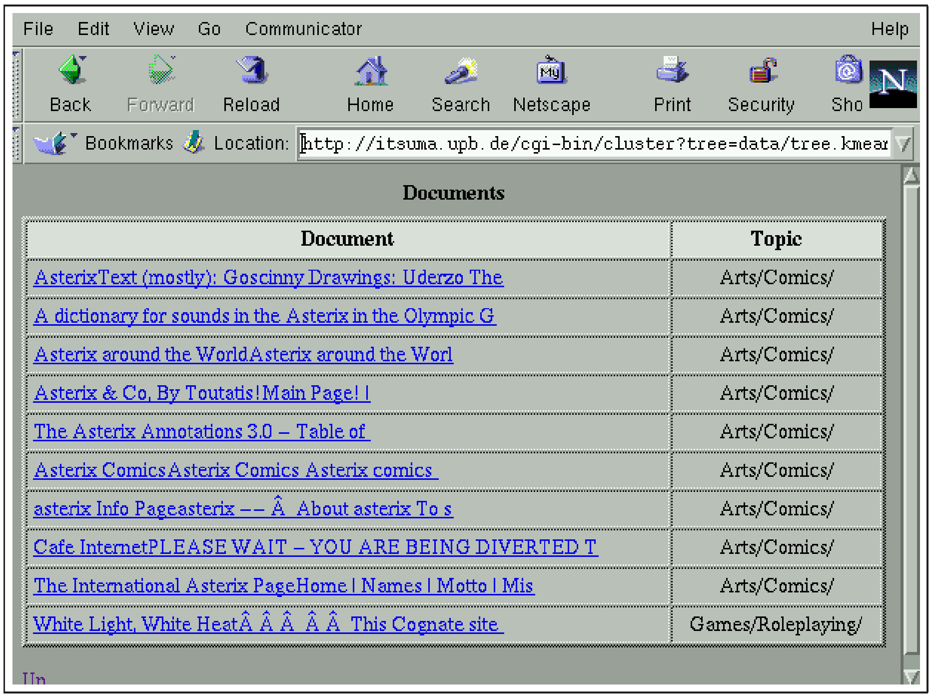



Abbildung 4.4: Ausgabe von `cluster2html`: Blatt

Größe $m$ überschreiten, teilt `kmeans` dann anschließend rekursiv weiter auf. Ist eine Menge dagegen klein genug, werden alle sich darin befindenden Dokumente direkt dem entsprechenden Knoten zugewiesen und die Rekursion abgebrochen. Das Ergebnis ist eine Hierarchie mit bis zu $k$ Ästen pro Knoten, bei der Dokumente nur in den Blättern liegen.

Für die Initialisierung der Zentroide zu Beginn der Berechnung werden $k$ Vektoren aus den ersten $b$ zufällig ausgewählt. Der für jeden Block aufgerufene kanonische *K-Means* wird abgebrochen, wenn sich in einer Iteration entweder keine neuen Zentroide mehr ergeben haben oder die maximale Anzahl $t$ von Iterationen erreicht ist.

### 4.2.2 Spektrales Clustering

Das Programm `spectral` implementiert das spektrale Clustering-Verfahren aus Abschnitt 3.2.2. Es bietet folgende Parameter zur Konfiguration:

- Die Anzahl $k$ von Teilmengen (d. h. Singularwerten) bei jeder Partitionierung.
- Die maximale Größe $m$ einer Teilmenge.

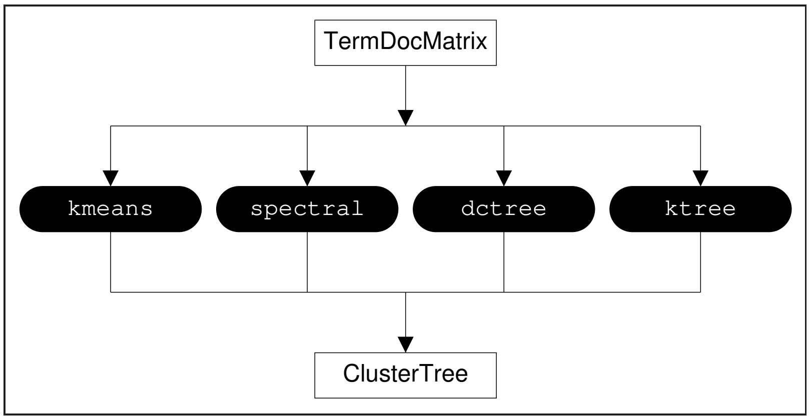



Abbildung 4.5: Clustering-Verfahren eingefügt in die Testumgebung

- Eine Wahrscheinlichkeit $p$, mit der ein Dokumentvektor an der Singularwert-Dekomposition beteiligt wird.
- Eine maximale Anzahl $g$ von Dokumentvektoren, die komplett der Singularwert-Dekomposition unterzogen werden.

`spectral` beginnt mit einem `ClusterTree` nur aus dem leeren Wurzelknoten bestehend, teilt dann mit Hilfe des spektralen Clustering-Verfahrens die Menge aller Dokumentvektoren in $k$ Teilmengen und erzeugt für jede einen Nachfolger der Wurzel. Jede von diesen Mengen, deren Größe die Grenze von $m$ überschreitet, wird dann auf gleiche Art rekursiv verfeinert.

Die textuellen Zusammenfassungen werden von `spectral` nicht über Zentroide generiert, sondern nach der in Abschnitt 3.2.2 beschriebenen alternativen Methode mittels der Konzeptvektoren. Diese stehen durch die durchgeführten Singularwert-Dekompositionen sowieso zur Verfügung und erlauben die Generierung der Zusammenfassungen praktisch ohne Zusatzaufwand. Da `spectral` dementsprechend auf die Berechnung von Zentroiden verzichten kann, füllt es auch die entsprechenden Knotenattribute nicht.

Aufgrund des hohen Aufwands zum Berechnen der Singularwert-Dekompositionen besitzt `spectral` die Möglichkeit, wie in Abschnitt 3.2.2 erläutert, die Dekomposition nur für wenige, ausgewählte Dokumentvektoren auszuführen. Muß an einem Knoten die Singularwert-Dekomposition berechnet werden, so prüft `spectral` zunächst, ob die Anzahl an Vektoren kleiner als $g$ ist. Falls ja, wird die Dekomposition der aus allen Vektoren entstehenden Matrix berechnet. Falls sie größer ist, werden einige ausgewählt: Jeder Vektor wird mit konstanter Wahrscheinlichkeit $p$ in die zu dekomponierende Matrix aufgenommen. Die Singularwert-Dekomposition der verkleinerten Matrix wird dann wieder mittels der aus dem Programm *General Text Parser (GTP)* [Gtp] übernommenen

Implementierung sowie der Klasse `GTPParser` vorgenommen (siehe Abschnitt 4.1.5).

Die Auswahl von Spalten mit *konstanter* Wahrscheinlichkeit ist nicht ganz die in [DFK+99] vorgestellte und zum Beweis der Approximationsgüte benutzte Wahrscheinlichkeitsverteilung (dabei ginge auch noch die Länge der Dokumente mit ein). Experimentelle Ergebnisse in dem Artikel legen aber nahe, daß sich bereits so genügend gute Ergebnisse ergeben.

### 4.2.3 DC-Bäume

Der auf *DC-Bäumen* basierende Clustering-Algorithmus aus Abschnitt 3.2.3 ist in dem Programm `dctree` implementiert. Als Parameter benötigt es:

- Die beiden Schwellwerte $t_1$ und $t_2$.
- Den Verzweigungsfaktor $b$.
- Den Sollwert $m$ für die minimale Anzahl von Nachfolgern eines Knotens.
- Das zu benutzende Distanzmaß.

`dctree` benutzt den zu Beginn nur mit einer Wurzel ausgestatteten `ClusterTree` direkt als *DC-Baum*. Dazu wird folgende Interpretation der Knoten verwendet (siehe Abbildung 4.6):

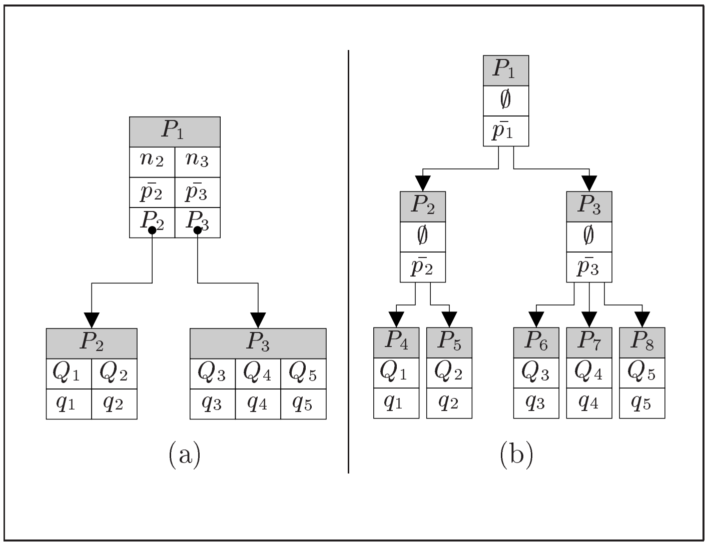



Abbildung 4.6: Abbildung *DC-Baum* (a) auf `ClusterTree` (b)

- Ein *innerer Knoten P eines DC-Baums* hat bis zu $b$ Einträge der Form $(P_i, n_i, \bar{p_i})$, wobei $P_i$ jeweils ein Nachfolgeknoten, $n_i$ die Gesamtzahl von Dokumenten in $P_i$ oder einem seiner Nachfolger und $\bar{p_i}$ deren Zentroid ist. Im `ClusterTree` entspricht dem gerade ein innerer Knoten ohne eigene Dokumente und mit bis zu $b$ Nachfolgern. Die Zentroide $\bar{p_i}$ stehen direkt in den entsprechenden Attributen der Nachfolger, und die Werte $n_i$ werden vom `ClusterTree` berechnet und bereitgestellt.
- Ein *Blatt Q eines DC-Baums* hat bis zu $b$ Einträge mit der Form $(Q_i, \bar{q_i})$, wobei $Q_i$ jeweils eine Menge von Dokumentvektoren und $\bar{q_i}$ ihr Zentroid ist. Diesem Blatt entspricht im `ClusterTree` ein *innerer* Knoten mit bis zu $b$ Blättern als Nachfolger. Die Menge $Q_i$ steht als Menge von `Knoten-ID`s in diesen Blättern, ebenso der dazugehörige Zentroid.

Mit dieser Abbildung eines *DC-Baums* auf einen `ClusterTree` lassen sich die Algorithmen `DCTree_insert` und `DCTree_split` direkt implementieren. Das Programm `dctree` fügt dann der Reihe nach jeden Dokumentvektor in die Hierarchie ein. Da sich die Zentroide an den Knoten mit jeder Einfügeoperation ändern können, werden die textuellen Zusammenfassungen erst bestimmt, wenn alle Vektoren bearbeitet sind. Ein rekursiver Durchlauf durch die gesamte Hierarchie erzeugt dann an jedem Knoten aus den darin stehenden Zentroiden die Zusammenfassungen.

### 4.2.4 K-Bäume

Der im Rahmen dieser Arbeit entstandene *K-Baum*-Algorithmus ist in dem Programm `ktree` implementiert. Es akzeptiert die folgenden Parameter:

- Den Schwellwert $t$.
- Den Verzweigungsfaktor $b$.
- Die maximale Zahl von *K-Means* Iterationen $j$.
- Die Höchstzahl $m$ von Nachfolgern eines Knotens.
- Das zu benutzende Distanzmaß.

Im Unterschied zu einem *DC-Baum* lassen sich *K-Bäume* direkt auf Instanzen der Klasse `ClusterTree` abbilden, da ihre Strukturen übereinstimmen. Ein Knoten im *K-Baum* entspricht genau einem Knoten im `ClusterTree`, und auch die Attribute sind fast identisch. Die einzige zusätzliche Eigenschaft, die der *K-Baum*-Algorithmus benötigt, ist die Unterscheidung zwischen statischen und dynamischen Knoten, die im `ClusterTree` nicht direkt vorgesehen ist. Als Ausweg wird in der implementierten Version eines der drei eigentlich für die Qualitätsmessung gedachten Attribute zweckentfremdet und als Indikator benutzt. Da die Unterscheidung nur während der Bearbeitung des Baumes benötigt wird, kann sie später ohne Probleme von dem Programm `qmeasure` überschrieben werden.

Mit diesem Hilfsmittel lassen sich die beiden Algorithmen `DCTree_insert` und `DCTree_convertNode` leicht implementieren. Der für letzteren benutzte kanonische *K-Means* verwendet jeweils wieder zufällig aus seiner Eingabemenge gewählte Vektoren als Startwerte und iteriert bis sich die Zentroide nicht mehr ändern oder die Höchstzahl $j$ an Iterationen erreicht ist.

# 5 Evaluierung

Nachdem in den vorherigen Kapiteln die untersuchten Clustering-Verfahren und ihre Implementierungen vorgestellt worden sind, kann nun ein qualitativer Vergleich erfolgen. Dafür wird zunächst die grundsätzliche Methode erläutert, anhand derer die Qualität der einzelnen Ergebnisse beurteilt werden soll. Anschließend erfolgt erst eine Einzelbewertung der vier Verfahren und schließlich ein Vergleich mit Interpretation der Ergebnisse.

## 5.1 Methode

Das Ziel der Clustering-Verfahren ist jeweils eine Hierarchie, die der Semantik der Dokumente möglichst weitgehend entspricht. Leider ist Güte in diesem Sinne eine sehr subjektive Qualität, die sich kaum exakt messen läßt. Selbst bei manuell gepflegten Hierarchien wie etwa dem bekannten Webkatalog von *Yahoo! Inc.* [Yah] läßt sich in vielen Fällen trefflich darüber diskutieren, ob eine bestimmte Seite nicht in einer anderen Kategorie besser aufgehoben wäre.

Dennoch werden für die Bewertung der vorgestellten Clustering-Algorithmen objektive Kriterien benötigt, um einen sinnvollen Vergleich zu ermöglichen. Mit einer festen Menge von Dokumenten als Eingabe gibt es dafür zwei prinzipielle Möglichkeiten (Einteilung nach [SKK00]):

*Interne Qualitätsmaße:*
: Die Bewertung erfolgt ohne zusätzliches Wissen über die Dokumente allein aufgrund ihrer relativen Lage in der erzeugten Hierarchie.

*Externe Qualitätsmaße:*
: In die Bewertung gehen zusätzliche Informationen ein, die den Clustering-Verfahren nicht zur Verfügung gestanden haben.

Bei den internen Maßen handelt es sich in der Regel um Kostenfunktionen, die die mittlere Distanz der als zusammengehörig erkannten Dokumentvektoren bestimmen.

