Internetsuche und Neuronale Netze: Stand der Technik

Udo Heuser Prof. Dr. W. Rosenstiel

WSI 98-10

23. September 1998

Wilhelm-Schickard-Institut
Universität Tübingen
D-72076 Tübingen, Germany
E-Mail: heuser@informatik.uni-tuebingen.de

ISSN 0946-3852

„Imagine walking up to a librarian and saying ‘travel‘. They‘re going to look at you with a blank face.“
Brian Pinkerton, Gründer der WebCrawler Suchmaschine

1. Einführung

In dem vorliegenden Internen Bericht möchte ich den Stand der Technik wiedergeben, wie er sich heute sowohl für Internet-Suchmaschinen als auch für solche künstliche neuronale Netze darbietet, welche sich auf die Suche in größeren Datenbeständen konzentrieren. Einige wenige Beispiele neuronaler Netze sind bekannt, in denen diese dazu benutzt werden, die Suche im Internet zu verbessern bzw. um herkömmliche Internet-Suchmaschinen oder Kataloge zu ersetzen. Diese werden, zusammen mit anderen Anwendungen neuronaler Netze, im letzten Kapitel dieses Berichtes vorgestellt. Zunächst konzentriert sich der Bericht jedoch, nach einer kurzen Einführung in konventionelle Internet-Suchtechniken, auf das sogenannte Information Retrieval, der Suche in größeren und unter Umständen anwachsenden Datenmengen. Das Internet kann als eine solche Datenmenge aufgefaßt werden. Eine Möglichkeit, die Suche in einem solchen Informationsraum zu verbessern besteht darin, ähnliche Dokumente zusammenzufassen, um sie anschließend durch einen gemeinsamen Prototyp zu repräsentieren. Dies leisten im klassischen Sinne statistische Clusterverfahren, die in Kapitel 5. Statistische Clusterverfahren, S. 20 vorgestellt werden sollen. Anschließend werden solche neuronale Netze besprochen, die vorherige klassische statistische Verfahren auf nicht-lineare Weise erweitern. Nach der Vorstellung verschiedener Anwendungen neuronaler Netze bietet das letzte Kapitel eine Zusammenfassung und einen Ausblick auf weitere Vorgehensweisen.

Dieser Interne Bericht wurde im Rahmen des von der Europäischen Union geförderten internationalen Kooperationsprojektes „*O*pen *A*rchitecture *S*erver for *I*nformation *S*earch and Delivery“ (*OASIS*), Projekt Nr. PL96 1116, im Rahmen des INCO Copernicus Programms angefertigt. Neben dem Lehrstuhl für Technische Informatik am Wilhelm-Schickard-Institut für Informatik sind das University College Dublin, Department of Computer Science, die Saint-Petersburg State University, Faculty of Applied Mathematics and Processes of Control sowie St. Petersburgs größter Internet-Provider Peterlink Co. Ltd. beteiligt. Ziel des Projektes ist es, einen intelligenten Internetsuch- und Lieferdienst zu entwickeln, der auf einer vorhergehenden Klassifikation verschiedenster Dokumente und Anfragen durch neuronale

Netze aufbaut. Die verteilten Suchstrategien und neuronalen Netzansätze sollen dabei einen skalierbaren Informationssuch- und Lieferdienst motivieren, welcher auf einer offenen Architektur basiert und auf unterschiedlichen Rechnerplattformen implementiert werden kann.
Mehr zu dem OASIS-Projekt kann unter `http://www.oasis-europe.org` bzw. `http://www-ti.informatik.uni-tuebingen.de/oasis` eingesehen werden.

2. Internet-Suchmaschinen

Internet-Suchmaschinen („Search Engines“; fälschlicherweise auch „Crawler“ oder „Spider“ genannt) sind recht neue und leicht handhabbare Werkzeuge, um auf schnelle und einfache Art relevante Internet-Dokumente zu finden. Dabei werden vor allem HTML-Seiten [RLJ97] (s. auch [BPS98, Conn98, Cover98, Fiel+97]) des sogenannten „World-Wide-Webs“ gesucht. Das *World-Wide-Web* (*WWW*) ist eine Multimedia-Anwendung des Internets, die per „(Hyper-)Links“ („Hypertext-Verweise“) weltweit verteilt liegende Internet-Dokumente über verschiedene Internet-Protokolle miteinander vernetzt und so ein Geflecht („Web“) aus HTML-Seiten (auch „Hypertexte“ genannt) und Multimedia-Objekten im Internet schafft ([Klut95]; s. a. [HaCr94] und [Etzi96]). Üblicherweise gibt der Benutzer/die Benutzerin[1] der Internet-Suchmaschine ein oder mehrere Suchwörter ein, und die Suchmaschine gibt diejenigen HTML-Seiten aus, die zu dieser Suchanfrage am besten passen. Obwohl Internet-Suchmaschinen noch nicht lange verbreitet sind, haben sich einige bereits in weiten Benutzerkreisen etabliert und werden häufig benutzt. Erwähnt seien hier nur die bekanntesten: *AltaVista* (`http://www.altavista.digital.com`), *Lycos* (`http://www.lycos.com` bzw. `http://www.de.lycos.de`) und *Yahoo* (`http://www.yahoo.com`). Die beiden erstgenannten extrahieren im wesentlichen Wörter aus HTML-Dokumenten und speichern sie in sogenannten invertierten Listen ab, über die anschließend effizient gesucht werden kann. (AltaVista indiziert auf diese Art über 100 Mio. Web-Seiten!) Yahoo ist eigentlich keine Suchmaschine im herkömmlichen Sinne, sondern ein Internet-Katalog („Directory“): Er baut redaktionell betreute (mehrschichtige) thematische Kataloge auf, über die der Benutzer Zugang zu den gewünschten Internet-Seiten erhält. Im Unterschied zu den Search Engines müssen dem Katalog-Dienst Web-Seiten explizit bekanntgemacht werden, d. h. sie werden nicht automatisch durch sogenannte Crawler gefunden. Die Lycos Suchmaschine könnte in dieser Hinsicht als hybrides System bezeichnen werden, da sie zusätzlich einen Katalog-Dienst, genannt „Web Guides“, enthält.
Northern Light (`http://www.northernlight.com`), erst seit August 1997 in Dienst, kann zusätzlich Web-Seiten nach Themen klassifizieren und unterscheidet sich damit von den üblichen Suchmaschinen, wie z. B. AltaVista. Sie sucht zudem in speziellen Web-Sammlungen („collections“), wie z. B. Nachrichtenkanälen („newswires“), Magazinen und Datenbanken nach Informationen, die den anderen Suchmaschinen verschlossen bleiben [Sull98].
Außer zu den oben beschriebenen existieren spezialisierte sowie lokale Suchmaschinen, wie z. B. *Neuroscience Web Search* (`http://www.acsiom.org/nsr/neuro.html`), eine Suchmaschine, die sich auf die Suche von Webseiten aus dem Gebiet der Neurowissenschaften spezialisiert hat, bzw.

1. Im folgenden soll die maskuline immer auch die feminine Form mit einschließen.

Web.de (`http://www.web.de`), die sich auf deutschsprachige Seiten konzentriert hat. Schließlich seien noch die sogenannten „Meta Search Engines“ erwähnt, wie der *MetaCrawler* (`http://www.metacrawler.com`), die Suchanfragen an verschiedene Suchmaschinen weiterleiten und die gesammelten Ergebnisse aufbereitet an den Benutzer ausgeben. Seit neuestem etablieren sich Suchmaschinen, die für den Benutzer Fotos, Graphiken und Videos aufspüren. Erwähnt sei die *Webseek* Search Engine an der Universität Columbia (`http://www.ctr.columbia.edu/webseek`), die zudem vordefinierte Kataloge anbietet, um die Suche thematisch - von „Animals“, „Architecture“ bis „Travel“ - einzugrenzen.

Herkömmliche Internet-Suchmaschinen bestehen im wesentlichen aus drei Elementen (s. [Sull98]):

1. Der *Crawler* (auch *Spider* oder *Indexing robot* genannt) sucht im Web nach HTML-Seiten und verfolgt sukzessive Links, die von gefundenen Seiten auf weitere Seiten oder Objekte zeigen. In regelmäßigen Zeitabständen überprüft der Crawler, ob bereits betrachtete Seiten erneuert wurden und gibt die aktuellen an

2. den *Index* (manchmal auch *Katalog* genannt) weiter. Der Index beinhaltet alle Web-Seiten als Kopien. Erst nachdem eine Seite in diesen Index eingetragen ist, kann

3. die *Search Engine Software* die Seiten des Index mit einer Suchanfrage vergleichen und solche Indizes oder Hyperlinks geordnet ausgeben, die am besten zu passen scheinen. Die Rangfolge bzw. Relevanz der Dokumente wird dabei meistens über die sogenannte *location/frequency Methode* ermittelt: Dokumente, die das Suchwort im Dokumententitel oder in einem der ersten Absätze des Dokuments enthalten, werden höher bewertet als andere („location“). Zugleich werden solche als wichtiger eingeschätzt, die im Text das Suchwort öfter verwenden („frequency“). Die location/frequency Methode wird von vielen Suchmaschinen durch sogenannte „Relevancy Boosters“ ergänzt: WebCrawler benutzt z. B. eine „Link-Popularität“, um die Relevanzabschätzung zu verbessern: Web-Seiten, auf die oft von anderen Seiten über einen Link verwiesen wird, bekommen einen höheren Relevanzwert. Hybride Suchmaschinen wiederum bewerten solche Seiten höher, die mehr als einmal durch den Spider besucht wurden. Schließlich können Schlüsselwörter im sogenannten „meta tag“ die Relevanz von Dokumenten erhöhen. Dies nutzen z. B. *HotBot* (`http://www.hotbot.com`) und *Infoseek* (`http://www.infoseek.com`) aus. *Excite* (`http://www.excite.com`) hingegen liest diese Einträge überhaupt nicht. *WebCrawler* (`http://www.webcrawler.com`) wiederum bewertet Dokumente höher, die die Schlüsselwörter im Titel tragen.
   Schließlich lassen sich Suchmaschinen durch die Tatsache kategorisieren, ob sie ALT-Texte und Kommentare indizieren, ob sie Stopp-Listen und Stemming-Verfahrenn (s. a. Abschnitt 4.1.) einsetzen, ob sie Groß- und Kleinschreibweisen beachten und ob sie solche HTML-Seiten öfters besuchen, die oft geändert werden. Eine umfassende Übersicht der verwendeten Methoden der gängigsten Suchmaschinen liefert [Sull98].

Das herausfordernde bei der Suche im Internet bzw. WWW liegt vor allem in der riesigen Datenmenge von einigen Millionen weltweit verbreiteten HTML-Seiten, zu denen täglich noch hunderte neu hinzukommen. (Die Anzahl der WWW-Sites ist von 130 im Juni 1993 auf über 650.000 im Januar 1997 angewachsen.[2] Eine Site bezeichnet dabei eine Ansammlung von Internet-Dokumenten oder -Seiten eines „Internet-Knotens“.) So haben viele der oben erwähnten konventionellen Suchmaschinen Probleme mit

2. M. Grey: *Internet Statistics: Growth and Usage of the Web and the Internet*, Massachusetts Institute of Technology, 1996 (`http://www.mit.edu/people/mkgray/net/web-growth-summary.html`)

der zunehmenden Datenmenge und liefern teilweise nur recht schlechte Suchantworten auf Benutzeranfragen. Neueste Entwicklungen auf diesem Gebiet zielen deshalb darauf ab, die Suche im Internet durch den Einsatz intelligenter Methoden zu verbessern.

Die Suche in größeren Datenmengen ist nicht neu. Viele Verfahren wurden schon früher vor allem im Umfeld von Datenbankanwendungen konzipiert. Intelligente Internet-Suchmaschinen stellen in vielerlei Hinsicht Erweiterungen früherer (Datenbank-)Suchtechniken dar. Die Methode einer intelligenten Internet-Suchmaschine läßt sich also durchaus unter den Aspekten des sogenannten „Data Mining“, des „Knowledge Discovery in Databases“ bzw. des „Information Retrieval“ betrachten. Eine Internet-Suchmaschine verfolgt nämlich nichts anderes, als bestimmte Einträge aus der Gesamteingabemenge der Internet-Homepages zu finden. Dabei wäre es von Vorteil, auf intelligente Art versteckte und noch unentdeckte Zusammenhänge oder Korrelationen zwischen einzelnen Einträgen zu entdecken und zu nutzen, um die Suche nach relevanten Dokumenten zu erleichtern. Nichts anderes leistet aber das „Information Retrieval“ bzw. das „Knowledge Discovery in Databases“. Letzteres beschäftigt sich dabei - wie der Name schon sagt - mit dem Finden unentdeckten Wissens innerhalb unterschiedlicher Datenbankeinträge, das „Information Retrieval“ behandelt hingegen allgemein eine größere Menge von Eingabedokumenten, die möglicherweise ungeordnet vorliegen können.

Im nächsten Abschnitt sollen zunächst die oben genannten Begriffe - das „Data Mining“, das „Knowledge Discovery in Databases“ und das „Information Retrieval“- einander gegenübergestellt und deren unterschiedlichen Ansätze, aber auch Gemeinsamkeiten herausgestellt werden. In Abschnitt 4. wird aufgezeigt, wie eine Internet-Suchmaschine als „intelligentes Information Retrieval System“ aufgefaßt und konzipiert werden kann.

3. Data Mining, Knowledge Discovery in Databases und Information Retrieval

Wie bereits erwähnt, beschäftigen sich sowohl das *Knowledge Discovery in Databases* (*KDD*; *Wissensextraktion aus Datenbanken*) als auch das *Information Retrieval* (*IR*; etwa: *Informationssysteme*) mit dem Auffinden unentdeckter Zusammenhänge oder versteckten Wissens innerhalb einer Gesamteingabemenge unterschiedlichster Dokumente. Dieses Wissen kann dazu benutzt werden, die Suche nach einem bestimmten Dokument zu erleichtern, d. h. die Effizienz und Effektivität der Suche zu erhöhen. Die beiden Verfahren unterscheiden sich im wesentlichen durch die Art der Eingabedaten: Diese liegen im KDD-Prozeß als Einträge einer einheitlich strukturierten, in sich geschlossenen, sowie zentral verwalteten Datenbank, im („intelligenten“) IR jedoch eher als Elemente einer „offenen“, ungeordneten bzw. heterogenen sowie dynamischen Eingabemenge vor. Das Informationssystem einer intelligenten Internet-Suchmaschine läßt sich also insofern unter das (intelligente) IR einordnen, als das Internet als zugrunde liegende Eingabemenge unstrukturiert ist und durch ständig neu hinzukommende HTML-Seiten sowohl offen ist als auch dynamischen Charakter besitzt; die Eingabemenge erfordert also eine

„intelligente“ Suchtechnik, wie es das intelligente IR anbietet.
Im nächsten Abschnitt sollen die Unterschiede zwischen IR und KDD besprochen werden, bevor in Abschnitt 3.2. auf den zentralen Schritt des KDD eingegangen wird.

3.1. KDD vs. IR

Das Knowledge Discovery in Databases (in [vRij79] auch *Data Retrieval - DR* - und in [Ferb96] *Fakten-Retrieval* genannt) unterscheidet sich vom Information Retrieval in folgenden Punkten: zum einen arbeitet das KDD in der Regel auf einer abgeschlossenen Menge von Einträgen z. B. einer Datenbank, das IR jedoch auf einer sehr viel größeren (evtl. sogar anwachsenden) und ungeordneten Menge von Eingabedokumenten, wie z. B. dem *World-Wide-Web* (*WWW*). Weiter versucht das KDD als Ergebnis einer (Datenbank-)Suchanfrage einen exakt passenden Eintrag („exact match“) zu finden, das IR beschränkt sich dagegen bei der Fülle der zu untersuchenden Dokumenten auf eine Menge „partiell zutreffender“ Suchantworten, von denen wiederum die am geeignetsten zur weiteren Untersuchung ausgewählt werden. Die Informationssuche ist also *probabilistisch* beim IR und *deterministisch* beim KDD. Die deterministische Suchtechnik beruht auf der *deduktiven Logik*, d. h. auf Regeln aRb und bRc, aus denen sich sich die Regel aRc ableiten läßt, wobei R eine beliebige Relation definiert. Wird diese deterministische Suche weiter strukturiert, läßt sich üblicherweise ein sogenannter *Entscheidungsbaum* generieren. Die deduktive Logik entspricht der Verbindungsanalyse des Data-Minings (s. a. Abschnitt 3.2.). Beim probabilistischen IR hingegen sind alle obengenannten Relationen mit einer gewissen „Unsicherheit“ behaftet, wodurch die „Zuverlässigkeit“ der insgesamt abgeleiteten *induktiven Inferenzregel* variabel bleibt. Die induktiven Inferenzregeln des IR werden im allgemeinen durch das Bayessche Theorem modelliert. KDD und IR lassen sich weiterhin in Bezug auf die zugrunde liegende Klassifikation unterscheiden: das KDD basiert auf einer *monothetischen Klassifikation*, das IR auf der *polythetischen Klassifikation*. Monothetische Klassifikationen behandeln attributbehaftete Objekte, die diese eindeutig einer eindeutig bestimmten Klasse zuordnen. Objekte einer Klasse der polythetischen Klassifikation hingegen besitzen Attribute, welche das Objekt zu verschiedenen Klassen zuordnen können. Überwiegen dabei Attribute, die das Objekt einer bestimmten Klasse zuordnen, so wird das gesamte Objekt dieser Klasse zugeordnet. Schließlich ist die Suchanfrage-Sprache des KDD *künstlich*, die des IR *natürlich*. Eine künstliche Suchanfrage-Sprache versucht das gewünschte Objekt vollständig durch die Suchfrage zu spezifizieren. Hierzu dienen auch boolesche Operatoren, die Wörter der Sprache miteinander verknüpfen, wie z. B. die Operatoren „AND“, „OR“ und „NOT“. Der Benutzer verknüpft also Wörter, die gefunden werden sollen und schließt gleichzeitig solche aus, die auf keinen Fall im Text vorkommen sollen. Solche Informationssysteme bezeichnet man folgerichtig als *Boolesches Retrieval* (s. auch [Ferb96], S. 15 ff.). Natürlichsprachige Suchanfragen sind dagegen unvollständig und vage und spiegeln die Tatsache wider, daß sich das IR darauf beschränkt, relevante, nicht jedoch exakt zutreffende Dokumente zu finden. Die Suchantwort des IR gibt die Wahrscheinlichkeit der Relevanz jedes der gefundenen Objekte wider. Künstliche Suchanfrage-Sprachen reagieren im Gegensatz zu natürlichen empfindlicher auf Eingabefehler. Weitere Unterschiede der beiden Suchtechniken liefert detailliert [vRij79].

3.2. Data Mining

Als ein wesentlicher Schritt im Gesamtprozeß des KDD extrahiert das sogenannte *Data-Mining (DM;* etwa: *Regelextraktion aus Datenbanken)* Muster, Modelle bzw. Regeln aus größeren Datenmengen, wie z. B. aus Datenbanken und anderen textbasierten Datenspeichern. Die verschiedenen Schritte im KDD-

Prozeß wie Datenpräparierung, -auswahl, -säuberung, die Inkorporation geeigneten apriori-Wissens sowie eine entsprechende Interpretation der Ergebnisse, sorgen schließlich dafür, daß wirklich nützliches Wissen aus den Daten abgeleitet werden kann. Eine Vorverarbeitung besteht dabei hauptsächlich aus einer Minimierung des sogenannten Hintergrundrauschens und einem (sinnvollen) Hinzufügen fehlender Daten zur selektierten Datenmenge. Die solcherart vorverarbeiteten Daten werden anschließend durch Datenreduktions- und Projektionstechniken transformiert. Der zentrale und wichtigste Schritt im gesamten Prozeßablauf, das sogenannte Data-Mining, kann als eine Art einfache Mustersuche, Klassifikations- oder Clusteraufgabe innerhalb der reduzierten Datenmenge verstanden werden. Die resultierenden Muster werden schließlich in angemessener Weise visualisiert und dem Benutzer angeboten (s. Abb. 1.1. und [FPS96, FPSU98]).

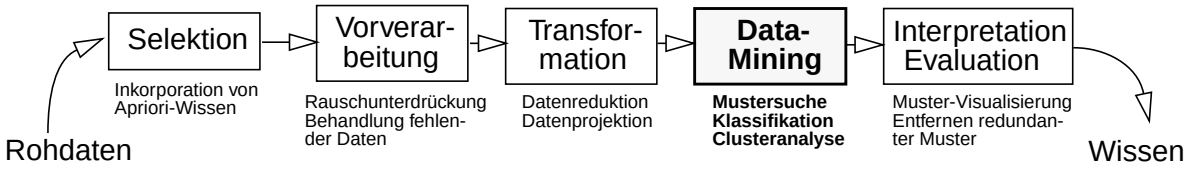


Abb. 1.1.: Die Einbettung des DM-Schrittes in den gesamten Prozeßablauf des KDD (nach [FPS96], S. 29)

Im Gegensatz zum Data-Mining Schritt konzentriert sich das Knowledge Discovery in Databases, das Data Retrieval, oder allgemeiner das Information Retrieval auf den gesamten Prozeß des Entdeckens von verborgenem Wissen oder unentdeckten Zusammenhängen innerhalb der gesamten Datenmenge. Diese Methoden umfassen dabei neben den zuvor erwähnten Schritten auch die Art und Weise, wie die Daten gespeichert und angesprochen werden, wie effektive Algorithmen auch bei Anwendungen auf größere Datenmengen formuliert werden, wie Ergebnisse interpretiert und visualisiert werden und wie vor allem Mensch-Maschine Interaktionen, wie z. B. Benutzer-Feedbacks, modelliert werden können. Benutzer-Feedbacks dienen dazu, die Anfrage-Ergebnisse zu evaluieren bzw. zu interpretieren. Der Benutzer entfernt dabei überflüssige, d. h. irrelevante Muster oder Dokumente und gibt diese mit den relevanten an das Suchsystem zurück. Das Suchsystem wiederum verfeinert mit Hilfe dieser zusätzlichen Information die Suche nach relevanten Dokumenten in einem erneuten Anlauf (*refinement step*).

Gegenwärtig existieren folgende Modellfunktionen für den Data-Mining Schritt: die *Sequenz-Analyse* für nicht-stationäre oder zeitabhängige Daten, die *Verbindungsanalyse (link analysis)*, welche Relationen zwischen (Datenbank-)Feldern im Datensatz zu erkennen versucht, die *zusammenfassende Analyse (summarization)*, welche kompakte Beschreibungen von Untermengen des gesamten Datensatzes liefert, indem sie z. B. den Mittelwert und die Standardabweichung zu allen Feldern berechnet, die *Klassifikation*, die Dateneinträge auf eine oder mehrere vordefinierte kategorische Klassen abbildet, d. h. klassifiziert und die sogenannte *Cluster-Analyse*, die eine Abbildung festschreibt, die analog zur Klassifikation Cluster durch das Finden natürlicher Gruppierungen der Dateneinträge, basierend auf Ähnlichkeitsmetriken, definiert. Gewöhnliche Modellrepräsentationen umfassen Entscheidungsbäume (decision trees), und Regeln, lineare Modelle, nicht-lineare Modelle, wie sie die künstlichen neuronalen Netze darstellen, beispiel-basierte Methoden (z. B. nearest neighbour und case-based reasoning Methoden), Wahrscheinlichkeitsgraphen (z. B. Bayessche Netze) und relationale Attributmodelle. Aus den Methoden der künstlichen neuronalen Netze ragen vor allem folgende heraus: das Multi-Layer Perceptron (MLP) als Feed-Forward Netz, dem gewöhnlicherweise der Backpropagation-Algorithmus als Lernmethode zugrunde liegt. Dieses überwacht lernende neuronale Netz erweitert auf nicht-lineare Weise klassisch-statistische Regressions- oder Diskriminanzanalyse-Techniken. Kohonen‘s Selbst-Organisierende Karte (Self-Organising Feature Map; SOM) ist der wichtigste Repräsentant unüber-

wachter, d. h. selbstorganisierender neuronaler Netze und stellt eine nicht-lineare Erweiterung der Hauptkomponentenanalyse dar (s. hierzu auch [Brau96], S. 111). Diese wird vor allem zur Klassifikation oder zu Cluster-Aufgaben angewendet. Besonders die sogenannten Competitive Learning Neuronale Netze erweitern auf nicht-lineare Weise statistische Modus-Suchverfahren. Näheres hierzu in Abschnitt 6.

3.3. Internetsuche als (intelligentes) IR

Wie im vorigen gezeigt, läßt sich die Suche im Internet unter dem Aspekt des IR betrachten. Die Internet-Suchmaschine handelt mit größeren, heterogenen und dynamischen Datensätzen. Die Suche in einer solchen Datenfülle muß, wie gesehen, unscharf bleiben. Um dennoch den Anforderungen an möglichst exakten Suchantworten gerecht zu werden, bedarf das IR einem angepaßten DM-Schritt. Hier bieten sich vor allem die probabilistischen Suchtechniken, die Klassifikations-Methoden und die Clusteranalyse-Verfahren an. Diese Art von DM könnte als intelligentes Data-Mining bezeichnen werden, den gesamten IR-Prozeß demzufolge als *intelligentes IR*.
Die intelligente DM-Stufe nimmt dabei im gesamten Prozeßablauf des intellgenten IR, bzw. des intelligenten KDD oder DR, die zentrale Position ein (vgl. Abb. 1.1.). Kommerziellen Suchmaschinen verzichten jedoch in der Regel auf den intelligenten DM-Schritt und beschränken sich lediglich auf den Vorverarbeitungs- und vor allem auf den Transformations-Schritt mit anschließender einfacher Mustersuche (s. auch Abschnitt 2.). Die Transformationsstufe wird insbesonders durch automatische Textanalyse- bzw. Termindizierungs-Methoden bestimmt. Diese generieren verkürzte und durch Rechnersysteme verarbeitbare Repräsentationen von Eingabedokumenten, die anschließend in invertierten Listen abgelegt werden. Neueste Entwicklungen konzentrieren sich dabei eher auf die Selektions- oder Vorverarbeitungsstufe. Hierunter lassen sich auch die Bemühungen des *World Wide Web Consortiums* (*W3C*; `http://www.w3.org`) einordnen, eine Metabeschreibungs-Sprache für WWW-Seiten zu entwerfen. Eine solche Sprache soll dazu dienen, den Inhalt und/oder Typ (Text, Video, Graphik, etc.) von WWW-Seiten zu beschreiben ([DeHe97, DeWe96, LaSw98, GuBr97]; s. a. [Lyn97]), um damit die Suche nach Dokumenten wesentlich zu vereinfachen.
Trotz der neuesten Entwicklungen hat sich an dem generellen Ablauf des (intelligenten) IRs seit der Herausgabe des inzwischen als de-facto Standardwerk gehandelten Buches von C. J. van Rijsbergen [vRij79] nichts wesentliches verändert (vgl. hierzu u. a. [Ferb98], [Fuhr97], [FaOa95] und [SaMc83]): Der IR-Prozeß besteht demzufolge aus einer automatischen Volltextanalyse bzw. Termindizierungsmethode, gefolgt von einer automatischen Klassifikation oder Cluster-Analyse bzw. einer einfachen Mustersuche im nicht-geclusterten Raum - also z. B. durch Verwendung invertierter Listen. Wenn die Eingabemenge klassifiziert oder geclustert wurde, muß schließlich festgelegt werden, auf welche Art in der so geclusterten Repräsentation der Gesamteingabemenge aller Dokumente gesucht werden soll. Dies könnte auch als „intelligente Mustersuche“ bezeichnen werden. Unter Einbeziehung der Benutzer-Rückkopplungen (feedbacks) kann die gesamte Arbeitsweise des (intelligenten) IR wie in Abb. 1.2. graphisch veranschaulicht werden.

