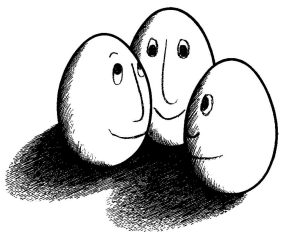
Diplomarbeit

Benutzergeleitetes Clustering von Musikdaten

Eduard Heinle



Diplomarbeit
am Fachbereich Informatik
der Universität Dortmund

4. April 2005

Betreuer:

Prof. Dr. Katharina Morik
Dipl.-Inform. Michael Wurst

Danksagung

Ich möchte mich zuerst bei Prof. Dr. Katharina Morik und Dipl.-Inform. Michael Wurst bedanken, die diese Arbeit betreut und dafür Sorge getragen haben, dass ich stets auf dem richtigen Wege blieb. Ihre Anregungen und Hinweise konnten mir oft helfen, gefährliche Klippen zu umschiffen und den Kurs zu behalten. Dipl.-Inform. Stefan Rüping danke ich für die „sanfte Einführung“ in die SVMs und die wertvollen Ideen bzgl. einiger, in dieser Arbeit vorgestellter Optimierungsverfahren. Dipl.-Inform. Ingo Mierswa verhalf mir den Nebel über der Experimentierumgebung YALE zu lichten und stand stets bei Rückfragen und Problemen zur Verfügung. Die in seiner Diplomarbeit extrahierten Merkmale für die Musikdaten boten einen hervorragenden Ausgangspunkt für diese Arbeit. Dipl.-Inform. Hanna Köpcke danke ich für die interessanten Ideen bzgl. der Möglichkeit der Verwendung der CSPs. Der größte Dank gebührt allerdings meiner Frau, Ilona, und meiner Tochter, Andrea, die unermüdlich für mein seelisches Wohl sorgten und mir immer wieder Kraft gaben.

Inhaltsverzeichnis

1 Einleitung

»Ich verstehe nichts von Musik. In meinem Fach ist das nicht nötig.«
Elvis Presley

Die Musik ist zu einem großen Teil unseres Lebens geworden. Überall werden wir von ihr begleitet oder auch bedrängt – durch Radio, Fernseher, Internet oder durch andere Medien. Dies beruht vor allem auf der Möglichkeit die Musikstücke analog oder digital zu speichern und beliebig oft wiederzugeben. Bis vor nicht allzu langer Zeit konnte man der Flut der Daten noch entkommen, indem man eine Auswahl traf und nur die einen interessierenden Musikstücke hörte.

Allerdings verändern sich die Vorlieben für die Musik im Laufe der Zeit. Schon bald besaß man eine beachtliche Sammlung an Musikstücken, und die Regale platzten vor der Menge an CDs oder anderen Tonträgern. Man verlor auch hier die Übersicht. Zur Hilfe kamen diverse Datenbanken, in die der Benutzer die eigene Musiksammlung eintragen konnte, um z.B. anschließend besser nach den Musikstücken suchen zu können. Schon da stellte sich heraus, dass der Benutzer nicht gewillt war alleine diesen immensen Aufwand zu betreiben.

Verschärft hat sich die Situation, als sowohl die Musikdaten auf den Rechnern in kompakteren Formaten als auf der CD gespeichert werden konnten als auch der Speicherplatz sehr billig wurde, und dadurch die Regale mit den CDs in den Keller verschwinden konnten, da man alle Musikstücke auf einer Festplatte aufbewahren und wiedergeben konnte. Zudem kamen noch weitere Möglichkeiten des Erwerbs an Musikstücken, z. B. durch Mitschnitt der Internetradios oder ähnlichem. Der Benutzer gab schon lange auf, die auf ihn einströmende Datenflut *kategorisieren* oder gar *katalogisieren* zu wollen und sammelte weiter, in der Hoffnung, dass sich unter der entstandenen Masse an Musikstücken solche befinden könnten, die er irgendwann einmal hören wollte. Daraus entstanden Sammlungen von tausenden von Musikstücken, die ganze Festplatten füllten, und keiner mehr wusste, was sie enthielten. Die einzige Möglichkeit für die Analyse war die Suche nach den Musiktiteln. Doch welcher Benutzer erinnert sich nach Jahren an den Titel eines Liedes, welches er unter tausenden nur einmal probeweise angehört hatte und dann aus damaligem Desinteresse verstauben ließ?

Nicht nur die gewöhnlichen Benutzer haben mit der immensen Datenflut zu kämpfen. Auch Menschen, die sich täglich mit Musik beschäftigen, wie z.B. DJs, haben Probleme eine Musikauswahl zu erstellen, die die Stimmung des Publikums, welches an dem Abend unterhalten werden soll, steigern und sie über die ganze Zeit hinweg erhalten können. Um dies zu erreichen, müssten sie auf *ähnliche* Musikstücke schnell zugreifen können.

Die in den ersten Abschnitten vorgestellten Szenarien sind zusammen mit vielen anderen eine Motivation, ein rechnergestütztes System aufzubauen, welches dem Benutzer wenigstens zum Teil die Arbeit abnimmt, in den Musikstücken eine für ihn sinnvolle Struktur zu finden.

In unserem Zusammenhang bedeutet eine „für den Benutzer sinnvolle Struktur“ eine Partitionierung der Menge der Musikstücke so, dass der Benutzer jeder erzeugten Partition eine für ihn sinnvolle Bedeutung zuschreiben kann. Die beste Struktur wäre erzeugt, wenn der Benutzer selbst alle Musikstücke manuell zuordnen würde. Da dies aber nicht möglich ist, suchen wir nach einer Zwischenlösung, die auf der einen Seite den Benutzer bei der Festlegung der zusammengehörenden Musikstücke einbezieht, auf der anderen Seite aber für die von dem Benutzer nicht betrachteten Musikstücke aufgrund eines internen *Ähnlichkeitsmaßes* so verschiedenen Partitionen zuordnet, dass sie den, von dem Benutzer vorgegebenen, so sehr wie möglich ähneln. Schon hier wird klar, dass man dadurch nie die vollkommene Lösung erreichen kann, da sich die Ähnlichkeitsvorstellungen des Benutzers von dem von der Maschine verwendeten Ähnlichkeitsmaß unterscheiden können. Die Vorgaben des Benutzers geben aber eine be-

stimmte Suchrichtung vor, die er immer wieder korrigieren kann, so dass das System und der Benutzer sich nach ein paar Iterationsschritten finden könnten.

Wir versuchen einen Mittelweg zwischen der vollkommenen Benutzerkontrolle ohne maschineller Hilfe und der maschinellen Selbständigkeit, *unsupervised learning* genannt, zu finden. Der Benutzer alleine ist mit dieser umfangreichen Aufgabe überfordert, und die Maschine alleine hat keine Ansatzpunkte für den von dem Benutzer erwünschten Aufbau der Partitionen. Dieses Dilemma wird in der Abbildung 1.1 grafisch dargestellt.

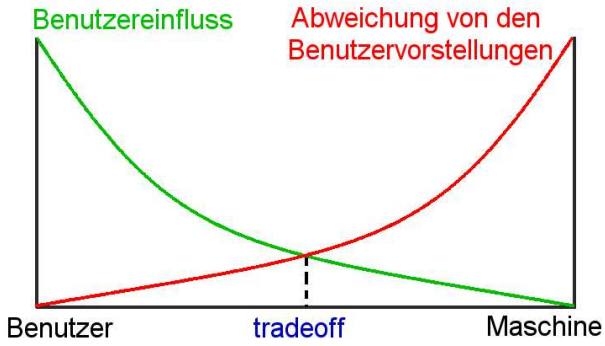


Abbildung 1.1: **Benutzer-Maschine – Dilemma.** Das Ziel ist es, den Benutzer soweit wie möglich zu entlasten, also seine Beteiligung an dem System zu minimieren, und gleichzeitig die maschinellen Entscheidungen so weit wie möglich den Vorstellungen des Benutzers anzupassen. Allerdings widersprechen diese beiden Ziele einander. Je mehr man den Benutzer entlastet, umso schwieriger wird es für die Maschine dessen Vorstellungen zu erraten. Man ist gezwungen einen Kompromiss (*tradeoff*) einzugehen.

Es sind einige Versuche unternommen worden, in den Musikdaten eine Struktur zu finden, bzw. ihnen eine Struktur aufzudrängen. Zentral war und ist die maximale Entlastung des Benutzers, bzw. Programmierers, bei maximaler Entsprechung zu dessen Partitionierungsvorstellungen. Im Vordergrund stand die geläufige Klassifikation nach den Musikgenren. In [69] hat man versucht die Musikdaten vollautomatisch durch Clustering (siehe unten) nach Genres zu partitionieren, wobei man sich auf die „*ideal theory of the information content in individual objects* (Kolmogorov-Komplexität)“, Informationsdistanz und eine „*universelle Ähnlichkeitsmetrik*“ berief. Einige Klassifikationsansätze wurden unternommen, um durch die Vorgaben der statistischen Eigenschaften der bekannten Genres die Musikdaten einzuordnen, z.B. [46, 65]. Es wurden Versuche unternommen, die in den Musikstücken enthaltenen Emotionen anhand von psychologischen Erkenntnissen herauszufiltern, um die Benutzerpräferenz zu erraten [41]. Auch wurden Musikstücke anhand bestimmter Kriterien, wie z.B. die Stimme des Sängers, darauf untersucht, inwiefern sie sich vollautomatisch gruppieren lassen [64].

Unser Ansatz basiert auf der klassischen *Clusteranalyse*, auch *Clustering* genannt. Clustering ist eine Einteilung der Daten in Gruppen aus ähnlichen Objekten [9]. Dieser Ansatz wird ausführlich im Kapitel 2 vorgestellt.

Clusteranalyse ist *unüberwachtes Lernen* [9]. Deswegen genügt dieser Ansatz alleine aufgrund der oberen Betrachtungen nicht, um das geforderte System aufzubauen. Der Benutzer muss eine Schnittstelle zu dem System erhalten, um die Struktur der Partitionen nach seinen Vorstellungen zu gestalten. Dazu wird der Ansatz *Constraints Clustering*, ein Teilbereich von *Semisupervised Clustering*, vorgestellt [7,54,70] und in das System eingebunden. Die Constraints sind die Bedingungen, die der Benutzer spezifizieren kann, um seine Vorstellungen von dem Aufbau der Cluster dem System mitzuteilen. Sie sind die Schnittstelle zwischen dem System und dem Benutzer und müssen sowohl für den Benutzer verständlich sein als auch ädaquat implementiert werden können, so dass die Semantik auch für die Maschine eindeutig ist.

Die Constraints müssen auf Musikdaten definiert werden. Dabei stellen sich zwei Fragen:

1. Wie *repräsentiert* man die Musikstücke sowohl *signifikant* als auch *effizient* in unserem System?

2. Was sind die *Parameter*, die wir manipulieren müssen, um die Constraints zu erfüllen?

Musikstücke werden durch digitalisierte Wellenform repräsentiert (Siehe Kap. 3). Es sind mehrere Megabytes an Daten. Wir suchen aber nach einer signifikanten Repräsentation der Musikstücke, deren Merkmalsraum so klein wie möglich ist. Wir verwenden die Merkmale, die von Ingo Mierswa in [46] extrahiert wurden (Siehe Kap. 3.1). Dadurch haben wir es mit wenigen Dimensionen zu tun, wodurch die Komplexität des Problems sinkt.

Die Merkmale repräsentieren die Musikstücke und dürfen deswegen nicht verändert werden. Allerdings kann man sie mit *Gewichten* multiplizieren, welche die Wichtigkeit der einzelnen Merkmale ausdrücken. Die Menge aller Gewichte bildet einen *Gewichtsvektor*. Durch Veränderungen der Werte dieses Vektors verändern sich die Abstände der Musikstücke zueinander (Siehe Kap. 6). Der Gewichtvektor ist der Parameter, an dem modifiziert werden muss, um die Constraints zu erfüllen. Es sind auch andere Parameter denkbar, wie z.B. eine Gewichtsmatrix (Kap. 6.4), aber in dieser Arbeit wollen wir nur den Gewichtsvektor untersuchen.

Wir versuchen so viele Constraints wie möglich mit unserem System zu erfüllen. Das bedeutet, dass wir es mit einem *Optimierungsproblem* zu tun haben. Dieses wird in dem Kapitel 6 definiert. Man kann es auch als Suchproblem formulieren. Man sucht in der Menge der möglichen Lösungen nach einem Optimum [73]. Allerdings können wir trotzdem mit der Hilfe von Heuristiken, Approximierungen oder Problemvereinfachungen schnell gute Lösungen konstruieren. Dazu wurde eine Reihe von Algorithmen entwickelt, von denen einige in dieser Arbeit vorgestellt und verglichen werden (Siehe Kap. 6.7).

Aufgrund der oberen Ausführungen müssen konkrete Anforderungen formuliert werden, die das System erfüllen muss. Einige dieser Anforderungen überschneiden sich mit den allgemeinen Anforderungen an einen Algorithmus oder an ein System, allerdings erhalten sie in unserem speziellen Zusammenhang eine neue Bedeutung und Daseinsberechtigung.

1. **Interaktivität.** Das System muss den Benutzer als einen festen Bestandteil einplanen. Er setzt gleichzeitig die Constraints und ist die letzte Instanz für deren Auswertung. Das bedeutet, dass ganz gleich wie gut unsere Evaluationsfunktion letztendlich sein mag, das letzte Wort hat immer der Benutzer.

    a) **Effizienz.** Ein System kann nur interaktiv sein, wenn es auch effizient ist. Kein Benutzer, von Forschern einmal abgesehen, wird stundenlang auf die Generierung der Ergebnisse warten können oder wollen. Entweder man erhält die Ergebnisse innerhalb weniger Minuten, oder der Benutzer ist nicht mehr an den Ergebnissen interessiert. Diese Anforderung ist sehr hart, da man von nun an auf effiziente Algorithmen angewiesen ist. Mit „effizient“ wollen wir Algorithmen bis zu einer Komplexität von $\mathcal{O}(n^2)$ bezeichnen, wobei n die Anzahl der zu verarbeitenden Musikstücke ist. Diese Schranke basiert auf einer einfachen Überlegung, dass wir es stets mit mehreren tausend Musikstücken zu tun haben, so dass wir mit einer höheren Komplexität als $\mathcal{O}(n^2)$, z.B. $\mathcal{O}(n^3)$, es mit mehr als einer Milliarde Schritten zu tun hätten. Dadurch wäre die Interaktivität ziemlich gefährdet. Wenn wir also feststellen müssen, dass wir keinen effizienten Algorithmus finden, müssen wir bewusst das Problem vereinfachen, um diesem Punkt gerecht zu werden. Wir dürfen aber nie den Fehler, den wir dabei machen, unterschlagen, sondern müssen ihn stets im Hinterkopf behalten und z. B. bei der Auswertung der Ergebnisse mit einbeziehen.

    b) **Verständlichkeit für den Benutzer.** Wie oben schon erwähnt wurde, hat Interaktivität keinen Sinn, wenn der Benutzer nicht genau weiß, welche Folgen die Veränderungen der

Constraints haben. Er kann sich nicht präzise ausdrücken und das führt von vorne hinein zu einem verfälschten Ergebnis.

c) **Verständlichkeit für die Maschine.** Genauso kritisch wie das Verständnis der Constraints für den Benutzer ist auch die Verständlichkeit der Constraints für die Maschine. Diese muss aufgrund der Vorgaben des Benuzers die interne Bedingungen des Systems so verändern, dass die Vorgaben so gut wie möglich erfüllt werden. Es muss idealerweise eine eins-zu-eins – Abbildung der Wünsche des Benutzers auf mathematisch auswertbare Constraints existieren.

d) **Effektivität.** Das System muss „gute Lösungen“ im Sinne des Benutzers erzeugen. Dies ist eine heikle Aufgabe, da man, wie oben schon erwähnt, nicht immer bestimmen kann, was der Benutzer sich wünscht, weil dieser sich dessen oft selbst nicht sicher ist. Wir versuchen hier so optimal wie es nur geht zu sein, allerdings muss das System stets bereit sein, die Einstellungen aufgrund neuer Benutzervorgaben zu ändern.

e) **Iterativität des Systems.** Das System versucht die Maschine an den Benutzer anzugleichen. Zudem ändert sich die Ansicht des Benutzers im Laufe der Zeit. Deswegen muss das System als eine Endlosschleife konzipiert werden, in der ständig die Eingaben korrigiert werden können.

2. **Minimale Benutzerbeteiligung.** Der Benutzer hat normalerweise nicht die Zeit, einen zu großen Aufwand an Arbeit zu betreiben, um dem Programm die Richtung vorzugeben. Er sollte also nicht mehr Eingaben als unbedingt nötig machen müssen.

3. **Integration in YALE.** Eine Integration in eine *Experimentierumgebung*, in der die Wahl der Testdaten und der Evaluation der Ergebnisse vereinfacht und die Experimentierschritte und die Ergebnisse in einer übersichtlichen grafischen Form dargestellt werden können, ist ein weiterer Schritt in Richtung Interaktivität. Der Entwickler braucht sich nicht um viele technischen Details zu kümmern, er kann sich direkt auf seine Aufgabe konzentrieren. Unser System wurde in YALE – Yet Another Learning Environment[1] eingebunden. Die Integration in diese Umgebung wird in dem Kapitel 6.8 vorgestellt.

Nachdem die Kriterien konzipiert wurden, werden die in dieser Arbeit eingesetzten Optimierungsverfahren vorgestellt.

Unser Basisverfahren ist ein einfaches Score-basiertes Verfahren (Kap. 6.7.1). Durch seine Einfachheit ist es sehr schnell. Allerdings ist es wiederum zu einfach, um garantiert gute Evaluationsergebnisse zu erzeugen. Als Vergleichsmaß zu den anderen Ansätzen eignen sich die Ergebnisse dieses Verfahrens aber ausgezeichnet.

Man könnte versuchen, auf dem Wege der Analysis ein Optimum zu berechnen. Im Kapitel 6.7.2 wird ein solches Verfahren entwickelt. Wie wir sehen werden, unterscheidet sich dieser Ansatz in seinem Kern allerdings kaum von dem Score-basierten Verfahren.

Eine Klasse von Verfahren, welche durch eine besonders trickreiche Datenrepräsentation gute Heuristiken erlaubt, sind die *Constraint Satisfaction Problems (CSPs)* [53]. Auch bei diesen Klassen von Verfahren haben wir die Möglichkeit, Constraints zu definieren und einzusetzen. Wie wir sehen werden (Kap. 6.7.3), ist dieser Ansatz nicht der richtige Weg für die Lösung unseres Problems, obwohl man anhand des Namens zunächst das Gegenteil annehmen könnte.

Bei dem nächsten Verfahren werden die Constraints auf die Eingabe für die *Support Vector Machine, SVM* [68] abgebildet, und diese berechnet daraufhin das Optimum. Die SVMs werden im Kapitel 6.7.4 vorgestellt.

Das Newton-Verfahren ist ein wohlbekanntes Optimierungsverfahren mit schnellem Konvergenzverhalten. Dieses wird in dem Distance Metric-Lernverfahren (Kap. 6.7.5) angewandt, nachdem unser Problem auf eine differenzierbare Funktion abgebildet wurde.

[1] Die neueste Version von YALE ist unter: `http://sf.net/projects/yale` zu finden. Weitere Informationen und die Dokumentation findet man in [45] und in [19].

Eine dem biologischen Vorbild folgende, darvinistisch geprägte Bewegung der Programmierung hat sich in den letzten Jahren etabliert und bietet für viele schwierige Probleme elegante Lösungen. Die Rede ist von den *Evolutionären Algorithmen, (EA)* und deren unterschiedlichen Teilbereichen, wie z.B. der für Clustering am meisten verwendete [35] Bereich *Genetisches Programmieren (GA)* [15, 34]. Die Grundidee ist, aus einer Population an Lösungen durch Mutation und Kreuzung eine neue aufzubauen, wobei der darvinistischen Devise „*survival of the fittest*“ folge geleistet wird. Das bedeutet, dass schlechte Lösungen durch einen Auswahlprozess aus der Population ausgesondert werden (Siehe Kap. 6.7.6). Leider benötigen die Evolutionäre Algorithmen unter Umständen viel Zeit, um gute, bzw. optimale Lösungen zu finden.

Auf der Basis des Vergleichs dieser Verfahren werden wir die Möglichkeiten und die Grenzen des Constrained Clustering (Kap. 2.6.3) testen und diskutieren.

1.1 Übersicht über die Kapitel

Die Arbeit gliedert sich in die folgenden Kapiteln:

Im Kapitel 2 wird eine kleine, problemorientierte Einführung in die Clusteranalyse gegeben, zusammen mit einer Übersicht über die in der Literatur verwendeten Clusteringverfahren. Einige, für diese Arbeit oder möglicherweise auch für die nachfolgenden, relevante Verfahren werden ausführlich mit ihren Vor- und Nachteilen vorgestellt (Kap. 2.6). Im Mittelpunkt der Diskussion steht die Frage: „Wie bewerte ich die Güte der generierten Cluster?“ Dazu werden ein paar philosophische Überlegungen gemacht (Kap. 2.5). Anschließend gehen wir von der allgemeinen Betrachtung der Clusteranalyse zu der spezifischen Betrachtung der Besonderheiten von Musikdaten über (Kap. 2.7).

Das nächste Kapitel 3 widmet sich aussagekräftigen Merkmalen, die aus den Musikstücken extrahiert werden. Dazu werden die in der Arbeit von Ingo Mierswa [46] extrahierte Merkmale aufgelistet (Kap. 3.1). Da die Musikdaten auch Metainformationen, wie das *Erscheinungsjahr* oder das *Album* enthalten können, wird in dem Kapitel 3.2 über die Möglichkeit und die Problematik diskutiert, die Merkmalsmenge mit ihnen zu erweitern.

Die *Constraints*, die die endgültige Partitionierung der Musikstücke beeinflussen, werden in dem Kapitel 4 definiert. Es werden effiziente Methoden zur Auswertung der Constraints entwickelt und deren Effizienz bewiesen (Kap. 4.4).

Die allgemeine Vorgehensweise des Systems wird in dem Kapitel 5 aufgezeigt. Hier wird vor allem die Benutzer-Maschine – Interaktion geklärt, in die das Optimierungssubsystem integriert wird.

Das Kapitel 6 definiert und erläutert die Optimierung der Erfüllung der Constraints. Die allgemeine Methodik der Optimierung (Kap. 6.1) und die Abbildung der Constraints auf ein Optimierungsproblem (Kap. 6.2 - 6.5) wird aufgezeigt, und die dabei entstandene Probleme beleuchtet (Kap. 6.6). Die in unserem System verwendeten, oben aufgeführten, Optimierungsverfahren werden daraufhin vorgestellt (Kap. 6.7) und deren Einbindung in die Experimentierumgebung YALE erläutert (Kap. 6.8).

Der Frage nach der Güte der vorgestellten Verfahren wird in dem Kapitel 7, in dem einige Experimente gestartet und die Verfahren untereinander verglichen werden, nachgegangen. Dabei steht vor allen Dingen im Vordergrund, wie gut die Verfahren den Wünschen des Benutzers entsprechen. Zunächst wird die Problematik der Evaluation im Kapitel 7.1 diskutiert. Die Evaluation selbst ist in zwei Phasen unterteilt. In der ersten Phase (Kap. 7.2.1) werden die Verfahren aufgrund der Ergebnisse der Evaluationsfunktion untereinander verglichen. In der zweiten Phase (Kap. 7.2.2) wird überprüft, inwieweit das System Benutzerwünschen gerecht werden kann. Fünf Benutzer sollten mit jedem der Verfahren versuchen in einer begrenzten Anzahl von Iterationen eine für sie ideale Gruppierung der Musikstücke zu erzeugen. Die Reaktion des Benutzers auf das System ist in dem Kapitel 7.2.3 festgehalten.

Das Kapitel 8 fasst die Ergebnisse noch einmal zusammen, überprüft, inwiefern die in der Einleitung gefassten Ziele dieser Arbeit erfüllt worden sind, und zeigt auf, in welchen Bereichen für die nachfolgenden Arbeiten Ansaztpunkte für weitere Forschung existieren.

2 Clusteranalyse

Das menschliche Individuum versucht in seine Umgebung stets eine Ordnung zu bringen, um sie besser verstehen und überblicken zu können. Geordnete Elemente können von Menschen besser aufgefasst, verarbeitet und gelernt werden [44]. Der Mensch bildet dafür vorzugsweise Klassen, die meist durch einen Oberbegriff beschrieben werden, nach denen er die Objekte aus seiner Umwelt, die Daten, einordnet. Er nimmt also eine Komprimierung der Daten vor [30].

Wenn man aber nicht gewillt ist Klassen zu bilden, sei es aus der Unüberschaubarkeit und Menge der Daten oder aus dem Wunsch, „verborgene Strukturen“ in den Daten auszumachen, dann wäre es wünschenswert, die Bildung von Klassen zu automatisieren.

Aus Gründen, die aus dem Kapitel 2.1 ersichtlich werden, wollen wir von dem Begriff „*Klassifikation*“ Abstand nehmen und statt dessen „*Partitionierung*“ benutzen, bis wir einen besseren Begriff definiert haben.

Wie bringt man eine Maschine dazu, automatisch sinnvolle Partitionen zu bilden? Und was bedeutet „*sinnvoll*“ in diesem Zusammenhang? Wenn man mehrere Menschen einen Sack Reis partitionieren lassen würde, dann würde jeder eigene Partitionierungen bilden und die Reiskörner danach einordnen. Einer könnte die Größe als ein Partitionierungskriterium nehmen, ein anderer die Qualität der einzelnen Reiskörner. Wie könnte man aber diese beiden Partitionierungen untereinander vergleichen? Welche ist besser? Offensichtlich kann man sie nicht vergleichen, und doch hat jede von ihnen ihre Berechtigung. Wir können höchstens überprüfen, ob die Partitionierung nach den jeweiligen Kriterien korrekt durchgeführt wurde.

Wir benötigen also für die Partitionierung ein Kriterium, an dem die Ähnlichkeit von Objekten gemessen wird, das *Ähnlichkeitsmaß* (engl. *similarity measure*).

Dieses Ähnlichkeitsmaß ist unser einziger Anhaltspunkt für die Richtung der Partitionierung, da wir keine weitere Information über die Beschaffenheit der Partitionen haben. Allerdings sollte man, vor allem in Anbetracht unserer Daten, den Musikstücken jede Information über die gewünschte Struktur in die Partitionierung aufnehmen. Diese Idee wird in den folgenden Abschnitten dieses Kapitels näher erläutert.

Die Objekte können durch eine oder mehrere Variablen beschrieben werden. Die Wertemengen der Variablen bestimmen die Variablenarten. Folgende Variablenarten können vorkommen:

- **Kontinuierliche Variablen** sind Variablen, deren Wertemenge überabzählbar unendlich ist.
- **Diskrete Variablen** sind Variablen, die eine endliche, oder im schlimmsten Fall, abzählbar unendliche Wertemenge haben.
- **Binäre Variablen** sind Variablen, deren Wertemenge nur aus zwei Werten besteht.

Bleiben wir noch bei den möglichen Kriterien für eine Partitionierung. Wir können das Kriterium „Größe“ haben, aber auch das Kriterium „Farbe“. Was sind die Gemeinsamkeiten dieser Kriterien? Bei beiden kann man feststellen, ob die Objekte diese Eigenschaft besitzen oder nicht, also, ob sie von gleicher Farbe oder von gleicher Größe sind. Für die Größe können wir aber zusätzlich sagen, dass ein Objekt größer als das andere ist, was bei der Farbe unmöglich ist. Wir können sogar bestimmen, um wie viel das eine Objekt größer ist als das andere.

DEFINITION 1: (**Die Skalen [4,30]**)
*Eine **Skala** ist eine Messvorschrift für gleichartige Objekte.*
*Wenn zwei Variablen mit einander nur auf Gleichheit verglichen werden können, so spricht man von einer **Nominalskala**.*
*Wenn unter ihnen eine Ordnung existiert, es also zusätzlich bestimmt werden kann, ob eine Variable größer oder kleiner ist als eine andere, dann spricht man von einer **Ordinalskala**.*
*Wenn man zusätzlich noch bestimmen kann, um wieviel eine Variable größer oder kleiner ist als eine andere, dann spricht man von einer **Intervallskala**.*
*Als eine weitere Skala existiert noch die **Rationalskala**. Diese hat einen bedeutungsvollen Nullpunkt. Man kann das Verhältnis der Variablen zueinander berechnen und sagen, die Variable x_A ist um $\frac{x_A}{x_B}$ besser als die Variable x_B.*
Die Skalen sind hierarchisch geordnet.
*Die Variablen auf der Nominal- oder der Ordinalskala nennt man **kategorische** oder **qualitative** Variablen.*
*Die Variablen auf den anderen Skalen nennt man **quantitative** Variablen.*

Wenn alle Variablen nur auf einer Skala sind, können sie in gleicher Weise behandelt werden. Wenn sie aber mehrere Skalen enthalten, dann müssen sie auf eine Skala normiert werden. Die Transformationen zwischen den Skalen werden in dem Buch von M.R. Anderberg [4] ausführlich erläutert.

Unser Ziel ist es, eine automatische Einordnung in Gruppen vorzunehmen. Diese wird *Clustering* genannt und ist folgendermaßen definiert:

DEFINITION 2: (**Clustering**)
*Gegeben eine Datenmenge $X = x_1, \ldots, x_n$, in welcher $x_i,\ i \in \{1, \ldots, n\}$, die **Datenpunkte** (**Objekte**, **Instanzen**, **Fälle**, **Muster**, **Tupeln** und **Transaktionen** [9]) sind. Jedes der Datenpunkte x_i besteht aus einer Menge von numerischen oder kathegorischen **Merkmalen** (auch **Attribut** (engl. **feature**), **Variable**, **Dimension**, und **Feld** genannt) $x_i = (x_{i_1}, \ldots, x_{i_d}) \in A$ aus dem Merkmalsraum A.*
*Der Prozess der automatischen Partitionierung von X in die Menge $\{C_1, \ldots, C_k\}$ mit $C_i \subseteq X,\ i \in \{1, \ldots, k\}$ und $\bigcup_{i=1}^{k} C_i = X$ wird **Clustering** genannt. C_i sind die einzelnen **Cluster** [52].*
*Clustering, auch **Clusteranalyse** genannt, ist ein Instrumentarium zum Erkennen von **Strukturen** in einer Menge von Objekten [30].*
*Objekte, die zu einem Cluster gehören, sollen sich ähnlich sein (**Homogenität**), und die verschiedenen Cluster sollen möglichst unterschiedliche Objekte erhalten (**Heterogenität**) [30].*

Die Clusteranalyse wird in vielen verschiedenen Bereichen der Mustererkennung verwendet, Maschinelles Lernen, Data Mining, Dokumentenretreaval, Bildsegmentierung, Musikdatenstrukturierung, etc. [35]. Immer, wenn wenig Informationen für die Struktur der Daten vorgegeben ist (z.B. kein statistisches Modell o.ä.) und man die Struktur aus den Daten ermitteln muss, kann dieser Ansatz zur Anwendung kommen [35].
Der Clustering-Ansatz besteht aus folgenden Schritten [35]:

1. **Musterrepräsentation.** Die gewünschte Anzahl der Cluster, die Anzahl der verwendeten Datenpunkte und die Anzahl, Typ und Skala der verwendeten Merkmale werden in diesem Schritt bestimmt. Vorher müssen die Merkmale evtl. noch durch Merkmalsauswahl oder Merkmalsextraktion ermittelt werden. In Kap. 3 wird diese Vorgehensweise näher erläutert.

2. **Definition der Musterähnlichkeit.** Hier wird das oben erwähnte Ähnlichkeitsmaß abhängig von den Daten definiert. In Kapitel 2.2 werden die gängigen Distanzmaße vorgestellt und erläutert.

3. **Clustering oder Gruppierung.** Die Daten werden mit Hilfe eines ausgewählten Clusteringalgorithmusses gruppiert. Eine Übersicht über die Clusteringalgorithmen bietet das Kapitel 2.6.

4. **Datenabstraktion.** (falls nötig) Extraktion einer vereinfachten Darstellung der Cluster aus dem Ergebnis des Clusterings, z.B. betrachtet man nur das Clusterzentrum statt aller Datenpunkte im Cluster. Das hat den Vorteil der komprimierten Darstellung der Cluster. Oft wird die Datenabstraktion schon implizit in den Clusteringverfahren angewandt (Kap. 2.6).

2.1 Clustering und Klassifikation

Wir unterscheiden Clustering von der Klassifikation. In dem Bereich der Statistik werden die beiden Begriffe nicht unterschieden. Im Bereich des *Maschinellen Lernens* (ML) und *Information Retrieval* (IR) haben sie eine grundverschiedene Bedeutung [67]. Da viele Verfahren, die in diesen beiden Gebieten angewandt werden, ursprünglich aus dem Bereich der Statistik kommen, entstehen noch heute aufgrund des unterschiedlichen Gebrauchs der Begriffe, Missverständnisse, die wir in dieser Arbeit von vornherein vermeiden wollen. In den folgenden Absätzen werden die Bedeutungen dieser beiden Begriffe in dem Bereich Maschinelles Lernen genauer voneinander getrennt, ohne den Anspruch zu haben, eine formale Definition aufzustellen.

Bei der *Klassifikation* werden neue Objekte in schon vorhandene, fest umrissene, Klassen eingeordnet. Man könnte als Synonym für die Klassifikation „*Identifikation*“ verwenden [67].

Ein Beispiel für die Klassifikation wäre alle Lebewesen der Erde in die Klassen „Säugetier“, „Fisch“, „Kaltblüter“, etc. einzuordnen, sie mit diesen Klassen zu *identifizieren*.

Beim *Clustering* wird in den Daten die verborgene Struktur gesucht, die durch das Ähnlichkeitsmaß vorgegeben ist. Man versucht zusammenhängende Bereiche auszumachen und sie voneinander abzugrenzen. Nicht umsonst wurde in den älteren Büchern das Wort „*Klumpenbildung*“ für Clustering verwendet [30]. Als Synonym für das Clustering könnte man „*Gruppierung*“ verwenden [67]. Die Cluster werden ausschließlich anhand der Daten ermittelt (*data driven*) [35].

Ein Beispiel für Clustering wäre es, die Menge der Schüler, die sich auf dem Schulhof versammelt haben, zu *gruppieren*, also die in der Menge vorhandene Gruppen zu erkennen und zu umreißen. Dabei kann als Ähnlichkeitsmaß die Entfernung zwischen den Schülern angenommen werden.

2.2 Ähnlichkeitsmaße

Die Notwendigkeit der Definition des Ähnlichkeitsmaßes (engl. *similarity measure*) wurde oben in diesem Kapitel schon ausreichend motiviert. In diesem Abschnitt werden die gebräuchlichsten Ähnlichkeitsmaße vorgestellt.

Es ist üblich die Unähnlichkeit zweier Datenpunkte (engl. *dissimilarity*) als Grundlage fürs Clustering zu nehmen [35]. Sie wird mit Hilfe eines *Distanzmaßes* berechnet. Es haben sich je nach der verwendeten Skala verschiedene Distanzmaße etabliert.

Die *Minkowski-Metrik* bietet gleich mehrere Distanzmaße:

$$d_p(\mathbf{x}_i, \mathbf{x}_j) = \left(\sum_{k=1}^{d} |x_{i,k} - x_{j,k}|^p \right)^{\frac{1}{p}} = ||\mathbf{x}_i - \mathbf{x}_j||_p. \tag{2.1}$$

Der Parameter p bestimmt das endgültige Distanzmaß. Für $p = 1$ erhalten wir die *Hamming-Distanz*, für $p = 2$ erhalten wir die gebräuchlichste *Euklidische Distanz*. Wenn $p \to \infty$, dann erhalten wir die *Maximum-Distanz*. Die Abbildung 2.1 veranschaulicht diese drei Distanzmaße.

Das Euklidische Distanzmaß ist das gebräuchlichste und auch das im zwei- und dreidimensionalen Fall für den Menschen das intuitivste. Es kann besonders gut angewandt werden, wenn die Datenmenge „kompakte“, bzw. „isolierte“ Cluster enthält [35].

Bei diesen Distanzen geht man davon aus, dass alle Punkte mit der gleichen Distanz zu einem anderen Punkt auf einem gedachten Kreis um diesen Punkt liegen. Implizit bedeutet das, dass man nur kreisförmige (*isotrope*) Cluster konstruieren kann, wenn man zur Repräsentation der Cluster Clusterzentren

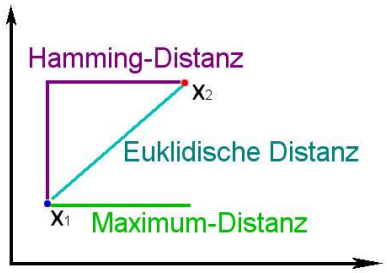


Abbildung 2.1: **Die Distanzen der *Minkowski-Metrik* im zweidimensionalen Fall.** Je nach der Belegung des Parameters p für die Gleichung 2.1 wird ein anderes Distanzmaß zwischen den Datenpunkten x_1 und x_2 definiert.

verwendet, also ein *zentroid-basierte Verfahren* wählt. Um dem zu entgehen und auch ellipsenförmige (*non-isotrope*) Cluster zu erlauben, benutzt man die quadrierte *Mahalanobis-Distanz*:

$$d_M(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i - \mathbf{x}_j)A^{-1}(\mathbf{x}_i - \mathbf{x}_j)^T, \tag{2.2}$$

wobei $\mathbf{x}_i, \mathbf{x}_j$ als Spaltenvektoren aufgefasst werden. A ist die Kovarianzmatrix der Muster. Mit ihr wird die Stauchung des Kreises zu der Ellipse beschrieben. Die Ermittlung dieser Matrix ist nicht immer einfach.

Die bis jetzt vorgestellten Maße eignen sich allerdings nur für quantitative Variablen. Für qualitative Variablen wird eine *Distanzmatrix* vorher berechnet, die aus den paarweisen Distanzen zwischen den Datenpunkten besteht. Im Falle einer Nominalskala sind es nur Einträge für „die beiden Punkte sind gleich“ oder „die beiden Punkte sind nicht gleich“. Dies kann binär kodiert werden. Es sind $n(n-1)/2$ Distanzwerte, die zusätzlich zu dem eigentlichen Clusteringverfahren an Rechenzeit anfallen.

Eine ganze Reihe weiterer Distanzmaße wurde entwickelt, um den Erfordernissen der speziellen Probleme gerecht zu werden. Als Beispiel wären die *Hausdorff-Distanz* oder die *mutual neighbour distance (MND)* [35] zu nennen. Interessante Distanzmaße sind die *Improved Hetergogeneous Distance Functions* [76], weil sie gemischte Daten, also Daten, die sowohl nominale als auch ordinale Daten enthalten, einbeziehen. Die ausführliche Behandlung aller, in der Literatur vorkommenden Distanzmaße würde weit über das Ziel dieser Arbeit hinausgehen. Der interessierte Leser sei auf die Übersichten in [35] oder [9] verwiesen.

2.3 Clusterarten

Die Frage, der in diesem Abschnitt nachgegangen wird, ist die über die möglichen Erscheinungsformen der Cluster. Diese hängen stark davon ab, für welche Zwecke die Cluster verwendet werden sollen und welche Daten sie enthalten dürfen. Für einige mag es genügen, dass man die Datenpunkte mit Bezeichnern (engl. *label*) versieht, welche das zugehörige Cluster angeben, z.B. wenn der Punkt x_i in das Cluster C_j gehört, so erhält der Punkt den Bezeichner j. Ein anderer bevorzugt eine abstraktere Vorstellung, bei der es mehrere Repräsentationen der Daten geben darf und dadurch ein Datum in mehreren Clustern enthalten sein darf. Diese Einzelheiten über die Beschaffenheit der Cluster bestimmt das zugehörige *Clustermodell*, welches vorab von dem Analytiker gewählt werden muss.

DEFINITION 3: (Clustermodell)
*Ein **Clustermodell** ist die Vorschrift, mit der die Daten den Clustern zugeordnet werden. Es beschreibt, welche Daten die Cluster enthalten dürfen.*

Zwei Attribute bestimmen das Clustermodell, *Hierarchie* und *Überdeckung*. Je nachdem, ob die Cluster sich überdecken dürfen und ob sie hierarchisch angeordnet werden sollen, trifft eins der folgenden Fälle zu:

1. **Überdeckung:** Die Überdeckung besagt, dass ein Datum in mehreren Clustern enhalten sein darf. Allerdings darf kein Cluster vollständig in dem anderen enthalten sein.

2. **Partition:** Bei der Partition darf ein Datum genau einem Cluster zugeordnet werden. Die Cluster überschneiden sich also nicht mehr und wir erhalten eine Partition über die Menge der Daten.

3. **Quasihierarchie:** Die Quasiehierarchie wird durch eine Folge von Überdeckungen gebildet. Die Daten der aktuellen Überdeckung sind die Cluster aus dem vorherigen Schritt. Die letzte Überdeckung enthält alle Elemente der Datenmenge.

4. **Hierarchie:** Die Hierarchie ist ein Spezialfall der Quasihierarchie, bei der statt Überdeckungen Partitionen verwendet werden.

Für welche Clusterart man sich auch immer entscheidet, vorher sollte man sich überlegen, ob alle Daten für das Clustering verwendet werden sollten, oder nur eine Auswahl. Im ersten Fall spricht man von dem *exhaustiven (erschöpfenden) Clustering,* im zweiten Fall von dem *nichtexhaustiven Clustering*. Das nichtexhaustive Clustering ist dann sinnvoll, wenn wir von verrauschten Daten ausgehen und befürchten müssen, dass sogenannte *Ausreißer*, also verfälschte Daten, die nicht der internen Struktur entsprechen, die Clusterzuordnung zu sehr verfälschen könnten.

In unserem System, das wir in den folgenden Kapiteln nach und nach vorstellen werden, werden wir durchweg das Clustermodell der Partition verwenden. Allerdings spricht nichts dagegen, in den Nachfolgearbeiten zu versuchen die Daten hierarchisch zu Clustern.

2.4 Evaluation der Cluster

Eine Clusterstruktur ist valide, wenn sie nachweislich nicht durch Zufall erzeugt wurde, oder ein Nebenprodukt eines Clustering-Algorithmus ist [35].

Für eine Analyse des Clusteringergebnisses reicht es manchmal schon aus, die Daten in einer evaluierbaren Form vorliegen zu haben [9]. Diese zeichnet sich vor allem dadurch aus, dass sie die Daten reduziert darstellt. Dabei können statistische Werte wie *Median*, *arithmetisches Mittel* und die *Varianz* als evaluierbare Repräsentanten ausreichen.

Eine andere Art, die Daten dem Analytiker vorzustellen, ist die über grafische Verfahren. Wenn die Daten in weniger als vier Dimensionen vorliegen, können sie grafisch dargestellt und die Cluster verschiedenartig eingefärbt werden. Bei hierarchischen Verfahren (siehe Kap. 2.3) ist es sinnvoller ein sogenanntes *Dendrogramm* zu erstellen. Ein **Dendrogramm** ist eine baumartige Struktur, dessen Kanten jede Stufe der Hierarchie mit den Elementen aus der vorherigen Stufe verbindet, aus denen es entstanden ist.

Ein Beispiel für ein Dendogramm ist in der Abbildung 2.2 dargestellt.

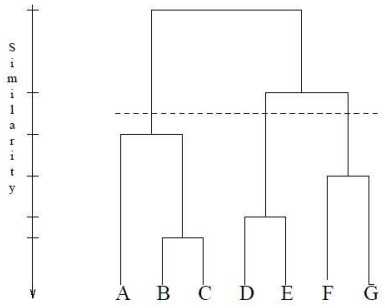


Abbildung 2.2: **Ein Beispiel für ein Dendrogramm.** Die Elemente A,...,G werden von unten nach oben von Hierarchiestufe zur Hierarchiestufe zu Clustern zusammengefasst.

Es existieren drei Typen der Evaluationsstudien [35]:

1. **Externe.** Diese Studie vergleicht die erzeugte Struktur mit einer vorgegebenen, *a priori* - Struktur.
2. **Interne.** Diese Studie versucht zu ermitteln, ob die ermittelte Struktur mit der intrinsischen Struktur übereinstimmen kann.
3. **Relative.** Diese Studie vergleicht zwei Strukturen und misst deren relativen Wert anhand von bestimmten Indizes.

Für die Güte des Ergebnisses eines Clusteringalgorithmus bestimmt man oft ein bestimmtes Kriterium, welches erfüllt werden muss. Da keine „goldenen Standards“, bis auf wenige präzise umrissenen Anwendungsgebiete, sich für Clustering etabliert haben, beruht die Bewertung der Validität auf objektiven Kriterien [35].

Ein wichtiges, manchmal einziges, objektives Kriterium für die Messung der Güte des Clusterverfahrens ist die Distanz. Wir versuchen die *Homogenität* innerhalb der Cluster und die *Heterogenität* zwischen den Clustern (Definition 2, S. 14) zu maximieren. Das bedeutet: Punkte innerhalb eines Clusters müssen zueinander eine kleine Distanz aufweisen, während Punkte in verschiedenen Clustern sehr große Distanz zueinander aufweisen sollten.

Die Auswertung der Homogenität und der Heterogenität kann mit relativ einfachen Mitteln wie dem Arithmetischen Mittel und der Varianz erfolgen. Näheres dazu findet man, z.B. in Hartung [30].

Wenn man Ergebnisse von Clusteringalgorithmen, bzw. verschiedene Clusterstrukturen vergleichen möchte, so bieten sich dem Analytiker etablierte Messkriterien. Die gebräuchlichsten sind das *Rand-Kriterium* [70], *F-Measure* [10] und *Conditional Entrophy* [9].