Um so geringer diese Entfernung ist, als desto besser wird die Qualität interpretiert. Die Bewertung mit solchen internen Maßen besitzt aber zwei wesentliche Nachteile:

- Aufgrund der in die Kostenfunktion eingehenden Vektordistanzen eignen sie sich nicht für den Vergleich von Verfahren, die unterschiedliche Distanzmaße benutzen [SGM00].
- Die Art der Bewertung setzt bereits voraus, daß das Distanzmaß mit der Semantik der Dokumente korreliert. Möchte man aber auch gerade dies überprüfen – und damit das Vektorraummodell an sich – , so liefern einem die internen Maße keine Hilfe.

Vielversprechender ist die Benutzung von externen Maßen. Bei den genannten zusätzlichen Informationen handelt es sich typischerweise um manuelle Kategorisierungen der Dokumente. Ein externes Maß vergleicht dann das Ergebnis eines Clustering-Verfahrens mit dieser Vorgabe. Befinden sich in der Hierarchie gleich kategorisierte Dokumente in zusammenhängenden Teilmengen, geht dies positiv in die Bewertung ein. Sind dagegen viele Dokumente unterschiedlicher Kategorien zusammengefaßt, verringert sich die Qualität.

Auch dieser Ansatz zeigt einige Probleme:

- Die gegebene Vorkategorisierung wird als Optimum interpretiert, obwohl auch sie wieder sehr subjektiv ist. Hier zeigt sich erneut das bereits angesprochene Problem, daß verschiedene Personen vermutlich auch unterschiedliche Kategorisierungen erzeugen werden.
- Die manuelle Kategorisierung ist bei großen Dokumentenmengen unter Umständen mit einigem Zeitaufwand verbunden.
- Dokumente lassen sich oft nicht eindeutig einem Thema zuordnen. Ein Algorithmus mag daher ganz andere Zusammenhänge erkennen als die manuelle Kategorisierung vorgibt. Zur Milderung könnte man zwar einem Dokument mehrere Kategorien zuordnen, aber auch auf diese Art würden sicher nicht alle Querverbindungen erkannt werden.

Trotz dieser Schwierigkeiten sind externe Maße insbesondere zum Vergleich verschiedener Ansätze die bessere Wahl [SGM00]. Daher wird in dieser Arbeit die Qualität mit Hilfe von drei externen Maßen bewertet. Die beschriebenen Probleme im Hinterkopf sei damit für alle Beurteilungen definiert: Um so höher die nachfolgend beschriebenen Qualitätsmaße eine Hierarchie bewerten, desto besser ist sie.

Ergänzend zur Bewertung der Verfahren nach ihren Qualitätswerten wird dennoch jeweils noch ein subjektiver Blick auf die erzeugten Hierarchien geworfen. Ohne Anspruch auf Vollständigkeit werden dabei jeweils einige ins Auge fallende Eigenschaften vorgestellt. Das können sowohl Form (etwa Balance einer Hierarchie) als auch Inhalt (z. B. besonders gute oder schlechte Kategorisierungen) sein. Das Ziel dieser Beschreibungen soll dabei aber weniger eine Bewertung im strengen Sinne darstellen als mehr einen Eindruck von den erzeugten Ergebnissen vermitteln. Auf diese Art lassen sich auch generell etwas die Möglichkeiten und Grenzen der in dieser Arbeit vorgestellten Modellierung erkennen.

### 5.1.1 Qualitätsmaße

Die drei in dieser Arbeit benutzten externen Qualitätsmaße, *Purity*, *Entropy* und *F-Measure*, sollen hier nun zunächst beschrieben werden. Dazu sei jedem Dokument $v_i$ ($i \in \{1, \ldots, n\}$) zusätzlich ein Thema $t_i \in T$ mit $T := \{1, \ldots, m\}$ zugeordnet. Dabei ist $m$ die Gesamtzahl der verschiedenen in der Eingabemenge vorkommenden Themen. Ferner sei zu einem Thema $t \in T$ und einer Menge $S \subseteq \{ \ v_i \mid i \in \{1, \ldots, n\} \ \}$ der Wert $n_t^{(S)}$ definiert als die Anzahl der Dokumente mit Thema $t$ in $S$.

#### Purity

Ein einfaches Maß ist *Purity* [SGM00]. Es bewertet zunächst einmal nur eine einzelne Menge $S \subseteq \{ \ v_i \mid i \in \{1, \ldots, n\} \ \}$. Dazu bestimmt es das Thema $t$, das in der Menge am häufigsten vertreten ist, und berechnet den Qualitätswert dann als den Anteil der Dokumente mit diesem Thema an der Gesamtmenge:

$$\text{Purity}(S) := \frac{max_{t \in T} \ \left\{ n_t^{(S)} \right\}}{|S|} \quad \text{für} \quad S \neq \emptyset$$

Die Werte liegen nach Definition im Intervall $[0, 1]$. Das Optimum von 1 bedeutet, daß die Dokumente in der Menge $S$ alle dasselbe Thema besitzen (trivialerweise wird das Optimum für eine einelementige Menge erreicht).

Das *Purity*-Maß geht davon aus, daß zu jeder Menge $S$ ein dominantes Thema $t$ gehört und bewertet jedes davon abweichende Dokument als negativ. Eine weitergehende Sichtweise liefert [LC96]: Interpretiert man die zu dem (postulierten) dominanten Thema gehörenden Dokumente als *relevant* und alle anderen als *nicht relevant* in bezug zur Menge S, erhält man genau den dort definierten *separation factor*, in welchem sich wieder die *Cluster-Hypothese* (siehe Abschnitt 2.1.1) zeigt: Relevante Dokumente unterscheiden sich auf eine gewisse Art von den nicht relevanten und sollten daher durch ein Clustering-Verfahren in getrennten Mengen plaziert werden.

Im Rahmen dieser Arbeit wird das *Purity*-Maß von den Einzelmengen einer Partition (wie in [SGM00]) auf eine Hierarchie verallgemeinert: Jedem Knoten der Hierarchie wird ein *Purity*-Wert zugeordnet, der aus allen Dokumenten entsteht, die in der Hierarchie diesem Knoten oder einem seiner Nachfolger zugeordnet sind. Ferner wird der Hierarchie insgesamt ein Wert zugeordnet, der dem Mittelwert der *Purity*-Werte aller Knoten entspricht. Bei der Bildung dieses Mittelwertes geht jeder Knoten mit der Anzahl der durch ihn repräsentierten Dokumente als Gewicht ein. Ist $H$ ein Hierarchiebaum mit Knoten $H_1, \ldots, H_u$ und $S_i$ die Menge der Dokumente, die dem Knoten $H_i$ oder einem seiner Nachfolger zugeordnet sind, so ist damit definiert:

$$\begin{aligned}\mathrm{Purity}(H_i) &:= \mathrm{Purity}(S_i) \\ \mathrm{Purity}(H) &:= \frac{\sum_{i=1}^{u} \mathrm{Purity}(S_i) * |S_i|}{\sum_{i=1}^{u} |S_i|}\end{aligned}$$

Der so definierte *Purity*-Wert von $H$ ist *nicht* der *Purity*-Wert der Wurzel.

### Entropy

Das zweite benutzte Maß ist die Entropie (*Entropy*). Es ist (mit leicht unterschiedlichen Details) weit verbreitet (siehe z. B. [SKK00] oder [KHK99]). Für $S \subseteq \{ v_i \mid i \in \{1, \ldots, n\} \}$ sei es hier in Anlehnung an [SGM00] definiert als:

$$\mathrm{Entropy}(S) := \frac{1}{-log(|T|)} \sum_{t \in T} \frac{n_t^{(S)}}{|S|} \, log\left(\frac{n_t^{(S)}}{|S|}\right) \quad \text{für} \quad S \neq \emptyset$$

Im Unterschied zu *Purity*, bei dem nur eines der in der Menge $S$ vorkommenden Themen in die Bewertung eingeht, bezieht *Entropy* die Verteilung aller Themen ein. Für jedes Thema wird dessen Anteil an der Gesamtmenge berechnet und mit seinem Logarithmus skaliert. Die Summe aus all diesen Werten stellt schon das wesentliche Qualitätsmerkmal dar. Für den endgültigen *Entropy*-Wert wird sie noch mittels $\frac{1}{log(|T|)}$ skaliert (denn umso mehr Themen es gibt, desto besser ist es zu bewerten, wenn sie gut getrennt werden) und schließlich noch in den positiven Bereich transformiert. Damit erreicht *Entropy* das Optimum (nur ein Thema kommt in $S$ vor) bei 0. Umso stärker dagegen verschiedene Themen an der Menge $S$ beteiligt sind, desto näher kommt der *Entropy*-Wert der 1. Wie *Purity* erreicht auch *Entropy* das Optimum trivialerweise bei einer einelementigen Menge.

Auch dieses Qualitätsmaß bewertet erstmal nur eine einzelne Menge. Analog zu dem Vorgehen bei *Purity* läßt sich aber auch *Entropy* für eine Hierarchie verallgemeinern: Sei wieder $H$ ein Hierarchiebaum mit Knoten $H_1, \ldots, H_u$ und $S_i$ die Menge der Dokumente, die dem Knoten $H_i$ oder einem seiner Nachfolger zugeordnet sind. Dann definiere:

$$\begin{aligned}\text{Entropy}(H_i) &:= \text{Entropy}(S_i)\\ \text{Entropy}(H) &:= \frac{\sum_{i=1}^{u} \text{Entropy}(S_i) * |S_i|}{\sum_{i=1}^{u} |S_i|}\end{aligned}$$

### F-Measure

Das *F-Measure*-Maß [vR79] arbeitet etwas anders als die beiden bisher vorgestellten. *Purity* und *Entropy* bewerten immer nur die Zusammensetzung einer Menge. Sie ignorieren dabei, wie groß der Anteil an allen Dokumente eines Themas innerhalb der Menge ist. Im Extremfall bewerten sie lauter einelementige Mengen als optimal, obwohl die zusammengehörenden Dokumente dabei gar nicht weiter getrennt sein könnten. *F-Measure* versucht dies mit Hilfe der Konzepte *Recall* und *Precision* aus dem *Information Retrieval* zu korrigieren.

Erstmals vorgestellt wird das *F-Measure*-Maß in [vR79] während der Konstruktion eines allgemeinen Bewertungsrahmens für die Qualität von Suchergebnissen. Dafür werden wieder die zu einer Anfrage an eine Dokumentensammlung *relevanten* von den *nicht relevanten* Dokumenten unterschieden. Bezogen auf das konkrete Resultat einer Suche heißt dann der Anteil der relevanten Dokumente am Ergebnis *Precision (P)* und der Anteil der gelieferten relevanten Dokumente an der Gesamtzahl *Recall (R)*. Als ein Qualitätsmaß, bei dem *Precision* und *Recall* die gleiche Wichtigkeit erhalten, wird für ein Anfrageergebnis $S$ hergeleitet:

$$\text{F-Measure}(S) := \frac{2PR}{P+R}$$

Nach Definition gilt *F-Measure*$(S) \in [0,1]$, und das Maß erreicht sein Optimum 1 für ein Ergebnis $S$, das genau alle relevanten Dokumente enthält (d. h. *Recall* und *Precision* sind jeweils 1).

In [LA99] wird das *F-Measure*-Maß auf die Bewertung von Clustering-Ergebnissen übertragen. Dazu sei wie zuvor $H$ ein Hierarchiebaum mit Knoten $H_1, \ldots, H_u$ und $S_i$ die Menge der Dokumente, die dem Knoten $H_i$ oder einem seiner Nachfolger zugeordnet sind. Dann wird für ein festes Thema $t \in T$ einmal jede Menge $S_i$ als ein Suchergebnis interpretiert. Dabei sind nach Definition alle Dokumente mit Thema $t$ relevant, alle anderen nicht relevant. Ist $m_t$ die Gesamtzahl von Dokumenten mit Thema $t$, so gilt:

$$\text{Precision}_t^{(S_i)} = \frac{n_t^{(S_i)}}{|S|} \quad \text{und} \quad \text{Recall}_t^{(S_i)} = \frac{n_t^{(S_i)}}{m_t}$$

Die Annahme ist nun, daß es für jedes Thema in der Hierarchie einen Knoten $H_i$ gibt, dessen Dokumente $S_i$ es gut repräsentieren. Um diesen Knoten zu finden, wird für jeden Knoten der *F-Measure*-Wert einmal für jedes Thema $t \in T$ bestimmt:

$$\text{F-Measure}_t(H_i) := \frac{2\ \text{Precision}_t^{(S_i)}\ \text{Recall}_t^{(S_i)}}{\text{Precision}_t^{(S_i)} + \text{Recall}_t^{(S_i)}}$$

Für jedes Thema $t$ ist dann der Knoten $H_i$, der *F-Measure*$_t(H_i)$ maximiert, der gesuchte. Ein Thema bekommt diesen Wert als *F-Measure*-Wert zugewiesen:

$$\text{F-Measure}(t) := max_{i \in \{1,\dots,u\}}\ \{\ \text{F-Measure}_t(H_i)\}$$

Schließlich wird der gewichtete Mittelwert von *F-Measure*$(t)$ der *F-Measure*-Wert der gesamten Hierarchie:

$$\text{F-Measure}(H) := \frac{1}{n} \sum_{t=1}^{|T|} m_t\, \text{F-Measure}(t)$$

Diese Art der Bewertung unterscheidet sich deutlich von *Purity* und *Entropy*. Statt sich nur die einzelnen Mengen anzuschauen, nimmt das *F-Measure*-Maß die Themen als Ausgangspunkt und betrachtet die ganze Hierarchie daraufhin, ob es für jedes einen Knoten gibt, der gut mit ihm übereinstimmt. Die Bewertung der Gesamthierarchie ist um so höher, je besser sich solche Knoten identifizieren lassen. Dabei spielt es dann keine Rolle mehr, wie die Hierarchie an anderen Stellen aussieht. Setzt man allerdings als Ziel die gute Gesamtform einer Hierarchie, mag *F-Measure* das unter Umständen aus genau diesem Grund nicht zu erfassen. Falls es daher im folgenden abzuwägen gilt zwischen guten *Purity*- oder *Entropy*-Werten und der *F-Measure*-Beurteilung, so wird ersteren der Vorzug gegeben.

### 5.1.2 Testdaten

Damit die zuvor beschriebenen Qualitätsmaße benutzt werden können, wird eine Dokumentenkollektion benötigt, in der jedes Element zusätzlich mit einem Thema ausgezeichnet ist. Da die Clustering-Verfahren im Rahmen dieser Arbeit mit Webseiten als Eingabe getestet werden sollen, stellt diese Kategorisierung ein Problem dar, denn nur die wenigsten dieser Dokumente enthalten von sich aus auch entsprechende – sinnvolle – Auszeichnungen wie etwa Schlüsselwörter (und selbst diese sind typischerweise völlig unstandardisiert und damit nicht vergleichbar).

Ein Ansatz wäre die manuelle Nachkategorisierung von Dokumenten, die zuvor nach irgendeinem Schema gesammelt worden sind. Dies wird aber spätestens bei mehr als 1000 Seiten mühsam und ist im Rahmen dieser Arbeit nicht machbar. Aber diese manuelle, vom Autor der Seite unabhängige Klassifizierung liefert dennoch eine gute Idee: Es gibt im Internet bereits große Kataloge von thematisch eingeordneten Seiten; einer der bekanntesten ist etwa der schon einmal angesprochene Katalog von *Yahoo! Inc.* . Es liegt daher nahe, sich diese bereits geleistete Arbeit zunutze zu machen.