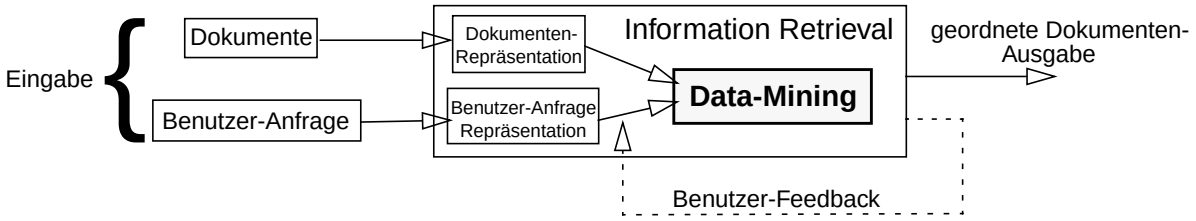


Abb. 1.2.: Vereinfachte Darstellung eines typischen (intelligenten) IR-Systems (nach [vRij79], S. 4 und [Cres95], S. 2)

4. Intelligentes Information Retrieval

In den folgenden Abschnitten 4.1. bis 4.5. sollen die einzelnen Schritte des intelligenten IR vorgestellt werden, wobei der Dokumenten-Klassifikation oder Cluster-Analyse als dem spezifischen Merkmal des intelligenten DM besonderes Augenmerk gewidmet werden soll. Jeder der einzelnen Schritte wird dabei unter dem Aspekt der anschließenden Klassifikation bzw. der Clusterung betrachtet, da hierfür insbesonders auch (unüberwacht lernende oder selbstorganisierende) künstliche neuronale Netze geeignet sind. Nachdem in den Abschnitten 5. und 6. gezeigt wird, wie die Cluster-Analyse (als intelligentes DM) auf statistische Art bzw. mit neuronalen Netzen realisiert werden kann, endet das Kapitel in Abschnitt 7. mit Anwendungsbeispielen neuronaler Netze für das intelligente IR, und hier insbesonders für die Internet-Suche. Auch im letzten Abschnitt werden insbesondere Verfahren besprochen, die Klassifikations- bzw. Cluster-Aufgaben durchführen, für die sich selbstorganisierende neuronale Netze hervorragend eignen. Frühere DM-Methoden, wie sie auch im ID3-Algorithmus von Ross Quinlan Anwendung fanden, beruhten oft auf überwachten Lernmethoden in Kombination mit Entscheidungsbäumen [Quin86]. Näheres zu diesen Lernmethoden enthält [Ferb98], S. 56 ff.
Die ersten, die bereits 1958 bzw. 1961 Klassifikations- oder Cluster-Verfahren als nützliche Tools für das IR erkannten, waren I. J. Good und R. A. Fairthorne [Good58, Fair61].

4.1. Termindizierung

Bevor Dokumente zu Clustern zusammengefaßt werden können, müssen sie durch Termindizierungsmethoden beschrieben werden, d. h. es müssen Vektoren generiert werden, auf denen schließlich die Cluster-Verfahren operieren können. Die Termindizierungsmethode operiert dabei auf kompletten Dokumenten, auf Zusammenfassungen eines Dokuments (Abstract), dem Titel oder einer Liste von Wörtern (sehr oft von solchen, die in den ersten Abschnitten des Dokuments erscheinen). Sie kodiert

Wörter i. d. R. nach der Häufigkeit ihres Auftretens, nicht jedoch - wie vielleicht gewünscht - nach deren semantischer Bedeutung, was damit erklärt werden kann, daß linguistische Analyse-Verfahren immer noch recht aufwendig zu implementieren sind. Gute IR-Systeme wurden jedoch in der Vergangenheit ohne linguistisches Wissen über die Dokumente erfolgreich konstruiert und berechtigen daher die Beschränkung auf einfachere Termindizierungsmethoden. Alle gewöhnlichen basieren auf der Feststellung von H. P. Luhn:

> *„(...) the frequency of word occurance in an article furnishes a useful measurement of word significance“* [Luhn58].

Generell können die beiden folgenden Termindizierungsmethoden unterscheiden werden:

- In dem Ansatz des sogenannten *Signature-Files* wird aus jedem Dokument ein Bit-String (*Signature*) erzeugt. Dazu wird eine Hashing-Funktion über alle Wörter des Dokuments und eine vorgeschaltete Kodierung benutzt. Die Signature-Dateien werden unabhängig von den Dokumenten gespeichert und können sehr viel schneller durchsucht werden. Signature-Dateien treten i. d. R. in Zusammenhang mit sogenannten *Stopp-Listen* auf, welche - basierend auf der Feststellung von Luhn - dazu dienen, gewöhnliche d. h. sehr häufig vorkommende Wörter zu verwerfen. Zudem hilft die aus der Computerlinguistik stammende Prozedur der *Lemmatisierung*, die nicht-gewöhnliche Wörter zu ihren Grundformen reduziert. Hierbei werden verschiedene Flexionsformen eines Wortes als zusammengehörig betrachtet. Man unterscheidet zwischen der *Grundformreduktion*, die die Wörter auf ihre grammatikalische Grundform (bei Substantiven auf den Nominativ singular, bei Verben auf den Infinitiv) zurückführt, und der *Stammformreduktion* (*Stemming*), die die Wortformen auf ihren Stamm zurückführt (vgl. [Ferb98], S. 20 ff.). Letzteres Verfahren basiert auf der Annahme, daß zwei Wörter zur gleichen konzeptuellen Wortklasse gehören, wenn sie den gleichen Wortstamm besitzen. Das Stemming-Verfahren arbeitet recht gut für die englische Sprache, erweist sich jedoch als schwierig für die deutsche, da Endungen im Deutschen häufig auch deren Stämme verändern. Im Deutschen werden deshalb statt gewöhnlicher Stemming-Verfahren lexikonbasierte Morphologieprogramme benutzt. Harrison schlug als Alternative die Benutzung von *n*-Grammen (*n* aufeinanderfolgende Buchstaben anstelle von Wörtern) als Eingabe zur Hash-Funktion vor.

- *Inversion*: Die Schlüsselwörter (keywords) eines Dokuments werden bei invertierenden Verfahren z. B. alphabetisch in Index-Dateien (oder *invertierte Listen*) gespeichert. Für jedes Schlüsselwort wird ein Zeiger zu einem qualifizierenden Dokument bereitgehalten. Dieses Verfahren spart viel Rechenzeit, kostet dagegen aber viel Speicherplatz; eine invertierte Liste kann genausoviel Speicherplatz einnehmen wie die Dokumente, über die sie Auskunft gibt (vgl. [Ferb96], S. 2). Diese Methode wird in Kombination mit dem booleschen Retrieval von nahezu allen kommerziellen Systemen angewandt.

Alternativ zum Stemming-Verfahren und der Benutzung von Stopp-Listen gibt es die Möglichkeit, Wörter lediglich aufgrund ihrer Häufigkeit zu verwerfen und auf die Auflistung der zu verwerfenden Wörter zu verzichten (*Termhäugigkeit*). Diese Idee wurde von H. P. Luhn als Weiterentwicklung des Zipfschen Gesetzes entwickelt. Das *Zipfsche Gesetz* stellt zunächst fest, daß das Produkt aus Worthäufigkeit und Häufigkeitsrang der Wörter konstant ist [Zipf49]. Trägt man die Wörter nach ihrer Häufigkeit im Dokument als Histogramm auf, so entsteht eine hyperbolische Kurve, die mit abnehmendem Häufigkeitsrang der Wörter den Wert *0* approximiert. Unter der Annahme, daß lediglich solche Wörter, die nicht zu selten, aber auch nicht zu häufig vorkommen, ein Dokument signifikant beschreiben können, lassen sich eine obere und untere Schwelle definieren, die eben genau solche Wörter einschließen. Zur Erzeugung solcher Schwellen werden oft Heuristiken benutzt. Anstelle der Häufigkeit von Termen wird im Infor-

mation Retrieval oft die *Dokumenthäufigkeit* (*document frequency*) oder die invertierte Dokumenthäufigkeit verwandt. Die Dokumenthäufigkeit ist die Anzahl der Dokumente, in denen ein Term auftritt (s. .a. [Ferb96], S. 34 ff.).

Eine andere Möglichkeit, die wie die Luhnsche Idee auch bei invertierten Listen Anwendung finden kann, besteht darin, *Thesauri* zu verwenden, um Wörter nach ihren Bedeutungen zu strukturieren. Thesauri sind Zusammenstellungen von Begriffen, die nach ihrer Bedeutung als Oberbegriffe, Spezialisierungen oder verwandte Begriffe geordnet sind. Die manuelle Konstruktion und Pflege eines Thesaurus‘ ist allerdings aufwendig. Außerdem setzt die korrekte Formulierung einer Suchanfrage durch den Benutzer genaue Kenntnisse sowohl des Thesaurus‘ als auch des Fachgebiets voraus. Die Nutzung bleibt damit nach R. Ferber im wesentlichen auf spezialisierte und professionell gepflegte Informationsdienste beschränkt ([Ferb96], S. 3).
Eine weitere Art, Wörter zu indizieren, ist das sogenannte *probabilistische Indizierverfahren*. Dieses Verfahren berücksichtigt die statistische Verteilung von Wörtern im Text und kodiert sogenannte „Spezialwörter“ („content-bearing speciality words“) genau dann als Indexterme, wenn ihre statistische Verteilung im Text z. B. von einer Poisson-Verteilung abweicht. Auf dieses Verfahren kann aus Platzgründen an dieser Stelle nicht weiter eingegangen werden. Interessierte Leser werden auf [vRij79], S. 15 ff. verwiesen.

Die Dokumenten-Repräsentation ist schließlich eine Liste von Klassennamen von Klassen konzeptuell äquivalenter Wörter. Die Klassennamen werden - wie bereits erwähnt - auch als *Indexterme* oder *Schlüsselwörter* (*keywords*) bezeichnet.

4.2. Vektorraummodell und Clusterhypothese

Eine Clustergenerierungs-Methode operiert in natürlicher Weise auf Vektoren im *n*-dimensionalen Vektorraum. Jedes Dokument (und jede Benutzeranfrage an das System) muß folglich durch einen ebensolchen Vektor repräsentiert werden und erfüllt damit das *Vektorraummodell* (*vector space model*).
Eine konventionelle automatische Indiziermethode, welche Vektoren aus Dokumenten erzeugt, besteht zunächst aus folgenden Schritten (s. a. Abschnitt 4.1.):

1. Entfernen gewöhnlicher Einträge (wie z. B. häufig vorkommende Wörter).

2. Reduktion der Wörter auf ihre jeweiligen Wortstämme durch das Entfernen von Wortsuffixen und -präfixen.

3. Das Zuteilen von Wortstämmen zu konzeptuellen Klassen oder Termen unter Zuhilfenahme von Synonymlisten

Im folgenden wird im binären Dokumenten-Vektormodell das Vorhandensein oder Fehlen eines Terms durch die Einträge *1* bzw. *0* angezeigt. Im kontinuierlichen Modell werden diese durch positive ganze Zahlen (*Termgewichte*) ersetzt, die die Wichtigkeit des Terms für das Dokument reflektieren sollen. Das Vektorraummodell läßt sich für diesen Fall mathematisch wie folgt formulieren (s. [Ferb96], S. 29 ff.): Es sei $T = (t_1, \ldots, t_n)$ eine endliche Menge von Indextermen und $D = (d_1, \ldots, d_m)$ eine Menge von Dokumenten. Für jedes Dokument $d_i \in D$ sei zu jedem Term $t_k \in T$ ein Gewicht $w_{ik} \in \mathbf{R}$ gegeben. Die Gewichte des Dokumentes d_i lassen sich zu einem Vektor $w_i = (w_{i1}, \ldots, w_{in}) \in \mathbf{R}^n$ zusammenfassen. Dieser Vektor beschreibt das Dokument im Vektorraummodell; er ist seine Repräsentation und wird *Dokumentvektor* genannt. Auf gleiche Art wird mit Anfragen an das System verfahren: diese erzeugen den sogenannten *Anfrage-* oder *Queryvektor* $q \in \mathbf{R}^n$.

Verschiedene Gewichtsfunktionen wurden in der Vergangenheit vorgeschlagen (vgl. [Spar72]), wie zum Beispiel

- die Häufigkeit des Auftretens (*Frequenz*) des Terms *k* in Dokument *i*: FREQ_{ik}
- die *Termspezifität* $\log N - \log(\mathrm{DOCFREQ}_k) + 1$, wobei $\log(\mathrm{DOCFREQ}_k)$ die Anzahl der Dokumente angibt, die Term *k* enthalten und *N* die Gesamtzahl aller Dokumente ist.
- die *inverse Dokumentenfrequenz*: $\frac{\mathrm{FREQ}_{ik}}{\mathrm{DOCFREQ}_k}$
- $\mathrm{FREQ}_{ik} \cdot \mathrm{TERMREL}_k$, wobei $\mathrm{TERMREL}_k = \frac{r_k/(R - r_k)}{s_k/(I - s_k)}$ den Termrelevanzfaktor angibt. *R* ist hierbei die gesamte Anzahl relevanter Dokumente, r_k ist die Anzahl relevanter Dokumente, die Term *k* enthalten, *I* ist die gesamte Anzahl irrelevanter Dokumente und s_k die Anzahl irrelevanter Dokumente, die Term *k* enthalten.

Weitere Gewichtungsmethoden, die lokale oder kontextabhängige Einflussfaktoren mit berücksichtigen, nutzen Informationen über die Struktur der Dokumente. So können z. B. Terme, die in Titeln oder Hyperlinks oder innerhalb eines Meta-Keyword Tags (`<META NAME="keywords" CONTENT="...">`) eines HTML-Dokuments vorkommen höher gewichtet werden, als solche, die innerhalb der `<body>`-Umgebung vorkommen.
Weitergehende Möglichkeiten bieten Dokumente, die mit *SGML* (*Standard General Markup Language;* [Cover98]) formatiert sind. Bei diesen Dokumenten sind verschiedene Textteile, wie Kapitel, Sektionen, Absätze und die dazugehörigen Überschriften im Quelltext markiert. Dadurch können z. B. Terme, die in einer Kapitelüberschrift auftauchen, stärker gewichtet werden als solche, die nur in einer Fußnote auftauchen ([Ferb96], S. 35).

Bevor nun Cluster durch das intelligente Information Retrieval gebildet werden können, muß zunächst ein *Ähnlichkeitsmaß* oder *Assoziationsmaß* entworfen werden, um die Ähnlichkeit der Textdokumente oder anderer Objekte zu quantifizieren. Diese Maße gewährleisten, daß ähnliche Objekte eine gemeinsame Klasse oder Cluster generieren, die von Objekten anderer Cluster unterschieden werden können. Das einfachste aller Assoziationsmaße ist der sogenannte *Simple-Matching Koeffizient*, der die Anzahl der gemeinsamen Indexterme zwischen zwei Objekten *X* und *Y* mißt und folgendermaßen mathematisch ausgedrückt werden kann ([vRij79], S. 24 ff.):

$$X \cap Y \quad \text{(Simple-Matching Koeffizient)} \tag{1.1}$$

Indexterme können - wie gesehen - aus (Schlüssel-)Wörtern (keywords) oder anderen charakterisierenden Elementen bestehen. Der Simple-Matching Koeffizient berücksichtigt dabei nicht die Größe von *X* und *Y*, d. h. die Länge der beiden Objekte oder Dokumente. Da die Länge des Dokumentenvektors stark das Ähnlichkeitsmaß beeinflußt, werden erweiterte Maße entworfen, die die Größe des Objekts bzw. die Länge des Dokumentenvektors $|\,.\,|$ berücksichtigen und also das *normalisierte Assoziationsmaß* beschreiben. Unter diesen sind der Dicesche Koeffizient, Jaccards Koeffizient, der Kosinuskoeffizient und der Überlappungskoeffizient zu nennen:

$$\frac{|X \cap Y|}{|X| + |Y|} \qquad \text{(Dicescher Koeffizient)} \tag{1.2}$$

$$\frac{|X \cap Y|}{|X \cup Y|} \qquad \text{(Jaccards Koeffizient)} \tag{1.3}$$

$$\frac{|X \cap Y|}{\sqrt{|X|}\ \sqrt{|Y|}} \qquad \text{(Kosinuskoeffizient)} \tag{1.4}$$

$$\frac{|X \cap Y|}{\min(|X|, |Y|)} \qquad \text{(Überlappungskoeffizient)} \tag{1.5}$$

Salton schlug binäre Vektoren als Dokumentenrepräsentationen vor, welche in einem *n*-dimensionalen Euklidschen Raum eingebettet sind, wobei *n* die Gesamtzahl der Indexterme widergibt (vgl. [vRij79], S. 26 bzw. [Salt68]). Gleichung (1.4) kann unter dieser Bedingung als Kosinus des Winkels zwischen den beiden binären Vektoren *X* und *Y* betrachtet werden. Sind *X* und *Y* reellwertig (d. h. Vektoren über gewichtete Schlüsselwörter), so läßt sich (1.4) umschreiben zu:

$$\frac{(X, Y)}{\|X\|\ \|Y\|} \tag{1.6}$$

wobei (X, Y) das innere Produkt und $\| \cdot \|$ die Länge des Vektors angibt. Ist der Vektorraum Euklidsch, so erhält man für $X = (x_1, x_2, \ldots, x_n)$ und $Y = (y_1, y_2, \ldots, y_n)$:

$$\frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\ \sqrt{\sum_{i=1}^{n} y_i^2}} \tag{1.7}$$

Einige Autoren haben versucht, das Assoziationsmaß auf ein probabilistisches Modell zu gründen. Sie messen die Assoziation (Ähnlichkeit) zwischen zwei Objekten über das Ausmaß der Abweichung der Verteilung der Objekte von einer stochastischen Unabhängigkeitsverteilung (s. a. [MaKu60]).

4.2.1. Clusterhypothese

Der eigentliche Zweck jedes Cluster-Verfahrens ist es, Dokumente bzw. ihre Vektoren so zusammenzufassen, daß eine daran anschließende Suche beschleunigt werden kann (*document clustering*), oder aber, um Thesauri automatisch zu erstellen (*keyword clustering*). Leider läßt sich oft kein geeignetes Verfahren im voraus angeben, da die Clusterbildung sehr stark von den Eingabedaten abhängt. Die Güte des Cluster-Verfahrens kann daher oft nur durch die Suche in einer solchen geclusterten Dokumenten-Repräsentation beurteilt werden. Im weiteren soll nur auf die Dokumenten-Clusterung eingegangen werden. Erste Überlegungen zur Effizienzsteigerung durch Dokumenten-Clusterung wurden bereits 1968 von G. Salton angestellt [Salt68]. Dabei geht allen Dokumenten-Clusterverfahren folgende Überlegung von C. J. van Rijsbergen voraus:

> *„Closely associated documents tend to be relevant to the same request“* ([vRij79], S. 30)

Diese These, Voraussetzung aller IR-Systeme, welche Cluster-Verfahren benutzen, wird auch „Dokumenten-Clusterhypothese“ oder einfach *Clusterhypothese* genannt. Sie besagt, daß zu einer gegebenen Benutzeranfrage alle solche Dokumente passen, die miteinander verwandt sind, und also gemeinsam

dem Benutzer dargeboten werden sollten. Der „Verwandtschaftsgrad“ wird dabei durch obige Assoziationsmaße definiert. Zudem sollte eine Clustermethode folgende Zuverlässigkeitskriterien erfüllen ([vRij79], S. 31):

- Stabilität unter Wachstum: Die Methode produziert eine geclusterte Repräsentation des Datensatzes, für die eine Änderung bei einem späterem Hinzufügen von Objekten unwahrscheinlich ist.

- Stabilität in dem Sinne, daß kleine Fehler in den Objektbeschreibungen nur zu kleinen Änderungen in der geclusterten Repräsentation führen.

- Unabhängigkeit zur initialen Ordnung der Objekte

Viele Cluster-Verfahren, wie z. B. das dem von Salton und McGill entwickelte SMART-Retrievalsystem zugrunde liegende Clusterfahren (s. [SaMc83], S. 127 ff.), sind sogenannte Single- oder One-Pass Clusterverfahren (s. a. [vRij79], S. 35 ff., [Murt85], S. 29 und [Ferb98], S. 36 ff.). Das SMART Clustering-System ist dabei, wie heuristische Clusterverfahren, stark von der Ausgangskonfiguration oder der Reihenfolge der zu analysierenden Dokumente abhängig (s. [SaMc83], S. 235). Dessen generelle Arbeitsweise soll hier kurz dargelegt werden:

1. Die Objektbeschreibungen (Dokumenten-Vektoren) werden seriell verarbeitet.

2. Das erste Objekt wird die Cluster-Repräsentative (d. h. das Cluster-Centroid) des ersten Clusters.

3. Jedes nachfolgende Objekt wird gegen alle Cluster-Repräsentativen verglichen, welche zur Verarbeitungszeit existieren.

4. Ein gegebenes Objekt wird einem oder mehreren Clustern (falls Überlappungen erlaubt sind) zugeteilt. Dies geschieht aufgrund des Assoziationsmaßes und zuvor festgelgten Schwellen.

5. Die Cluster-Repräsentative wird für alle solche Cluster neu berechnet, denen ein Objekt zugeteilt wurde.

6. Wenn ein Objekt keinem schon existierenden Cluster zugeteilt werden kann, so generiert das Objekt ein neues Cluster, für welches es die Cluster-Repräsentative darstellt.

Das Verfahren ist dabei nicht nur vom gewählten Assoziationsmaß, sondern auch von einigen empirischen Parametern abhängig: die Anzahl der gewünschten Clustern, die minimale und maximale Größe jedes Clusters, ein Schwellenwert zum Assoziationsmaß, der festlegt, bis zu welchem Wert ein Objekt noch zu einem Cluster zugeteilt werden darf, eine Kontrolle der Überlappung zwischen verschiedenen Clustern, und schließlich eine Zielfunktion, welche optimiert werden soll.

Generell lassen sich Cluster-Verfahren (nach A. K. Jain und R. C. Dubes [JaDu88] auch „unüberwachte statistische Klassifikationen“ genannt) grob in die sogenannten Hierarchischen Klassifikationen bzw. Hierarchischen Clusterungen und die Optimierenden bzw. Partitionierenden Clusterungen unterteilen. In die erste Hauptgruppe fallen graphentheoretische Ansätze, wie z. B. das Single-Link- und das Complete-Link-Verfahren, die zweite Hauptgruppe läßt sich wiederum in die sogenannten globalen Cluster-Verfahren und in lokale Clusterungen einteilen. Zu diesen zählen das Square-Error Clustering, von wel-

chem das K-Mean die bekannteste Implementierung darstellt, das Minimal Spanning Tree, sowie das Nearest Neighbor, u. a. (s. a. [JaDu88], S. 55 ff. und [HeHe95], S. 44 ff.). Eines der modernsten statistischen Verfahren stellt das BANG-Clustering dar.

Cluster-Methoden lassen sich weiterhin auf nicht-lineare Art durch künstliche neuronale Netze realisieren. In diesem Bereich dominieren vor allem die Verfahren des Competitive Learning als vektorquantisierende Methoden. Diese lassen sich weiter in die sogenannten Hard Competitive Learning (auch „Winner-take-all“ Verfahren genannt) und die Soft Competitive Learning Verfahren unterteilen. In die erste Hauptgruppe fallen das LBG und das K-Means, in die zweite Hauptgruppe, die sich noch weiter in Verfahren differenzieren läßt, deren Netzwerkdimensionalität vorher vorgegeben ist und in solche mit vorher nicht vorgegebener Netzwerkdimensionalität, fallen die SOM, die Growing Cell Structures, das Growing Grid, das (Growing) Neural Gas, das Competitive Hebbian Learning, das Rival Penelized Competitive Learning, u. a. (s. a. [Fritz97]). Einen umfassenden Überblick über klassische statistische Clusterverfahren, sowie solche, die durch neuronale Netze realisiert werden, wird in den Abschnitten 5. und 6. gegeben.

Das Auffinden von Dokumenten kann weiter verbessert werden, indem Clusterhierarchien gebildet werden. Hierbei werden Cluster zu sogenannten Superclustern zusammengefaßt, welche wiederum zu weiteren Clustern zusammengefaßt werden können, u.s.w. Diese auch als *geordnete Klassifikation* bezeichnete Clustermethode ist nach van Rijsbergen besonders bedeutend für eine effiziente Suche im Dokumenten-Clusterung (s. [vRij79], S. 29 ff.). Die einfachste Methode besteht darin, den Suchanfrage-Vektor mit allen Cluster-Zentroiden der obersten Hierarchiestufe zu vergleichen und die Vergleiche an Cluster-Zentroiden solcher Cluster fortzusetzen, die Subcluster eines Clusters darstellen, dessen Zentroid der Suchanfrage am ähnlichsten gewesen ist. Diese Suche läßt sich bis auf die Cluster der untersten Hierarchiestufe fortsetzen und damit bis zum direkten Suchanfrage-Dokumentenvektor-Vergleich. Im Gegensatz zu dieser *Top-Down-Clustersuche* beschränkt sich die *Bottom-Up-Clustersuche* auf die Vergleiche zwischen Anfragevektor und Cluster-Zentroiden der untersten Cluster: Nach einem ersten Anfrage-Zentroid-Vergleich können bei dieser Methode der Suchanfrage-Vektor gleich mit den Dokumenten-Vektoren des entsprechenden Clusters verglichen werden. Die Bottom-Up-Clustersuche reduziert den Speicheraufwand des Strukturbaumes erheblich und beschleunigt den Vorgang, da Abgleiche mit Cluster-Zentroiden höherer Hierarchiestufen entfallen. ([SaMc83], 232 ff.)

4.3. Multimediale Dokumente

Das WWW enthält nicht nur textbasierte HTML-Seiten, sondern auch multimediale Dokumente. Als multimediale Dokumente werden solche bezeichnet, welche Objekte wie z. B. Bilder, Graphiken, Ton oder Video enthalten. Bislang gibt es nur wenige Verfahren, die in der Lage sind, die Inhalte solcher Objekte automatisch zu beschreiben. „AltaVista“ und „InfoSeek“ unterstützen bislang lediglich sogenannte *Image Maps*, also Bilder, die auch als Links auf andere Seiten dienen [Sull98]. Für spezielle Bilder oder Graphiken gibt es zwar elaborierte Bilderkennungsverfahren, die jedoch nur unter spezifischen Bedingungen eingesetzt werden können (vgl. [GoWi87]). Viele andere Objektbeschreibungssprachen sind Gegenstand der aktuellen Forschung und sollen hier nicht weiter verfolgt werden.
Ein interessanter Ansatz soll jedoch erwähnt werden. Es handelt sich dabei um das Verfahren von L. Page et. al. der Stanford University in Kalifornien: Sie nutzen die Linkstruktur des WWW, um Objekte durch die Namen derjenigen Hyperlinks zu beschreiben, die auf das entsprechende Objekt verweisen. Die Methode basiert auf der Erfahrung, daß die Namen der Hyperlinks tatsächlich die Objekte, auf die sie verweisen, recht gut beschreiben können. Dabei lassen sich mit dieser Methode nicht nur multimediale Objekte, sondern auch gewöhnliche HTML-Seiten beschreiben. Mehr zu diesem Verfahren, das

zugleich WWW-Dokumente nach deren Wichtigkeit über die von ihnen entwickelte PageRank-Methode ordnet, kann in [PBMW98, BrPa98] eingesehen werden.
Weitere Verfahren, die wie obige Methode auf Generalisierungen des sogenannten *Kozitierens* und *bibliographischen Koppelns* basieren, sind bekannt. Diese verwenden, wie z. B. die Arbeiten von Rivlin et. al. [RBS94, BRS92], graphentheoretische Ansätze zusammen mit Kompaktheitsmaßen. Interessierte Leser seien an dieser Stelle an [Klein98] verwiesen, der auf S. 7 einen kurzen Überblick über diese Verfahren gibt.