2.5 Ein paar philosophische Überlegungen zur Güte der Cluster

Die Frage der Bewertung der Güte[1] wird in diesem Abschnitt noch einmal aufgegriffen. Es ist eine sehr schwierig zu beantwortende Frage. Sie hängt vor allem von dem Zweck ab, für den die Clusteranalyse verwendet wurde.

Am spannendsten ist der Gebrauch der Clusteranalyse als ein Instrument zur Erforschung der Strukturen in den Daten[2]. Dabei ist zu wenig über die wirkliche Struktur der Cluster bekannt, als dass man den resultierenden Clustern überhaupt irgend eine Relevanz zuschreiben könnte. Im Bereich des Data-Mining existieren für dieses Szenario genügend Beispiele.

Die gesuchte Struktur der Daten hat einen Einfluss auf die Struktur der Cluster, aber die wirkliche Struktur der Daten wird nicht durch die Struktur der Cluster repräsentiert. [4]

Die gesuchte Struktur kann für die Daten einmalig sein, und es könnte sein, dass sie nicht durch die vorhandenen Daten vollkommen beschrieben werden kann. Es müssten weitere Daten hinzukommen, um die Struktur vollkommen aufzudecken.

Die Suche nach der internen Struktur in den Daten basiert auf den Ideen und Annahmen, die der Analytiker über die tatsächliche Struktur macht. Mit der Clusteranalyse überprüfen wir, ob die Annahmen des Analytikers eine Implikation auf die wirkliche Struktur haben oder nicht, und nichts mehr. Ohne diese Ideen des Analytikers haben die Ergebnisse der Clusteranalyse keine weitere Relevanz, als dass sie die Daten beschreiben.

Allerdings kann die Clusteranalyse von dem Analytiker dazu verwendet werden, die Daten, bzw. deren angenommene Struktur von einem ganz neuen Winkel zu betrachten. Durch die neue Sichtweise

[1] Die Überlegungen basieren hauptsächlich auf den Ausführungen von M.R. Anderberg in [4].
[2] Die anderen Anwendungen können in [4] nachgelesen werden.

können evtl. neue Ideen und Annahmen über die Struktur der Daten entstehen, die aber wiederum überprüft werden müssen. Wenn sie sich als nicht richtig erweisen, dann wird einfach nach neuen Ideen gesucht. Wenn sie sich aber verifizieren, dann interessiert es nachher niemanden mehr, dass man durch Clusteranalyseverfahren die Ideen erhalten hatte. Die Clusteranalyse ist also eine Hilfe zur Bildung von Hypothesen.

Bei der Suche nach den internen Strukturen werden zwei Sonderfälle oft übersehen:

1. *Die Struktur der Daten könnte überhaupt keine Cluster enthalten*. Es könnte nämlich sein, dass jedes der Datenpunkte von den anderen vollkommen unabhängig ist. Jede Gruppierung der Datenpunkte würde der echten Struktur der Cluster widersprechen.

2. *Die Daten könnten aus einem einzigen Cluster bestehen.* Sie könnten so eng miteinander korrelieren, dass jeder Versuch die Daten zu trennen zwangsläufig zu einer Verfälschung der inneren Struktur führen würde.

2.6 Übersicht über die Clustermethoden

Der Bedarf an Methoden, um automatisch in Daten nach verborgenen Strukturen zu suchen, hat im Laufe der Jahre in den unterschiedlichsten Bereichen, wie Biologie, Statistik oder Psychologie, eine schier unüberschaubare Menge an Verfahren hervorgebracht [35].

Die Clusteringverfahren können durch die in der Abbildung 2.3 dargestellten Taxonomie klassifiziert werden [9]. Dies ist eine von mehreren Ansätzen Clusteringmethoden zu klassifizieren (siehe z.B. [35]), und auch sie hat mit mehreren Unstimmigkeiten zu kämpfen, z.B. Überlagern sich einige Klassen dieser Taxonomie. Sie ist aber als Ansatzpunkt, für die Reihenfolge der Vorstellung der Methoden in dieser Arbeit vollkommen ausreichend.

Für die Klärung der Frage, welcher Clusteringalgorithmus für welchen Fall angewendet werden kann, bedarf es einer vorherigen Aufstellung der Anforderungen [9]:

1. Von welchem Typ sind die Attribute, die von dem Algorithmus verwendet werden?
2. Ist der Algorithmus auf große Datenmengen skalierbar?
3. Arbeitet der Algorithmus gut mit hochdimensionalen Daten?
4. Kann der Algorithmus Cluster mit irregulären Formen aufdecken?
5. Kann der Algorithmus gut mit Ausreißern (engl. *outlier*) umgehen?
6. Wie ist die Zeitkomplexität des Algorithmusses?
7. Hängt die Ausgabe des Algorithmusses von der Reihenfolge der Eingabe ab?
8. Handelt es sich um *Bezeichnung* oder *Zuordnung* (*Hard-* gegen *Fuzzy*-Clustering)?
9. Verwendet man Vorwissen oder von dem Benutzer vordefinierte Parameter?
10. Wie kann man die Ergebnisse interpretieren?

Die Vorstellung aller Clusteringmethoden auch nur im Ansatz würde die engen Grenzen diese Arbeit bei weitem sprengen. Wir beschränken uns auf die Aufzählung einiger ausgewählter Beispiele für jede Klasse der Clusteringmethoden. Eine detaillierte Darstellung aller Methoden findet sich, z.B. in [9, 35, 52]. Die Methoden, die für diese Arbeit relevant sind, werden aber intensiv behandelt.

Ganz oben in der Abbildung 2.3 sind die *Hierarchischen Clusteringverfahren* aufgeführt. Diese basieren auf der in dem Kapitel 2.3 vorgestellten Hierarchie als mögliche Ausgabe des Clusteringalgorithmusses. Wir haben einen Baum aus Clustern, welches, z.B. in einem *Dendrogramm* (Siehe Abb. 2.2)

dargestellt werden kann. Dabei gibt es zwei Richtungen. Entweder man fängt mit Clustern, in denen jeweils nur ein Datum enthalten ist und vereinigt die Cluster nach und nach, bis nur noch ein Cluster übrig bleibt (*Agglomeratives Clustering* (bottom-up)), oder man geht den umgekehrten Weg und fängt mit einem Cluster an und teilt es so lange in Untercluster, bis die Daten zum Schluss sich nur noch jeweils in einem Cluster befinden (*Divisives Clustering* (top-down)).

Die Vorteile des Hierarchischen Clusterings liegen in der Flexibilität bzgl. der Betrachtung der Hierarchieebene, der einfachen Integration beliebiger Ähnlichkeits- und Distanzmaße und die konsequente Anwendbarkeit auf beliebige Attributtypen. Die Nachteile liegen in der Vagheit der Definition der Abbruchkriterien und in dem Umstand, dass die meisten Methoden die Cluster nach der Erstellung nicht wieder besuchen [9].

Bekannte hierarchische Clusteringmethoden sind, z.B. AGNES [37], SLINK [59], CLINK [16], COBWEB [20], CURE [26], ROCK [27] und CHAMELEON [36].

Eine andere Herangehensweise an die Problematik bieten die *Partitionierungsalgorithmen* . Diese ermitteln eine einzige Partition der Daten und lernen die Cluster direkt, entweder durch die Verschiebung der Punkte zwischen den Untermengen (*Relokationsalgorithmen*) oder durch die Identifizierung der Cluster als Bereiche mit einer hohen Datendichte (*Density-Based-Algorithmen*) [9].

Die Relokationsalgorithmen gehören zu dem Bereich von *iterative optimization* [9]. Da die Überprüfung aller Untermengen eine zu lange Laufzeit erfordert, werden verschiedene *Greedy-Heuristiken* verwendet, um die Punkte zwischen den k Clustern zu verschieben (engl. *reassign*).

Probabilistisches Clustering basiert auf der Annahme, dass man die Daten mit einem bestimmten Modell identifizieren kann, deren Parameter noch bestimmt werden müssen, wir suchen also nach der Verteilung der probabilistischen Modelle [9]. Zu dieser Klasse gehören, z.B. die *Expectation-Minimation* (EM) – Methode [47] oder SNOB [71].

In dem Bereich *Fuzzy Clustering* untersuchte man graduelle Zugehörigkeit der Daten zu Clustern, welche durch eine *Zugehörigkeitsfunktion* (engl. *membership function*) definiert wird [81]. Die Daten können also nicht mehr nur einem Cluster angehören, sondern gehören allen Clustern zu einem bestimmten Anteil an. Einige Algorithmen wie, z.B. k-Means wurden so modifiziert, dass sie auch Fuzzy-Cluster bilden konnten (z.B. *Fuzzy-k-Means* [35] oder *Gustafson-Kessel-Clustering* (GK)).

Andere Ansätze wie *k-Means-* (Kap. 2.6.1) und *k-Medoids*-Methoden definieren eine *Zielfunktion*, die von der Partition abhängt. Dabei verwendet man jeweils nur einen Repräsentanten pro Cluster, was die Laufzeit der Algorithmen erheblich senkt. In k-Medoids-Methoden werden die Cluster durch die Punkte innerhalb des Clusters repräsentiert, die zu den Clusterzentren am nächsten liegen. Beispiele für k-Medoids wären die Algorithmen CLARA (CLustering LARge Applications) [37] und PAM (Partitioning Around Medoids) [51]. Der große Vorteil der k-Medoids-Methoden ist, dass sie sich, im Gegensatz zu k-Means, von Ausreißern nicht so stark verfälschen lassen.

Die Vorteile der bis jetzt vorgestellten Partitionierungsalgorithmen sind, dass sie wegen ihrer Effizienz hervorragend auf große Datenmengen anwendbar sind. Die Nachteile liegen vor allem in der Schwierigkeit die optimale Anzahl der Cluster zu bestimmen, was zu einer inakzeptablen Verschlechterung der Laufzeit führen kann, und in der gleichen Form (z.B. Kugel) der Cluster [35].

Die *Density-Based-Algorithmen* folgen der Idee, dass in dem Euklidischen Raum Bereiche mit großer Dichte lokalisiert und abgegrenzt werden können. Um dies technisch zu realisieren, bedarf es der Klärung der Begriffe *Dichte*, *Verbundenheit* und *Dichtegrenzen*, was sehr stark mit den nächsten Nachbarn eines Punktes zusammenhängt [9]. Es werden zwei Bereiche dieser Ansätze unterschieden, *Density-Based-Connectivity*, die auf der Nähe zu den nächsten Nachbarn basieren, und *Density-Functions*, die auf der Definition der Dichtefunktionen über dem Merkmalsraum aufbauen [9]. Beispiele für Density-Based-Connectivity sind DBSCAN [18], OPTICS [5], DBCLASD [79] und für Density-Functions ist der Algorithmus DENCLUE [32]. DBSCAN wird in dem Kapitel 2.6.2 näher erläutert. Ein weiterer Ansatz, welcher etwas aus dem Rahmen fällt, ist der auf den *Support Vector Machines* (SVMs) basierende Clusteringansatz von Ben-Hur et al. [8]. Dieser ist unter anderem fähig mit hochdimensionalen Daten umzugehen.

Der Vorteil dieser Methoden ist, dass sie „einen natürlichen Schutz gegen Ausreißer bereitstellen“ [9].

Sie erzeugen Cluster mit beliebigen zusammenhängenden Formen. Außerdem ist es nicht mehr notwendig die Anzahl der Cluster im Voraus anzugeben. Allerdings sind es andere Parameter, wie die Dichtegrenze, die richtig gesetzt werden müssen. Und nicht zuletzt bleiben diese Methoden auf Datenmengen in metrischen Räumen beschränkt.

Die sogenannten *Grid-Based-Methoden* arbeiten indirekt mit Daten, indem sie diese zusammenfassen und somit den Raum segmentieren. Man verlagert also die Aufmerksamkeit von den Daten zur Raumpartitionierung [9]. Beispiele für diese Methoden sind die Algorithmen DENCLUE [32], CLIQUE [1], COBWEB [20], MAFIA [22], BANG [57], STING [72] und WaveCluster [56].

Die Methoden, die auf *Co-Occurence of Categorical Data* basieren, sind eng verknüpft mit transaktionellen Datenbanken, das bedeutet, sie beziehen zusätzliche Informationen zu dem Ähnlichkeitsmaß aus den Informationen über das gleichzeitige Auftreten von Daten bei Transaktionen. Beispiele dazu sind die Algorithmen ROCK [27], SNN [17] und CACTUS [21].

Durch die Hinzunahme von zusätzlichen Informationen in Form von Bedingungen (engl. *constraints*) erhält man eine Art *semisupervised clustering*. Diese Ansätze werden *Constraints Clustering* genannt und im Kapitel 2.6.3 ausführlich erläutert.

Eine weitere Sparte der Clusteringalgorithmen stellen die *naturanalogen Verfahren*. Zu diesen zählen Algorithmen aus dem Gebiet der *Künstlichen Neuronalen Netzen* (KNNs oder ANNs), also *Gradientenabstiegsmethoden* und die *Evolutionären Algorithmen* (siehe Kap. 6.7.6). Beispiele für die erste Kategorie wären LKMA [43], SOM [39] und für die evolutionären Ansätze GCA [28].

Das Clustern von großen Datenbanken setzt erweiterbare (engl. *scalable*) Algorithmen voraus. Das bedeutet, dass diese nun mit einer wachsenden Datenmenge umgehen müssen. Beispiele für diese Algorithmen sind DIGNET [62] und BIRCH [82].

Mit dem Ansteigen der Anzahl der Dimensionen in den Daten kommt das Problem der Verringerung der Trennbarkeit (engl. *separation*) der Metrik (*curse of dimensionality*) [9]. Auf der einen Seite könnte man *dimensionality reduction* betreiben, wie z.B. in den Algorithmen PCA [42] und Wavelets [38]. Auf der anderen Seite existieren Algorithmen, die auf Unterräumen arbeiten (*subspace clustering*). Beispiele dafür sind CLIQUE [1], MAFIA [22], ENCLUS [12] und OptiGrid [33]. Eine dritte Herangehensweise an diese Problematik ist der Versuch die Merkmale selbst zu clustern und dadurch eine reduzierte Merkmalsmenge zu erhalten. Dieser Ansatz wird mit *Co-Clustering*, *simultaneous clustering*, *bi-dimensional clustering* etc. [9] bezeichnet und geht auf den Ansatz von Anderberg [4] zurück.

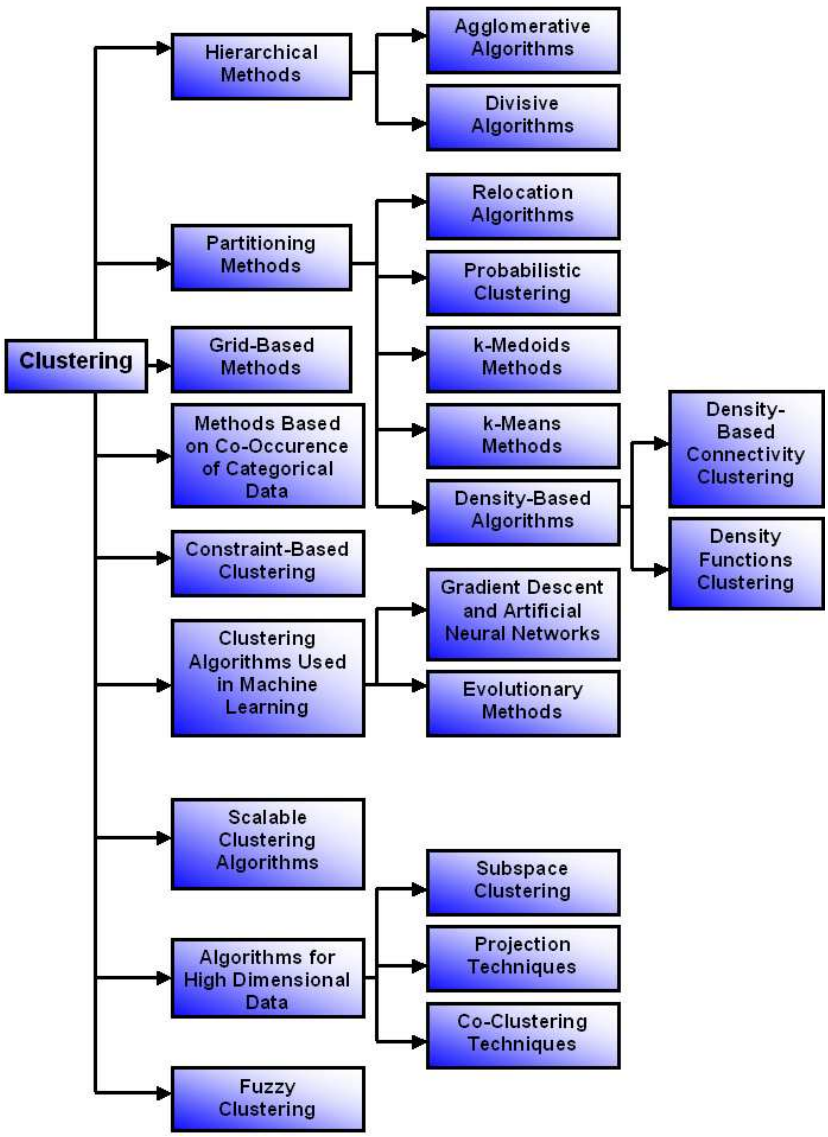


Abbildung 2.3: **Hierarchische Übersicht der Clusteringalgorithmen.**

2.6.1 k-Means

Eine der bekanntesten und am weitesten angewandten Algorithmen ist *k-Means* [9,35]. Es existiert eine große Anzahl von Definitionen dieses Algorithmusses, z.B. [9, 10, 35, 70]. Aus Gründen, die im Laufe dieser Arbeit ersichtlich werden, präsentieren wir hier die auf einer Zielfunktion basierende Version [7].

k-Means ist ein Clusteringalgorithmus, welcher auf der iterativen Relokation der Datenpunkte basiert und diese in k Cluster so einordnet, dass die quadrierte Euklidische Distanz zwischen den Punkten im Cluster und dem Clusterzentrum minimiert wird [70]. Für eine Menge an Datenpunkten $X = x_1, \ldots, x_n, x_i \in \mathbb{R}^d$ erzeugt k-Means eine k-fache Partition $\{X\}_{l=1}^{k}$, so dass $\mu_1, \ldots, \mu_n$ die Clusterzentren (*centroids*) repräsentieren und die Zielfunktion

$$J_{\text{kmeans}} = \sum_{l=1}^{k} \sum_{x_i \in X_l} ||x_i - \mu_l||^2 \tag{2.3}$$

lokal minimiert wird. Es kann gezeigt werden, dass k-Means equivalent ist zu dem oben erwähnten EM-Algorithmus [10].

Ein allgemeiner k-Means-Algorithmus geht wie folgt vor [35]:

1. Wähle k Clusterzentren.
2. Orde jeden Datenpunkt dem nächsten Clusterzentrum zu
3. Berechne erneut die Clusterzentren, indem jeweils der Mittelwert über alle Datenpunkte im Cluster ermittelt wird
4. Wenn ein Abbruchkriterium nicht greift, dann gehe zu Schritt 2), ansonsten STOP. Typische Abbruchkriterien sind: bestimmte Anzahl an Iterationen, keine (oder kaum) Veränderungen der Zugehörigkeiten der Datenpunkte, oder kaum Verminderung des quadratischen Fehlers.

Das Verfahren arbeitet sehr schnell, die Laufzeit kann mit $\mathcal{O}(n)$ abgeschätzt werden, da man eine konstante Anzahl an Iterationen benötigt, wenn man annimmt, dass das Abbruchkriterium eine vorgegebene Anzahl an Iterationen ist. Das ist ein wichtiger Grund für die Popularität dieser Methode. k-Means arbeitet nicht gut mit qualitativen Variablen, dafür hat es eine gute geometrische und statistische Begründung für quantitative Variablen [9]. Da man für Cluster ihre Clusterzentren als Repräsentanten wählt, ist dieses Verfahren anfällig für Ausreißer und Verzerrungen. Es bleibt in lokalen Optima hängen, welche weit von dem tatsächlichen Optimum sein können. Es ist auch nicht klar, wie man das Optimale k wählen sollte. Die Cluster haben wegen der Repräsentation der Cluster durch Clusterzentren alle die gleiche Form (meistens eine Kugel) und die gleiche Größe, das bedeutet, dass das Verfahren keine nicht-isotrophe Cluster entdecken kann. Und, in den ersten Versionen von k-Means, konnten die Cluster sehr unballanciert, oder sogar leer sein.

Es wurden etliche Versuche unternommen, um die aufgezählten Nachteile von k-Means zu vermindern. Leider ging das oft auf Kosten der Laufzeit, was ja eins der hervorragendsten Eigenschaften dieses Verfahrens ist. In [9] findet sich eine Aufzählung dieser Algorithmen.

Der Algorithmus wurde wegen seiner Vorteile als Grundlage für viele Ansätze für das *Semisupervised Clustering* (siehe Kap. 2.6.3) in abgewandelter Form verwendet, z.B. in [7, 10, 54, 70].

2.6.2 DBSCAN - Density Based Spatial Clustering of Applications with Noise

K-Means hatte, wie wir gesehen haben, den Nachteil, nur kugelförmige Cluster von gleicher Größe zu bilden. Dies ist für allgemeine Daten eine zu große Einschränkung. Wenn man, z.B. auf Satellitenfotos nur die Häuser, die entlang eines Flusses stehen, clustern will, dann weiß man schon intuitiv, dass diese nicht in kreisförmigen Clustern angeordnet sind. Ein Mensch, der diese Satellitenfotos betrachtet,

erkennt sofort, wie die Daten richtig partitioniert werden sollten. Die Hauptursache dafür ist, dass innerhalb eines Cluster eine *Dichte* an Punkten vorliegt, die beträchtlich höher ist, als außerhalb des Clusters. Außerdem ist die Dichte innerhalb der Bereiche mit verrauschten Daten niedriger als die Dichte innerhalb eines beliebigen Clusters [18]. Die Dichtebasierten Algorithmen versuchen den Begriff *Dichte* so genau wie möglich zu definieren und darauf aufbauend effizient Cluster zu bilden, die eine beliebige Form haben können.

In diesem Kapitel stellen wir den effizienten dichtebasierten Algorithmus DBSCAN [18] vor, dessen Ergebnisse wir in dieser Arbeit mit denen von k-Means für die Musikdaten vergleichen werden. Durch die Beliebigkeit der Formen der Cluster hoffen wir dabei genauere Ergebnisse zu erzielen. Allerdings sollte man stets auch den Nachteil dieses Verfahrens im Auge behalten, dass DBSCAN nur für wenigdimensionale Daten gute Ergebnisse liefert [18].

Definition der Dichtebegriffe

Zunächst wäre anzumerken, dass DBSCAN so konzipiert ist, dass es für beliebige Distanzmaße verwendbar ist, insbesondere für das in dieser Arbeit verwendete Euklidische Distanzmaß.

DEFINITION 4: (Eps-Nachbarschaft [18])
*Die **Eps-Nachbarschaft** eines Punktes x_i, welche als $N_{\mathrm{Eps}}(x_i)$ bezeichnet wird, ist definiert durch* $N_{\mathrm{Eps}}(x_i) = \{x_j \in X |\ D(x_i, x_j) \leq \mathrm{Eps}\}, \mathrm{Eps} \in \mathbb{N}$.
$D(x_i, x_j)$ ist die Distanz zwischen den Punkten x_i und x_j.

Ein naiver Ansatz könnte nun so aufgebaut sein, dass für jeden Punkt in einem Cluster eine Mindestanzahl an Punkten (*MinPts*) existieren muss, die in der *Eps-Nachbarschaft* von diesem Punkt liegen. Aber dieser Ansatz führt nicht zu einem guten Ergebnis, denn es existieren zwei Arten von Punkten in einem Cluster: *core points* und *border points*. Border points sind Punkte am Rande des Clusters und die Core points sind die anderen Punkte in dem Cluster. Die Eps-Nachbarschaft eines Core points ist signifikant größer als die eines Border points [18]. Daraus folgt, dass die minimale Anzahl der Punkte relativ klein gewählt werden muss, um alle Punkte zu erkennen, die zum gleichen Cluster gehören. Darauf basiert die folgende Definition.

DEFINITION 5: (Directly density-reachable [18])
*Ein Punkt x_i ist **directly density-reachable** von einem Punkt x_j bzgl. Eps, MinPts, wenn*

1. $x_i \in N_{\mathrm{Eps}}(x_j)$ *und*
2. $|N_{\mathrm{Eps}}(x_i)| \geq MinPts$ *(core-point-Bedingung)*

Diese Definition nimmt die Relationen zwischen den Punkten als symmetrisch an, allerdings stimmt das nicht, wenn ein Core point mit einem Border point auftritt. Deswegen wird die folgende Definition aufgestellt.

DEFINITION 6: (Density reachable [18])
*Ein Punkt x_i ist **Density reachable** von einem Punkt x_j, wenn es eine Kette von Punkten $x_1, \ldots, x_n,\ x_1 = x_i, x_n = x_j$ existiert, so dass x_{k+1}* directly density-reachable *von dem Punkt x_k ist.*

Diese Erweiterung der Definition 5 ist eine Relation, die zwar transitiv ist, allerdings nicht symmetrisch. Ein weiterer Fall muss noch durch unsere Definitionen abgedeckt werden, denn zwei Border points eines Clusters sind nicht unbedingt density reachable zueinander [18].

DEFINITION 7: (Density connected [18])
*Ein Punkt x_i ist **Density connected** zu einem Punkt x_j, wenn ein Punkt x_k existiert, so dass x_i und x_j density reachable von x_k sind.*

Nun sind wir soweit, dass wir eine Definition der Cluster aufstellen können.

DEFINITION 8: (Cluster [18])
Sei X eine Datenbank aus Punkten. Ein Cluster C ist eine nicht leere Untermenge von X, welche die folgende Bedingungen erfüllt:

1. $\forall x_i, x_j$: *Wenn $x_i \in C$ und x_j ist density reachable von x_i, dann ist $x_j \in C$. (**Maximalität**).*
2. $\forall x_i, x_j \in C$: *x_i ist density connected zu x_j (**Konnektivität**).*

DEFINITION 9: (Rauschen)
*Seien $C_1 \ldots C_k$ die Cluster der Datenbank X. Das **Rauschen** ist definiert als die Menge der Punkte, die zu keinem der Cluster C_i zugehören, also Rauschen = $\{x_i \in X | \forall j : x_i \notin C_j\}$*

In unserem System, welches wir in dieser Arbeit vorstellen werden, dürfen keine verrauschten Daten existieren, genauer gesagt, alle Musikstücke müssen jeweils einem Cluster zugewiesen werden. Wir umgehen diese Problematik, indem wir ein zusätzliches Cluster erschaffen, dem die von DBSCAN unberücksichtigten Musikstücke zugewiesen werden.

Abschätzung von Eps und MinPts

In [18] werden die beiden Werte Eps und MinPts durch eine Heuristik abgeschätzt, die die Werte des Clusters mit der geringsten Dichte ermittelt.

Die Heuristik basiert auf der folgenden Beobachtung: Sei d die Distanz des Punktes p zu dem k-Nächsten Nachbarn. Dann enthält die d-Nachbarschaft des Punktes p in den meisten Fällen genau $k+1$ Punkte.

Für ein gegebenes k definieren wir eine Funktion k-dist, welche jeden Punkt auf die Distanz zu seinem k-Nächsten Nachbarn abbildet. Wenn man nun die Punkte nach der Entfernung zu deren k-Nächsten Nachbarn sortiert, ergeben sich einige Hinweise auf die Dichteverteilung in der Datenbank.

Wir suchen den ersten Knick in dem Graphen, welcher das Rauschen von den Clusterdaten trennt. Die Abbildung 2.4 veranschaulicht dies genauer. Der Wert an diesem Knickpunkt ist dann der Eps-Wert und MintPts ist gleich k. Für $k > 4$ unterscheiden sich die Graphen nicht mehr signifikant, so dass k auf 4 gesetzt werden kann [18].

Leider ist die automatische Erfassung eines „Knicks“ schwierig, so dass in [18] vorgeschlagen wird, diesen Knickpunkt von dem Benutzer angeben zu lassen, da dieser die „Knicke“ sofort erkennen könne.

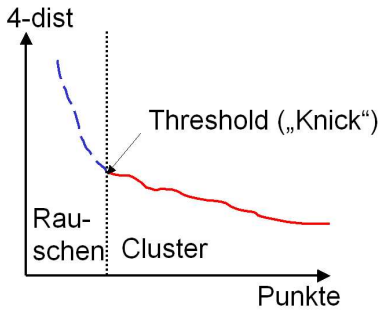


Abbildung 2.4: **Ein sortierter 4-dist-Graph aus [18].** Der Knickpunkt des Graphen bestimmt den Grenzwert zwischen dem Rauschen und den Punkten, die in ein Cluster gehören. Der 4-dist-Wert an dieser Stelle ist der zugehörige Eps-Wert. Der MinPts-Wert ist 4.

DBSCAN – Der Algorithmus

Aus diesen Definitionen können wir nun den Algorithmus aufbauen, welcher effizient aus unseren Daten ein Clustering erstellt.

Zunächst werden die Parameter Eps und MinPts des Clusters mit der geringsten Dichte mit Hilfe der im vorherigen Abschnitt vorgestellten Heuristik abgeschätzt.

Die allgemeine Vorgehensweise von DBSCAN ist folgendermaßen:

1. Starte mit einem beliebigen Punkt x_i und ermittele alle Punkte, die density reachable zu diesem Punkt sind.

2. Wenn x_i ein Core point ist, dann bilden die gefundenen Punkte schon ein Cluster.

3. Wenn x_i ein Border point ist, dann gibt es keinen Punkt, welcher zu diesem Punkt density reachable ist.

4. Besuche den nächsten Punkt in der Datenbank.

Die Umsetzung dieses Scripts in eine effiziente rekursive Implementierung, die eine durchschnittliche Laufzeit von $\mathcal{O}(n \log n)$ hat, wird ausführlich in [18] beschrieben.

Die Vorteile dieses Algorithmusses sind, dass er Cluster beliebiger Form entdecken kann, genau die Ausreißer zu lokalisieren und zu eliminieren vermag, sehr effizient ist, und in der Güte der Clusterfindung anderen Clusteringalgorithmen, wie z.B. CLARANS [37] überlegen ist [18]. Außerdem ist er speziell für große Datenmengen entwickelt worden.

Dieser Algorithmus hat nur wenige Nachteile. Am gravierendsten ist wohl, dass alle Punkte benötigt werden, um ein Cluster zu repräsentieren. Bei k-Means, z.B. reichen schon die Clusterzentren, um eine eindeutige Repräsentation der Cluster zu erhalten. Die heuristische Ermittlung der Parameter gilt ebenfalls als ein Nachteil, denn nach [18] muss der Benutzer in diesen Prozess mit einbezogen werden, da eine automatische Ermittlung der Parameter zu aufwendig wäre.

2.6.3 Semisupervised Clustering

Die Aufgabe, Daten zu clustern ist von Natur aus subjektiv. Jeder Clusteringalgorithmus benutzt einen gewissen Anteil an Vorwissen, entweder imlizit oder explizit [35]. In diesem Abschnitt wollen wir solche mit explizitem Vorwissen behandeln.

Semisupervised Clustering integriert einen kleinen Anteil an annotierten (engl. *labelled*) Daten, um dem *unsupervised learning* zu helfen [10]. Die Etikettierung muss durch einen Interaktionsprozess mit dem Benutzer erzeugt werden. Die Bereiche dieses Gebiets werden durch die Art der Etikettierung unterschieden. *Constrained Clustering* ist ein Teilbereich, in dem die Etikettierung durch *Bedingungen* (engl. *constraints*) erzeugt wird, die die Gruppierung der Daten im Sinne des Benutzers verbessern.

Dabei unterscheidet man *Prozessorientiertes Clustering* und *Ergebnisorientiertes Clustering* (z.B. in [9] oder [35]). Prozessorientiertes Clustering bindet den Benutzer in den Clusteringprozess ein, während Ergebnisorientiertes Clustering die Constraints vor dem Start des Clusteringprozesses vollständig ermittelt. Beide Ansätze haben ihre Berechtigung, allerdings werden wir in dieser Arbeit den zweiten bevorzugen, da dieser die Prozesse der Constraintsdefinition und der Constraintsanwendungen besser trennt, so dass der Benutzer nichts von den Internas der verwendeten Clusteringmethode mitkriegt.

Welche Constraints können dabei formuliert werden? Es existieren mehrere Ansätze mit dem folgenden Constraintspaar [7, 10, 70]:

- **Must-Link** spezifiziert, dass zwei Instanzen auf jeden Fall dem gleichem Cluster zugeordnet werden können.

- **Cannot-Link** spezifiziert, dass zwei Instanzen auf keinen Fall dem gleichen Cluster zugeordnet werden dürfen.

Diese Art Constraints werden *Instance level constraints* [23] genannt.

In [54] werden noch sechs weitere Constraints spezifiziert, die in dieser Arbeit in einer für die Musikdaten angepassten Form verwendet werden (siehe Kap. 4). Es handelt sich sowohl um *Objektconstraints*, die einzelne Datenpunkte betreffen, als auch um *Parameterconstraints*, die auf Merkmalen arbeiten. Eine weitere Art von Constraints sind die *Existential Constraints* oder *Aggregate Constraints* [23], die durch Grenzen einer Funktion (Min, Var, etc.) über jeden Cluster beschrieben werden können (Constraints vom Typ 4-6, Kap. 4.4.4 - 4.4.6). Ein Ziel dieser Arbeit ist zu klären, welche dieser Constraints sich, sowohl aus der Sicht des Benutzers als auch aus der Sicht der objektiven Evaluation[3] als sinnvoll erweisen werden.

Am Häufigsten werden abgewandelte Formen von k-Means für Constrained Clustering Algorithmen verwendet [9]. Beispiele dafür sind SEGMENT [54], COP-k-Means [70], SPK-Means [7]. Dabei unterscheidet man, welchen Einfluss die Constraints auf das k-Means haben können. *Seeding* [7] ist das Setzen der Startbelegung, aus der k-Means nach dem üblichen Verfahren Cluster bildet. Die andere Möglichkeit ist die Integration der Constraints in jede Iteration des Verfahrens.

Lernen der Distanzmetrik

Ein Teil der Verfahren, die für Semisupervised Clustering entwickelt wurden, beruht auf der Idee, die vorgegebene Distanzmetrik so durch Gewichte oder Gewichtsmatrizen zu modifizieren, dass eine Struktur entsteht, in der das vorhandene Wissen über die Distanzen, also die Constraints, mitverarbeitet worden ist. Dadurch können Verfahren, die ansonsten kritisch von einer guten Metrik abhängen, wie z.B. die Clusteringmethoden, trotz der Nebenbedingungen zum Metriklernen verwendet werden, ohne dass der Benutzer dazu gezwungen wird manuell die Metrik zu justieren [78].

Viele Verfahren beruhen auf dieser Idee, ohne sie explizit zu nennen (z.B. [54], [70]). Andere (z.B. [78], [66]) ziehen ihre gesamte Argumentation anhand dieser Betrachtung auf. In dieser Arbeit wird auf diese Herangehensweise im Detail im Kapitel 6.7.5 eingegangen.

Die Idee basiert auf den oben vorgestellten Must- und Cannot-Links, die die vorhandene Wissensbasis repräsentieren. Diesmal versucht man aber systematischer vorzugehen und beschränkt sich nicht auf bloße Angabe über Zugehörigkeit zu den Clustern, sondern argumentiert mit Distanzen. Die Distanz zwischen den Instanzen eines Must-Link Paares soll klein sein, während die Distanz zwischen den Instanzen eines Cannot-Link Paares möglichst groß sein soll. Dabei wird implizit angenommen, dass kleinere Distanzen auch zu ähnlicheren Clustern führen. Aufgrund dieser Vorgaben werden Parameter (Gewichte) gewählt, die die gesamte Metrik (z.B. die Euklidische Distanz) manipulieren. Für die Parametrisierung der Metriken wurden in der Literatur mehrere Methoden untersucht [3], [75]. Wir verwenden den im Kapitel 6.3 vorgestellten Gewichtsvektor, um die Distanzen zu manipulieren.

Information Bottleneck-Ansätze

Die bisher vorgestellten Constraintsarten enthalten *positive Informationen*, das bedeutet, der Benutzer gibt Informationen über das, was er gerne hätte, an, also, z.B. Must-Links oder auch Cannot-Links. Eine andere Möglichkeit besteht darin, dass der Benutzer *negative Informationen* angibt, also zeigt, welche Bedingungen nicht erfüllt werden dürfen [23, 24]. Es sind, z.B. negative Beispiele dafür, wie ein Clustering nicht aussehen darf.

Ein solcher Ansatz ist die *Information Bottleneck Methode* (IB) [23, 24, 63]. Sie basiert auf Ideen aus der *Informationstheorie* und *Signalverarbeitung*, die wiederum auf den Arbeiten von Shannon aufbauen, welcher das Augenmerk auf das Problem des Transports der Information, statt der bloßen Beurteilung der erhaltenen Information, gelenkt hat.

Sei das Signal $x \in X$ die relevante Information über ein anderes Signal $y \in Y$. Wenn wir ein Signal x erhalten, dann müssen wir nicht nur y vorhersagen können, sondern auch spezifizieren, welche Merkmale von X eine Rolle für die Vorhersage spielen. Man formuliert das Problem als das Finden einer kürzesten Kodierung von X, welche aber eine maximale Information über Y enthält. „That is,

[3] Die Constraints sollen zu einer Verbesserung beitragen und nicht mögliche Optima blockieren.

we squeeze the information that X provides about Y through a '*bottleneck*' formed by a limited set of codewords X" [63]. Es ist also die Suche nach der *relevanten* oder *nicht redundanten* Information. Man geht davon aus, dass eine zusätzliche Variable existiert, die die Relevanz der Information beschreibt. In diesem Fall ist dies die oben erwähnte negative Information.

Die vorhandenen Strukturinformationen werden dazu verwendet, um neue, bis dahin nicht vorgekommene Strukturen zu Tage treten zu lassen[4]. Dieses Ziel unterscheidet sich von dem, in dieser Arbeit verfolgten, dadurch, dass wir einen Teil der Struktur vorgeben und erwarten, dass das System den Rest nach den Vorgaben konstruiert (*positive Information*), während in dem Informational Bottleneck die bekannten Strukturen als irrelevant, bzw. nicht erwünscht (*negative Information*) erklärt werden.

Die Informational Bottleneck Ansätze optimieren die *gemeinsame Information* (engl. *mutual information*) $I(C;Y|Z)$ über die Clustermenge C mit der durch die bekannte Informationsmenge Z bedingten Merkmalsmenge Y, um damit die Wahrscheinlichkeiten $P_{C|X}$ für jedes $x \in X$ einem Cluster $c \in C$ zugewiesen zu werden abzuschätzen [23, 24]. Dabei kann man als Nebenbedingung eine gewisse „Schärfe" (engl. *sharpness*) festlegen, um zu eine zu vage Zuweisung zu verhindern. Dieser Ansatz wird *Conditional Information Bottleneck* (CIB) [63] genannt. Als eine weitere Einschränkung dieser Methode wurde in [23] die *Coordinated Conditional Information Bottleneck* (CCIB) Methode vorgestellt, bei der zusätzlich Lösungen mit einer globalen Konsistenz bevorzugt werden. Implementiert werden diese Ansätze mit einem ausgefeilten alternierenden Optimierungsschema, welches iterativ sich an die optimalen Zielwahrscheinlichkeiten annähert.

2.7 Zusammenfassung

Musik ist eine Kunst. Das bedeutet, die Empfindungen, die man beim Hören der Musikstücke hat, hängen sehr stark von dem Benutzer ab. Genauso subjektiv ist auch die Gruppierung der Stücke. Was für den einen die ideale Aufteilung ist, ist für den anderen nur zufällig gebildete Cluster. Daher ist der Ansatz des *Constrained Clustering* (Kap. 2.6.3) besonders attraktiv.

Zunächst einmal müssen die Musikdaten in eine gut maschinell zu verarbeitende Form gebracht werden, es müssen repräsentative Merkmale extrahiert werden, die eine effiziente Verarbeitung erlauben. Dies geschieht im nächsten Kapitel 3.

[4] Siehe dazu auch den Diskurs über die Anwendungsgebiete von Clusteringmethoden in dem Kapitel 2.5

3 Merkmale und deren Extraktion

»Beschriebene Musik ist halt wie ein erzähltes Mittagessen.«
Franz Grillparzer

Schallwellen werden durch die Schwingungen fester Körper erzeugt und mit dem Medium Luft übertragen. Die Vibration des festen Körpers überträgt sich auf die Luftmoleküle, die wiederum andere Moleküle anstoßen und so weiter bis sie das menschliche Ohr erreichen und das Trommelfell durch ihren Aufprall zum Schwingen bringen. Da sie mit der gleichen Frequenz aufprallen, wie sie von dem Tonerzeuger angestoßen wurden, empfangen wir genau die gleichen Signale, die von dem Erzeuger abgesendet wurden, nur schwächer und evt. vermischt mit anderen Schwingungen, die von anderen Tonträgern ausgehen.

Das menschliche Ohr leitet diese Schwingungen weiter und kodiert sie in elektrochemische Signale, die durch die Nervenbahnen weitergeleitet und vom Gehirn ausgewertet werden, so dass wir das empfangene Geräusch wahrnehmen und verarbeiten können.

Musikstücke sind also, genauso wie alle Geräusche, akkustische Wellen und damit reellwertige Funktionen der Zeit [46].

Es sind analoge Wellen, die zunächst digitalisiert werden müssen. Die Digitalisierung erfolgt, indem man die Wellen in kleine Zeitintervalle zerteilt, und denen einen Wert zuweist. Dadurch erhalten wir eine endliche Menge an Zeitabschnitten mit einer endlichen Menge an Werten. Je kleiner die Zeitintervalle sind, desto genauer bilden wir das Musikstück ab, und desto mehr Informationen müssen wir speichern.

DEFINITION 10: (**Sampling [46]**)
Die Anzahl der Zeitintervalle pro Sekunde wird mit ***sampling rate*** *bezeichnet und dessen Einheit ist Herz, Hz.*
Der Prozess der Einteilung der Musikstücke in Zeitintervalle wird ***sampling*** *genannt. In der Statistik heißt eine solche Reihe an Daten* ***Wertereihe****. Im Folgenden werden diese beiden Begriffe äquivalent verwendet.*
Die Zeitintervalle, also die kleinsten noch unterscheidbaren Zeiteinheiten eines digitalen Musikstückes werden ***sample points*** *genannt.*
Zu jedem dieser ***sample points*** *gehört ein Wert der Schwingung, der als* ***Elongation*** *bezeichnet wird. Sie gibt die momentane Entfernung von der Ruhelage wieder. Sie ist proportional zu der Auslenkung einer Lautsprechermembran beim Abspielen des Musikstückes.*

Wir haben es also mit einer diskreten Wertemenge zu tun, die ein Musikstück sehr gut repräsentiert. Leider können wir mit dieser Masse an Daten nicht sehr viel anfangen, denn auch die schnellsten Algorithmen stoßen dabei auf ihre Grenzen.

Eine Idee wäre es, nur einen Ausschnitt aus dem gesamten Musikstück zu betrachten. Doch auch dabei stellt sich die Frage, welcher Ausschnitt am besten für alle möglichen Musikstücke ist, denn wir wollen ja die Musikstücke untereinander vergleichen, bzw. aufgrund derer Repräsentationen ein Clustering vornehmen und Distanzmaße aufstellen. Es müsste also für alle Musikstücke die gleiche Position sein, um sie vergleichbar zu machen. Leider sind die Musikstücke nicht exakt gleich lang. Es braucht nur ein einzelner *sample point* zu einem Musikstück hinzuzukommen, und schon wird es zu seinem vorherigen selbst nicht mehr als gleich betrachtet.

Man sieht, man kommt nicht mit der Betrachtung des *samplings* weiter. Man benötigt wenige, aber dafür sehr repräsentative Merkmale, die sich nicht durch die formalen Unterschiede der Musikstücke beirren lassen.

Ingo Mierswa hat in seiner Arbeit [46] wenige signifikante Merkmale, die ein Musikstück repräsentieren, mit den Methoden der Wertereihenanalyse und einem darauf aufbauenden Genetischen Algorithmus extrahieren können. Er erhielt 49 reelwertigen Merkmale, mit denen er hervorragende Ergebnisse für die Klassifikation nach den Musikgenres erzielt hat. Im Folgenden werden wir sie als eine Repräsentation der Musikstücke verwenden ohne näher auf deren Extraktion einzugehen. Uns reicht es zu wissen, dass diese Merkmale gut genug die Musikdaten repräsentieren können. Der an die Einzelheiten der Merkmalsgenerierung interessierte Leser sei auf [46] verwiesen.

3.1 Die Merkmale

Im Folgenden werden die in unserem System verwendeten Merkmale aufgelistet und erläutert.

Länge des Musikstückes: Die Länge der Wertereihe, die das Musikstück repräsentiert, ist für alle Benutzer klar verständlich. Sie kann sehr gut zur Klassifikation der Musikstücke verwendet werden, da z.B. klassische Musikstücke oft viel länger sind als die modernen Stücke, zumal sie von den Radiostationen auf 3.30 Minuten beschränkt werden [46].

Die Länge eines Musikstückes entspricht der Anzahl n der *sample points* der gegebenen Wertereihe.

Mittlere Lautstärke: Die Lautstärke ist ein Merkmal, welches der Benutzer eindeutig spezifizieren kann und welches auch relativ leicht zu ermitteln ist. Da die Lautstärke über die Wertereihe hinweg variiert, müssen wir einen Wert finden, welches die Gesamtlautstärke gut genug repräsentiert. Das arithmetische Mittel bietet sich da an.