Statt einem der rein kommerziellen Kataloge ist allerdings eine freie Alternative als Ausgangsbasis verwendet worden: der von vielen Personen weltweit gepflegte Katalog des *Open Directory Projects* [Odp]. Er steht zwar unter der Aufsicht der *Netscape Communications Corporation*, ist aber – mit leichten Einschränkungen – frei benutzbar. Eine gute Qualität der Einträge sollte dabei allein schon dadurch zu erwarten sein, daß auch Anbieter wie *Google Inc.* oder *Terra Lycos* diese Daten nutzen. Als der größte Vorteil dieses Katalogs stellt sich heraus, daß der komplette Datenbestand in einem XML-Format vorliegt und sich die Adressen daher sehr leicht extrahieren lassen. Einen kurzen Ausschnitt als Beispiel zeigt die Abbildung 5.1.

Aus diesem Katalog sind 23 Kategorien (von insgesamt 604) auf der zweiten Hierarchiestufe willkürlich ausgewählt worden (siehe Tabelle 5.1). Dies hat zu einer Testkollektion mit etwas über 40.000 Dokumenten[1] geführt, die rund 280.000 verschiedene Wortstämme enthält und eine Größe von knapp 500MB besitzt. Die genaue Verteilung der Dokumente auf die einzelnen Themen ist ebenfalls in Tabelle 5.1 zu erkennen. Es ist durchaus gewollt, daß darunter sowohl Gebiete mit sehr vielen als auch solche mit nur wenigen Dokumenten sind. Aus all diesen Webseiten hat dann das Programm `coll2store` (Abschnitt 4.1.5) einen `DocStore` (Abschnitt 4.1.3) mit einer Größe von etwa 60MB erzeugt (Dauer ca. sieben Minuten auf dem später genauer vorgestellten Rechner), aus dem `makematrix` anschließend in rund acht Minuten eine ungefilterte Matrix mit einer Größe von etwa 50MB generiert hat. Diese Matrix stellt nun die Basis der weiteren Betrachtungen dar.

[1] Ingesamt enthält der Katalog des *Open Directory Projects* rund 4,3 Mio. Adressen

```
<Topic r:id="Top/Arts/Literature/Authors/C/Carroll,_Lewis">
  <catid>56350</catid>
  <link r:resource="http://home.earthlink.net/~lfdean/carroll/index.html"/>
  <link r:resource="http://www.poets.org/LIT/poet/lcarrfst.htm"/>
  [...]
</Topic>

<ExternalPage about="http://home.earthlink.net/~lfdean/carroll/index.html">
  <d:Title>The Lewis Carroll E-text Collection</d:Title>
  <d:Description>&quot;A resource for Carroll fans, containing e-texts of
  his poetry, essays, puzzles, and lesser known works . . . .&quot; An
  excellent and comprehensive site on Carroll's writing, including a
  substantial archive of his work.
  </d:Description>
</ExternalPage>

<ExternalPage about="http://www.poets.org/LIT/poet/lcarrfst.htm">
  <d:Title>Lewis Carroll</d:Title>
  <d:Description>&quot;The Academy of American Poets presents a biography,
  photograph, and selected poems.&quot;</d:Description>
</ExternalPage>
[...]
```

Abbildung 5.1: Ausschnitt aus dem XML-Katalog des *ODPs*

Die Webseiten, die zu den aus dem *ODP*-Katalog extrahierten Adressen gehören, sind mit dem Crawler[2] *Larbin* [Lar] zusammengetragen worden. Damit *Larbin* die Seiten in dem in Abschnitt 4.1.5 vorgestellten Format ablegt, ist es nötig gewesen, das Programm entsprechend anzupassen. *Larbin* hat die Seiten im übrigen nicht rekursiv verfolgt und hat daher nur die Seiten in die Dokumentensammlung aufgenommen, auf die direkt aus dem Katalog verwiesen wird. Ferner ist zu bemerken, daß *Larbin* in der verwendeten Version 1.2.0 noch keine Weiterleitungen (*redirections*) beherrscht hat, wodurch Seiten, die auf einen anderen Server umgezogen sind, nicht als solche erkannt werden konnten. Neuere Versionen von *Larbin* ($\geq$ 1.2.1) beherrschen dies inzwischen korrekt.

## 5.2 Einzelbewertung

Im folgenden sollen die Ergebnisse der vier in dieser Arbeit betrachteten Clustering-Verfahren auf der Testkollektion zunächst für sich vorgestellt werden. Da nicht von vorneherein klar ist, welche Wahl der diversen Parameter das jeweils beste Ergebnis liefert, werden zunächst verschiedene Variationen überprüft. Anschließend wird anhand der sich ergebenen Qualitätswerte jeweils eine konkrete Parametrisierung ausgewählt,

[2]Unter *Crawler* wird ein Programm verstanden, das automatisiert große Mengen von Webseiten sammeln kann.

Tabelle 5.1: Themen der Testkollektion

| ***Thema*** | ***Anzahl*** |
|---|---|
| `Arts/Comics` | 2671 |
| `Arts/Movies` | 8762 |
| `Business/Employment` | 861 |
| `Business/Venture_Capital` | 462 |
| `Computers/Desktop_Publishing` | 13 |
| `Computers/Programming` | 5854 |
| `Games/Gambling` | 1338 |
| `Games/Roleplaying` | 2435 |
| `Health/Fitness` | 331 |
| `Health/Pharmacy` | 859 |
| `Home/Family` | 2364 |
| `Home/Gardens` | 1942 |
| `News/Politics` | 80 |
| `News/Weather` | 98 |
| `Recreation/Birdwatching` | 468 |
| `Recreation/Climbing` | 354 |
| `Reference/Knowledge_Management` | 361 |
| `Reference/Museums` | 1749 |
| `Science/Chemistry` | 1221 |
| `Science/Math` | 4477 |
| `Society/Crime` | 947 |
| `Society/Philosophy` | 2061 |
| `Sports/Golf` | 825 |
| *Gesamt* | 40533 |

die dann als Basis für die weiteren Betrachtungen dient.

Ein eher subjektiver Blick auf die damit entstandenen Hierarchien liefert dabei zunächst jeweils einen Eindruck von den möglichen Ergebnissen. Im Abschnitt 5.3 dienen die erzielten Qualitätswerte der ausgewählten Parametrisierungen dann einem Vergleich der vier Clustering-Verfahren untereinander. Die Messung der Laufzeiten wird ebenfalls auf Abschnitt 5.3 verschoben, da der Zeitaufwand nicht für alle durchgeführten Varianten exakt gemessen werden konnte. Der Grund dafür liegt vor allem darin, daß die Testläufe parallel auf mehreren Rechnern mit unterschiedlichen Ausstattungen vorgenommen worden sind. Zusätzlich haben diese Rechner während der Berechnungen nicht exklusiv zur Verfügung gestanden, so daß die Rechenzeit teilweise mit anderen Prozessen zu teilen gewesen ist. Dadurch ist ein sinnvoller Vergleich gar nicht möglich. Nachdem aber die interessanten Parametrisierungen erkannt worden sind, haben diese dann noch einmal zur Laufzeitmessung in einer wohldefinierten Umgebung gedient.

Bei der Wahl der zu variierenden Parameter ist eine gewisse Beschränkung nötig gewesen: Sowohl das Vektorraummodell als auch die zusätzlichen Eingabewerte der Algorithmen bieten eine große Fülle von Möglichkeiten, deren Kombinationen sich nicht alle haben ausprobieren lassen.

Für das Vektorraummodell kommen daher nur folgende Variationen zum Einsatz:

- An Distanzmaßen werden die Manhattan-Distanz und das Cosinus-Maß benutzt. Wenn die Manhattan-Distanz zum Einsatz kommt, werden die Vektoren der Term-/Dokumentmatrix mit der $L_1$-Norm normiert, beim Cosinus-Maß mit der euklidischen $L_2$-Norm.
- Für die Dimensionsreduktion werden (sofern sinnvoll) sowohl *Content Bearing Words* als auch *Latent Semantic Indexing* verwendet.

  Bei der *CBW*-Reduktion werden Schwellwerte $c \in \{\ 0,4\ ,\ 0,5\ ,\ 0,6\ \}$ verwendet (kleinere Schwellwerte führen zu einer Vielzahl von Dokumenten, die nur noch durch den Nullvektor repräsentiert werden; größere Grenzen reduzieren die Dimension der Vektoren nicht genug). Wird die Dimension mittels *LSI* reduziert, werden etwa $k \approx 100$ Singularwerte betrachtet (aufgrund der für die Singularwert-Dekomposition benutzten Algorithmen läßt sich dies nicht ganz exakt vorgeben). In der Tabelle 5.2 sind zur Veranschaulichung die Dimensionen der entstehenden Vektoren gegenübergestellt.
- Die Term-/Dokumentmatrix wird binär und mit *TFIDF* gewichtet; zusätzlich kommt auch noch die ungewichtete Matrix zum Einsatz.

Tabelle 5.2: Dimensionen der Vektoren

| ***Reduktion*** | ***Dimension*** |
|---|---|
| Ohne | 279449 |
| CBW | |
| $c = 0,4$ | 24448 |
| $c = 0,5$ | 30839 |
| $c = 0,6$ | 61128 |
| LSI | $\approx 100$ |

Die Parameter der einzelnen Clustering-Verfahren und ihrer Implementierungen werden größtenteils fest gewählt. Bei der Wahl der Konstanten handelt es sich dabei jeweils um Werte, die sich in kurzen, eher informalen Voruntersuchungen als geeignet erwiesen haben.

### 5.2.1 Simple Single Pass K-Means

Zur Evaluierung des *Simple Single Pass K-Means* Verfahrens sind die folgenden Parameter fest gewählt worden:

- Maximal $k = 30$ Teilmengen pro Partitionierung.
- Maximal $m = 40$ Dokumente in einem Blatt.
- Höchstens $t = 20$ Iterationen des kanonischen *K-Means*.
- Eine Blockgröße von $b = 600$ Vektoren.

Bei der Bestimmung der Qualitätswerte ist zu beachten, daß *K-Means* durch die zufällige Wahl der Startzentroide in der beschriebenen Implementierung einen randomisierten Algorithmus darstellt und die Ergebnisse daher schwanken können. Aus diesem Grund sind sämtliche Messungen dreimal durchgeführt worden. Bei den hier vorgestellten Werten handelt sich es jeweils um die Mittelwerte der drei Resultate.

In Tabelle 5.3 finden sich die Ergebnisse für den Fall, daß zur Dimensionsreduktion die Methode der *Content Bearing Words* zum Einsatz kommt. Als erstes ist zu bemerken, daß der direkte Vergleich zwischen den beiden Distanzmaßen klar ausfällt: Das Cosinus-Maß liefert ohne Ausnahme die deutlich besseren Ergebnisse; es ist teilweise mehr als doppelt so gut. Dementsprechend wird die Manhattan-Distanz hier nicht weiter betrachtet.

Die besten *Purity*- und *Entropy*-Werte zeigen sich bei jeder der drei Gewichtungen für einen *CBW*-Schwellwert von $c = 0,4$, die höchsten *F-Measure*-Werte bei $c = 0,5$ (binär) oder $c = 0,6$ (ohne Gewichtung bzw. mit *TFIDF*). Alle drei Maße erreichen ihre insgesamt besten Werte bei der Gewichtung mit *TFIDF*.

Bei der Dimensionsreduktion mittels *Latent Semantic Indexing* haben sich die Werte in Tabelle 5.4 ergeben. Auch hier ist dieselbe deutliche Diskrepanz zwischen Manhattan-Distanz und Cosinus-Maß zu erkennen. Alle drei Qualitätswerte erreichen ihre Bestwerte bei binärer Gewichtung, kommen aber nicht an die Vorgaben der *CBW*-Reduktion heran.

Als Kandidat für die weitere Untersuchung sei daher eine der Hierarchien ausgewählt, die bei *TFIDF*-Gewichtung, *CBW*-Dimensionreduktion mit einem Schwellwert von $c = 0,4$ und Cosinus-Maß entstanden sind, da ihr Mittelwert bei zwei Qualitätsmaßen die insgesamt besten Werte zeigt. Beim dritten Maß, *F-Measure*, liegt ihr Wert zumindest im Mittelfeld. Wie in Abschnitt 5.1.1 erläutert, wird hier den ersten beiden Werten der Vorzug gegeben. Von den drei Hierarchien, die zur Bildung der Mittelwerte beigetragen haben, soll diejenige genommen werden, deren Bewertungen am dichtesten an den Durchschnittswerten liegen. Die entsprechenden Einzelergebnisse sind in Tabelle 5.5

Tabelle 5.3: K-Means: Dimensionsreduktion mittels *Content Bearing Words*

| c | ***Purity*** | | ***Entropy*** | | ***F-Measure*** | |
|---|---|---|---|---|---|---|
| | $L_1$ | $L_2$ | $L_1$ | $L_2$ | $L_1$ | $L_2$ |
| 0,4 | 0,219 | **0,561** | 0,798 | **0,452** | 0,252 | 0,424 |
| 0,5 | 0,213 | 0,545 | 0,804 | 0,470 | 0,232 | 0,412 |
| 0,6 | 0,415 | 0,540 | 0,415 | 0,475 | 0,415 | **0,455** |

(a) Ohne Gewichtung

| c | ***Purity*** | | ***Entropy*** | | ***F-Measure*** | |
|---|---|---|---|---|---|---|
| | $L_1$ | $L_2$ | $L_1$ | $L_2$ | $L_1$ | $L_2$ |
| 0,4 | 0,216 | **0,537** | 0,800 | **0,477** | 0,237 | 0,420 |
| 0,5 | 0,212 | 0,523 | 0,806 | 0,496 | 0,230 | **0,426** |
| 0,6 | 0,207 | 0,482 | 0,810 | 0,537 | 0,227 | 0,409 |

(b) Binäre Gewichtung

| c | ***Purity*** | | ***Entropy*** | | ***F-Measure*** | |
|---|---|---|---|---|---|---|
| | $L_1$ | $L_2$ | $L_1$ | $L_2$ | $L_1$ | $L_2$ |
| 0,4 | 0,218 | **0,580** | 0,799 | **0,439** | 0,239 | 0,430 |
| 0,5 | 0,213 | 0,576 | 0,804 | 0,442 | 0,236 | 0,455 |
| 0,6 | 0,211 | 0,568 | 0,807 | 0,451 | 0,230 | **0,494** |

(c) *TFIDF*-Gewichtung

aufgeführt. Ausgewählt wird demnach das Ergebnis des zweiten Durchlaufs. Im übrigen läßt diese Tabelle beispielhaft auch erkennen, daß die Schwankungsbreite der Werte insgesamt recht gering ist. Dies gilt auch für alle anderen, hier nicht einzeln aufgeführten Werte.