4.4. Dokumentensuche

Die Suche innerhalb einer geclusterten Repräsentation der Dokumente ist bedeutend einfacher als die Clustergenerierung selbst. Die an das System vorgetragene Benutzeranfrage wird als *n*-dimensionaler Vektor dargestellt, wobei die selben Kodierungs- und evtl. Kompressionstechniken wie bei den bereits klassifizierten Dokumenten benutzt werden. Dieser Vektor wird anschließend mit allen *Clustermittelpunkten* (*cluster centroids*) verglichen. Die Suche endet dabei bei solchen Clustern, deren Ähnlichkeit (der Clustermittelpunkte) mit dem Anfragevektor eine vorher festgelegte Schwelle überschreitet. Zu diesem Zweck muß eine Cluster-zu-Anfrage *Ähnlichkeitsfunktion* oder *Ähnlichkeitsmaß* definiert werden. Dabei können i. d. R. alle Maße benutzt werden, die bereits als Assoziationsmaß benutzt wurden, Ähnlichkeiten zwischen zwei verschiedenen Dokumenten zu messen (s. Abschnitt 4.2.). Eine häufig verwendete Ähnlichkeitsfunktion ist das *Kosinusmaß* oder die *Kosinuskorrelation*:

$$\mathrm{sim}(Q, D_i) = \frac{\sum_{j=1}^{t} q_j d_{ij}}{\sqrt{\sum_{j=1}^{t} q_j^2}\sqrt{\sum_{j=1}^{t} d_{ij}^2}} \tag{1.8}$$

d. h. der Kosinus des Winkels zwischen einer Benutzeranfrage $Q = (q_1, q_2, \ldots, q_t)$ und einem Dokument $D_i = (d_{i1}, d_{i2}, \ldots, d_{it})$ aus einem geeignetem Dokumentencluster. Dabei bezeichnen die Werte d_{ij} Dokumentengewichte und q_j Anfragegewichte ([vRij79], S. 76, [SaMc83], S. 128 ff.).
Üblich ist auch die Verwendung des sogenannten *Pseudo-Kosinusmaßes*. Bei diesem sind gegenüber dem Kosinusmaß die euklidschen Längen durch die Summe der Vektoreinträge, d. h. durch die L_1-Längen, ersetzt:

$$\mathrm{sim}(Q, D_i) = \frac{\sum_{j=1}^{t} q_j d_{ij}}{\left(\sum_{j=1}^{t} q_j\right)\left(\sum_{j=1}^{t} d_{ij}\right)} \tag{1.9}$$

Auch dieses Maß kann als Skalarprodukt (L_1-) normierter Vektoren aufgefaßt werden. Solange alle Vektoreinträge positive Zahlen besitzen, reagiert diese Normierung weniger stark auf große Werte unter den Einträgen (s. [Ferb98], S. 39).

Die gefundenen Dokumente werden nach deren Ähnlichkeitswerten geordnet ausgegeben, so daß die zur Benutzeranfrage ähnlichsten (und damit hoffentlich relevantesten) dem Benutzer zuerst präsentiert werden.
Schließlich wird die Benutzeranfrage durch sogenannte *Benutzerrückkopplungen* (*relevance feedbacks*) verfeinert. Diese Rückkopplungen sind zum einen dadurch motiviert, daß Benutzeranfragen meist zu kurz sind, um eine der Dokumentengewichtung äquivalente Termgewichtung vornehmen zu können.

Zum anderen kann der Benutzer anhand der gefundenen Dokumente am besten selbst entscheiden, was er sucht, also die Relevanz der ausgegebenen Dokumente einschätzen. Benutzerrückkopplungen lassen sich nach Rocchio folgendermaßen realisieren ([Rocc71]; s. a. [SaMc83], S. 150 ff.): Der Benutzer markiert diejenigen der ausgegebenen Dokumente, die er für wichtig hält. Das Suchsystem formuliert nun unter Einbeziehung dieser Dokumente von neuem den Anfragevektor und beginnt die Suche noch einmal (*Anfrageverfeinerung*; *refinement*). Üblicherweise werden zum Anfragevektor die (gewichteten) Vektoren der relevanten Dokumente durch einfache Vektoraddition hinzugefügt und nicht-relevante durch Vektorsubtraktion abgezogen. Dieses interaktive oder Online-Verfahren wird fortgesetzt, bis der Benutzer das Verfahren beendet. Die Benutzerrückkopplung läßt sich dabei mathematisch wie folgt modellieren:
Nach dem Relevanzurteil des Benutzers ergeben sich zwei Dokumentenmengen $R = (d_1^+, \ldots, d_r^+)$, die Menge der relevanten und $I = (d_1^-, \ldots, d_i^-)$, die der irrelevanten Dokumente. Bezeichnet v_d den zu Dokument *d* gehörenden Dokumentvektor, dann wird ein neuer Anfragevektor *q'* nach folgender Formel berechnet:

$$q' = \alpha q + \beta\left(\frac{1}{r}\sum_{d \in R} v_d\right) - \gamma\left(\frac{1}{i}\sum_{d \in I} v_d\right) \tag{1.10}$$

α, β und γ werden dabei frei aus der Menge der reellen Zahlen gewählt. Der neue Anfragevektor wird also im Dokumentenvektorraum zu denjenigem Unterraum hingezogen, der die relevanten Dokumente enthält und gleichzeitig von denjenigem abgestoßen, der die irrelevanten Dokumente enthält. Da man davon ausgehen kann, daß letztere mehr oder weniger gleichverteilt im Raum vorliegen, setzt man häufig $\gamma = 0$.
Der gesamte Prozeß des Generierens von Clustern und der Suche in dieser geclusterten Repräsentation kann mit geeigneten Cluster-erzeugenden neuronalen Netzen vollständig automatisiert werden.

4.5. Die Effektivität des Suchsystems

Die *Effektivität* des Dokumenten-Suchsystems kann durch die beiden Maße *Recall* (etwa: „Vollständigkeit“) und *Precision* („Genauigkeit“) gemessen werden:

$$\text{Recall} = \frac{\text{Anzahl der gefundenen relevanten Dokumente}}{\text{Anzahl der relevanten Dokumente}} \tag{1.11}$$

$$\text{Precision} = \frac{\text{Anzahl der gefundenen relevanten Dokumente}}{\text{Anzahl der gefundenen Dokumente}} \tag{1.12}$$

Mathematisch exakter lassen sich die beiden Gleichungen (1.11) und (1.12) wie folgt wiedergeben (s. [Ferb98], S. 43):
Es sei $D = \{d_1, \ldots, d_m\}$ eine Menge von Dokumenten, $q \in Q$ eine Anfrage und D_q die Menge der in *D* zur Anfrage *q* gefundenen Dokumente. Ferner seien $r : D \times Q \to \{0, 1\}$ eine Relevanzrelation und $r_q : D \to \{0, 1\}$; $r_q(d) := r(d, q)$ die zur Anfrage *q* gehörende Relevanzfunktion. Dann beschreibt

$$R(q, D) := \frac{\left|D_q \cap r_q^{-1}(\{1\})\right|}{\left|r_q^{-1}(\{1\})\right|} \tag{1.13}$$

den Recall oder die Vollständigkeit der Antwort auf die Anfrage q und

$$P(q, D) := \frac{\left|D_q \cap r_q^{-1}(\{1\})\right|}{\left|D_q\right|} \tag{1.14}$$

die Genauigkeit der Antwort auf die Anfrage q. Zuvor muß natürlich eine entsprechende Relevanzfunktion definiert werden (s. hierzu [Ferb98], S. 42).
Die beiden Maße werden i. d. R. nie gleichzeitig erfüllt. Dies ist leicht zu erkennen, wenn man die Precision- gegenüber den Recall-Werten in einem Diagramm aufträgt: Es entsteht eine hyperbolische diskrete Kurve, bei der mit zunehmendem Recall die Precision gegen *0* strebt. Ein IR-System ist also nur dann besser als ein anderes Suchsystem, wenn es sowohl bessere Precision- als auch Recall-Werte liefert. Für IR-Systeme, also auch für Internet-Suchmaschinen, welche Benutzer-Rückkopplungen benutzen, ist sicherlich der Recall von größerem Interesse, da die Precision bei erneuten Durchgängen iterativ verbessert werden kann.
Andere wichtige Evaluationsmaße berücksichtigen die Zeit, die zur Indizierung von Dokumenten benötigt wird und die Geschwindigkeit mit der Dokumente mit den Anfragen verglichen und anschließend ausgegeben werden können. Außerdem kann dabei der Speicherplatzverbrauch berücksichtigt werden. Diese Maße bestimmen also die *Effizienz* der IR-Systems und können durch Benchmarktests evaluiert werden. Die Effizienz ist dabei wesentlich einfacher zu bestimmen als die Effiktivität, welche auf empirische Methoden angewiesen ist.

Im folgenden konzentriert sich die Arbeit auf den zentralen Teil des Information Retrievals: dem Data-Mining Schritt. Hier sollen - wie bereits erwähnt - vor allem den Clusterverfahren ein besonderes Augenmerk gewidmet werden. Nach Salton und McGill ([SaMc83], S. 236 ff.) ist durch das Gruppieren von ähnlichen Dokumenten in Klassen oder Cluster prinzipiell ein besserer Precision- und Recall-Wert zu erwarten, da das Finden eines relevanten Dokuments zugleich zu weiteren relevanten Dokumenten führt. Dies muß allerdings mit erhöhtem Rechen- und Speicheraufwand für die Cluster-Verfahren erkauft werden - es leidet also die Effizienz. Clusteranalyse-Verfahren lassen sich auf klassische statistische Art oder durch neuronale Netze, und hier insbesonders durch (unüberwacht lernende oder selbstorganisierende) künstliche neuronale Netze, lösen. Die statistischen Verfahren werden nach [JaDu88] in die „partitionierenden Klassifikationen“ und die „hierarchischen Klassifikationen“ unterteilt. Die für die Realisierung von Clusteranalyse-Verfahren geeignetsten neuronale Netze werden Competitive Learning Neuronale Netze genannt. Diese lassen sich wiederum in die sogenannten „Hard Competitive Learning“, auch „Winner-take-all“ genannt, und in die „Soft Competitive Learning“-Verfahren unterteilen. In Abschnitt 5. sollen zunächst die statistischen Verfahren, anschließend die neuronalen Netze-Verfahren vorgestellt werden. Soweit möglich, sollen an entsprechender Stelle Bezüge zum IR hergestellt und Beispiele aufgezeigt werden.

5. Statistische Clusterverfahren

Nach Lance und Williams [LaWi67] (s. a. [JaDu88], S. 55 ff.) lassen sich statistische (hierarchische) Clusterungen nach dem in Abb. 1.3. wiedergegebenen Schema in die allgemeinen Klassifikations-Verfahren einordnen. Die (hierarchischen) Clusterungen sind dabei Spezialfälle der Klassifikationsverfahren, die im folgenden kurz umrissen werden sollen:

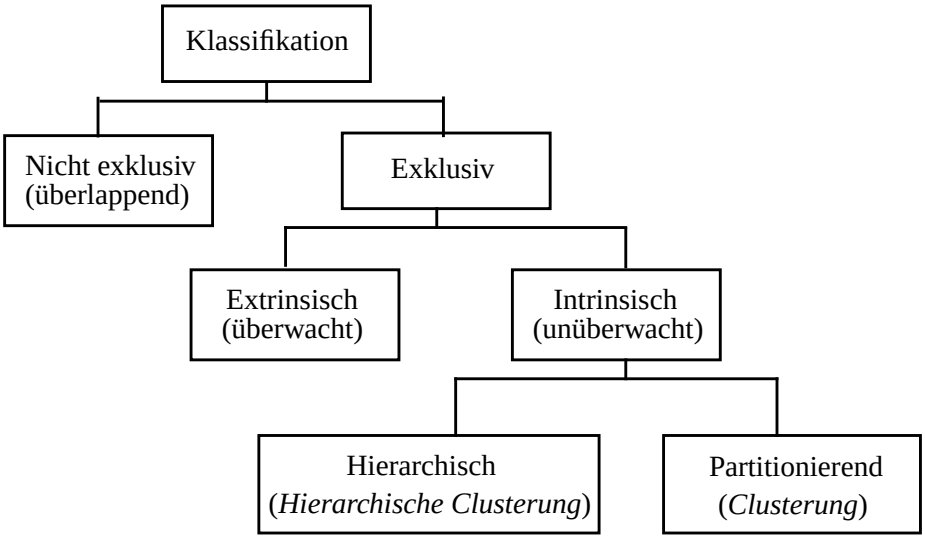


Abb. 1.3.: Klassifikationstypen ([JaDu88], S. 56)

Eine *exklusive* oder *nicht-überlappende Klassifikation* erzeugt eine Partition einer Menge von Objekten, das jedes Objekt genau einer Untermenge oder Cluster zuordnet. *Nicht-exklusive* oder *überlappende Klassifikationen* können dagegen Objekte verschiedenen Clustern zuordnen. *Intrinsische Klassifikationen* nutzen Nachbarschaftsbeziehungen zwischen einzelnen Objekten, d. h. deren Euklidschen Abstände, die gewöhnlich in einer sogenannten Nachbarschaftsmatrix („proximity matrix“) abgelegt werden. Die intrinsische Klassifikationen werden auch unüberwacht genannt, da sie keine a-priori festgelegte Kategorisierung der Partitionierung ausnutzen. *Extrinsische Klassifikationen* hingegen benötigen einen sogenannten „teaching input“, der eben diese Kategorisierung vorgibt. Clusteranalyse-Verfahren beruhen auf intrinsischen Klassifikationen.
Exklusive, intrinsische Klassifikationen werden weiter in die hierarchischen und partitionierenden Klassifikationen unterteilt. Eine *hierarchische Klassifikation* besteht aus ineinander verschachtelten Partitionen, die *partitionierende Klassifikation* besteht nur aus einer einzigen Partition. Jain und Dubes [JaDu88] verwenden den Begriff *Clustering* (*Clusterung*, *Cluster-Analyse*), um eine exklusive, intrinsische, partitionierende Klassifikation zu bezeichnen und den Begriff *Hierarchische Clusterung* (*Hierarchische Cluster-Analyse*) für exklusive, intrinsische, hierarchische Klassifikationen (s. a. [HeHe95], S. 44 ff.). Algorithmisch lassen sich exklusive, intrinsische Klassifikationen zudem auf folgende Weise unterscheiden:

- Sowohl die partitionierende als auch die hierarchische Clusterung kann algorithmisch gesehen auf *agglomerative* Art arbeiten: Sie plaziert dabei zunächst jedes einzelne Objekt in ein eigenes Cluster und faßt diese einfachen Cluster sukzessive zu immer größeren sogenannten „Superclustern“ zusammen. Die *unterteilende* Klassifikation (*divisive* classification) startet demgegenüber bei einer Partition, die alle Objekte vereint und unterteilt diese sukzessive in immer kleinere Subcluster.

- Die Klassifikation kann entweder seriell oder simultan arbeiten. *Serielle* Klassifikationen bearbeiten die Eingabemuster nacheinander, *simultane* Klassifikationen berücksichtigen bei jedem Schritt immer die gesamte Eingabemenge.

- *Monothetische* Klassifikationen behandeln attributbehaftete Objekte, die diese eindeutig einer einzigen Klasse zuordnen. Objekte einer Klasse der *polythetischen* Klassifikation hingegen besitzen Attribute, welche das Objekt verschiedenen Klassen zuordnen können. Überwiegen dabei Attribute, die das Objekt einer bestimmten Klasse zuordnen, so wird das gesamte Objekt dieser Klasse zugeordnet (s. a. Abschnitt 3.1.).

- Cluster können sowohl als Graph unter Zuhilfenahme *graphentheoretischer* Ansätze oder anhand des durchschnittlichen quadratischen Fehlers unter Zuhilfenahme der *Matrixalgebra* gebildet werden.

Im weiteren sollen zunächst einige Algorithmen der hierarchischen Clusterung vorgestellt werden, um anschließend solche der partitionierenden Clusterung aufzuzeigen. Dabei wird deutlich, daß hierarchische Clusterungen eher dazu geeignet sind Clusterhierarchien zu visualisieren. Sie benötigen dazu eine Nachbarschaftsmatrix, in der die Euklidschen Distanzen aller Objekte zueinander eingetragen sind. Die partitionierende Clusterung benötigt dagegen lediglich die Mustermatrix, in der die Objekte in vektorieller Form abgelegt sind. Hierarchische Techniken werden dabei vor allem in der Biologie und in der Sozial- und Verhaltenswissenschaft benutzt, um Taxonomien zu erstellen. Partitionierende Techniken werden vor allem in den Ingenieurswissenschaften benutzt, vor allem um größere Datenmengen effektiv zu komprimieren und zu repräsentieren. Diese Datenkompression, die versucht, einen größeren Datensatz auf nur wenige Referenzvektoren (auch „Codebookvektoren“ genannt) zu reduzieren, wird auch Vektorquantisierung genannt. Dies ist dann besonders praktikabel, wenn der Eingabe-Datensatz aus voneinander separierbaren Modi besteht (s. 5.2.2.). Die Visualisierungstechniken der hierarchischen Clusterung lassen sich bei einigen hundert Daten nicht mehr sinnvoll anwenden. Um größere Mengen von HTML-Dokumenten zu klassifizieren, eignen sich also insbesonders die partitionierenden Clusterungen, auf die in Abschnitt 5.2. näher eingegangen werden soll.

5.1. Hierarchische Clusterung

Eine hierarchische Clusterung ist eine Folge von Partitionen, in der jede Partition in diejenige Partition verschachtelt ist, die durch das nächste Folgenelement beschrieben wird. Dabei ist die Partition **A** in Partition **B** verschachtelt, falls jede Komponente, d. h. Subcluster, von **A** eine echte Untermenge einer Komponente von **B** darstellt. Ein *Dendrogramm* ist eine spezielle Baumstruktur, die das Ergebnis einer hierarchischen Clusterung visualisiert: Blätter repräsentieren die untersten Cluster und werden miteinander verbunden, falls sie Supercluster bilden. Trennt man ein Dendrogramm durch eine horizontale Linie, so entsteht eine Cluster-Struktur einer gewissen Ebene. Die bekanntesten hierarchischen Clusterungsmethoden sind die sogenannten Single-Link und Complete-Link Methoden.

5.1.1. Single-Link Algorithmus

Ausgehend von einer Nachbarschaftsmatrix D, die die Dissimilaritäten aller Objekte vereint, wird im Single-Link Algorithmus ein sogenannter Threshold-Graph erzeugt. Gegeben sei z. B. folgende Nachbarschaftsmatrix („proximity matrix“), die die Dissimilaritäten von 5 Objekten wiedergibt:

$$D = \begin{matrix} \begin{bmatrix} 0 & 2 & 1 & 0 & 3 \\ 2 & 0 & 2 & 1 & 3 \\ 1 & 2 & 0 & 5 & 4 \\ 0 & 1 & 5 & 0 & 6 \\ 3 & 3 & 4 & 6 & 0 \end{bmatrix} & \begin{matrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{matrix} \\ x_1 x_2 x_3 x_4 x_5 \end{matrix} \tag{1.15}$$

Der zu erzeugende Threshold-Graph $G(v)$ unter der Dissimilarität *v* ist ein ungerichteter und ungewichteter Graph von 5 Knoten ohne Schleifen und Mehrfachkanten. Jeder Knoten repräsentiert ein Objekt. Eine Kante zwischen Knoten *i* und *j* wird eingefügt, falls Objekte i und v die Ähnlichkeit v überschreiten. Für obere Nachbarschaftsmatrix und für die Schwellenwert $v := 2$ erhält man also folgenden Threshold-Graph:

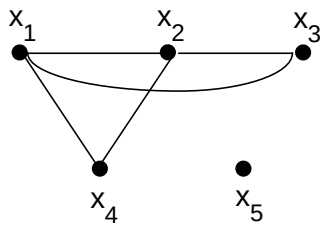


Abb. 1.4.: Threshold-Graph für Schwellenwert v := 2.

Der auf dem Threshold-Graph basierende agglomerative Single-Link Algorithmus kann nun wie folgt formuliert werden:

- *1. Schritt:* Beginne mit einer disjunkten Cluster-Struktur, die durch den Threshold-Graph mit Schwelle $v = 0$ festgelegt wird. Der Threshold-Graph enthält keine Kanten und jedes Objekt erzeugt genau einen Cluster oder Knoten. Setze $k := 1$.

- *2. Schritt:* Konstruiere den Threshold-Graph $G(k)$. Wenn die Anzahl der Komponenten, d. h. die Anzahl der maximal verbundenen Subgraphen, kleiner ist als die Anzahl der Cluster der augenblicklichen Cluster-Struktur, redefiniere die momentane Cluster-Struktur, indem jede Komponente als Cluster bezeichnet wird.

- *3. Schritt:* Wenn $G(k)$ aus einem einzigen verbundenen Graph besteht, beende das Verfahren, ansonsten setze $k := k + 1$ und gehe zu Schritt 2.

Das Threshold-Dendrogramm in Abb. 1.5. veranschaulicht die Cluster-Strukturen in der Reihenfolge der Schwellenwerte, angefangen bei der kleinsten Schwelle $v = 0$ bis zur höchsten $v = v_{MAX}$.

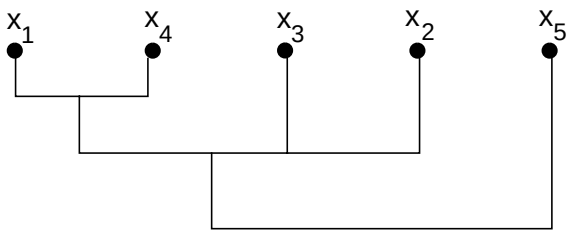


Abb. 1.5.: Threshold-Dendrogramm.

5.1.2. Complete-Link Algorithmus

Analog zum Single-Link läßt sich der agglomerative Complete-Link Algorithmus formulieren:

- *1. Schritt:* Beginne mit Threshold-Graph $G(0)$ wie beim Single-Link Algorithmus.
- *2. Schritt:* Konstruiere Threshold-Graph $G(k)$. Falls zwei der momentanen Cluster einen maximal verbundenen kompletten Subgraph bilden, fasse diese beiden Cluster in ein Supercluster zusammen.
- *3. Schritt:* Wenn $k = n(n-1)/2$, so daß $G(k)$ den kompletten Graph über den *n* Knoten darstellt, beende das Verfahren. Ansonsten setze $k := k + 1$ und gehe zu Schritt 2.

Das Complete-Link Threshold-Dendrogramm basiert auf vollständig verbundenen Subgraphen und entspricht dem Single-Link Threshold-Dendrogramm.

5.2. Clusterung

Wie in 5. dargestellt, eignen sich partitionierende Clusterungen (oder einfach: Clusterungen) eher zur Cluster-Analyse bei großen Mengen von Eingabedaten, wie sie auch bei der Clusterung von HTML-Seiten auftreten. Die Verfahrensweise der Clusterung kann folgendermaßen umrissen werden: Gegeben seien *n* Eingabevektoren oder Muster in einem *d*-dimensionalen Vektorraum. Finde eine Partition, die die Muster in *K* Partitionen oder Cluster aufteilt, so daß Muster, die einem Cluster angehören, sich ähnlicher sind als solche anderer Cluster. Dabei kann die Anzahl der Partitionen *K* im voraus festgelegt werden oder nicht. Allerdings muß zuvor ein Cluster-Kriterium, wie z. B. der mittlere quadratische Fehler, festgelegt werden. Das Kriterium wird dabei als *global* bezeichnet, wenn Cluster durch Prototypen definiert werden und Muster im weiteren Verlauf des Cluster-Verfahrens den ähnlichsten Prototypen zugewiesen werden. *Lokale* Kriterien formen Cluster durch das Erkennen lokaler Strukturen innerhalb der Datenmenge. Das kann z. B. die Entdeckung von Regionen hoher Wahrscheinlichkeitsdichte (auch „Modi“ genannt) sein, oder wenn ein Muster mit seinen *k* nächsten Nachbarn einem Cluster zugeordnet wird. Alle Verfahren arbeiten dabei nach dem gleichen Schema: Wähle ein Cluster-Kriterium aus, evaluiere dieses auf Grundlage der gefundenen Partitionen und wähle diejenige Partition aus, die das Kriterium minimiert. Muster werden dabei von einem Cluster auf ein zweites abgebildet, um das Kriterium zu minimieren. Problematisch erweist sich, daß die Algorithmen oftmals in lokalen Minima terminieren. Eine andere Möglichkeit, die Rechenzeit gering zu halten, ist, solche Partitionen zu verwerfen und nicht weiter zu betrachten, die wahrscheinlich nicht in der Lage sind, das Kriterium zu minimieren.

Nach Jain und Dubes gibt es allerdings kein „bestes“ Cluster-Kriterium, da auch keine präzise Definition eines Clusters existiert. Zudem können Cluster unterschiedliche Größen und Formen im multidimensionalen Raum annehmen.
Im folgenden sollen die gebräuchlichsten Algorithmen der Clusterungen kurz umrissen werden: die Ansätze des Quadratischen Fehlers und K-Means, der Dichteabschätzung oder Modus-Suche, des Minimal aufspannenden Baumes, des Nächsten Nachbarn und des Fuzzy Clusterings (s. a. [JaDu88], S. 89 ff.):

5.2.1. Quadratischer Fehler und K-Means

Das Cluster-Kriterium des quadratischen Fehlers versucht diejenige Partition zu finden, die für eine feste Anzahl von Clustern den quadratischen Fehler minimiert. Angenommen, die gegebene Menge von *n* Mustern eines *d*-dimensionalen Raumes wurde in *K* Cluster $\{C_1, C_2, \ldots, C_K\}$ partitioniert, so daß Cluster C_k n_k Muster einthält und jedes Muster genau einem Cluster zugeordnet ist. Der Mittelpunktsvektor, das Cluster-Zentrum oder Cluster-Centroid des Clusters C_k wird dann wie folgt definiert:

$$\boldsymbol{m}^{(k)} = \frac{1}{n_k}\sum_{i=1}^{n_k} x_i^{(k)} \tag{1.16}$$

wobei $x_i^{(k)}$ das i-te Muster darstellt, das zu Cluster C_k gehört. Der quadratische Fehler für Cluster C_k, auch „Cluster-Variation“ („within-cluster variation“) genannt, ist die Summe der quadrierten Euklidschen Distanzen zwischen jedem Muster und dem Cluster-Centroid $\boldsymbol{m}^{(k)}$:

$$e_k^2 = \sum_{i=1}^{n_k} (x_i^{(k)} - \boldsymbol{m}^{(k)})(x_i^{(k)} - \boldsymbol{m}^{(k)}) \tag{1.17}$$

Der quadratische Fehler für die gesamte Partitionierung, die *K* Cluster enthält, ist die Summe der Cluster-Variationen über alle Cluster:

$$E_K^2 = \sum_{k=1}^{K} e_k^2 \tag{1.18}$$

Das Ziel ist nun, eine solche Partition über *K* Cluster zu finden, die bei festem *K* E_K^2 minimieren. Die gefundene Partition besitzt minimale Varianz.

Der bekannteste iterative Cluster-Algorithmus, der das quadratische Fehlerkriterium implementiert, ist das *K-Means* Verfahren von McQueen, das dieser bereits im Jahre 1967 vorgestellt hat ([McQu67], s. a. [JaDu88], S. 96 ff.):

1. *Schritt:* Wähle eine initiale Partition mit *K* Clustern und wiederhole die Schritte 2 bis 5, bis die Cluster-Zugehörigkeit sich stabilisiert hat.

2. *Schritt:* Generiere eine neue Partition, indem jedes Muster seinem nächsten Cluster-Centroid zugeordnet wird.

3. *Schritt:* Berechne die Clustermittelpunkte oder Cluster-Centroiden nach jedem K-Means Zyklus („K-means pass“), d. h. nachdem ein beliebiges Muster einem Cluster zugeordnet wurde neu.[3]

4. *Schritt:* Wiederhole Schritte 2 und 3, bis die quadratische Fehlerfunktion ein (lokales) Mini-

mum gefunden hat, eine maximale Anzahl von Iterationen erreicht ist oder die Fehlerfunktion zwischen zwei aufeinanderfolgenden Zyklen nicht verkleinert werden konnte.

5. *Schritt:* Verwerfe kleine und außenliegende Cluster und fasse dicht beieinander liegende Cluster zusammen. Zu weit entfernt liegende Cluster werden dabei als Rauschen oder Codierungsfehler betrachtet. Ein Cluster kann zudem in zwei oder mehrere Cluster aufgetrennt werden, falls es zu viele Muster enthält, oder die Varianz des Merkmals mit dem größten Wertebereich eine maximale Varianz übersteigt.