Die mittlere Lautstärke ist das arithmetische Mittel über alle Werte der Wertereihe.

Die Veränderlichkeit der Lautstärke: Manche Musikstücke sind ruhig, bei denen verändert sich die Lautstärke kaum, bei anderen schwankt der Lautstärkepegel hin und her. Beide können aber den gleichen Durchschnitt aufweisen.

Die Veränderlichkeit einer Wertereihe wird durch die Varianz repräsentiert. Sie ist unabhängig von den Schwankungen an den genauen Zeitpunkten.

Die absolute Lautstärke: Da die Elongationen des *sample* der Schwingung folgend abwechselnd positive und auch negative Werte annehmen, und die mittlere Lautstärke stets einen Wert nahe bei 0 hat, ist es sinnvoll die Beträge der Werte zu verwenden, um ein Bild von der „echten“ Lautstärke zu gewinnen.

Das Tempo: Das Tempo ist ein für den Benutzer vertrautes und leicht nachvollziehbares Merkmal und sagt eine ganze Menge über das Musikstück aus. Damit können z.B. schnelle und rhythmische Musikstücke leicht von den langsamen und behäbigeren unterschieden werden. Die Ermittlung des Tempos erweist sich als nicht ganz so einfach, aber Ingo Mierswa hat dafür ein Verfahren verwendet, welches auf der Phasenverschiebung der Wertereihen basiert und das tatsächliche Tempo eines Liedes zu 85% vorhersagen kann. Die genaue Vorgehensweise steht in [46].

Die Varianz der Autokorrelation: Genauso wie bei der Lautstärke würde uns die Veränderlichkeit des Tempos über die Zeit interessieren. Diese wird durch die Varianz über die das Tempo bestimmende Autokorrelation ermittelt.

Extremwertdifferenz: Innerhalb der Wertereihe kommen mehrere Extremwerte vor. Die Extremwertdifferenz beschreibt den durchschnittlichen Abstand der Extrema.

Bei diesem Merkmal ist es nun schwieriger, eine dem Benutzer verständliche Erklärung zu präsentieren. Es existieren z.B. Musikstücke, die gegen einen Höhepunkt streben, der meistens ziemlich laut ist. Wenn also der mittlere Abstand der Extrema sehr groß wird, kann es auf solch ein Musikstück hindeuten.

Die Varianz der Extremwertdifferenz: Genauso wie beim Tempo oder der Länge des Musikstückes repräsentiert die Varianz der Extremwertdifferenz die Veränderung der Distanzen der Extrema über die Wertereihe. Dies kann man sich so erklären, dass Musikstücke, die eine kleine Varianz aufweisen, regelmäßiger Extrema aufweisen, also einen geordneteren Ablauf haben als Musikstücke, deren Varianz ansteigt. Man könnte dieses Merkmal auch die *Experimentierfreudikeit* oder andersherum die *Schemabindung* nennen.

Nullstellendifferenz: Die Nullstellendifferenz stellt den mittleren Abstand zwischen zwei Durchquerungen der Nullstelle dar. Die Anwendung ist leicht zu ermitteln. Je größer der mittlere Abstand zwischen zwei Nullstellen ist, desto größere Wellen werden erzeugt und desto sanfter klingt das Musikstück. Je öfter die Nullstellen auftreten, desto schneller kommen die Wellen hintereinander und desto abgehackter wirkt das Musikstück. Man könnte dieses Merkmal auch die *Sanftheit* des Musikstückes nennen.

Die Varianz der Nullstellendifferenz: Genauso wie bei den vorherigen Merkmalen ist auch bei der Nullstellendifferenz die Veränderlichkeit von Interesse. Manche Musikstücke fangen fließend mit weiten Wellen an und wechseln aber den Rhythmus. In diesem Fall würde die Varianz steigen. Wenn die Musikstücke einen gleichmäßigen Wellenverlauf haben, dann ist auch die Varianz der Nullstellendifferenz klein.

Der Winkel im Zustandsraum: Bei diesem Merkmal versucht man neue Informationen zu erhalten, indem man die Wertereihe in einen neuen Raum, dem *Phasenraum*, transformiert. Dabei entstehen neue Vektoren, die für jeden *sample point* dessen Wert und dessen m Nachfolgewerte aufnimmt. m ist die Dimension des entstehenden Phasenraums. Dieses Vorgehen ist auch als *Zustandsraumrekonstruktion* bekannt. Es wird die Dynamik des Systems anhand der daraus resultierenden Bewegungskurve gemessen. Dynamische Musikstücke benutzen viele Dimensionen des Systems und zeigen auf einem Diagramm ein ziemlich zerstreutes Bild, während eher gleichmäßige Musikstücke, wie z.B. klassische sich in der Form immer mehr einer gleichmäßigen Ellipse angleichen.

Es ist also sinnvoll den durchschnittlichen Winkel zwischen den Vektoren in dem Phasenraum zu betrachten. Wenn der Winkel groß ist, dann haben wir eine runde Struktur, wenn sie aber klein sind, dann sind sehr viele Zacken vorhanden und wir schließen daraus, dass es sich um ein sehr dynamisches Musikstück handelt.

Die Varianz der Winkel im Zustandsraum: Auch die Varianz dieses Merkmals kann als ein weiteres Merkmal angesehen werden. Wenn die Varianz klein ist, dann sind die Winkel alle ungefähr gleich groß. Das bedeutet, dass die Dynamik des Musikstückes sich sehr wenig verändert. Wenn die Varianz groß ist, dann verändert sich die Dynamik ständig und es ist ein sehr wechselhaftes Musikstück.

Die Abstände im Zustandsraum: Die Länge der Teilstücke zwischen den Punkten im Phasenraum schwankt unterschiedlich stark. Deswegen ist es sinnvoll deren Durchschnitt als ein weiteres Merkmal zu verwenden. Es lässt sich aber nicht so gut für den Benutzer interpretieren, wie die anderen Merkmale.

Die Varianz der Abstände im Zustandsraum: Da sich die Varianz eines durchschnittlichen Merkmals zu betrachten bisher als sinnvoll erwiesen hat, wird auch die Veränderlichkeit der Abstände im Zustandsraum als ein weiteres Merkmal verwendet.

1. Peak: Frequenz: Für dieses Merkmal betracht man die Extremstellen des *Frequenzspektrums* eines Musikstückes. Ein Frequenzspektrum sind alle Frequenzen und deren, in dem Musikstück auftretenden, Intensitäten. Es wird also nach den intensivsten Frequenzen gesucht.

Wir betrachten für die größte Extremstelle deren Stelle, Höhe und Breite. Diese Information wird auch *Peak* genannt.

Dieses Merkmal ist die Stelle, also die Frequenz des höchsten Peaks.

Die Interpretation dieses Merkmals gelingt einfach. Bassreiche Musikstücke haben in den tiefen Frequenzen eine hohe Intensität, während die Musikstücke, bei denen viele Streichinstrumente vorkommen, eher in den hohen Frequenzen eine große Intensität haben.

1. Peak: Höhe: Dieses Merkmal ist die Höhe, also die Amplitude, des höchsten Peaks. Die Intensität der stärksten Frequenz eines Musikstückes kann zu der Intensität der stärksten Frequenz eines anderen Stückes sich enorm unterscheiden.

1. Peak: Breite: Dieses Merkmal ist die Breite des höchsten Peaks.

Der höchste Peak kann über mehrere Frequenzen gehen. Je mehr Frequenzen er einnimmt, desto voller ist das Musikstück.

2.Peak: Frequenz: Dieses Merkmal ist die mittlere Frequenz der zwei größten Peaks.

Es ist sinnvoll, mehrere Peaks zu betrachten, da es zwar intensive Frequenzen geben kann, die aber nicht das gesamte Musikstück bestimmen, bzw. nur an wenigen Stellen, z. B. beim Intro oder bei einem Solo zwischen den Strophen, auftreten können. Im Folgenden werden dieser Idee folgend bis zu fünf Peaks betrachtet.

2. Peak: Höhe: Dieses Merkmal ist die durchschnittliche Höhe der beiden höchsten Peaks.

2.Peak: Breite: Dieses Merkmal ist die durchschnittliche Breite der beiden höchsten Peaks.

3.Peak: Frequenz: Dieses Merkmal ist die mittlere Frequenz der drei größten Peaks.

3. Peak: Höhe: Dieses Merkmal ist die durchschnittliche Höhe der drei höchsten Peaks.

3.Peak: Breite: Dieses Merkmal ist die durchschnittliche Breite der drei höchsten Peaks.

4.Peak: Frequenz: Dieses Merkmal ist die mittlere Frequenz der vier größten Peaks.

4. Peak: Höhe: Dieses Merkmal ist die durchschnittliche Höhe der vier höchsten Peaks.

4.Peak: Breite: Dieses Merkmal ist die durchschnittliche Breite der vier höchsten Peaks.

5.Peak: Frequenz: Dieses Merkmal ist die mittlere Frequenz der fünf größten Peaks.

5. Peak: Höhe: Dieses Merkmal ist die durchschnittliche Höhe der fünf höchsten Peaks.

5.Peak: Breite: Dieses Merkmal ist die durchschnittliche Breite der fünf höchsten Peaks.

Die Steigung des Spektrums: Dieses Merkmal basiert auf der Steigung einer Ausgleichsgeraden, auch *Regressionsgerade* genannt, die durch die Punktmenge in dem Frequenzspektrum so gelegt wird, dass sie diese Punkte so gut wie möglich beschreibt.

Interpretiert kann dieses Merkmal so, dass, wenn die Steigung der Ausgleichsgerade positiv ist, die höheren Frequenzen stärker verwendet werden, während die negative Steigung der Ausgleichsgerade darauf hindeutet, dass die niedrigeren Frequenzen eher bevorzugt werden. Schwankt die Steigung allerdings um die Null herum, dann wird der gesamte Frequenzspektrum gleichmäßig verwendet.

y-Achsenabschnitt des Spektrums: Dieses Merkmal bestimmt den y-Achsenabschnitt der Regressionsgeraden.

Die Diskrepanz des Spektrums: Dieses Merkmal bestimmt die mittlere Abweichung (*Diskrepanz*) der Werte von der berechneten Regressionsgeraden. Es drückt die Gleichmäßigkeit aus, mit der die Frequenzen verwendet werden.

Verhältnis Maximum/Arithmetisches Mittel: Dieses Merkmal berechnet, wie der Name schon sagt, das Verhältnis zwischen dem maximalen Wert einer Reihe und deren arithmetischen Mittel. Es wird auch als *Spectral Crest Factor* (SCF) bezeichnet.

Gefensterte maximale Frequenz: Für dieses Merkmal wird ein Zeitfenster über das Frequenzspektrum geschoben und dabei jeweils die intensivste Frequenz berechnet. Dadurch erhält man Informationen bezüglich der zeitlichen Ordnung. Die maximalen Werte über alle Fenster hinweg bilden eine neue Wertereihe.

Es wird der Durchschnitt über alle Maxima ermittelt.

Varianz der gefensterten maximalen Frequenz: Dieses Merkmal bestimmt die Varianz der maximalen Frequenzen bei der Fensterung. Es wird die Veränderlichkeit der maximalen Intensitäten der Frequenzen gemessen.

1. Frequenzband: Startwert: Es werden in dem Frequenzband Bereiche gekennzeichnet, die innerhalb einer Dimension ähnliche Eigenschaften aufweisen. Das können z.B. Frequenzen sein, die ein Musikstück bestimmen. Sie werden *Intervalle* genannt. Die Bestimmung der Intervalle ist in [46] näher erläutert.

Dieses Merkmal repräsentiert die linke Intervallgrenze des größten Intervalls.

1. Frequenzband: Endwert: Dieses Merkmal repräsentiert die rechte Intervallgrenze des größten Intervalls.

1. Frequenzband: Dichte: Dieses Merkmal repräsentiert die Dichte ρ, also die Anzahl der Elemente in dem größten Intervall.

2. Frequenzband: Startwert: Wir betrachten das zweitgrößte Intervall. Dieses Merkmal repräsentiert dessen Startwert, also die linke Intervallgrenze.

2. Frequenzband: Endwert: Dieses Merkmal repräsentiert die rechte Intervallgrenze des zweitgrößten Intervalls.

2. Frequenzband: Dichte: Dieses Merkmal repräsentiert die Dichte ρ, also die Anzahl der Elemente in dem zweitgrößten Intervall.

3.2 Erweiterung von Merkmalen

Eine allgemeine Erweiterung von Merkmalen erscheint zunächst nicht sinnvoll, denn je mehr Merkmale man hat, desto höhere Dimensionalität besitzt unser Raum und desto ineffizienter werden unsere Lösungsverfahren.

Allerdings spricht ebenfalls einiges für eine Erweiterung der Merkmalsmenge. Die Güte des gesamten Ansatzes der Constraints basiert auf der Verständlichkeit der Constraints für den Benutzer. Da aber, z.B. die Constraints 2 und 3 (Kap. 4.4.2 und 4.4.3), die genaue Manipulation einzelner Merkmale erfordern, ist die Verständlichkeit der Merkmale von großer Bedeutung.

Leider sind die meisten oben vorgestellten Merkmale weit davon entfernt für den Benutzer vollkommen verständlich zu sein, bzw. viele Benutzer können mit solchen Informationen wie *Tempo* oder *Lautstärke* nur indirekt etwas anfangen. Sie wissen nur, ob ihr Musikstück *langsam* oder *laut* sein soll, aber mit den genauen Wertemengen der Merkmale kennen sie sich nicht aus. Z.B. ist es für einen Benutzer nicht klar, wann der Übergang zwischen langsamen und schnellen Musikstücken ansetzen soll.

Durch diese und viele andere Gründe erscheint es sinnvoll, *Metainformationen* in die Merkmalsmenge aufzunehmen.

DEFINITION 11: (Metainformationen)
Metainformationen *sind alle Informationen, die zu dem Musikstück gehören, nicht aber aus den Klangdaten abgeleitet werden können. Die Klangdaten sind die digitalisierte Wellenform des Musikstückes.*
Metamerkmale *sind Merkmale, die aus den Metainformationen gebildet worden sind. Dabei sind sie entweder äquivalent den Metainformationen, oder sie stellen eine Teilmenge der Wertemenge der einzelnen Metainformationen dar.*

Als Beispiele wären dafür zu nennen: das Erscheinungsjahr, der Interpret, der Titel, das Album, die Tracknummer, das Genre etc.

3.2.1 ID3-Tags

Die Metainformationen können bei vielen Formaten wie MPEG Layer 3 oder kurz MP3 in den Musikstücken mitgespeichert werden und stehen dem Benutzer und unserer Auswertung direkt zur Verfügung. Im MP3 Format werden diese Informationen *ID3-Tags* genannt. Mittlerweile wurden neuere Versionen der Tags entwickelt[1], die nicht mehr auf wenige Bytes mit festgelegten Datentypen beschränkt sind, sondern beliebige Informationen wie das Bild des Interpreten oder des Albums und den Songtext enthalten können. Viele aktuelle Betriebssysteme[2] und MP3-Player[3] bieten die Möglichkeit, direkt die Tags zu betrachten und zu manipulieren.

Nicht alle Metainformationen ergeben für den Zweck der Gruppierung einen Sinn. Die Titel der Musikstücke sind ein Beispiel dafür. Diese sind für jedes Musikstück einzigartig und dadurch für das Clustering wertlos. Andere Tags, wie z.B. die Sprache des Musikstückes, kommen nur in bestimmten Tagversionen vor und können nicht wegen der Allgemeinheit unseres Ansatzes in Betracht gezogen werden.

3.2.2 Probleme der erweiterten Merkmalsmenge

Ein erstes Problem mit dieser Art von Merkmalen ist, dass sie oft nicht direkt aus dem Musikstück zu ermitteln sind. Selten ist es so, dass die z.B. in den `.mp3` Dateien gespeicherten Musikstücke alle erwünschte Informationen enthalten. Oft erhält man beim Kopieren eines Albums von der CD die folgende Liste:

```
Track1.mp3
Track2.mp3
...
TrackN.mp3
```

Man kann noch nicht einmal darauf vertrauen, dass die Musikstücke ihre Originaltitel erhalten. Es hindert auch niemand den Benutzer daran, die Titel nach seinem Belieben zu verändern, z.B. in `Mein_Lieblingslied.mp3`.

Eine Lösung aus diesem Dilemma bietet das Internet. Die aktuellere Kopiersoftware bietet die Möglichkeit, direkt beim Kopiervorgang alle Informationen über die Musikstücke aus dem Internet herunterzuladen und als Metainformation in den Musikstücken abzuspeichern. Auch im Nachhinein ist es möglich die Metainformationen zu erhalten, denn es existieren spezielle Dienste, die für eine komplette CD aufgrund der Tracklänge die Metainformationen der Musikstücke ermitteln können. Leider ist diese Möglichkeit immer noch mit einem erheblichen Aufwand verbunden, so dass dies die Laufzeit schmerzlich beeinflussen würde.

[1] Die verbreitesteten sind ID3 v1, ID3 v2.0 – ID3 v2.4 und Lyrics3. Mehr Informationen zu den Tags sind unter `www.id3.org` zu finden.
[2] Z.B. Windows, Linux, Mac OS
[3] Z.B. Windows Media Player, XMMS bei Linux, i-Tunes bei Mac

Eine weitere Lösung ist einfach davon auszugehen, dass in den Musikstücken, die betrachtet werden, die Metainformationen, die von dem System als Merkmale verwendet werden, von vorne hinein enthalten sind. Es ist eine Notlösung, allerdings könnte man den Benutzer darauf hinweisen, dass er, wenn er die Metamerkmale verwenden möchte, dafür Sorge tragen müsse, dass diese auch dort enthalten sind.

Es bringt nichts die Metamerkmale als „nicht angegeben“ o.ä. zu markieren, denn dann können die Musikstücke mit diesen Merkmalen beliebig eingeordnet werden, was in den seltensten Fällen im Sinne des Benutzers wäre. Wenn, z.B. der Interpret für ein Musikstück nicht angegeben wäre, der Benutzer aber genau den Interpreten dieses Musikstückes in den Constraints spezifiziert hat, so könnte dieses Musikstück in einem falschen Cluster auftauchen, und das System würde es noch nicht einmal bemerken und felsenfest behaupten, dass das von dem Benutzer angegebene Constraint vollständig erfüllt sei.

Ein anderes Problem ist die Wertemenge der Metainformationen. Bis auf das Erscheinungsjahr sind es alles *qualitative* Merkmale, deren Daten noch nicht einmal eine Ordnung untereinander aufweisen. Als Beispiel wäre dazu der Interpret des Musikstückes zu nennen. In dieser Arbeit werden wir aber ein System aufbauen, welches auf *quantitativen* Merkmale basiert. Sowohl das Distanzmaß (Kap. 6.3), als auch die verschiedenen Optimierungsmethoden (Kap. 6.7) können nichts mit den meisten Metainformationen anfangen.

Eine Lösung aus diesem Dilemma wäre, vollends auf die Metainformationen zu verzichten, was für den Benutzer ein großer Verlust an Ausdrucksmöglichkeit wäre.

Eine weitere Lösung wäre es, nur die Metamerkmale aufzunehmen, die quantitativ sind. Leider gibt es bis auf das Erscheinungsjahr kaum solche Merkmale, so dass der Ertrag dieser Vorgehensweise recht arm ausfallen dürfte.

Die dritte Möglichkeit wäre es, für ein ausgewähltes Merkmal, wie z.B. das Genre, eine quantitative Darstellung zu erzeugen. Die einzelnen Genrearten würden dann ein binäres Attribut erhalten, also z.B. Rock, Pop, Volksmusik, etc. Das Problem ist, dass wir uns dabei nur auf eine Auswahl aller möglichen Attribute beschränken müssen, um die Dimension der Merkmale nicht zu sehr zu erhöhen.
Ein binäres Attribut für ein Metamerkmal erhalten wir, indem wir ein qualitatives Element der Wertemenge des Merkmals herausgreifen und ihn zu einem neuen Merkmal erklären. Dieser erhält die Wertemenge $\{0, 1\}$, 1, wenn das Attribut bei diesem Lied vorkommt, 0, wenn nicht. Dadurch erhalten wir ein quantitatives Attribut, welches in den Verfahren wie k-Means (Kap. 2.6.1) verwendet werden kann.

Letztendlich ist es also sinnvoll, den Benutzer auch in diesen Prozess einzubeziehen. Wir gehen davon aus, dass die Metainformationen, an denen der Benutzer interessiert ist, auch tatsächlich für jedes Musikstück zur Verfügung stehen. Er kann, bevor die Iteration startet, angeben, welche Metamerkmale und welche Werte dieser Merkmale er haben möchte. Z.B. gibt er für den Interpreten: „Heino“, „Aerosmith“ und „Andere“ ein und erhält dadurch drei neue binäre Merkmale.

3.2.3 Einbindung von ID3 Tags in unser System

Die neuen Metamerkmale müssen in unserem System vor allen anderen Schritten bestimmt werden, denn wir können ansonsten aufgrund der Veränderung der Distanzen[4] nicht mit konsistenten Ergebnissen rechnen.

Wir beschränken uns auf die geläufigsten ID3-Tags *Erscheinungsjahr*, *Interpret*, *Album* und *Genre*. Das einzige quantitative Merkmal ist das Erscheinungsjahr. Für die anderen müssen wir durch alle Musikstücke durchgehen und alle vorhandenen Interpreten, Alben und Genres sammeln. Anschließend muss der Benutzer spezifizieren, welche von diesen er als neue binäre Merkmale haben möchte. Erst danach werden die ausgewählten Merkmale in die Merkmalsmenge eingefügt.

[4] Siehe Kap. 6.3

4 Constraints

In dem Kapitel 2.6.3 wurden Constraints als Bedingungen, die ein Benutzer spezifizieren kann, um einen Clusteringalgorithmus dazu zu bewegen, nach seinen Vorstellungen zu clustern, definiert. Auch wurden einige Constraintsarten vorgestellt. In diesem Kapitel wollen wir die in dieser Arbeit verwendeteten Constraints aufstellen und begründen.

Die Constraints ermöglichen uns, die Einteilung der Musikstücke in die Cluster zu bewerten.

Sie sind Abbildungen der Wünsche des Benutzers auf eine, einer Maschine verständlichen, Ebene.

Die Constraints werden einzeln gewichtet. Die Gewichte, die sich in dem Intervall $[0, 1]$ befinden, beschreiben den Grad der Wichtigkeit (engl. *importance*) der Constraints.

Einige Constraints für ein allgemeines Clustering wurden in [54], vorgeschlagen. In modifizierter Form erscheinen sie als eine Vorgabe für das Clustering von Musikdaten durchaus sinnvoll.

4.1 Kriterien für die Constraints

Da die Constraints die Schnittstelle zwischen dem Benutzer und dem Clusteringalgorithmus darstellen, ist ihre Spezifikation und Auswertung kritisch für die Funktionalität des ganzen Systems. Es erweist sich als sinnvoll Bedingungen aufzustellen, die *wohldefinierte* Constraints erfüllen müssen, wobei evtl. eine Bedingung einer anderen widersprechen kann:

1. **Verständlichkeit.** Die Constraints müssen für einen Benutzer *klar verständlich* sein. Das bedeutet, dass der Benutzer genau wissen muss, was es für Auswirkungen hat, wenn er dieses Constraint manipuliert.

2. **Auswertungsgüte.** Die Constraints müssen *gut auswertbar* sein. Das bedeutet, dass es eine eindeutige Abbildung von der Benutzersicht auf die Rechnersicht geben muss, so dass maschinell die Absicht des Benutzers verstanden wird.

3. **Auswertungseffizienz.** Die Constraints müssen *schnell auswertbar* sein.

Die Bedingung 3 ist insofern kritisch, da man bei einigen Optimierungsalgorithmen, die in dieser Arbeit für die Erfüllung der Clonstraints verwendet werden, wie z.B. dem Evolutionären Algorithmus (Kap. 6.7.6), viele Lösungen erstellt und jeweils auswerten muss.

4.2 Auswertung der Constraints

Nachdem der Benutzer einzelne Constraints spezifiziert hat, startet ein Algorithmus, welcher so weit wie möglich die Constraints zu erfüllen versucht. Wenn er fertig ist, muss maschinell überprüft werden können, inwieweit die Constraints erfüllt worden sind.

Dazu wird in [54] die folgende globale Constraints-Erfüllungs-Gleichung spezifiziert:

$$\Theta(C) = \frac{1}{\sum_{i=1}^{T} q_i} \sum_{i=1}^{T} q_i * \theta_i(C) \tag{4.1}$$

Wir haben $\{q_1 \dots q_T\}$, die Menge aller Gewichte der Constraints. Der Benutzer kann mit ihnen angeben, wie wichtig ihm die Erfüllung dieses Constraints ist. Der Wert 1 bedeutet, dass sie sehr wichtig, der Wert 0 bedeutet, dass sie überhaupt nicht wichtig ist.

$\{\theta_1 \dots \theta_T\}$ ist die Menge der Erfüllungsfunktionen für die einzelnen Constraintstypen. Alle $\theta_i, i = 1, \dots, T$ sind in dem Intervall $[0, 1]$, welcher den Grad beschreibt, zu dem das jeweilige Constraint erfüllt ist.

C ist das aktuelle *Clustering*, also eine Datenstruktur aus allen Musikstücken zugewiesen zu ihren Clustern. Die globale Constraintserfüllungsfunktion $\Theta(C)$ ist in dem Intervall $[0, 1]$ und beschreibt den Grad zu dem alle Constraints durch das Clustering C erfüllt worden sind.

Die einzelnen θ_i werden in den nachfolgenden Abschnitten bei den zugehörigen Constraints beschrieben. Sie unterscheiden sich zum Teil sehr stark von der Vorgabe in [54], weil sie entweder erweitert oder modifiziert wurden, um entweder mehr Anwendungsfälle abzudecken, oder um die Auswertung der Constraints zu beschleunigen. Z.B. ist der Wertebereich der einzelnen Merkmale der Musikdaten kontinuierlich und nicht, wie in [54] angenommen wird, diskret, was eine andere Auswertung erfordert. Als Beispiel für die Beschleunigung, wäre das Constraint vom Typ 7, Cluster Adherence (Kap. 4.4.7), zu nennen, bei dem eine komplexe Rechnung durch einen einfachen Minimum-Operator ersetzt wurde, aufgrund der Aussage aus [54], dass dieses Constraint das *logische UND* zwischen den Constraints vom Typ 2 und 3 (Kap. 4.4.2 und 4.4.3) darstellt.

4.3 Erfüllbarkeit der Constraints

Es stellt sich die Frage, ob Constraints immer vollständig erfüllt sein können. Dieser Fall muss nicht eintreten, denn es sind auch widersprüchliche Constraints denkbar, so dass die Erfüllung des einen Constraints die Erfüllung des anderen behindert, bzw. unmöglich macht.

Ein Beispiel dafür wäre, dass man die Anzahl der möglichen Cluster (Constraint vom Typ 4, Kap. 4.4.4) ziemlich hoch setzt, während man gleichzeitig auch die minimale Anzahl der Musikstücke, die einem Cluster zugewiesen werden sollen (Constraint vom Typ 5, Kap. 4.4.5) enorm groß wählt. Beides kann nicht gleichzeitig vollkommen erfüllt werden und das System wird versuchen die echten Werte den Vorgaben des Systems so weit wie möglich anzunähern, diese also zu optimieren.

4.4 Die Constraints

4.4.1 Constraint Typ 1: Instance Association

Bei diesem Constrainttyp gibt der Benutzer für eine Auswahl an Musikstücken an, welche in ein Cluster kommen und welche auf keinen Fall zusammen in einem Cluster erscheinen sollen. Dieses Constraint entspricht den schon in dem Kapitel 2.6.3 vorgestellten Constraints *Must-Link* und *Cannot-Link*.

Wie ist ein solches zusammengesetztes Constraint technisch zu realisieren? Der Benutzer weist einigen Musikstücken ein Cluster zu. Das bedeutet, diese Stücke gehören zusammen. Wenn er nun ein paar andere Musikstücke einem anderen Cluster zuweist, bedeutet das, dass die Stücke, die zu diesem Cluster zugewiesen wurden, zusammengehören (must-links), und dass die Stücke, die anderen Clustern zugewiesen wurden, auf keinen Fall zusammengehören (cannot-links). Dadurch können mehr als zwei Musikstücke auf einmal als zusammengehörend, bzw. nicht zusammengehörend deklariert werden. Der einzige Fall, der durch diese Realisierung nicht abgedeckt werden kann, ist der, dass man zwei jeweils zusammengehörende Mengen an Musikstücken definiert, für welche jedoch egal ist, ob sie dem gleichen Cluster zugewiesen werden oder verschiedenen. In diesem Fall ist der Benutzer gezwungen sich für eine der beiden Möglichkeiten zu entscheiden: Entweder werden aus den beiden zusammengehörenden Mengen eine einzige oder es werden zwei nicht zusammengehörende Mengen. Diese Vorgehensweise wird im Kapitel 6.5 genauer erläutert.

Die Auswertungsformel

In [54] wird zunächst die folgende Funktion definiert:

$$\sigma(x,a,b,c) = \begin{cases} 0, & x < 1 \\ \frac{x-1}{a-1}, & 1 \leq x < a \\ 1, & a \leq x < b \\ \frac{b-x}{c-b}, & b \leq x \leq c \\ 0, & x > c \end{cases} \tag{4.2}$$

Mit dieser Funktion können wir die Constraints auf der Grundlage der Fuzzy-Logik definieren und hoffen dadurch eine effiziente Auswertung zu bekommen. In dieser Art von Logik geht man nicht mehr von festen Werten, z.B. 0 oder 1, aus, sondern von *Zugehörigkeitsfunktionen*, die Fuzzy-Zahlen repräsentieren und verschiedene Formen annehmen können. Die Form der σ-Funktion ist die $Trapez$-Form einer Fuzzy-Zahl. Weitere Informationen bzgl. Fuzzy-Mengen und Fuzzy-Zahlen sind in [81] zu finden. Der Verlauf der Funktion ist in der Abbildung 4.1 dargestellt.

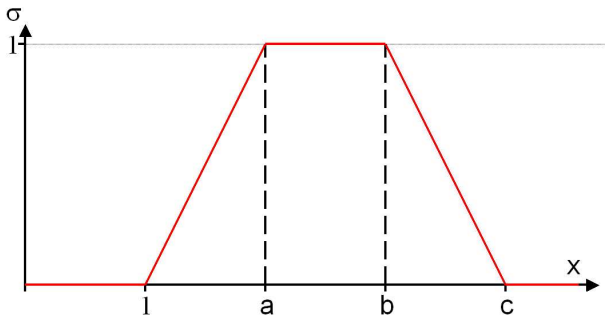


Abbildung 4.1: **Der Verlauf der σ-Funktion.** Bis zu dem Wert 1 hat sie den Funktionswert 0. Dann steigt dieser linear an, bis an der Stelle a das Maximum bei 1 erreicht wird. Dieser Wert wird bis zu der Stelle b beibehalten. Danach sinkt er linear bis zu der Stelle c, an der und an allen folgenden Stellen eine 0 angenommen wird.

Die Variable n' ist Anzahl aller Paare von Must-Links und Cannot-Links, die zu irgend einem Cluster zugehören müssen, m, ist die Anzahl aller Paare von Must-Links und Cannot-Links, die erfüllt sind, also beide Elemente eines Must-Links sich tatsächlich im gleichen Cluster, und beide Elemente des Cannot-Links sich in verschiedenen Clustern befinden.

Die Formel lautet dann:

$$\theta_1(C) = \sigma\left(1 + \frac{m}{n'}, 2, 2, 2\right) = \frac{m}{n'} \tag{4.3}$$

$\theta_1(C)$ ist genau dann 1, wenn $m = n'$, also wenn alle Paare der must-links und cannot-links sich in den richtigen Clustern befinden. Umgekehrt ist $\theta_1(C) = 0$, wenn keines der ausgewählten Musikstücke sich in dem Cluster befindet, dem der Benutzer es zugewiesen hat.

Wir wollen überprüfen, inwiefern dieses Constraint die in Kap. 4.1 definierten Kriterien erfüllt.

1. Verständlichkeit für den Benutzer Dieses Constraint ist das intuitivste Constraint, denn bei diesem kann der Benutzer direkt anhand von Beispielen seine gewünschte Aufteilung in die Cluster anzeigen.

2. Güte der Auswertung Leider haben wir eine Auswertung einer mäßigen Güte zu erwarten, denn dadurch, dass der Benutzer nur für eine Auswahl angibt, welche Musikstücke zusammengehören und welche nicht, wissen wir nichts über seine Präferenz bezüglich der anderen Musikstücke. Wir

müssen uns auf das Ähnlichkeitsmaß verlassen, welches allerdings ein, für den Benutzer nicht immer richtiges, Kriterium ist für die Zusammengehörigkeit von zwei Musikstücken. Es ist also der in der Einleitung erwähnte *tradeoff* zwischen der vollkommenen Kontrolle des Benutzers und der Eigenständigkeit der Maschine.

Vor allen Dingen kann ein einfacher Algorithmus dieses Constraint immer vollkommen erfüllen. Dieser Algorithmus nimmt alle Musikstücke, die von dem Benutzer ausgewählt wurden, und weist sie den, von dem Benutzer für sie vorgesehenen, Clustern zu. Schon ist dieses Constraint erfüllt. Der Algorithmus hat aber nichts geleistet, denn diese Aufteilung hat ja der Benutzer schon selbst vorgenommen.

3. Effizienz n' zu berechnen ist einfach: man zählt die must-links und cannot-links in dem Constraint. Man kann auch die Datenstruktur der Zuweisungen soweit modifizieren, dass man diese Zahl mitspeichert.

m zu berechnen, erfordert, dass man alle must-links und cannot-links durchgeht und überprüft, wieviele von denen erfüllt sind. Wenn die Anzahl der must-links und cannot-links g ist, dann kann die Laufzeit für die Berechnung der Formel 4.3 durch $\mathcal{O}(g)$ abgeschätzt werden.

4.4.2 Constraint Typ 2: Value Association

Bei diesem Constrainttyp gibt der Benutzer für einen Wert, bzw. ein Intervall von Werten eines Merkmals A_h an, dass alle Instanzen, die für dieses Merkmal diese Werte enthalten, möglichst dem gleichen Cluster zugewiesen werden sollten. Dieses und das nachfolgende Constraint entsprechen den im Kapitel 2.6.3 aufgeführten *Parameterconstraints*.

Dieses Constraint unterscheidet sich von dem in [54] dadurch, dass nun mehrere Merkmale auswählbar sind, bei denen auch mehrere Werte, bzw. Intervalle ausgewählt werden können. Auf der einen Seite führt dies eindeutig zu einer Erschwerung der Auswertung dieses Constraints, auf der anderen Seite erhöht es seine Einsatzmöglichkeiten.

Die Auswertungsformel

Wenn man auf die Sichtweise der Statistik übergeht, bieten sich einem etablierte, gut begründete und weit verbreitete Methoden für die Aufstellung der Auswertungsformel für dieses Constraint, an. Die Rede ist von den *Konzentrationsmaßen*. Diese sind umso größer, je konzentrierter eine Verteilung ist.

Wir betrachten in jedem Schritt nur einen von dem Benutzer ausgewählten Wert für nur ein Merkmal. Uns interessiert die Verteilung der Kardinalitäten der Musikstücke für den ausgewählten Wert für das ausgewählte Merkmal über die Cluster.

Die Abbildung 4.2 veranschaulicht diese Argumentation.

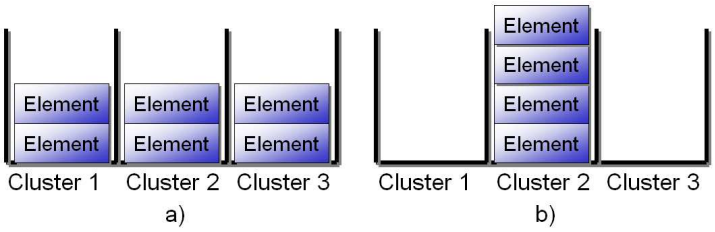


Abbildung 4.2: **Die Verteilung der Elemente in den Clustern bzgl. des ausgewählten Wertes.** a) zeigt die schlechteste Verteilung, b) die beste Verteilung

Die Statistik stellt mehrere Konzentrationsmaße bereit, die alle ihre Vor- und Nachteile haben. Der gebräuchlichste Konzentrationskoeffizient ist der *Gini-Koeffizient*[1] G. Dieser verwendet die Werte der

[1] Benannt nach einer 1910 erschienenen Arbeit von C.Gini

Lorenzkurve[2], die den Anteil an der gesamten Merkmalssumme, den die k kleinsten[3] Merkmalsträger auf sich vereinigen, beschreibt (Siehe dazu auch [6]). G ist folgendermaßen definiert:

$$G = \frac{\text{Fläche zwischen der Diagonalen D und der Lorenzkurve L}}{\text{Fläche zwischen der Diagonalen D und der x-Achse}} \tag{4.4}$$

Und die Formel dafür lautet:

$$G^4 = \frac{2\sum_{i=1}^{k} i \cdot \tau_j^{(i)}(A_h) - (k+1)\sum_{i=1}^{k} \tau_j^{(i)}(A_h)}{(k-1)\sum_{i=1}^{k} \tau_j^{(i)}(A_h)} \tag{4.5}$$

Dabei ist $\tau_j^{(i)}(A_h)$ die Anzahl der Musikstücke, die sich in dem i-ten Cluster befinden und für das ausgewählte Merkmal A_h den ausgewählten Wert v_j haben. Der Parameter k ist die Anzahl der Cluster.

Allerdings treffen wir auf das Problem, dass die Anzahl der von dem Benutzer ausgewählten Werte unendlich sein kann, z.B. wenn der Benutzer ein Intervall angibt. Deswegen ist es sinnvoll, vorher alle ausgewählten Intervalle in eine Reihe von diskreten Werten zu verwandeln, und zwar in nur solche, für die es mindestens ein Musikstück gibt, welches für das ausgewählte Merkmal diesen Wert annimmt.

Die gesamte Vorgehensweise bei der Auswertung dieses Constraints ist in dem Algorithmus 1 aufgezeigt.

Algorithmus 1: Auswertung des Constraints vom Typ 2

```
for jedes ausgewählte Merkmal A_h do
  for jeden ausgewählten Wert v_j do
    if v_j ist ein Intervall then
      for jedes Musikstück do
        if Wert v* des Merkmals A_h in dem Intervall then
          if v* nicht schon als Einzelwert in der Menge der ausgewählten Werte enhalten then
            Füge v* zu der Menge der ausgewählten Werte hinzu
          end if
        end if
      end for
      Brich die Ausführung dieses Iterationsschritts ab
    end if
    Berechne G
  end for
end for
Berechne den Mittelwert über alle G mit der neuen Anzahl der Werte
```

$\theta_2(C)$ ist 1, wenn alle Gini-Koeffizienten für alle Werte eine 1 liefern. Dies ist nur möglich, wenn alle Werte v_j sich auf eine minimale Anzahl von Clustern konzentrieren. Wenn das Gegenteil eintritt und alle Gini-Koeffizienten eine 0 liefern, so ist jedes der ausgewählten Werte gleichmäßig verteilt in den Clustern, so dass wir auch für $\theta_2(C)$ eine 0 erhalten.

Wir wollen überprüfen, inwiefern dieses Constraint die in Kap. 4.1 definierten Kriterien erfüllt.

1. Verständlichkeit für den Benutzer Auf den ersten Blick erscheint dieses Constraint einem Benutzer nichtssagend, doch, wenn man Intervalle betrachtet und angeben will, dass, z.B. alle Musikstücke, deren Tempo in einem bestimmten Bereich liegen, auf jeden Fall zusammen gehören,

[2] Benannt nach M.O. Lorenz, der diese Kurve 1904 zur Messung der Vermögenskonzentration verwandte.

[3] In unserem Fall bedeutet „kleinster“ das Cluster, welches am wenigsten Musikstücke enthält, die für das ausgewählte Merkmal den ausgewählten Wert haben.

[4] Hier wird der *normierte Gini-Koeffizient* G^* verwendet, $G^* = \frac{n}{n-1}G$, um zu verhindern, dass der Maximalwert des Gini-Koeffizienten negativ wird

so ergibt dieses Constraint plötzlich einen neuen Sinn und eröffnet dem Benutzer neue Möglichkeiten, seine Wünsche zu äußern. Wenn man nun die Merkmalsmenge insofern erweitert, dass zusätzlich auch Metadaten eines Musikstückes betrachtet werden, so wird dieses Constraint für den Benutzer unverzichtbar. So kann er, z.B. angeben, dass die Musikstücke, die älter sind als 1960 in das selbe Cluster kommen sollen. Diese Erweiterung der Merkmalsmenge wird ausführlich in 3.2 diskutiert.

2. Güte der Auswertung Hier haben wir eine etwas bessere Abschätzung als beim Constraint Typ 1. Zwar können wir auch hier nur das überprüfen, was der Benutzer eingestellt hat, aber diesmal überprüfen wir damit die Erfüllung für *alle* Musikstücke, die für ein eingegebenes Merkmal den bestimmten Wert haben. Leider können wir hier nicht sagen, was mit den Musikstücken, die für bestimmte Merkmale andere Werte haben, geschehen soll. Dazu brauchen wir die Constraints vom Typ 3 und 7 (Kap. 4.4.3 und 4.4.7).

3. Effizienz Die Laufzeit dieses Konstrukts kann folgendermaßen ermittelt werden. Die Anzahl der ausgewählten Werte sei V^*. Da wir nur n Musikstücke haben, kommen pro Merkmal nur n neue Werte hinzu. Die Anzahl H der Merkmale ist aber konstant. Im worst case sind es also $V^* + H * n$ Elemente, die untersucht werden müssen. Der Test, ob die neuen Werte nicht schon in den alten enthalten sind, kann mit Hilfe eines Hashsets auf $\mathcal{O}(1)$ reduziert werden. Für die Berechnung von G braucht man eine Laufzeit von $\mathcal{O}(k * n)$. Insgesamt erhalten wir also eine Laufzeit von $\mathcal{O}(k * V^* * n)$.

4.4.3 Constraint Typ 3: Value Separation

Bei diesem Constrainttyp wählt der Benutzer Merkmale aus, deren Wertemengen sich gleichmäßig über alle Cluster verteilen sollen.

Auswertungsformel

Genauso wie in dem Constraint vom Typ 2 (Kap. 4.4.2) haben wir bei diesem Constrainttyp das Problem, das der Benutzer in diesem Fall mehrere Attribute auswählen kann. Wir müssen also die in [54] vorgeschlagene Formel leicht modifizieren:

$$\theta_3(C) = \frac{1}{H^* \cdot k} \sum_{h=1}^{H^*} \sum_{i=1}^{k} 1 - 2\frac{VAR(\vec{\tau}^{(i)}(A_h))}{R(\vec{\tau}^{(i)}(A_h))} \tag{4.6}$$

H^* ist die Anzahl der ausgewählten Merkmale. $\vec{\tau}^{(i)}(A_h)$ ist ein Vektor, der für alle Werte, die das Attribut A_h über alle Musikstücke in dem Cluster i annimmt, die Kardinalitäten dieser Werte enthält. Die Funktion VAR ist die normalisierte Standardabweichung über alle Werte von $\vec{\tau}^{(i)}(A_h)$, und die Funktion R ist die Differenz zwischen dem maximalen und dem minimalen Wert des Merkmals A_h inerhalb des Clusters i.

Die Funktion VAR wird folgendermaßen berechnet:

$$VAR(\vec{\tau}^{(i)}(A_h)) = \frac{1}{\sum_{j=1}^{L_h} \tau_j(A_h)} * \sqrt{\sum_{j=1}^{L_h} [\tau_j(A_h)(v_j - E(\vec{\tau}^{(i)}(A_h)))^2]} \tag{4.7}$$

Dabei ist $\tau_j(A_h)$ der j-te Wert aus dem Vektor $\vec{\tau}^{(i)}(A_h)$, und L_h ist die Anzahl der Elemente in diesem Vektor. v_j ist der j-te Wert des Attributes A_h.

$$E(\vec{\tau}^{(i)}(A_h)) = \frac{1}{\sum_{j=1}^{L_h} \tau_j(A_h)} \sum_{j=1}^{L_h} v_j \tau_j(A_h). \tag{4.8}$$

Wir wollen überprüfen, inwiefern dieses Constraint die in Kap. 4.1 definierten Kriterien erfüllt.

1. Verständlichkeit für den Benutzer Der Benutzer kann hier klar bestimmen, die Wertemengen welcher Attribute er über die Cluster verteilt haben möchte. Dies kann sinnvoll sein, wenn er z. B. die Clustermengen nach Tempo aufgeteilt haben möchte, also, dass langsame Lieder einem anderen Cluster zugewiesen werden sollen als schnellere. Noch besser wäre dieses Constraint mit Metadaten zu erklären. Wenn der Benutzer seine Musiksammlung über die Erscheinungsjahre klassifizieren möchte, so braucht er dafür nur in diesem Constraint das entsprechende Attribut auszuwählen und schon werden die Musikstücke mit verschiedenen Jahreszahlen so weit wie möglich auf die Cluster verteilt, jedoch beliebig und nicht in zusammenhängenden Intervallen. Auch unterliegt man hier der Beschränkung durch die Anzahl der Cluster. Wenn man, z.B. nur zwei Cluster hat, so können die Musikstücke auch nur in zwei Mengen aufgeteilt werden.

2. Güte der Auswertung Die Güte der Auswertung bezieht sich nur auf die ausgewählten Attribute und deren Wertemengen. Das hat einen Einfluss auf die Verteilung der Musikstücke auf die Cluster.