Betrachtet man nun diese ausgewählte Hierarchie näher, lassen sich einige Auffälligkeiten erkennen. Die Tabelle 5.6 zeigt einen Ausschnitt aus dem Wurzelknoten. Zu sehen sind einige klar betitelte Kategorien (etwa Nummer 17, 24 und 27). Dem *Entropy*-Wert nach scheint z. B. die mathematische Kategorie mit Nummer 17 auch sehr gut mit der Vorkategorisierung der darin liegenden Dokumente übereinzustimmen. Interessant ist auch ein Blick in die Philosophie unter Nummer 24 (siehe Tabelle 5.7): In Unterkategorie 15 sind sehr schön die drei bedeutendsten griechischen Philosophen der Antike erkannt

Tabelle 5.4: K-Means: Dimensionsreduktion mittels *Latent Semantic Indexing*

| **Gewichtung** | ***Purity*** | | ***Entropy*** | | ***F-Measure*** | |
|---|---|---|---|---|---|---|
| | $L_1$ | $L_2$ | $L_1$ | $L_2$ | $L_1$ | $L_2$ |
| Ohne | 0,205 | 0,464 | 0,811 | 0,541 | 0,227 | 0,335 |
| Binär | 0,201 | **0,545** | 0,814 | **0,470** | 0,225 | **0,412** |
| TFIDF | 0,211 | 0,523 | 0,806 | 0,486 | 0,231 | 0,387 |

Tabelle 5.5: K-Means: Einzelergebnisse der ausgewählten Parametrisierung

| **Nr.** | ***Purity*** | ***Entropy*** | ***F-Measure*** |
|---|---|---|---|
| 1 | 0,583 | 0,436 | 0,434 |
| 2 | 0,579 | 0,440 | 0,429 |
| 3 | 0,579 | 0,442 | 0,426 |
| Mittel | 0,580 | 0,439 | 0,430 |

worden. Wie allerdings etwa Nummer 17 thematisch hineinpaßt, ist kaum zu erklären; darunter befinden sich Dokumente unter anderem aus den Bereichen „Garten“, „Chemie“ und „Arznei“[3] ohne erkennbaren Zusammenhang.

Ebenfalls an dieser Philosophie-Kategorie zu erkennen ist die Unregelmäßigkeit in den Größen der Unterkategorien. Dies ist ein deutliches Merkmal der zentroidbasierten Aufteilung, wie sie der *K-Means*-Algorithmus vornimmt, denn oft bekommt ein Zentroid dabei nur relativ wenige Vektoren zugewiesen. Dies muß aber kein Nachteil sein, da auch die Dokumente – wie in der Testkollektion – sehr ungleichmäßig auf verschiedene Themen verteilt sein können. Dennoch ist zu bemerken, daß sich gerade für die weniger stark vertretenen Themen zumindest am Beginn der Hierarchie nur bei wenigen eine entsprechende Kategorie ausmachen läßt.

[3] Titel: „`History of Coffee Brewing Devices`“

Tabelle 5.6: K-Means: Wurzel der Hierarchie (Ausschnitt)

| ***Nr.*** | ***Zusammenfassung*** | ***Äste*** | ***Dokum.*** | ***Pur.*** | ***Entr.*** |
|---|---|---|---|---|---|
| ... | ... | ... | ... | ... | ... |
| 14 | `game war math thevinesnetwork` | 30 | 1733 | 0.533 | 0.429 |
| 15 | `univers kb spp visual` | 30 | 894 | 0.293 | 0.763 |
| 16 | `elian error casino game` | 30 | 524 | 0.342 | 0.719 |
| 17 | `math mathemat calculu algebra` | 30 | 2148 | 0.821 | 0.286 |
| 18 | `fractal ultrafract fractint mandelbrot` | 30 | 1767 | 0.552 | 0.547 |
| 19 | `parent jun txt bin` | 30 | 1080 | 0.613 | 0.428 |
| 20 | `le nietzsch de heidegg` | 30 | 899 | 0.204 | 0.743 |
| 21 | `plant bulb flower soil` | 30 | 1615 | 0.761 | 0.347 |
| 22 | `climb mt pic elendor` | 30 | 694 | 0.307 | 0.670 |
| 23 | `code cipher gsm ture` | 30 | 465 | 0.222 | 0.762 |
| 24 | `philosophi conscious philosoph plato` | 30 | 1635 | 0.528 | 0.508 |
| 25 | `rand objectivist objectiv ayn` | 30 | 875 | 0.402 | 0.625 |
| 26 | `gc sin dom jacob` | 30 | 679 | 0.274 | 0.763 |
| 27 | `murder crime victim polic` | 30 | 2572 | 0.602 | 0.439 |
| ... | ... | ... | ... | ... | ... |

Tabelle 5.7: K-Means: Ein bißchen Philosophie (Ausschnitt)

| ***Nr.*** | ***Zusammenfassung*** | ***Äste*** | ***Dokum.*** | ***Pur.*** | ***Entr.*** |
|---|---|---|---|---|---|
| ... | ... | ... | ... | ... | ... |
| 10 | `anim treehawk ethic wirehead` | 0 | 7 | 0.714 | 0.251 |
| 11 | `pool titlevii iamond careerfair` | 0 | 3 | 0.667 | 0.200 |
| 12 | `system ideafish softwar ontologi` | 0 | 33 | 0.212 | 0.672 |
| 13 | `thy est god moral` | 0 | 29 | 0.690 | 0.394 |
| 14 | `god moral human natur` | 30 | 103 | 0.845 | 0.212 |
| 15 | `aristotl plato socrat kant` | 30 | 71 | 0.775 | 0.305 |
| 16 | `physic properti entropi symmetri` | 30 | 41 | 0.488 | 0.432 |
| 17 | `wavelet br superconduct signal` | 0 | 14 | 0.429 | 0.433 |
| 18 | `dewei theori haberma ricoeur` | 30 | 109 | 0.523 | 0.246 |
| 19 | `lightboard bulb` | 0 | 1 | 1.000 | 0.000 |
| 20 | `hume stoic lock cicero` | 30 | 88 | 0.682 | 0.321 |
| 21 | `chapter aristotl greek russell` | 0 | 14 | 0.357 | 0.494 |
| 22 | `logic philosoph wittgenstein philosophi` | 30 | 175 | 0.646 | 0.248 |
| 23 | `genet ethic social human` | 0 | 38 | 0.474 | 0.503 |
| 24 | `pic db wingman counti` | 0 | 10 | 0.500 | 0.368 |
| 25 | `nietzsch zarathustra moral postmodernist` | 0 | 20 | 0.750 | 0.282 |
| ... | ... | ... | ... | ... | ... |

### 5.2.2 Spektrales Clustering

Die zusätzlichen Eingabeparameter beim spektralen Clustering sind alle fest gewählt geworden:

- Maximal $k = 50$ Teilmengen pro Partitionierung (und damit jeweils höchstens $k$ berechnete Singularwerte).
- Maximal $m = 50$ Dokumente in einem Blatt.
- Keine Begrenzung in der Zahl der Vektoren, die der Singularwert-Dekomposition unterzogen werden ($g$ größer als die Anzahl der Dokumente und damit auch Auswahlwahrscheinlichkeit $p$ beliebig), da die Größe der Testkollektion noch keine Laufzeitprobleme erwarten läßt.

Die Wahl von $k = 50$ stellt hierbei nur einen gewissen Kompromiß dar. Die anderen drei Clustering-Verfahren bekommen jeweils eine Vorgabe von höchstens 30 Ästen pro Knoten der Hierarchie, um eine gute interaktive Navigation zu ermöglichen. Für *Latent Semantic Indexing*, dessen Prinzip sich wie beschrieben auch hinter dem spektralen Clustering verbirgt, wird in der Literatur aber empfohlen, eine größere Anzahl von Singularwerten zu betrachten (in [BB99] etwa werden 100 bis 300 vorgeschlagen). Dementsprechend ist der Wert hier zumindest etwas erhöht worden.

Das spektrale Clustering ist das einzige Verfahren, bei dem auf eine *vorherige* Dimensionsreduktion der Vektoren verzichtet werden kann, da der Algorithmus das *Latent Semantic Indexing* selbst direkt benutzt. Die Ergebnisse bei Variation von Norm und Gewichtungsfunktion finden sich in Tabelle 5.8. Deutlich ist zu sehen, daß alle drei Maße ihre Bestwerte bei *TFIDF*-Gewichtung und euklidisch normierten Vektoren annehmen. Beim Vergleich der beiden Normierungen ist kein einheitliches Bild möglich: Die $L_1$-Norm liefert zwar ohne Gewichtungsfunktion die besseren Werte, aber die $L_2$-Norm gewinnt sowohl bei binärer als auch bei *TFIDF*-Gewichtung.

Tabelle 5.8: Spektrales Clustering: Ohne vorherige Dimensionsreduktion

| **Gewichtung** | ***Purity*** | | ***Entropy*** | | ***F-Measure*** | |
|---|---|---|---|---|---|---|
| | $L_1$ | $L_2$ | $L_1$ | $L_2$ | $L_1$ | $L_2$ |
| Ohne | 0,358 | 0,337 | 0,665 | 0,669 | 0,389 | 0,311 |
| Binaer | 0,302 | 0,310 | 0,723 | 0,714 | 0,210 | 0,229 |
| TFIDF | 0,386 | **0,410** | 0,640 | **0,604** | 0,407 | **0,414** |

Unterzieht man die Vektoren vor dem Clustering einer *CBW*-Reduktion, so ergibt sich sozusagen eine doppelte Dimensionsreduktion, deren Resultate in Tabelle 5.9 aufgeführt sind. Interessant ist zum einen, daß bei binärer Gewichtung eher die $L_1$-Norm die bessere Wahl zu sein scheint, wohingegen die $L_2$-Norm bei *TFIDF*-Gewichtung deutlich vorne liegt. Ohne Gewichtung ist kein Sieger auszumachen. Weiter fällt auf, daß bei allen Gewichtungen die Bestwerte bei einem *CBW*-Schwellwert von $c = 0,4$ erzielt werden und diese (mit einer Ausnahme) sämtlich besser sind als ohne zusätzliche Dimensionreduktion.

Tabelle 5.9: Spektrales Clustering: Dimensionsreduktion mittels *CBW*

| c | ***Purity*** | | ***Entropy*** | | ***F-Measure*** | |
|---|---|---|---|---|---|---|
| | $L_1$ | $L_2$ | $L_1$ | $L_2$ | $L_1$ | $L_2$ |
| 0,4 | 0,429 | **0,439** | 0,597 | **0,583** | **0,452** | 0,411 |
| 0,5 | 0,386 | 0,429 | 0,640 | 0,598 | 0,394 | 0,397 |
| 0,6 | 0,366 | 0,399 | 0,659 | 0,636 | 0,410 | 0,387 |

(a) Ohne Gewichtung

| c | ***Purity*** | | ***Entropy*** | | ***F-Measure*** | |
|---|---|---|---|---|---|---|
| | $L_1$ | $L_2$ | $L_1$ | $L_2$ | $L_1$ | $L_2$ |
| 0,4 | **0,413** | 0,350 | **0,629** | 0,670 | **0,445** | 0,340 |
| 0,5 | 0,338 | 0,307 | 0,699 | 0,699 | 0,295 | 0,290 |
| 0,6 | 0,328 | 0,324 | 0,699 | 0,693 | 0,295 | 0,281 |

(b) Binäre Gewichtung

| c | ***Purity*** | | ***Entropy*** | | ***F-Measure*** | |
|---|---|---|---|---|---|---|
| | $L_1$ | $L_2$ | $L_1$ | $L_2$ | $L_1$ | $L_2$ |
| 0,4 | 0,483 | **0,487** | **0,549** | **0,549** | 0,530 | **0,542** |
| 0,5 | 0,442 | 0,475 | 0,599 | **0,549** | 0,422 | 0,483 |
| 0,6 | 0,379 | 0,448 | 0,642 | 0,575 | 0,405 | 0,434 |

(c) *TFIDF*-Gewichtung

Die wichtigste Beobachtung ist aber die Position der global besten Qualitätswerte: Alle drei Maße bewerten die Ergebnisse mit *CBW*-Schwellwert $c = 0,4$, euklidischer Normierung und *TFIDF*-Gewichtung am höchsten. Damit ist diese Parametrisierung als klarer Sieger auszumachen und wird daher als der weiter zu untersuchende Kandidat ausgewählt.

Bei der Betrachtung der entsprechenden Hierarchie fällt auf, daß bereits der Wurzelknoten (siehe Tabelle 5.10) deutlich weniger als die zugelassenen $k = 50$ Untermengen enthält. Das deutet daraufhin, daß der Algorithmus zur Singularwert-Dekomposition nicht mehr Singularwerte berechnen konnte (entweder weil nicht mehr existieren oder zumindest sich nicht in der geforderten Genauigkeit haben berechnen lassen). Dies zeigt sich auch an weiteren Stellen der Hierarchie: Teilweise enthalten Blätter deutlich mehr als die eigentlich zugelassenen $m = 50$ Dokumente, weil der Algorithmus sie nicht mehr weiter aufteilen konnte.

Tabelle 5.10: Spektrales Clustering: Wurzel der Hierarchie

| ***Nr.*** | ***Zusammenfassung*** | ***Äste*** | ***Dokum.*** | ***Pur.*** | ***Entr.*** |
|---|---|---|---|---|---|
| 1 | `math java mathemat univers` | 28 | 1947 | 0.577 | 0.567 |
| 2 | `casino review game gambl` | 24 | 1610 | 0.418 | 0.456 |
| 3 | `review java movi math` | 18 | 4679 | 0.256 | 0.769 |
| 4 | `review casino java movi` | 25 | 1007 | 0.270 | 0.805 |
| 5 | `javascript java program softwar` | 25 | 1778 | 0.424 | 0.700 |
| 6 | `parent children child adopt` | 17 | 3027 | 0.409 | 0.601 |
| 7 | `game program java javascript` | 19 | 4945 | 0.627 | 0.521 |
| 8 | `comic game garden casino` | 18 | 1845 | 0.512 | 0.456 |
| 9 | `garden comic plant golf` | 23 | 1810 | 0.804 | 0.309 |
| 10 | `golf comic game movi` | 20 | 389 | 0.661 | 0.395 |
| 11 | `golf movi film garden` | 14 | 1282 | 0.472 | 0.361 |
| 12 | `film game golf movi` | 30 | 894 | 0.834 | 0.242 |
| 13 | `museum mathemat math garden` | 5 | 1220 | 0.888 | 0.200 |
| 14 | `mathemat object math univers` | 10 | 1976 | 0.900 | 0.176 |
| 15 | `object mathemat program movi` | 33 | 1183 | 0.392 | 0.731 |
| 16 | `movi film review charact` | 17 | 4091 | 0.982 | 0.041 |
| 17 | `server movi film math` | 15 | 1488 | 0.256 | 0.787 |

Ganz hervorragend scheint das spektrale Clustering eine Film-Kategorie (Nummer 16) ausgemacht zu haben: Den Werten nach ist etwa die Hälfte aller Dokumente aus dem Bereich „Film" direkt auf höchster Ebene dort einsortiert worden. Themenfremde Dokumente liegen dagegen kaum darin. Andererseits gibt es eine Reihe von Kategorien, die nicht besonders gut getroffen sind. Sehr oft findet man auch in den Blättern Dokumente aus ziemlich vielen, gar nicht verwandten Bereichen. Dazu kommt, daß die Zusammenfassungen nicht besonders gut sind. Wie in Abschnitt 3.2.2 beschrieben, werden sie bei diesem Algorithmus nicht über Zentroide, sondern mittels der Konzeptvektoren erstellt. Dies scheint subjektiv nicht wirklich überzeugend, denn in vielen Zusammenfassungen auf einer Ebene kommen immer wieder dieselben Wörter vor. Um abschließend beurteilen zu können, ob das an den Konzeptvektoren oder an der Gruppierung der Dokumente liegt, müßte allerdings noch ein direkter Vergleich mit zentroidbasierten Zusammenfassungen vorgenommen werden.