5.2.2. Dichteabschätzung und Modus-Suche

Cluster werden im Ansatz der Dichteabschätzung oder Modus-Suche als Orte hoher Wahrscheinlichkeitsdichte des zugrundeliegenden Merkmalsraums, auch *Modi* genannt, definiert. Ein *Modus* ist also ein Ort, an dem sich Merkmalsvektoren konzentrieren, im Gegensatz zu Orten, an denen die Merkmalsvektoren nur recht spärlich vorkommen. Jeder Modus besitzt ein Modus-Zentrum, und Merkmalsvektoren werden solchen Modus-Zentren zugeteilt, die diesen am nächsten zu liegen kommen. Die Wahrscheinlichkeitsdichte-Abschätzung, d. h. der Erwartungswert der Wahrscheinlichkeitsdichte an einem Punkt *x* des Merkmalvektorraums ist proportional zur Anzahl der Merkmalsvektoren k_n, die innerhalb einer Umgebung V_n von *x* auftreten:

$$E(p_n(x)) = \frac{k_n / n}{V_n} \tag{1.19}$$

n bezeichnet dabei die gesamte Anzahl von Mustervektoren des Merkmalvektorraums und $p_n(x)$ die Wahrscheinlichkeitsdichte-Funktion um Vektor *x*. Der Erwartungswert bzw. der Mittelwert $E(.)$[4] wird in dichten Regionen hoch sein, da bei fester Umgebung V_n die Anzahl der in dieser Umgebung liegenden Mustervektoren k_n groß ist. Die Wahl der Umgebung V_n ist bei einer kleinen Anzahl von Mustervektoren *n* kritisch und wird entweder durch den Ansatz des *Parzen window* oder durch den *nächsten Nachbar*-Ansatz festgelegt. Im ersten Ansatz wird V_n, im zweiten Ansatz k_n als Funktion von *n* betrachtet. Dabei bleibt die Umgebungsgröße im ersten Ansatz konstant und ist im zweiten Ansatz abhängig vom Ort der Merkmalsvektoren.
Falls die Anzahl der Mustervektoren gegenüber der Anzahl der Merkmale (Dimension des Merkmalvektorraums) groß genug ist, kann ein Modus am einfachsten dadurch bestimmt werden, indem der Vektorraum in nicht-überlappende Zellen unterteilt wird. Eine Zelle, die eine relativ große Anzahl von Vektoren umfaßt, stellt einen potentielle Modus dar. Im unimodalen Fall können Histogramme, die die Anzahl der Vektoren über den gesamten Merkmalraum in einem Graph auftragen, dazu benutzt werden, eine solche Zelle über die „Histogrammtäler“ zu separieren. Zellen reagieren allerdings empfindlich auf ihre Größe, da eine falsche Zellengröße die Wahrscheinlichkeitsdichte innerhalb der Zelle nur sehr verrauscht wiedergibt. Ein interessantes Verfahren, das diesen Histogramm-Ansatz auf höherdimensionale und multimodale Räume (d. h. auf Merkmalsräume, die mehr als einen Modus enthalten) überträgt, ist das „BANG-Clustering System“ von Schikuta und Erhart ([ScEr97], s. a. [Schi96]): Sie kombinieren

3. Nach Forgys Methode [Forg65] werden die Cluster-Centroiden erst dann neu berechnet, wenn alle Muster neuen Clustern zugeordnet wurden. Friedman und Rubin [FrRu67] wiederum benutzen einen „Hill-Climbing pass“, der es einem Muster erst dann erlaubt, einem neuen Cluster zugeordnet zu werden, wenn hierdurch bereits das Fehlerkriterium minimiert wird.

4. Der Erwartungswert eines diskreten und abzählbaren Ereignisraumes *X* mit gleicher Wahrscheinlichkeit der Einzelereignisse $P(X = x_i)$ ist: $E(X) := \sum_{i \in I} x_i P(X = x_i) = 1/|X| \sum_{i \in I} x_i$

den „Zellenansatz“ mit einer Nachbarschaftsbeziehung zwischen Zellen unterschiedlichster Auflösung. Die Speicherung der Zellen unterschiedlicher Auflösungsstufen erfolgt hier durch einen Binärbaum, der die Zellenverfeinerung repräsentiert. Die Autoren stellen fest, daß sie mit ihrem Ansatz bis zu einer Million 2-dimensionale Mustervektoren klassifizieren können, geben jedoch zugleich zu, daß sie hierzu 48 MByte Speicherplatz benötigen.
Kittler, der in seinem Parzen window-Ansatz hyperkubische Funktionen [Kitt76] benutzt, verfährt wie folgt: Der erste Mustervektor wird zufälligerweise ausgewählt und ein zweiter wird genau dann zum ersten hinzugefügt, wenn dieser eine maximale Dichte in der gegebenen hyperkubischen Fensterfunktion um den ersten Mustervektor besitzt. Der dritte Vektor muß nun maximale Dichte in der Funktion besitzen, die durch die Vereinigung der beiden ersten hyperbolischen Fensterfunktionen entsteht. Gleichermaßen wird mit allen weiteren Vektoren verfahren. Schließlich entsteht eine eindimensionale Folge von Dichteabschätzungen, deren größte Werte dichte Regionen oder Modi bezeichnen.
Die Dichte kann auch durch die Methode des k_n*-nächsten Nachbarn* abgeschätzt werden [WoLa83]: Zwei Mustervektoren x_i und x_j werden Nachbarn genannt, wenn x_i unter den k_n nächsten Nachbarn von x_j gehört und umgekehrt. Die Dissimilarität zwischen den beiden Mustervektoren x_i und x_j wird dabei durch

$$d(x_i, x_j) = \frac{1}{2E(p_n(x_i))} + \frac{1}{2E(p_n(x_j))} \tag{1.20}$$

gegeben. Mustervektoren, die nicht k_n-benachbart sind, erhalten maximale Dissimilaritäten. Auf die so generierte Dissimilaritätsmatrix wird schließlich die Single-link Methode (s. 5.1.1.) angewandt, um eine hierarchische Cluster-Analyse zu erhalten.

5.2.3. Minimal aufspannender Baum

Graphentheoretische Ansätze, wie der des *Minimal aufspannenden Baumes* (*minimum spanning tree*; *MST*) von Zahn [Zahn71], eignen sich besonders zur Erkennung globulärer, d. h. nicht-gaußverteilter oder irregulär geformter Cluster (s. a. [JaDu88], 120 ff.). Der MST besteht aus Knoten, die die Mustervektoren repräsentieren, sowie Kanten zwischen den Knoten, deren Kantengewichte die Distanzen zwischen den entsprechenden Mustervektoren darstellen. Der MST Algorithmus kann wie folgt beschrieben werden:

1. *Schritt:* Konstruiere den MST für alle *n* Mustervektoren.

2. *Schritt:* Identifiziere inkonsistente Kanten des MST. Eine inkonsistente Kante liegt dann vor, wenn ihr Gewicht, d. h. die Distanz zwischen zwei Mustervektoren, signifikant über dem Durchschnitt der Gewichte benachbarter Kanten liegt. Hierzu kann auch die Standardabweichung benutzt werden.

3. *Schritt:* Entferne alle inkonsistente Kanten aus Schritt 2 und nenne die noch verbundenen Bäume Cluster (s. a. Abb. 1.6.).

Wird der Ansatz des MST in Vektorräumen angewandt, die mehr als zwei Dimensionen besitzen, müssen spezielle Heuristiken herangezogen werden, um inkonsistente Kanten detektieren zu können.
Weitere graphentheoretische Ansätze, wie die des Relativen Nachbarschaftsgraphen (RNG), des Gabriel Graphs (GG) und der Delaunay Triangulierung (DT) sind bekannt, wobei die Graphen im Hinblick auf die Anzahl der verwendeten Kanten folgende Einbettung besitzen:

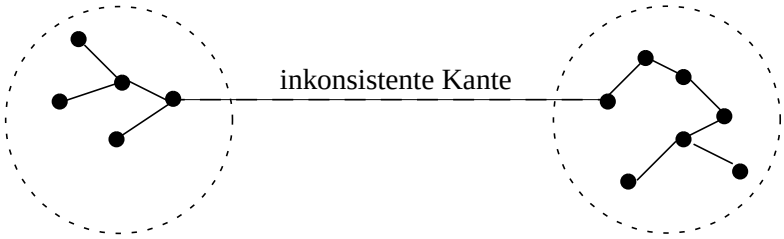


Abb. 1.6.: Inkonsistente Kante und erzeugte Cluster in einem 2-dimensionalen MST.

$$K(MST) \subseteq K(RNG) \subseteq K(GG) \subseteq K(DT) \tag{1.21}$$

$K(\,.\,)$ bezeichnet hierbei die Menge der verwendeten Kanten. Die DT beruht auf der „Dirichlet-Tesselation“ („Dirichlet-Mosaik“) bzw. dem „Voronoi-Diagramm“, die den *n*-dimensionalen Raum so in Zellen um jeden Mustervektor x_i einteilt, daß alle Vektoren in jeder dieser Zelle um x_i näher zu x_i als zu jedem anderen Mustervektor zu liegen kommen. Die Zellgrenzen teilen damit die Kanten zwischen zwei Mustervektoren genau in der Mitte. Eine Kante der DT existiert nun genau dann, wenn die beiden Mustervektoren eine gemeinsame Zellengrenze besitzen. Ein Mustervektor kann dabei in der DT mehr als eine Kante besitzen, im MST jedoch immer nur eine einzige (vgl. Abb. 1.7.).

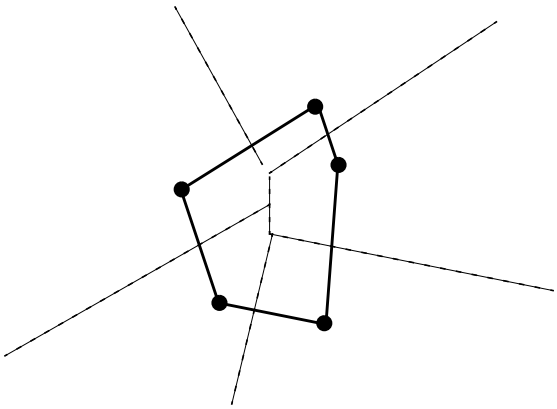


Abb. 1.7.: Voronoi-Diagramm mit zugehörigen Zellgrenzen von fünf Mustervektoren des 2-dimensionalen Raumes.

5.2.4. Nächster Nachbar

Der Ansatz des nächsten Nachbarn („nearest neighbor“), der bereits in den graphentheoretischen Methoden des vorherigen Abschnitts Erwähnung fand, wird auch von Lu und Fu benutzt, um Cluster zu detektieren [LuFu78]. Die Anzahl der durch diese Methode erzeugten Cluster ist dabei abhängig vom Schwellenwert *t*, der vorher festgelgt werden muß:

1. *Schritt:* Setze $i := 1$, $k := 1$ und ordne x_1 C_1 zu.

2. *Schritt:* Setze $i := i + 1$. Finde den nächsten Nachbarn von x_i unter den Mustervektoren, die

bereits Clustern zugeordnet wurden. Sei dieser nächste Nachbar in Cluster *m* und habe Distanz d_m von x_i. (Alternativ kann die durchschnittliche Distanz von x_i zu den *p* nächsten Nachbarn des Clusters *m* betrachtet werden.)

3. *Schritt:* Wenn $d_m \leq t$ für einen Schwellenwert *t*, dann teile x_i Cluster C_m zu, ansonsten setze k := k + 1 und teile x_i einem neuen Cluster C_k zu.

4. *Schritt:* Wenn jeder Mustervektor einem Cluster zugeteilt worden ist, beende die Verarbeitung, ansonsten gehe zu Schritt 2.

Eine alternative Nachbarschaftsrelation ordnet zwei Mustervektoren x_i und x_j genau dann demselben Cluster zu, falls x_i und x_j *k*-nächste Nachbarn sind und zugleich k_t gemeinsame nächste Nachbarn besitzen. Große Werte von *k* begünstigen dabei die Detektierung globulärer Cluster, während kleine Werte von *k* eher längliche Cluster entdecken.
Eine weitere Alternative, die „beidseitige Nachbarschaft“ („mutual neighborhood“), betrachtet Mustervektoren als benachbart, falls x_j der *p* nächste Nachbar von x_i und zugleich x_i der *q* nächste Nachbar von x_j ist, d. h. x_i und x_j den beidseitigen Nachbarschaftswert *(p + q)* besitzen. Je kleiner der beidseitige Nachbarschaftswert *(p + q)* ist, desto ähnlicher sind sich die beiden Mustervektoren.

5.2.5. Fuzzy Clustering

Die von Zadeh [Zad65] entwickelte Fuzzy-Set Theorie (etwa: „Theorie der unscharfen Mengen“) kann insbesonders zur Cluster-Analyse überlappender Cluster verwendet werden: Ein Mustervektor x_i besitzt einen Grad der Clusterzugehörigkeit zu Cluster *q*: $f_q(x_i) \geq 0$ mit $\sum_q f_q(x_i) = 1$, der die Zuverlässigkeit angibt, mit der x_i Cluster *q* angehört. Mit diesem Zuverlässigkeitsgrad kann ein Mustervektor durchaus zwei Clustern gleichzeitig angehören. Dies gilt jedoch nicht, wenn x_i die Cluster-Wahrscheinlichkeit $P_k(x_i)$ hat. Wahrscheinlichkeitsmaße beziehen sich immer nur auf ein spezifisches Cluster.
Im Fuzzy-Clustering werden zunächst alle Mustervektoren Clustern zugeteilt, und ein Ähnlichkeitsmaß $\delta(x, C_i)$ mißt die Ähnlichkeit von Mustervektor *x* zu Cluster C_i. Die Clusterzugehörigkeitsfunktion kann wie folgt bestimmt werden:

$$f_{C_i}(x) = \frac{P_i \delta(x, C_i)}{\sum_{k=1}^{K} P_k \delta(x, C_k)} \tag{1.22}$$

wobei $P_k = n_k / n$ die relative Größe von Cluster C_k angibt. Das Ähnlichkeitsmaß $\delta(x, C_i)$ kann auf dem Distanz-, Nachbarschafts- oder Wahrscheinlichkeitsmaß basieren. Die Fuzzy Cluster-Analyse versucht nun insgesamt - ähnlich wie bei der Minimierung des quadratischen Fehlers - die sogenannte „induzierte Unschärfe“ („induced fuzzyness“) zu minimieren. Die induzierte Unschärfe ist minimal, wenn $f_{C_k}(x) \in \{0, 1\}$, d. h. wenn die gewonnene Partition nicht unscharf (fuzzy) ist. Weiteres hierzu siehe [JaDu88], S. 130 ff.

In den folgenden Abschnitten sollen solche künstliche neuronale Netze als Erweiterungen zu den zuvor behandelten statistischen Verfahren vorgestellt werden, die sich besonders gut zu Klassifikations- oder Cluster-Aufgaben eignen. Die Cluster-Analyse kann man sich auch als (konventionelle) multivariate Datenanalyse vorstellen, also als eine Abbildung $f: A \rightarrow B$, die deskriptive *m*-dimensionale (d. h. multivariate) Vektoren des Vektorraums *A* auf *n* Kategorien der Kategorien- oder Indikatormenge *B* abbil-

det. Die Cluster-Analyse soll nun nichts anderes leisten, als eine Abbildung f zu finden, die zur Klassifikation der Eingabe geeignet erscheint. Von vielen bekannten neuronalen Netzen, wie z. B. dem Multi-Layer Perceptron (MLP) oder Kohonens unüberwacht lernender Selbst-Organisierender Karte (SOM) ist bekannt, daß sie klassische statistische Aufgaben lösen können - und mehr noch - diese Ansätze auf nicht-lineare Art erweitern. Neuronale Netze sind zudem „datenbasiert“ („data-driven“), im Gegensatz zu den statistischen Verfahren, die als „modellbasiert“ („model-driven“) gelten ([Murt94], S. 284). Dabei hängt es beim neuronalen Ansatz immer wesentlich von der Eingabe (und beim MLP auch von der Ausgabe) ab, welche Aufgabe das neuronale Netz letztlich erfüllt: Das MLP kann je nach zugrunde liegendem Datensatz eine Funktionsapproximation, eine nicht-lineare Regression (bei verrauschten Daten), eine Diskriminanzanalyse (bei zu lernenden diskreten Kategorien der Indikatormenge B), ein nicht-lineares autoregressives Modell (bei vorherzusagenden zeitabhängigen Daten) oder eine Kombination aus diesen leisten.
Murtagh [Murt96, Murt94] benutzt z. B. das MLP, um eine multiple nicht-parametrische und nicht-lineare Diskriminanzanalyse (als überwachte Klassifikation) zu realisieren und vergleicht die Ergebnisse mit denen des k-nächsten Nachbarn-Verfahrens (s. 5.2.4.). Lineare Diskriminanzanalysen versuchen (lineare) Hyperebenen zwischen den Kategorien zu erlernen. Die (überwacht lernende) nicht-lineare Diskriminanzanalyse, d. h. die Eigen-Analyse der zwischen den Klassen definierten Kovarianz-Matrix, wird im Gegensatz dazu in der versteckten Schicht eines dreischichtigen MLP realisiert und setzt voraus, daß die Kategorien in B bekannt sind. Sie lernt nun die nicht-linearen Hyperebenen als Klassengrenzen zwischen den Kategorien in B. Weiter berichtet Murtagh über die Verwendung des MLP als quadratischer Gaußscher und polynomieller Klassifizierer und als Maximum-Likelihood-Estimator. Wird das MLP auto-assoziativ, d. h. selbst-überwachend, benutzt, indem also für die Lerneingabe (Teaching-Input) die Eingabe genutzt wird und das MLP nur eine verdeckte Schicht von Neuronen mit linearen Aktivitätsfunktionen besitzt, so kann gezeigt werden, daß die Neuronen der verdeckten Schicht die Projektionen der Eingaben auf die Hauptkomponenten des Eingaberaumes lernen [BaHo89]. Dabei besitzt die versteckte Schicht sehr viel weniger Neuronen als die Ein- oder Ausgabeschicht: Das MLP realisiert also insgesamt eine Approximation der Hauptkomponentenanalyse des Eingabevektorraumes (principal component analysis; PCA, auch Karhunen-Loeve-Transformation genannt), d. h. eine optimale Datenreduktion (s. a. [Brau95], S. 83 ff.).
Die von Kohonen entwickelte überwacht lernende Vektorquantisierung (LVQ; s. [Koh95], S. 203-218) kann, wie das MLP, zur nicht-parametrischen Diskriminanzanalyse benutzt werden und steht damit wiederum in Relation mit dem Verfahren des k-nächsten Nachbarn. Auch hierbei müssen die Kategorien oder Klassen der Zielmenge B bekannt sein, und die Klassenprototypen (auch „Codebook-Vektoren“ genannt) werden iterativ so den Klassenmitgliedern angepaßt, daß sie diese nach Abschluß des Trainings besser repräsentieren können.
Die Hebbsche Korrelationslernregel $\Delta w := \beta y x$ wiederum adaptiert das zwischen dem Eingabevektor x und Ausgabevektor y gelegene Gewicht w um so stärker, je ähnlicher sich die beiden Vektoren x und y sind. Sie kann dazu erweitert werden, eine Projektion auf die erste Hauptkomponente der zugehörigen Kovarianz-Matrix zu realisieren. Konkurrieren mehrere Hebbsche Neuronen um die Generierung der Hauptkomponenten, so lassen sich mit dieser Methode auch mehrere Hauptkomponenten erlernen. Das geschieht entweder mit Hilfe lateral inhibitierender Neuronen, wie z. B. in den Ansätzen der lateral inhibitierenden selbstorganisierenden zellulären neuronalen Netze [Brau96] oder mit Hilfe spezieller Lernregeln, wie z. B. beim „Generalized Generalized Hebbian Algorithm“ [Hyöt96a, Hyöt96b]. Nicht-lineare Erweiterungen dieser PCA-lernenden neuronalen Netze, d. h. solche, die „nicht-lineare Eigenvektoren“ lernen und somit nicht-gaußverteilte Modi besser komprimieren und charakterisieren können, wurden u. a. von Oja und Karhunen vorgestellt [Oja94, KWV95]. Ein umfassender Überblick über neuronale Netze, die solche nicht-lineare PCAs realisieren, ist bei Karhunen [Karh96] zu finden. In allen diesen (unüberwacht lernenden) Verfahren sind die Kategorien in B, im Gegensatz zur Diskriminanzanalyse, nicht bekannt. Folglich kann auch kein Teaching-Input gelernt werden. Es wird also versucht,

Cluster durch eine komprimierte Darstellungsweise zu beschreiben bzw. zu charakterisieren. Anwendungen dieser Verfahren auf die Klassifikation textueller Dokumente sind u. a. bei Hyötyniemi [Hyöt96b], Anwendungen (sowohl von linearen als auch von nicht-linearen PCA-Netzen) in Bezug auf HTML-Dokumente z. B. bei Heuser et. al. [HBR98] zu finden.

Kohonens unüberwacht lernende Selbst-Organisierende Karte (SOM) stellt schließlich eine nicht-lineare Erweiterung der datenreduzierenden Hauptkomponentenanalyse oder des multidimensionalen Skalierens („multidimensionl scaling“) dar. Im Fall normalisierter Gewichtsvektoren stellt jeder Klassenprototyp eines SOM-Clusters den größten Eigenvektor des zugehörigen lokalen Clusters dar. Der SOM-Algorithmus kann also insgesamt als eine nicht-lineare Abbildung betrachtet werden, die den Eingaberaum zunächst in Unterräume (Cluster) unterteilt, deren Cluster-Prototypen die jeweils wichtigsten Merkmale (größten linearen Eigenvektoren) repräsentieren (s. [Brau96], S. 111). Je nach SOM-Karten- und Eingabedimension kann der SOM-Algorithmus auch zur nicht-linearen Regression genutzt werden [BGPW97].

Besonders die Ansätze des Dichte-Abschätzens und der Modus-Suche, die vor allem bei großen und hoch-dimensionalen Datensätzen ihre Stärke zeigen (vgl. 5.2.2.), lassen sich mit sogenannten „wettbewerbslernenden“ („Competitive Learning“) neuronalen Netzen realisieren. Diese erweitern auf nicht-lineare Weise die Suche nach Orten hoher Wahrscheinlichkeitsdichte im hoch-dimensionalen Eingaberaum. Die Competitive Learning Verfahren können in die sogenannten „Hard Competitive Learning“, auch „Winner-take-all“ und in die „Soft Competitive Learning“-Verfahren weiter unterteilt werden. Die „Soft Competitive Learning“-Verfahren, bei denen nicht nur das gewinnende Neuron (der „Gewinner“), sondern auch einige in deren Nachbarschaft liegende Neuronen adaptiert werden, können weiter in Verfahren mit vorher festgelegter Dimension der Ausgabeschicht, und in solche mit vorher nicht festgelegter Ausgabedimension unterschieden werden. Das einfache Competitive Learning wird allgemein als Vorläufer zur SOM betrachtet. Verschiedene Weiterentwicklungen des Competitive Learnings gelten als Alternative zur SOM. Die SOM selbst ist ein „Soft Competitive Learning“ mit vorher festgelegter Ausgabedimension, hier auch „Ausgabekarte“ genannt. Die weiteren Abschnitte sollen die wichtigsten Varianten des „Competitive Learnings“ näher vorstellen, da sich diese besonders gut dazu eignen, einen hochdimensionalen Eingaberaum aus HTML-Dokumentenvektoren zu clustern, d. h. Dokumentenvektor-Modi zu detektieren. Einen umfassenden Überblick über diese Verfahren liefert auch [Fritz97].

6. Competitive Learning

Mit dem Competitive Learning Algorithmus lassen sich zwei gegensätzliche Ziele verfolgen: zum einen eine Fehlerminimierung, zum anderen eine Entropie-Maximierung. Es seien als Eingabe zum Competitive Learning eine kontinuierliche Signalverteilung $p(\xi)$, erzeugt durch eine kontinuierliche Wahrscheinlichkeitsdichte-Funktion $p(\xi), \xi \in \mathbf{R}^n$, und eine Menge von N Neuronen $\mathbf{A} = \{c_1, c_2, \ldots, c_N\}$ gegeben. Jedes Neuron c besitze einen Referenz- oder Gewichtsvektor $w_c \in \mathbf{R}^n$. Dann ist das Ziel eines fehlerminimierenden Competitive Learning neuronalen Netzes, den

erwarteten Quantisierungs- oder Verzerrungsfehler („distortion error“) zu minimieren. Dies ist gleichbedeutend mit dem Finden geeigneter Werte für die Referenzvektoren w_c, $c \in \boldsymbol{A}$, so daß der Quantisierungsfehler

$$E(p(\xi), \boldsymbol{A}) = \sum_{c \in \boldsymbol{A}} \int_{V_c} \|\xi - w_c\|^2 p(\xi) d\xi \tag{1.23}$$

minimiert wird. V_c stellt dabei die Voronoi-Umgebung um Referenzvektor *c* und $E(\,.\,)$ den Erwartungswert dar (s. a. 5.2.3. und [Fritz97], S. 7 ff.). Im Fall einer diskreten und endlichen Eingabe-Datenmenge $\boldsymbol{D} = \{\xi_1, \ldots, \xi_M\}, \xi_i \in \boldsymbol{R}^n$ und einer Voronoi-Menge R_c um Referenzvektor *c* läßt sich der zu minimierende Erwartungswert (1.23) umformen in

$$E(\boldsymbol{D}, \boldsymbol{A}) = \frac{1}{|\boldsymbol{D}|} \sum_{c \in \boldsymbol{A}} \sum_{\xi \in R_c} \|\xi - w_c\|^2 \tag{1.24}$$

Eine wichtige Anwendung zur Fehlerminimierung ist die Vektorquantisierung, die einen größeren Datensatz auf nur wenige Referenzvektoren (auch „Codebookvektoren“ genannt) zu reduzieren versucht. Dies ist wiederum erst dann praktikabel, wenn der Eingabe-Datensatz geclustert ist, d. h. aus voneinander separierbaren Modi besteht (s. a. 5.2.2.). Die Vektorquantisierung leistet dann nichts anderes als eine Cluster-Analyse des (hochdimensionalen) Datensatzes.

Im Gegensatz zur Fehlerminimierung aus Gleichungen (1.23) und (1.24) versucht die Entropie-Maximierung Referenzvektoren so im Eingaberaum zu verteilen, daß jeder Referenzvektor die gleiche Chance erhält, Gewinner eines Eingabevektors zu werden. Anschaulich können die Unterschiede der beiden Verfahren an dem Beispiel gezeigt werden, in dem die Hälfte der Eingabe in einem Modus konzentriert und die andere Hälfte gleichverteilt in einem Hyperwürfel des Eingaberaumes verteilt ist. Um die Entropie zu maximieren, müssen jeweils 50 % der Referenzvektoren in jede der beiden Regionen verteilt werden. Will man jedoch den Quantisierungsfehler minimieren, so genügt es, ein einziges Neuron in die Mitte des Modus‘ zu plazieren (und den Rest gleich im Hyperwürfel zu verteilen).

Ist es gewünscht, die im Eingabedatensatz vorherrschenden Ähnlichkeitsbeziehungen zu visualisieren, so werden die (hochdimensionalen) Eingabedaten topologieerhaltend auf einen (üblicherweise) zweidimensionalen Ausgabe-Raum (auch „Karte“ genannt) abgebildet. Die erhaltenen Beziehungen zwischen den Eingabedaten lassen sich mit Hilfe der Karte leicht visualisieren. Das dazu benutzte neuronale Netz, wie z. B. die Selbst-Organisierende Karte (SOM), muß dabei im voraus festlegen, welche Dimension die Ausgabekarte hat. Die topologieerhaltende Abbildung wird auch „feature mapping“ genannt.

6.1. Hard Competitive Learning

Das Hard Competitive Learning oder Winner-take-all umfaßt alle solche Methoden, die jeweils nur ein Neuron - das gewinnende Neuron oder den „Gewinner“ - adaptieren. Sie können weiter in die sogenannten Stapel- oder Batch-Verfahren, wie z. B. der LBG- bzw. generalisierter Lloyd-Algorithmus, und Online-Methoden unterschieden werden. Die Online-Methoden können dabei konstante oder über die Zeit abnehmende Lernraten besitzen (s. a. [Fritz97], S. 10 ff.). K-Means (s. a. 5.2.1.) ist ein Beispiel für letztere Methode[5].

5. Dies zeigt, daß einige Verfahren, je nach Sichtweise, sowohl den klassisch-statistischen Verfahren als auch den künstlichen neuronalen Netzen zugeordnet werden können.