3. Effizienz Wir müssen alle H^* von dem Benutzer ausgewählte Attribute betrachten. Für jedes dieser Attribute müssen wir alle k Cluster betrachten und die Funktionen VAR und R berechnen. H^* ist aber durch die konstante Anzahl aller Attribute beschränkt und kann als ein konstanter Faktor vernachlässigt werden. VAR kann in $\mathcal{O}(L_h)$ berechnet werden, da wir E nur einmal berechnen müssen. L_h ist aber jeweils durch die Anzahl der Musikstücke in dem Cluster beschränkt. $\sum_{k=1}^{K} L_h^{(i)}$ ist gleich n, die Anzahl aller Musikstücke.

Insgesamt erhalten wir also eine Laufzeit von $\mathcal{O}(n)$.

4.4.4 Constraint Typ 4: Number of Clusters

Der Benutzer hat hier die Möglichkeit eine untere (k_{min}) und eine obere (k_{max}) Grenze dafür anzugeben, in wieviele Cluster er die Musikstücke eingeordnet haben möchte.

Dieses Constraint ist so übernommen wie es in [54] beschrieben wurde. Dieses und die nächsten zwei Constraints gehören zu der im Kapitel 2.6.3 aufgeführten Sparte der *Existential Constraints*.

Auswertungsformel

Wir verwenden hier die in der Gleichung 4.2 definierte σ-Funktion und erhalten die folgende Formel für die Auswertung dieses Constraints:

$$\theta_4(C) = \sigma(k, k_{min}, k_{max}, k_{max} + 1) \tag{4.9}$$

Diese Definition gleicht der einer Fuzzy-Menge. Wenn k innerhalb von $[k_{min}, k_{max}]$ liegt, wird eine 1 zurückgegeben. Ansonsten erhält man einen Wert in $[0, 1[$ zurück, der die Nähe zum Intervall $[k_{min}, k_{max}]$ beschreibt.

Wir wollen überprüfen, inwiefern dieses Constraint die in Kap. 4.1 definierten Kriterien erfüllt.

1. Verständlichkeit für den Benutzer Der Benutzer kann bei diesem Constraint beliebig genau festlegen, in wieviele Cluster er die Musikstücke unterteilt haben möchte. Die Anzahl der Cluster hängt stark von der Verwendung ab, die der Benutzer dem System zugedacht hat.

2. Güte der Auswertung Dieses Constraint erlaubt nur eine auf die Anzahl der Cluster beschränkte Auswertung. Über den Inhalt der Cluster sagt dieses Constraint nichts aus und überprüft ihn auch nicht.

3. Effizienz Diese Auswertung kann in $\mathcal{O}(1)$ Schritten durchgeführt werden.

4.4.5 Constraint Typ 5: Cluster Cardinality

Der Benutzer hat bei diesem Constrainttyp die Möglichkeit, eine Mindestanzahl m_{min} an Musikstücken, die einem Cluster zugewiesen werden, zu bestimmen.

Dieses Constraint wurde unverändert aus [54] übernommen.

Auswertungsformel

Wir benutzen wieder die Funktion σ aus 4.2.

$$\theta_5(C) = \frac{1}{k}\sum_{i=1}^{k}\sigma(m_i, m_{min}, n, n) \tag{4.10}$$

Dabei ist m_i die Kardinalität des Clusters i, n ist die Anzahl der Musikstücke und k ist die Anzahl der Cluster.

Hier wird ein Mittelwert über alle Cluster gebildet und zwar so, dass die einzelnen σ eine 1 zurückgeben, wenn das aktuelle Cluster mehr Elemente hat als m_{min} und ansonsten einen Wert in $[0, 1[$ liefert, welcher angibt, wie weit man von dem erwünschten Ziel entfernt ist.

Wir wollen überprüfen, inwiefern dieses Constraint die in Kap. 4.1 definierten Kriterien erfüllt.

1. Verständlichkeit für den Benutzer Im Allgemeinen kann solch ein Constrainttyp dazu verwendet werden, um Ausnahmen unter den Musikstücken, sogenannten *Ausreißern* zu begegnen, indem man z. B. Cluster mit nur einem, bzw. zu wenigen Elementen verbietet.

2. Güte der Auswertung Die Auswertung beschränkt sich nur auf die Mindestanzahl der Musikstücke in den Clustern.

3. Effizienz Diese Auswertung kann in $\mathcal{O}(k)$ Schritten durchgeführt werden.

4.4.6 Constraint Typ 6: Cardinality Ratio

Dieses Constrainttyp ermöglicht dem Benutzer die Eingabe des Verhältnisses von $R = \frac{m_{min}}{m_{max}}$, also der maximal und der minimal möglichen Anzahl der Musikstücke in einem Cluster. Das tatsächliche Verhältnis muss größer oder gleich R sein. Dadurch gibt man die *Ballance* der Kardinalitäten der Cluster an.

Auswertungsformel

Wir benutzen wieder die Funktion σ aus 4.2. In [54] werden zunächst die aktuellen m_{min} und m_{max} bestimmt:

$$\begin{aligned} m_{min} &= \min_i\{m_i\} \quad i = 1, \ldots, k \\ m_{max} &= \max_i\{m_i\} \quad i = 1, \ldots, k \end{aligned} \tag{4.11}$$

m_i ist die Kardinalität des Clusters i, k ist die Anzahl der Cluster.

Die Formel für die Auswertung dieses Constraints lautet:

$$\theta_6(C) = \sigma(1 + \frac{m_{max}}{m_{min}}, 1 + R, 2, 2) \tag{4.12}$$

Der Wert von σ beträgt, wenn $1 + \frac{m_{max}}{m_{min}} \geq 1 + R$ ist, 1 und ansonsten ist es ein Wert in $[0, 1[$, welcher die Annäherung des Quotienten an R angibt.

Wir wollen überprüfen, inwiefern dieses Constraint die in Kap. 4.1 definierten Kriterien erfüllt.

1. **Verständlichkeit für den Benutzer** Hier kann der Benutzer also den Freiheitsgrad für die Befüllung der einzelnen Cluster angeben. Manch ein Benutzer möchte, dass alle Cluster so ziemlich die gleiche Anzahl an Elementen erhalten, also wählt er ein Verhältnis von $R = 1$. Ein anderer bevorzugt lieber Cluster mit einer möglichst unterschiedlichen Anzahl von Elementen, wodurch der Clusteringalgorithmus mehr Freiheiten erhält, die einzelnen Musikstücke auf die Cluster zu verteilen. Dazu wählt er ein Verhältis von $R < 1$.

2. **Güte der Auswertung** Auch hier bezieht sich die Auswertung nur darauf, wie gut das aktuelle Verhältnis von m_{min} und m_{max} dem gewünschten entspricht. Weder über die tatsächlichen Minima, bzw. Maxima noch über die einzelnen Musikstücke wird etwas gesagt.

3. **Geschwindigkeit der Auswertung** Diese Auswertung kann wegen der Bestimmung des aktuellen Verhältnisses in $\mathcal{O}(k)$, mit der Speicherung der Maximalwerte sogar in $\mathcal{O}(1)$ durchgeführt werden.

4.4.7 Constraint Typ 7: Cluster Adherence

Dieser Constrainttyp ist ein „logisches UND“ [54] zwischen den Constraints vom Typ 2 und 3 (Kap. 4.4.2 und 4.4.3). Das bedeutet, dass der Benutzer auf der einen Seite angeben kann, die Musikstücke mit dem spezifizierten Wert für dieses Merkmal gehören in ein Cluster, aber die Musikstücke mit anderen Werten dieses Merkmals müssen so weit wie möglich über die Cluster verteilt werden.

Auswertungsformel

Da wir es mit dem „logischen UND“ der beiden Constraintstypen 2 und 3 zu tun haben, und die Einzelauswertungen der beiden Constraintstypen in dem Intervall $[0, 1]$ liegen, ist es am intuitivsten einen Fuzzy-UND-Operator zu verwenden, um diese beiden Werte zu verknüpfen. Ein Standard UND-Operator ist der ZADEH-Minimumoperator T_m [81]:

$$T_m(a, b) = \min(a, b) \tag{4.13}$$

Damit erhalten wir die folgende Gleichung für dieses Constraint:

$$\theta_7 = \min(\theta_2, \theta_3) \tag{4.14}$$

Wir wollen überprüfen, inwiefern dieses Constraint die in Kap. 4.1 definierten Kriterien erfüllt.

1. **Verständlichkeit für den Benutzer** Der Benutzer kann genauer spezifizieren, was mit den Musikstücken für alle Werte der Merkmale geschehen soll, z.B. könnte der Benutzer explizit darauf hinweisen wollen, dass die Musikstücke, die das Merkmal Tempo von 180bpm[5] haben, in ein Cluster gehören, während die Musikstücke mit einem anderen Tempo möglichst anderen Clustern zugewiesen werden.

2. **Güte der Auswertung** Eine Kombination der Güten der beiden Constraintstypen 2 und 3.

3. **Effizienz** Die Auswertung dieses Constraints kann offensichtlich in $\mathcal{O}(1)$ Schritten durchgeführt werden, wenn θ_2 und θ_3 vorher bestimmt wurden.

[5] bpm = beats per minit, eine Einheit für das Tempo eines Musikstückes.

5 Die allgemeine Vorgehensweise

In diesem Kapitel wird die Interaktion zwischen dem Benutzer und dem System näher erläutert, so dass verständlich wird, wie die Constraints von dem Benutzer an das System vermittelt werden und wie das System daraus adäquate Cluster konstruiert.

Im Hinterkopf müssen wir immer bedenken, dass unser Benutzer von der eigentlichen Funktionalität des Systems keine Ahnung hat. Er formuliert die Constraints und möchte, dass mit dieser Information Cluster erstellt werden, die seinen Vorstellungen entsprechen.

Leider kann der Benutzer oft selbst nicht genau formulieren, was er letztendlich möchte, bzw. versteht er evtl. die Constraints falsch. Im Laufe der Zeit können sich auch seine Musikpräferenzen ändern (*concept drift*) [46] und er würde auch hier seine Constraints korrigieren wollen. Deswegen ist unser System, wie in der Einleitung schon festgestellt, eine interaktive Endlosschleife.

Zu Beginn erhält der Benutzer die vorgeclusterten Daten. Die Gewichte der Merkmale sind zunächst auf 1 gesetzt. Das Vorclustern ist sinnvoll, da der Benutzer keine Lust haben darf irgendwelche Constraints zu setzen und trotzdem seine Daten irgendwie geordnet sehen möchte. Die Constraints haben am Anfang alle die Gewichte auf 0 gesetzt, da sie noch keinen Sinn ergeben.

Nach der oberflächlichen[1] Betrachtung der Cluster und der enthaltenen Musikstücke modifiziert der Benutzer die Constraints nach seinen Präferenzen. Er stellt dabei auch ein, wie wichtig ihm die Erfüllung dieser Constraints ist.

Diese Eingaben erhält ein interner Optimierungsalgorithmus (Kap. 6), der die Merkmalsgewichte so modifiziert, dass ein erneut angewandter Clusteringalgorithmus neue Cluster hervorbringt, die möglichst viele Constraints erfüllen.

Danach ist der Benutzer entweder mit der Aufteilung zufrieden und belässt das System, bis sich seine Präferenzen ändern, oder er stellt fest, dass die Constraints noch nicht richtig eingestellt worden sind und modifiziert diese. Im Anschluss wird der Kreislauf erneut gestartet.

Dieses Vorgehen verdeutlicht die Abbildung 5.1.

5.1 Aufgabe

Die Vorgehensweise für den Benutzer ist einleuchtend. Doch welche *Hauptaufgabe* muss unser System erledigen?

Abbildung der Constraints auf Merkmalsgewichte. Wir müssen die Constraints auf die Merkmalsgewichte abbilden, so dass die Eingabe der Constraints die Gewichte so modifiziert, dass das anschließende Clustering so viele Constraints wie möglich erfüllt.

[1] Keinem Benutzer kann zugemutet werden, dass er sich mit einer großen Datenbank mit mehreren tausend Musikstücken eingehender als nur oberflächlich beschäftigt (siehe Einleitung).

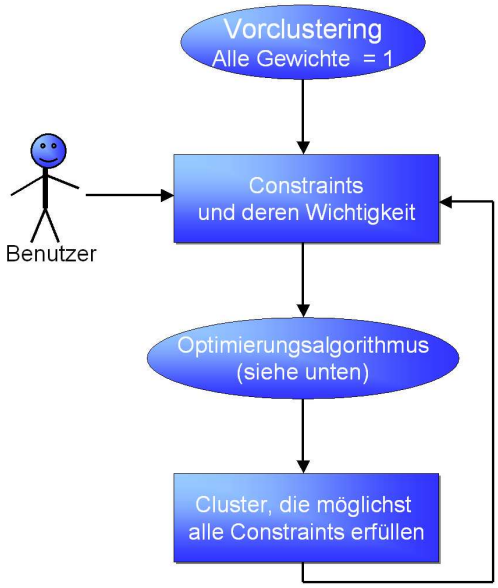


Abbildung 5.1: **Die allgemeine Arbeitsweise des Systems.** Der Benutzer modifiziert aufgrund der vorgeclusterten Daten die Constraints. Ein interner Optimierungsalgorithmus verändert die Merkmalsgewichte so, dass die Musikstücke nach einer erneuten Anwendung eines Clusteringalgorithmusses so neu geclustert werden, dass die Cluster so viele Constraints wie möglich erfüllen.

6 Optimierung

»The best is the enemy of the good.«
Voltaire

Die zentrale Aufgabe des Algorithmus, welcher nach den Constraints des Benutzers clustern soll, ist die Optimierung der Gewichte der Attribute der Musikstücke mit dem Ziel der constraintskonformeren Veränderung der Distanzen.

DEFINITION 12: (**Optimierungsaufgabe [29]**)
Minimiere eine reelwertige Funktion Φ auf einer gegebenen Menge M, der Menge der zulässigen Lösungen, das heißt, finde ein $\mathbf{p}^* \in M$ mit $\Phi(\mathbf{p}^*) \leq \Phi(\mathbf{p}), \forall \mathbf{p} \in M$

- *Nebenbedingungen allgemein:*
  $G_j(\mathbf{p}) = G_j(p_1, p_2, \ldots, p_H) \leq 0, j = 1, 2, \ldots, m$
- $\mathbf{p} = p_1, p_2, \ldots, p_H$ *sind die Optimierungsparameter*
- $\omega = \Phi(\mathbf{p})$ *ist der Wert der Zielfunktion* Φ
- $\mathbf{p}^*$ *heißt die Lösung der Optimierungsaufgabe*
- *Durch die Umwandlung der Aufgabe nach den Regeln von* DE MORGAN *erhält man auch eine Maximierungsaufgabe*

Das Ziel ist es also eine Zielfunktion über alle Constraints aufzustellen, welche maximiert werden soll.

6.1 Methodik

In diesem Abschnitt wird die Vorgehensweise für die Aufstellung der Zielfunktion für die Optimierung im Einzelnen erläutert und motiviert. Als Übersicht und Ankerpunkt dient dazu die Abbildung 6.1.

Gegeben ist der Algorithmus A, mit dem so viele Constraints wie möglich erfüllt werden sollen, nachdem eine Gewichtung der Merkmale gefunden wurde. X ist die Eingabemenge, also die einzelnen Musikstücke mit ihren sie repräsentierenden Merkmalsvektoren, und Y ist die Ausgabe, also die Partition der Daten in Cluster. $P = \{p_1, \ldots, p_S\}$ ist die Menge der Gewichtungen der Merkmale. Um Missverständnisse vorzubeugen sei hier erklärt, dass nicht jedes Musikstück für seinen Merkmalsvektor eine eigene Gewichtung P erhält, sondern, dass man nur eine Gewichtung über alle Musikstücke hat. Die Gewichte in P sind die Parameter, über welche optimiert wird. Am Anfang sind die Parameter $p_i = 1$, also alle Merkmale sind für den Clusteringalgorithmus A von gleicher Bedeutung. Wenn wir im Laufe des Algorithmus nicht genau wissen sollten, wie einzelne Merkmale zu gewichten sind, so erhalten diese das Standardgewicht 1. Wir erhalten also die folgende Gleichung: $Y = A(X, P)$.

Auch wenn der Benutzer nicht genau weiß, wie Y letztendlich aussehen sollte, kann er die Constraints $V = \{V_1, \ldots, V_T\}$ angeben, welche Y am Ende erfüllen muss. Für jedes Constraint kann er auch die Gewichtung $Q = \{q_1, \ldots, q_T\}$ in [0,1] bestimmen, welche den Grad angibt, wie wichtig dem Benutzer die Erfüllung dieses Constraints ist.

Im ersten Schritt wird jedes Constraint in eine numerische Funktion f_i kodiert, welche in dem Intervall [0,1] ist und den Grad angibt, zu welchem die gegebene Ausgabe das Constraint erfüllt.

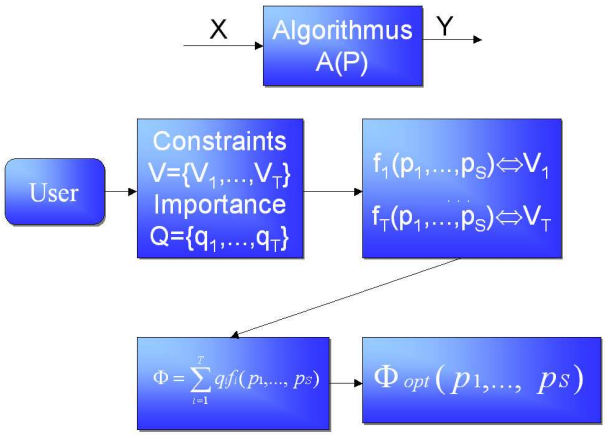


Abbildung 6.1: **Allgemeine Vorgehensweise bei der Aufstellung der Zielfunktion für die Optimierung.**

Die Funktionen f_i werden aufsummiert, um eine globale Zielfunktion aufzustellen, welche maximiert werden soll. Dieser Ansatz ist notwendig, da man nicht ausschließen kann, dass die Constraints des Benutzers einander widersprechen können, so dass man nicht mehr darauf hoffen kann, ein globales Maximum zu erhalten. Eine tiefergehende Diskussion über die Erfüllbarkeit der sich widersprechenden Constraints wird in 4.3 durchgeführt.

Um die Constraints zu erfüllen, müssen wir die Gewichte der Merkmale modifizieren. Wenn die analytische Abhängigkeit zwischen den Gewichten und der Ausgabe bekannt ist, dann ist unser Problem einfacher, da es ausreichend ist, die Region in dem Raum der Gewichte zu bestimmen, in dem die erwünschte Ausgabe auftritt. Diese Situation kommt aber selten vor. Meistens ist die implizite Funktion der Gewichte, welche die Ausgabe bestimmt, unbekannt. Also muss man hier einen Lernalgorithmus ansetzen, welcher heuristisch die optimale Gewichtung ermittelt.

6.2 Die Vorgehensweise bei der Optimierung

In diesem Abschnitt werden wir den groben Ablauf der Optimierung vorstellen, welcher nach der Spezifikation der Constraints in Kraft tritt. Er hat die Aufgabe die Cluster so neu zu ordnen, dass sie so viele Constraints wie möglich erfüllen.

Der Algorithmus erwartet eine Menge X der Musikstücke, die K Clustern zugeteilt werden sollen. Die Musikstücke werden durch die Menge der Merkmale $A = A_1, \ldots, A_H$ und deren Gewichte $P = p_1, \ldots, p_H$ repräsentiert. Die Merkmalswerte sind fest, wir dürfen also nur die Gewichte manipulieren. Als weitere Eingabe haben wir die Menge der Constraints $V = V_1, \ldots, V_T$ und deren Wichtigkeit $Q = q_1, \ldots, q_T$. Die initiale Ausführung von k-Means (DBSCAN) erzeugt uns eine Aufteilung in Cluster C.

1. Führe, falls von dem Optimierungsalgorithmus benötigt, Constraint Preprocessing aus, woraus aus den unterschiedlichen Constraints nur Must-Links und Cannot-Links entstehen (Kap. 6.5).

2. Starte eines der *Optimierungsverfahren* (Kap. 6.7) und berechne die neuen Gewichte für die Merkmale. Diese sollen die Musikstücke so im Raum verschieben, dass die Constraints so gut wie möglich erfüllt werden.

3. Anschließend wird erneut k-Means (DBSCAN) aufgerufen, wobei die Musikstücke mit dem neuen Gewichtsvektor verwendet werden.

4. Es wird automatisch überprüft (Kap. 4.4), inwieweit die Constraints erfüllt sind. Die Ergebnisse der Evaluation und die neue Clusterung wird ausgegeben.

5. Es wird eine neue Iteration des Systems gestartet.

Die Abbildung 6.2 veranschaulicht noch einmal den Algorithmus.

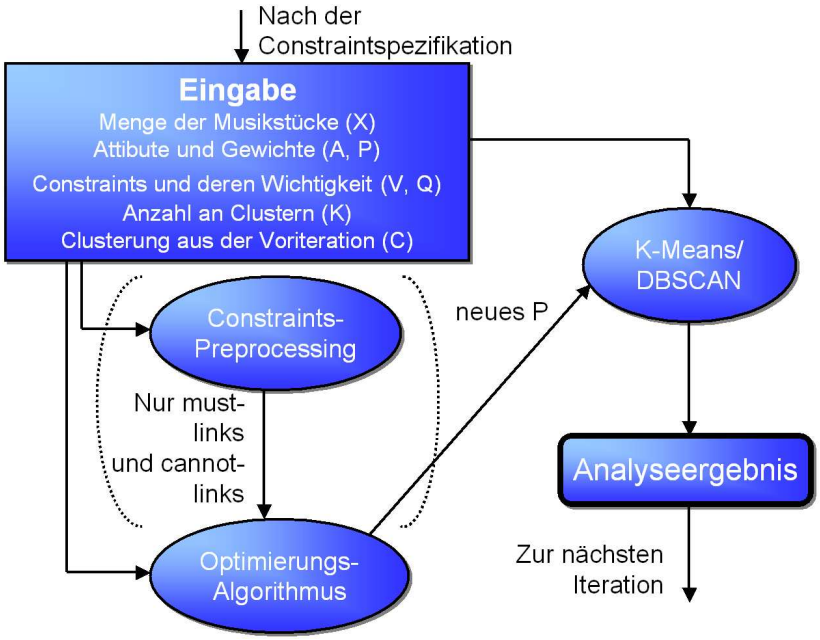


Abbildung 6.2: **Der Interne Optimierungsalgorithmus.** Zunächst werden, falls es für den Optimierungsalgorithmus notwendig ist, alle Constraints in Must-Links und Cannot-Links umgewandelt. Diese Paare werden von einem Optimierungsalgorithmus dazu verwandt, eine neue Gewichtung zu berechnen. Mit Hilfe dieser Gewichtung berechnet ein Clusteringalgorithmus (k-Means, DBSCAN) ein neues Clustering und gibt es zusammen mit dessen Evaluation aus.

6.3 Distanz

Jedes Musikstück hat einen Merkmalsvektor, welches aus Merkmalswerten und deren Gewichten besteht. Die Merkmalswerte aller Musikstücke sind so normiert, dass der Mittelwert bei 0 liegt und die Standardabweichung bei 1, bzw. -1.

Um zu bestimmen, welche Musikstücke nahe beieinander liegen und welche nicht, muss man ihre Position und vor allem ihre *Distanz* untereinander in dem Raum kennen, welchen der Merkmalsvektor aufspannt.

Hier wählen wir das für *quantitative* Daten gebräuchlichste Distanzmaß, die *Euklidische Distanz.* Es ist ein anderes Distanzmaß, als in [54] vorgeschlagenes, weil man dort prinzipiell von *qualitativen* Daten ausgeht und kein besseres Maß angeben kann. Die Euklidische Distanz hat aber den entscheidenden Vorteil, dass man für jeden Punkt eine absolute Position angeben kann und von den relativen Distanzmaßen unabhängig agieren kann.

DEFINITION 13: (Distanz)
Die **Distanz** *ist eine Funktion, welche den Abstand zweier Musikstücke in dem, von dem Merkmalsvektor aufgespannten, Raum angibt.*
Musikstücke innerhalb eines Clusters müssen eine kleine Distanz untereinander aufweisen, während Musikstücke in verschiedenen Clustern eine große Distanz untereinander aufweisen müssen.

Sei X die Menge aus n Musikstücken. $\mathcal{A} = \{A_1, \ldots, A_H\}$ die Merkmalsmenge, wobei $A_h \in \mathbb{R}$ ist. Sei $v^{(i)}$ der Wert des Merkmals A_h in dem Musikstück x_i. Für jedes Merkmal wird eine Distanzfunktion der Merkmalswerte und deren Gewichte eingeführt. Die Distanz zwischen zwei Musikstücken x_1 und x_2 für das Attribut A_h bezeichnen wir mit $d_h(v^{(1)}, v^{(2)}) = \left|v^{(1)} - v^{(2)}\right|$. Die globale Distanz zwischen zwei Musikstücken ist die Summe der Einzeldistanzen multipliziert mit den Merkmalsgewichten:

$$D(x_1, x_2) = \sqrt{\sum_{h=1}^{H} p_h \cdot d_h \left(v^{(1)}, v^{(2)}\right)^2} = \sqrt{\sum_{h=1}^{H} p_h \cdot d_h^2}. \tag{6.1}$$

Ein optimaler Clusteringalgorithmus maximiert eine Zielfunktion, welche auf der Distanz aufbaut.

Die Werte der Attribute bewegen sich in dem *Euklidischen Raum*, für den alle Eigenschaften einer Metrik gelten, vor allem die Dreiecksungleichung. Mit dieser Erkenntnis brauchen wir kein spezielles Ähnlichkeitsmaß zu definieren, sondern können in einem für uns vertrauten Raum bleiben.

Die einzelnen Musikstücke bewegen sich in dem Raum $\mathbb{R}^H$. Jedes einzelne Merkmal A_h multipliziert mit dem Merkmalsgewicht p_h ist eine Koordinate.

6.4 Gewichtung der Merkmale

In dieser Arbeit gehen wir, scheinbar willkürlich, von einem Gewichtsvektor aus, der für jeden Merkmalstyp einen Wert in $[0, 1]$ annehmen kann. Außerdem besteht zunächst kein Grund für die Art der Gewichtung der Distanzfunktion (Kap. 6.3) mit den Gewichten dieses Vektors. Es ist allerdings so, dass die als fest vorgegebene Designentscheidungen vorgestellte Kriterien in Wirklichkeit optimierbare Parameter sind. In der Literatur wurden diverse Möglichkeiten für die Gewichtung untersucht und deren Vor- und Nachteile studiert. Eine gut strukturierte Übersicht bietet z.B. [75][1].

6.4.1 Dimensionen der Gewichtungsmethoden

Die Merkmalsgewichtungsmethoden können in mehreren Dimensionen organisiert werden:

1. *Bias*
2. *Weight Space* (Gewichtsraum)
3. *Representation*
4. *Generality* (Allgemeinheit)
5. *Knowledge* (Domainwissen)

Bias

In dieser Dimension unterscheidet man zwischen zwei möglichen Werten: Entweder werden die Gewichtsaktualisierungen durch *Performanzrückmeldungen* gesetzt (*open loop*, *wrapper models*), oder ist es ein vorbestimmtes (*preset*) Kriterium (z. B. die Constraints), nach welchem sich die Güte der Gewichtung richtet (*closed loop*, *filter models*).

[1] Die Ausführungen in diesem Kapitel basieren auf [75].

Performance Bias-Methoden haben den Vorteil, dass bei ihrer Suche nach der nächsten Belegung des Gewichtsvektors sie stets darüber informiert werden, wie weit sie vom Optimum entfernt sind. Dabei werden zwei Gruppen unterschieden: *Online Search*-Gruppen, Algorithmen, die sequentiell jede Instanz ein Mal durchlaufen, und *batch-optimization*-Gruppen von Algorithmen, die wiederholt die Instanzen betrachten. Ein Beispiel für Online-Search ist der Algorithmus EACH [55], bei dem die Gewichte der unpassend gesetzten Merkmale erhöht, während die Gewichte der guten Merkmale erniedrigt werden. Die Gewichte werden durch die Addition eines Δ auf folgende Weise verändert:

$$p_i = p_i + \Delta. \tag{6.2}$$

Dieser und ähnliche Ansätze haben vor allem den Nachteil, dass deren Güte von der Reihenfolge, in der die Instanzen eingelesen werden, abhängt. Beispiele für die Batch-Optimizer sind die Genetischen- oder die Evolutionären Algorithmen, die im Kapitel 6.7.6 ausführlich behandelt werden.

Preset-Bias-Methoden benutzen eine existierendes Modell, um die Gewichte zu bestimmen. Es sind hauptsächlich Methoden, bei denen bedingte Wahrscheinlichkeiten eine Rolle spielen. Es werden drei Klassen dieser Methoden differenziert: *conditional probabilities*, *class projection* und *mutual information*. Ein Beispiel für Conditional Probabilities ist die *Cross-Category Feature Importance*-Methode (CCF) von Creecy [14], die versucht höhere Werte den Merkmalen zuzuweisen, die in wenigen Klassen vorkommen. Die Gewichtsaktualisierung kann dann folgendermaßen vorgenommen werden:

$$p_i = \sum_{c_j \in C} p(c_j|a_i)^2, \tag{6.3}$$

wobei c_j das j-te Cluster in C und a_i das i-te Merkmal ist. Als Beispiel für Class Projection kann *Value Difference Metric* (VDM) [61] angenommen werden, welche höhere Werte den Merkmalen zuweist, welche eine ungleichmäßige Verteilung der Werte über die Cluster aufweisen. Die letzte Klasse, Mutual Information, wurde im Kapitel 2.6.3 dieser Arbeit ausführlich behandelt. Diese weisen höhere Gewichte den Merkmalen zu, die sicherere Informationen anbieten.

Weight Space

Entweder wir betreiben *Merkmalsgewichtung (feature weighting)* mit einem *kontinuierlichen* Gewichtsraum, oder es ist eine *Merkmalsauswahl (feature selection)*, bei der nur ein *binärer* Gewichtsraum zugelassen ist.

Es wurden viele verschiedene Ansätze vorgestellt, die Merkmalsauswahl betreiben. Dabei wurden die Merkmale anhand von Entscheidungsbäumen [11], Hill-Climbing mit zufälligen Mutationen [60], paralleler Suche [50], Beam-Search mit schrittweiser Auswahl [2] ausgewählt. Oft ist es möglich den gleichen Algorithmus sowohl für Merkmalsgewichtung als auch für Merkmalsauswahl zu benutzen, in dem man an der Wertemenge der Gewichte manipuliert.

Representation

Die Repräsentation der Merkmale kann so übernommen werden, wie sie ist (*given representation*), aber sie kann aber auch transformiert werden, um eine bessere Performanz zu erhalten (*transformed representation*).

Die Transformation kann interagierende und korrelierende Merkmale stärker berücksichtigen. Dies kann entweder durch eine trianguläre Gewichtsmatrix, die Gewichte kombiniert, realisiert werden (Kap. 6.7.5), oder durch eine Transformation der Repräsentation der Merkmale, bevor diese gewichtet werden. Ein Beispiel dafür ist *Quantification Method II* (QM2m) [49]. Ein signifikanter Nachteil der Merkmalstransformationsmethoden ist, dass die transformierten Merkmale keine Aussagekraft mehr haben [75].

6.4.2 Generality

Die Gewichte können sich auf alle Instanzen gleichwertig auswirken (*global*). Sie können sich aber auch nur auf eine Auswahl an Instanzen auswirken oder auf einen bestimmten Instanzraum (*local*).

Ein Beispiel für ein lokalen Ansatz ist VDM [61]. Diese weist für einen jeden Wert eines Merkmals ein anderes Gewicht zu. Zwei Nachteile dieser Art Ansätze wurden herausgearbeitet: Auf der einen Seite sind sie empfindlich für Ausreißer, auf der anderen Seite kann die notwendig riesige Anzahl von verschiedenen Distanzfunktionen nützliche Merkmalsinformationen verschleiern [75]. Durch Kombination von lokalen und globalen Ansätzen hat man allerdings sehr gute Ansätze erzielt [75].

Knowledge

Wenn man mehr Wissen über die Domain besitzt, kann man auch speziellere Algorithmen bauen, die optimalere oder effizientere Lösungen liefern. Andererseits, wenn man nicht so viel von der Domain weiß, sind allgemeinere Algorithmen vorzuziehen. Bei den domainspezifischen Ansätzen versucht man die automatisierten Gewichtungskomponenten mit Heuristiken zu verbinden, die auf dem Domainwissen basieren. In *Explanation Based Learning* (EBL) [48] wird, z.B. die Similarity-Funktion zweier Objekte aufgrund des Domainwissens modifiziert und bessere Ergebnisse erzielt.

6.4.3 Einordnung unseres Ansatzes

Wie man schon sehen konnte, kann unser Ansatz nicht eindeutig in diese Aufteilung der Gewichtungsmethoden eingeordnet werden, da wir teilweise über mehrere Dimensionen hinweg die Ansätze vergleichen. Nichtdestotrotz lassen sich einige Tendenzen erkennen.

In der Dimension *Bias* sind unsere Ansätze alle Performance-Methoden, da wir keine Annahmen über den Merkmals- und Gewichtsraum machen und aufgrund der Performanzrückmeldungen vorgehen. Im Kapitel 2.6.3 wird über die Möglichkeiten diskutiert, auf Preset-Methoden umzusteigen.

Wir können die Gewichtsoptimierung sowohl als Merkmalsauswahl als auch Merkmalsgewichtung betrachten, allerdings wenden wir wegen genaueren Ergebnisse stets Merkmalsgewichtung an. Merkmalsauswahl kann Performanzvorteile bieten, allerdings verlieren wir dabei wertvolle Informationen, die wir, da der Merkmalsraum unbekannt ist, unbedingt brauchen.

Wie im Kapitel 6.3 deutlich wird, betreiben wir keinerlei Transformation. Allerdings wird im Kapitel 6.7.5 ausführlich über diese Möglichkeit diskutiert.

Unsere Ansätze sind allesamt global ausgerichtet und die Literatur [75] gibt uns Recht. Allerdings könnte man über eine kombinierte Verwendung von lokalen und globalen Methoden nachdenken, die in [75] gut abgeschnitten haben.

Alles, was wir über die Domain wissen, wird in Constraints codiert und diese beeinflussen unsere Algorithmen.

Wie man sieht, eröffnet die Fülle an verschiedenen Möglichkeiten, die Gewichte zu manipulieren, Stoff für zukünftige Untersuchungen im Rahmen unseres Systems.

6.5 Bildung von Must-Links und Cannot-Links aus Constraints

Die Menge an unterschiedlichen Constraints führt leider dazu, dass man von jedem Constraint eine andere Ausgabe erhält. Das bedeutet, dass die Optimierungsmethoden für die Erfüllung der Constraints entsprechend für jedes Constraint eine eigene Methode haben müssten, um dieses zu erfüllen.

In diesem Kapitel wollen wir untersuchen, inwiefern sich die Ausgabe der Constraints als Must-Links und Cannot-Links darstellen lässt. Für manche Constraints werden wir feststellen, dass dies nicht vollkommen möglich ist, für die meisten aber lässt sich eine Abbildung finden.

Diese Vorgehensweise ist keine unnötige Spielerei, sondern ein wichtiger Schritt in Richtung der Lösung des Problems. Wenn ein Löser sich mit einer einheitlichen Eingabe konfrontiert sieht, ist zu erwarten, dass er sowohl schneller, als auch stabiler arbeitet, denn eine neue Eingabe bedeutet auch neue Fehlerquellen. Natürlich kann die Transformation auch zu Fehlern führen, allerdings wird versucht, diese Routinen so einfach und durchsichtig wie möglich zu halten.

6.5.1 Instance Association

Dieses Constraint enthält der Definition nach genau die Paare von Must-Links und Cannot-Links. Allerdings haben wir, aufgrund einer leichteren Bedienbarkeit für den Benutzer wie im Kapitel 4.4.1 erwähnt, eine Menge von Zuweisungen von Musikstücken zu Clusternummern, z.B. „Musikstück x_i gehört zu Cluster C_j“, kurz $x_i \rightarrow C_j$. Wir müssen also die Paare erst noch bilden. Dies geschieht folgendermaßen:

Sei M_j die Menge aller Elemente x_i, die dem gleichen Cluster C_j zugewiesen wurden. Diese werden paarweise zu Must-Links zusammengefasst. Das bedeutet, dass ein jedes Element $x_{i_l} \in M_j$ mit jedem anderen Element $x_{i_m} \in M_j,\ l \neq m$ ein Must-Link bildet. Die Cannot-Links werden aus Paaren aller Elemente aus zwei unterschiedlichen Mengen M_j und M_k gebildet, jedes Paar besteht also aus Elementen, die zwei verschiedenen Clustern zugewiesen wurden. Und schließlich sollten Must-Links gebildet werden, die ausweisen, dass jedes Element x_i in das gleiche Cluster gehört wie er selbst, also Paare (x_i, x_i). Der letzte Punkt scheint auf den ersten Blick überflüssig zu sein, jedoch erweist sich diese Forderung als eine wichtige Grundlage für die später erläuterten Algorithmen (z.B. Kap. 6.7.3).

Ein Beispiel soll die Sachlage näher erläutern. Vier Musikstücke, $x_1, \ldots, x_4$ seien vom Benutzer ausgewählt, x_1 und x_2 wurden dem Cluster Nummer 1 zugewiesen und x_3 und x_4 dem Cluster Nummer 2. Es werden die folgenden Must-Links gebildet: $(x_1, x_1); (x_2, x_2); (x_1, x_2)$ und die folgenden Cannot-Links: $(x_1, x_3); (x_1, x_4); (x_2, x_3); (x_2, x_4)$.

6.5.2 Value Association

Es wird eine Abbildung über zwei Schritte durchgeführt. Zuerst wollen wir einem Element eine Clusternummer zuweisen und danach genauso verfahren wie bei Instance Association. In allen folgenden Constraintsabbildungen wollen wir dieses Konzept beibehalten. Die Idee ist, dass alle Elemente mit dem ausgewiesenen Wert v_i für ein bestimmtes Merkmal A_j, möglichst einem Cluster C_l zugewiesen werden sollten. Man wählt das Cluster $C_{\max}$ mit den meisten Elementen mit dem gesuchten Wert v_i für A_j aus. Diesem weist man alle Elemente, die den ausgewiesenen Wert haben, sich aber in einem anderen Cluster C_k befinden, zu. Genauso verfährt man mit allen ausgewiesenen Werten. Die Werte können auch Intervalle sein.

Ein Beispiel: Für das Merkmal „Länge des Musikstückes“ habe der Benutzer angegeben, dass er alle Musikstücke unter 2 Minuten Länge möglichst in einem Cluster haben möchte. Es wird das Cluster, C_1, mit den meisten Musikstücken mit der gesuchten Länge ermittelt. Es werden alle Musikstücke außerhalb von C_1 gesucht, die kürzer sind als 2 Minuten. Diese werden C_1 zugewiesen.

6.5.3 Value Separation

Auch hier wollen wir zunächst Elemente und die zugehörige Clusternummern finden. Die Wertemenge M_v der ausgewiesenen Merkmale A_i sollte sich so weit wie es nur geht über die Cluster verteilen. Dazu bestimmt man die Kardinalität von M_v für ein Merkmal A_l, $|M_{v_l}|$, und teilt diese durch die Anzahl der Cluster k, also $\frac{|M_{v_l}|}{k}$. Dies ist der Mittelwert für die Mindestanzahl der Elemente mit verschiedenen Werten, die in einem Cluster enthalten sein sollen. Wenn die aktuelle Anzahl der Elemente in einem Cluster C_m unter diesem Mittelwert liegt, so werden Elemente mit entsprechenden Werten aus anderen Clustern diesem zugewiesen. Wenn sie aber über dem Mittelwert liegt, werden dessen Elemente auf andere Cluster mit Bedarf ausgelagert.

Ein Beispiel: Das Merkmal „Länge des Musikstückes“ sei von dem Benutzer ausgewählt. Es seien insgesamt 10 Musikstücke, 2 mit der Länge von 1,5 Minuten, 2 mit der Länge von 2, 2 mit der Länge von 3, 2 mit der Länge von 3,5, 1 mit der Länge von 4 und 1 mit der Länge von 5 Minuten. Es sind also 6 verschiedene Werte für das Merkmal lokalisiert. Angenommen, es existieren 3 Cluster. Es wird der Mittelwert $6/3 = 2$ gebildet. Dementsprechend muss jedes Cluster, C_i, mindestens zwei Misikstücke enthalten, welche für das ausgewählte Merkmal zwei verschiedene Werte aus der vorhandenen Wertemenge annehmen. Wenn dem nicht so ist, wird in den anderen Clustern $C_j, i \neq j$ nach solchen Musikstücken gefahndet und diese dem Cluster C_i zugewiesen.

6.5.4 Number of Clusters

Bei diesem Constraint stoßen wir auf Granit. Da dieser die Anzahl der Cluster vorschreibt, können wir mit Paaren nichts anrichten. Es ist aber auch nicht nötig. Zentroidbasierte Verfahren wie k-Means, welche den Kern unseres Algorithmus bilden (Kap. 6.2) bieten die Möglichkeit, die Information über die Anzahl k der gewünschten Cluster als Parameter einzugeben. Schwieriger wird es bei DBSCAN (siehe Kap. 2.6.2).

6.5.5 Cluster Cardinality

Dieses Constraint bestimmt die Mindestanzahl an Elementen aus X, die ein Cluster C_j enthalten sollte. Solange das durch die Gesamtanzahl der Elemente und der Cluster möglich ist, sollten die Elemente x_i so umpositioniert werden, dass anschließend möglichst viele Cluster die geforderte Anzahl erhalten. Dabei werden Elemente aus gut gefüllten Clustern den unterbesetzten Clustern zugewiesen.

Ein Beispiel: Der Benutzer habe angegeben, die Mindestanzahl der Musikstücke in einem Cluster solle 20 sein. Ein Cluster, C_1, enthalte nur zwei Musikstücke. Es wird ein Cluster, C_2 mit mehr als 20 Elementen gesucht und dessen Musikstücke solange C_1 zugeordnet, bis entweder C_1 die geforderte Anzahl von Elementen erreicht, oder C_2 nur noch 20 Elemente besitzt. Im zweiten Fall sucht man ein weiteres Cluster, C_3, mit mehr als 20 Musikstücken, usw.

6.5.6 Cardinality Ratio

Das Verhältnis von der minimalen Anzahl der Elemente in einem Cluster zu der maximalen Anzahl der Elemente sollte größer als der Parameter R sein, also $\frac{m_{min}}{m_{max}} > R$. Man weise so lange von dem maximalen Cluster Elemente einem minimalen Cluster zu, bis das Verhältnis R erreicht. Dabei ist das maximale Cluster im nächsten Iterationsschritt nicht unbedingt mehr das maximale, denn es hat ja ein Element abgegeben. Genauso ist das minimale auch nicht unbedingt mehr ein minimales Cluster, denn es hat ja ein Element hinzugewonnen.

Es kann aber sein, dass das optimale R nicht mit dieser Anzahl der Cluster erreicht werden kann. So müssen wir nach einer bestimmten Anzahl an Iterationsschritten, oder wenn sich das Verhältnis kaum noch ändert, abbrechen und uns mit einer suboptimalen Lösung zufrieden geben.

Eine weitere Lösung wäre die Anzahl der Cluster zu verändern, allerdings müsste man zunächst die Neuzuweisung der Elemente mit einem imaginären Cluster vollführen und später die tatsächliche Anzahl der Cluster verändern. Nicht nur dass es vielleicht mit den Einstellungen von Number of Clusters in Konflikt käme, welches dann aufzulösen wäre, es wäre auch fraglich, ob dieser immense Zusatzaufwand eine Verbesserung bringen würde.

Ein Beispiel: Angenommen, der Benutzer habe $R = 0,5$ spezifiziert. Es sind zwei Cluster, C_1 und C_2, bestehend aus 10 und 2 Musikstücken, gegeben. Das Verhältnis zwischen dem Maximum und Minimum ist $2/10 = 0,2 < R$. Es werden solange Musikstücke aus C_1 dem Cluster C_2 zugewiesen, bis dieses Verhältnis mindestens R entspricht. In diesem Fall müssen dafür genau 3 Musikstücke C_2 zugewiesen werden, denn dann wäre das Verhältnis $5/10 = 0,5 = R$.

6.6 Probleme des vorgestellten Systems

6.6.1 Grenzen der Distanzänderung durch Gewichtmodifikation

Bevor man Verfahren für die Lösung der Optimierung des Gewichtsvektors entwickeln kann, muss geklärt werden, ob man wirklich alleine durch Änderung des Gewichtvektors jedes Constraint erfüllen kann und ob es überhaupt möglich ist, damit Distanzen zwischen den Musikstücken zu ändern.

Dass die Distanzen sich verändern können, zeigt ein einfaches Beispiel. Angenommen, wir hätten drei Punkte: A, B und C in einem zweidimensionalen Merkmalsraum. Diese haben die Kordinaten $A = (1, 1), B = (1, 2)$ und $C = (2, 2)$. Wie man auf der Abbildung 6.3 a) sieht, bilden sie ein rechtwinkliges

Dreieck. Wenn man nun ein Must-link (A, B) hätte, dann würde es vollkommen ausreichen die Punkte mit dem Gewichtvektor $(1, 0)$ zu multiplizieren. Damit fallen die Punkte A und B aufeinander, während C weiterhin auf Abstand zu den beiden ist. Die Abbildung 6.3 b) veranschaulicht diese Ausführungen. Dies war ein einfaches Beispiel und man hat eher Merkmalsselektion betrieben, aber dennoch konnten wir zeigen, dass es möglich ist, Distanzen der Punkte zueinander nach unseren Vorstellungen zu ändern.