Zum Abschluß sei noch ein Beispiel für eine Kategorie angeführt, deren hohe Güte von keinem der drei Qualitätsmaße erkannt wird. Die Nummer 17 der Wurzel (Tabelle 5.10) scheint den Bewertungen nach eine der schlechteren zu sein. Schaut man aber hinein, so stellt sich heraus, daß grob geschätzt etwa 80 bis 90 Prozent der sich darin befindenden Dokumente die in Tabelle 5.11 beispielhaft zu sehende Gestalt haben. Leider spiegelt die textuelle Zusammenfassung dies nicht im geringsten wieder.

Tabelle 5.11: Spektrales Clustering: Gut gefunden, aber schlecht bewertet (Ausschnitt)

| ***Nr.*** | ***Zusammenfassung*** | ***Thema*** |
|---|---|---|
| ... | ... | ... |
| 11 | `302 FoundFound The document has moved here.` | Science/Chemistry |
| 12 | `301 Moved PermanentlyMoved Permanent` | Science/Math |
| 13 | `301 Moved PermanentlyMoved Permanen` | Science/Math |
| 14 | `301 Moved PermanentlyMoved Permanently` | Society/Philosophy |
| 15 | `MO Systems Solutions, Inc - PowerBuilde` | Computers/Programming |
| 16 | `301 Moved PermanentlyMoved Perman` | Arts/Movies |
| 17 | `301 Moved PermanentlyMoved Permanently Th` | Computers/Programming |
| 18 | `302 FoundFound The document has moved he` | Business/Venture_Capital |
| 19 | `301 Moved PermanentlyMoved Permane` | Arts/Movies |
| 20 | `301 Moved PermanentlyMoved Permanently The` | Computers/Programming |
| 22 | `301 Moved PermanentlyMoved Permanentl` | Home/Family |
| 23 | `301 Moved PermanentlyMoved Permanent` | Home/Family |
| 24 | `301 Moved PermanentlyMoved Permanently The doc` | Home/Family |
| 25 | `301 Moved PermanentlyMoved Permanently The docum` | Science/Math |
| 26 | `301 Moved PermanentlyMoved Permanently` | Computers/Programming |
| 27 | `301 Moved PermanentlyMoved Permanently T` | Home/Gardens |
| ... | ... | ... |

### 5.2.3 DC-Bäume

Für die Bewertung des auf *DC-Bäumen* basierenden Clustering-Verfahrens sind folgende Parameter fixiert worden:

- Der Schwellwert $t_2 = 0,4$.
- Der Verzweigungsfaktor $b = 30$.
- Der Sollwert $m = 15$ für die minimale Zahl von Nachfolgern.

Für den Schwellwert $t_1$ sind zunächst jeweils die beiden Werte $0,9$ und $1,0$ gewählt worden. Die Ergebnisse bei Dimensionsreduktion mittels *Content Bearing Words* und Cosinus-Maß sind in Tabelle 5.12 aufgeführt. Die Unterschiede zwischen den beiden

Tabelle 5.12: DC-Bäume: Dimensionsreduktion mittels *CBW* und Cosinus-Maß

| c | ***Purity*** | | ***Entropy*** | | ***F-Measure*** | |
|---|---|---|---|---|---|---|
| | ***0,9*** | ***1,0*** | ***0,9*** | ***1,0*** | ***0,9*** | ***1,0*** |
| 0,4 | 0,315 | 0,547 | 0,721 | 0,483 | 0,314 | **0,509** |
| 0,5 | 0,315 | **0,554** | 0,719 | **0,473** | 0,315 | 0,481 |
| 0,6 | 0,317 | 0,551 | 0,719 | 0,477 | 0,310 | 0,459 |

(a) Ohne Gewichtung

| c | ***Purity*** | | ***Entropy*** | | ***F-Measure*** | |
|---|---|---|---|---|---|---|
| | ***0,9*** | ***1,0*** | ***0,9*** | ***1,0*** | ***0,9*** | ***1,0*** |
| 0,4 | 0,330 | 0,541 | 0,702 | **0,472** | 0,311 | 0,368 |
| 0,5 | 0,329 | **0,544** | 0,704 | 0,479 | 0,307 | **0,428** |
| 0,6 | 0,321 | 0,537 | 0,714 | 0,483 | 0,316 | 0,377 |

(b) Binäre Gewichtung

| c | ***Purity*** | | ***Entropy*** | | ***F-Measure*** | |
|---|---|---|---|---|---|---|
| | ***0,9*** | ***1,0*** | ***0,9*** | ***1,0*** | ***0,9*** | ***1,0*** |
| 0,4 | 0,309 | 0,565 | 0,726 | 0,461 | 0,317 | **0,509** |
| 0,5 | 0,305 | **0,582** | 0,730 | **0,441** | 0,305 | 0,507 |
| 0,6 | 0,303 | 0,567 | 0,733 | 0,449 | 0,307 | 0,452 |

(c) *TFIDF*-Gewichtung

Werten für $t_1$ sind drastisch: Ohne Ausnahme zeigt die Wahl von $t_1 = 1,0$ die deutlich besseren Ergebnisse. Das Interessante hieran ist, daß ein Wert von $t_1 = 1,0$ die Schwelle faktisch deaktiviert, denn nach der Definition eines Distanzmaßes ist jeder Abstand kleiner oder gleich $1,0$, und damit kann jeder neu eingefügte Dokumentvektor jederzeit einen Ast hinuntergereicht werden. Was passiert, wenn der Schwellwert bei $t_1 = 0.9$ aktiv wird, zeigt beispielhaft die Tabelle 5.13: Die Form der Hierarchie degeneriert völlig. Das liegt vermutlich daran, daß nun Dokumente oft nicht mehr hinuntergereicht werden können und daher direkt in der Wurzel (oder zumindest weit oben in der Hierarchie) hängen bleiben. Dadurch wird die Wurzel häufig aufgeteilt. Das aber ist in Bezug auf die Form der Hierarchie problematisch, denn dadurch entstehen zwei neue Knoten auf zweiter Ebene, die alle Dokumente enthalten, die bisher schon im Baum gewesen sind. Auf diese Art verschiebt sich der ganze Baum eine Stufe nach unten. Die nun freien Plätze in der Wurzel werden dann (oft) wieder mit Einzeldokumenten gefüllt, bis der Knoten schließlich wieder geteilt werden muß. Als Nebeneffekt wird der Baum sehr groß, und das Clustering dauert lange. Dieser Effekt tritt – wie eine kurze Stichprobe ergeben hat – im übrigen auch bei Werten $0,9 < t_1 < 1,0$ auf (selbst bei $t_1 = 0,99!$). Zumindest wenn man die entstehende Hierarchie direkt zur Navigation verwenden möchte, scheint ein aktiver Schwellwert $t_1$ daher nicht geeignet. Ein kurzer Test mit der Manhattan-Distanz hat dies ebenfalls bestätigt. Aus Zeitgründen sind für dieses Maß allerdings nicht alle Variationen durchprobiert worden und die Ergebnisse daher hier nicht aufgeführt.

Tabelle 5.13: DC-Bäume: Degenerierter Baum bei $t_1 = 0,9$

| ***Nr.*** | ***Zusammenfassung*** | ***Äste*** | ***Dokum.*** | ***Pur.*** | ***Entr.*** |
|---|---|---|---|---|---|
| ... | ... | ... | ... | ... | ... |
| 9 | `black` | 1 | 1 | 1.000 | 0.000 |
| 10 | `medic reason th` | 1 | 1 | 1.000 | 0.000 |
| 11 | `nicklau golfer jack option` | 1 | 1 | 1.000 | 0.000 |
| 12 | `golf club woodburi par` | 4 | 4 | 1.000 | 0.000 |
| 13 | `gc problem version` | 1 | 1 | 1.000 | 0.000 |
| 14 | `rand objectivist objectiv ayn` | 8 | 8 | 1.000 | 0.000 |
| 15 | `server movi film object` | 18 | 36575 | 0.211 | 0.820 |
| 16 | `de plato confuciu india` | 13 | 17 | 0.941 | 0.070 |
| ... | ... | ... | ... | ... | ... |

Der weitere Vergleich beschränkt sich also auf die mit $t_1 = 1$ erzeugten Ergebnisse. Es fällt auf, daß bei allen Gewichtungen die Qualitätsmaße ihre besten Werte jeweils bei einem *CBW*-Schwellwert von $c = 0,4$ oder $c = 0,5$ erreichen. Die insgesamt höchsten Bewertungen erreicht die *TFIDF*-Gewichtung. Dabei sind *Purity* und *Entropy* bei $c = 0,5$ am besten, *F-Measure* bei $c = 0,4$. Mit Priorität für die ersten beiden scheint bei Benutzung der *CBW*-Reduktion also eine *TFIDF*-Gewichtung mit $c = 0,5$ eine gute Wahl zu sein.

Tabelle 5.14: DC-Bäume: Dimensionsreduktion mittels *LSI* und Cosinus-Maß

| **Gewichtung** | ***Purity*** | | ***Entropy*** | | ***F-Measure*** | |
|---|---|---|---|---|---|---|
| | ***0,9*** | ***1,0*** | ***0,9*** | ***1,0*** | ***0,9*** | ***1,0*** |
| Ohne | 0,339 | 0,397 | 0,690 | 0,627 | 0,315 | 0,328 |
| Binaer | 0,345 | **0,456** | 0,687 | **0,566** | 0,330 | **0,377** |
| TFIDF | 0,329 | 0,400 | 0,704 | 0,628 | 0,318 | 0,315 |

Zum Vergleich finden sich in Tabelle 5.14 die Ergebnisse bei Dimensionsreduktion mittels *Latent Semantic Indexing*. Auch hier ist ein Unterschied zwischen den beiden Schwellwerten zu erkennen, der allerdings nicht ganz so deutlich ausfällt. Das liegt vermutlich daran, daß die Dimension der per *LSI* erzeugten Vektoren deutlich niedriger ist als bei dem *CBW*-Verfahren, so daß die Dokumentvektoren einander ähnlicher werden. Auf alle Fälle liegen die mittels *LSI* erzielten Qualitätswerte aber deutlich unter den zuvor betrachteten. Als Untersuchungskanditat sei daher die mit den oben erwähnten *CBW*-Parametern erzeugte Hierarchie ausgewählt.

In Tabelle 5.15 findet sich der Wurzelknoten der erzeugten Hierarchie. Deutlich ist die besondere Struktur zu sehen, die der Algorithmus mit einem Schwellwert von $t_1 = 1$ erzeugt: Nur drei Kategorien sind vorhanden. Das liegt wohl daran, daß jedes neue Dokument sofort einen der Äste hinuntergereicht wird. Vergrößert wird die Wurzel erst, wenn aufgrund des rekursiven Teilens in einem der Äste einer der direkten Nachfolger genügend oft sein Größenlimit erreicht hat. Dies wird umso unwahrscheinlicher desto größer der Baum wird. Zur Navigation ist ein derartiger Hierarchiestart natürlich nicht sonderlich geeignet. Auch wenn zumindest die Unterkategorien 1 und 2 scheinbar eindeutige Titel tragen, zeigen die Bewertungen, daß sich viele unterschiedliche Dokumente in den Zweigen befinden müssen (genau das, was der Schwellwert $t_1$ eigentlich hätte verhindern sollen).

Tabelle 5.15: DC-Bäume: Wurzel der Hierarchie

| ***Nr.*** | ***Zusammenfassung*** | ***Äste*** | ***Dokum.*** | ***Pur.*** | ***Entr.*** |
|---|---|---|---|---|---|
| 1 | `movi film review video` | 17 | 9403 | 0.568 | 0.510 |
| 2 | `mathemat univers film depart` | 17 | 10459 | 0.287 | 0.720 |
| 3 | `server object museum game` | 22 | 16801 | 0.268 | 0.790 |

Folgt man allerdings einem der drei Äste, finden sich auf der zweiten Ebene relativ gute Unterkategorien. In Tabelle 5.16 ist beispielsweise Kategorie 2 dargestellt. Gut erkennen lassen sich etwa Bereiche zur Mathematik (Nr. 2), Chemie (Nr. 4), zum Film (Nr. 5) oder zur Philosophie (Nr. 11), die den Bewertungen nach alle nur wenige thematisch fremde Dokumente enthalten.

Tabelle 5.16: DC-Bäume: Kategorie 2 der Wurzel (Ausschnitt)

| ***Nr.*** | ***Zusammenfassung*** | ***Äste*** | ***Dokum.*** | ***Pur.*** | ***Entr.*** |
|---|---|---|---|---|---|
| 1 | `math graph softwar problem` | 15 | 432 | 0.676 | 0.389 |
| 2 | `number prime algebra theori` | 16 | 483 | 0.901 | 0.172 |
| 3 | `statist depart research pharmaci` | 16 | 461 | 0.607 | 0.470 |
| 4 | `chemistri nmr chemic magnet` | 18 | 579 | 0.855 | 0.215 |
| 5 | `film festiv movi omh` | 22 | 623 | 0.944 | 0.103 |
| 6 | `gross star movi htm` | 30 | 1112 | 0.687 | 0.405 |
| 7 | `climb mountain comic stori` | 24 | 647 | 0.365 | 0.561 |
| 8 | `rose manga list plant` | 23 | 877 | 0.336 | 0.577 |
| 9 | `don book re issu` | 20 | 532 | 0.353 | 0.699 |
| 10 | `mind mean word truth` | 18 | 419 | 0.573 | 0.500 |
| 11 | `human moral natur ethic` | 20 | 558 | 0.842 | 0.242 |
| ... | ... | ... | ... | ... | ... |

### 5.2.4 K-Bäume

Bei dem *K-Baum*-Clustering sind folgende fixe Parameter zum Einsatz gekommen:

- Der Schwellwert $t = 0,4$.
- Der Verzweigungsfaktor $b = 30$.
- Die Höchstgrenze $j = 30$ für die Anzahl an *K-Means*-Iterationen.
- Die Höchstzahl $m = 100$ für die Anzahl von Nachfolgern eines Knotens.