Ein bekanntes Problem der Hard Competitive Learning Verfahren sind „tote“ oder unbenutzte Neuronen („dead units“), d. h. solche, die nie eine Eingabe gewinnen, ewig ihre initiale Position beibehalten und also nicht zur Fehlerminimierung beitragen können. Dem versucht man i. a. dadurch zu begegnen, indem man die Referenzvektoren nach der Wahrscheinlichkeitsdichte des Eingabedatenraums zu initialisieren versucht. Zudem finden die Referenzvektoren oft nur ein lokales Minimum der Fehlerfunktion. Das läßt sich umgehen, indem nicht nur ein einziges, sondern mehrere Neuronen im unterschiedlichen Ausmaße die Eingabe gewinnen können, was zum Soft Competitive Learning führt (s. 6.2.).

6.1.1. LBG-Algorithmus

Der *LBG-* (oder *generalisierte Lloyd-*) Algorithmus [LBG80, Forg65, Lloyd57] ist ein Hauptvertreter der *Batch-Verfahren* und eine Erweiterung von Forgys Methode zur Minimierung des quadratischen Fehlers (s. a. 5.2.1.). Er bewegt wiederholt seine Referenzvektoren zum arithmetischen Mittel der Voronoi-Mengen:

1. *Schritt:* Initialisiere die Menge **A** der Neuronen $c_i \; \boldsymbol{A} = \{c_1, c_2, \ldots, c_N\}$ mit zugehörigen Referenzvektoren $w_{c_i} \in \boldsymbol{R}^n$ so, daß die w_{c_i} zufällig mit ausgewählten Eingabevektoren aus **D** zusammenfallen.

2. *Schritt:* Berechne die Voronoi-Menge R_c für alle Neuronen $c \in \boldsymbol{A}$.

3. *Schritt:* Bewege den Referenzvektor jedes Neurons zum Mittelpunkt seiner Voronoi-Menge: $w_c = \frac{1}{|R_c|} \sum_{\xi \in R_c} \xi$

4. *Schritt:* Falls einer der Referenzvektoren in dritten Schritt bewegt wurde, gehe zu Schritt 2, ansonsten gebe die Referenzvektoren aus.

Das LBG konvergiert in einer endlichen Anzahl von Schritten und findet ein lokales Minimum der Fehlerfunktion.

6.1.2. Online-Verfahren mit konstanten und abnehmenden Lernraten

Wenn der Eingabedatensatz zu groß ist, um ihn mit Batch-Verfahren praktikabel verarbeiten zu können, bieten sich *Online-Verfahren* an. Online-Verfahren mit konstanter Lernrate eignen sich dabei im wesentlichen nur bei nicht-stationären, d. h. zeitabhängigen und sich über die Zeit rasch verändernden Daten (s. a. [Fritz97], S. 13 ff.). Bei stationären Daten wird dem gegenüber die Lernrate über der Zeit kontinuierlich verringert, wie z. B. im *K-Means* Ansatz ([McQu67], s.a. 5.2.1.), in dem die Lernrate ε für jedes Neuron $c \in \boldsymbol{A}$ eine harmonische Reihe über die Zeit *t* annimmt: $\varepsilon(t) = 1 / t$. Dies führt dazu, daß der Referenzvektor $w_c(t)$ jedes Neurons *c* das arithmetische Mittel derjenigen Eingabedaten $\xi_1^c, \xi_2^c, \ldots, \xi_t^c$ annimmt, für die das Neuron *c* bisher Gewinner war. Der Nachteil des K-Means besteht darin, daß das Verfahren wegen der divergierenden harmonischen Reihe der Lernrate nicht konvergiert. In Simulationsexperimenten nehmen die Referenzvektoren jedoch rasch Positionen ein, die sie nicht mehr verlassen (vgl. [Fritz97], S. 16). Ritter et. al. [RMS91] schlagen über die Zeit exponentiell abfallende Lernraten vor, wie sie üblicherweise auch im SOM-Algorithmus Verwendung finden:

$$\varepsilon(t) = \varepsilon_i \left(\frac{\varepsilon_f}{\varepsilon_i}\right)^{\frac{t}{t_{max}}} \tag{1.25}$$

ε_i bezeichnet dabei die initiale, ε_f die finale Lernrate, t_{max} ist die Gesamtanzahl der Adaptionsschritte des Verfahrens. Diese Lernrate ist unempfindlicher gegenüber einer schlechten Initialisierung der Referenzvektoren und findet zu Beginn des Lernen eher aus einem lokalen Minimum heraus. Das beruht darauf, daß sie zu Beginn des Trainings einen wesentlich höheren Wert annimmt als die harmonische Reihe des K-Means, also ein gewisses Maß an Rauschen dem System zufügt, das mit der Zeit *t* verringert wird. Fritzke zeigt, daß für viele Datensätze das Hard Competitive Learning mit exponentiell abnehmender Lernrate kleinere mittlere quadratische Fehler erzeugt als Hard Competitive Learning mit konstanter Lernrate und K-Means ([Fritz97], S. 18). Der Online Hard Competitive Learning Algorithmus ist im wesentlichen für konstante und abnehmende Lernraten identisch und soll im folgenden wiedergegeben werden:

1. *Schritt:* Initialisiere die Menge **A** der Neuronen $c_i \; \boldsymbol{A} = \{c_1, c_2, \ldots, c_N\}$ mit zugehörigen Referenzvektoren $w_{c_i} \in \boldsymbol{R}^n$ so, daß die w_{c_i} zufällig mit ausgewählten Eingabevektoren aus **D** zusammenfallen.

2. *Schritt:* Wähle zufällig ein Eingabedatum ξ aus der Eingabe-Datenmenge **D** aus.

3. *Schritt:* Bestimme den Gewinner *s* der Eingabe ξ: $s(\xi) = \arg\min_{c \in \boldsymbol{A}} \|\xi - w_c\|$

4. *Schritt:* Adaptiere den Referenzvektor des Gewinners unter Benutzung der (konstanten oder abnehmenden) Lernrate ε: $\Delta w_s = \varepsilon(\xi - w_s)$

5. *Schritt:* Gehe zu Schritt 2, falls die maximale Anzahl der Adaptionsschritte noch nicht erreicht ist.

6.2. Soft Competitive Learning

Im folgenden sollen die bekanntesten Vertreter des Soft Competitive Learning vorgestellt werden, zunächst solche mit mit vorher festgelegter Ausgabedimensionalität und im Anschluß jene mit unbestimmter Ausgabedimensionalität.

6.2.1. Selbst-Organisierende Karte

Der bekannteste Vertreter eines Soft Competitive Learnings mit vorher festgelegter Dimensionalität der Ausgabeschicht ist Kohonens *Selbst-Organisierende Karte* (*Self-Organising Feature Map; SOM* [Koh82, Koh84, Koh95]). Die SOM benutzt eine (üblicherweise zwei-dimensionale) Ausgabeschicht, auch „Karte“ genannt, eine mit der Zeit abnehmende Lernrate und einen zur Karte gehörenden Nachbarschaftsradius oder -kern, der ebenfalls mit der Anzahl der Adaptionsschritte abnimmt.

Kohonens Algorithmus nimmt dabei als Eingabe eine Menge von Eingabedaten aus der Eingabe-Datenmenge **D**, welche jeweils durch *n*-dimensionale Vektoren beschrieben werden, und bildet diese auf Neuronen ab, die regelmäßig auf der Kartenstruktur angeordnet sind. Jede Eingabedimension heißt *Merkmal*

(*feature*). Jedes Kartenneuron besitzt außer seiner räumlichen Anordnung auf der (zwei-dimensionalen) Karte einen Referenz- oder Gewichtsvektor w_c, $c \in \boldsymbol{A}$, der in Kohonens Ansatz auch Codebook-Vektor genannt wird, und der die Verbindung zwischen Eingabevektor und Neuron darstellt. Jedem Kartenneuron wird im Kohonen-Lernschritt ein *n*-dimensionaler Eingabevektor zugeordnet. Die vektoriellen Komponenten dieses Kartenneurons kann auch als das *Gewicht* (*weight*) der Verbindung zwischen Eingabevektor und entsprechendem Neuron angesehen werden. Initiale Gewichtskomponenten sind kleine zufällige Werte. Sie werden durch folgenden Lernprozeß iterativ angepaßt (s. a. [Zell94], S. 138 ff.):

1. *Schritt:* Bestimme die Anzahl der Neuronen $N = N_1 \cdot N_2$ in der (zweidimensionalen) Ausgabeschicht $\boldsymbol{A} = \{c_1, c_2, \ldots, c_N\}$

2. *Schritt:* Wähle einen Eingabevektor ξ zufällig aus der Menge aller Eingabevektoren **D** aus.

3. *Schritt:* Finde den Gewinner *s* der Eingabe ξ, d. h. dasjenige Kartenneuron, dessen Gewichtsvektor am nächsten zum Eingabevektor ist, unter Benutzung des Euklidschen Abstandes $s(\xi) = \arg\min_{c \in \boldsymbol{A}} \|\xi - w_c\|$ oder des Skalarproduktes, falls alle Gewichte zuvor normiert wurden.

4. *Schritt:* Passe den Gewichtsvektor des gewinnenden Neurons w_s so an, daß er im *n*-dimensionalen Vektorraum noch näher zum aktuellen Eingabevektor rückt.

5. *Schritt:* Passe ebenso die Gewichtsvektoren derjenigen Kartenneuronen an, die auf der (zweidimensionalen) Kartenstruktur zum gewinnenden Neuron benachbart zu liegen kommen. Dabei wird eine Nachbarschaftsfunktion $h_{rs} = e^{\frac{-(d(r,s))^2}{2\sigma^2}}$ und eine Distanzfunktion $d(r, s)$ zwischen Gewinner $s = a_{ij}$ und einem beliebigen Kartenneuron $r = a_{km}$ benutzt, die angeben, welche Kartenneuronen adaptiert werden. Die Distanzfunktion $d(r, s)$ ist dabei die L_1-Norm, auch „Manhattan-Distanz“ genannt: $d(r, s) = |i - k| + |j - m|$. Die Standardabweichung, und damit der Nachbarschaftskern h_{rs}, wird dabei über die Zeit *t* durch die Gleichung $\sigma(t) = \sigma_i\left(\frac{\sigma_f}{\sigma_i}\right)^{\frac{t}{t_{max}}}$ verkleinert. Gleiches gilt für den Lernparameter $\varepsilon(t) = \varepsilon_i\left(\frac{\varepsilon_f}{\varepsilon_i}\right)^{\frac{t}{t_{max}}}$ (bei Kohonen auch „gain term“ genannt; s. a. Gl. (1.25)).
   Der Trainings- oder Adaptionschritt aus Schritt 4 und 5 kann zusammenfassend wie folgt formuliert werden:
   $\Delta w_r = \varepsilon(t) h_{rs}(\xi - w_r)$ oder $w_r(t+1) = w_r(t) + \varepsilon(t) h_{rs}(\xi(t) - w_r(t))$

6. *Schritt:* Falls $t < t_{max}$, gehe zu Schritt 2.

Wenn der Trainingsprozeß beendet ist, wurde jedes Eingabedokument topologieerhaltend auf ein Kartenneuron abgebildet. Der Nachbarschaftskern h_{rs} bezeichnet dabei die bekannte Gaußsche Glockenkurve. Auch weitere Nachbarschaftskerne, wie z. B. zylindrische oder konische Kerne, die Kosinusfunktion und einfache, nicht differenzierbare Kerne, werden verwandt und sind in [Zell94], S. 140 beschrieben.

Das Verfahren wird - wie alle anderen Competitive Learning Verfahren- durch die Tatsache charakterisiert, daß der oben bezeichnete Lernschritt keine „externe“ Lerneingabe (teaching input) benötigt, der Algorithmus also einen unüberwachten oder selbstorganisierenden Lernprozeß darstellt. Um zu garantieren, daß dieser selbstorganisierende Lernprozeß zufriedenstellend funktioniert, müssen dem Verfahren zwei Kontrollmechanismen vorgestellt werden: Der erste - nämlich h_{rs} - bewirkt ein graduelles Schrumpfen des Nachbarschaftsradius‘ während der gesamten Lernzeit. Ein weiter Nachbarschaftsradius hilft dabei der Karte, eine globale Ordnung herzustellen, wohingegen eine kleine Nachbarschaftsbeziehung zwischen Kartenneuronen die Karte vor allem auf stabile Art konvergieren läßt. Der zweite Mechanismus wird durch den adaptiven Lernparameter $\varepsilon(t)$ ausgedrückt, der langsam den Wert *0* annimmt. Beim Wert *0* ist die Gewichtsänderung schließlich beendet, und die Karte hat konvergiert. Insgesamt stellt der selbstorganisierende SOM-Algorithmus - wie bereis erwähnt - einen nachbarschafts- oder topologieerhaltenden Prozeß dar, d. h. Eingabevektoren, die im Eingaberaum benachbart zu liegen kommen, werden auch auf benachbarte Kartenneurone abgebildet. Auf diese Art erhält der SOM-Algorithmus die topologischen Beziehungen des Eingaberaumes und überträgt diese auf seine (sehr viel niedrigdimensionalere) Kartenstruktur. Nachbarschaftsbeziehungen zwischen Eingabevektoren können nun auf einfache Art durch geeignete (zwei-dimensionale) graphische Ausgabetechniken visualisiert werden und erlauben eine einfache Klassifizierung des Eingaberaums. In dieser Hinsicht steht die SOM zwar in Beziehung zu adaptiven, d. h. neuronalen, Erweiterungen des K-Means, ihre Hauptaufgabe liegt jedoch eher in der (algorithmisch sehr viel aufwendigeren) Konstruktion der topographischen Merkmalskarte, die zur Datenvisualisierung verwendet werden kann (s. [XKO93], S. 636 und [Fritz96], S. 63).

6.2.2. Growing Cell Structures

Die „wachsenden Zellstrukturen“ („Growing Cell Structures; GCS“) von B. Fritzke [Fritz92, Fritz94] haben, wie die SOM, eine a-priori festgelegte Netzwerkdimensionalität. Im Gegensatz zur SOM können die GCS jedoch „wachsen“, indem an Orten, bzw. Neuronen, hoher (quadratischer) Fehler weitere zu den bereits vorhandenen Neuronen hinzugefügt werden, um diesen Fehler zu minimieren. Diesem Hinzufügen von Neuronen liegt der Gedanke zugrunde, daß an diesen Orten die Wahrscheinlichkeitsdichte vermutlich am höchsten ist. Das Modell besitzt für jede vorher festzulegende Ausgabedimensionalität „Basiselemente“ bzw. „Hypertetraheder“, aus denen das neuronale Netz aufbaut. Diese sind bei eindimensionalen Netzen eine Linie, bei zweidimensionalen Netzen ein Dreieck und ein Tetraheder bei dreidimensionalen Netzen. Der Lernalgorithmus der GCS ist im folgenden beschrieben (s. a. [Fritz97], S. 31 ff.):

1. *Schritt:* Wähle eine Netzwerkdimensionalität *k*. Initialisiere **A** mit *k+1* Neuronen $\boldsymbol{A} = \{c_1, c_2, \ldots, c_{k+1}\}$ und verbinde alle Neuronen miteinander, so daß sie ein *k*-dimensionales Basiselement bilden. Initialisiere die zugehörigen Refernzvektoren $w_{c_i} \in \boldsymbol{R}^n$ zufällig mit Werten aus **D**.

2. *Schritt:* Wähle einen Eingabevektor ξ zufällig aus der Menge aller Eingabevektoren **D** aus.

3. *Schritt:* Bestimme den Gewinner *s* der Eingabe ξ: $s(\xi) = \arg\min_{c \in \boldsymbol{A}} \|\xi - w_c\|$

4. *Schritt:* Addiere die quadrierte Distanz zwischen Eingabe- und Referenzvektor des Gewinners *s* zur lokalen Fehlervariablen E_s des Gewinners: $\Delta E_s = \|\xi - w_s\|^2$

5. *Schritt:* Adaptiere sowohl den Gewinner *s* als auch seine direkten topologischen Nachbarn $i \in N_s$ mit Hilfe der Lernparameter ε_b bzw. ε_n:

$$\Delta w_s = \varepsilon_b(\xi - w_s) \tag{1.26}$$

$$(\Delta w_i = \varepsilon_n(\xi - w_i)) \qquad \forall i \in N_s \tag{1.27}$$

6. *Schritt:* Wenn die Anzahl der bisher generierten Eingabevektoren ein Mehrfaches des Parameters λ erreicht hat, füge ein neues Neuron nach folgenden Regeln ein:

   - Bestimme das Neuron *q* mit dem höchsten akkumulierten Fehler: $q = \arg\max_{c \in \boldsymbol{A}} E_c$
   - Füge ein neues Neuron *r* in die Mitte der längsten Verbindung, die von Neuron *q* zu seinem Nachbarneuron *f* geht, ein. Um die *k*-dimensionalen Basiselemente zu erhalten, verbinde das neu hinzugefügte Neuron *r* mit all den Neuronen, die zugleich direkte Nachbarn von *q* als auch von *f* sind.
   - Interpoliere den Referenzvektor von Neuron *r* zwischen den Werten von Neuron *q* und *f*: $w_r = (w_q + w_f) / 2$
   - Verkleinere die Fehlervariablen aller Nachbarneuronen von *r* um einen Bruchteil, der durch die Anzahl der topologischen Nachbarn N_r von *r* bestimmt wird:

$$\Delta E_i = -\frac{\alpha}{|N_r|} E_i \qquad (\forall i \in N_r) \tag{1.28}$$

   - Setze die Fehlervariable des neuen Neurons *r* gleich dem mittleren Fehler seiner Nachbarn $i \in N_r$:

$$E_r = \frac{1}{|N_r|} \sum_{i \in N_r} E_i \tag{1.29}$$

7. *Schritt:* Verkleinere die Fehlervarible aller Neuronen $c \in \boldsymbol{A}$: $\Delta E_c = -\beta E_c$

8. *Schritt:* Wenn ein Stopp-Kriterium (z. B. eine festgelegte Netz- oder Fehlergröße) noch nicht erfüllt ist, gehe zu Schritt 2.

Die trainierten GCS bestehen schließlich aus einer Vielzahl miteinander verbundener Basiselemente, welche an Orten hoher quadratischer Fehler, d. h. an Orten hoher Wahrscheinlichkeitsdichte, positioniert werden. Die GCS kann man sich dabei als eine um eine Wachstumskomponente erweiterte SOM vorstellen, wobei sich das neuronale Netz aus einem einzigen Basiselement (z. B. aus einem Dreieck bei zweidimensionalen GCS) heraus zur endgültigen Größe entwickelt. Die GCS sind jedoch eher mit dem Growing Neural Gas (GNG) Algorithmus des nächsten Abschnittes verwandt. Im Gegensatz zum GNG müssen jedoch bei den GCS die Ausgabedimensionalität der neuronalen Netze im voraus festgelegt werden.
Werden die Wachstumskomponenten der GCS auf ein rechteckiges Netz als Basiselement anstelle des Dreiecks angewandt, so entsteht eine echte inkrementelle Variante der SOM: das ebenfalls von B. Fritzke erfundene „Growing Grid“ ([Fritz97], S. 34 ff.). Die Lernschritte des Growing Grids bestehen

aus zwei Phasen: der „Wachstumsphase“ und der „Fine-Tuning Phase“. Die Wachstumsphase dient dazu, das aus 2x2 Neuronen bestehende Basis-Netz bis zur endgültigen Größe anwachsen zu lassen, indem jeweils vollständige Spalten bzw. Reihen dem neuronalen Netz hinzugefügt werden. In der „Fine-Tuning Phase“ wird die Größe des Netzes nicht mehr verändert, sondern nur noch die Refernzvektoren der vorhandenen Neuronen angepaßt. Der Nachbarschaftskern h_{rs} der Wachstums- und Fine-Tuning Phase gleicht dabei dem der SOM, wird im Gegensatz zu diesem aber nicht durch $\sigma(t)$ über die Zeit verkleinert, sondern bleibt konstant. Bedingt durch das Anwachsen des Growing Grids werden dennoch am Anfang recht viele und gegen Ende nur noch sehr wenige Neuronen adaptiert. Weitere Details in Bezug auf die Wachstums- und Fine-Tuning Lernphasen des Growing Grids kann S. 34 in [Fritz97] entnommen werden.
Neben den beschriebenen Competitive Learning Verfahren mit vorher festgelegter Netzwerdimensionalität wurden noch weitere Varianten entwickelt: Bauer und Villmann schlugen 1995 ein Verfahren vor, das ein hyperkubisches Netz generiert [BaVi95]. Im Gegensatz zum Growing Grid bestimmt dieses Verfahren die Dimensionalität des Netzes im voraus. Blackmore und Miikkulainen lassen ihr Netz unter der Bedingung wachsen, daß alle Neuronen auf einem zweidimensionalen Netz zu liegen kommen [BlMi92]. Rodrigues und Almeida fügen durch symmetrische Interpolation zwischen vorhandenen Neuronen einer gewöhnlichen SOM weitere Neuronen hinzu und können dadurch die Traingszeit der SOM herabsetzen [RoAl90].

6.2.3. Growing Neural Gas

Das „wachsende neuronale Gas“ („Growing Neural Gas; GNG“) von B. Fritzke [Fritz94b, Fritz95] basiert auf dem Neuronalen Gas (s. 6.2.5.) und verknüpft als eines der wichtigsten Vertreter eines Competitive Learnings mit vorher nicht festgelegter Netzwerdimensionalität die Wachstumskomponente der Growing Cell Structures (GCS; s. 6.2.2.) mit der Generierung einer Topologie durch das „Competitive Hebbian Learning“, welches 1991 von T. M. Martinetz und K. J. Schulten vorgestellt wurde (s. 6.2.5.). Neue Neuronen werden, wie bei den GCS, in der Nähe solcher Neuronen eingefügt, die den meisten akkumulierte Fehler besitzen. Das GNG startet jedoch, im Gegensatz zu den GCS, mit nur zwei unverbundenen Neuronen:

1. *Schritt:* Initialisiere **A** durch zwei unverbundene Neuronen: $\boldsymbol{A} = \{c_1, c_2\}$ mit zugehörigen Referenzvektoren $w_{c_i} \in \boldsymbol{R}^n$, die mit zufälligen Werten aus **D** belegt werden.

2. *Schritt:* Wähle zufällig ein Eingabedatum ξ aus der Eingabe-Datenmenge **D** aus.

3. *Schritt:* Bestimme den Gewinner s_1 und das zweitnächste Neuron (der „zweite Gewinner“) s_2:

$$s_1 = \arg\min_{c \in \boldsymbol{A}} \|\xi - w_c\| \tag{1.30}$$

$$s_2 = \arg\min_{c \in \boldsymbol{A} \setminus \{s_1\}} \|\xi - w_c\| \tag{1.31}$$

4. *Schritt:* Wenn zwischen s_1 und s_2 noch keine Kante existiert, erzeuge diese. Setze das Alter der Verbindung auf *0* („Refresh“): $\text{age}_{(s_1, s_2)} = 0$

5. *Schritt:* Addiere die quadrierte Distanz zwischen Eingabe- und Gewinnerneuron zu einer lokalen Fehlervariablen E_{s_1}: $\Delta E_{s_1} = \|\xi - w_{s_1}\|^2$

6. *Schritt:* Adaptiere den Referenzvektor sowohl des Gewinners s_1 als auch seiner topologischen Nachbarn $i \in N_{s_1}$ mit Hilfe der Lernraten ε_b bzw. ε_n:

$$\Delta w_{s_1} = \varepsilon_b(\xi - w_{s_1}) \tag{1.32}$$

$$\Delta w_{s_i} = \varepsilon_n(\xi - w_{s_i}) \qquad (\forall i \in N_{s_1}) \tag{1.33}$$

7. *Schritt:* Erhöhe das Alter derjenigen Kanten, die vom Gewinner s_1 zu seinen topologischen Nachbarn ausgehen: $\text{age}_{(s_1, i)} = \text{age}_{(s_1, i)} + 1 \qquad (\forall i \in N_{s_1})$

8. *Schritt:* Entferne solche Kanten, die ein maximales Alter age_{max} überschritten haben. Falls Neuronen entstehen, von denen keine Kanten mehr ausgehen, entferne auch diese Neuronen.

9. *Schritt:* Wenn die Anzahl der bisher generierten Eingaben ein Mehrfaches des Parameters λ erreicht, füge ein Neuron nach folgendem Muster ein:

   - Bestimme das Neuron *q* mit maximalem akkumulierten Fehler: $q = \arg\max_{c \in \mathbf{A}} E_c$
   - Bestimme unter den Nachbarn dieses Neurons *q* dasjenige Neuron *f* mit maximalem akkumuliertem Fehler: $f = \arg\max_{c \in N_q} E_c$
   - Füge ein neues Neuron *r* zu dem neuronalen Netz hinzu und interpoliere seinen Referenzvektor von *q* und *f*: $w_r = (w_q + w_f) / 2$
   - Ersetze die Kante von Neuron *q* nach *f* durch die beiden Kanten von Neuron *q* nach *r* und von Neuron *r* nach *f*.
   - Verkleinere die Fehlervariable von Neuron *q* und *f* um α: $\Delta E_q = -\alpha E_q$, $\Delta E_f = -\alpha E_f$
   - Interpoliere die Fehlervariable von Neuron *r* von Neuron *q* und *f*: $E_r = (E_q + E_f) / 2$

10. *Schritt:* Verkleinere die Fehlervariable aller Neuronen $c \in \mathbf{A}$: $\Delta E_c = -\beta E_c$

11. *Schritt:* Falls ein Stopp-Kriterium (z. B. die Netz- oder Fehlergröße) noch nicht erreicht ist, fahre mit Schritt 2 fort.

Das GNG ist wegen der zuvor nicht festgelegten Netzwerkdimensionalität sehr viel eher in der Lage als die SOM oder die GCS, lokale Strukturen des Datensatzes zu erkennen. So paßt sich die Dimensionalität des GNG automatisch den lokalen Dimensionalitäten des Datensatzes an. Liegen die Daten in verschiedenen Hyperebenen unterschiedlicher Dimensionalitäten, so paßt sich die Dimension des GNG in diesen Bereichen den zugrundeliegenden Dimensionen an; d. h. lokal zweidimensionale Daten werden durch lokale GNG-Strukturen beschrieben, die sich aus zweidimensionalen Basiselementen zusammensetzen, lokal dreidimensionale Daten werden durch örtlich dreidimensionale GNG-Strukturen mit entsprechenden Basiselementen beschrieben, u.s.w. (s. a. Abb. 1.8.)

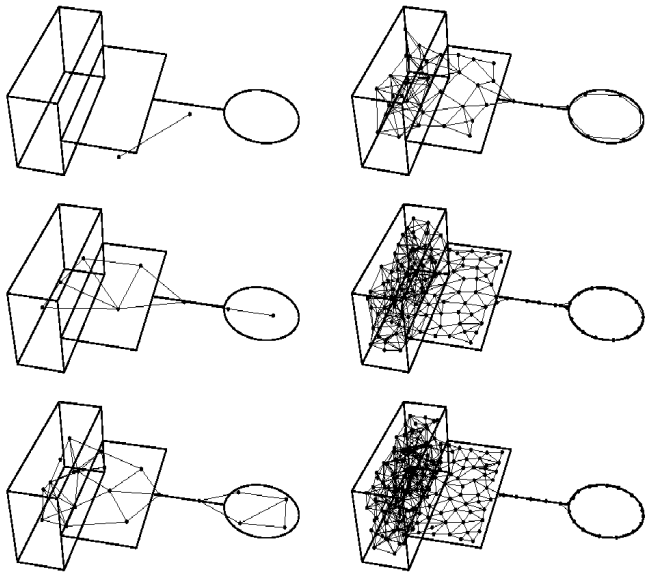


Abb. 1.8.: Das GNG-Netzwerk paßt sich einer Signalverteilung des Eingaberaums an, die verschiedene Dimensionalitäten in verschiedenen Bereiches besitzt. Gezeigt sind das initiale Netz, bestehend aus 2 Neuronen, sowie die Entwicklung des Netzes nach 600, 1.800, 5.000, 15.000 und 20.000 Lernschritten (aus [Fritz95], S. 630).