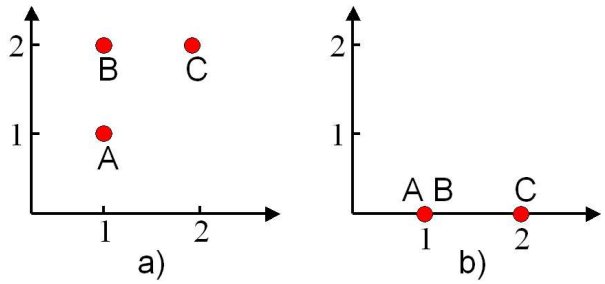


Abbildung 6.3: **Beispiel für Distanzänderung mit Gewichten.** a) Die drei Punkte sind in einem zweidimensionalen Merkmalsraum. Der Gewichtsvektor ist $(1, 1)$. b) Nach dem Setzen des neuen Gewichtsvektors $(1, 0)$. Die Punkte A und B fallen aufeinander. Ihre Distanz beträgt 0, so dass sie optimal den Must-Link (A, B) erfüllen.

Nun kommen wir zu der anderen Frage dieses Kapitels. Funktioniert unsere Vorgehensweisen jedes Mal? Können wir nur durch Gewichtsmodifikation beliebige Constraints erfüllen? Wir werden sehen, dass dies nicht immer möglich ist.

Betrachten wir wieder unser vorher vorgestelltes Beispiel auf der Abbildung 6.3. Nun haben wir ein Must-Link (A, C) und ein Cannot-Link (A, B). Schon bei der Betrachtung des Bildes kann man erkennen, dass unsere vorherige Vorgehesweise nicht zum Ziel führt, denn egal, ob wir das erste oder das zweite Glied des Gewichtsvektors auf 0 setzen, erhalten wir zwischen B und A keinen größeren Abstand als zwischen A und C.

Die vorhergehenden Beispiele werfen einige grundsätzliche Fragen auf:

1. Hängt diese Verhaltensweise von der Metrik ab, die wir verwenden? Führte eine andere Metrik zu einem anderen Verhalten?

2. Wird dieses Verhalten bei mehr als zwei Dimensionen gemildert, bzw. ganz aufgehoben?

3. Kann man schon im Voraus erkennen, dass ein Konflikt auftritt? Wenn dies möglich ist, dann kann man evtl. ein konfliktverusachendes Constraint zugunsten des anderen entfernen.

Leider scheinen diese Fragen in der Literatur noch gar nicht aufgeworfen worden zu sein, bzw. noch nicht ins Zentrum der Aufmerksamkeit der Forschung gerückt, denn kaum eine Arbeit vermittelt den Eindruck sich ernsthaft damit zu beschäftigen. Was auf den ersten Blick anhand der Beispiele als einfach erscheint, erfordert offensichtlich profunde theoretische Herangehensweise, die, aufgrund des Umfangs dieses Unterfangens, den nachfolgenden Arbeiten überlassen werden muss.

6.6.2 Problematische Merkmalsauswahl

Bis jetzt haben wir die in [46] generierten Merkmale ohne weiteres Hinterfragen in unser System übernommen. Doch diese wurden mit dem Ziel generiert, eine optimale Klassifikation nach den Genres zu gewährleisten. Es ist nicht klar, ob sie auch für beliebige Partitionierungen der Daten geeignet sind.

Es sind unendlich viele Merkmale denkbar, jedes ist für eine beliebige Partition im Durchschnitt gleich nützlich. Doch für bestimmte Partitionen stechen einige Merkmale so stark heraus, dass sie ohne sie nicht denkbar sind. Ein einfaches Beispiel dafür ist die Gruppierung der Musikstücke nach dem

Erscheinungsjahr. Würde dieses Merkmal fehlen, wäre es sehr schwer oder vielleicht sogar unmöglich die gewünschte Partitionierung anhand der anderen Musikstücke hinzukriegen.

Es ist also viel naheliegender, statt Merkmalsgewichtung, bzw. Merkmalsauswahl, eine *Merkmalsgenerierung* (engl. *feature generation*) bei jeder Anfrage des Benutzers durchzuführen, um die Menge der relevanten Merkmale zu erzeugen. Allerdings nimmt dieser Prozeß sehr viel Zeit in Anspruch[2], so dass diese Lösung für unser interaktives System nicht in Frage kommt.

Wir bleiben aus diesem Grund bei der im Kapitel 3.1 ausgesuchten Merkmalsauswahl, die für viele Fälle sich als gut genug erwiesen hat, wohlwissend, dass sie für viele Spezialfälle[3] nicht ausreichend oder sogar vollkommen nutzlos ist.

6.7 Optimierungsverfahren

In diesem Kapitel werden sechs Optimierungsverfahren vorgestellt und geprüft, inwiefern sie in unser System integrierbar sind. Es stellt sich heraus, dass nur fünf dieser Verfahren dafür geeignet sind.

Alle fünf haben die Aufgabe die gegebenen Constraints so optimal wie möglich zu erfüllen.

Gleichzeitig soll die Laufzeit nicht außer Acht gelassen werden, denn dem Benutzer ist lieber ein Verfahren, welches nicht ganz optimal entscheidet, dafür aber in wenigen Minuten eine Lösung liefern kann. Ein angenehmer *tradeoff* zwischen den zwei Anforderungen, Optimalität und Geschwindigkeit, ist höchst wünschenswert.

Bei allen Verfahren muss man dafür das Problem der Suche nach der Lösung der benutzerdefinierten Constraints in die als Eingabe für den jeweiligen Algorithmus erwartete Problemstellung transferieren.

6.7.1 Ein einfaches Score-basiertes Verfahren

In diesem Kapitel wollen wir ein sehr einfaches, aber sehr schnelles Verfahren vorstellen, um aus den Must-Links und Cannot-Links einen Gewichtsvektor zu konstruieren, der beim nächsten Durchlauf k-Means (DBSCAN) dazu veranlassen soll constraintskonformere Cluster zu bilden.

Die Vorgehensweise zeigt der Algorithmus 2.

Algorithmus 2: Der Score-Algorithmus

for jedes Must-Link- oder Cannot-Link Paar cp **do**
  for jedes Gewicht p_i des Gewichtsvektors **do**
    if cp ist Must-Link **then**
      Score = $(x_i - y_i)^2$
    else if cp ist Cannot-Link **then**
      Score = $1 - (x_i - y_i)^2$
    end if
    $p_i = p_i + \text{Score} \cdot q_{cp}$
  end for
end for
for jedes Gewicht p_i des Gewichtsvektors **do**
  $p_i = \frac{p_i}{\text{Anzahl Paare } cp}$
end for

Die Variable q_{cp} ist die Wichtigkeit des Constraints, welches dieses Paar erzeugt hatte. Wenn sie nicht so groß ist, dann bewirkt dieses Paar keine große Gewichtsveränderung.

Der Score soll für Must-Links umso höher sein, je weiter die Musikstücke des Paares voneinander entfernt sind. In diesem Fall soll eine große Gewichtsänderung erfolgen. Für Cannot-Links wünschen

[2] Durchschnittlich mehrere Tage Rechenzeit.

[3] Nach Murphy's Gesetz sind es oft genau die Fälle, die uns am meisten interessieren.

wir uns den umgekehrten Fall. Für diese muss der Score umso höher sein, je näher die Musikstücke des Paares beieinander sind. Dadurch wird auch hier eine Gewichtsveränderung bewirkt.

Wie man sieht, gibt man mit diesem Algorithmus noch nicht einmal die Richtung für die Veränderung der Gewichte vor, sondern ermittelt lediglich, dass eine Gewichtsänderung wünschenswert wäre. Deswegen überrascht es auch kaum, dass durch die zunächst weniger sinnvolle vertauschte Score-Funktion, also für Must-Links $1-(x_i-y_i)^2$ und für Cannot-Links $(x_i-y_i)^2$ ähnliche Ergebnisse wie die in dem Algorithmus 2 vorgestellten erbrachte.

Der Algorithmus 2 hat die Laufzeit $\mathcal{O}(g \cdot H)$, wobei g die Anzahl der Must-Link und Cannot-Link Paare ist und H die Anzahl der Merkmale. Da H allerdings als konstant angenommen werden kann[4], erhalten wir nur noch eine Laufzeit von $\mathcal{O}(g)$.

Dieser Algorithmus verschenkt Informationen. Obwohl wir konkrete Zahlenwerte erhalten, berufen wir uns ausschließlich auf grobe Heuristiken. Als Ausgangspunkt und zum Vergleich mit den folgenden, besser begründeten Verfahren, ist der Algorithmus allerdings sehr gut geeignet.

6.7.2 Analytisches Verfahren

Die schnellste Methode für eine Optimierung ist direkt eine analytische Lösung zu ermitteln. Wir versuchen unseren Merkmalsraum in den Raum der Distanzen so abzubilden, dass wir einen optimalen Gewichtsvektor direkt ausrechnen können.

Sei S die Menge der Must-Link-Paare und D die Menge der Cannot-Link-Paare. Daraus stellen wir die zu minimierende Funktion auf.

$$f(\mathbf{p}) = \sum_{x,y \in S} q \cdot d(x,y) - \sum_{x,y \in D} q \cdot d(x,y) \rightarrow \min \tag{6.4}$$

x und y sind die Paare der Musikstücke in den Must-Links oder den Cannot-Links, $d(x,y)$ ist das, in der Gleichung 6.1 definiertes Distanzmaß, q ist die Wichtigkeit des jeweiligen Constraintpaares.

Um die Gleichung 6.4 auf analytischem Wege zu minimieren, muss man die Ableitung bilden und diese gleich 0 setzen:

$$\nabla_i f(\mathbf{p}) = \sum_{x,y \in S} q \cdot (x_i - y_i)^2 - \sum_{x,y \in D} q \cdot (x_i - y_i)^2 + 2\lambda p_i = 0 \tag{6.5}$$

Um spätere Rechnungen einfacher darstellen zu können, setzen wir

$$k_i = \sum_{x,y \in S} q \cdot (x_i - y_i)^2 - \sum_{x,y \in D} q \cdot (x_i - y_i)^2, \ \forall i \tag{6.6}$$

Daraus folgt für p_i :

$$p_i = -\frac{k_i}{2\lambda} \tag{6.7}$$

Jetzt müssen wir nur noch λ ermitteln. Dabei hilft die folgende Nebenbedingung, die wir aus der Euklidischen Norm herleiten:

$$\|p_i\|^2 = \sum_{i=1}^{n} \frac{k_i^2}{4\lambda^2} = 1 \tag{6.8}$$

Diese Nebenbedingung gilt, weil wir nicht möchten, dass die Gewichte beliebig groß gesetzt werden dürfen. Sie ist eigentlich trivial, aber sie hilft uns in diesem Fall eine Lösung für λ zu ermitteln.

$$\lambda = \frac{1}{2}\sqrt{\sum_{i=1}^{n} k_i^2} \tag{6.9}$$

[4] Wir arbeiten mit genau 49 Merkmalen. Siehe Kap. 3.

und daraus ist es nun ganz leicht den endgültigen Wert für p_i zu ermitteln.

$$p_i = -\frac{k_i}{\sqrt{\sum_{i=1}^{n} k_i^2}} \tag{6.10}$$

Wir können also die Werte für die Gewichte p_i des Gewichtsvektors $\mathbf{p}$ direkt ausrechnen. Nicht nur, dass es ein sehr schneller Ansatz ist, es ist auch gut mathematisch begründet. Wir finden ein Optimum, allerdings ist es nicht unbedingt ein globales Optimum, da unsere Ausgangsgleichung 6.4 nicht konvex ist. Deswegen entspricht dieser Ansatz der Greedy-Suche, bei denen lokale Optima auftreten können.

Wenn man diesen Ansatz mit dem Score-Ansatz (Kap. 6.7.1) vergleicht, stellt man erstaunliche Ähnlichkeiten fest, obwohl sie zunächst weit voneinander zu liegen scheinen. Deswegen überrascht es kaum, dass die Evaluierungsergebnisse beider Ansätze sich ziemlich ähneln, obwohl ganz unterschiedliche Gewichtsvektoren gebildet werden.

6.7.3 Constraint Satisfaction Problems (CSPs)

In dem Titel fällt das Wort „Constraint“ auf. Unser Problem enthält ebenfalls eine Reihe von Bedingungen, Constraints, die gelöst werden müssen. Dem Namen nach scheint es also zu dieser Klasse von Problemen hinzu zu gehören. Wir werden in diesem Kapitel untersuchen, ob diese Behauptung wirklich zutrifft und ob wir unser Problem mit den Verfahren zur Lösung von Constraint Satisfaction Problems effizient lösen können.

DEFINITION 14: (Constraint Satisfaction Problems (CSPs) [53])
*Ein **Constraint Satisfaction Problem (CSP)** besteht aus:*

- *Einer Menge von **Variablen**: $X_1, \ldots, X_n$*
- *Jede Variable stammt aus einer nicht leeren Wertemenge (**Domain**) D_i der möglichen Werte. Sie kann diskret oder kontinuierlich sein.*
- *Einer Menge von **Constraints**, also Bedingungen, die für die Variablen $X_1, \ldots, X_n$ gelten sollen.*
- *Zuständen. Ein **Zustand** ist eine vollständige Belegung aller Variablen $X_1, \ldots, X_n$*

*Eine Belegung, die keins der Constraints verletzt, nennt man eine **konsistente Belegung**. Der zugehörige Zustand wird **konsistenter Zustand** genannt.*
*Das **Ziel** der CSPs ist, die Ermittlung eines konsistenten Zustands. Es existieren auch einige Formulierungen des CSPs, bei denen eine Lösung verlangt wird, die eine Zielfunktion maximiert.*

Die Klasse der Constraint Satisfaction Problems bietet eine Repräsentation für Probleme, die durch Strukturinformationen effiziente Suchstrategien ermöglicht. Tatsächlich ist es so, dass CSPs leicht als *Suchprobleme* definiert werden können [53]. Eine wichtige Eigenschaft von CSPs ist dass bei der Suche nach der optimalen Lösung der Suchpfad irrelevant ist. Dadurch ist es möglich effiziente Suchpfade auszuwählen, ohne sich an einen, von einem Suchalgorithmus vorgegebenen, zu halten. Suchprobleme sind auch Optimierungsprobleme, wenn man diese als Suche nach einer optimalen Lösung betrachtet.

Im Folgenden bezeichnen wir die Constraint Satisfaction Problems aus Gründen der Einfachheit mit der Abkürzung CSPs.

Die CSPs werden nach der Anzahl der von ihnen einbezogenen Variablen klassifiziert. Besonders oft kommen die *unären*, also die Constraints, die nur eine einzige Variable betreffen und die *binären*, also die Constraints, die zwei Variablen einbeziehen, vor. Ein Beispiel für ein unäres Constraint wäre: $X_1 < 0$, und für ein binäres: $X_1 + X_2 > 1$.

Bei den letzten zwei Beispielen verwendeten wir eine *constraint language* [53], die der arithmetischen Ausdrucksweise entspricht. Im Prinzip sind auch andere Sprachen für die Bestimmung der Constraints möglich.

Jedes Constraint, welches eine größere Anzahl an Variablen besitzt als ein binäres, kann in ein binäres Constraint durch Hinzunahme von genügend *Hilsfsvariablen* überführt werden [53]. Dies ist insbesondere deswegen sinnvoll, weil man die Constraints als einen *Constraintsgraphen* darstellen kann, auf welchen dann alle Erkenntnisse der Graphentheorie angewendet werden können. Im Constraintsgraphen bilden die Variablen die Knoten und die Constraints, welche zwei Variablen verbinden, sind die Kanten zwischen den beiden.

Die CSPs sind im allgemeinen NP-vollständige Probleme, das bedeutet, dass wir nicht für den allgemeinen Fall erwarten können, schnell zu sein. Sie sind schon für den Fall, dass die Domains der Variablen binär sind, also nur 0 oder 1 enthalten dürfen, NP-vollständig, denn damit enthalten sie das NP-vollständige Problem SAT als Spezialfall [53].

Allerdings gibt es für spezielle Sonderfälle sehr schnelle Lösungen, die die andere Suchmethoden bei weitem übertreffen. Leider gelten diese Sonderfälle vor allem für Variablen mit diskreten Wertemengen. Die Wertemenge unserer Variablen ist aber kontinuierlich.

Eine weitere Möglichkeit, die Effizienz der Lösung von CSPs zu verbessern, ist sich sinnvolle Heuristiken zu überlegen, die die Suchrichtung raten und dadurch schneller zum Ziel gelangen.

Besonders interessant wird es, wenn der Constraintgraph einen Baum darstellt, also einen Graphen, bei dem zwischen zwei Knoten höchstens ein Weg existiert. Sei d die Tiefe dieses Baumes und n die Anzahl der Knoten. Dann wird das CSP in Zeit $\mathcal{O}(nd^2)$ gelöst. Leider tritt dieser Sonderfall nur sehr selten auf, und der Versuch beliebige Graphen in Bäume umzuwandeln scheitert daran, dass die Umwandlung eine exponentielle Laufzeit benötigt [53].

Formulierung unseres Problems als CSP

Zunächst einmal muss geklärt werden, was die Variablen in unserem System sind. Die einzigen Größen, die wir manipulieren dürfen, sind die Merkmalsgewichte. Sie sind kontinuierlich, und erlauben damit keine zu schnellen Algorithmen, die für diskrete Variablen existieren. Eine Idee ist es, die Wertemengen der einzelnen Gewichte zu diskretisieren. Wir betrachten nur noch die Werte 0 oder 1 für die einzelnen Gewichte, so dass eine Merkmalsauswahl erzeugt wird. Der Wert 0 für ein bestimmtes Gewicht bedeutet, dieses Merkmal wird nicht bei der Berechnung des nächsten Clusterings betrachtet und der Wert 1 bedeutet, dass dieses Merkmal mitbetrachtet wird. Die Domain D_i ist also in $\{0, 1\}$.

Der Zustand ist ein belegter Gewichtsvektor. Die einzelnen Gewichte dürfen wie oben beschrieben nur Werte in $[0, 1]$ annehmen.

Um die Constraints zu definieren, definieren wir eine Funktion:

```
boolean constraintErfuellt(element1, element2, mustlink?).
```

Die Parameter `element1` und `element2` sind die Elemente eines Must-Link oder Cannot-Link, `mustlink?` ist wahr, wenn dieses Paar ein Must-Link ist, für ein Cannot-Link ist es falsch. Diese Funktion gibt „wahr“ zurück, wenn im Falle eines Must-Links die beiden Elemente einem Cluster zugeordnet sind und im Falle eines Cannot-Links verschiedenen Clustern zugeordnet sind. Ansonsten gibt sie „falsch“ zurück. Die Constraints für unser CSP entsprechen den in die Funktion eingesetzten Must-Links und Cannot-Links.

Wenn man die Standard-Definition der CSPs betrachtet, fällt auf, dass *alle* Constraints erfüllt sein müssen. Wir haben schon darauf hingewiesen, dass die Constraints an sich widersprüchlich sein können, und dementsprechend auch die daraus gebildeten Paare. Außerdem haben die ursprünglichen Constraints eine Gewichtung, welche eine Teilerfüllung der Constraints zulässt. Es wäre also wünschenswert, ein CSP-Problem zu definieren, bei dem eine Lösung erzeugt werden kann, die die Anzahl der erfüllten Constraints maximiert. Dieser Ansatz wird in der Literatur *Partial Solution CSPs* [80] genannt.

Aber bevor man diesen Ansatz weiteverfolgt, fällt auf, dass die Merkmale nicht vollständig voneinander unabhängig sind, z.B. existiert das Merkmal: mittlere Lautstärke, aber auch die Varianz der Lautstärke und die absolute mittlere Lautstärke (siehe Kap. 3.1). Das bedeutet, wenn man ein Gewicht verändert, sollte man auch die anderen mitverändern, sonst ist das Ergebnis falsch. Wir wissen außerdem nicht, welche Auswirkungen die Gewichtsveränderungen auf die Cluster haben können. Das heißt, wir

können keine Suchrichtung für den Algorithmus bestimmen. Wir können nur nach Versuch und Irrtum vorgehen, was sehr viel Zeit in Anspruch nehmen würde, zumal wir jedesmal ein Clusteringalgorithmus und die Evaluierungsroutinen starten müssen, um die Auswirkungen der Gewichtsveränderungen zu überprüfen.

Man erkennt, dass unser Problem, obwohl es einen änlich klingenden Namen hat, nicht gut als Constraint Satisfaction Problem formuliert werden kann. Man kann dabei keines der Vorteile der CSPs ausnutzen. Da CSPs im allgemeinen NP-Vollständig sind [73], können wir nicht mit einer effizienten Lösung rechnen und müssen somit anerkennen, dass dieser Ansatz für ein interaktives System ungeeignet ist (Siehe dazu Kap. 7).

6.7.4 Support Vector Machines (SVMs)

»... the story of the sheep dog who was herding his sheep, and serendipitously invented both large margin classification and Sheep Vectors... «
Einleitung von [58]

Support Vector Machines (SVMs) sind Funktionen, die eine optimale Trennebene zwischen den Daten aus verschiedenen Klassen im Merkmalsraum bestimmen. Diese Disziplin wurde nicht vor allzu langer Zeit für die Praxis wiederentdeckt und erstmals von V. Vapnik [68] vorgestellt. In diesem Kapitel werden wir die Must-Links und Cannot-Links in eine Eingabe für die SVM transferieren, so dass wir als Ausgabe einen Gewichtsvektor erhalten, welcher zu einem constraintkonformeren Clusteringergebnis führt.

Zunächst einmal müssen wir genauer auf den Aufbau einer SVM eingehen[5]. Angenommen, wir haben *Instanzen* (auch *Fälle*, *Eingaben* oder *Beobachtungen* genannt) in einem Merkmalsraum (*Domain*), $X = \{x_1, \ldots, x_n\}$, und deren Zuweisung zu einer der beiden Klassen $y_i = +1$ oder $y_i = -1$. Die y_i werden *labels* oder *targets* genannt. Formal erhält man die folgende Konstruktion:

$$(x_1, y_1), \ldots, (x_m, y_m) \in X \times \{\pm 1\} \tag{6.11}$$

Das Ziel der SVM ist es, diese Separierung auf neue Instanzen zu *generalisieren*. Das bedeutet, sie muss eine Funktion erlernen, die einem neuen Punkt x_{n+1} ein Label $y_{n+1} \in \{\pm 1\}$ zuweist. Dafür müssen wir ein Ähnlichkeitsmaß (engl. *similarity*) der Instanzen untereinander definieren, also bestimmen, wann zwei Objekte der gleichen und wann sie verschiedenen Klassen angehören. Die Ähnlichkeit ist folgendermaßen definiert:

$$k : X \times X \rightarrow \mathbb{R} \tag{6.12}$$
$$(x, x') \mapsto k(x, x') \tag{6.13}$$

Es ist also eine Funktion, die Instanzen x und x' auf eine reelle Zahl, die die Ähnlichkeit ausdrückt, abbildet. Diese Funktion k wird *Kernel* genannt. Das einfachste Ähnlichkeitsmaß ist das *Standardskalarprodukt*, welches für zwei Vektoren $\mathbf{x},\ \mathbf{x}' \in \mathbb{R}^n$ folgendermaßen definiert ist:

$$\langle \mathbf{x}, \mathbf{x}' \rangle = \sum_{i=1}^{n} [\mathbf{x}]_i [\mathbf{x}']_i \tag{6.14}$$

$[\mathbf{x}]_i$ ist dabei die i-te Instanz des Vektors $\mathbf{x}$.

Die Instanzen müssen allerdings nicht in einem Raum sein, in dem das Skalarprodukt gilt. In diesem Fall müssen wir unsere Instanzen in einen Raum $\mathcal{H}$, den sogenannten Merkmalsraum, abbilden, in dem das Skalarprodukt gilt. Wir benutzen also die Abbildung:

$$\Phi : X \rightarrow \mathcal{H} \tag{6.15}$$
$$x \mapsto \mathbf{x} = \Phi(x). \tag{6.16}$$

[5] Die folgenden Ausführungen basieren auf dem Buch von Schoelkopf [58]

Auch wenn unsere Instanzen sich schon in einem Raum befinden, in dem das Skalarprodukt gilt, kann es sinnvoll sein trotzdem eine Abbildung in einen allgemeineren Raum $\mathcal{H}$ durchzuführen. Die Abbildung Φ braucht dabei nicht unbedingt eine lineare Abbildung zu sein.

Aus diesen Überlegungen können wir nun ein Ähnlichkeitsmaß in $\mathcal{H}$ definieren:

$$k(x, x') = \langle \mathbf{x}, \mathbf{x}' \rangle = \langle \Phi(x), \Phi(x') \rangle \,. \tag{6.17}$$

Diese Herangehensweise erlaubt uns mit den Instanzen auf eine geometrische Weise mit den Werkzeugen der linearen Algebra und analytischer Geometrie zu verfahren.

Wir gehen zunächst einmal davon aus, dass unsere Daten linear trennbar sind, das bedeutet, dass es eine lineare Hyperebene existiert, die mit dem Label $y = +1$ von den Daten mit dem Label $y = -1$ in dem Raum $\mathcal{H}$ separiert. Dies ist keine Beschränkung, denn nur der Raum $\mathcal{H}$ muss linear separierbar sein, der ursprüngliche Merkmalsraum kann auch durch andere Funktionen, wie z.B. eine Parabel oder eine exponentielle Funktion separierbar sein. Es reicht den Ursprungsraum mit der Hilfe der oben erwähnten Funktion Φ so auf $\mathcal{H}$ abzubilden, dass die Daten linear separierbar werden. Diese Argumentation wird in der Abbildung 6.4 verdeutlicht. Es existieren eine ganze Reihe von sogenannten *Kernfunktionen*, die für verschiedene Separierungsprobleme das Gewünschte leisten.

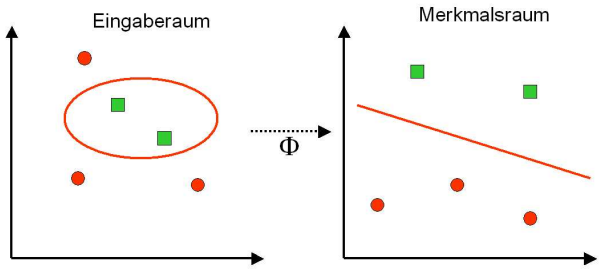


Abbildung 6.4: **Die Idee der SVM.** Die Trainingdaten werden in den Merkmalsraum mit Φ abgebildet und eine Trennhyperebene konstruiert. Dies führt zu einer nicht linearen Entscheidungsgrenze in dem Eingaberaum. Mit einer Kernfunktion ist es möglich eine Trennhyperebene zu konstruieren, ohne explizit die komplette Abbildung in den Merkmalsraum durchzuführen.

Die Hyperebene ist folgendermaßen definiert:

$$\langle \mathbf{w}, \mathbf{x} \rangle + b = 0, \text{ wobei } \mathbf{w} \in \mathcal{H}, b \in \mathbb{R} \tag{6.18}$$

Diese Formel entspricht der Entscheidungsfunktion:

$$f(x) = \text{sign}\left(\langle \mathbf{w}, \mathbf{x} \rangle + b\right) \tag{6.19}$$

$\mathbf{w}$ ist die Normale auf der Trennebene und x ist die Position eines Individuums. Die **optimale Hyperebene** ist die Lösung von:

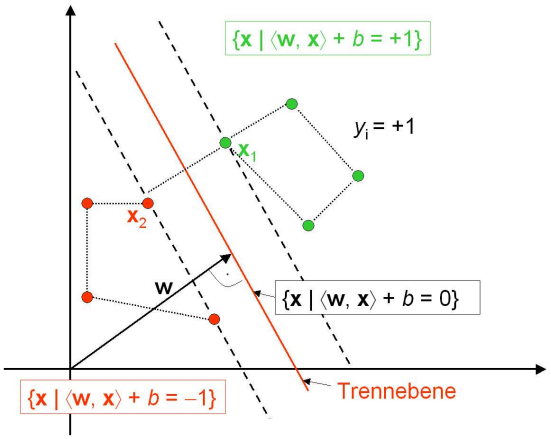
$$\text{maximize}_{\mathbf{w} \in \mathcal{H}, b \in \mathbb{R}} \min \{ \|\mathbf{x} - \mathbf{x_i}\| \;\; | \mathbf{x} \in \mathcal{H}, \langle \mathbf{w}, \mathbf{x} \rangle + b = 0, \; i = 1, \ldots, n \} \tag{6.20}$$

Diese Herangehensweise ist ebenfalls attraktiv aus der Implementierungssicht, denn sie entspricht dem *quadratischen Programmieren*, für welches effiziente Lösungsalgorithmen existieren [58].

Die Abbildung 6.5 veranschaulicht diese Formel.

Bevor wir allerdings die optimale Hyperebene berechnen können, müssen wir den Normalenvektor $\mathbf{w}$ der Hyperebene bestimmen, indem wir die folgende Gleichung lösen:

$$\text{minimize}_{w \in \mathcal{H}, b \in \mathbb{R}} \, \tau(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\|^2, \text{ wobei } y_i(\langle \mathbf{w}, x_i \rangle + b) \geq 1 \; \forall i = 1, \ldots, n. \tag{6.25}$$



$$\langle \mathbf{w}, \mathbf{x_1} \rangle + b = +1 \tag{6.21}$$

$$\langle \mathbf{w}, \mathbf{x_2} \rangle + b = -1 \tag{6.22}$$

$$\Rightarrow \quad \langle \mathbf{w}, (\mathbf{x_1} - \mathbf{x_2}) \rangle = 2 \tag{6.23}$$

$$\Rightarrow \quad \left\langle \frac{\mathbf{w}}{||\mathbf{w}||}, (\mathbf{x_1} - \mathbf{x_2}) \right\rangle = \frac{2}{||\mathbf{w}||} \tag{6.24}$$

Abbildung 6.5: **Die Bildung einer Hyperebene.** Die optimale Hyperebene ist die rot dargestellte Gerade. Da das Problem separierbar ist, existiert ein Vektor $\mathbf{w}$ und ein Grenzwert b, so dass $y_i(\langle \mathbf{w}, x_i \rangle + b) > 0$ $(i = 1, \ldots, n)$. Wenn man $\mathbf{w}$ und b so wählt, dass die nächsten Punkte zu der Hyperebene $|\langle \mathbf{w}, x_i \rangle + b| = 1$ erfüllen, so erhält man eine *kanonische Form* $(\mathbf{w}, b)$ der Hyperebene und es gilt $y_i(\langle \mathbf{w}, x_i \rangle + b) \geq 1$. In diesem Fall ist der Abstand (engl. *margin*), also die Distanz zu dem nächsten Punkt an der Hyperebene, gleich $\frac{1}{\|\mathbf{w}\|}$. Dies kann gezeigt werden, indem man zwei Punkte x_1, x_2 annimmt, welche an den verschiedenen Seiten des Randes liegen, so dass gilt $\langle \mathbf{w}, x_1 \rangle + b = 1$, $\langle \mathbf{w}, x_2 \rangle + b = -1$, und diese auf den Normalvektor $\frac{\mathbf{w}}{\|\mathbf{w}\|}$ der Hyperebene projiziert.

Die Bedingung „≥ 1“ auf der rechten Seite der Nebenbedingungen fixiert die Skalierung von $\mathbf{w}$. Theoretisch würde dies mit jeder beliebigen positiven Zahl funktionieren.

Die Funktion τ wird *Zielfunktion* und die Bedingungen werden *Ungleichheits-Nebenbedingungen* genannt. Zusammen ergeben sie ein sogenanntes *Beschränktes Optimierungsproblem*. Solche Probleme werden üblicherweise mit der Methode der *Lagrange Multiplikatoren* $\alpha_i \geq 0$ und der sogenannten *Lagrangefunktion*

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i(\langle \mathbf{w}, x_i \rangle + b) - 1 \right), \tag{6.26}$$

gelöst. Die Lagrangefunktion muss nach ihren Hauptvariablen $\mathbf{w}$ und b minimiert werden und nach den dualen Variablen α_i maximiert werden. Es ist also eine Suche nach dem Sattelpunkt der Funktion. Der Ergebnisvektor $\mathbf{w}$, den wir erhalten, ist gleich $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i x_i$. Er hängt offensichtlich nur von einer Untermenge der Trainingsindividuen ab, nämlich von solchen mit $\alpha_i \neq 0$. Diese werden *Support Vectors* genannt und liegen nach ihrer Definition genau auf dem Rand (gestrichelte Geraden auf der Abbildung 6.5). Da die Hyperebene durch die ihr am nächsten liegenden Individuen determiniert wird, hängt die Lösung nicht von den übrigen Individuen ab.

Wenn wir nun die Lösung für $\mathbf{w}$ in die Lagrangefunktion integrieren, dann erhalten wir das sogenannte *duale Optimierungsproblem*:

$$\text{maximize}_{\alpha \in \mathbb{R}^n} W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle, \text{für } \alpha_i \geq 0, \ \forall i = 1, \ldots, n \text{ und } \sum_{i=1}^{n} \alpha_i y_i = 0. \tag{6.27}$$

Dies führt mit dem Einsetzen der Kernfunktion zu der folgenden Entscheidungsfunktion:

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} y_i \alpha_i \langle \Phi(x), \Phi(x_i) \rangle + b \right) = \text{sign}\left(\sum_{i=1}^{n} y_i \alpha_i k(x, x_i) + b \right) \tag{6.28}$$

und zum folgenden quadratischen Programm:

$$\text{maximize}_{\alpha \in \mathbb{R}^n} W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j k(x_i, x_j), \text{für } \alpha_i \geq 0,\ \forall i = 1, \ldots, n \text{ und } \sum_{i=1}^{n} \alpha_i y_i = 0. \tag{6.29}$$

Dieses Programm bildet die Basis für alle Implementierungen der SVMs. Als einen Einstieg in die SVM sollte dieses Kapitel genügen. Die SVMs bieten aber weitaus mehr, denn z.B. wurden mittlerweile eine ganze Reihe an verschiedenen Kernfunktionen entwickelt und angewandt, so dass für die meisten Anwendungen eine entsprechende Funktion bereitsteht. Durch die Einführung von sogenannten *Slackvariablen* ist es möglich auch gut mit verrauschten Daten zu arbeiten. Und es existieren Ansätze für *Support Vector Regression*. Der an diesen und vielen Einzelheiten der SVMs interessierte Leser sei auf [58] verwiesen.

Es wird ebenfalls auf eine Erläuterung der Implementierung der SVM verzichtet, denn es existieren genügend effiziente Implementierungen, unter anderem in der Experimentierumgebung YALE (siehe Kap. 6.8), auf welche diese Arbeit aufsetzt. Es genügt, wenn wir eine Eingabe für die SVM aus den Vorgaben unserer Constraints konstruieren können. Dies wird in dem nächsten Abschnitt genauer erläutert.

Modifikation der Constraints als Eingabe für die SVMs

Wir betreiben eine Parameteroptimierung für die SVMs. Die Parameter müssen unseren Gewichtsvektor enthalten. Allerdings müssen wir die Eingabe für die SVM so weit modifizieren, dass eine Trennebene zwischen den positiven und negativen Beispielen gezogen werden kann. Alternativ könnte man direkt versuchen unser Problem in eine SVM umzuwandeln, also z.B. die anderen, in dieser Arbeit vorgestellten Verfahren, zu „kernelisieren“. Dies erweist sich aber als schwierig, denn viele Kernel (wie z.B. der Gauss-Kernel) vergrößern die Dimensionalität des Merkmalsraums evtl. bis zur Unendlichkeit [66]. Nichtdestotrotz existieren in der Literatur diesbezügliche Ansätze (z.B. [25], [74]), die aber nur eine Merkmalsauswahl in dem Eingaberaum vornehmen und nicht in dem Merkmalsraum [66]. Andererseits gibt es auch Ansätze, die auch diese Hürde überwinden und sowohl im Eingaberaum als auch im Merkmalsraum eine Merkmalsauswahl treffen können [66]. Unser Ansatz basiert zwar auf ähnlichen Ideen wie in [66], allerdings versuchen wir keine neue SVM zu entwickeln, sondern eine Eingabe für eine bestehende Implementierung einer SVM[6] zu generieren.

Das Problem bei der Eingabe ist, dass Must-Links und Cannot-Links aus jeweils zwei Elementen bestehen, die erst durch die Relationen für unser Vorgehen einen Sinn ergeben. Wir müssen also die Punktepaare auf eindeutige Zahlenwerte abbilden. Da kommt uns die Distanz zwischen den Punkten gerade recht. Wir betrachten allerdings die quadratische Distanz, um nicht bei der Berechnung mit Wurzeln agieren zu müssen[7]. Für alle Paare von Elementen in der Menge S der Must-Links und der Menge D der Cannot-Links definieren wir:

$$\forall x, y \in S: \quad d^2(x, y) < 1, \tag{6.30}$$

$$\forall x, y \in D: \quad d^2(x, y) > 1 \tag{6.31}$$

Das bedeutet, dass alle Paare $x,\ y$, die ein Must-Link erfüllen, eine quadratische Distanz kleiner als 1, während die Cannot-Links eine quadratische Distanz größer als 1 haben sollen. Statt 1 könnte man eine beliebige Konstante c als Trennpunkt annehmen.

Durch Umformungen erhalten wir:

$$\forall x, y \in S: \quad d^2(x, y) - 1 < 0, \tag{6.32}$$

$$\forall x, y \in D: \quad d^2(x, y) - 1 > 0 \tag{6.33}$$

[6] Wir benutzen die in der Entwicklungsumgebung YALE angebotene Implementierung, `JMySVM`, von Stefan Rüping. Siehe Kap. 6.8.

[7] Siehe dazu die Definition der Distanz, Kap. 6.3.

Sei $S_p = (x, y)$ das p-te Must-Link und $D_p = (x, y)$ das p-te Cannot-Link. Dann können wir unsere Überlegungen folgendermaßen darstellen:

$$\forall S_p \in S: \quad d^2(S_p) - 1 < 0, \tag{6.34}$$

$$\forall D_p \in D: \quad d^2(D_p) - 1 > 0 \tag{6.35}$$

Nun folgt die Annahme über den Merkmalsraum:

$$d^2(S_p) - 1 = \sum_i \alpha_i y_i k(x_i, S_p) + b, \tag{6.36}$$

womit wir eine Entscheidungsfunktion $f(x)$ aus der Gleichung 6.28 erhalten. Für alle D_p gilt die äquivalente Annahme. Diese Funktion können wir dann mit der Abbildungsfunktion ϕ folgendermaßen schreiben:

$$d^2(S_p) - 1 = \sum_i \alpha_i y_i \Phi(x_i)\phi(S_p) + b. \tag{6.37}$$

Wähle

$$\Phi(S_p) = \phi(\mathbf{x}, \mathbf{y}) = \begin{pmatrix} (x_1 - y_1)^2 \\ \dots \\ (x_H - y_H)^2 \end{pmatrix} = \mathbf{p} \cdot \phi(S_p) + b \tag{6.38}$$

Die quadratische Distanz wird aus der Definition im Kapitel 6.3 folgendermaßen abgeleitet:

$$d^2(x_1, x_2) = \sum_{h=1}^{H} p_h \cdot (x_i - x_j)^2 . \tag{6.39}$$

Damit konstruieren wir für alle Paare (x_i, x_j) eine Abbildung in den Merkmalsraum ϕ :

$$(x_i, x_j) \rightarrow \phi(x_i, x_j) = (x - y)^2 \tag{6.40}$$

Damit haben wir die Verbindung von SVMs zu der Distanz hergestellt und haben eine Begründung für unsere Vorgehensweise gefunden.

Konkret bilden wir ein neues Itemset aus den Constraint-Paaren, wobei wir für jedes der Merkmale die quadratische Distanz verwenden. Zusätzlich führen wir einen Indikatormerkmal ein, der für jedes neue Item eine 1 zurückliefert, wenn es ein Must-Link ist und eine -1, wenn es ein Cannot-Link ist. Die Tabelle 6.1 zeigt diesen Aufbau genauer. Aus diesen Beispielen kann eine SVM eine optimale Gewichtung für die optimale Trennebene berechnen.

Merkmale:	1	2	...	M-1	M	Indikator
ConstraintPair 1 ($d_1^2(x_i, y_j)$):	0.3	0.01	...	0	0.24	-1
ConstraintPair 2 ($d_2^2(x_k, y_l)$):	0.1	0.2	...	0	0.24	1
...						

Tabelle 6.1: **Eingabevektor für die Gewichtsoptimierung mit der SVM**

Bei diesem Ansatz bekommen die Must-Links, die anzeigen, dass ein Element in das gleiche Cluster gehört, wie er selbst eine wichtige Rolle. Denn würden wir sie nicht mitberücksichtigen, würden bei der erwarteten gleichen Anzahl von Must-Links und Cannot-Links immer eine Trennebene von 45° ergeben, was für ein anschließendes Clustering keine Verbesserung ergeben würde.

Ein Beispiel soll dies verdeutlichen: Wir haben vier Punkte, so wie auf der Abbildung 6.6 a) dargestellt. Die Must-Links sind genau komplementär zu den Cannot-Links. Wenn wir nun die Distanzabbildung aus den Gleichungen 6.30 und 6.31 anwenden, erhalten wir die Abbildung 6.6 b). Und hier sehen wir, dass, wenn wir die Must-Links mit gleichen Elementen nicht beachten, sich eine recht langweilige Trennebene von 45° ergibt. Diese ist durch die gestrichelte Linie dargestellt. Wenn die Must-Links mit den gleichen Elementen aber dazu kommen, erhalten wir eine bessere Trennebene, die eine Verbesserung durch die Modifikation der Gewichte zulässt.

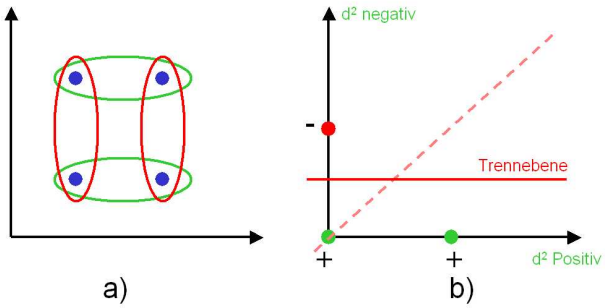


Abbildung 6.6: **Ein Beispiel zur Vorgehensweise für die SVM.** a) Zeigt vier Punkte in einem zweidimensionalen Merkmalsraum. Die grünen Ellipsen sind die Must-Links und die roten die Cannot-Links. b) Zeigt die Abbildung der links nach der Modifikation der Distanzen in den Distanzraum. Die positiven Beispiele sind grün und die negativen rot. Wenn man die Must-Links mit zwei gleichen Elementen außer Acht lässt, dann entsteht die gestrichelte Trennebene. Wenn sie aber hinzukommen, dann entsteht die rote Trennebene.

6.7.5 Distance Metric Learning-Verfahren

In diesem Abschnitt wollen wir ein weiteres Verfahren zur Optimierung der Gewichte vorstellen. Da durch die Manipulation der Gewichte sich die Distanzen ändern, können wir unsere Ansätze als das Lernen einer constraintkonformen Distanzmetrik verstehen. In [78] wird ein Ansatz präsentiert, der eine Distanzmetrik in $\mathbb{R}^n$ lernt[8]. Das Problem wird in ein konvexes Optimierungsproblem (*konvexe Programmierung* [66]) umgewandelt, so dass wir uns um lokale Optima keine Gedanken machen müssen.

Zunächst wird die Distanz neu definiert, um aus diesen Ergebnissen besser die Vorgehensweise veranschaulichen zu können. Alle in dieser Arbeit vorgestellten Verfahren lernen eine komplette Metrik $d : \mathbb{R}^n \times \mathbb{R}^n :\mapsto \mathbb{R}$ im Gegensatz zu anderen Verfahren, wie z.B. *Multidimensional Scaling (MDS)* [13], die sich nur auf eine Umgebung der Punkte in dem Trainingsraum konzentrieren.

Die Euklidische Distanz (Kap. 6.3) kann man auch als ein Spezialfall der Mahalonobis-Distanz, ansehen:

$$d(x,y) = d_A(x,y) = \|x-y\|_A = \sqrt{(x-y)^T A(x-y)} \tag{6.41}$$

Dabei ist A eine positiv definite Matrix, die unsere Merkmalswerte modifiziert. Wenn man $A = I$ setzt, dann ist es die Euklidische Distanz ohne Gewichte. Wenn wir zusätzlich von der Matrix nur die Diagonale A_{ii} betrachten, sind es genau unsere Gewichte, mit denen wir alle Werte eines Merkmals über alle Musikstücke auf einmal modifizieren.

Es stellt sich die Frage, ob eine komplette (Dreiecks-)Matrix bessere Ergebnisse liefern könnte als nur die Diagonalelemente. Immerhin werden dort Gewichte für alle Merkmale für jedes einzelne Paar von Musikstücken gelernt. Es ist zunächst der ungleich größerer Aufwand, sowohl beim Aufbau und der Verwaltung der Matrix als auch beim Lernen der Gewichte, der uns negativ ins Auge fällt. Andererseits ignoriert die ausschließliche Verwendung von Diagonalelementen einer Matrix die möglichen Korrelationen zwischen den Merkmalen [78]. In [78] wurde aber festgestellt, dass die Ergebnisse einer kompletten Matrix sich trotzdem nicht signifikant von den Ergebnissen mit nur Diagonalelementen unterscheiden. Hier haben wir einen Beleg dafür, dass unser Ansatz mit der globalen Gewichtung seine Berechtigung hat.