Die Wahl von $m = 100$ ist recht hoch, denn auf diese Art kann ein dynamischer Knoten bis 100 Nachfolger erhalten. Der Grund für diese Wahl liegt in dem zur Knotenumwandlung benutzten *K-Means*-Algorithmus: Bei 100 Vektoren sollte er in in der Lage sein, relativ gute Zentroide bestimmen zu können.

Die *K-Baum*-Ergebnisse bei Dimensionsreduktion mittels *Content Bearing Words* finden sich in Tabelle 5.17. Da diese Clustering-Methode durch den Einsatz des kanonischen *K-Means* zur Knotenumwandlung auch ein randomisiertes Verfahren darstellt, sind die aufgeführten Werte wieder die Mittelwerte aus jeweils drei Läufen.

Deutlich zeigt sich, daß das Cosinus-Maß generell bessere Werte liefert als die Manhattan-Distanz. Bei jeder der drei Gewichtungen werden die mit Abstand besten Bewertungen jeweils durch das Cosinus-Maß erzielt. Etwas undeutlich ist die beste Wahl für den *CBW*-Schwellwert $c$: Je nach Gewichtung liefert 0,4 (ohne Gewichtung), 0,5 (binär) oder 0,6 (*TFIDF*) die besseren Resultate bei *Purity* und *Entropy*. Bei *F-Measure* schwankt es noch einmal zusätzlich. Der insgesamt beste *Purity*-Wert wird mit binärer Gewichtung erzielt, die besten *Entropy*- und *F-Measure*-Werte mit *TFIDF*.

Die geeignetste Parametrisierung bei der Reduktion mittels *Content Bearing Words* scheint daher das Cosinus-Maß mit *TFIDF*-Gewichtung und Schwellwert $c = 0,6$ zu sein. Neben den besten Werten bei zwei der drei Qualitätsmaße nimmt es auch beim dritten, *Purity*, immerhin den zweiten Platz ein. Den Vergleich mit der Dimensionsreduktion mittels *Latent Semantic Indexing* zeigt Tabelle 5.18. Hier liefert interessanterweise die Manhattan-Distanz mit *TFIDF*-Gewichtung die höchste Bewertung bei allen drei Maßen. Allerdings kommen die Werte nicht an die der *CBW*-Reduktion heran. Daher sei die dort beschriebene Parametrisierung zur weiteren Untersuchung ausgewählt.

Um einen Eindruck von dem Ergebnis bekommen zu können, soll wieder eine der drei mit diesen Parametern erzeugten Hierarchien etwas näher untersucht werden. Analog zu dem Vorgehen beim *Simple Single Pass K-Means* sei dafür wieder diejenige genommen, deren Bewertung am dichtesten an den Mittelwerten liegt. In Tabelle 5.19 ist dies die dritte.

Tabelle 5.17: K-Bäume: Dimensionsreduktion mittels *Content Bearing Words*

| c | ***Purity*** | | ***Entropy*** | | ***F-Measure*** | |
|---|---|---|---|---|---|---|
| | $L_1$ | $L_2$ | $L_1$ | $L_2$ | $L_1$ | $L_2$ |
| 0,4 | 0,433 | **0,664** | 0,614 | **0,363** | 0,482 | **0,533** |
| 0,5 | 0,627 | 0,649 | 0,393 | 0,370 | 0,495 | 0,505 |
| 0,6 | 0,599 | 0,641 | 0,417 | 0,374 | 0,461 | **0,533** |

(a) Ohne Gewichtung

| c | ***Purity*** | | ***Entropy*** | | ***F-Measure*** | |
|---|---|---|---|---|---|---|
| | $L_1$ | $L_2$ | $L_1$ | $L_2$ | $L_1$ | $L_2$ |
| 0,4 | 0,610 | 0,658 | 0,409 | 0,367 | 0,431 | **0,537** |
| 0,5 | 0,590 | **0,698** | 0,426 | **0,342** | 0,445 | 0,455 |
| 0,6 | 0,556 | 0,670 | 0,457 | 0,370 | 0,359 | 0,445 |

(b) Binäre Gewichtung

| c | ***Purity*** | | ***Entropy*** | | ***F-Measure*** | |
|---|---|---|---|---|---|---|
| | $L_1$ | $L_2$ | $L_1$ | $L_2$ | $L_1$ | $L_2$ |
| 0,4 | 0,626 | 0,680 | 0,441 | 0,343 | 0,525 | 0,555 |
| 0,5 | 0,639 | 0,676 | 0,378 | 0,339 | 0,506 | 0,615 |
| 0,6 | 0,608 | **0,683** | 0,406 | **0,334** | 0,477 | **0,626** |

(c) *TFIDF*-Gewichtung

In Tabelle 5.20 ist ein Ausschnitt aus dem Wurzelknoten zu sehen. Sehr schön und direkt auf oberster Ebene erkennbar sind Dokumente z. B. zu den Themen Glücksspiel (Nr. 16) und Film (Nr. 21). Folgt man dann etwa dem Thema Film, lassen sich auch gelungene Unterkategorien erkennen: Tabelle 5.21 zeigt beispielsweise weitere Zweige zu den Regisseuren Alfred Hitchcock (Nr. 6) und Stanley Kubrick (Nr. 11) oder zum Thema DVD (Nr. 3; interessanterweise gleich in Verbindung mit „Hacker“). Ein Blick hinein zeigt, daß die darin liegenden Dokumente größtenteils auch tatsächlich gut zu den Zusammenfassungen passen.

Aber die Tabelle 5.21 zeigt auch eine nicht so schöne Eigenschaft, die sich an mehreren Stellen in der Gesamthierarchie wiederfindet. Die mit Abstand meisten Dokumente enthält Unterkategorie 13 – und zwar mit derselben Zusammenfassung versehen, die auch die gesamte Kategorie in der Wurzel besitzt. Das *K-Baum*-Verfahren scheint etwas ungleichmäßige Hierarchien zu erzeugen. Auf den einzelnen Ebenen sind viele kurze Äste mit relativ wenigen Dokumenten und ein paar wenige, die in die Tiefe gehen. Dafür

Tabelle 5.18: K-Bäume: Dimensionsreduktion mittels *Latent Semantic Indexing*

| **Gewichtung** | ***Purity*** | | ***Entropy*** | | ***F-Measure*** | |
|---|---|---|---|---|---|---|
| | $L_1$ | $L_2$ | $L_1$ | $L_2$ | $L_1$ | $L_2$ |
| Ohne | 0,524 | 0,480 | 0,489 | 0,531 | 0,337 | 0,301 |
| Binär | 0,498 | 0,492 | 0,510 | 0,518 | 0,323 | 0,284 |
| TFIDF | **0,557** | 0,517 | **0,444** | 0,496 | **0,477** | 0,378 |

Tabelle 5.19: K-Bäume: Einzelergebnisse der ausgewählten Parametrisierung

| **Nr.** | ***Purity*** | ***Entropy*** | ***F-Measure*** |
|---|---|---|---|
| 1 | 0,686 | 0,332 | 0,617 |
| 2 | 0,680 | 0,337 | 0,642 |
| 3 | 0,683 | 0,333 | 0,619 |
| Mittel | 0,683 | 0,334 | 0,626 |

sind aber gerade in den kurzen Ästen die Dokumente oft sehr gut zusammengefaßt. Der Grund für diese Form des Baumes dürfte darin liegen, daß nach der Umwandlung eines dynamischen Knotens in einen statischen ein neues Dokument immer den nächstgelegenen Ast hinuntergereicht wird. Um so mehr Dokumente so einem speziellen Ast folgen, für um so mehr Dokumente stellt dessen Zentroid daher einen Mittelwert dar. Der Zentroid wird also immer allgemeiner und damit wiederum ähnlicher zu neuen Dokumenten, die ebenfalls die entsprechende Hierarchiestufe erreichen.

Tabelle 5.20: K-Bäume: Wurzel der Hierarchie (Ausschnitt)

| ***Nr.*** | ***Zusammenfassung*** | ***Äste*** | ***Dokum.*** | ***Pur.*** | ***Entr.*** |
|---|---|---|---|---|---|
| ... | ... | ... | ... | ... | ... |
| 15 | `support view displai requir` | 30 | 887 | 0.166 | 0.857 |
| 16 | `casino game gambl win` | 30 | 1168 | 0.802 | 0.297 |
| 17 | `object orient class method` | 30 | 817 | 0.326 | 0.776 |
| 18 | `webr fan don godzilla` | 30 | 300 | 0.383 | 0.592 |
| 19 | `feb bond jan directori` | 30 | 370 | 0.208 | 0.810 |
| 20 | `game star war rpg` | 30 | 1626 | 0.610 | 0.376 |
| 21 | `movi film review video` | 30 | 6488 | 0.882 | 0.197 |
| 22 | `java program code file` | 30 | 4875 | 0.701 | 0.380 |
| 23 | `request modern member read` | 30 | 472 | 0.273 | 0.771 |
| 24 | `garden museum plant rose` | 30 | 2650 | 0.508 | 0.344 |
| 25 | `mathemat math depart univers` | 30 | 4290 | 0.599 | 0.363 |
| 26 | `golf de cours club` | 30 | 1044 | 0.556 | 0.544 |
| 27 | `parent adopt child famili` | 29 | 2533 | 0.451 | 0.542 |
| 28 | `strip comic archiv bloom` | 30 | 155 | 0.819 | 0.268 |
| ... | ... | ... | ... | ... | ... |

Tabelle 5.21: K-Bäume: Kategorie „Film“ (Ausschnitt)

| ***Nr.*** | ***Zusammenfassung*** | ***Äste*** | ***Dokum.*** | ***Pur.*** | ***Entr.*** |
|---|---|---|---|---|---|
| ... | ... | ... | ... | ... | ... |
| 3 | `dvd hacker video movi` | 32 | 44 | 0.864 | 0.202 |
| 4 | `godzilla monster movi toho` | 6 | 20 | 1.000 | 0.000 |
| 5 | `actual halloween jack count` | 10 | 12 | 0.500 | 0.464 |
| 6 | `hitchcock sam psycho film` | 46 | 63 | 0.968 | 0.051 |
| 7 | `clan kindr avalon braveheart` | 10 | 10 | 0.800 | 0.157 |
| 8 | `dream hoop arthur gump` | 23 | 26 | 0.692 | 0.362 |
| 9 | `bride princess westlei buttercup` | 23 | 33 | 0.909 | 0.127 |
| 10 | `sai lola kid bad` | 14 | 15 | 0.600 | 0.339 |
| 11 | `kubrick lolita film stanlei` | 46 | 63 | 0.905 | 0.153 |
| 12 | `gladiat blade runner maximu` | 22 | 40 | 0.925 | 0.084 |
| 13 | `movi film review video` | 30 | 4587 | 0.966 | 0.071 |
| 14 | `matter call left segment` | 8 | 13 | 0.308 | 0.593 |
| 15 | `warrior evil movi monster` | 20 | 20 | 0.650 | 0.314 |
| 16 | `urban legend scream travi` | 27 | 37 | 0.730 | 0.332 |
| 17 | `film festiv cinema filmmak` | 30 | 412 | 0.966 | 0.065 |
| ... | ... | ... | ... | ... | ... |

## 5.3 Vergleich und Folgerung

Nachdem die Ergebnisse der verschiedenen Verfahren für sich vorgestellt worden sind, kann nun ein abschließender Vergleich stattfinden. Da alle Verfahren ihre besten Bewertungen mittels *CBW*-Dimensionsreduktion erzielt haben, sollen die damit erzielten Ergebnisse den Schwerpunkt bilden. In Abbildung 5.2 sind die mit den drei *CBW*-Schwellwerten jeweils erzielten Qualitätswerte noch einmal graphisch gegenübergestellt.

Das beste Ergebnis liefert demnach das *K-Baum*-Verfahren. Bei allen Kombinationen aus Qualitätsmaß und *CBW*-Schwellwert liegt es deutlich vorne. Den nach den Qualitätswerten zweiten Platz nimmt bei Priorität für *Purity* und *Entropy* knapp das *DC-Baum*-Verfahren ein. Es liegt bei den beiden Maßen zwar in etwa gleich mit *K-Means*, hat aber zusätzlich zumindest zweimal einen besseren *F-Measure*-Wert vorzuweisen. Der Abstand zu *K-Means* ist damit allerdings gering. Am schlechtesten schneidet das spektrale Verfahren ab, das zwar bei *F-Measure* noch mithalten kann, aber bei *Purity* und *Entropy* die schlechtesten Werte erzielt.

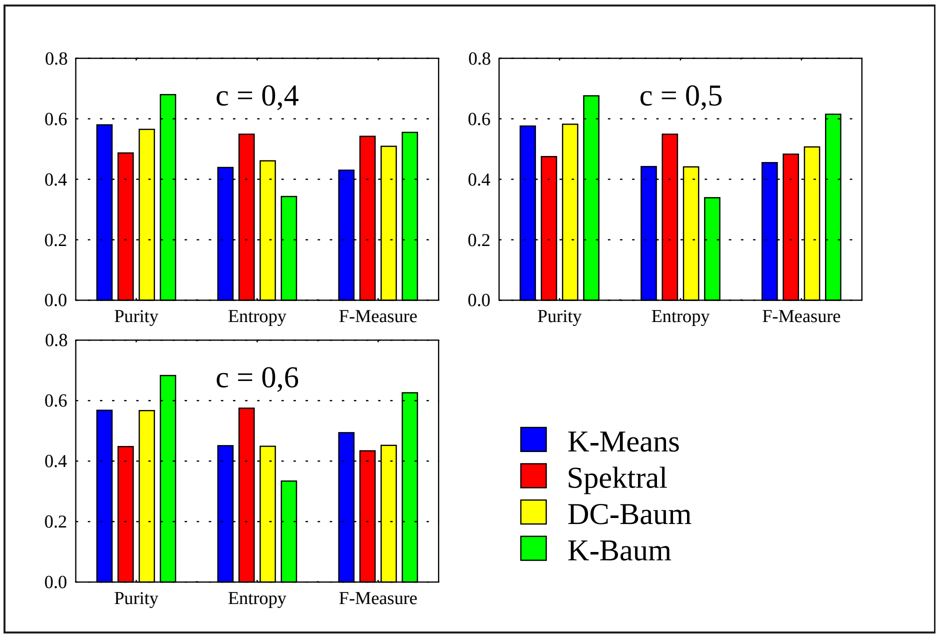



Abbildung 5.2: Vergleich der Qualitätswerte bei *CBW*-Reduktion

Wie sind die guten Ergebnisse des *K-Baum*-Verfahrens zu erklären? Der Algorithmus stellt eine Kombination aus dem *DC-Baum*-basierten Clustering und dem kanonischen *K-Means* dar und scheint von beiden gewisse Stärken zu übernehmen.