6.2.4. Rival Penalized Competitive Learning

Das von L. Xu, A. Krzyzak und E. Oja entwickelte „Rival Penalized Competitive Learning“ („RPCL“; [XKO93]) hat wie das GNG keine vorher festgelegte Netzwerkdimensionalität und basiert auf dem „Frequency Sensitive Competitive Learning“ („FSCL“; s. [AKCM90]), das die Adaptionsrate solcher Gewinner reduziert, die bereits häufig Eingaben gewonnen haben. Solche Neuronen werden auch als Neuronen mit „Bewußtsein“ („conscience“) bezeichnet, da sie sich über eine Zählvariable „merken“, wie oft sie bereits Gewinner waren.[6] Dabei besitzt das RPCL im Gegensatz zum GNG nicht nur keine vorher festgelegte, sondern überhaupt keine Netzwerkdimensionalität, d. h. das Verfahren positioniert die Neuronen an Stellen hoher Wahrscheinlichkeitsdichte, wobei die Neuronen untereinander keine Nachbarschaftsbeziehung aufweisen.
In [XKO93] berichten die Autoren, daß die Erkennungsrate von zweidimensional verteilten Modi erhöht werden kann, wenn nicht nur der Gewinner s_w zur Eingabe hin angepaßt wird, sondern auch der zweite Gewinner bzw. Rivale s_r, also das zur Eingabe zweitähnlichste Neuron, um einen Faktor von der Eingabe wegbewegt wird, der kleiner ist als die Lernrate von s_c. Das RPCL kann daher auch als unüberwachte Erweiterung von Kohonens überwacht lernender Vektorquantisierung (LVQ2 [Koh90]; s. a. [Koh95], S. 203-218) betrachtet werden. Beim überwacht lernenden LVQ2 müssen wie beim K-Means (s. 5.2.1.) und im Gegensatz zu unüberwachten Competitive Learning Verfahren die Anzahl der Cluster oder Modi im voraus bekannt sein, wodurch sich das Verfahren nicht für größere und daher im allgemeinen unbekannte Datensätze eignet. Es folgt der RPCL-Algorithmus:

6. Ähnliche Neuronen mit „Bewußtsein“ wurden bereits 1988 von DeSieno entwickelt: Er fügte für alle häufigen Gewinner einen sogenannten „Strafterm“ („penalty term“) zur Distanz zur Eingabe hinzu, so daß andere Neuronen auch eine Chance bekommen konnten [DeSi88].

1. *Schritt:* Initialisiere die Menge **A** der Neuronen c_i $\boldsymbol{A} = \{c_1, c_2, \ldots, c_N\}$ mit zugehörigen Referenzvektoren $w_{c_i} \in \boldsymbol{R}^n$ so, daß die w_{c_i} zufällig mit ausgewählten Eingabevektoren aus **D** zusammenfallen.

2. *Schritt:* Wähle zufällig ein Eingabedatum ξ aus der Eingabe-Datenmenge **D** aus.

3. *Schritt:* Bestimme den Gewinner s_w der Eingabe ξ mit zugehörigem Referenzvektor w_w, so daß $\gamma_w \|\xi - w_w\|^2 = \min_j \gamma_j \|\xi - w_j\|^2$.
   Bestimme zudem den zweiten Gewinner (Rivalen) s_r der Eingabe ξ mit Referenzvektor w_r, so daß $\gamma_r \|\xi - w_r\|^2 = \min_{j \neq w} \gamma_j \|\xi - w_j\|^2$

4. *Schritt:* Adaptiere den Referenzvektor des Gewinners unter Benutzung der (konstanten oder abnehmenden) Gewinner-Lernrate ε_w: $\Delta w_w = \varepsilon_w(\xi - w_w)$.
   Adaptiere zugleich den Referenzvektor des Rivalen unter Benutzung der (konstanten oder abnehmenden) Rivalen-Lernrate ε_r: $\Delta w_r = -\varepsilon_r(\xi - w_r)$.
   Alle sonstigen Neuronen werden nicht adaptiert, d. h. $\Delta w_i = 0$.

5. *Schritt:* Gehe zu Schritt 2, falls die maximale Anzahl der Adaptionsschritte noch nicht erreicht ist.

Der RPCL-Algorithmus verwendet die beiden Lernraten ε_w und ε_r, für die gilt: $0 \leq \varepsilon_w, \varepsilon_r \leq 1$. Sie werden üblicherweise mit der Zeit *t* verkleinert und es gilt ferner zu jedem Zeitpunkt *t*: $\varepsilon_w(t) \gg \varepsilon_r(t)$. Der Parameter γ_j ist die Zählvariable jedes Neurons c_j und mißt, wie oft das entsprechende Neuron bereits eine Eingabe gewonnen hat:

$$\gamma_j = \frac{n_j}{\sum_{i=1}^{N} n_i} \tag{1.34}$$

Nach dem Training des RPCL können Klassen oder Cluster detektiert werden, indem die Referenzvektoren als Cluster-Zentroiden betrachtet und alle Eingabevektoren den jeweils nächsten Zentroiden zugeordnet werden: Wenn also $i = \arg\min_j \|\xi - w_j\|^2$, dann ordne Eingabe ξ dem Cluster mit dem Index *i* zu. Falls einem Referenzvektor weniger als *m* Eingaben zugeordnet wurden, können die entsprechenden Cluster auch verworfen werden (s. [XKO93], S. 642).

Die Autoren zeigen mit Hilfe eines künstlich erzeugten zweidimensionalen modalen Datensatzes, daß das RPCL überschüssige Neuronen aus dem Eingaberaum entfernt, falls mehr Eingabeneuronen als Modi vorhanden sind. Solche überschüssigen Neuronen gewinnen nie oder selten eine Eingabe und werden, nach Xu et. al., als Rivalen nach und nach von den Gewinnern und deren Modi verdrängt und können schließlich verworfen werden (s. Abb. 1.9.). Sind dagegen weniger Neuronen als Modi vorhanden, so drückt sich dies laut Xu et. al. darin aus, daß nach dem Training keine überschüssigen Neuronen vorhanden sind. Die restlichen Neuronen werden deshalb zwischen den überzähligen Modi oszillieren. Die Autoren empfehlen für diesen Fall das Training mit der doppelten Anzahl von Neuronen zu wiederholen.

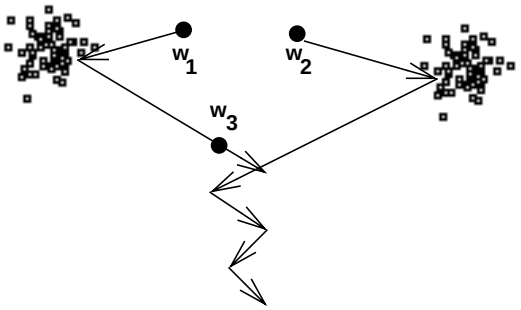


Abb. 1.9.: RPCL: Der Rivale w_3 von w_1 und w_2 wird Schritt für Schritt von den beiden Modi entfernt (nach [XKO93]).

6.2.5. Neuronales Gas

Ein Competitive Learning Verfahren ohne existierende Netzwerkdimensionalität stellt das von T. M. Martinetz und K. J. Schulten erfundene „Neuronale Gas“ („Neural Gas“; [MaSc91]) dar: Das Neuronale Gas bringt nach jeder erzeugten Eingabe und abhängig von den Distanzen zu ξ die Referenzvektoren aller Neuronen in eine Reihung. Sodann werden eine Reihe von Referenzvektoren adaptiert, wobei die Adaptionsrate und die Anzahl der zu adaptierenden Neuronen mit der Zeit abnimmt. Im Gegensatz zur Weiterentwicklung des Growing Neural Gas (GNG; 6.2.3.) besitzt dieses Verfahren keine Netzwerkdimensionalität:

1. *Schritt:* Initialisiere die Menge **A** der Neuronen c_i $\boldsymbol{A} = \{c_1, c_2, \ldots, c_N\}$ mit zugehörigen Referenzvektoren $w_{c_i} \in \boldsymbol{R}^n$ so, daß die w_{c_i} zufällig mit ausgewählten Eingabevektoren aus **D** zusammenfallen. Initialisiere den Zeitparameter $t := 0$.

2. *Schritt:* Wähle zufällig ein Eingabedatum ξ aus der Eingabe-Datenmenge **D** aus.

3. *Schritt:* Bringe alle Neuronen aus **A** abhängig von den Distanzen ihrer Referenzvektoren w_{c_i} zur Eingabe ξ in eine Reihung. Der Index $k_i(\xi, \boldsymbol{A})$ soll dabei die Ordnung von w_{c_i} wiedergeben, angefangen bei $k_i(\xi, \boldsymbol{A}) = 0$, falls w_{c_i} der Gewinner ist, $k_i(\xi, \boldsymbol{A}) = 1$, falls w_{c_i} der zweite Gewinner ist, u.s.w.

4. *Schritt:* Adaptiere die Referenzvektoren durch: $\Delta w_i = \varepsilon(t) \cdot h_\lambda(k_i(\xi, \boldsymbol{A})) \cdot (\xi - w_i)$ mit folgenden Zeitabhängigkeiten:

$$\lambda(t) = \lambda_i \left(\frac{\lambda_f}{\lambda_i}\right)^{t / t_{max}} \tag{1.35}$$

$$\varepsilon(t) = \varepsilon_i \left(\frac{\varepsilon_f}{\varepsilon_i}\right)^{t / t_{max}} \tag{1.36}$$

$$h_\lambda(k) = e^{(-k / \lambda_t)} \tag{1.37}$$

5. *Schritt:* Erhöhe den Zeitparameter $t := t + 1$.

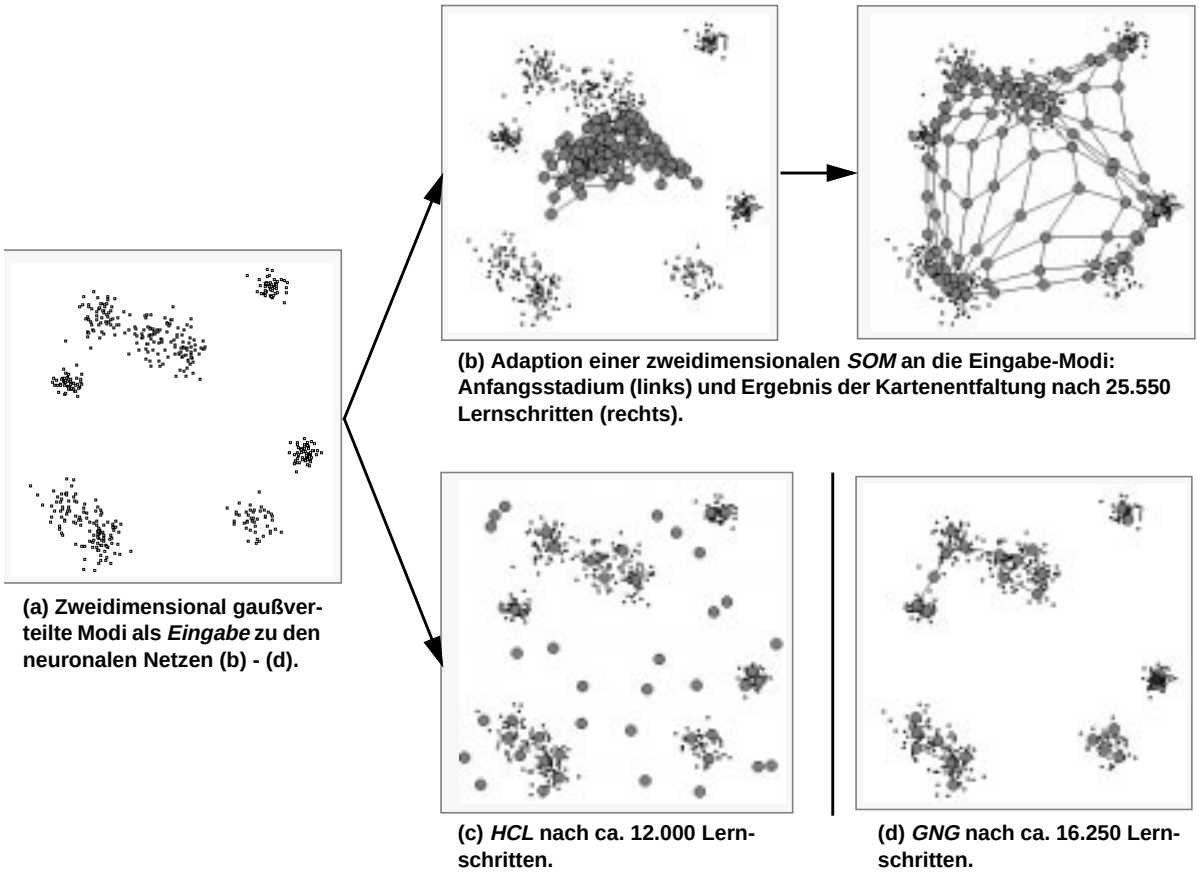
6. *Schritt:* Falls $t < t_{max}$, fahre mit Schritt 2 fort.

λ_i und ε_i sind initiale, λ_f und ε_f finale Parameterwerte.

Eine weitere Competitive Learning Methode mit vorher nicht festgelegter Netzwerkdimensionalität ist das Verfahren von Kangas et. al. [KKL90]: Sie schlagen vor, den minimal aufspannenden Baum (MST; s. a. 5.2.3.) als Nachbarschaftstopologie unter den Neuronen zu benutzen, um die Netzwerkdimension nicht im voraus bestimmen zu müssen.
B. Fritzke wiederum erweitert das LBG-Verfahren aus Abschnitt 6.1.1. um ein „Nützlichkeitsmaß“ („utility measure“): Sein LBG-U („LBG with Utility“; [Fritz97b]) entdeckt solche Codebook-Vektoren, die nichts zur Fehlerminimierung beitragen. Ein Codebook-Vektor w_a, der in diesem Sinne nicht nützlich ist, vergrößert nicht, oder nur in geringem Maße, den quadratischen Fehler, wenn er aus der Menge der Neuronen entfernt wird. Solche unnützen Neuronen werden nun in die Umgebung derjenigen Neuronen w_b bewegt, die den größten Fehler erzeugen. In der Umgebung dieser Neuronen können sich die unnützen nützlich machen. Die Umgebung von w_b wird dabei bestimmt durch dessen Standardabweichung $\sqrt{E(w_b) / |R_b|}$, wobei R_b die Menge der Eingabedaten-Vektoren umfaßt, die am nächsten zu w_b liegen. Die w_a sollen nun deutlich innerhalb dieser Standardabweichung zu liegen kommen. Optimalerweise könnte man w_a in die Richtung des größten Eigenvektors von R_b legen. Dies bedeutet allerdings einen größeren Aufwand bei der Berechnung der Kovarianzmatrix von R_b und dem zugehörigen größten Eigenvektor.

Zum Schluß dieses Abschnittes soll an einem beispielhaften zweidimensionalen multimodalen Datensatz Ergebnisse präsentiert werden, die mit einem Hard Competitive Learning, einem Soft Competitive Learning mit vorher festgelegter Netzwerkdimensionalität und einem Soft Competitive Learning mit vorher nicht festgelegter Netzwerkdimensionalität erzielt wurden. Als stellvertretende Verfahren dieser drei Kategorien soll ein Standard Hard Competitive Learning (HCL), die Selbst-Organisierende Karte (SOM) bzw. das Growing Neural Gas (GNG) benutzt werden, um Modi im vorhandenen Eingaberaum zu detektieren. Die Modi liegen mehr oder weniger gaußverteilt vor (s. Abb. 1.10.)[7]:
Das vorläufige Experiment zeigt, daß die SOM zwar generell alle Modi detektieren kann, sie jedoch auch nach 25.550 Lernschritten immer noch Neuronen besitzt, die nur noch wenig adaptiert werden und somit nicht zur Fehlerminimierung beitragen. Diese Neuronen sind in Abb. 1.10. (b; rechts) in der Mitte des Eingabedatenraumes und zwischen allen Modi zu erkennen. Die topologieerhaltende Abbildung des SOM-Algorithmus‘ „zwingt“ die Neuronen in ein a-priori festgelegtes und unveränderbares Netzwerk und ist daher zur Modusdetektion eher hinderlich. Ähnliche Defizite zeigt das HCL aus Abb. 1.10. (c): Hier werden die zwischen den Modi initialisierten Neuronen bereits nach ca. 12.000 Lernschritten überhaupt nicht mehr adaptiert, da diese wegen ihrer großen Distanz zu den Modi nie die Chance bekommen, eine Eingabe zu gewinnen. Lediglich das GNG (Abb. 1.10. (d)) produziert bereits nach ca. 16.250 Lernschritten zufriedenstellende Ergebnisse, wobei die Cluster-Zentroiden nach abgeschlossenem Training noch durch eine Berechnung des Mittelwerts der Referenzvektoren aller verbundener Neuronen bestimmt werden müssen.

7. Die Untersuchungen wurden mit Hilfe des „DemoGNG v1.3“ Java Applets von H. S. Loos und B. Fritzke der Systems Biophysics, Institute for Neural Computation, Ruhr-Universität Bochum durchgeführt [LoFr97].



(a) Zweidimensional gaußverteilte Modi als *Eingabe* zu den neuronalen Netzen (b) - (d).

(b) Adaption einer zweidimensionalen *SOM* an die Eingabe-Modi: Anfangsstadium (links) und Ergebnis der Kartenentfaltung nach 25.550 Lernschritten (rechts).

(c) *HCL* nach ca. 12.000 Lernschritten.

(d) *GNG* nach ca. 16.250 Lernschritten.

Abb. 1.10.: Detektierung gaußverteilter zweidimensionaler Modi (a) durch die SOM (b), das HCL (c) und das GNG (d).

7. Neuronale Netze und Intelligentes Information Retrieval

Die Aufgabe eines Dokumenten-Suchsystems kann darin gesehen werden, eine Benutzeranfrage, die in einer natürlichen Sprache formuliert wird, mit einer großen Menge natürlichsprachiger Dokumente zu vergleichen. Künstliche neuronale Netze, die in der IR-Gesellschaft auch unter die sogenannten *Machine Learning* (*ML*)-Verfahren eingeordnet werden, sind dafür bekannt, daß sie Mustererkennungs- oder Clusteraufgaben für größere Datenmengen recht problemlos lösen können. Sie bieten sich daher

an, gängige Dokumentensuchtechniken weitestgehend zu automatisieren. In den folgenden Abschnitten soll versucht werden, eine chronologische Reihenfolge der wichtigsten Arbeiten auf diesem Gebiet wiederzugeben. Erst neuere Arbeiten wenden sich dabei der Suche im World-Wide-Web (WWW) zu.

7.1. Dokumenten-Ranking durch Feed-Forward Netze

Frühere Ansätze im Bereich des Information Retrievals benutzten neuronale Netze, Thesauri (s. a. Abschnitt 4.1.) entweder manuell oder automatisch zu konstruieren und anschließend ein Neuron in der versteckten Schicht eines mehrschichtigen Feed-Forward Netzes einem bestimmten Thesaurus-Konzept zuzuordnen. R. Wilkinson, P. Hingston und K. L. Kwok [WiHi91, Kwok89] konstruierten Feed-Forward neuronale Netze, die jedem Neuron entweder genau einen Term eines Dokuments oder einen Anfrageterm zugeordnen. Zusätzlich gibt es - falls sie existieren - in diesen neuronalen Netzen bidirektionale und asymetrische Verbindungen (Gewichte) zwischen jedem Anfrageterm-Neuron und einem entsprechenden Dokumententerm-Neuron (d. h. solche Neurone, die Terme innerhalb des Dokuments repräsentieren), sowie zwischen einem Dokumenten-Neuron und jedem der Dokumententerm-Neuronen. Anfragen und Dokumente werden als Neuronen der gleichen Kategorie betrachtet und sind deshalb entweder Eingabe- oder Ausgabeneuronen. Verbindungen innerhalb der gleichen Schicht sind nicht erlaubt. Ein Neuron erzeugt wie üblich ein Signal, wenn seine Aktivität eine gewisse Grenze überschritten hat. Die Signalstärke, die ein Neuron erreicht, ist dabei das Produkt aus den aktuellen Aktivitäten der sendenden Neuronen und den Gewichten der Verbindungen zwischen diesen. Diese Technik erzeugt zum einen eine Standard-Reihung (ranking) aller Eingabe-Dokumente, basierend auf dem zuvor beschriebenen Kosinusmaß. Zum anderen findet das System Wörter, die basierend auf dem initialen Ranking relevant zu sein scheinen, um diese dann dazu zu verwenden, das initiale Ranking weiter zu verfeinern. Schließlich führt diese Technik auf einfache Art und Weise Relevanz-Feedbacks in das Vektorraummodell ein. Einige Verbesserungen in Bezug auf den Verfeinerungsschritt wurden vorgeschlagen, indem eine neue Neuronenschicht eingeführt wurde, deren Neuronen Termgruppen von solchen Termen repräsentierten, die in mehreren Dokumenten vorkommen.
Kwok führte in seiner Arbeit eine verbesserte initiale Gewichtung der Verbindungen der neuronalen Netze ein, die er „Indizierung basierend auf Prinzipien des Dokumenten-Wiederfindens“ (*document self-recovery*) nannte [Kwok89]. Diese stellt laut Kwok eine optimale Termgewichtung dar, wenn jedes Objekt (Dokument oder Anfrage) als eine „Anfrage“ betrachtet wird und vorausgesetzt wird, daß diese Anfrage sich selbst am besten unter allen anderen Objekten finden sollte. Im Gegensatz zur bekannten Deltaregel des Backpropagation-Lernalgorithmus‘ in Feed-Forward Netzen benutzt Kwok die Relevanz-Informationen nicht als die durch das Netz zu reproduzierende Lerneingabe (teaching input), sondern als Bestätigung, die festlegt, welche Gewichtsverbindung modifiziert werden soll. Was also gelernt werden soll, ist - nach dem probabilistischen Modell - der Erwartungswert der Auftrittswahrscheinlichkeit eines Terms in der Menge der betrachteten relevanten Dokumente. Dadurch würde nach Kwok das probabilistische Suchsystem, das einzelne Terme als Komponenten ihrer Dokumente nutzt und in neuronale Netze implementiert, ein optimales Ranking erzielen, wobei das Ranking auf der Wahrscheinlichkeit der Relevanz eines Dokuments im Hinblick auf eine gegebene Anfrage basiert.

Neuere Anwendungsbeispiele künstlicher neuronaler Netze im Bereich des Information Retrievals gehen weg von Feed-Forward Netzen, konzentrieren sich eher auf Clusteraufgaben und verwenden dabei hauptsächlich Kohonens unüberwacht lernende Selbst-Organisierende Karte (SOM) oder Varianten und Erweiterungen dieses Algorithmus‘ (vgl. Abschnitt 6.2.1. ff.). Ein Grund dafür mag sein, daß

Feed-Forward Netze bei größeren Datenmengen i. d. R. versagen und die SOM bzw. deren Verwandte gerade wegen ihres unüberwachten Lernens Vorteile bei diesen a-priori nicht kategorisierbaren Eingabemengen besitzen. Diese Ansätze sollen im folgenden vorgestellt werden:

7.2. Selbstorganisierende semantische Karten

Lin et. al. [LSM91] benutzen die SOM, um eine *selbstorganisierende semantische Karte* zu erzeugen. Die semantische Karte soll semantische Beziehungen zwischen Eingabedokumenten visualisieren. Sie wandten die SOM auf die Sammlung von Dokumenten der LISA-Datenbank an. Die Sammlung beinhaltet 140 Titel, die durch den beschreibenden Term „Artificial Intelligence“ indiziert wurden. In ihrer Arbeit besitzt ein Dokumentenvektor in einer Komponente den Eintrag *1*, falls das zugehörige Wort im Dokumententitel vorkommt, ansonsten den Eintrag *0* (binäre Repräsentation). Die Dokumentenvektoren werden dazu benutzt, eine Merkmalskarte, bestehend aus 140 Neuronen a 25 Merkmale, zu trainieren. Die trainierte Karte wird in Konzept- (oder Wort-)gebiete unterteilt, welche auf folgende Art gekennzeichnet werden können: Vergleiche alle Einheitsvektoren (ein Vektor, der nur in einer Komponente den Eintrag *1*, ansonsten *0* hat) mit jedem Neuron der Karte und bezeichne das gewinnende Neuron mit dem Wort des zugehörigen Einheitsvektors. Wenn zwei Worte in das gleiche Gebiet fallen, so werden die beiden Gebiete zusammengefaßt. Das ist dann der Fall, wenn die beiden Worte häufig zusammen auftauchen und somit einen gemeinsamen Term repräsentieren (z. B. die beiden Kartengebiete, die durch die Terme „natural language“ oder „machine learning“ gekennzeichnet werden). Das Hauptanliegen der semantischen Karte ist dabei die Dimensionsreduktion, da es den hochdimensionalen Raum der Eingabedokumente auf den der zweidimensionalen Kartenstruktur abbildet. Ein weiteres Beispiel einer Dimensionalitätsreduktions-Technik, auf das an dieser Stelle nicht weiter eingegangen werden kann, ist das sogenannte *Latent Semantic Indexing* [Furn+88]. Diese Methode ist verwandt mit der oben beschriebenen Methode und verwendet den Erwartungswert einer verborgenen (latenten) Struktur, um einen semantischen Raum zu erzeugen, der eine viel geringere Dimensionalität, als die der ursprünglichen Dokumententerm-Vektoren aufweist. Im semantischen Raum werden wiederum Terme und Dokumente, welche miteinander über Assoziationen verbunden sind, nebeneinander abgebildet. Während das Latent Semantic Indexing die Dimension der Eingabevektoren auf einen Wert von ca. *100* reduziert, erweitert die selbstorganisierende semantische Karte dieses Ergebnis, um die Dimension nochmals auf den Wert *2* der (visualisierbaren) Kartendimension zu reduzieren.

7.3. Verarbeitung natürlicher Sprachen durch die SOM

In den Arbeiten von T. Honkela et. al. des Neural Networks Research Centre der Helsinki University of Technology [HPK95, Honk97] wird eine neue Methode vorgestellt, um mit der Selbst-Organisierenden Karte semantische Beziehungen zwischen Wörtern in natürlichen Sprachen anzuzeigen. Die semantischen Beziehungen oder „Rollen“ (roles) der Wörter werden dabei durch die Kontexte wiedergegeben, in denen diese auftreten. Die Arbeiten wurden dadurch motiviert, daß es für viele Methoden, die mit natürlichen Sprachen zu tun haben, wünschenswert wäre, in textanalytischen Verfahren echte Wortbedeutungen (Semantiken) zu verwenden. Da diese immense Aufgabe bis heute nur in unzureichendem Maße durch die Linguistik geleistet werden kann, betrachteten sie statt dessen die Statistiken der Wortkontexte im Text. Diese sollten nach Honkela et. al. in der Lage sein, assoziative Beziehungen zwischen Wörtern zur Verfügung zu stellen, Wortkategorien automatisch zu generieren, und somit Worthistogramme deutlich zu verkleinern. Damit steht die Methode im Zusammenhang mit klassischen Wortka-

tegorisierungs-Methoden, die über Stammbildung und Benutzung von Stopplisten Worthistogramme oder automatisch generierte Thesauri erzeugen („keyword clustering“; s. auch Abschnitt 4.2.1.). Erste Ansätze zu selbstorganisierenden semantischen Karten stammen bereits aus dem Jahre 1989 von H. Ritter und T. Kohonen [RiKo89, RiKo90].
Bei T. Honkela et. al. wird der (ins englische übersetzte) Rohtext der Grimmschen Märchen ohne vorherige semantische oder syntaktische Kategorisierungen der Wörter als Eingabequelle verwendet. Wenn, wie in ihrem Beispiel, das Vokabular sehr groß ist, sollten die Wörter durch quasi-orthogonale Zufallsvektoren einer sehr viel kleineren Dimensionalität kodiert werden, im Gegensatz zu der einfacheren Methode, welche jedem Wort einen Einheitsvektor zuordnet. Der Text aller Märchen wurde zunächst durch Konkatenation in einer einzigen Datei zusammengefaßt; Satzzeichen und Artikel (wie z. B. „a“, „an“, „the“, ...) wurden entfernt, Großbuchstaben wurden in Kleinbuchstaben umgewandelt. Schließlich dienen als Eingabevektoren $x(t)$ Wort-Triplets, bestehend aus „Vorgänger“, „Schlüsselwort“ (key) und „Nachfolgewort“. Die *150* am häufigsten vorkommende Wörter der Textdatei wurden dazu ausgewählt, die Schlüsselwörter der zu trainierenden SOM zu repräsentieren. Die Wörter wurden jeweils durch *90*-dimensionale Vektoren zufälliger reeller Einträge kodiert. Die Eingabevektoren zur SOM bestehen also aus *270*-dimensionalen Vektoren, welche sich aus den drei Vektoren der Triplet-Wörtern zusammensetzen. Die SOM-Größe wurde auf *42x36* Neuronen festgesetzt. Um die Berechnung zu beschleunigen, werden die initialen Kartengewichte aus den beiden größten Eigenvektoren des Eingaberaums abgeleitet. Die Eingabevektoren werden vor dem SOM-Trainingsschritt zusätzlich dadurch transformiert, daß den Vorgänger- und Nachfolger-Vektoren ihre jeweils (berechnete) bedingte Wahrscheinlichkeit $E\{x_i^T \mid x_j\}$ zugeordnet wird. $E\{\ .\ \}$ gibt dabei den Erwartungswert bzw. den Mittelwert der Eingabedaten, x_i den Vorgänger- bzw. Nachfolger-Vektor und x_j den Schlüsselwort-Vektor an. Weitere Details des recht aufwendig konzipierten Lernprozesses dieses Verfahrens liefert [HPK95], S. 2ff. Die resultierende allgemeine Organisation der Selbst-Organisierenden Karte spiegelt sowohl syntaktische als auch semantische Kategorien der eingegebenen Wörter wider. So können alle Verben in der oberen Hälfte der trainierten Karte wiedergefunden werden, wohingegen Substantive in der rechten unteren Ecke der Karte zu liegen kommen. In der Mitte findet man Wörter, die mehreren Kategorien angehören: Adverben, Pronomen, Präpositionen, Konjunktionen, etc. (s. auch Abb. 1.11.).