Sei S die Menge aller Must-Links, D die Menge aller Cannot-Links. Um eine erwünschte Metrik zu erhalten ist es am einfachsten, für die Punkte $(x_1, x_2) \in S$ eines Must-Links eine kleine quadratische Distanz zu fordern. Das Minimum wird allerdings auch mit $A = 0$ erreicht, was nicht unser Ziel ist, denn dann wären alle Musikstücke einander gleich. Wir müssen also zusätzliche Nebenbedingungen

[8] Die folgenden Ausführungen basieren hauptsächlich auf [78].

einführen, um zu verhindern, dass beim Lernen A auf einen einzigen Punkt kollabiert. Diese besagen, dass die Elemente Cannot-Links $(x_1, x_2) \in D$ einen Mindestabstand zu einander haben müssen. Insgesamt haben wir das folgende Optimierungsproblem zu lösen, q ist dabei das Gewicht des jeweiligen Constraint-Paares.

$$\begin{array}{ll} \min\limits_{A} & \sum_{(x_i,x_j)\in S} q \cdot \|x_i - x_j\|_A^2 \\ N.B. & \sum_{(x_i,x_j)\in D} q \cdot \|x_i - x_j\|_A \geq 1 \\ & A \succeq 0. \end{array} \tag{6.42}$$

Die Wahl der Konstante 1 ist beliebig, wenn man diese durch eine beliebige Konstante c ersetzt, wird A nur durch $c^2 A$ ausgetauscht. Es ist leicht zu verifizieren, dass beide Bedingungen dieses Problems konvex sind [78].

Unser Ziel ist es die optimale Diagonale $A = \mathrm{diag}(A_{11}, A_{22}, \ldots, A_{nn})$ zu lernen. Dazu definieren wir die folgenden Funktion $g(A)$:

$$g(A) = g(A_{11}, \ldots, A_{nn}) = \sum_{(x_i,x_j)\in S} q \cdot \|x_i - x_j\|_A^2 - log\left(\sum_{(x_i,x_j)\in D} q \cdot \|x_i - x_j\|_A\right). \tag{6.43}$$

In [78] wird festgestellt, dass die Minimierung von $g(A)$ bzgl. $A \succeq 0$ äquivalent ist mit der Multiplikation von A mit einer positiven Konstante, um das Originalproblem (Gleichung 6.42) zu lösen.

Um $g(A)$ effizient zu optimieren verwenden wir das *Newton-Raphson* Iterationsverfahren. Dieses ist folgendermaßen definiert:

DEFINITION 15: (Das Newton-Raphson-Verfahren[9])
Sei $g(x)$ eine stetige und zweimal differenzierbare Funktion, $g'(x)$ die erste und $g''(x)$ die zweite Ableitung von $g(x)$. Sei $g''(x) \neq 0$. x_0 sei ein beliebiger Startpunkt. Die Iteration

$$x_{i+1} = x_i - \frac{g'(x_i)}{g''(x_i)} \tag{6.44}$$

nähert sich einem lokalen Nullpunkt von $g'(x)$, der gleichzeitig ein Extrempunkt (entweder Maximum oder Minimum) von $g(x)$ ist. Diese Vorgehensweise wird ***Newton-Raphson-Verfahren*** *genannt und hat folgende Eigenschaften:*

- *Das Verfahren konvergiert im Allgemeinen nur bei einem gut gewählten Startpunkt.*
- *Es hat eine durchschnittliche quadratische Konvergenzgeschwindigkeit. Bei mehreren Nullstellen konvergiert sie aber entsprechend langsamer.*
- *Ist konvergent für beliebige Startpunkte x_0 gegen eine eindeutige Lösung s in einem geschlossenen Intervall $I = [a, b]$, wenn folgende Bedingungen gelten:*
  1. $g'(a) \cdot g'(b) < 0$
  2. $g''(x) \neq 0,\ \forall x$
  3. *Entweder* $g'''(x) \leq 0$ *oder* $g'''(x) \geq 0,\ \forall x$
  4. *Sei c derjenige Randpunkt des Intervalls I, für den $|g''(x)|$ den kleineren Wert hat, so ist*

$$\left|\frac{g'(c)}{g''(c)}\right| \leq b - a. \tag{6.45}$$

Um $g(A)$ minimieren zu können, müssen wir also zunächst die erste und die zweite Ableitung bestimmen. Die Ableitung wird für jeweils ein Diagonalelement A_{ii} durchgeführt und diese werden, wie im Folgenden genauer erläutert wird, einzeln optimiert.

Aus der Gleichung 6.41 folgt für ein einzelnes A_{ii} :

$$\|x-y\|_{A_{ii}} = \sqrt{(x_i-y_i)^2 \cdot A_{ii}} = (x_i-y_i)\cdot\sqrt{A_{ii}}. \tag{6.46}$$

Und daraus folgt für $g(A)$:

$$g(A) = \sum_{(x_i,x_j)\in S}\sum_{k=1}^{n} q\cdot(x_{i_k}-x_{j_k})^2 A_k - \log\left(\sum_{(x_i,x_j)\in D}\sum_{k=1}^{n} q\cdot(x_{i_k}-x_{j_k})\sqrt{A_k}\right). \tag{6.47}$$

Die erste Ableitung von $g(A_p)$ für das p-te Diagonalelement ist:

$$g'(A_p) = \sum_{(x_i,x_j)\in S} q\cdot(x_{i_p}-x_{j_p})^2 - \frac{\sum_{(x_i,x_j)\in D}\frac{q\cdot(x_{i_p}-x_{j_p})}{2\sqrt{A_p}}}{\sum_{(x_i,x_j)\in D}\sum_{k=1}^{n} q\cdot\sqrt{A_k}(x_{i_k}-x_{j_k})}. \tag{6.48}$$

Die zweite Ableitung ist dann entsprechend:

$$g''(A_p) = \frac{\sum_{(x_i,x_j)\in D}\frac{q\cdot(x_{i_p}-x_{j_p})}{4A_p\sqrt{A_p}}\sum_{(x_i,x_j)\in D}\sum_{k=1}^{n} q\cdot\sqrt{A_k}(x_{i_k}-x_{j_k}) + \left(\sum_{(x_i,x_j)\in D}\frac{q\cdot(x_{i_p}-x_{j_p})}{2\sqrt{A_p}}\right)^2}{\left(\sum_{(x_i,x_j)\in D}\sum_{k=1}^{n} q\cdot\sqrt{A_k}(x_{i_k}-x_{j_k})\right)^2}. \tag{6.49}$$

Bleibt noch zu klären, wie wir aus der Optimierung für die Einzelgewichte (einzelne Diagonalelemente) die Optimierung für den gesamten Vektor bekommen. Dies erreichen wird dadurch, dass wir die Gewichte zunächst einzeln optimieren, ohne den Gewichtsvektor zu verändern. Erst danach werden alle Gewichte auf einmal aktualisiert und die nächste Newton-Iteration gestartet. Diese Vorgehensweise wird in dem Algorithmus 3 im Detail ausgeführt. Dabei ist ϵ eine Abbruchbedingung, die festlegt, wie nahe $|g'(A)|$ an 0 heranreichen soll. Da es oft sehr schnell möglich ist, einen in der Nähe der Null liegenden Wert zu erreichen, aber das tatsächliche Erreichen des Nullpunktes noch viele Schritte erfordert, erscheint die Einführung dieser Schranke sinnvoll. Obwohl wir aufgrund unserer Argumentation gute Ergebnisse erwarten können, hinkt dieses Verfahren vor allem an einer zu langsamen Konvergenz, wenn die Anzahl der Merkmale, wie es in unserem System der Fall ist, groß ist [66]

A^0 ist die Ausgangsgewichtung, in unserem Fall sind alle Gewichte am Anfang auf 1 gesetzt.

Algorithmus 3: Die Newton Methode

Require: A^0, Precision ϵ
  $A := A^0$,
  $A^{\text{old}} := A$.
  repeat
    for $\forall A_p, p \in \{1,\ldots,n\}$ **do**
      $A_p = A_p^{\text{old}} - \frac{g'(A_p^{\text{old}})}{g''(A_p^{\text{old}})}$
    end for
    $A^{\text{old}} := A$
  until $|g'(A)| \leq \epsilon$

[9] Nähere Informationen zu dem Verfahren sind unter anderem in [58] oder unter `http://www.uni-ulm.de/~s_rschw3/old/facharb/4.htm` zu finden.

6.7.6 Evolutionäre Algorithmen

Im Folgenden wollen wir eine randomisierte Suchheuristik betrachten, bei der wir weder eine gute Qualität der Lösung noch eine gute Rechenzeit garantieren können. Wir erwarten zumindest, dass unsere Lösungen lokal optimal sind, wobei „lokal“ fair definiert ist. Wenn die Nachbarschaft jedes Suchpunktes der ganze Suchraum ist, dann fallen die Begriffe lokal optimal und global optimal zusammen. Dennoch führen auch „vernünftige“ Lokalitätsbegriffe zu so großen Nachbarschaften, dass es nicht mehr effizient ist, die gesamte Nachbarschaft systematisch zu durchsuchen [73].

Die Natur hat bei der Entwicklung der Arten Evolution zur Optimierung eingesetzt. Wesentlicher Bestandteil ist dabei die geschlechtliche Vermehrung. Neue Objekte werden von anderen Objekten abgeleitet und erben deren Eigenschaften [73].

Die Evolutionäre Algorithmen sind ein Ansatz, welcher die Evolution der Lebewesen auf algorithmische Probleme abbildet, in der Hoffnung diese Probleme besser und vor allem schneller zu lösen. Dabei spielt das darvinistische Gesetz „*survival of the fittest*“ die wichtigste Rolle.

Für die Evolution benötigt man eine Population von Individuen, also Suchpunkten in dem zu durchsuchenden Raum. Der Suchraum ist meist durch $\mathbb{R}^H$ oder $\{0,1\}^H$ gegeben. Jeder Punkt x entspricht einem Vektor in diesem Raum [46].

DEFINITION 16: (Evolutionäre Algorithmen)
*Evolutionäre Algorithmen versuchen in einem **Suchraum** den optimalen Suchpunkt für die Lösung der Aufgabe zu finden. Die Suchpunkte, die ein evolutionärer Algorithmus verwaltet, nennt man **Individuen**; Verwaltet er mehrere gleichzeitig, so nennt man diese eine **Population**.*

Aus dieser Population können Nachfolgepopulationen, *Generationen* erzeugt werden. Dies kann auf drei Arten geschehen:

1. Die Individuen der alten Generation werden unverändert in die neue übernommen.
2. Mutation
3. Kreuzung (engl. *Crossover*)

Der erste Punkt bedarf einer Erläuterung. Wenn die Individuen der alten Generation gut genug sind, um mit der jüngeren Generation mitzuhalten, spricht nichts dagegen sie weiter in der Population beizubehalten. Es gibt die Option, das beste Individuum in jedem Fall zu übernehmen (*elitist selection*) oder sogar die k besten Individuen.

Die *Mutation* ist, genauso wie bei dem biologischen Vorbild, die fehlerhafte Replikation der Individuen. In der Evolution haben Mutationen die genetische Vielfalt erhöht. Mutanten mit geringerer Fitness starben aus, während erfolgreiche Mutanten im Laufe der Zeit die ganze Population beeinflussten [73]. Bei der Mutation werden einzelne Bereiche des Vektors eines Individuums verändert, so dass prinzipiell auch von dem Elter ganz unterschiedliche Nachfolger erzeugt werden können. Allerdings sollte die Wahrscheinlichkeit einen dem Elter ähnlichen Nachfolger zu erhalten größer sein, als die für einen unähnlichen.

DEFINITION 17: (Mutation)
*Unter **Mutationen** verstehen wir alle Operationen, die aus einem Individuum ein anderes ableiten.*

Die *Kreuzung* oder auch *Crossover* ist eine Erzeugung von neuen Individuen aus zwei[10] Elternteilen, indem man für jedes Merkmal des neuen Individuums zufällig entweder den Wert von dem einen oder den von dem anderen Elternteil übernimmt. Das neue Individuum kann nicht mehr wie bei der Mutation zu den Eltern beliebig unähnlich sein, allerdings, wenn die Eltern schon relativ ähnlich sind, kann es sein, dass das Kind keinen genügend großen Unterschied aufweisen kann, um näher an das Optimum zu kommen.

[10] Üblicherweise sind es nach dem biologischen Vorbild zwei Individuen. Prinzipiell spricht aber nichts dagegen, auch mehr als zwei Eltern zu benutzen.

DEFINITION 18: (Kreuzungen)
*Unter **Kreuzungen** oder auch **Crossover** verstehen wir alle Operationen, die aus zwei oder mehreren Individuen ein anderes ableiten.*

Es wird nicht die gesamte Population für die Bildung der Nachkommen ausgewählt, sondern nur die „fittesten" Individuen kommen dafür, genauso wie in der Natur, in Frage. Um zu bewerten, wie „fit" das einzelne Individuum ist, benötigt man eine *Fitnessfunktion*, die die Eigenschaften der Individuen auf eine reelle Zahl, meistens auf das Intervall $[0, 1]$, abbildet. Je näher ein Individuum sich einem Optimum befindet, desto größer ist der Wert der Fitnessfunktion.

Der Vorgang der Auswahl der Eltern für die Bildung der neuen Population wird *Selektion* genannt.

DEFINITION 19: (Selektion)
***Selektion** bezeichnet die Klasse von Verfahren, die zur Auswahl von Individuen für die Nachfolgepopulation benutzt werden. Häufig gilt, dass Individuen mit einer höheren Fitness mit einer größeren Wahrscheinlichkeit selektiert werden.*

Die Auswahl der Startpopulation ist kritisch, da bei zu ähnlichen Individuen der Algorithmus in lokalen Extrema hängen bleiben kann. Andererseits, da man nichts über den Merkmalsraum weiß, ist es schwierig eine geeignete weit verstreute Population zu erzeugen. Es bleibt uns nichts anderes übrig, als die Population zufällig und gleichverteilt zu erzeugen. Um die Ergebnisse zu verbessern, könnte man mit mehreren verschiedenen Startpopulationen starten.

Auch die Anzahl der Individuen für eine Population ist kritisch, denn wenn man zu wenige Individuen auswählt, dann hat man nicht die nötige Diversität der Daten. Mit der steigenden Anzahl der Individuen aber steigt die Komplexität des Algorithmus. Die gleichen Überlegungen gelten auch für die Anzahl der von der Fitnessfunktion für die Fortpflanzung ausgewählter Individuen.

Die größe der Population sollte mit der Anzahl der Generationen nicht wachsen. Deswegen muss man bei der Erzeugung neuer Individuen wieder andere aus der Population entfernen. Es ist oft sinnvoll die Individuen mit der geringsten Fitness zu entfernen. Unter den Entfernten können sich sogar gerade erzeugte Individuen befinden, wenn sie nicht fit genug sind.

Wenn alle Bausteine des evolutionären Algorithmus spezifiziert wurden, lässt man die Fortpflanzungsmaschine laufen und braucht nichts weiter zu tun, als darauf zu warten, dass die Abbruchbedingung erfüllt ist. Diese kann, z.B. ein Individuum mit einer speziellen Fitness, eine Zeitbegrenzung oder die Anzahl der Generationen sein. Die Abbildung 6.7 zeigt die schematische Vorgehensweise der evolutionären Algorithmen. Bei der Frage nach der Laufzeit oder nach der Güte der Lösung müssen wir passen. Die Theorie Evolutionärer Algorithmen ist nicht so weit entwickelt, um deren Verhalten erklären zu können [73]. Allerdings erzielt man mit diesem Ansatz bei unterschiedlichsten Problemstellungen oft erstaunlich gute Resultate.

Der Suchraum für die Constraints

Wir müssen unser Problem geeignet kodieren, um es mit dem Ansatz der Evolutionären Algorithmen lösen zu können. Dabei ist die Laufzeit kritisch, so dass wir genauso wie in dem Constraint Satisfaction Ansatz uns auf eine Merkmalsauswahl beschränken, die Gewichte dürfen also nur noch 0 oder 1 annehmen.

Die Merkmalsgewichte sind die Variablen, die wir modifizieren dürfen. Die Constraints sind unser Kriterium, an dem wir die Güte der aktuellen Gewichtung evaluieren können.

Jeder Punkt in unserem Suchraum ist der in der Tabelle 6.2 dargestellte Attributsvektor, wobei M die Anzahl der Merkmale ist: Dabei werden die Nummern nicht abgespeichert, sondern nur die Gewichte.

MerkmalsNr. :	1	2	3		M-1	M
Gewicht :	0	0	1	...	1	0

Tabelle 6.2: **Der Merkmalsvektor.**

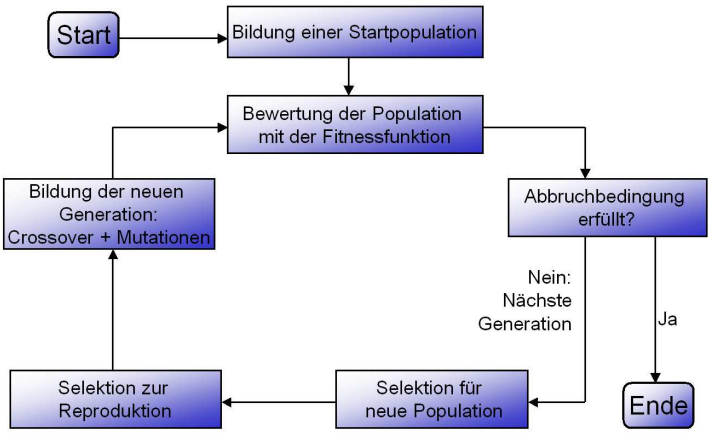


Abbildung 6.7: **Ablauf eines evolutionären Algorithmus.** Nachdem die Anfangspopulation spezifiziert wurde, erzeugt der Algorithmus aus einem nach der Fitnessfunktion ausgewählten Teil der Population fortwährend neue Generationen mit Hilfe der Kreuzung und der Mutation, bis eine Abbruchbedingung erfüllt ist.

Die Gewichte sind in dem Bereich $\{0, 1\}$. Dieser eine Gewichtsvektor beeinflusst die Positionen aller Musikstücke in dem Merkmalsraum.

Mutation

Die Mutation ist die zufällige Veränderung zufällig ausgewählter Gewichte.

Jedes der Gewichte kann mit der gleichen Wahrscheinlichkeit $\frac{1}{M}$ für die Mutation ausgewählt werden. Die erwartete Anzahl der modifizierten Gewichte für eine Mutation ist also 1.

Kreuzung

Die Kreuzung kann man so realisieren, dass man für zwei Individuen die Elemente der Gewichtsvektoren paarweise nacheinander betrachtet. Zufällig werden mit der gleichen Wahrscheinlichkeit die Gewichte für das neue Individuum entweder von dem einen Elter oder von dem anderen übernommen.

Die Wahrscheinlichkeit für die Auswahl des Gewichts zu beeinflussen, so dass die Gewichte des fitteren Individuums bevorzugt werden, ist auf den ersten Blick sehr verlockend. Allerdings gilt zu bedenken, dass gerade dies verhindern kann, dass ein lokales Optimum verlassen wird, wenn es erreicht wird. Es ist nämlich der große Vorteil der Kreuzungen, dass sie auch „Gräben“ in dem Merkmalsraum leicht überwinden können, im Gegensatz zu der Mutation, bei der die Wahrscheinlichkeit, ein lokales Optimum zu verlassen oft exponentiell klein ist. Als Beispiel wäre hierfür die *Real-Royal-Road-Funktion*, [73, S. 84], zu nennen. Die Tabelle 6.3 veranschaulicht die Vorgehensweise anhand eines Beispiels.

Ausgewählt:	(2)	(1)	(1)	...	(2)	(2)
MerkmalsNr. :	1	2	3		M-1	M
Gewicht (Individuum 1) :	0	0	1	...	1	0
Gewicht (Individuum 2) :	0	1	0	...	0	1
Gewicht (Neues Individuum):	0	1	1	...	0	1

Tabelle 6.3: **Ein Beispiel für die Kreuzung.** Die Individuen 1 und 2 werden gekreuzt. Zufällig wird für jedes Gewicht bestimmt, von welchem Individuum man es nehmen sollte. Zur besseren Übersicht wurde die Nummer dieses Individuums in der Zeile „Ausgewählt“ für jedes Gewicht eingetragen. Das neue Individuum enthält Elemente von beiden Individuen.

Die Fitnessfunktion

Ein Gewichtstupel ist umso fitter, je mehr Constraints es erfüllt. Die Evaluation der Constraints ist ausführlich in dem Kapitel 4.4 beschrieben.

Die Auswahl der Population

Die Anfangspopulation muss mit Bedacht gewählt werden, denn diese entscheidet über die Güte des Endergebnisses.

Eine simple Idee ist es, die Gewichtstupel gleichverteilt zufällig zu erzeugen. Dies ist immer eine gute Ausgangslage, die aber evtl. zu lange brauchen könnte, um in den richtigen Teil des Raumes zu gelangen.

Bei der Suche nach alternativen Auswahlmethoden haben wir nur wenige weiteren Anhaltspunkte für die Beschaffenheit des Merkmalsraums, denn, wie aus dem Kapitel 4.3 ersichtlich wird, sogar ein einzelnes Constraint schon widersprüchliche Vorgaben liefern kann.

Eine einfache Regel der Evolutionären Algorithmen besagt, dass man, wenn man gar nichts von der Struktur des Raumes weiß, mit simpleren Ansätzen mehr erreichen kann. Deswegen bleiben wir bei dem Ansatz mit der Gleichverteilung [73].

Bleibt noch zu klären, wie groß die Population sein soll, die wir erzeugen. Im Regelfall kommen bei Evolutionären Algorithmen mehrere hundert Individuen zum Einsatz. Allerdings haben wir, wie oben schon angesprochen, harte Laufzeitvorgaben und sollten deswegen die Population so klein wie möglich halten. Aus mehreren Testläufen hat sich eine Populationsgröße von 8 Individuen als sinnvoll erwiesen.

Nachdem ein neues Individuum erzeugt wurde, wird es in die Population eingefügt. Danach wird ein Individuum mit der schlechtesten Fitness aus der Population entfernt.

Populationen oder Multistarts?

Die Frage ist, ob man statt einer größeren Population nicht mehrere Durchläufe des Algorithmus favorisieren würde. Beim ersten Ansatz lässt man den Algorithmus auf einer Population laufen, bis die Abbruchbedingung erfüllt ist. Dann wählt man das Individuum mit dem besten Wert der Fitnessfunktion. In dem zweiten Ansatz startet man mehrere Anläufe mit verschiedenen Populationen und wählt von allen das beste Ergebnis aus. Diese Frage ist nicht eindeutig zu beantworten, jedoch sprechen einige Indizien dafür, dass die Multistarts bessere Ergebnisse liefern können [73, S. 80]. Mehrere Anläufe benötigen entsprechend mehr Laufzeit. Da aber eine einzige Iteration schon mehrere Stunden oder sogar Tage dauern kann, so erscheinen Multistarts für unser interaktives System nicht verwendbar, es sei denn man verfügte über genügend Rechenressourcen, so dass man die Iterationen parallel durchführen könnte.

6.8 Einbindung in die Enwicklungsumgebung YALE

Ein Ziel, welches in der Einleitung formuliert war, bestand darin das in diesem Kapitel vorgestellte System in einer Experimentierumgebung einzubinden. Dazu wurde YALE ausgewählt [45]. Zu dieser Entscheidung hat vor allem die Tatsache geführt, dass YALE logisch aufgebaut und einfach erweiterbar ist. Ein weiterer Grund war, dass in YALE schon viele Algorithmen und Lernmethoden aus dem Bereich *Maschinelles Lernen* implementiert waren und ohne Probleme für Experimente verwendet werden konnten. Und nicht zuletzt wurde diese Experimentierumgebung von dem Lehrstuhl, an dem diese Arbeit entstand, entwickelt, so dass man zu jeder Zeit dem maximalen Support bei auftretenden Enwicklungs- und Verständnisproblemen erhalten konnte.

Einige Verfahren, wie z.B. der Evolutionäre Algorithmus (*EvolutionaryWeighting*) waren in die Umgebung bereits integriert und konnten ohne Weiteres verwendet werden. Auch die SVM (*JMySVM*) war

schon eingebunden, es musste nur noch einen *Operator*[11] für die Vorverarbeitung (*ConstraintsSVMPreprocessing*) entwickelt werden. Für das Score-Verfahren wurde der Operator, *ScoreOptimization*, für den analytischen Ansatz der Operator, *AnalyticalOptimization*, und für den Distance Metric Ansatz mit dem Newton Verfahren der Operator, *DistanceMetricOptimization*. Des Weiteren wurde ein Operator für die Eingabe der Constraints erzeugt, *InteractiveConstraintClustering*. Dieser wird in den Abbildungen 6.9 und 6.10 ausführlich erläutert. Über allem thront eine *OperatorChain*, *ConstraintControlChain*, für die Realisierung des iterativen Verhaltens des Systems. Nicht zuletzt wurden weitere Operatoren für das Einlesen, die Vorbereitung der Daten und die Auswertung der Ergebnisse enweder aus dem riesigen Bestand von YALE mitverwendet oder hinzuimplementiert. Als Beispiel für den Aufbau des Systems in Yale soll die Abbildung 6.8 dienen.

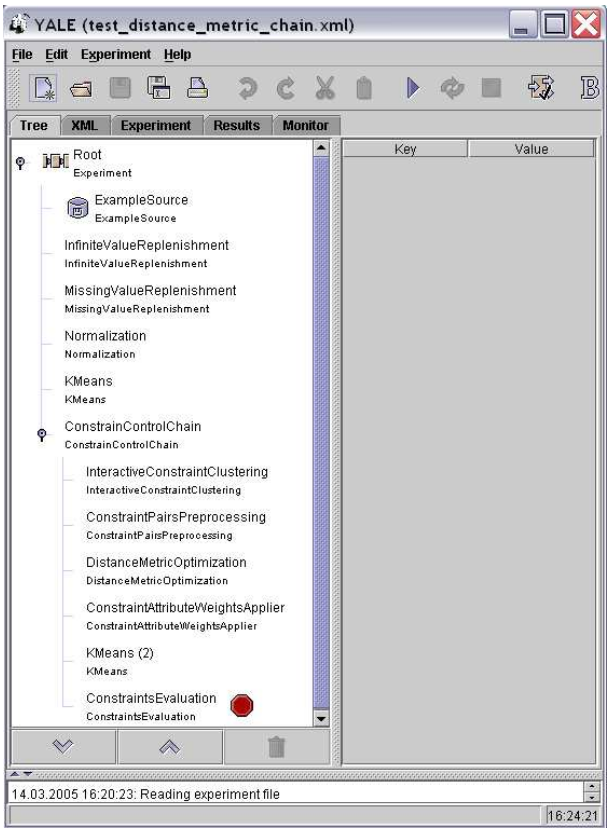


Abbildung 6.8: **Beispielhafter Aufbau des Systems als ein Experiment in YALE mit dem Distance Metric-Verfahren und k-Means als Kernclusteringalgorithmus.** Zunächst werden die Daten eingelesen, vorverarbeitet und normalisiert. Anschließend wird das Vorclustering mit k-Means durchgeführt. Ab diesem Punkt startet die Hauptschleife mit ConstraintControlChain. Dort kann der Benutzer die Constraints spezifizieren. Diese werden in Constraintpaare umgewandelt und den Optimierungsalgorithmus (hier DistanceMetricOptimization) geschickt. Dieser berechnet einen neuen Gewichtsvektor, welcher auf die Daten angewendet wird. Mit den veränderten Daten wird erneut k-Means aufgerufen und der Iterationsschritt ist zu Ende. Der Evaluationsoperator ist da, um das erzeugte Clustering zu bewerten. Dieser Aufbau entspricht unserem, in den Kapiteln 5 und 6.2 vorgestellten Systemen.

[11] Die Definition des Begriffs *Operator* und anderer Bestandteile der Entwicklungsumgebung YALE sind in deren Dokumentation unter `http://sourceforge.net/projects/yale/` und in [19] zu finden.

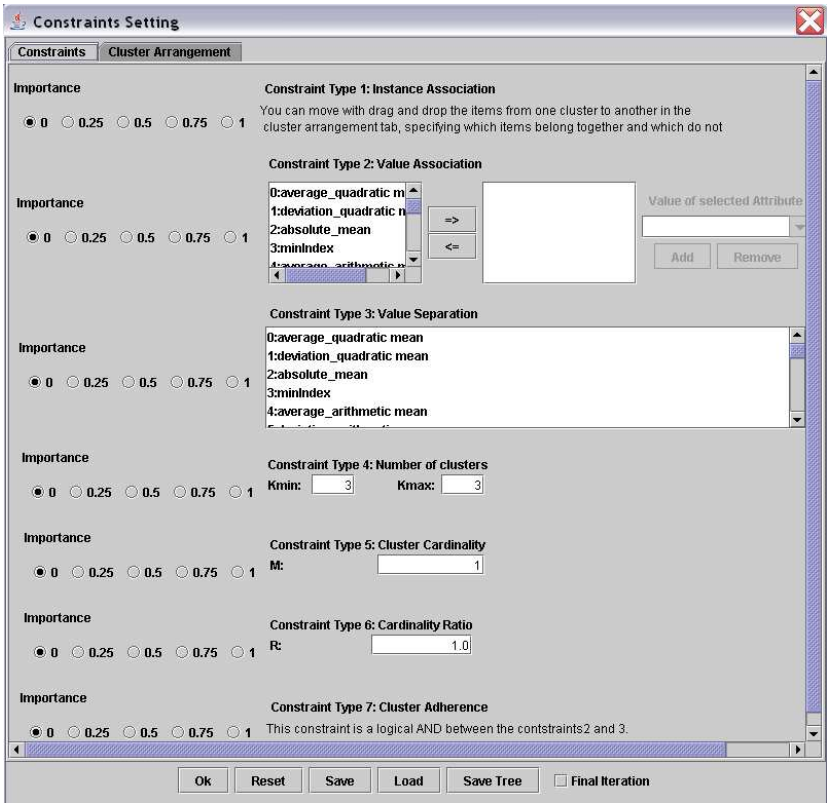


Abbildung 6.9: **InteractiveConstraintClustering Operator für die Eingabe der Constraints, Constraintsmanipulationsfenster.** In diesem Fenster kann man für alle Constraints die Importance festlegen. Des Weiteren kann man alle Constraints bis auf Instance Association verändern. Das Constraint Instance Association wird in dem Fenster auf der Abbildung 6.10 manipuliert. Die Musikstücke werden dem gewünschten Cluster zugewiesen. Für das Constraint 2, Value Association muss man zunächst das Merkmal markieren und mit dem Button **[=>]** in die Auswahlliste verschieben. Für die Merkmale in dieser Liste kann man Werte oder Intervalle in das Feld **Values of selected Attributes** eintragen und mit dem Button **[Add]** bestätigen. Mit dem Button **[Remove]** löscht man die Werte einzeln aus der Liste. Für das Constraint Value Separation reicht es die Merkmale zu markieren, die man separiert haben möchte. Für die Einstellung der übrigen Constraints trägt man die gewünschten Werte in die entsprechende Felder ein. Wenn man mit der Einstellung zufrieden ist, so bestätigt man mit dem Button **[Ok]**. Wenn man die Werte zurücksetzen möchte, so klickt man auf das Button **[Reset]**. Um die Constraints zu speichern, klicke man auf **[Save]**. Um diese wieder zu laden, klicke man auf **[Load]**. Um die aktuell eingestellte Clustermenge zu speichern klicke man auf **[Save Tree]**. In allen drei Fällen erscheint ein Standardbrowser zum Auswählen der Zieldatei. Wenn der aktuelle Iterationsdurchlauf der letzte des Systems sein soll, so setze man ein Häckchen vor **[Final Iteration]**.

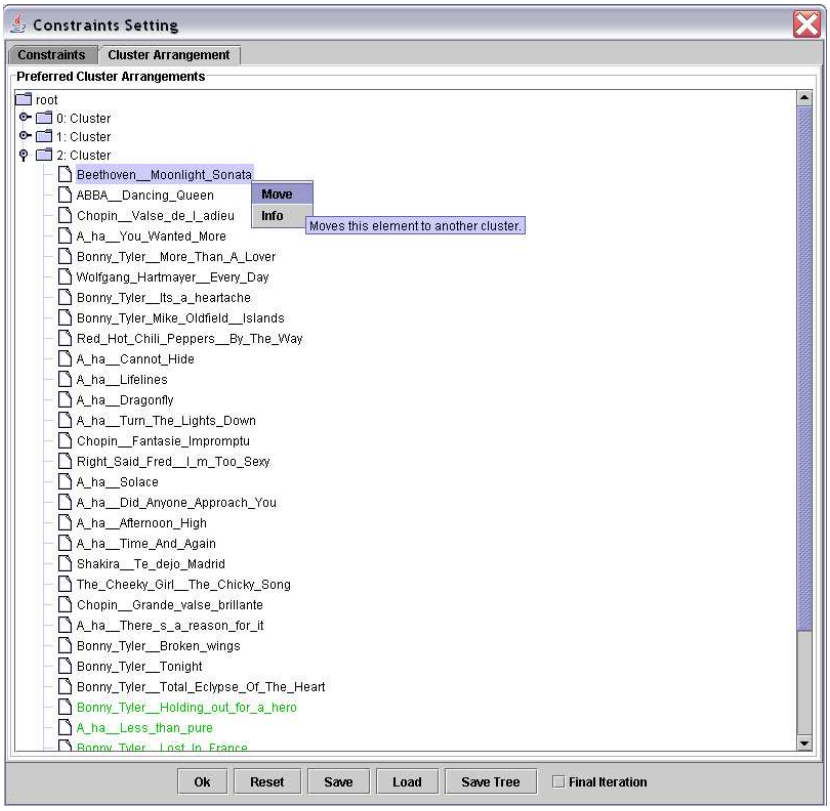


Abbildung 6.10: **InteractiveConstraintClustering Operator für die Eingabe der Constraints, Instance Association-Fenster.** In diesem Fenster sind die Cluster und die, in ihnen enthaltenen Musikstücke, als ein Baum dargestellt. Die Cluster sind die mittleren Knoten und die Musikstücke sind die Blätter. In diesem Fenster hat man zusätzlich zu der Bedienung der schon in der Abbildung 6.9 erläuterten Buttons die Möglichkeit, die Musikstücke einem anderen Cluster zuzuweisen. Dies kann durch Drag and Drop oder durch ein Popup-Menü geschehen, welches durch das Drücken der rechten Maustaste an dem zu verschiebenden Musikstück aufgerufen wird (In der Abbildung wird dies bei dem Musikstück *Bethoven__Moonlight_Sonata* durchgeführt). Es erscheint ein Dialog, in dem man ein Zielcluster auswählen kann. Das markierte Musikstück wird daraufhin dorthin verschoben und zur besseren Übersicht grün eingefärbt. Mit diesen Zuweisungen setzt man das Constraint Instance Association.

7 Evaluation

Nachdem die einzelnen Verfahren zur Optimierung der Cluster anhand der Vorgaben der Constraints vorgestellt wurden, möchten wir untersuchen, welches Verfahren sich in welcher Situation am besten eignet und ob es ein Verfahren gibt, welches für alle Situationen am besten die gestellte Aufgabe lösen kann.

Für die Evaluation verwenden wir einen Datensatz aus 139 zufällig zusammengestellten Musikstücken. Es sind Musikstücke aus verschiedenen Genres vertreten, sowohl Klassik als auch Pop und Rock. Diese Ansammlung sollte möglichst der privaten ungeordneten Sammlung eines Benutzers entsprechen. Zudem war es wichtig, dass für beliebige Benutzer einige Musikstücke in der Liste waren, die ihm gefielen, aber auch einige, die für ihn überhaupt nicht in Frage kamen, so dass für jeden Benutzer wenigstens die Aufteilung nach den Kriterien, Gefallen und nicht Gefallen gemacht werden könnte.

7.1 Evaluationsproblematik

Bei der Evaluation der Ergebnisse besteht immer die Gefahr deren Missdeutung. Um dies Vorzubeugen wird in diesem Kapitel versucht auf mögliche Probleme, Eigenarten und Fehlerquellen sowohl der Daten als auch der verwendeten Verfahren hinzuweisen und diese in den Evaluationsphasen zu berücksichtigen. Es handelt sich oft um unauflösbare Paradoxien der Problematik oder um spezifische Eigenarten der Algorithmen, die in der von uns konstruierten Konstellation Schwierigkeiten bereiten. Eine empirische Untersuchung der Wichtigsten Probleme erfolgt in dem Kapitel 7.3.

Einige der im Kapitel 6.7 vorgestellten Verfahren, die CSPs (Kap. 6.7.3) und die Evolutionäre Algorithmen (Kap. 6.7.6) haben kritische Laufzeiten. Wir wollen versuchen, die Ursachen und mögliche Lösungen dafür zu ergründen.

Für die Evaluation des erhaltenen Gewichtsvektors muss ein Clusteringalgorithmus wie k-Means ausgeführt werden, um zu sehen, welche Cluster sich damit bilden. Für das Score-Basierte-, das Analytische Verfahren und die SVMs (Kap. 6.7.1, 6.7.2 und 6.7.4) stellt das kein Problem dar, da diese nur einmal die Evaluationsroutine aufrufen. Anders ist es bei den CSPs und den Evolutionären Algorithmen. Hier stellt die Evaluation einen *Performancebottleneck* dar, denn wir müssen diese Operation bei jedem erzeugten Gewichtsvektor durchführen. Oft müssen aber, z.B. im Falle der Evolutionären Algorithmen, sehr viele Individuen erzeugt und dementsprechend viele Evaluationsdurchläufe gestartet werden. Auch wenn wir den Durchlauf von k-Means pro Individuum auf eine Iteration beschränken, müssen wir immer noch mit einer für ein interaktives System inakzeptablen Laufzeit rechnen. Andererseits ergaben die Versuche in dieser Arbeit, dass Evolutionäre Algorithmen im Durchschnitt schon innerhalb der ersten zwanzig bis dreißig Iterationen ein zufriedenstellendes Ergebnis liefern konnten, welches oft auch nach über hundert Iterationen nicht überboten wurde. Die Abbildung 7.1 veranschaulicht dies beispielhaft. Aus diesem Grunde werden wir die Ergebnisse der Evolutionären Algorithmen zum Vergleich in unsere Evolutionsphasen einbinden.

Des Weiteren bleibt die im Kapitel 6.6 schon gestellte und für diese Arbeit zentrale Frage im Raum, ob die ausgewählten Merkmale ausreichend sind, um beliebige Partitionierungen der Daten durch Gewichtung vornehmen zu können. Es erscheint sinnvoll statt Merkmalsgewichtung, bzw. Merkmalsauswahl, Merkmalsgenerierung zu betreiben. Dies würde aber von dem Gesamtkonzept eines interaktiven Systems abweichen und müsste mit einer anderen Herangehensweise als in dieser Arbeit vorgestellten, erreicht werden.

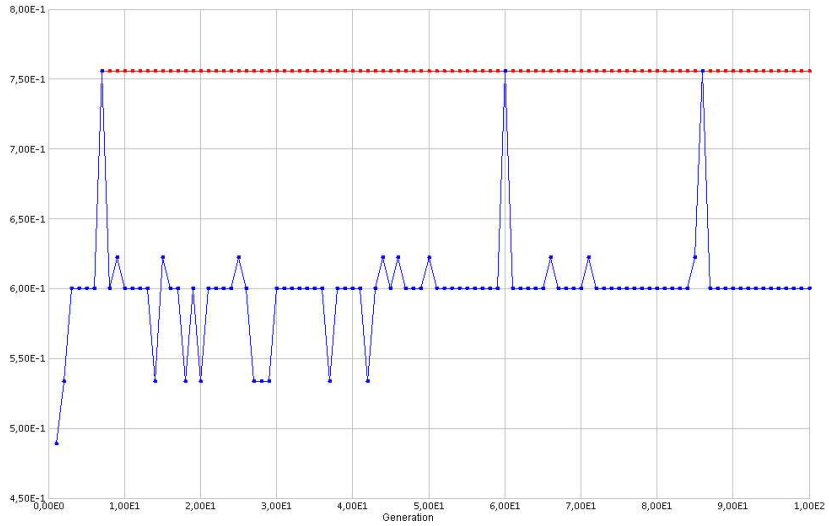


Abbildung 7.1: **Ein Repräsentativbeispiel für das Verhalten des EA.** Die vertikale Achse gibt die Anzahl der Generationen an, die horizontale den Wert der Evaluierungsfunktion. Die blauen Punkte beschreiben die aktuellen Werte der besten Funktion für die jeweilige Generation. Die roten Punkte beschreiben die bislang gefundenen besten Werte. Wie man gut erkennen kann, wird der beste Wert schon innerhalb der ersten zehn Generationen gefunden.

Ein weiteres Problem ist die Abbildung der Ideen des Benutzers auf Constraints (Kap. 4) und von dort aus auf die Constraintpaare (Kap. 6.5). Es sind zwei kritische Stellen, die das Endergebnis verfälschen, ja sogar wertlos machen könnten. Die Frage lautet: Sind die vorgeschlagenen Constraints wirklich das Medium, mit dem der Benutzer seine Vorstellungen von einem, ihm vorschwebenden, Clustering vermitteln kann? Ist er gezwungen, Informationen darüber zurückzuhalten, weil er diese in der dargebotenen Sprache der Constraints nicht ausdrücken kann? Und wenn das gegeben ist, erhalten die Abbildungen auf die Constraint-Pairs die ursprünglich durch den Benutzer vermittelte Information oder wird sie verfälscht? Unter dem Aspekt dieser Fragen sollten die Ergebnisse dieser Arbeit betrachtet werden und auf keinen Fall als absolute Werte angesehen werden. Das Ziel zukünftiger Arbeiten könnte darauf basieren, die soeben ansatzweise aufgeworfenen Fragen zu untersuchen und die angesprochenen Probleme genauer zu definieren oder sie gar zu lösen.

Semisupervised Clustering hat zwei Seiten. Auf der einen ermöglicht es, die Suche nach Benutzervorlieben ohne eine genaue Differenzierung der Klassen durchzuführen. Auf der anderen Seite benutzt es als Kern Clusteringmethoden, also Methoden, die unüberwacht lernen. Einige unter ihnen, z.B. k-Means, haben den Nachteil, dass sie, je nach der Wahl der ersten Zentroiden für die gleiche Datenmenge mit den gleichen Parametereinstellungen verschiedene Clusterings erzeugen können. Andere, z.B. DBSCAN, haben eine sehr schwer zu bestimmende Parametrisierung. Sogar in der Literatur wird die Invention des Benutzers bei dieser, eigentlich für ihn zu sehr ins Detail des Algorithmus gehenden, Frage vorgeschlagen (Kap. 2.6.2). Bei den in dieser Arbeit angewendeten Versuchen, wurde eine Parameteroptimierung vor dem eigentlichen Clustering betrieben, weil der Benutzer auf keinen Fall mit den Interna des Algorithmus belästigt werden sollte. Dies erschien vor allem im Hinblick dessen sinnvoll, dass die internen Algorithmen austauschbar bleiben sollten. Parameteroptimierung ist aber wie bereits erwähnt ein sehr zeitraubender Prozess und gefährdet die Interaktivität des Systems. Deswegen musste auch hier ein Tradeoff zwischen der Effizienz und der Güte getroffen werden.

Unklar ist die Frage, was mit den gesetzten Constraints nach einer Iteration geschehen soll. Zum einen sollten die Constraints wie Number of Clusters, oder Cluster Cardinality beibehalten werden. Zum anderen aber werden nach einem Iterationsdurchlauf die Cluster und deren Inhalte verändert, so dass die alten Zuweisungen unnötig, bzw. sinnlos erscheinen und vor allem den Benutzer verwirren könnten. Deswegen würde man zunächst dafür plädieren, das Constraint Instance Association bei jedem Iterationsschritt zurückzusetzen. Andererseits entsteht dabei das Problem der fehlenden Kontinuität der Iterationsschritte. Jeder Iterationsschritt hat dann praktisch nichts mehr mit dem vorhergehenden zu tun und könnte genausogut bei einem Neustart des Systems unabhängig von den anderen Schritten erfolgen. Deswegen ist es doch sinnvoll die gesetzten Constraints vollkommen zu behalten, auch wenn sie anfänglich für Verwirrung sorgen könnten.

Eine umstrittene Designentscheidung ist, ob man Verschlechterungen, also kleinere Ergebnisse als die Ausgangswerte zulassen dürfte, oder ob man generell, falls kleinere Ergebnisse auftreten, die ursprünglichen beibehält. Aus reiner Anwendungssicht ist die zweite Lösung sinnvoll, nicht aber aus der Sicht dieser Arbeit, die an den tatsächlichen Ergebnissen interessiert ist. Also werden aus Forschungsgründen die Ergebnisse beibehalten, egal wie niedrig sie im Vergleich zu den Ausgangswerten sind.

Für die Evaluation ist der Unterschied zwischen dem Ausgangsclustering und dem, durch die Manipulation des Gewichtsvektors erzeugten, Clustering wichtig. Deswegen sollten die Constraints für die benutzerunabhängigen Tests mit Bedacht gewählt werden. Es kann sein, dass das Ausgangsclustering dadurch schon nahe am Optimum liegt, so dass wir, ganz gleich was für schwere algorithmische Geschütze wir auffahren, immer nur eine geringe relative Verbesserung messen können.

7.2 Evaluationsphasen

Es wurden zwei Evaluationphasen angedacht, die im Folgenden vorgestellt werden. Beide Phasen wurden jeweils für k-Means (Kap. 2.6.1) und DBSCAN (Kap. 2.6.2) als Kernclusteringalgorithmen durchgeführt. Alle Werte sind Ergebnisse der globalen Evaluierungsfunktion (Funktion 4.1).