Eine Eigenschaft der *DC-Bäume* ist die Klassifikation neuer Dokumente mittels der Zentroide der einzelnen Äste. Ein einzufügendes Dokument wird an jedem Knoten jeweils den Ast hinuntergereicht, dessen Zentroid ihm an ähnlichsten ist. Diese Methode scheint gute Ergebnisse zu liefern. Zur Begründung sei die Situation einmal aus einem anderen Blickwinkel betrachtet. In [HK00] untersuchen Hal et. al. explizit die Klassifikation von Dokumenten mit Hilfe von Zentroiden. Sie bilden dafür zunächst eine Trainingsmenge von Dokumenten, die per Hand klassifiziert werden. Für jedes Thema werden anschließend die Zentroide aller Dokumentvektoren gebildet, die in derselben Kategorie plaziert worden sind (*Lernphase*). Anschließend fügen sie neue Dokumente automatisiert hinzu, indem sie für jeden Vektor jeweils die Kategorie wählen lassen, deren Zentroid ihm am nächsten liegt. Sie argumentieren, daß auf diese Art sehr gute Ergebnisse erzielt werden und überprüfen das auch experimentell. Dies ist aber genau die Methode, die auch die beiden inkrementellen Verfahren an jedem Knoten anwenden. Sie betrachten – in dieser Terminologie – die Dokumente, die sich schon in den jeweiligen Äste befinden, als Trainingsmenge und klassifizieren das neue Dokument entsprechend. Unter der Annahme einer bereits bestehenden guten Hierarchie ist damit zu erwarten, daß das Dokument an einer geeigneten Position landet.

Dies führt gleich zu einem Nachteil der *DC-Bäume*. Damit die Klassifikation über Zentroide funktioniert, muß die bestehende Hierarchie bereits sinnvoll aufgebaut sein. Das Problem liegt darin, daß der Algorithmus dies ohne Trainingsmenge selber zu erledigen hat. Zu Beginn kennt er nur einen leeren Baum, in den er sukzessive Dokumente einfügen soll. Gerade nach den ersten Dokumenten ist daher das Aussehen der Hierarchie gezwungenermaßen recht zufällig. Dennoch nimmt der Algorithmus einmal getroffene Entscheidungen nicht wieder zurück; abgesehen von der Aufteilung eines Knotens verändert kein Dokument mehr seine Position. Der *K-Baum*-Algorithmus besitzt diese Eigenschaft zwar auch, scheint dabei aber die bessere Methode des inkrementellen Aufbaus zu haben. Er reicht neue Dokumente auf jeden Fall die Hierarchie ganz hinunter, sammelt dann aber dort (genauer: auf der vorletzten Ebene) jeweils erst einmal eine ganze Reihe von Vektoren (in den Testläufen etwa 100). Bei diesen Vektoren steht ihre Position innerhalb der Hierarchie zunächst nur bis zu einem gewissen Grad fest. Die Position festigt sich, wenn der entsprechende Knoten seine Maximalgröße erreicht hat. Alle Dokumente, die sich unter ihm befinden, werden dann mittels *K-Means* auf eine relativ kleine Anzahl von Ästen (Verzweigungsfaktor) aufgeteilt. Die dahinter liegende Annahme ist, daß die Dokumente, die den dynamischen Knoten bis dahin erreicht haben, einen einigermaßen repräsentativen Ausschnitt aus all denen darstellen, die später ebenfalls zu dieser Stelle durchgereicht werden. Die Testergebnisse scheinen dies zu bestätigen.

Bei der Umwandlung eines Knotens wird der kanonische *K-Means* Algorithmus benutzt. Als eines der Standardverfahren sollte er zumindest einigermaßen gute Ergebnisse bei der Partitionierung einer Menge liefern. Seine quadratische Laufzeit ist dabei nicht von Belang, da nur vergleichsweise kleine Mengen bearbeitet werden müssen. Da die Qualitätswerte des Gesamtverfahrens sehr gut sind, scheint auch der *K-Means* seine Aufgabe besser zu erfüllen als die entsprechende Aufteilungsheuristik der *DC-Bäume*. Das mag unter anderem auch daran liegen, daß er deutlich mehr Eingabevektoren bekommt und damit mehr Informationen zur Verfügung hat. Andererseits läßt sich im *DC-Baum*-Verfahren nicht einfach der entsprechende Parameter, der Verzweigungsfaktor $b$, erhöhen, da dies Einfluß auf die Zahl der Äste an jedem Knoten der Hierarchie hätte. Beim *K-Baum*-Algorithmus wirkt sich der entsprechende Parameter, die Grenze $m$, nur auf die direkten Vorgänger der Blätter aus. Die Zahl der Nachfolger der statischen Knoten – die die Navigation deutlich mehr beeinflußt – bleibt dagegen begrenzt (wiederum durch den Verzweigungsfaktor $b$).

Ferner dürfte die Art und Weise, wie die Hierarchie wächst, Einfluß auf das Ergebnis haben. Bei einem *DC-Baum* werden neue Knoten immer oberhalb des zu teilenden eingefügt. Der Baum wächst damit „nach oben". In einem *K-Baum* dagegen landen neue Knoten immer unterhalb des vollen Knoten; der Baum wächst „nach unten" (vergleiche die Abbildungen 3.2 und 3.3).

Die Beurteilung des spektralen Verfahrens fällt schwierig aus. Aufgrund des völlig unterschiedlichen Ansatzes im Vergleich zu den restlichen Verfahren ist kaum zu sagen, warum das spektrale Clustering relativ schlecht abschneidet. Es ist allerdings zu vermuten, daß dies in dem *Latent Semantic Indexing* selber begründet ist. In Abbildung 5.3 sind einmal die Qualitätswerte der drei anderen Verfahren bei Dimensionsreduktion mit *LSI* sowie des spektralen Clusterings ohne vorherige Dimensionsreduktion gegenübergestellt. Zumindest zwischen *DC-Baum* und spektralem Verfahren sind die Unterschiede dabei nicht allzu groß. Generell läßt sich sagen, daß die mit *LSI* erzielten Bewertungen durchgängig hinter den *CBW*-Ergebnissen zurückstehen. Dies überrascht durchaus ein wenig, da in der Literatur *LSI* als eines der bewährten Standardverfahren anzusehen ist. Die *CBW*-Reduktion dagegen scheint, soweit bekannt, in dem Kontext des Clusterings von Dokumenten bisher nicht verwendet worden zu sein. Die hier vorgestellten Ergebnisse legen aber eine Überlegenheit nahe. Dazu kommt, daß sich dabei textuelle Zusammenfassungen leicht erzeugen lassen, wohingegen *LSI*-reduzierte Vektoren dies nicht mehr direkt erlauben. Interessant zu untersuchen wäre noch, inwieweit eine Kombination beider Verfahren Sinn macht. Immerhin erzielt das spektrale Clustering die besten Werte, wenn die Dimension der Eingabevektoren zuvor mit *CBW* reduziert worden ist.

Deutlich fällt die Einschätzung der anderen Parameter des Vektorraummodells aus: Unabhängig von dem Clustering-Verfahren zeigt die Kombination aus euklidischer Normierung der Vektoren und *TFIDF*-Gewichtung fast überall die besten Ergebnisse. Dies ist durchaus in Übereinstimmung mit der Literatur [SGM00] [BB99]. Der einzige Parameter, der schwankt, ist der Schwellwert der *CBW*-Reduktion; je nach Verfahren zeigen sich die Bestwerte bei $c = 0,4$, $c = 0,5$ oder $c = 0,6$.

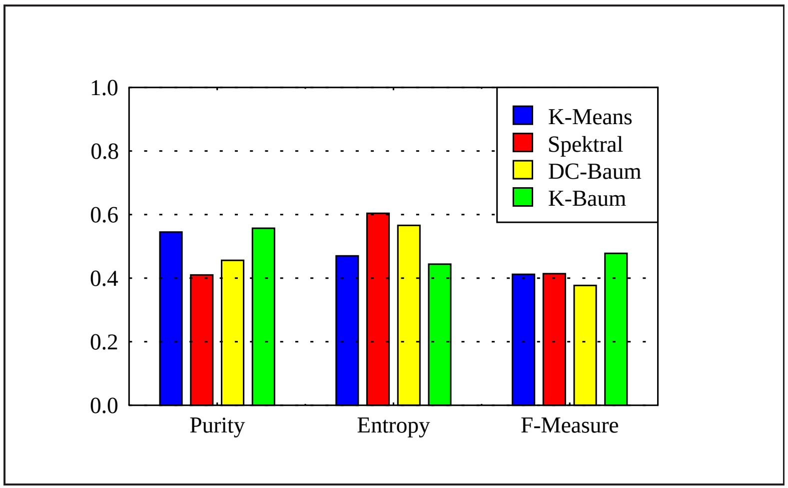



Abbildung 5.3: Vergleich der Qualitätswerte bei *LSI*-Reduktion

Nach dem Vergleich der erzielten Qualitätswerte fehlt noch ein Vergleich der Laufzeiten. Die Abbildung 5.4 zeigt einen Vergleich der Laufzeiten bei *CBW*-Reduktion. Wie bereits in Abschnitt 5.1.2 angedeutet, sind die Werte auf einem exklusiv zur Verfügung stehenden Rechner ermittelt worden. Es hat sich dabei um einen PC mit *Intel Pentium III* (800MHz) und 1GB RAM gehandelt. Da die Bibliothek, wie erwähnt, nicht auf Effizienz, sondern auf Flexibilität optimiert worden ist, sollen die absoluten Werte nicht genauer betrachtet werden. Interessanter ist der relative Vergleich der vier Verfahren. Während *K-Means*, *DC-Baum* und *K-Baum* in etwa jeweils gleiche Zeit benötigen, ist das spektrale Clustering mit Abstand am schnellsten. Das ist auch recht einfach zu erklären: Der größte Zeitaufwand wird beim spektralen Clustering für die Singularwert-Dekomposition benötigt; ansonsten geschieht nicht viel. Die Singularwert-Dekomposition ist zwar im allgemeinen teuer zu berechnen, aber bei der Größe der Testkollektion und der schon mit *CBW* in der Dimension reduzierten Vektoren hält sich die benötigte Zeit noch in einem recht niedrigen Rahmen. Zum Vergleich: Das Programm `lsi` (siehe Abschnitt 4.1.5) benötigt für die Dekomposition der nicht reduzierten Term-/Dokumentmatrix und anschließenden Projektion in den entstehenden Unterraum etwa 20 Minuten (mit einer Höchstgrenze von 100 Singularwerten)[4].

Die Abbildung 5.5 zeigt noch einen Vergleich der Laufzeiten bei *LSI*-reduzierten Vektoren. Dabei ist die Term-/Dokumentmatrix jeweils mit *TFIDF* gewichtet und euklidisch normiert worden. Sehr deutlich ist zu sehen, wie eine niedrige Dimension der Vektoren die Laufzeit der Verfahren verringert (vergleiche Tabelle 5.2). Dies ist natürlich ein

[4]Dabei ist allerdings zu beachten, daß die entsprechende, aus GTP [Gtp] stammende Implementierung, sehr wohl auf Geschwindigkeit optimiert ist.

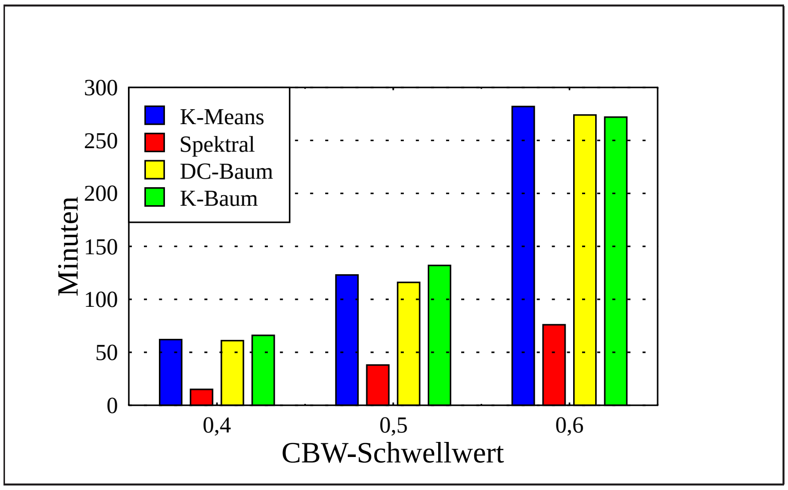



Abbildung 5.4: Laufzeiten bei *CBW*-Reduktion

deutlicher Vorteil von *LSI*. Allerdings ist beim Vergleich der Laufzeiten mit Tabelle 5.4 zu beachten, daß die Ausführungszeit für das Programm `lsi` zusätzlich benötigt wird.

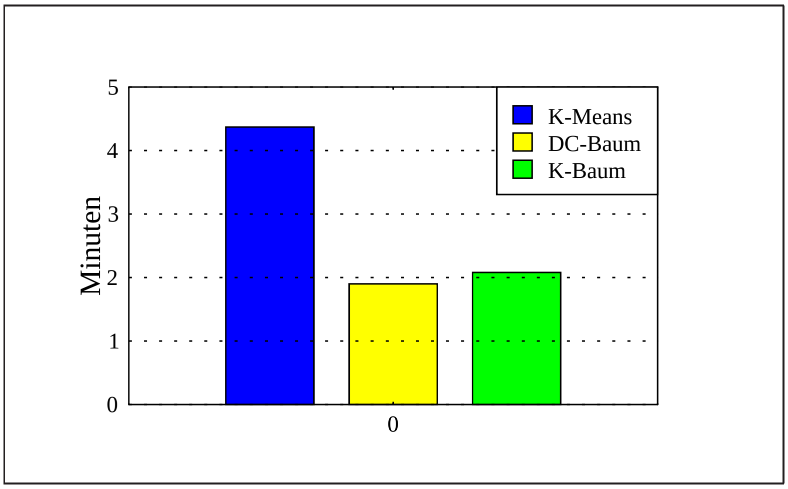



Abbildung 5.5: Laufzeiten bei *LSI*-Reduktion

Schwierig ist eine subjektive Einschätzung der erzeugten Hierarchien. Unabhängig vom Verfahren lassen sich in jedem Ergebnis sehr gute Kategorisierungen ausmachen, aber ebenso auch solche, die diesen Namen nicht einmal verdienen. Insbesondere die großen Kategorien (in der Testkollektion vor allem „Film“) lassen sich fast immer gut identifizieren. Die kleineren bis kleinsten dagegen sind zumindest am Beginn der Hierarchie fast durchgängig schwer bis gar nicht auszumachen.

Es ist kaum möglich, durch reines Navigieren in den Hierarchien einen einigermaßen gerechten Vergleich zu erstellen. Dies ist zu erwarten gewesen, weshalb die Bewertung mittels Qualitätsmaßen eingeführt worden ist. Dennoch ist grundsätzlich festzuhalten, daß der Ansatz des Vektorraummodells in Verbindung mit Clustering-Algorithmen Sinn macht und insgesamt doch erstaunlich gute Ergebnisse liefert. Immerhin beruht die gesamte Methode als Basis nur auf der Zählung von Worthäufigkeiten. Dazu kommt die undankbare Zusammenstellung der Testkollektion: Mehr oder weniger willkürlich ausgewählte Webseiten stellen sicher jede automatische – und nicht nur die – Kategorisierung vor eine große Hürde. In der Literatur werden Clustering-Verfahren oft eher mit relativ gut strukturierten Dokumenten wie etwa Meldungen einer Nachrichtenagentur oder ähnlichem getestet [SKK00]. Es ist zu vermuten, daß dies die leichtere Aufgabe ist.