7.4. WEBSOM

Am selben Neural Network Research Centre arbeitet eine Forschergruppe unter Leitung von T. Kohonen an einem intelligenten Informationssuchdienst, um große Ansammlungen von Volltextdokumenten zu verarbeiten, wie sie beispielhaft das World-Wide-Web (WWW) darstellt. In [HKLK96, KHLK96, KKLH96] stellt das Forscherteam die von ihnen entwickelte WEBSOM-Methode vor, die mehrere SOMs benutzt, um kodierte Dokumente auf der Karte zu verteilen, die dann eine generelle Sicht auf die Textansammlung gewährleistet. Die generelle Ansicht der Karte veranschaulicht Ähnlichkeitsbeziehungen zwischen Dokumenten des gewählten Kartenausschnitts. Die Karte wird hier mehr dazu benutzt, das zugrunde liegende Material in der Art des „Internet-Surfens“ oder „Browsens“ zu entdecken, als eine traditionelle Suche durchzuführen; d. h. es handelt sich bei der WEBSOM nicht um eine Erweiterung gewöhnlicher Internet-Suchmaschinen, sondern eher um eine geschickte Erweiterung und Erleichterung des Surfverhaltens der Benutzer bzw. um einen interaktiven Internet-Katalog („Directory“) im Stil des *Yahoo*-Katalogs (s. a. 2.). Hierbei verwendet die WEBSOM-Methode die selbstorganisierende semantische Karte aus Abschnitt 7.3. als Vorverarbeitungsstufe, um bei der Kodierung von Dokumenten semantische Wortähnlichkeiten auszunützen.
Als Eingabe zur WEBSOM-Methode dienen 4.600 Volltextdokumente mit zusammenfassend ca. 1,2 Millionen Wörtern der Usenet-Newsgroup „comp.ai.neural-nets“ (inzwischen wurde die Methode auf über eine Million Dokumente aus 80 Newsgroups erweitert; Stand: April 1998). Vorverarbeitend wur-

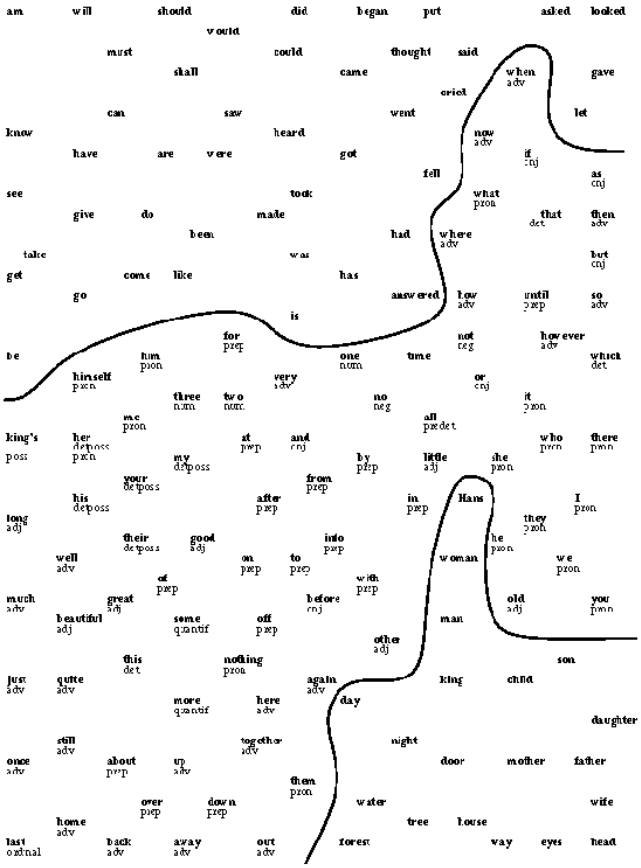


Abb. 1.11.: Die 150 häufigsten Wörter der Grimmschen Märchen und ihre statistische kontextuelle Repräsentation durch die zwei-dimensionale SOM (aus [HPK95]).

den nicht-textuelle Informationen, wie ASCII-Graphiken, sowie nur spärlich und sehr häufig auftretende Wörter verworfen. Solcherart vorverarbeitete Dokumente werden nun Wort für Wort auf die zuvor generierte selbstorganisierende semantische Karte (hier *word category map* genannt; s. auch Abschnitt 7.3.) abgebildet. Die selbstorganisierende semantische Karte faßt semantisch ähnliche Wörter innerhalb eines Neurons oder einer Neuronengruppe zu Wortkategorien zusammen. Die Treffer der Dokumente auf dieser Karte bilden die Histogramme oder *Dokumenten-Fingerabdrücke* (*fingerprints*) zur Eingabe in eine nachgeschaltete Standard-SOM. Ähnliche Newsgroup-Artikel tauchen nebeneinander auf der trainierten Karte auf. Zusätzlich wurde eine Browser-Schnittstelle entworfen, die es dem Benutzer erlaubt, durch die Karte zu „surfen“, um schließlich zu dem Dokument zu gelangen, welches ihn interessiert. Dabei kann auf eine Kartenregion geklickt werden, die nun den darunterliegenden Kartenausschnitt vergrößert widergibt (*Zooming*). Innerhalb der gezoomten Region kann wiederum ein Neuron ausgewählt werden, dessen darunterliegenden Dokumente, die auf das Neuron abgebildet wurden, dem Benutzer präsentiert werden. Der Benutzer kann schließlich einen Hyperlink anwählen, welches zu dem gewünschten Dokument führt und dessen Inhalt ausgibt. Der gesamte Ablauf dieses Browsens wird in Abb. 1.12. wiedergegeben. Eine Demoversion der WEBSOM ist unter der WWW-Adresse `http://www.hut.fi/websom/comp.ai.neural-nets-new/html/root.html` zu erreichen.

7.5. ET-Map

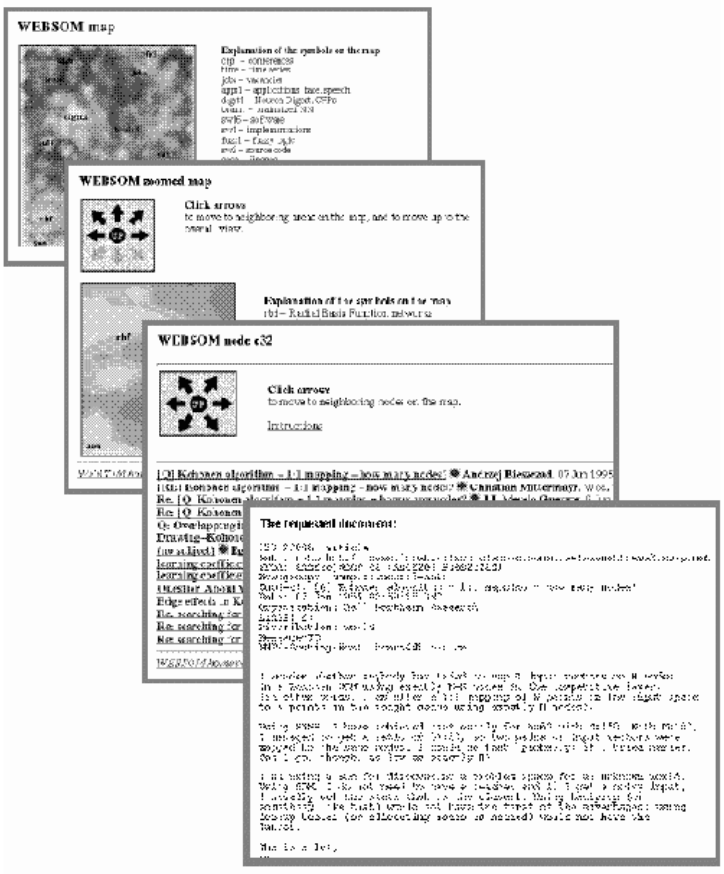


Abb. 1.12.: Die vier Betrachtungsebenen der WEBSOM Browsing-Schnittstelle (v.o.n.u.): die gesamte Karte, die Ausschnittskarte, die Dokumenten-Hyperlinks und die Dokumentenansicht (aus [HKLK96])

Ein ähnliches Konzept verfolgen die Forscher um H. Chen in dem Projekt „Concept-based Categorization and Search on Internet: A Machine Learning, Parallel Computing Approach“ des Artificial Intelligence Lab der University of Arizona in Tucson [CSO96]. Ihre *ET-Map* genannte Methode hat zum Ziel, eine skalierbare, automatische und konzept-basierte Internet-Homepage-Kategorisierung und -Suche durchzuführen. Sie soll dabei, wie die konventionelle Internet-Suchmaschine *Lycos*, die Informationsgranularität erhalten, zugleich jedoch einen mehrschichtigen Katalog in der Art von *Yahoo* anbieten. Ähnlich wie T. Kohonen et. al. im WEBSOM-Projekt analysierten sie mehr als *10.000* Homepages, die sich bei ihnen jedoch mit Unterhaltung (entertainment) beschäftigen, und produzierten eine mehrschichtige Selbst-Organisierende Karte, welche Regionen verschiedener Themenbereiche beinhaltet. Umfassendere Themenbereiche aus der Gesamteingabemenge beanspruchen dabei größere Kartenregionen, und konzeptionell verwandte Themenbereiche werden oftmals auf benachbarte Kartenabschnitte abgebildet. Kartenregionen, die über *100* URLs enthalten, generieren eine neue Karte, wohingegen Regionen, welche weniger als *100* URLs besitzen, eine geordnete Liste ihrer zugrundeliegenden Dokumente ausgeben. Die Methode erlaubt es, wie die WEBSOM, durch die hierarchisch angelegten Karten und ihren darauf abgebildeten Themenbereichen zu „surfen“. Eine Demo-Version von ET-Map steht unter der WWW-Adresse `http://ai.bpa.arizona.edu/ent/entertainment/entertain.html` zur Verfügung.

7.6. Textdokumenten-Klassifikation mit Hilfe wachsender und sich teilender Netze

M. Köhle und D. Merkl der Technischen Universität Wien [KöMe96] benutzen anstelle der SOM ein wachsendes und sich teilendes neuronales Netz [Fritz93], um Ähnlichkeiten in Volltextdokumenten oder anderen hochdimensionalen Eingaberäumen zu visualisieren. Im Gegensatz zu den Growing Cell Structures (GCS; s. 6.2.2.) wächst dieses neuronale Netz nicht nur, sondern es teilt sich je nach Beschaffenheit des augenblicklich zu lernenden Eingabe(unter-)raums, indem es Neuronen löscht, die nicht genügend den Eingaberaum repräsentieren, und produziert hierdurch isolierte Cluster, die verwandte Eingaben zusammenfassen.[8]
Die Experimente von Köhle und Merkl beschränken sich auf Volltextdokumente-Sammlungen, die „Software-Reuse"-Themen beinhalten. Die Volltextdokumente werden durch das Single-Term-Indexing-Verfahren kodiert, wobei sehr häufige und sehr seltene Wörter verworfen und durch Präfix- und Suffixverwerfung Wortstämme oder konzeptuelle Wortklassen gebildet werden: Jedes Wort (bzw. Wortkonzept) besitzt einen Vektoreintrag gleich *1*, falls das Wort im Text vorkommt und *0*, falls nicht (binäre Vektor-Repräsentation). Die Vektoren werden benutzt, um sowohl das wachsende und sich teilende neuronale Netz als auch die SOM zu trainieren. In einem ersten Experiment klassifizieren sie MS-DOS Kommandos: Die Vektoren zu den Kommandos werden durch die Wörter erzeugt, die im Text vorkommen, wenn das `help`-Kommando über das entsprechende Kommando aufgerufen wird. Es entstehen *37*-dimensionale Merkmalsvektoren von 36 MS-DOS Kommandos. Das Verfahren erzeugt nach abgeschlossenem Training 11 voneinander separierte Cluster, die jeweils verwandte Kommandos zusammenfassen. In einem zweiten Experiment benutzen sie als Eingabe die NIH-Klassenbibliothek, die eine Menge von Klassen der C++ Programmiersprache umfaßt. Jede Klasse dieser Bibliothek wird durch einen *489*-dimensionalen Vektor repräsentiert, der über die Schlüsselwörter der entsprechenden Manualeinträge konstruiert wird. Wieder werden ähnliche C++-Klassen in dasselbe Cluster abgebildet. Es entstehen 13 Cluster, die deutlich C++-Klassen voneinander trennen können, die sich u. a. mit „File I/O", „Basic data types", „Containerklassen" beschäftigen.
Als ein Vorteil des wachsenden und sich teilenden neuronalen Netzes gegenüber der SOM erwies sich bei beiden Experimenten, daß das benutzte Netz aufgrund dessen adaptiven Architektur die Clustergrenzen explizit repräsentieren kann, da Verbindungen zwischen Clustern während des Trainings gelöscht werden können. Doch genau dieser Vorteil bedingt auch den größten Nachteil dieses Verfahrens. Er führt dazu, daß Inter-Cluster-Beziehungen verloren gehen. Ein zweiter Nachteil ist die hohe Sensibilität gegenüber den vorzugebenden Lernparametern. Insofern behaupten die Autoren, daß die SOM im Vergleich zu dem von ihnen benutzten Netz ein extrem robustes Verfahren darstellt.

7.7. Textdokumenten-Klassifikation mit Hilfe einer erweiterten SOM

D. Merkl benutzt in einem weiteren Ansatz die gleiche zu klassifizierende Eingabemenge, nämlich die NIH-Klassenbibliothek aus Abschnitt 7.6. Jede Klasse dieser Bibliothek wird wiederum durch einen *489*-dimensionalen Vektor repräsentiert. Im Gegensatz zu seinen früheren Versuchen benutzt der Autor

8. Leider bezeichnen die Autoren das von ihnen verwendete Verfahren als „Growing Cell Structures (GCS)". Das von ihnen benutzte Verfahren ist jedoch ein (ebenfalls von Fritzke erfundenes) „wachsendes und sich teilendes Netz" [Fritz93] bzw. ein GCS mit anschließendem Pruning!

in [Merkl97] eine um eine „adaptive Koordinatentechnik“ erweiterte SOM. Die sogenannten adaptiven Koordinaten ahmen dabei die Bewegungen der Referenzvektoren der SOM-Neuronen nach. Die Neuronen sind dabei wie im SOM-Verfahren zunächst regelmäßig auf einem zweidimensionalen Neuronengitter angeordnet. Während des Trainings werden nun nicht nur die Referenzvoktoren, sondern auch alle Neuronen außer des Gewinners auf der Ausgabekarte in Richtung des Gewinners bewegt. Auf diese Art repräsentiert schließlich die Clusterung der Neuronen um den Gewinner die Eingabedichte des Eingabedatenraums nach jedem Trainingszyklus. Nach abgeschlossenem Training kann schließlich der durch die adaptive Koordinatentechnik erzeugte Ausgaberaum visualisiert werden. Cluster sind durch diese Technik besser voneinander unterscheidbar als durch das SOM-Verfahren, da die Neuronen nicht regelmäßig auf einem Neuronengitter zu liegen kommen, sondern selbst zu den Eingaben hin bewegt werden und also selbst Cluster bilden.
Die Ausgabegebiete, die z. B. durch die NIH-Klassen `Point` und `String` oder `String` und `OIOofd` bezeichnet werden, liegen auf der Ausgabekarte der Standard-SOM so dicht beeinander, daß man geneigt ist, die Klassen zu einer einzigen zusammenzufassen. Diese Gefahr besteht bei der erweiterten SOM nicht mehr, da die entsprechenden Klassen auf der Ausgabekarte genügend weit voneinander zu liegen kommen und für den Benutzer als eigene Cluster erkennbar sind (s. Abb. 1.13.).

Abb. 1.13.: Addaptive Koordinatenrepräsentation einer 10 x 10 SOM (aus [Merkl97], S. 108).

Merkl hebt hervor, daß sein Verfahren, im Gegensatz zu anderen SOM-Erweiterungen bzw. Alternativen, wie z. B. den wachsenden und sich teilenden neuronalen Netzen (s. 7.6.) und den GCS (s. 6.2.2.), keine zusätzlichen Lernparameter benötigt und damit robuster als diese ist.

7.8. Textdokumenten-Klassifikation mit Hilfe Hierarchischer Mermalskarten

In einem weiteren Experiment verwendet D. Merkl [Merkl97b, Merkl97c] die gleiche NIH-Klassenbibliothek, bzw. die aus deren Manualeinträge gewonnenen binären Dokumentenvektoren, als Eingabe zu Hierarchischen Merkmalskarten (*Hierarchical Feature Maps*; *HFM*), wie sie bereits 1990 von R. Miikkulainen vorgestellt wurden [Miikk90, Miikk92]. Die HFM bestehen aus mehreren Schichten zweidimensionaler Standard-SOMs: Für jedes Ausgabeneuron einer Karte wird eine neue zweidimensionale

SOM zur Hierarchie hinzugefügt. Das Training jeder dieser SOMs in den HFM folgt dem gewöhnlichen Trainingsalgorithmus für die Selbst-Organisierende Karte. Das vollständige Training der HFM erolgt über die „Top-Down Methode“, d. h. nach abgeschlossenem Training der obersten SOM werden alle SOMs der darunterliegenden Schicht trainiert, bis diese ebenfalls einen stabilen Trainingszustand erreicht haben. Dabei werden zum Training einer SOM der zweiten Schicht lediglich die Eingaben genutzt, die auf ein Neuron der ersten SOM abgebildet wurden. Dies ist gleichbedeutend mit einer Subclusterung von Clustern der ersten SOM. Zudem können bei einem Schritt in die nächste Stufe der HFM die Eingabevektoren um die Koeffizienten verkürzt werden, die gleiche Einträge besitzen. Dadurch wird die Trainingszeit der nachgeschalteten SOMs verringert. Erreichen alle SOMs der zweiten Ebene der HFM stabile Zustände, lassen sich SOMs der dritten Ebene auf analoge Art anlernen, u.s.w. Die HFM repräsentieren nach abgeschlossenem Training durch die Ausgabeschicht der SOM der ersten Ebene eine allgemeine und grobe Klassifizierung des Datensatzes und durch die Ausgabekarten aller nachgeschalteten SOMs in der HFM-Hierarchie eine weitergehende Klassifizierung von Klassen der ersten Ebene - also eine Subklassifizierung des gesamten Eingabe-Datenraumes.
Wie bereits erwähnt, verwendet Merkl als Eingabemenge eine Menge von binären Dokumentenvektoren, die aus Manualeinträgen einer NIH-Klassenbibliothek gewonnen wurden (s. a. 7.6. und 7.7.). Die *489*-dimensionalen Dokumentenvektoren werden dazu benutzt, eine *2x2* SOM der ersten HFM-Schicht zu trainieren. Die erste SOM zeigt nach abgeschlossenem Training die generelle Struktur der zugrundeliegenden NIH-Klassenbibliothek, nämlich eine Abbildung sämtlicher `File I/O` Klassen auf das erste Neuron, sämtliche `Datentypen` auf das zweite, `Datenstrukturen` auf das dritte und „`key-attributes access`“-`Datenstrukturen` auf das letzte Neuron. Das Training der HFM wird mit dem Training aller SOMs der zweiten HFM-Schicht fortgesetzt, und zwar mit jeweils denjenigen Vektoren, die auf ein Neuron der ersten SOM abgebildet wurden. Dies bedeutet, daß jeweils eine *2x2* SOM der zweiten HFM-Schicht mit Vektoren trainiert wird, die aus der `File I/O` Klassenmenge stammen, eine zweite mit Vektoren aus der `Datentypen`-Menge, u.s.w. Schließlich wird noch eine dritte HFM-Schicht, bestehend aus insgesamt *16 3x3* SOMs, mit entsprechenden Vektoren trainiert. Eine Visualisierung der Trainingsergebnisse für die beiden ersten HFM-Schichten ist in Abb. 1.14. wiedergegeben.

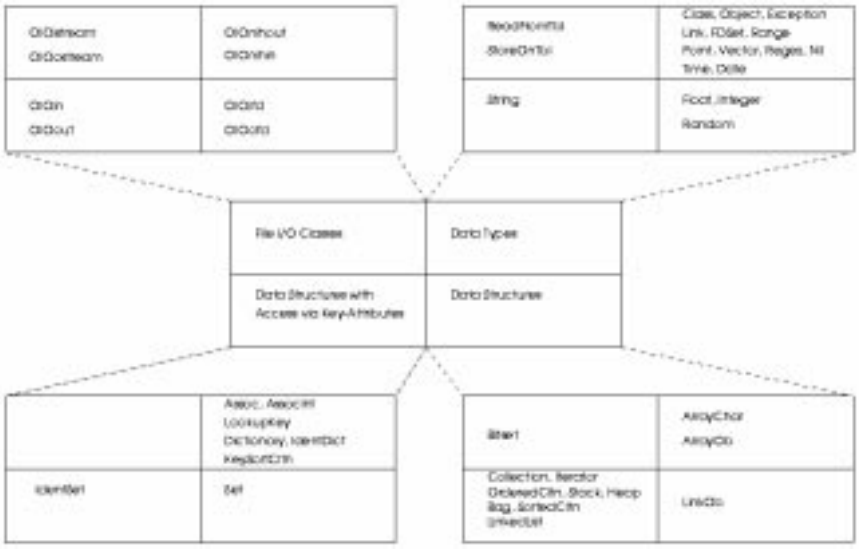


Abb. 1.14.: Die ersten beiden Ebenen der HFM, trainiert mit NIH-Klassenbibliothekeinträgen (aus [Merkl97b], S. 320).

Den umgekehrten Weg gehen A. Rauber und D. Merkl in [RaMe98]: Sie konstruierten HFM in Art der „Bottom-Up Methode“ anstelle der oben beschriebenen „Top-Down Methode“. Als Eingabemenge dienten *245* Dokumente des *CIA-Worldfactbooks* (`http://www.odci.gov/cia/publications/factbook/index.html`), das Informationen zu allen Ländern der Welt beinhaltet. *1056* Wörter, die in mehr als *15* und weniger als *220* Dokumenten vorkamen, wurden benutzt, um gewichtete Dokumentenvektoren zu erzeugen. Um eine digitale Bibliothek zu simulieren, teilten die Autoren den Eingabedatensatz zufällig in *5* Eingabemengen mit je *50* Dokumenten. Mit jeder Eingabemenge wurde je eine *7x7* SOM trainiert. Im nachfolgenden Bottom-Up Schritt wurde eine *7x7* SOM mit den Gewichtsvektoren der unter ihr liegenden SOMs trainiert. Diese SOM der ersten Ebene repräsentiert so die allgemeine Struktur des gesamten Datensatzes, so wie er durch die einzelnen, darunterliegenden SOMs „gesammelt“ wurde. Dieser Schritt kann iterativ zu weiteren Stufen fortgesetzt werden.

Abschließend soll die recht interessante Arbeit von J. Zavrel [Zavr95] genannt werden. Sie vergleicht die Cluster-Eigenschaften der GCS (s. 6.2.2.) mit anschließendem Pruning sowohl mit neuronalen Verfahren, wie der SOM und dem Gridnet ([BlMi92], s. a. 6.2.2.), als auch mit verschiedenen herkömmlichen statistischen Verfahren, wie Saltons Kosinus-Korrelationsmaß (s. Gleichung (1.6) bzw. (1.7)), dem Single- und Complete-Link (s. 5.1.1. bzw. 5.1.2.) sowie weiteren statistischen Methoden. Die Testreihen basieren auf den Cranfield- und Keen-Testdatensammlungen. Die beiden Sammlungen enthalten eine Menge von Dokumenten und Anfragen, sowie solche Dokumente, die von Experten als relevant zur entsprechenden Anfrage bewertet wurden[9]. J. Zavrel stellt zusammenfassend fest, daß die besten GCS nicht nur die SOM und das Gridnet übertreffen, sondern selbst viele klassisch-statistische Cluster-Verfahren schlagen. Gerade die Größe der Eingabemenge von „Real-World“-Daten, wie sie z. B. die Test-Datensammlungen darstellen, sind für neuronale Netze unproblematischer als für herkömmliche Verfahren. Die Güte der GCS-Cluster ist nach J. Zavrel ebenfalls als unproblematisch zu bezeichnen. Lediglich die Konvergenz der GCS kann sich bei einer allzu großen Eingabemenge als problematisch erweisen. Für weitere wertvolle Ergebnisse dieser Arbeit wird auf [Zavr95] verwiesen.

8. Zusammenfassung und Ausblick

Der vorliegende Interne Bericht bemühte sich um eine Bestandsaufnahme sowohl der herkömmlichen Internet-Suchtechniken als auch solcher Verfahren, die sich allgemein mit der Suche in größeren und anwachsenden Datenmengen beschäftigen. Behandelt wurden nicht nur die bekanntesten Internet-Suchmaschinen und -kataloge, wie AltaVista, Lycos und Yahoo, sondern auch Information Retrieval-Verfahren, wie das SMART-Retrievalsystem. Schließlich wurde noch näher auf die Möglichkeit eingegangen, ähnliche Eingabedokumente sinnvoll zusammenzufassen, d. h. zu „clustern“, um die anschließende Suche im Informationsraum zu verbessern. Die klassischen statistischen Verfahren, die solche Aufgaben bisher zu meistern vorgaben und im fünften Kapitel besprochen wurden, werden heute immer öfter

9. Siehe z. B. `http://local.dcs.gla.ac.uk/idom/ir_resources/test_collections/`

durch künstliche neuronale Netze ersetzt bzw. auf nicht-lineare Art erweitert. Folgerichtig wurden die geeignetsten unter diesen im nachfolgenden Kapitel einander gegenübergestellt. Das letzte Kapitel zeigte auf, in wie weit heute schon neuronale Netze dazu benutzt werden, um das Information Retrieval auf intelligente Art weiter zu entwickeln. Einige unter diesen Ansätzen, wie z. B. das von T. Kohonen geleitete WEBSOM-Projekt versucht dabei, klassische Internet-Kataloge, wie Yahoo, zu ersetzen.

Eine umfassende vergleichende Bewertung der gängigen neuronalen Verfahren in Form eines Kriterienkatalogs soll, zusammen mit der Beschreibung einer selbst zu entwickelnden Methode, in einem späteren Bericht erfolgen. Das zu entwickelnde Verfahren soll dabei auf künstlichen neuronalen Netzen basieren und Nachteile bereits existierender Verfahren beseitigen. Die wesentlichen Nachteile der hauptsächlichen Vetreter der Competitive Learning Verfahren wurde bereits im Anschluß an Abschnitt 6.2.5. Neuronales Gas, S. 41 angerissen. Die zu entwickelnde Methode, die die Suche im Internet durch Clusterung der HTML-Seiten erleichtern soll, muß dabei von folgenden Voraussetzungen ausgehen:

1. Die Termindizierungsmethode, die Vektoren aus HTML-Seiten erzeugt, generiert Vektoren, die tatsächlich den Inhalt von HTML-Seiten repräsentieren können. Verschiedene Verfahren wurden entwickelt. Einige recht einfache Verfahren wurden bereits in diesem Internen Bericht vorgestellt. Sie reichen von Buchstabentrigramm-Kodierungen, die 27^3-dimensionale Vektoren erzeugen (s. a. [HBR98], S. 291), bis zu elaborierten Verfahren, wie dem sogenannten Latent Semantic Indexing (s. S. 45).