K-Means hatte die folgenden Ausgangsparameter: $k = 3$ und 100 Iterationen. Für DBSCAN wurden aufgrund vorheriger Parameteroptimierungsdurchläufe Eps = 0.155 und MinPts = 2 gesetzt. Das waren die Werte, die sechs gleichmäßig gefüllte Ausgangscluster erzeugten[1].

Die Ergebnisse von DBSCAN und k-Means sind nicht direkt miteinander vergleichbar, da sie unterschiedliche Ausgangsclusterings produzieren und manipulieren. Trotzdem kann man die relative Differenz zu den Ausgangswerten vergleichen, um festzustellen, welcher Algorithmus besser ist.

Die Aufteilung in zwei Evaluationsphasen hat den Sinn, dass zwischen der formalen Bewertung der Ergebnisse und der Bewertung durch einen Benutzer unterschieden werden soll. Da das in dieser Arbeit vorgestellte System interaktiv ist, ist eine Bewertung durch einen Benutzer noch ausschlaggebender als die rechnerbasierten Evaluationsmethoden. Diese dienen dem Vergleich und der Untersuchung, ob zwischen den Benutzerzufriedenheit und der guten Ergebnissen der Evaluationsprozeduren ein Zusammenhang besteht. Der Benutzer ist, wie in der Einleitung schon hervorgehoben wurde, die oberste Bewertungsinstanz. Wenn er nicht mit dem Ergebnis zufrieden ist, dann kann die Evolutionsprozedur noch so gute Werte liefern – trotzdem sind sie für ihn wertlos.

Über allen Evaluationsdurchläufen konnten einige interessante Bobachtungen gemacht werden, die in diesem Absatz aufgeführt werden. Zunächst sei angemerkt, dass die Laufzeit einiger Methoden zeitweise weit über der Schwelle eines interaktiven Systems lag. Dies lag vor allem daran, dass sehr viele Constraintpaare erzeugt wurden, im Durchschnitt waren es ca. $1\,000$ und im maximalen Fall sogar über $160\,000$. Besonders betroffen waren die SVMs. Meistens waren sie schon nach wenigen Sekunden fertig, doch manchmal brauchten sie über drei Stunden, um die Gewichtung zu ermitteln. Dies schien

[1] Diese Werte gelten nur für unsere spezifische Musiksammlung. Bei anderen sind weitere Optimierungsdurchläufe erforderlich, die zu anderen Ausgangsparametern führen. Ebenso bei jeder Änderung der Distanzmetrik durch den neuen Gewichtsvektor.

nicht ganz von der Anzahl der Constraint Pairs abzuhängen. Wahrscheinlich waren die Eingabedaten in diesen Fällen zu sehr verrauscht und zu widersprüchlich. Die Distance Metric Optimierung mit dem Newton-Verfahren hatte besonders Schwierigkeiten bei vielen Constraintpaaren. Dort waren Wartezeiten bis zu einer halben Stunde zu verzeichnen. Der Evolutionäre Algorithmus war immer gleich schnell. Er brauchte im Schnitt zehn Minuten, um ein Ergebnis zu berechnen. Für DBSCAN war er ca. 5% langsamer. Da er aber nur zum Vergleich herangezogen wurde, waren diese Zeiten vollkommen akzeptabel.

7.2.1 Evaluationsphase 1: Direkte Evaluation

In dieser Evaluationsphase werden für einige fest definierte Constraintmengen (engl. *Constraint-Sets*), die Ergebnisse der Evaluationsfunktion (Kap. 4.2) für alle angewandten Methoden nach *einem* Iterationsschritt aufgelistet.

Es werden 10 „ideale“, also möglichst gut evaluierbare Constraint-Sets konstruiert, welche sowohl die Erfüllung der Einzelconstraints als auch die Erfüllung einer Kombination der Constraints widerspiegeln. Alle Constraints in diesen Mengen sind mit dem Gewicht 1 oder 0 bewertet. Diese Constraintsmengen sind folgendermaßen aufgebaut:

1. Nur Instance Association: Zehn Zuweisungen von Musikstücken zu Clustern.
2. Nur Value Association: Es wird ein Merkmal ausgesucht und ein Intervall aus dem Bereich, der in den Daten vorkommt, bestimmt.
3. Nur Value Separation: Ein Merkmal wird zum Separieren ausgewählt.
4. Nur Number of Clusters: Die erwünschte Anzahl der Cluster wird auf einen Wert gesetzt, der größer als die Ursprungsanzahl der Cluster ist. Da diese Einstellungen für DBSCAN keine Rolle spielen, wird dieser Test nur für k-Means durchgeführt.
5. Nur Cluster Cardinality: Die neue Mindestkardinalität wird auf einen Wert eingestellt, der größer ist als die aktuelle Kardinalität des kleinsten Clusters.
6. Nur Cardinality Ratio: Der erwünschte Wert für R wird größer gesetzt als das aktuelle Verhältnis zwischen der minimalen und der maximalen Anzahl der Musikstücke in den Clustern.
7. Value Association, Value Separation und Cluster Adherence: Diese drei Werte gehören zusammen. Es werden für Value Association und Value Separation die oben verwendeten Einstellungen gemacht.
8. Instance Association, Value Association, Value Separation und Cluster Adherence: Alle Constraints, die sich konkret auf Musikstücke in den Clustern beziehen. Es werden für die einzelnen Constraintstypen die oben verwendeten Einstellungen beibehalten.
9. Number of Clusters (nur bei k-Means), Cluster Cardinality und Cardinality Ratio: Alle Constraints, die sich auf den formalen Aufbau der Cluster beziehen. Die Werte der einzelnen Constraints werden wie oben gesetzt.
10. Alle Constraints zusammen: Die oberen Werte für die einzelnen Constraints werden beibehalten.

Als zweiten Schritt werden zehn Benutzer gebeten *persönliche Constraint-Sets* zu konstruieren. Deren Evaluation soll das Verhalten des Systems für den allgemeinen Benutzer und das Verhalten des Benutzers angesichts des Systems widerspiegeln. Es ist nämlich überhaupt nicht klar, ob der Benutzer unser System ohne weitere Einwände akzeptiert, oder ob er unter einem „Musikgruppierungsprogramm“ sich etwas ganz anderes vorstellt. In diesem Teil dürfen die Constraints auch eine Importance zwischen 0 und 1 haben. Die Reaktionen der Benutzer auf das System in allen Evaluationsphasen sind in dem Kapitel 7.2.3 zusammengefasst.

Ergebnisse der Evaluationsphase 1

Die Tabellen 7.1 – 7.3 zeigen die Ergebnisse der Evaluationsphase 1 mit k-Means als Kernclusteringalgorithmus. Die Tabellen 7.4 – 7.6 zeigen die Ergebnisse der Evaluationsphase 1 mit DBSCAN als Kernclusteringalgorithmus.

Zunächst folgen ein paar Anmerkungen zur Notation. Mit blauer Farbe wurden zusammenfassende Ergebnisse, wie z.B. die durchschnittlichen Verbesserungen markiert. Fett sind alle Ergebnisse markiert, die überdurchschnittlich gut ausfallen, genauer gesagt mindestens 15% Verbesserung gegenüber dem Ausgangszustand erreichen. Rot markiert sind besonders schlechte Gesamtergebnisse, die eine durchschnittliche Verschlechterung von mindestens 10% erreichen.

Zu jeder Bewertung des Constraint Sets ist die Bewertung der dabei erzeugten Paare hinzugenommen. Wenn die beiden Werte sich nicht unterscheiden, wie z.B. beim Instance Association Constraint, dann sind sie nach dem gleichen Prinzip bewertet worden. Die Bewertung der Paare ist der einfache Quotient der Anzahl der erfüllten Must- und Cannot-Link Paare durch die Anzahl aller Paare. Diese Bewertung soll eine Hilfe sein zu bestimmen, inwiefern die Abbildung der Paare auf die Constraints gelungen ist. Für das Constraint, Number of Clusters, werden keine Paare erzeugt, deswegen ist diese Zeile in den Tabellen leer. DBSCAN berücksichtigt nicht die gewünschte Anzahl der Cluster, so dass in den Tests das gesamte Constraint ausgelassen wurde.

Nr.	**Constraint Sets**	**Kein Algo**	**Score**	**Analyt. Verf.**	**SVMs**	**Newton**	**Evol. Algo**
1:	Instance Association	0,5054	0,5494	0,5494	0,5494	0,5714	**0,727**
1a:	Pairs	0,5054	0,5494	0,5494	0,5494	0,5714	**0,727**
2:	Value Association	0,6	0,6071	**0,9**	**0,9**	0,5692	**0,975**
2a:	Pairs	0,5714	0,5714	0,5714	**0,75**	0,4642	**0,75**
3:	Value Separation	0,9278	0,9226	0,913	0,9164	0,9275	0,982
3a:	Pairs	0,6339	0,4117	0,5294	0,3986	0,6339	0,5
4:	Number Of Clusters	0,5	1,0	1,0	1,0	1,0	1,0
4a:	Pairs	—	—	—	—	—	—
5:	Cluster Cardinality	0,9242	0,7424	0,7424	0,8257	0,6136	**1.0**
5a:	Pairs	0,363	0,2321	0,2321	0,363	**0,738**	**0,815**
6:	Cardinality Ratio	0,8474	0,2015	0,2941	0,6237	0,1388	**1.0**
6a:	Pairs	0,4444	0,3888	0,4444	0,3333	**0,6111**	**0,778**
7:	Asso, Sepa, Adher	0,7092	0,6483	0,6783	**0,9239**	0,4731	**0,943**
7a:	Pairs	0,573	0,573	0,6432	**0,7251**	0,4912	**0,75**
8:	Inst, Asso, Sepa, Adher	0,6583	0,5984	0,5128	0,5346	0,6107	**0,871**
8a:	Pairs	0,5612	0,5397	0,5483	0,5505	0,4795	**0,817**
9:	Num, Card, Ratio	0,7572	0,5983	0,527	0,621	0,481	0,87
9a:	Pairs	0,4594	0,2207	0,1891	0,3243	**0,7657**	0,4828
10:	Alle	0,7007	0,5747	0,6565	0,6892	0,727	0,744
10a:	Pairs	0,5285	0,6201	0,6298	0,6366	0,6456	0,671
	Durchschn. Verbesserung	0	-0,0718	-0,0386	0,04237	-0,1047	**0,1952**
	Pairs	0	-0,0593	-0,0336	-0,001	0,0844	**0,1834**
	Varianz d. Verbesserung	0	0,0778	0,0828	0,0517	0,0977	0,0225
	Pairs	0	0,0121	0,0121	0,0171	0,0275	0,0288

Tabelle 7.1: Ergebnisse der Evaluationsphase 1 mit k-Means: Die ersten 10 Constraint Sets

Die Evaluation von k-Means zeigt vor allem für die ersten zehn Constraint Sets, wie die Ergebnisse mit der wachsenden Komplexität der Algorithmen sich verbessern. Trotzdem erreicht kaum ein Verfahren bei der durchschnittlichen Verbesserung positive Werte. Im Vergleich zu den überragend guten Ergebnissen des Evolutionären Ansatzes, von denen wir annehmen, dass sie nahe am Optimum liegen,

erscheint dies besonders entmutigend.

Die oben schon angesprochene strukturelle Ähnlichkeit des Score-Verfahrens, des Analytischen Verfahrens und der Vorbereitung der Eingabe für die SVM spiegelt sich auch in den Ergebnissen wider, wobei allerdings aufgrund der Verbesserungen deutlich wird, dass von einem Verfahren auf das nächste es sich auszahlt, mehr Domainformationen zu verwenden. Das Distanzmetriklernverfahren, basierend auf der Newton-Iteration tanzt aus der Reihe. Das sieht man besonders an der durchschnittlichen Verbesserung. Während sie für die im Kapitel 4.4 vorgestellte Evaluation sehr niedrig ist, ist sie für die Paare in einem stabilen positiven Bereich. Überhaupt scheint dieses Verfahren am besten dazu geeignet zu sein Constraintpaare zu optimieren, während die SVM dazu die schlechtesten Ergebnisse erzeugt, was die Gesamtauswertung (Tabelle 7.3) belegt. Kurioserweise ist das einzige Verfahren, außer den EAs, das es geschafft hat bei der durchschnittlichen Verbesserung in den positiven Bereich zu kommen, das analytische Verfahren, obwohl es oft vergleichsweise schlechte Ergebnisse lieferte.

Von großem Interesse war, in welchem Maße die einzelnen Constraints erfüllt wurden. Es konnte zwar nicht eindeutig nachgewiesen werden, welche Constraints relevanter waren als die anderen, aber über den Durchschnitt konnte man schon sagen, dass für k-Means die Constraints Value Association, Value Separation und Cluster Adherence viel öfter und eine viel höhere Verbesserung erfuhren als die anderen Constraints[2]. Für DBSCAN herrschte der umgekehrte Fall. Die *Existential Constraints* (Kap. 2.6.3) wurden dabei eindeutig bevorzugt. Eine interessante Ausnahme bildete das Constraint Instance Association, bei welchem sowohl mit k-Means als auch mit DBSCAN signifikante Verschlechterungen zu verzeichnen waren.

Wie erwartet schnitten jegliche Kombinationen von Constraints im Durchschnitt schlechter ab, als das Setzen der einzelnen Constraints.

Die Ergebnisse der Benutzercontraints sind erwartungsgemäß schlechter ausgefallen, als die der vorbereiteten Constraints. Denn die Benutzer haben die Bereiche des Constraintsraums angeschnitten, die aus der objektiven Sicht der Arbeit und mit den gegebenen Audiomerkmalen nicht erreicht werden konnten. Dabei konnte der in den ersten zehn Constraint Sets entstandene Eindruck, dass die Ergebnisse von Verfahren zu Verfahren besser werden, nicht reproduziert werden. Andererseits konnten noch nicht einmal die EAs eine so starke Verbesserung wie in der Tabelle 7.1 erzielen. Vielleicht lag es daran, dass die Constraints schon für die Ausgangslage fast optimal erfüllt waren.

Die, im Vergleich zu den EAs, ziemlich hohe Varianz der Verbesserungen[3], besagt, dass man dabei auf keine stabile Ergebnisse zählen darf. Im Gegensatz dazu steht die geringe Varianz der durchschnittlichen Verbesserung der Paare. Diese bleibt außer bei SVMs sogar unter den 3% der EA.

Im Gegensatz zu k-Means waren die Ergebnisse von DBSCAN für die zehn Constraint Sets überraschend gut, vor allem für die Bewertung der Paare. Es entstand der Eindruck, dass DBSCAN die Bildung Constraintpaare-konformerer Cluster begünstigt, denn während man im Durchschnitt froh war, 3% Verbesserung für die normale Bewertung zu erhalten, hatte man für die Bewertung der Paare oft eine Verbesserung von über 13%. Wenn man aber diese Ergebnisse wiederum mit denen von dem EA vergleicht, erscheinen sie immer noch sehr niedrig, denn dort war eine weitere Verbesserung von über 20% möglich.

Interessanterweise waren die Ergebnisse für das Constraint Instance Association vergleichsweise niedrig, was aber auf den zweiten Blick nicht weiter überrascht, denn das Ausgangsclustering lag schon nahe am Optimum.

Die Benutzerconstraints haben die Euphorie über die guten Ergebnisse von DBSCAN leicht gedämpft. Seltener konnten in diesem Evaluationsabschnitt signifikante Verbesserungen erzielt werden. Fast alle durchschnittliche Verbesserungen sind im negativen Bereich. Unter anderem konnte man einen durchschnittlichen Rekord der Verschlechterung von fast 22% bei dem Newton-Verfahren verzeichnen. Dies ist vor allen Dingen auf den Umstand zurückzuführen, dass den Wünschen der Benutzer bzgl. der Anzahl der Constraints nicht so eindeutig entsprochen werden konnte, wie bei k-Means.

[2] Mit Ausnahme des Constraints Number of Clusters, welches, wie oben schon erwähnt, eine Sonderstellung einnimmt.
[3] Teilweise bis zu 10%!

Nr.	Constraint Sets	Kein Algo	Score	Analyt. Verf.	SVMs	Newton	Evol. Algo
11:	1.Benutzerconstraints	0,546	**0,7267**	**0,7074**	**0,7369**	**0,704**	**0,815**
11a:	Pairs	0,711	0,648	0,6298	0,3185	0,6191	0,686
12:	2.Benutzerconstraints	0,6948	0,6269	0,4631	0,312	0,7532	0,766
12a:	Pairs	0,5115	0,5115	0,493	0,5787	0,6389	0,664
13:	3.Benutzerconstraints	0,5305	0,5067	0,5328	0,489	0,5454	0,627
13a:	Pairs	0,5305	0,5067	0,5328	0,489	0,5454	0,627
14:	4.Benutzerconstraints	0,5163	0,4575	0,4967	0,2614	0,5163	**0,7**
14a:	Pairs	0,5163	0,4575	0,4967	0,2614	0,5163	**0,7**
15:	5.Benutzerconstraints	0,6301	0,4657	0,6301	0,2603	0,4931	0,648
15a:	Pairs	0,6301	0,4657	0,6301	0,2603	0,4931	0,648
16:	6.Benutzerconstraints	0,4352	**0,7623**	**0,8**	**0,6453**	**0,7746**	**0,812**
16a:	Pairs	0,4352	0,5246	**0,601**	0,2906	0,5492	**0,812**
17:	7.Benutzerconstraints	0,4752	0,5505	0,5419	0,2924	0,4537	0,544
17a:	Pairs	0,4752	0,5505	0,5419	0,2924	0,4537	0,544
18:	8.Benutzerconstraints	0,7051	0,6596	0,6777	0,5513	0,6646	0,819
18a:	Pairs	0,6303	0,6678	0,6645	0,2769	0,6026	0,721
19:	9.Benutzerconstraints	0,8231	0,5796	0,6063	0,4667	0,5277	0,767
19a:	Pairs	0,6752	0,6438	0,6243	0,2591	0,5936	**0,802**
20:	10.Benutzerconstraints	0,4059	**0,6936**	**0,6993**	**0,7677**	**0,6969**	**0,724**
20a:	Pairs	0,4119	0,3871	0,3987	0,5354	0,3937	0,448
	Durchschn. Verbesserung	0	0,0267	0,0393	-0,0979	0,0367	**0,146**
	Pairs	0	-0,015	0,0097	-0,1953	-0,011	**0,1124**
	Varianz d. Verbesserung	0	0,0353	0,0373	0,0734	0,0359	0,0191
	Pairs	0	0,0053	0,0045	0,0377	0,0069	0,0126

Tabelle 7.2: Ergebnisse der Evaluationsphase 1 mit k-Means: Benutzerconstraints

Die Gesamtauswertung der Ergebnisse von DBSCAN ist auf der ganzen Linie schlechter als bei k-Means. Die Auswertung für die EAs belegt, dass eine Verbesserung von über 25% möglich wäre. Dies wurde aber von keinem der Algorithmen ausgenutzt.

Die Varianz der Ergebnisse ist im Vergleich zu k-Means ziemlich hoch. Sie übersteigt oft 10%-Marke. Sogar die EAs haben eine Varianz von über 6%.

Allgemein kann man sagen, dass die Algorithmen unerwartet oft schlechtere Ergebnisse produzierten als die Ausgangsclusterings. Die Ursachen für dieses Verhalten werden in dem Kapitel 7.3 empirisch Untersucht.

Gesamt:	Constraint Sets	Kein Algo	Score	Analyt. Verf.	SVMs	Newton	Evol. Algo
	Durchschn. Verbesserung	0	-0,0717	0,0003	-0,0269	-0,034	**0,1706**
	Pairs	0	-0,0381	-0,0114	-0,109	0,0361	**0,1542**
	Varianz d. Verbesserung	0	0,0737	0,0585	0,0643	0,0686	0,0204
	Pairs	0	0,0086	0,0083	0,036	0,0183	0,0203

Tabelle 7.3: Ergebnisse der Evaluationsphase 1 mit k-Means: Gesamtauswertung

Nr.	Constraint Sets	Kein Algo	Score	Analyt. Verf.	SVMs	Newton	Evol. Algo
1:	Instance Association	0,6666	0,3238	0,219	0,219	0,219	0,705
1a:	Pairs	0,6666	0,3238	0,219	0,219	0,219	0,705
2:	Value Association	0,54	0,2812	**0,9918**	0,5	**0,9918**	**0,992**
2a:	Pairs	0,6545	0,5272	0,1818	0,1818	0,1818	**0,818**
3:	Value Separation	0,8863	0,7747	0,468	0,7986	0,468	0,992
3a:	Pairs	0,5252	0,5225	0,7919	0,7683	**0,7919**	0,863
4:	Number Of Clusters	—	—	—	—	—	—
4a:	Pairs	—	—	—	—	—	—
5:	Cluster Cardinality	0,7777	**0,9**	0,5	0,625	0,5	**1,0**
5a:	Pairs	0,2263	0,2271	**0,903**	**0,8925**	**0,9035**	**0,904**
6:	Cardinality Ratio	0,1785	0,0362	**1,0**	0,1102	**1,0**	**1,0**
6a:	Pairs	0,5757	0,5606	0,5606	0,6818	0,5606	**0,712**
7:	Asso, Sepa, Adher	0,6554	0,6892	0,6433	0,7494	0,6426	0,749
7a:	Pairs	0,5399	0,537	**0,8229**	**0,8229**	**0,8229**	**0,924**
8:	Inst, Asso, Sepa, Adher	0,6582	0,6915	0,5372	0,6222	0,5367	0,648
8a:	Pairs	0,587	0,5269	0,5269	0,5269	0,5269	0,601
9:	Card, Ratio	0,4781	**0,7625**	**0,75**	0,3468	**0,75**	**1,0**
9a:	Pairs	0,2295	0,2286	**0,86**	**0,864**	**0,864**	**0,864**
10:	Alle	0,5982	0,6203	0,6081	0,5172	0,6078	0,731
10a:	Pairs	0,3833	0,3737	**0,6948**	**0,6948**	**0,6948**	**0,721**
	Durchschn. Verbesserung	0	-0,0391	0,0309	-0,1056	0,0307	**0,2653**
	Pairs	0	-0,0622	**0,1308**	**0,1404**	**0,1308**	**0,3027**
	Varianz d. Verbesserung	0	0,0321	0,1569	0,0201	0,1569	0,0746
	Pairs	0	0,0118	0,1547	0,1512	0,1553	0,0603

Tabelle 7.4: Ergebnisse der Evaluationsphase 1 mit DBSCAN: Die ersten 10 ConstraintSets

Nr.	Constraint Sets	Kein Algo	Score	Analyt. Verf.	SVMs	Newton	Evol. Algo
11:	1.Benutzerconstraints	0,5175	0,9162	0,2909	0,3357	0,4828	0,683
11a:	Pairs	0,6069	0,7857	0,2603	0,2934	0,25	0,659
12:	2.Benutzerconstraints	0,437	0,3042	**0,6934**	0,4208	0,6934	**0,987**
12a:	Pairs	0,5612	**0,7681**	0,3988	**0,728**	0,3987	**0,761**
13:	3.Benutzerconstraints	0,5664	0,2518	0,2261	0,26	0,218	0,694
13a:	Pairs	0,5664	0,2518	0,2261	0,26	0,218	0,694
14:	4.Benutzerconstraints	0,591	0,4269	0,6023	0,5906	0,2631	0,643
14a:	Pairs	0,591	0,4269	0,6023	0,5906	0,2631	0,643
15:	5.Benutzerconstraints	0,6105	0,6263	0,6105	0,6263	0,4579	0,679
15a:	Pairs	0,6105	0,6263	0,6105	0,6263	0,4579	0,679
16:	6.Benutzerconstraints	0,7736	0,2437	0,667	0,7701	0,5632	0,783
16a:	Pairs	0,5471	0,4773	0,533	0,5402	0,5264	0,566
17:	7.Benutzerconstraints	0,4903	0,4882	0,488	0,486	0,4839	0,499
17a:	Pairs	0,4903	0,4882	0,488	0,486	0,4839	0,499
18:	8.Benutzerconstraints	0,6998	0,5108	0,2484	0,6062	0,2533	0,711
18a:	Pairs	0,6752	0,6878	0,6734	0,6062	0,2533	0,711
19:	9.Benutzerconstraints	0,3556	0,2689	0,2428	0,5057	0,9265	0,668
19a:	Pairs	0,6133	0,5543	0,5886	0,6343	0,259	0,756
20:	10.Benutzerconstraints	0,5594	0,5536	0,5717	0,5445	0,4826	0,588
20a:	Pairs	0,5594	0,5536	0,5717	0,5445	0,4826	0,588
	Durchschn. Verbesserung	0	-0,1623	-0,096	-0,0468	-0,0776	**0,1293**
	Pairs	0	-0,0587	-0,0869	-0,0512	-0,2228	0,0734
	Varianz d. Verbesserung	0	0,0278	0,041	0,0153	0,0927	0,0196
	Pairs	0	0,0207	0,0209	0,0222	0,0244	0,0039

Tabelle 7.5: Ergebnisse der Evaluationsphase 1 mit DBSCAN: Benutzerconstraints

Gesamt:	Constraint Sets	Kein Algo	Score	Analyt. Verf.	SVMs	Newton	Evol. Algo
	Durchschn. Verbesserung	0	-0,1039	-0,0359	-0,0747	-0,0263	**0,2759**
	Pairs	0	-0,0604	0,0163	0,0396	-0,0553	**0,2435**
	Varianz d. Verbesserung	0	0,0326	0,0978	0,0174	0,1212	0,0998
	Pairs	0	0,0154	0,0942	0,0905	0,1157	0,0671

Tabelle 7.6: Ergebnisse der Evaluationsphase 1 mit DBSCAN: Gesamtauswertung

7.2.2 Evaluationsphase 2: Subjektive Evaluation

In dieser Evaluationsphase werden *fünf* Benutzer gebeten zunächst ihre ideale Vorstellung einer vollständigen Clustermenge anzugeben. Anschließend sollen sie, ausgehend von *einer* Ausgangsposition, diese Clustermenge durch iterative Constraintmodifikation versuchen zu erreichen. Nach jedem Iterationsschritt können sie sich anhand der Ergebnisse neu orientieren, Constraints verändern und den nächsten Schritt initiieren. Dabei gelten die folgenden Nebenbedingungen:

1. Jeder Benutzer darf nur *ein* Constraint pro Iteration ändern.

   Z.B. darf in einem Schritt nur die Anzahl der Cluster festgesetzt werden und im nächsten die zugehörigen Zuweisungen der Musikstücke zu den Clustern erfolgen. Dadurch soll ein monotones und aufeinander aufbauendes Annähern ermöglicht werden, bei dem man nach jedem Schritt erkennen kann, welche Auswirkungen die Veränderung des Constraints hatte. Der nächste Schritt sollte aus dem vorhergehenden ableitbar sein und nicht wie eine Neuordnung der Constraints erscheinen.

2. Die ausgesuchte Constraintart darf aber beliebig verändert werden.

   Z.B. darf der Benutzer für Instance Association so viele Beispiele setzen, wie er als ausreichend betrachtet, um seine Vorstellung von einer Gruppierung zu vermitteln. Dies hat den Sinn, dass der Benutzer in jedem Schritt eine abgeschlossene Idee vermitteln kann, auf die in dem nächsten aufgebaut werden kann. Es bringt überhaupt nichts, wenn man erst nach mehreren Iterationen seine Vorstellung mit den dazugehörigen Mindestbeispielen vermitteln konnte, obwohl dies auch in der ersten zu erledigen wäre. Damit ist aber nicht gemeint, dass der Benutzer *alle* Musikstücke so ordnet, wie er das haben möchte, es soll ihm nur möglich sein genügend *Mindestbeispiele* setzen zu können.

3. Jeder Benutzer hat maximal *fünf* Iterationen, um seinem Ziel näher zu kommen.

   Nach dieser Anzahl sollte es möglich sein festzustellen, ob der unbedarfte Benutzer fähig ist, sich mit unserem System seinem Ziel anzunähern. Wenn dies nicht der Fall sein sollte, werden die Ergebnisclusterings der Iterationen ein alternierendes und unvorhersehbares Verhalten aufweisen.

Zusätzlich wird nach jedem Iterationsschritt die Ähnlichkeit der gewünschten Clustermenge zu der erhaltenen Clustermenge gemessen und das wiederum für alle in dieser Arbeit verwendeten Optimierungsmethoden. Was auf den ersten Blick einfach erscheint, stellt sich bei näherer Betrachtung als eine recht hohe Hürde dar.

Zu bestimmen, ob zwei Clusterings gleich sind oder nicht, kann ohne Probleme erfolgen, indem man jedes Cluster des einen Clusterings mit allen anderen des anderen vergleicht und abbricht, sobald dessen Elementenmenge mit keinem Cluster des anderen übereinstimmen. Was ist aber, wenn man verschiedene Clustermengen hat und feststellen möchte, wie ähnlich sie einander sind? Wenn man jetzt anfängt, feststellen zu wollen, wieviele Elemente des einen Clusters in der anderen Clustermenge als Einheit noch enthalten ist, kommt man laufzeittechnisch ins Bedrängnis und muss schon bald angesichts der Komplexität dieses Problems aufgeben. Alternativ bedient man sich existierender statistischer Vergleichsmethoden, wie z.B. *Cophenetic Similarity*.

„*The cophenetic similarity of two objects a and b is defined as the similarity level at wich objects a and b become members of the same cluster during the course of clustering.*“ [40]

Diese Methode ist ursprünglich für hierarchische Clustermengen definiert, sie lässt sich aber auch auf die in unserem System benutzten „flachen“ Clusterings, anwenden, wenn man diese als ein Sonderfall einer hierarchischen Einordnung betrachtet. Eine tiefergehende Betrachtung dieser Methodik bietet, z.B. [4].

Obwohl in dieser Arbeit nicht mehr in Betracht gezogen, sind eine Reihe weiterer Ansätze denkbar. Zum Beispiel besteht die Möglichkeit unser Problem auf *Kreuzklassifikation* [31] unter Beachtung der

Konkordanz[4] [30] unter den Clustern, abzubilden und das erhaltene Ergebnis als Maßzahl zu verwenden.

Ergebnisse der Evaluationsphase 2

Die Abbildungen 7.2 – 7.6 veranschaulichen die Ergebnisse dieser Evaluationsphase mit k-Means als Kernclusteringalgorithmus. Die Abbildungen 7.8 – 7.12 veranschaulichen dies für DBSCAN als Kernclusteringalgorithmus. Jedes dieser Abbildungen besteht aus fünf Diagrammen, die die Versuche des jeweiligen Benutzers zeigen, an seine ideale Clustermenge mit allen fünf, im Kapitel 6.7 vorgestellten, Algorithmen in maximal fünf Schritten heranzukommen. Bei jedem Schritt wurden vier Werte aufgezeichnet, die alle in dem Intervall $[0, 1]$ liegen:

- das Ergebnis der Evaluationsroutine für die Constraints,
- das Ergebnis der Routine für die generierten Constraintpaare,
- die persönliche Bewertung des Benutzers und
- der Vergleichswert zwischen der erzeugten und der Zielclustermenge.

Die persönliche Bewertung des Benutzers sollte seine subjektive Ansicht bzgl. der erzeugten Clustermenge widerspiegeln. Es könnte nämlich sein, dass trotz der schlechten Annäherung an die Zielclustermenge, der Benutzer die Aufteilung interessant und für ihn besonders inspirierend findet. Andererseits könnte es sein, dass, obwohl der Benutzer ein Zielclustering vollkommen spezifiziert hatte, er trotzdem nur auf einige wenige Musikstücke und deren Relationen zu einander fixiert ist, so dass er eine andere Bewertung abgeben könnte als der automatische Vergleich der Clustermengen. Dies wollen wir anhand eines Beispiels aufzeigen. Angenommen, ein Cluster wurde fast vollkommen von dem Algorithmus nachgebildet. Nur zwei Musikstücke gehören nicht dazu. Der Bewertungsalgorithmus würde also einen ziemlich hohen Wert ausgeben. Für den Benutzer sei es so, dass genau diese zwei Beispiele die gesamte Aufteilung zerstören, z.B. wenn auch nur ein Lied des Volksmusikinterpreten Heino in das Cluster, welches nur aus Heavy Metal-Stücken bestehen sollte, zugewiesen wurde. Er gäbe diesem Cluster deswegen eine schlechte Bewertung.

Der Benutzer vergab Schulnoten, eine 1 bedeutete demnach *sehr gut*, die Clustermenge entsprach also vollkommen seiner Vorstellung, und eine 6 *ungenügend*, die Clustermenge entsprach überhaupt nicht seiner Vorstellung. Um diese Benotung besser in den Diagrammen darzustellen, wurden sie auf eine, in der Schule üblichen Weise[5], auf das Intervall $[0, 1]$ abgebildet. Die Tabelle 7.7 zeigt die genaue Abbildung.

Schulnote	**Abbildung**
1 (sehr gut)	1
2 (gut)	0,75
3 (befriedigend)	0,66
4 (ausreichend)	0,5
5 (mangelhaft)	0,25
6 (ungenügend)	0

Tabelle 7.7: Abbildungen der Bewertungen des Benutzers, die in den Diagrammen in der Evaluationsphase 2 verwendet werden.

Auch in dieser Evaluationsphase wurden Durchschnitte über alle Benutzer gebildet. Diese zeigen die Abbildungen 7.7 und 7.13.

[4] Konkordant = übereinstimmend, gleichläufig.

[5] In der Schule musste man mindestens 50% der Punkte erreichen, um die Note 4 (ausreichend) zu bekommen, 2/3 der Punktzahl für die Note 3 (befriedigend), 3/4 der Punkte für die 2 (gut), usw.

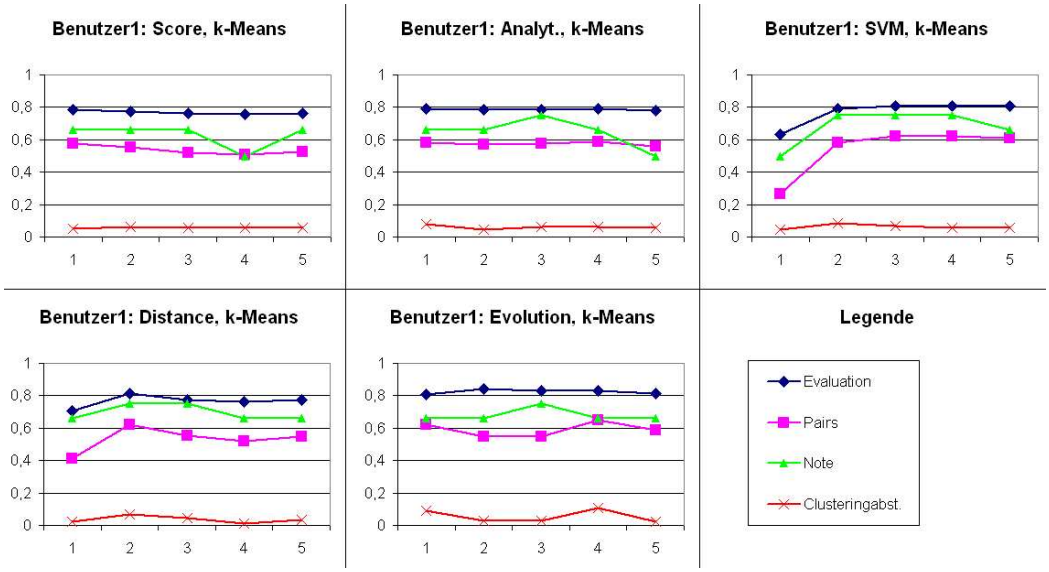


Abbildung 7.2: **Evaluationsphase 2 des Benutzers 1 mit k-Means.** Es sind für alle fünf Optimierungsmethoden fünf Schritte dargestellt, in denen der Benutzer sein Ziel erreichen sollte.

Die Constraintsdefinitionsphase war für den Erfolg dieser Phase entscheidend und wird deswegen auch detailliert dokumentiert. Dabei war die Reihenfolge, in der das Constraint Number of Cluster angewendet wurde, interessant. Wenn in dem ersten Schritt dieses Constraint angewandt wurde, waren aufgrund dessen nicht standardmäßigen Behandlung, das Constraint immer vollkommen erfüllt, was man von der Annäherung an das Zielclustering nicht sagen konnte. Deswegen weisen alle Diagramme an dieser Stelle einen Abfall der Werte, obwohl eigentlich erst dann die weiteren Constraints berücksichtigt wurden. Für Constraintpaare konnte an dieser Stelle kein Wert ermittelt werden, da keine Paare erzeugt wurden. Bei k-Means blieb die Anzahl der Cluster auch in den nachfolgenden Schritten erfüllt, während bei DBSCAN, aufgrund dessen, dass die Anzahl der Cluster nicht in den Parametern des Algorithmus einstellbar waren, diese Anzahl variierte. Nichtdestotrotz war gerade dieses Constraint für die Benutzer sehr wichtig, so dass DBSCAN deswegen eine durchweg niedrigere Benutzerbewertung aufweist. Es wurde nicht seitens des Benutzers gewagt andere Constraints als Instance Association und Number of Clusters zu verwenden, da man sich nicht die genauen Auswirkungen vorstellen konnte. Nur der Benutzer 3 hat noch zusätzlich das Constraint Cluster Cardinality verwendet, als er feststellen musste, dass ein Cluster seiner Vorstellung nach zu unterbesetzt war. Insgesamt wurde das über mehrere Verfahren wiederkehrende Setzen der Constraints als sehr ermüdend und nicht motivierend empfunden, insbesondere deswegen, weil die für die Auswertung relevante Werte bei jedem Schritt ermittelt und abgespeichert werden musste. Andererseits wird kaum ein Benutzer beim täglichen Verwenden dieses Systems einen solchen Aufwand betreiben wollen, können und müssen.

Bei vielen Diagrammen sieht man, dass von einem Schritt zum anderen sich kaum signifikante Änderungen ergeben, die erzeugten Clustermengen gleichen sich in diesen Fällen oft. Dies liegt an dem Umstand, dass nicht genügend zusätzliche Constraints spezifiziert wurden, bzw. die veränderten Constraints auf eine ähnliche Weise den Gewichtsvektor modifizierten wie die vorherigen. Es konnte allerdings nicht genau ermittelt werden, ab wann eine Veränderung erfolgen konnte. Mal war es eine einzige Zuweisung, die die komplette Zielclustermenge veränderte, mal reichten dazu nicht einmal zehn Zuweisungen. Man müsste auf eine für den Benutzer einfach zu verstehende Weise die Ähnlichkeit der Musikstücke zueinander aufzeigen, damit er eine Richtlinie hat, welche Zuweisungen in welchem Maße eine Veränderung für das erzeugte Clustering hervorbringen.

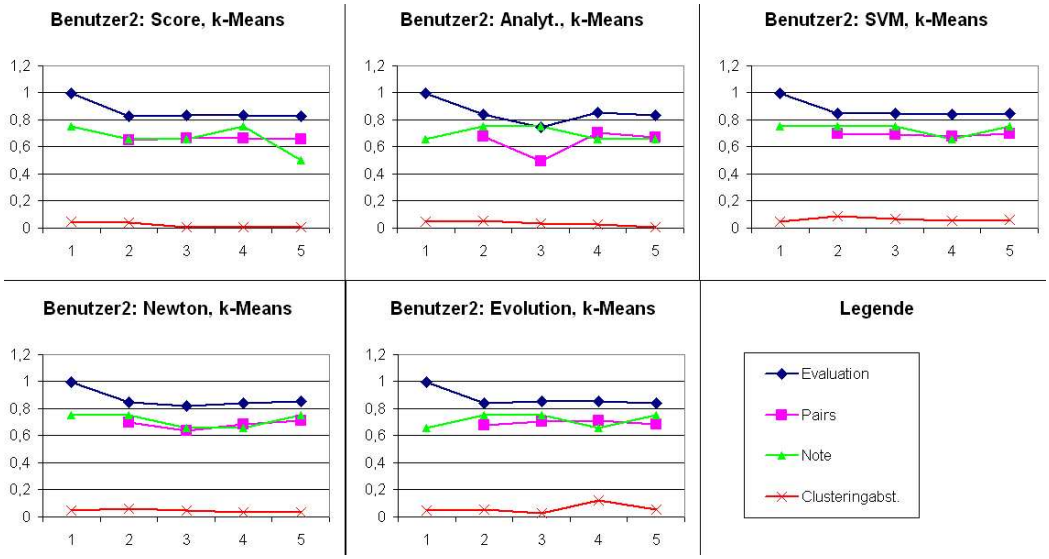


Abbildung 7.3: **Evaluationsphase 2 des Benutzers 2 mit k-Means.** Es sind für alle fünf Optimierungsmethoden fünf Schritte dargestellt, in denen der Benutzer sein Ziel erreichen sollte.

Wie man an den Diagrammen sieht, konnte praktisch keine signifikante Annäherung mit keinem Verfahren an das Zielcluster festgestellt werden. Hier bestätigt sich unser Verdacht, dass die Abbildung von den Constraints auf die Absicht des Benutzers nicht ausreichend ist. Ganz anders steht die Lage bei der Bewertung der erfüllten Constraints aus. Man kann erkennen, dass innerhalb der ersten drei Schritten die größte Verbesserung erzielt wird, während, die nächsten zwei Schritte kaum zu der Verbesserung beitragen. Die Bewertung der Erfüllung der Constraintspaare hat zwar andere, meist niedrigere Werte, entspricht aber dem Verlauf der Bewertungen der erfüllten Constraints.

Die Bewertung des Benutzers bleibt durchschnittlich in dem Bereich der Schulnoten 3 bis 4. Dies ist nicht überraschend angesichts der schlechten Ergebnisse. Die Form der Kurven aber entspricht oft den Kurven der objektiven Bewertung der Clustermengen bzgl. der Constraints. Wenn diese einen höheren Wert aufweisen, dann ist auch die Benutzernote meist besser. Die Benotung ist im Durchschnitt überraschend gut ausgefallen angesichts der geringen Annäherung an die Zielclustermengen. Offensichtlich ist der Benutzer schon damit zufrieden, wenn er einige seiner Ideen von der Clustermenge repräsentiert sieht. Er scheint ebenfalls offen dafür zu sein, neue Vorschläge zu akzeptieren, sich sozusagen dem System anzupassen. Dies ist eine interessante Erkenntnis, die Annäherung geschieht also von beiden Seiten, das System versucht die Absicht des Benutzers zu erraten, aber auch der Benutzer ist so tolerant, sei es aus Offenheit für Neues oder aus Faulheit an den Details zu feilen, und akzeptiert auch Modelle, die noch nicht einmal im Ansatz seiner Vorstellung entsprechen. Diese These ist zwar mitnichten anhand der wenigen, in dieser Arbeit durchgeführten, Experimente untermauert, es bedarf dafür weiterer Forschung auf diesem Gebiet.

7.2.3 Benutzerverhalten

Die Benutzer, die zu den oberen Tests hinzugezogen wurden, hatten auch die Möglichkeiten, sich zu dem Verfahren zu äußern und sowohl die Sinnhaftigkeit des Systems als auch die Schwierigkeit der Bedienung zu bewerten.

Im Allgemeinen hatten die Benutzer keine Schwierigkeiten, eigene Kriterien, nach denen sie die Musikstücke aufgeteilt haben wollten, zu finden und die Musikstücke eindeutig nach ihnen zu ordnen. Auch erschien durchweg allen das System als eine sinnvolle und praxisnahe Anwendung. Viele bemängelten allerdings die Eindeutigkeit der Zuweisungen. Sie würden gerne einige Musikstücke mehreren Gruppen

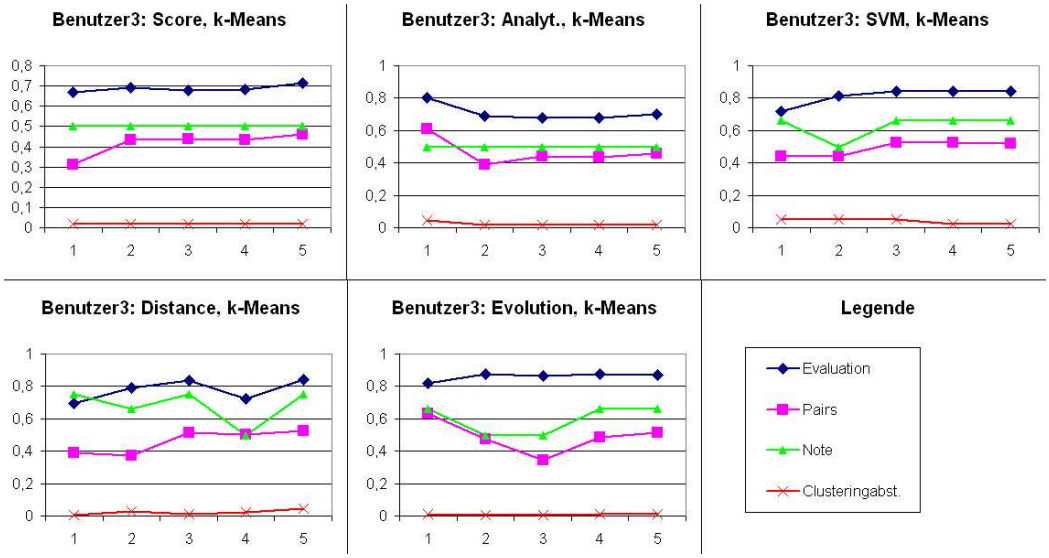


Abbildung 7.4: **Evaluationsphase 2 des Benutzers 3 mit k-Means.** Es sind für alle fünf Optimierungsmethoden fünf Schritte dargestellt, in denen der Benutzer sein Ziel erreichen sollte.

zuweisen können, ohne die Absicht, die Gruppen verschmelzen lassen zu wollen. Es ist wohl die Aufgabe späterer Arbeiten, die Möglichkeiten und Grenzen der Mehrfachzuweisungen in semiüberwachten Systemen, wie dem unseren, zu untersuchen.