Auffällig beim Navigieren durch die Hierarchien sind einige Besonderheiten in der Struktur, die die einzelnen Clustering-Verfahren erzeugen. Am deutlichsten fällt dies bei den beiden inkrementellen Algorithmen auf. Durch ihre inkrementelle Natur sieht der Baum zum Ende etwas „unfertig“ aus. Insbesondere enthalten die Blätter oft nur einzelne Dokumente. Im Gegensatz dazu erzeugen *K-Means* und spektrales Clustering typischerweise Blätter mit deutlich mehr Elementen, was den Überblick erleichtert und das Ergebnis besser aussehen läßt. Die inkrementellen Verfahren haben dagegen den Vorteil, daß in sie auch später weitere Dokumente auf gleiche Weise eingefügt werden können. Ist eine Hierarchie allerdings als abgeschlossen zu betrachten, kann es sich bei *DC-Baum* und *K-Baum* anbieten, den Baum abschließend noch einmal zu überarbeiten und ähnliche Knoten mit jeweils sehr wenigen Dokumenten zusammenzufassen.

Ähnliches kann insbesondere auch bei dem *K-Baum*-Verfahren helfen, die etwas ungleichmäßige Form des Baumes zu beheben. Zu überprüfen wäre, inwieweit sich dies bereits in regelmäßigen Abständen während des Clustering-Prozesses machen läßt. Könnte man auf diese Art für eine größere Balance der Hierarchien sorgen, wäre auch die in Abschnitt 3.2.4 gemachte Annahme über die Laufzeit gerechtfertigt. Damit würde insbesondere der *K-Baum*-Algorithmus zu einem für die Praxis gut geeigneten Clustering-Verfahren, das die in Abschnitt 3.1 gestellten Anforderungen erfüllt, dabei sehr gute Ergebnisse liefert und bei einer entsprechend optimierten Implementierung auch effizient arbeitet.

# 6 Zusammenfassung

Das Ziel dieser Arbeit ist eine Bewertung gewesen, inwieweit einzelne Clustering-Verfahren in der Lage sind, aus einer Kollektion von Dokumenten automatisiert eine thematische Hierarchie aufzubauen, die einer Anwenderin oder einem Anwender einen schnellen Zugriff auf interessierende Gebiete bietet.

Dazu sind die Dokumente zunächst mittels des aus dem *Information Retrieval* stammenden Vektorraummodells derart aufbereitet worden, daß sie als Eingabe typischer Clustering-Verfahren haben dienen können. Drei aus der Literatur ausgewählte, mit unterschiedlichen Ansätzen arbeitende Clustering-Algorithmen sowie ein neu im Rahmen dieser Arbeit entstandenes Verfahren sind, zum Teil mit leichten Anpassungen an die Problemstellung, in eine allgemein und flexibel gehaltene Testumgebung integriert und auf einer Testkollektion von Webseiten evaluiert worden. Drei wohldefinierte Qualitätsmaße haben dabei als Bewertungsmaßstab gedient, um schließlich einen objektiven Vergleich der Ergebnisse zu ermöglichen. Einige interessante Dinge haben sich dabei gezeigt.

Zum einen scheint der gewählte Ansatz des Vektorraummodells grundsätzlich gut geeignet zu sein, thematische Ähnlichkeiten von Dokumenten zu erkennen. Dabei haben mit einer Ausnahme stets die Parametrisierungen des Modells die gestellte Aufgabe am besten gemeistert, die auch in der Literatur als die erfolgversprechendsten angegeben sind. Bei der einen Ausnahme handelt es sich um die Art der Dimensionsreduktion. Die Methode, die weitere Bearbeitung nur auf die als inhaltstragend erkannten Wörter zu beschränken, scheint zumindest auf die in dieser Arbeit vorgestellte Art bisher nicht betrachtet worden zu sein. Umso interessanter ist die Feststellung, daß diese Herangehensweise gegenüber der etablierteren Methode des *Latent Semantic Indexing* unabhängig von dem konkreten Clustering-Algorithmus durchgängig die besseren Ergebnisse gezeigt hat. Interessant zu untersuchen bleibt, ob eine Kombination der beiden Verfahren zur Dimensionsreduktion noch einmal bessere Ergebnisse liefert. Die vorliegenden Resultate liefern zumindest ein Indiz dafür, daß dem so sein könnte.

Beim Vergleich der vier untersuchten Verfahren zum Clustering zeigt der neu vorgestellte, auf *K-Bäumen* basierende Algorithmus bei allen drei zur Bewertung herangezogenen Qualitätsmaßen die mit Abstand besten Ergebnisse. Da er Bestandteile aus zwei der anderen Verfahren übernimmt, deutet dies darauf hin, daß er es auf geschickte Art versteht, deren Vorteile zu kombinieren. Inwieweit sich diese Ergebnisse auf eine deutlich größere Anzahl von Dokumenten oder auf ganz andere Arten von Texten übertragen lassen, bleibt zu untersuchen. Betrachtet man die Laufzeit, so liegt der *K-Baum*-Algorithmus in etwa gleich mit zweien der anderen Verfahren. Lediglich das spektrale Clustering ist deutlich schneller, liefert aber gleichzeitig die schlechtesten Ergebnisse. Eine genaue Laufzeituntersuchung bleibt einer speziell optimierten Implementierung überlassen.

Auf die Frage, ob die erzeugten thematischen Hierarchien auch wirklich direkt zur Navigation einer Anwenderin oder eines Anwenders innerhalb der Dokumente dienen können, fällt die Antwort schwer. Neben guten Klassifikationen finden sich in jeder der erzeugten Hierarchien auch ebenso schlechte. Dabei ist kaum vorauszusagen, welche Zusammenhänge ein Algorithmus gut erkennen und welche er eher ignorieren wird. Allerdings dominieren stark vertretene Themen die Ergebnisse deutlich, während sich Dokumente, deren Thema nur wenige Vertreter besitzt, oft nur durch Zufall entdecken lassen. Die Verfahren lassen sich daher vermutlich zumindest nicht direkt, d. h. ohne weitere Nachbearbeitung der Ergebnisse, als Ersatz für eine manuelle Kategorisierung betrachten. Mit den thematischen Katalogen im Internet etwa werden sie sicher nicht konkurrieren können – was aber auch nicht zu erwarten gewesen ist.

Deutlich interessanter ist aber eine andere Sichtweise. Die Ergebnisse liefern im Vergleich zu einer manuellen Klassifikation einen ganz neuen Zugang. Sowohl in der Form der Gruppierung als auch im Kontext sind deutliche Unterschiede zu erkennen. Damit eignen sie sich aber sehr gut, um eine – bekannte oder unbekannte – Sammlung von Dokumenten einfach interaktiv zu erforschen. Auf diese Art lassen sich womöglich Zusammenhänge erkennen, die nicht offensichtlich sind. Betrachtet man die erzielten Resultate auf diese Art, ist damit die eh nur fließend vorhandene Grenze zum *Data Mining* endgültig überschritten.

# Literaturverzeichnis


[BB99] Berry, Michael W. und Murray Browne: *Understanding Search Engines: Mathematical Modeling and Text Retrieval*. Software, Environments, and Tools. Society for Industrial and Applied Mathematics, 1999.

[Bdb] *Berkeley DB*. <http://www.sleepycat.com>.

[BDO94] Berry, Michael W., Susan T. Dumais und Gavin W. O'Brien: *Using Linear Algebra for Intelligent Information Retrieval*. Technischer Bericht UT-CS-94-270, University of Tennessee, Knoxville, 1994.

[Bea00] Beazley, Davin M.: *Python Essential Reference*. New Riders Publishing, 2000.

[Ber92] Berry, Michael. W.: *Large-Scale Sparse Singular Value Computations*. The International Journal of Supercomputer Applications, 6(1):13–49, 1992.

[BFR98] Bradley, P. S., Usama Fayyad und Cory Reina: *Scaling Clustering Algorithms to Large Databases*. In: *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, Seiten 9–15, 1998.

[BGMZ97] Broder, Andrei Z., Steven C. Glassman, Mark S. Manasse und Geoffrey Zweig: *Syntactic clustering of the Web*. Computer Networks and ISDN Systems, 29(8–13):1157–1166, 1997.

[BKR98] Bookstein, A., S. T. Klein und T. Raita: *Clumping Properties of Content-Bearing Words*. Journal of the American Society of Information Science, 49(2):102–114, 1998.

[Cha00] Chakrabarti, Soumen: *Data mining for hypertext: A tutorial survey*. SIGKDD Explorations: Newsletter of the Special Interest Group (SIG) on Knowledge Discovery & Data Mining, ACM, 1, 2000.

[CKPT92] Cutting, Douglass R., David Karger, Jan O. Pedersen und John W. Tukey: *Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections*. In: *Proceedings of the Fifteenth Annual Inter-*

*national ACM SIGIR Conference on Research and Development in Information Retrieval*, Seiten 318–329, 1992.

[DDF+90] Deerwester, Scott, Susan T. Dumais, George W. Furnas, Thomas K. Landauer und Richard Harshman: *Indexing by Latent Semantic Indexing.* Journal of the American Society for Information Science, 41(6):391–407, 1990.

[DFK+99] Drineas, P., Alan Frieze, Ravi Kannan, Santosh Vempala und V. Vinay: *Clustering in large graphs and matrices.* In: *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, Seiten 291–299, 1999.

[DGL92] Duff, Ian S., Roger G. Grimes und John G. Lewis: *User's Guide for the Harwell-Boeing Sparse Matrix Collection (Release I).* Technischer Bericht TR/PA/92/86, CERFACS, Lyon, France, 1992.

[Eve93] Everitt, Brian S.: *Cluster Analysis (Third Edition).* Edward Arnold, 1993.

[FC] Frakes, B. und C. Cox: *Implementation of the Porter stemming algorithm.* `<http://www.cs.jhu.edu/~weiss/stem.c>`.

[FLE00] Farnstrom, Fredrik, James Lewis und Charles Elkan: *Scalability for Clustering Algorithms Revisited.* ACM SIGKDD Explorations, 2(1):51–57, 2000.

[FZ98] Fuller, Michael und Justin Zobel: *Conflation-based Comparison of Stemming Algorithms.* In: *Proceedings of the Third Australian Document Computing Symposium*, 1998.

[Gdb] *GNU Database Manager (GDBM).* <http://www.gnu.org/directory/gdbm.html>.

[Gtp] *General Text Parser (GTP).* <http://www.cs.utk.edu/~lsi/soft.html>.

[Hge] *HTMLgen.* <http://starship.python.net/crew/friedrich/HTMLgen/html/main.html>.

[HK00] Han, Eui-Hong (Sam) und George Karypis: *Centroid-Based Document Classification: Analysis & Experimental Results.* Technischer Bericht 00-017, Department of Computer Science and Engineering, University of Minnesota / Army HPC Research Center, 2000.

[JD83] Jain, Anil K. und Richard C. Dubes: *Algorithms for Clustering Data.* Prentice Hall, 1983.

[KHK99] Karypis, George, Eui-Hong (Sam) Han und Vipin Kumar: *Multilevel Refinement for Hierarchical Clustering.* Technischer Bericht 99-020, Department of Computer Science and Engineering, University of Minnesota, 1999.

[Kow97] Kowalski, Gerald: *INFORMATION RETRIEVAL SYSTEMS: Theory and Implementation.* Kluwer Academic Publishers, 1997.

[KPR98] Kleinberg, Jon, Christos Papadimitriou und Prabhakar Raghavan: *A Microeconomic View of Data Mining.* Data Mining and Knowledge Discovery, 2(4):311–324, 1998.

[KVV98] Kannan, Ravi, Santosh Vempala und Adrian Vetta: *On Clusterings – Good, Bad and Spectral.* In: *Proceedings of the 41st Foundations of Computer Science*, Seiten 367–377, 1998.

[LA99] Larsen, Bjornar und Chinatsu Aone: *Fast and Effective Text Mining Using Linear-time Document Clustering.* In: *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Seiten 16–22, 1999.

[Lar] *Larbin – Multi-purpose web crawler.* <http://pauillac.inria.fr/~ailleret/prog/larbin>.

[LC96] Leouski, Anton V. und W. Bruce Croft: *An Evaluation of Techniques for Clustering Search Results.* Technischer Bericht IR-76, Computer Science Department, University of Massachusetts at Amherst, 1996.

[Odp] *Open Directory Project (ODP).* <http://www.dmoz.org>.

[Por80] Porter, M.F.: *An algorithm for suffix stripping.* Program, 14(3):130–137, 1980.

[Sal94] Salus, Peter H.: *A Quarter Century of UNIX.* Addison-Wesley, 1994.

[SGM00] Strehl, Alexander, Joydeep Ghosh und Raymond Mooney: *Impact of Similarity Measures on Web-page Clustering.* In: *Proceedings of the 17th National Conference on Artificial Intelligence: Workshop of Artificial Intelligence for Web Search (AAAI 2000)*, Seiten 58–64, 2000.

[SKK00] Steinbach, Michael, George Karypis und Vipin Kumar: *A Comparison of Document Clustering Techniques.* Technischer Bericht 00-034, Department of Computer Science and Engineering, University of Minnesota, 2000.

[Ste93] Stevens, W. Richard: *Advanced Programming in the UNIX(R) Environment.* Addison-Wesley, 1993.

[Str91] Stroustrup, Bjarne: *The C++ Programming Language (Second Edition).* Addison-Wesley, 1991.

[Swi] *Simplified Wrapper and Interface Generator (SWIG).* `<http://www.swig.org>`.

[Vit99] Vitter, Jeffrey Scott: *External Memory Algorithms and Data Structures.* In: Abello, James und Jeffrey Scott Vitter (Herausgeber): *External Memory Algorithms and Visualization.* American Mathematical Society Press, Providence, RI, 1999.

[vR79] Rijsbergen, C.J. van: *Information Retrieval (Second Edition).* Butterworths, 1979.

[WA00] Wai-chiu Wong und Ada Wai-chee Fu: *Incremental Document Clustering for Web Page Classification.* In: *IEEE 2000 International Conference on Information Society in the 21st Century: Emerging Technologies and New Challenges (IS2000)*, 2000.

[Xml] *libxml – The XML C library for Gnome.* `<http://xmlsoft.org>`.

[Yah] *Yahoo! Inc.* `<http://www.yahoo.com>`.

[ZE98] Zamir, Oren und Oren Etzioni: *Web Document Clustering: A Feasibility Demonstration.* In: *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Seiten 46–54, 1998.

[ZRL96] Zhang, Tian, Raghu Ramakrishnan und Miron Livny: *BIRCH: An Efficient Data Clustering Method for Very Large Databases.* In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Band 25, Seiten 103–114, 1996.

Hiermit erkläre ich, daß ich diese Arbeit selbständig verfaßt, keine anderen als die angegebenen Quellen als Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Paderborn, Juli 2001 Robin Sommer