2. Die Menge der weltweit verteilten HTML-Seiten bilden Cluster. D. h. es existieren tatsächlich Modi, Orte hoher Wahrscheinlichkeitsdichten im Raum, der durch die Menge aller HTML-Seiten aufgespannt wird.

3. Ein Clusterverfahren kann nur dann effizient arbeiten, wenn die Eingabemenge, die das Clusterverfahren zu bearbeiten hat, sinnvoll auf eine Untermenge des WWW beschränkt wird. Sinnvoll ist es, das Verfahren auf bereits inhaltlich verwandte Seiten, also - nach Voraussetzung 1 und 2 - auf Subcluster des WWWs, anzuwenden. Dies muß der Crawler im Gesamtsystem der intelligenten Internet-Suchmaschine leisten, indem er nur solche HTML-Seiten in seine lokale Datensammlung lädt, die inhaltlich verwandt sind bzw. nur solche, die nahe zu bereits vorhandenen Clusterprototypen zu liegen kommen.

4. Schließlich soll auch für das zu entwickelnde Suchverfahren van Rijsbergens Clusterhypothese gelten: „Closely associated documents tend to be relevant to the same request“ (s. S. 14).

A

[AKCM90] S. C. Ahalt, A. K. Krishnamurty, P. Chen, D. E. Melton: Competitive learning algorithms for vector quantization, *Neural Networks*, Vol. 3, pp. 277-291, 1990

B

[BaHo89] P. Baldi, K. Hornik: Neural Networks and Principal Component Analysis: Learning from Examples Without Local Minima, *Neural Networks*, Vol. 2, pp. 53-58, 1989

[BaVi95] H.-U. Bauer, T. Villmann: *Growing a hypercubical output space in a self-organizing feature map*, TR-95-030, International Computer Science Institute, Berkeley, 1995

[BGPW97] H.-U. Bauer, T. Geisel, K. Pawelzik, F. Wolf: Selbstorganisierende neuronale Karten, *Spektrum der Wissenschaft, Dossier: Kopf oder Computer*, 4/97, S. 74-83, 1997

[BlMi92] J. Blackmore, R. Miikkulainen: *Incremental grid growing: encoding high dimensional structure into a two-dimensional feature map*, TR AI92-192, University of Texas, Austin, TX, 1992

[BPS98] T. Bray, J. Paoli, C. M. Sperberg-McQueen: *Extensible Markup Language (XML) 1.0*, W3C Recommendation, W3C: The World Wide Web Consortium (MIT, INRIA, Keio), February 1998
`http://www.w3.org/TR/REC-xml`

[Brau95] R. W. Brause: *Neuronale Netze*, 2. Auflage, B. G. Teubner, Stuttgart, 1995

[Brau96] R. W. Brause: Sensor Encoding Using Lateral Inhibited Self-organized Cellular Neural Networks, *Neural Networks*, Vol. 9, No. 1, pp. 99-120, 1996

[BrPa98] S. Brin und L. Page: The Anatomy of a Large-Scale Hypertextual Web Search Engine, The *7th International World Wide Web Conference* (*WWW7*), Brisbane, Australia, April 1998
`http://google.stanford.edu/~backrub/google.html`

[BRS92] R. Botafogo, E. Rivlin, B. Shneiderman: Structural analysis of hypertext: Identifying hierarchies and useful metrics, *ACM Trans. Inf. Sys.*, 10, pp. 142-180, 1992

C

[Conn98] D. Connolly: *Naming and Addressing: URIs*, W3C (MIT, INRIA, Keio), March 1998
`http://www.w3.org/Addressing`

[Cover98] R. Cover: *The SGML/XML Web Page*, SIL Academic Computing, Dallas, April 1998
`http://www.sil.org/sgml`

[Cres95] F. Crestani: Implementation and evaluation of a Relevance Feedback device based on Neural Networks, in *From Natural to Artificial Neural Computation: International Workshop on Artificial Neural Networks*, Malaga, Spain, Lecture Notes in Computer Science, Vol. 930, pp. 597-604, Springer-Verlag, June 1995
`http://www.dcs.gla.ac.uk/scripts/personal/fabio/get_paper?iwann95`

[CSO96] H.Chen, C. Schuffels, R. Orwig: Internet Categorization and Search: A Machine Learning Approach, *Journal of Visual Communication and Image Representation, Special Issue on Digital Libraries*, Vol. 7, Nr. 1, pp. 88-102, 1996

D

[DeHe97] L. Dempsey, R. Heery: *A review of metadata: a survey of current resource description formats*, Work Package 3 of Telematics for Research project DESIRE (RE 1004), UKOLN, University of Bath, UK, May 1997
http://www.ukoln.ac.uk/metadata/desire/overview/rev_toc.htm

[DeSi88] D. DeSieno: Adding a conscience to competitive learning, *IEEE International Conference on Neural Networks*, Vol. 1, pp. 117-124, New York, 1988

[DeWe96] L. Dempsey, S. L. Weibel: *The Warwick Metadata Workshop: A Framework for the Deployment of Resource Description*, in D-Lib Magazine, July/August 1996
http://www.dlib.org/dlib/july96/07weibel.html

E

[Etzi96] O. Etzioni: The World Wide Web: Quagmire or Gold Mine, *Communications of the ACM*, November 1996/Vol. 39, No. 11, pp. 65-68, 1996
http://www.research.microsoft.com/research/datamine/ACM-contents.html

F

[Fair61] R. A. Fairthorne: *The mathematics of classification, Towards Information Retrieval*, Butterworth, London, 1-10, 1961

[FaOa95] C. Faloutsos, D. W. Oard: *A Survey of Information Retrieval and Filtering Methods*, Technical Report No. CS-TR-3514, Dept. of Computer Science, Univ. of Maryland, August 1995
ftp://ftp.cs.umd.edu/pub/papers/papers/3514/3514.ps.Z

[Ferb96] R. Ferber: *Information Retrieval*, GMD-IPSI, August 1996
http://www.darmstadt.gmd.de/~ferber/ir-bb/frame.html

[Ferb98] R. Ferber: *Data Mining und Information Retrieval*, Skriptum zur Vorlesung im WS 1997/98, TU Darmstadt, Februar 1998
http://www.darmstadt.gmd.de/~ferber/dm-ir/skript/frame.html

[Fiel+97] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee: *Hypertext Transfer Protocol - HTTP/1.1*, RFC 2068, Network Working Group, W3C (MIT, INRIA, Keio), 162 pp., January 1997
http://www.w3.org/Protocols/rfc2068/rfc2068

[FlRi83] B. Flury, H. Riedwyl: *Angewandte multivariate Statistik - computergestützte Analyse mehrdimensionaler Daten*, Fischer Verlag, 1983

[Forg65] E. Forgy: Cluster analysis of multivariate data: efficiency versus interpretability of classifications, *Biometrics*, 21, 768, 1965, (Abstract)

[Fox96] E. Fox: *Information Storage and Retrieval*, Lecture Notes of the Virginia Polytechnic Institute and State University, Fall 1996
http://ei.cs.vt.edu/~cs5604

[FPS96] U. M. Fayyad, G. Piatetski-Shapiro, P. Smyth: The KDD Process for Extracting Useful

Knowledge from Volumes of Data, *Communications of the ACM*, November 1996/Vol. 39, No. 11, pp. 27-34, 1996

[FPSU98] U. M. Fayyad, G. Piatetski-Shapiro, P. Smyth, R. Uthurusamy: *Advances in Knowledge Discovery and Data Mining*, AAAI Press/The MIT Press, 625 pp., 1998

[Fritz91] B. Fritzke: Unsupervised Clustering with Growing Cell Structures, In *Proc. of the IJCNN-91*, Seattle, 1991
`http://www.neuroinformatik.ruhr-uni-bochum.de/ini/PEOPLE/fritzke/papers/fritzke.clustering.ps.gz`

[Fritz92] B. Fritzke: *Wachsende Zellstrukturen - ein selbstorganisierendes neuronales Netzwerkmodell*, Dissertation der Technischen Fakultät der Universität Erlangen-Nürnberg, Erlangen 1992
`http://www.neuroinformatik.ruhr-uni-bochum.de/ini/PEOPLE/fritzke/papers/thesis.ps.gz`

[Fritz93] B. Fritzke: Vector Quantization with a Growing and Splitting Elastic Net, *Proc. of the International Conference on Artificial Neural Networks* (*ICANN'93*), Amsterdam, The Netherlands, September 13-16, 1993
`http://www.neuroinformatik.ruhr-uni-bochum.de/ini/PEOPLE/fritzke/papers/fritzke.icann93.ps.gz`

[Fritz94] B. Fritzke: Growing Cell Structures, A self organizing neural network for unsupervised and supervised learning, *Neural Networks* 7(9), pp. 1441-1460, 1994

[Fritz94b] B. Fritzke: Fast learning with incremental RBF networks, *Neural Processing Letters*, 1(1), pp. 2-5, 1994
`ftp://ftp.neuroinformatik.ruhr-uni-bochum.de/pub/manuscripts/articles/fritzke.incremental_rbf.ps.gz`

[Fritz95] B. Fritzke: A growing neural gas network learns topologies, in G. Tesauro, D. S. Touretzky and T. K. Leen (Eds.): *Advances in Neural Information Processing Systems 7*, pp. 625-632, MIT Press, Cambridge MA, 1995
`ftp://ftp.neuroinformatik.ruhr-uni-bochum.de/pub/manuscripts/articles/fritzke.nips94.ps.gz`

[Fritz96] B. Fritzke: Growing Self-organizing Networks - Why?, In Proc. of the *European Symposium on Artificial Neural Networks* (*ESANN'96*), Brussels, Belgium, pp. 61-72, 1996
`ftp://ftp.neuroinformatik.ruhr-uni-bochum.de/pub/outgoing/fritzke/papers/fritzke.esann96.ps.gz`

[Fritz97] B. Fritzke: *Some Competitive Learning Methods*, Report of the Systems Biophysics, Institute for Neural Computation, Ruhr-Universität Bochum, April 1997
`http://www.neuroinformatik.ruhr-uni-bochum.de/ini/VDM/research/gsn/JavaPaper`

[Fritz97b] B. Fritzke: The LBG-U method for vector quantization - an improvement over LBG inspired from neural networks, *Neural Processing Letters*, Vol. 5, No. 1, 1997
oder: Internal Report IR-INI 97-01, Institut für Neuroinformatik, Ruhr-Universität Bochum, January 1997
`ftp://ftp.neuroinformatik.ruhr-uni-bochum.de/pub/manuscripts/IRINI/irini97-01/irini97-01.ps.gz`

[FrRu67] H. P. Friedman, J. Rubin: On some invariant criteria for grouping data, *Journal of the American Statistical Association*, 62, 1159-1178, 1967

[Fuhr97] N. Fuhr: *Information Retrieval*, Skriptum zur Vorlesung an der Universität Dortmund, Mai 1997
http://amaunet.cs.uni-dortmund.de/ir/teaching/courses/ir

[Furn+88] G. W. Furnas, S. Deerwester, S. T. Dumais, T. K. Landauer, R. A. Harshman, L. A. Streeter, Lochbaum: Information Retrieval using a singular value decomposition model of latent semantic structure, *11th International Conference on R&D in Information Retrieval*, Grenoble, France, ACM Press, pp. 465-480, 1988

G

[Good58] I. J. Good: Speculations concerning information retrieval, Research Report PC-78, IBM Research Centre, Yorktown Heights, New York, 1958

[GoWi87] R. C. Gonzales, P. Wintz, *Digital Image Processing*, 2nd Edition, Addison-Wesley Publishing Compony, 1987

[GuBr97] R.V. Guha, T. Bray: *Meta Content Framework Using XML*, NOTE-MCF-XML, W3C (MIT, INRIA, Keio), June 1997
http://www.w3.org/TR/NOTE-MCF-XML

H

[HaCr94] M. Handley, J. Crowcroft: *The World Wide Web - Beneath the Surf*, UCL Press, 1994
http://www.cs.ucl.ac.uk/staff/jon/book/book.html

[HBR98] U. Heuser, A. Babanine, W. Rosenstiel: HTML Documents Classification using (Non-linear) Principal Component Analysis and Self-Organizing Maps, *Proc. of the 4th International Conference on Neural Networks and their Applications* (*Neurap'98*), Marseilles, France, pp. 291-295, March 1998
http://www-ti.informatik.uni-tuebingen.de/~heuser/papers/neurap98.ps.gz

[HeHe95] R. Henrion, G. Henrion: *Multivariate Datenanalyse, Methodik und Anwendung in der Chemie und verwandten Gebieten*, Springer-Verlag, 1995

[HKLK96] T. Honkela, S. Kaski, K. Lagus, T. Kohonen: *Newsgroup Exploration with WEBSOM Method and Browsing Interface*, Report A32, Helsinki Univ. of Technology, Laboratory of Computer and Information Science, 1/1996
http://websom.hut.fi/websom/doc/websom.ps.gz

[Honk97] T. Honkela: Self-Organizing Maps in Natural Language Processing, *Ph. D. Thesis, Helsinki University of Technology, Neural Networks Research Centre*, 12/97
http://www.cis.hut.fi/~tho/thesis

[HPK95] T. Honkela, V. Pulkki, T. Kohonen: Contextual Relations of Words in Grimm Tales, Analysed by Self-Organizing Map, *Proc. of the International Conference on Artificial Neural Networks (ICANN-95)*, Paris, France, pp. 3-7, 1995

[Hyöt96a] H. Hyötyniemi: Constructing Non-Orthogonal Feature Bases, *ICANN'96*, Washington DC, pp. 1759-1764, June 1996

[Hyöt96b] H. Hyötyniemi: Text Document Classification with Self-Organizing Maps, Symposium

on Artificial Networks (Finish Artificial Intelligence Conference), published in *STeP'96 - Genes, Nets and Symbols*, edited by J. Alander, T. Honkela, M. Jakobsson, pp. 64-72, August 1996

J

[JaDu88] A. K. Jain und R. C. Dubes: *Algorithms for Clustering Data*, Prentice Hall Advanced Reference Series, Englewood Cliffs, New Jersey, 1988

K

[Karh96] J. Karhunen: Neural Approaches to Independent Component Analysis and Source Seperation, *Proc. of the 4th European Symposium on Artificial Neural Networks* (*ESANN'96*), Bruges, Belgium, pp. 249-266, April 1996

[KHLK96] S. Kaski, T. Honkela, K. Lagus, T. Kohonen: Creating an Order in Digital Libraries with Self-Organizing Maps, in *Proc. of World Congress on Neural Networks (WCNN'96)*, Mahwah, NJ, pp. 814-817, 1996
`http://websom.hut.fi/websom/doc/wcnn96o.ps.gz`

[Kitt76] J. Kittler: A locally sensitive method for cluster analysis, *Pattern Recognition*, 8, 22-33, 1976

[KKL90] J. A. Kangas, T. Kohonen, T. Laaksonen: Variants of self-organizing maps, *IEEE Transactions on Neural Networks*, 1(1): 93-99, 1990

[KKLH96] T. Kohonen, S. Kaski, K. Lagus, T. Honkela: Very Large Two-Level SOM for the Browsing of Newsgroups, in *Proc. of ICANN'96*, Bochum, Germany, pp. 269-274, 1996

[Klein98] J. M. Kleinberg: Authoritative Sources in a Hyperlinked Environment, In *Proc. of the ACM-SIAM Symposium on Descrete Algorithms*, 1998 und IBM Research Report RJ 10076, May 1997
`http://simon.cs.cornell.edu/home/kleinber/auth.ps`

[Klut95] R. Klute: *Das World Wide Web - Multimedialer Hypertext im Internet*, Addison-Wesley, 1995
`http://www.nads.de:82/~klute/WWW-Buch/1`

[Koh82] T. Kohonen: Self-organized Formation of Topology Correct Feature Maps, *Biological Cybernetics*, 43:59-69, 1982

[Koh84] T. Kohonen: *Self-Organization and Associative Memory*, Springer-Verlag, 1984

[Koh90] T. Kohonen: The self-organizing map, *Proc. of the IEEE*, Vol. 78, pp. 1464-1480, 1990

[Koh95] T. Kohonen: *Self-Organizing Maps*, Springer-Verlag, 1995

[KöMe96] M. Köhle, D. Merkl: Visualizing Similarities in High Dimensional Input Spaces with a Growing and Splitting Neural Network, in *Proc. of ICANN'96*, Bochum, Germany, pp. 581-586, 1996

[Kwok89] K. L. Kwok: A Neural Network for Probabilistic Information Retrieval, *ACM/SIGIR*, pp. 21-30, 1989

[KWV95] J. Karhunen, L. Wang, R. Vigario: Nonlinear PCA Type Approaches for Source Sepe-

ration and Independent Component Analysis, *Proc. of the IEEE International Conference on Neural Networks* (*ICNN'95*), Perth, Australia, pp. 995-1000, November-December 1995

L

[LaSw98] O. Lassila, R. R. Swick: *Resource Description Framework (RDF), Model and Syntax*, W3C (MIT, INRIA, Keio), Working Draft, February 1998
`http://www.w3.org/TR/WD-rdf-syntax`

[LaWi67] G. N. Lance, W. T. Williams: A general theory of classificatory sorting strategies: II. Clustering systems, in *Computer Journal*, 10, 271-277

[LBG80] Y. Linde, A. Buzo, R. M. Gray: An algorithm for vector quantizer design, *IEEE Transactions on Communication*, COM-28:84-95, 1980

[Lloyd57] S. P. Lloyd: Least squares quantization in pcm, Technical Note, Bell Laboratories, 1957. Published in *IEEE Transactions on Information Theory*, 1982

[LoFr97] H. S. Loos, B. Fritzke: *DemoGNG v1.3*, Systems Biophysics at the Institute for Neural Computation, Ruhr-Universität Bochum, 1997
`http://www.neuroinformatik.ruhr-uni-bochum.de/ini/VDM/research/gsn/DemoGNG/GNG.html`
`http://www.neuroinformatik.ruhr-uni-bochum.de/ini/VDM/research/gsn/DemoGNG/tex/DemoGNG/DemoGNG.html`

[LSM91] X. Lin, D. Soergel, G. Marchionini: A Self-organizing Semantic Map for Information Retrieval, In *Proc. of the International ACM SIGIR Conference on Research and Development in Information Retrieval* (*SIGIR'91*) , Chicago, IL, pp. 262-269, 1991

[LuFu78] S. Y. Lu, K. S. Fu: A sentence-to-sentence clustering procedure for pattern analysis, *IEEE Transactions on Systems, Man and Cybernetics*, SMC 8, 381-389, 1978

[Luhn58] H. P. Luhn: The automatic creation of literature abstracts, in *IBM Journal of Research and Development*, 2, pp. 159-165, 1958

[Lyn97] C. Lynch: *Searching the Internet*, Scientific American, Special Report, March 1997
`http://www.sciam.com/0397issue/0397lynch.html`

M

[MaKu60] M. E. Maron, J. L. Kuhns: On relevance, probabilistic indexing and information retrieval, *Journal of the ACM*, 7, 216-244, 1960

[MaSc91] T. M. Martinetz, K. J. Schulten: A „neural-gas" network learns topologies, in T. Kohonen, K. Mäkisara, O. Simula und J. Kangas (Eds.): *Artificial Neural Networks*, pp. 397-402, North-Holland, Amsterdam, 1991

[McQu67] J. B. McQueen: Some methods of classification and analysis of multivariate observations, in *Proc. of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pp. 281-297, 1967

[Merkl97] D. Merkl: Exploration of Document Collections with Self-Organizing Maps: A Novel Approach to Similarity Representation, in *Proc. of the First European Symposium on Principles of Data Mining and Knowledge Discovery* (*PKDD'97*), Trondheim, Norway, pp. 101-111, June 1997

ftp://ftp.ifs.tuwien.ac.at/pub/publications/pkdd97.ps.Z

[Merkl97b] D. Merkl: Lessons Learned in Text Document Classification, In *Proc. of the Workshop on Self-Organizing Maps* (*WSOM‘97*), Espoo, Finland, pp. 316-321, June 4-6, 1997
ftp://ftp.ifs.tuwien.ac.at/pub/publications/wsom97ll.ps.Z

[Merkl97c] D. Merkl: Exploration of Text Collections with Hierarchical Feature Maps, In *20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (*SIGIR‘97*), Philadelphia, PA, July 27-31, 1997
ftp://ftp.ifs.tuwien.ac.at/pub/publications/sigir97.ps.Z

[Miikk90] R. Miikkulainen: Script recognition with hierarchical feature maps, *Connection Science*, 2, 1990

[Miikk92] R. Miikkulainen: Trace feature map: A model of episodic associative memory, *Biological Cybernetics*, 66, 1992

[Murt85] F. Murtagh: Multidimensional Clustering Algorithms, *Compstat Lectures 4: lectures in computational statistics*, Physica-Verlag, 1985

[Murt94] F. Murtagh: Neural Networks and Related ‘Massively Parallel‘ Methods for Statistics: A Short Overview, *International Statistical Review*, 62, 3, pp. 275-288, 1994

[Murt96] F. Murtagh: Neural Networks for Clustering, In P. Arabie, L. J. Hubert und G. De Soete (Eds.): *Clustering and Classification*, World Scientific Publishers, River Edge, NJ, pp. 235-268, 1996

O

[Oja94] E. Oja: Beyond PCA: Statistical Expansions by Nonlinear Neural Networks, *ICANN‘94*, Vol. 2, pp. 1049-1054, 1994

P

[PBMW98] L. Page, S. Brin, R. Motwani und T. Winograd: The PageRank Citation Ranking: Bringing Order to the Web, Submitted to SIGIR‘98, 1998
http://google.stanford.edu/~backrub/pageranksub.ps

[PTVF95] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery: *Numerical Recipes in C - The Art of Scientific Computing*, 2. ed, Cambridge University Press, 1995

Q

[Quin86] J. R. Quinlan: Induction of Decision Trees, *Machine Learning*, Vol. 1, pp 81-106, 1986

R

[RaMe98] A. Rauber, D. Merkl: Creating an Order in Distributed Digital Libraries by Integrating Independent Self-Organizing Maps, In *Proc. of the 8th International Conferencee on Artificial Neural Networks* (*ICANN‘98*), Skövde, Sweden, September 2-4, 1998
ftp://ftp.ifs.tuwien.ac.at/pub/publications/icann98-dl.ps.Z

[RBS94] E. Rivlin, R. Botafogo, B. Shneiderman: Navigating in hyperspace: designing a structure-based toolbox, *Communications of the ACM*, 37(2), pp. 87-96, 1994

[Reed93] R. Reed: Pruning Algorithms - A Survey, *IEEE Transactions on Neural Networks*, Vol.

4, No. 5, pp. 740-747, September 1993

[RiKo89] H. Ritter, T. Kohonen: Self-organizing semantic maps, *Biological Cybernetics*, 61(4):241-254, 1989

[RiKo90] H. Ritter, T. Kohonen: Learning 'semantotopic maps' from context, in *Proc. of IJCNN-90-WASH-DC, International Joint Conference on Neural Networks*, Volume I, pp. 23-26, Hillsdale, NJ, 1990

[RLJ97] D. Raggett, A. Le Hors, I. Jacobs: *HTML 4.0 Specification*, W3C Recommendation, W3C (MIT, INRIA, Keio), December 1997
`http://www.w3.org/TR/REC-html40`

[RMS91] H. J. Ritter, T. M. Martinetz, K. J. Schulten: *Neuronale Netze*, Addison-Wesley, München, 1991

[RoAl90] J. S. Rodrigues, L. B. Almeida: Improving the learning speed in topological maps of pattern, In *Proc. of INNC*, pp. 813-816, Paris, 1990

[Rocc71] J. J. Rocchio, Jr.: Relevance Feedback in Information Retrieval, in G. Salton (Hrsg.): *The SMART Retrieval System - Experiments in Automatic Document Processing*, Prentice-Hall, Englewood Cliffs, Kap. 14, 1971

S

[Salt68] G. Salton: *Automatic Information Organization and Retrieval*, McGraw-Hill, New York, 18, 1968

[SaMc83] G. Salton, M. J. McGill: *Introduction to Modern Information Retrieval*, McGraw-Hill, New York, 1983
deutsche Übersetzung: G. Salton and M. J. McGill: *Information Retrieval - Grundlegendes für Informationswissenschaftler*, McGraw-Hill, Hamburg, New York, 1987

[ScEr97] E. Schikuta, M. Erhart: The BANG-Clustering System: Grid-Based Data Analysis, in X. Liu, P. Cohen, M. Berthold (Eds.): *Advances in Intelligent Data Analysis* (*IDA-97*), LNCS 1280, pp. 513-524, 1997

[Schi96] E. Schikuta: *Grid-Clustering: An Efficient Hierarchical Clustering Method for Very Large Data Sets*, Technical Report No. TR-96201, Institute of Applied Computer Science and Information Systems, University of Vienna, 1996
`http://www.pri.univie.ac.at/~schiki/research/paper/tech-rep/TR-26201.ps`
auch in: *Proc. of the 13th International Conference on Pattern Recognition*, Vienna, IEEE Computer Society Press, October 1996

[Sedg92] R. Sedgewick, *Algorithms in C++*, Addison-Wesley Publishing Company, 1992

[Spar72] J. K. Sparck: A statistical interpretation of term specificity and its application in retrieval, *Journal of Documentation*, 28, 111-21, 1972

[Sull98] D. Sullivan (Editor): *Search Engine Watch*, Mecklermedia Corporation, Westport, CT, 1996-1998
`http://www.searchenginewatch.com`

T

[Theil98] W. Theilmann: Informationssuche im World Wide Web, in P. J. Kühn (Ed.): *5. Berichtskolloquium des Graduiertenkollegs Parallele und Verteilte Systeme*, Institut für Nachrichtenvermittlung und Datenverarbeitung (IND), Universität Stuttgart, pp. 91-100, 3. Juli 1998
`http://www.informatik.uni-stuttgart.de/ipvr/vs/personen/theilmann/gk98.ps.gz`

[Turau98] V. Turau: Web-Roboter, *Informatik Spektrum*, Springer-Verlag, 21:159-160, 1998

V

[vRij79] C. J. van Rijsbergen: *Information Retrieval*, 2nd Edition, Butterworths, London, 1979
`http://www.dcs.glasgow.ac.uk/Keith/Preface.html`

W

[WiHi91] R. Wilkinson, P. Hingston: Using the Cosine Measure in Neural Network for Document Retrieval, *ACM/SIGIR*, pp. 202-210, July 1991

[WoLa83] M. A. Wong, T. Lane: A *k*th nearest neighbor clustering procedure, *Journal of the Royal Statistical Society*, B45, 362-368, 1983

X

[XKO93] L. Xu, A. Krzyzak, E. Oja: Rival Penalized Competitive Learning for Clustering Analysis, RBF Net, and Curve Detection, *IEEE Transactions on Neural Networks*, Vol. 4, No. 4, July 1993

Z

[Zad65] L. A. Zadeh: Fuzzy sets, *Information and Control*, 8, 338-353, 1965

[Zahn71] C. T. Zahn: Graph-theoretical methods for detecting and describing Gestalt clusters, *IEEE Transactions on Computers*, C 20, 68-86, 1971

[Zavr95] J. Zavrel: *Neural Information Retrieval - An Experimental Study of Clustering and Browsing of Document Collections with Neural Networks*, Ph.D. Thesis submitted at the Faculty of Arts, Department of Comutional Linguistics, University of Amsterdam, The Netherlands, February 1995
`ftp://pi1093.kub.nl/pub/zavrel/zavrel.scriptie.ps.Z`

[Zell94] A. Zell, *Simulation Neuronaler Netze*, Addison-Wesley Publishing Group, 1. Aufl., 1994

[Zipf49] H. P. Zipf: *Human Behaviour and the Principle of Least Effort*, Addison-Wesley, Cambridge, Massachusetts, 1949