Auch wurde die Sinnhaftigkeit der Constraints Value Association (Kap. 4.4.2), Value Separation (Kap. 4.4.3), Cluster Adherence (Kap. 4.4.7) und Cardinality Ratio (Kap. 4.4.6) angezweifelt. Es war sowohl sehr schwierig zu erklären, worum es sich dabei handelt, als auch die Benutzer zu motivieren überhaupt andere Constraints als Instance Association (Kap. 4.4.1) und Number of Clusters (Kap. 4.4.4) zu verwenden. Zudem waren die meisten Merkmale für die Benutzer nichtssagend und sie konnten sich nicht vorstellen, wie die Einstellungen bzgl. der Merkmale sich auf die endgültige Aufteilung in Cluster, auswirken würden. Aus diesem Grunde haben die meisten auf die Verwendung dieser Constraints verzichtet. Vielleicht könnte man durch eine benutzergerechtere Präsentation der kritischen Constraints in späteren Arbeiten etwas mehr Einsichtigkeit erreichen.

Die Benutzer verbrachten im Mittel überdurchschnittlich viel Zeit (ca. 1/2 h), um ihre Beispiele anzugeben. Zum einen lag es daran, dass sie, obwohl es nicht nötig war, aus Spaß und Neugier ihnen unbekannte Stücke anhören wollten. Zum anderen konnten viele sich nicht an Interpreten oder Titel der ihnen bekannten Stücke erinnern und mussten deswegen ebenfalls das eine oder andere Stück probehören. Diese Beobachtung gibt zu denken, vor allem angesichts der Annahme, dass durchschnittliche Anwendungen mehrere tausend Musikstücke enthalten sollten. Auf der einen Seite macht es Spaß, neue Musikstücke zu entdecken, andererseits besteht die Gefahr, dass ein vielbeschäftigter Benutzer nicht soviel Zeit für diese Aufgabe aufbringen möchte.

Obwohl jeder der Benutzer andere Kriterien für die Partitionierung ersann, konnten sich doch gemeinsame Trends kristalisieren. Ein jeder hatte mindestens ein Cluster für Musikstücke, die ihm nicht gefielen, bzw. nicht in die Aufteilung passten, reserviert, während nicht jeder ein Cluster für seine bevorzugten Stücke extra reservierte. Die meisten teilten die Cluster nach Stimmungslage, verbunden mit Tageszeiten, gesellschaftlichen Ereignissen[6], Lokalitäten[7], oder ihren Aktivitäten[8], ein. Es gab aber auch einige, die nach formalen Kriterien einordneten. Interessant war dabei ein Fall, bei dem der Benutzer nach der lexikografischen Ordnung der Namen der Interpreten vorgegangen ist, also ins erste

[6] Z.B. Geburtstag, Kaffeetrinken, Party, ...
[7] Z.B. Diskothek, Kneipe, Daheim, im Auto, ...
[8] z.B. Lesen, Enstpannen, Aerobic, Karate, Lernen, Programmieren, ...

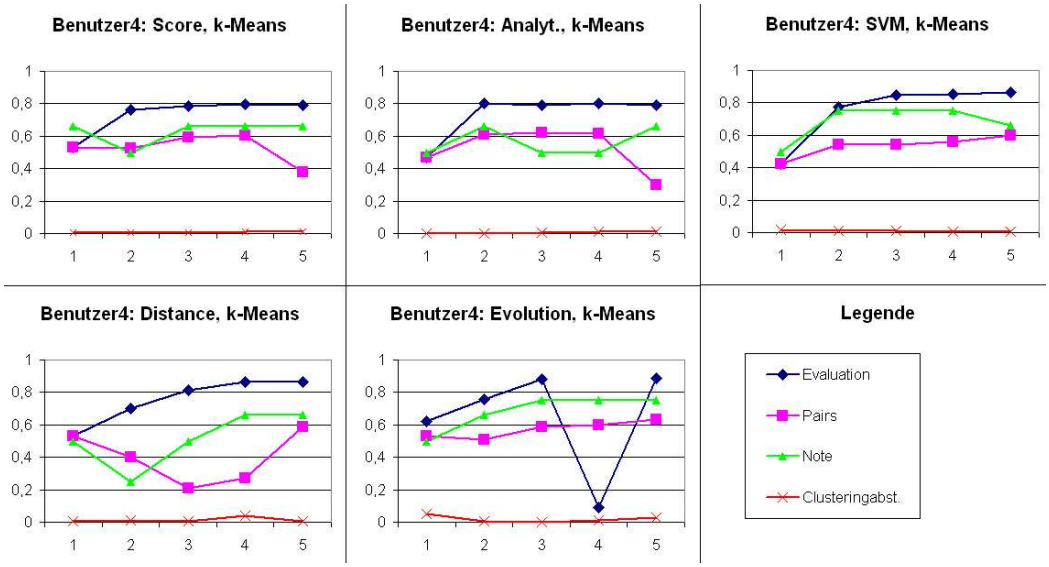


Abbildung 7.5: **Evaluationsphase 2 des Benutzers 4 mit k-Means.** Es sind für alle fünf Optimierungsmethoden fünf Schritte dargestellt, in denen der Benutzer sein Ziel erreichen sollte.

Cluster kamen alle Musikstücke, deren Interpreten mit dem Buchstaben „A. . . D“ anfingen, in das zweite alle mit „E. . . H“, etc. Bzgl. unserer Merkmale ist dies eine sinnlose Aufteilung, für den Benutzer aber offensichtlich kein unmögliche. Dies ist ein gutes Beispiel für das, in den Kapiteln 6.6.2 und 7.1 angesprochene, Problem mit der verwendeten Merkmalsmenge. Interessanterweise hat keiner der Benutzer versucht, nach der sonst geläufigen Klassifikation nach den Genres die Musikstücke gruppieren zu wollen.

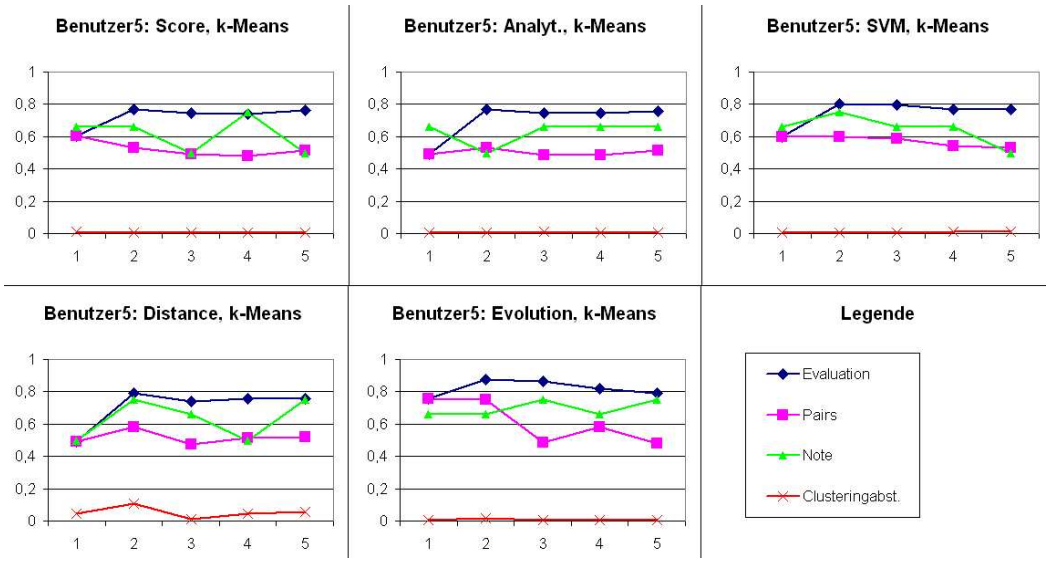


Abbildung 7.6: **Evaluationsphase 2 des Benutzers 5 mit k-Means.** Es sind für alle fünf Optimierungsmethoden fünf Schritte dargestellt, in denen der Benutzer sein Ziel erreichen sollte.

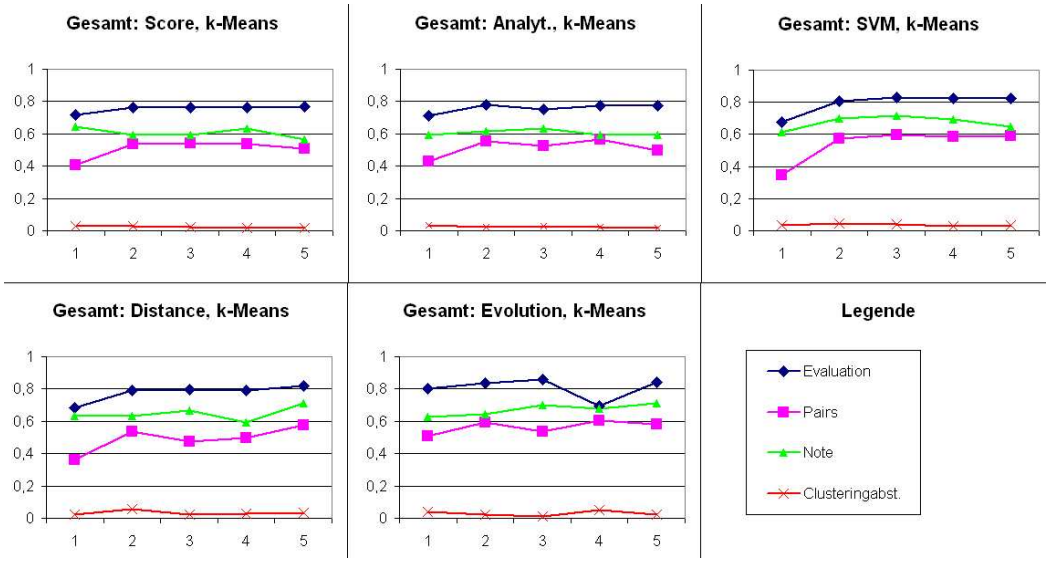


Abbildung 7.7: **Evaluationsphase 2 mit dem Durchschnitt über alle Benutzer mit k-Means.**

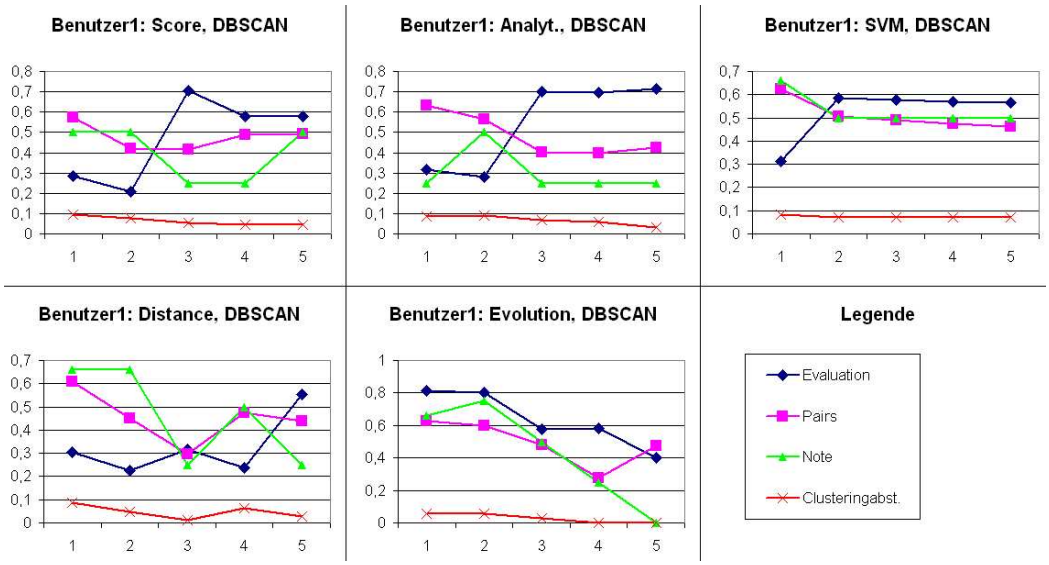


Abbildung 7.8: **Evaluationsphase 2 des Benutzers 1 mit DBSCAN.** Es sind für alle fünf Optimierungsmethoden fünf Schritte dargestellt, in denen der Benutzer sein Ziel erreichen sollte.

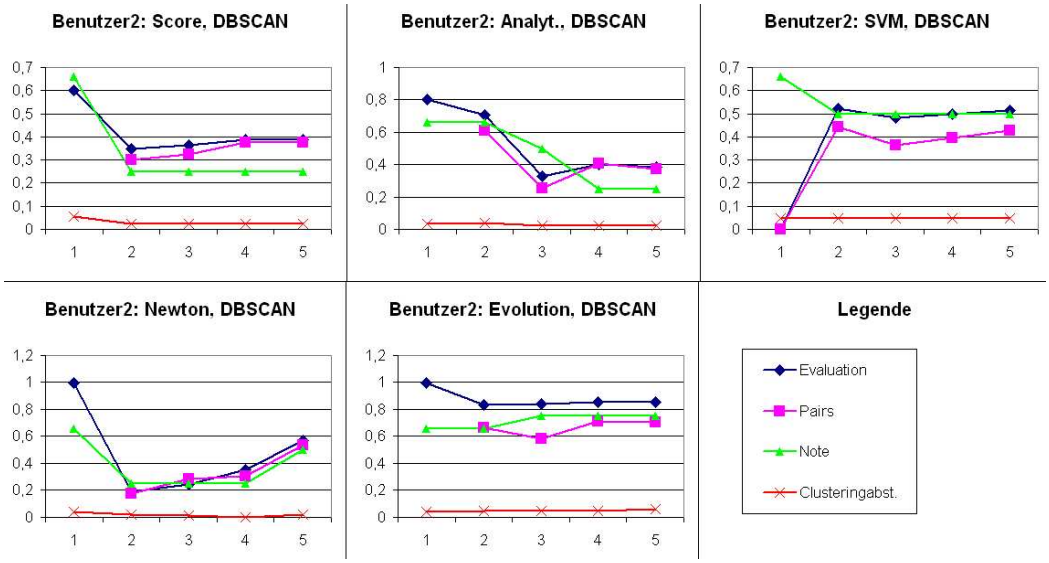


Abbildung 7.9: **Evaluationsphase 2 des Benutzers 2 mit DBSCAN.** Es sind für alle fünf Optimierungsmethoden fünf Schritte dargestellt, in denen der Benutzer sein Ziel erreichen sollte.

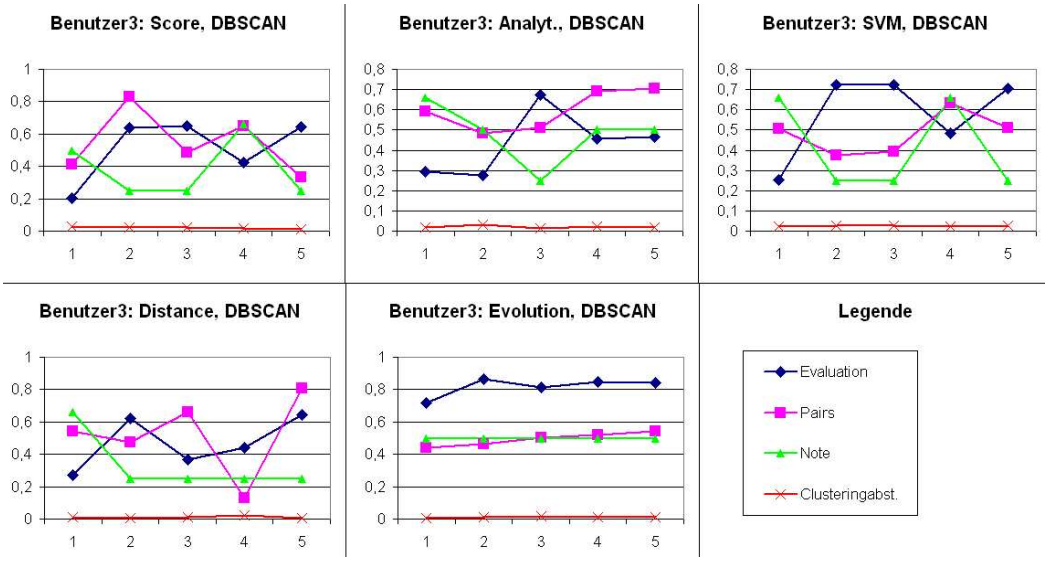


Abbildung 7.10: **Evaluationsphase 2 des Benutzers 3 mit DBSCAN.** Es sind für alle fünf Optimierungsmethoden fünf Schritte dargestellt, in denen der Benutzer sein Ziel erreichen sollte.

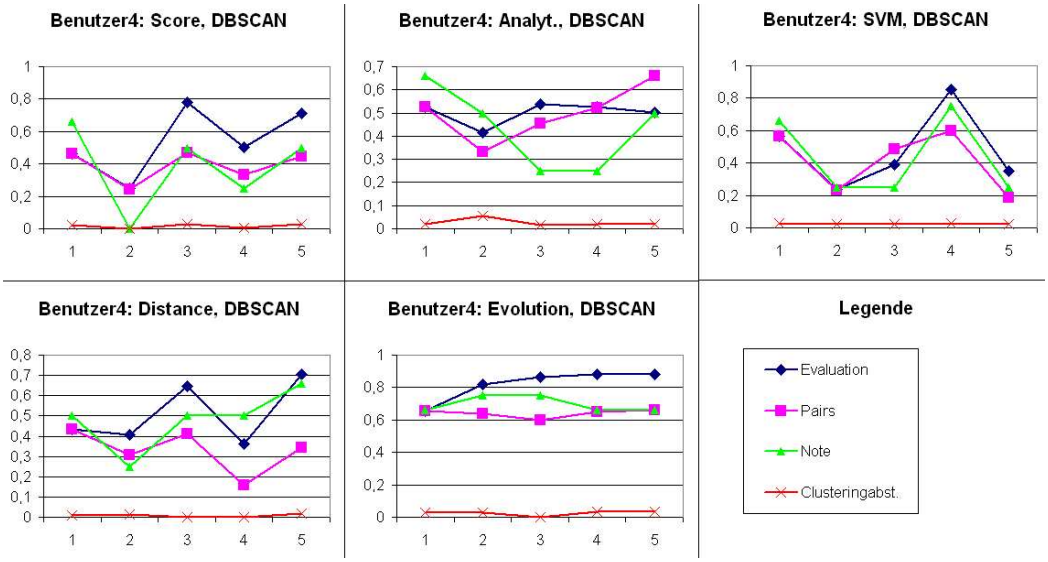


Abbildung 7.11: **Evaluationsphase 2 des Benutzers 4 mit DBSCAN.** Es sind für alle fünf Optimierungsmethoden fünf Schritte dargestellt, in denen der Benutzer sein Ziel erreichen sollte.

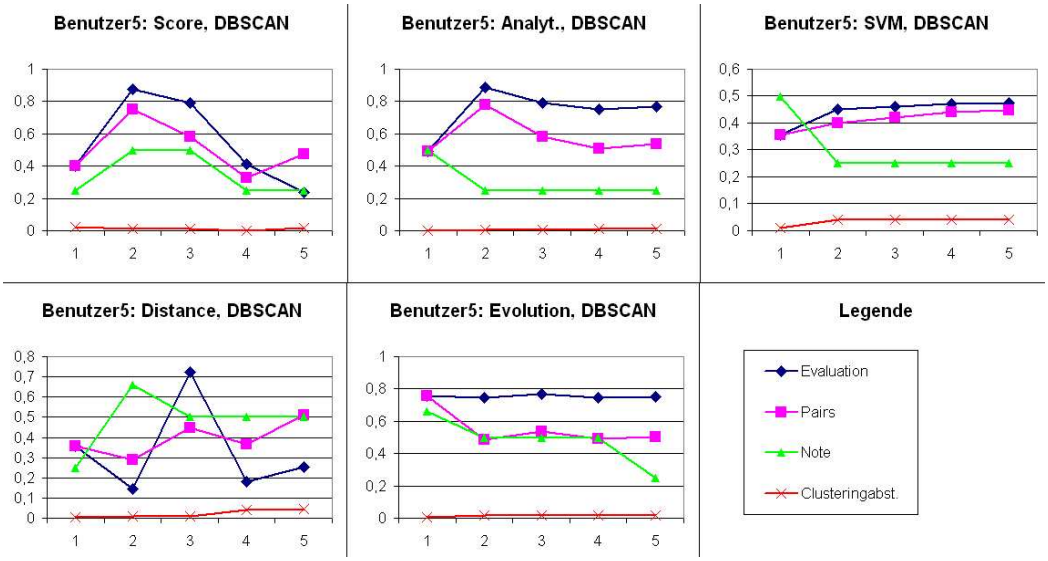


Abbildung 7.12: **Evaluationsphase 2 des Benutzers 5 mit DBSCAN.** Es sind für alle fünf Optimierungsmethoden fünf Schritte dargestellt, in denen der Benutzer sein Ziel erreichen sollte.

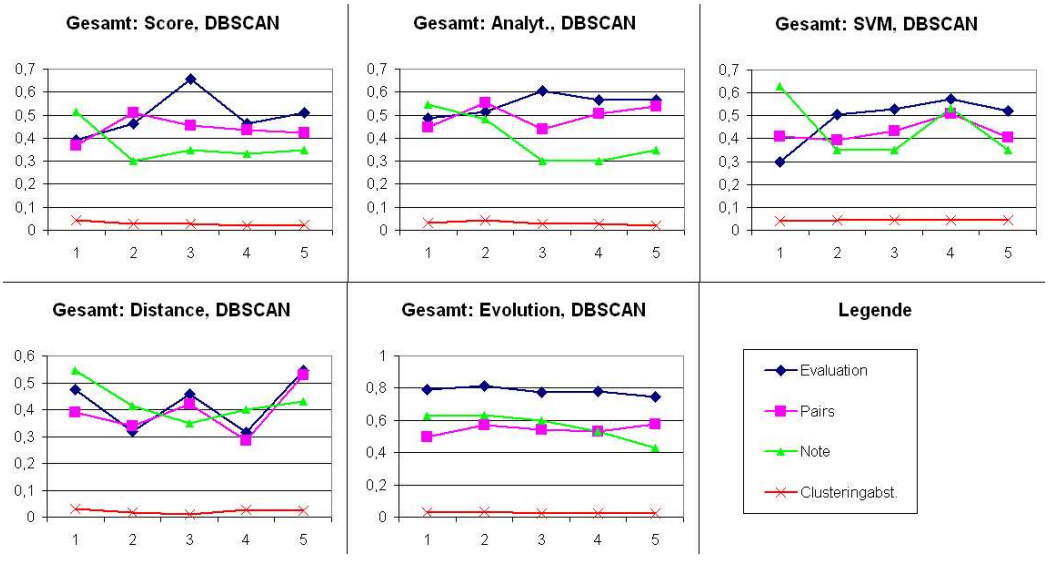


Abbildung 7.13: **Evaluationsphase 2 mit dem Durchschnitt über alle Benutzer mit DBSCAN.**

7.3 Weitere Evaluation

Die Ergebnisse in den Evaluationsphasen 1 und 2 erfordern eine weitergehende Suche nach den Gründen für das aufgezeigte Verhalten.

Als mögliche Ursache für die erhaltenen Ergebnisse konnten drei Hauptkriterien lokalisiert werden:

1. Die nicht ausreichende Anzahl der Audiomerkmale
2. Der Constrained Clustering-Ansatz
3. Die Abbildung der Constraints auf die Constraintpaare.

Im Folgenden wird für diese Kriterien geprüft, welches Gewicht ihnen im Einzelnen bei dem Verhalten des Systems zugewiesen werden kann.

7.3.1 Überprüfung der Audiomerkmale

In dem Kapitel 7.1 wurde bereits darauf hingewiesen, dass die Audiomerkmale nicht ausreichen, um alle Bedürfnisse des Benutzers zu repräsentieren. Dies wollen wir empirisch überprüfen.

Wir gehen von der im Kapitel 3.1 vorgestellten Audiomerkmalsmenge als Repräsentation für die Musikstücke aus. Um den Einfluss des Systems so weit wie mögliche zu verringern, gehen wir für dieses Experiment von *semisupervised*- zu *supervised*-Ansätzen über. Wir verzichten auf Clustering und verwenden Klassifikation. Es werden also Klassen vorgegeben und versucht die Musikstücke mit diesen Klassen zu „identifizieren“ (Kap. 2.1). Dabei sollte der Gewichtsvektor weiterhin der ausschlaggebende Parameter bleiben, da man ansonsten nichts über den Einfluss der Merkmale herausfinden könnte.

Wir versuchen mit einem Klassifizierer, K-Nearest Neighbours (kNN) [3], eingesetzt in einen Evolutionären Algorithmus, mit Hilfe des Gewichtsvektors eine vorgegebene Clustermege zu erreichen. Als Zielclustermenge verwenden wir die des Benutzers 1 aus der Evaluationsphase 2 für k-Means (Abbildung 7.2). Wir wollen damit prüfen, wie gut ein Klassifikationsalgorithmus die vorgegebene Klassifikation mit den gegebenen Audiomerkmalen erreichen kann. Wenn die Audiomerkmale ausreichten, dann kann man davon ausgehen, dass die Musikstücke zu mindestens 90% richtig eingeordnet werden.

Das Ergebnis belegt unsere Vermutung, dass die Merkmalsmenge nicht ausreichend ist. Wir erhielten eine Genauigkeit (engl. *accuracy*) von 57,55%. Es konnten also nur 60% der Musikstücke mit dem Klassifizierer richtig eingeordnet werden. Eine Genauigkeit von 50% kann man schon mit einfachem Würfeln erreichen und unser Ergebnis ist nur etwas besser. Da bereits für Klassifikation ein so niedriges Ergebnis erzielt wurde, können wir es für Clustering noch weniger erwarten. Da sich unser gesamtes System auf diese Merkmale stützt, können wir davon ausgehen, dass sie die Hauptursache für das festgestellte Verhalten sind.

7.3.2 Überprüfung des Constrained Clustering-Ansatzes

Um zu überprüfen, inwiefern der Constrained Clustering-Ansatz als Verfahren geeignet ist, müssen die Constraints sich nur auf ein Merkmal beziehen. So kann man den Einfluss eines fehlenden Audiomerkmals ausschließen. Zwei Versuche wurden durchgeführt:

Es wurde zunächst ein neues, binäres, Merkmal hinzugefügt, *classic*, welches für jedes Musistück indiziert, ob es ein klassisches Musikstück ist oder nicht. Es wurden mit dem Constraint Instance Association Beispielzuweisungen gebildet, die klassische Musikstücke einem Cluster zuordneten. Es konnte über 15% Verbesserung gegenüber dem Ausgangszustand erzielt werden, bei EAs sogar über 60%.

Als nächstes wurde ein weiteres binäres Merkmal hinzugefügt, *pop*, welches die Musikstücke aus dem Genre der Pop-Musik identifizierte. Es wurden erneut Zuweisungen bzgl. dieser Merkmale durch das Constraint Instance Association durchgeführt. Diesmal erhielt man durchweg eine Verbesserung von mehr als 40%.

Aufgrund dieser Ergebnisse folgt die Behauptung, dass der Constrained Clustering-Ansatz an sich sinnvoll ist und nicht die Ursache für das Verhalten des Systems darstellt.

7.3.3 Abbildung von Constraints auf Constraintpaare

Die Abbildung der Constraints auf Constraintpaare gab schon in der Evaluationsphase 1, aufgrund der unterschiedlichen Werte der Evaluation der Constraints und der Paare, Grund zu der Annahme, dass sie keine genügend genaue Abbildung darstellen. Andererseits ist die Abbildung der Paare heuristischer Natur, so dass nicht analytisch belegt werden kann, wie gut die Auswertung vorgenommen wird. Ein Indiz für die schlechte Güte der Abbildung könnten die enormen Unterschiede der Ergebnisse des EA zu den anderen Verfahren sein. Jedoch beruht diese Beobachtung auf einer bekannten Tatsache. Sie hat einen Pendant in den, im Kapitel 6.4 vorgestellten Unterschieden zwischen *performance-* und *preset-bias*, bzw. zwischen *wrapper-* und *filter-*Ansätzen. Wrapper-Ansätze, in unserem Fall der EA, optimieren direkt die Performanz und führen daher meist zu besseren Ergebnissen [3].

7.3.4 Fazit und Folgerungen

Es hat sich herausgestellt, dass die im Kapitel 3.1 vorgestellte Menge der Audiomerkmale nicht ausreicht, um beliebige Constraints zu erfüllen. Diese Feststellung überschattet die Ungenauigkeiten bei der Abbildung der Constraints, da es ein Problem grundlegenderer Natur ist. Es hat die Auswertungen in den Evaluationsphasen hauptsächlich verzerrt.

Als mögliche Folgerungen aus dieser Feststellung kommen folgende Ansätze in Frage:

1. Der Umstieg auf Merkmale, die von dem Benutzer generiert wurden, also auf Metamerkmale oder andere Informationen bzgl. der Musikstücke.
2. Verwendung der Metamerkmale aus den Musikdatenbanken aus dem Internet (siehe Kap. 3.2).
3. Es kann versucht werden, schnellere *feature construction* zu betreiben.
4. Als ein weiterer Ansatz bietet sich *Kollaboratives Clustering* [77] an.

8 Zusammenfassung und Ausblick

In der Einleitung wurden Kriterien für das System konzipiert und Ziele gesetzt, die damit erreicht werden sollten. In diesem Kapitel wollen wir überprüfen, inwiefern uns dies gelungen ist. Des Weiteren werden offene Fragen zusammengetragen, um eine Übersicht darüber zu geben, was für nachfolgende Arbeiten zu beachten und zu untersuchen wäre.

Das System sollte zunächst *interaktiv* sein. Dies ist vom Konzept her der Fall, da der Benutzer stets die Möglichkeit hat, in den Prozess einzugreifen. Allerdings hapert es dabei an der Effizienz. Wie im Kapitel 7 beschrieben wurde, sind nicht alle Verfahren immer effizient, manche, z.B. die SVMs, haben sogar im worst-case unakzeptable Laufzeiten.

Die Benutzer empfanden das System durchweg als *verständlich* und hatten keine Probleme ihren Willen kundzutun. Diese Aussage ist allerdings auf einige ausgewählte Constraints beschränkt. Mit einigen der Constraints konnten die Benutzer wenig oder gar nichts anfangen (siehe dazu Kap. 7.2.3).

Die Abbildung auf die systeminterne Repräsentation enthielt offensichtlich einige Tücken. Nicht nur dass ziemlich viele Constraintpaare gebildet wurden, es entstand auch durchweg der Verdacht, dass durch diese Art Abbildung der Informationsgehalt der Constraints nur zum Teil vermittelt wurde. Die Menge der Paare erzeugte zu viele widersprüchliche Informationen, so dass sie sich gegenseitig abschwächten. Die *Verständlichkeit für die Maschine* war also nicht vollkommen gewährleistet.

Es ist uns gelungen ein *iteratives System* aufzubauen, so dass der Benutzer stets Änderungen kann.

Dem Benutzer bleibt überlassen, inwiefern er in das System eingreifen möchte. Er kann sich mit dem Vorclustering zufrieden geben, er kann aber auch alles neu nach seiner Vorstellung gestalten. Somit ist die *minimale Benutzerbeteiligung* gegeben. Das Einzige, was man dagegen anführen könnte, ist, dass bei evtl. zu wenigen Änderungen keine Verbesserung erzielt werden könnte (siehe dazu Kap. 7.2.2).

Die *Integration in die Entwicklungsumgebung* YALE ist von vorne herein geplant und abgeschlossen. Dies wurde im Kapitel 6.8 erläutert.

Insgesamt haben wir durchweg keine zufriedenstellende Ergebnisse erhalten, obwohl wir fünf Verfahren zum Vergleich angewendet haben. In der Evaluationsphase 1 wurde nur ein Bruchteil der Verbesserung erzielt, welche möglich waren, was von den Ergebnissen des EA belegt wurde. In der Evaluationsphase 2 wurde zusätzlich festgestellt, dass kein erkennbares Konvergenzverhalten in den Iterationen festgestellt werden konnte, obwohl die objektiven Kriterien eine Verbesserung belegten und sogar der durchschnittliche Benutzer eine Verbesserung von einem Schritt zum anderen zu sehen glaubte. Die Untersuchungsergebnisse aus dem Kapitel 7.3 lassen darauf schließen, dass vor allem die gewählte Merkmalsmenge nicht ausreichte, um die Benutzerziele zu erfüllen. Hier bedarf es einer verstärkten Forschungsarbeit, vor allem im Bereich der Merkmalsgenerierung. Zusätzlich wäre der minimale Informationsgehalt zu bestimmen, der von dem Benutzer übertragen werden müsste, damit seine Absichten zu seiner Zufriedenheit von einer Maschine abgeleitet werden können. In diesem Zusammenhang wären die anderen im Kapitel 2.6.3 vorgestellten Ansätze für das Constraint-Clustering interessant, vor allem die Informational Bottleneck-Ansätze.

Diese Erkenntnisse fassen wir noch einmal als Hauptergebnis dieser Arbeit zusammen:

Es ist anzunehmen, dass es keine hinreichend kleine Menge von Audiomerkmalen existiert, welche für alle Aufgaben geeignet ist. Vor allem, wenn der Benutzer sich etwas Ausgefallenes vorstellt, dann benötigt er extra Merkmale, die generiert werden müssen. Dieser Umstand verträgt sich nicht mit dem interaktiven Charakter des Systems, weil es zu viel Zeit in Anspruch nimmt. Andererseits, wenn der Benutzer „standardmäßiger", z.B. nach den Musikgenres vorgeht, dann kann er auf etabliertere Metho-

den, wie Klassifikation, zurückgreifen und benötigt Constrained Clustering nicht. Das bedeutet, dass Audiomerkmale sich prinzipiell nicht mit dem Ansatz Constrained Clustering vereinbaren lassen.

Es folgen Zusammenfassungen weiterer Erkenntnisse dieser Arbeit:

Der Vergleich unter den einzelnen Algorithmen konnte trotz allem erfolgen. Man sieht, dass mit der Hinzunahme von zusätzlicher Information und durch die Verwendung aufwendigerer Optimierungsmethoden, man auch in diesem Bereich bessere Ergebnisse erzielen kann.

Auch konnte im Kapitel 7.3 nachgewiesen werden, dass der Constrained Clustering-Ansatz an sich die von ihm geforderte Aufgabe zu bewältigen vermag.

Der Evolutionäre Algorithmus konnte sich besonders auszeichnen. Zum einen erhielt man mit ihm überdurchschnittlich gute Ergebnisse, und zum anderen in einer akzeptablen Zeit (Siehe Kap. 7). Dies konnte im Kapitel 7.3 damit begründet werden, dass es sich um einen *wrapper*-Ansatz handelt [3]. Er wäre ein guter Startpunkt für die weitere Suche nach den Verbesserungen in dem System.

Im Kapitel 7.1 wurden einige kritische Fragestellungen aufgeworfen, die weiterhin unklar bleiben und eine Untersuchungsbasis für die nachfolgenden Arbeiten bieten.

Die Bewertung des Benutzers unterschied sich noch stark von dem Prozess des Ergebnisgewinns der Evaluationsalgorithmen. Es bleibt offen, ob man sie nicht weiter standardisieren und formalisieren könnte, um die Bewertungen einander anzunähern.

Nicht zuletzt könnte, wie es schon im Kapitel 6.6 angedeutet wurde, die von uns verwendete Metrik mit dem Euklidischen Distanzmaß, die Güte der Ergebnisse zu stark verzerren. Die Frage inwieweit diese Vermutung der Wahrheit entspricht und inwiefern andere Metriken besser geeignet wären, muss nachfolgenden Arbeiten überlassen werden.

Diese Arbeit behandelt ein relativ neues Gebiet des Constrained Clustering, in dem zwar stark geforscht wird, sich aber weder standardisierte Begriffe, noch eindeutige Herangehensweisen etabliert haben. Es wurde Versucht eine Vereinigung der verschiedenen in der Literatur vorkommenden Constraintarten aufzustellen und diese auf eine einheitliche Struktur der Must- und Cannot-Links sinnvoll abzubilden. Es wurden mit einer fest vorgegebenen Merkmalsmenge für fünf verschiedene Verfahren verglichen, wie gut sie diese Constraints erfüllen können. Dabei wurde eine Reihe von Problemen, Fragen und alternativen Ansichten beleuchtet. Zentral stellte sich heraus, dass der Constrained Clustering-Ansatz die von ihm geforderte Aufgabe zu erfüllen vermag, jedoch reicht die Ausgewählte Merkmalsmenge nicht aus, um alle Bedürfnisse des Benutzers abzudecken. Es bleibt zu hoffen, dass die aus dieser Arbeit gewonnenen Erkenntnisse dazu beitragen, die Forschung auf diesem Gebiet zu stärken und voranzubringen.

Literaturverzeichnis

[1] AGRAWAL, R., GEHRKE, J., GUNOPULOS, D., RAGHAVAN, P.: *Automatix subspace clustering of high dimensional data for data mining applications.*. In: *Proceedings of the ACM SIGMOD Conference*, S. 94–105, 1998.

[2] AHA, D.W., BANKERT, R.L.: *Feature selection for case-based classification of cloud types: An empirical comparison*. Case-Based Reasoning: Papers from 1994 Workshop, 1994.

[3] AHA,D.W.: *Feature weighting for lazy learning algorithms.*. Feature Extraction, Construction and Selection: A Data Mining Perspective, 1998.

[4] ANDERBERG, M.R.: *Cluster Analysis for Applications*. Academic Press, 1973.

[5] ANKERST, M., BREUNIG, M., KRIEGEL, H.-P., SANDER, J.: *OPTICS: Ordering points to identify clustering structure*. In: *Proceedings of the ACM SIGMOD Conference*, S. 49–60, 1973.

[6] BAMBERG, BAUR: *Statistik*. Oldenbourg, München, 10 Aufl., 1998.

[7] BASU, S., BANERJEE, A., MOONEY, R. (Hrsg.): *Semi-superviced Clustering by Seeding*. Nineteenth International Conference on Machine Learning, 2002.

[8] BEN-HUR, A., HORN, D., SIEGELMANN, H. T., VAPNIK, V.: *Support Vector Clustering*. Journal of Mashine Learning Research, Februar 2001.

[9] BERKHIN, P.: *Survey of Clustering Data Mining Techniques*. Techn. Ber., Accrue Software, Inc., 2000.

[10] BILENKO, M., BASU, S., MOONEY, R.J. (Hrsg.): *Integrating Constraints and Metric Learning in Semi-Superviced Clustering*. Twenteeth International Conference on Machine Learning, 2003.

[11] CARDIE, C.: *Using decision trees to improve case-based learning*. In: *Proceedings of the Tenth International Conference on Machine Learning*, S. 25–32, 1993.

[12] CHENG, C.H., FU, A., ZHANG, Y.: *Entropy-based Subspace Clustering for Mining Numerical Data*. In: *Knowledge Discovery and Data Mining*, S. 84–93, 1999.

[13] COX, T., COX, M.: *Multidimensional Scaling*. Chapman & Hall, London, 1994.

[14] CREECY, R.H., MASAND, B.M., SMITH, S.J., WALTZ, D.L.: *Trading MIPS and memory for knowledge engeneering.*. Communications of the ACM, 35:48–64, 1992.

[15] DAVID, E.G., KORB, B., AND DEB, K.: *Messy genetic algorithms: motivation, analysis, and first results*. Complex Systems, 3(5):493–530, 1989.

[16] DEFAYS, D.: *An efficient algorithm for a complete link method*. The Computer Journal, 20:364–366, 1977.

[17] ERTOZ, L., STEINBACH, M., KUMAR, V.: *Finding Clusters of different sizes, shapes and densities in noisy, high dimensional data*. Techn. Ber., Department of Computer Science, University of Minnesota, 2002.

[18] ESTER, M., KRIEGEL, H.P., SANDER, J., XU, X.: *A density based algorithm for discovering clusters in large spatial databases with noise*. In: *Proceedings of the* 2^{nd} *ACM SIGKDD*, S. 226–231, 1996.

[19] FISCHER, S., R. KLINKENBERG, I. MIERSWA und O. RITTHOFF: *Yale: Yet Another Learning Environment – Tutorial*. Techn. Ber. CI-136/02, Collaborative Research Center 531, University of Dortmund, Dortmund, Germany, Juni 2002. ISSN 1433-3325.

[20] FISHER, D.: *Iterative optimization and simplification of hierarchical clustering*. Mashine Learning, 2:139–172, 1987.

[21] GANTI, V., GEHRKE, J., RAMAKRISHNAN, R.: *CACTUS - Clustering Categorical Data Using Summaries*. In: *Knowledge Discovery and Data Mining*, S. 73–83, 1999.

[22] GOIL, S., NAGESH, H., GHOUDHARY, A.: *MAFIA: Efficient and scalable subspace clustering for vers large data sets*. Techn. Ber., CPDC-TR-9906-010 Nordwestern University, 1999.

[23] GONDEK, D., HOFMANN, T.: *Conditional Information Bottleneck Clustering*. In: *3rd IEEE International Conference on Data Mining, Workshop on Clustering Large Data Sets*, 2003.

[24] GONDEK, D., HOFMANN, T.: *Non-Redunant Data Clustering*. In: *4th IEEE International Conference on Data Mining*, 2004.

[25] GRANDVALET, Y., CANU, S.: *Adaptive Scaling for feature selection in SVMs*. In: *Advances in Neural Information Processing Systems 15*, 2003.

[26] GUHA, S., RASTOGI, R., SHIM, K.: *CURE: An efficient clustering algorithm for large databases*. In: *Proceedings of the ACM SIGMOD Conference*, S. 73–84, 1998.

[27] GUHA, S., RASTOGI, R., SHIM, K.: *ROCK: A robust clustering algorithm for categorical attributes*. In: *Proceedings of the* 15^{th} *ICDE*, S. 512–521, 1999.

[28] HALL, L.O., ÖZYURT, I.B., BEZDEK, J.C.: *Clustering with a Genetically Optimized Approach*. IEEE Trans. on Evolutionary Computation, 3(2):103–112, 1999.

[29] HARDHIENATA, S.: *Numerische Optimierungsstrategie für Simulationsmodelle mit Anwendungen in Informatik und Verfahrenstechnik*. Doktorarbeit, Universität Erlangen-Nürnberg, 1993.

[30] HARTUNG, J.: *Multivariate Statistik*. R. Oldenbourg Verlag München Wien, 1984.

[31] HARTUNG, J.: *Statistik*. R. Oldenbourg Verlag München Wien, 13. Aufl., 2002.

[32] HINNEBURG, A., KEIM, D.: *An efficient approach to clustering large multimedia databases with noise*. In: *Proceedings of the* 4^{th} *ACM SIKDD*, S. 58–65, 1998.

[33] HINNEBURG, A., KEIM, D.: *Optimal grid-clustering: towards breaking the curse of dimensionality in high-dimensional clustering*. In: *Proceedings of the* 25^{th} *Conference on VLDB*, S. 506–517, 1999.

[34] HOLLAND, J.H.: *Adaptation in Natural and Artificial Systems*. University of Michigan Press (1975), 1975.

[35] JAIN, A.K., MURTY, M.N., FLYNN, P.J.: *Data Clustering: A Review*. ACM Computing Surveys, 31(3), September 1999.

[36] KARYPIS, G., HAN, E.-H., KUMAR, V.: *CHAMELION: A hierarchical algorithm using dynamic modelling*. COMPUTER, 32:68–75, 1999.

[37] KAUFMANN, L., ROUSSEEUW, P.: *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley and Sons, 1990.

[38] KEOGH, E.J., CHAKRABARTI, K., PAZZANI, M.J., MEHROTRA S.: *Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases*. Knowledge and Information Systems, 3(3):263–286, 2001.

[39] KOHONEN, T.: *The self-organizing map*. In: *Proceedings of the IEEE*, Bd. 9, S. 1464–1479, 1990.

[40] LEGENDRE, P., LEGENDRE, L.: *Numerical Ecology*. Elsevier, Amsterdam, 1998.

[41] LIU, D., LU, L., ZHANG, H.J.: *Automatic Mood Detection from Acoustic Music Data*, 2003.

[42] MARDIA, K., KENT, J., BIBBY, J.: *Multivariate Analysis*. Academic Press, 1980.

[43] MARROQUIN, J.L., GIROSI, F.: *Some extensions of the k-means algorithm for image segmentation and pattern classification, A.I. Memo 1390. MIT, Cambridge*, 1993.

[44] METZ-GOECKEL, H.: *Sript: Allgemeine Psychologie 1*, WS 1999.

[45] MIERSWA, I., R. KLINKBERG, S. FISCHER und O. RITTHOFF: *A Flexible Platform for Knowledge Discovery Experiments: YALE – Yet Another Learning Environment*. In: *LLWA 03 - Tagungsband der GI-Workshop-Woche Lernen - Lehren - Wissen - Adaptivität*, 2003.

[46] MIERSWA, I.: *Automatisierte Merkmalsextraktion aus Audiodaten*. Diplomarbeit, Universität Dortmund, Februar 2004.

[47] MITCHELL, T.: *Machine Learning*. McGraw-Hill, 1997.

[48] MITCHELL, T.M., KELLER, R., KEDAR-CABELLI, S.: *Explanation-based learning: a unifying view*. Machine Learning, 1:47–80, 1986.

[49] MOHRI, T., TANAKA, H.: *An optimal weighting criterion of case indexing for both numeric and symbolic attributes*. Case-Based Reasoning: Papers from 1994 Workshop, 1994.

[50] MOORE, A.W., LEE, M.S.: *Efficient algorithms for minimizing cross validation error*. In: *Proceedings of the Eleventh International Conference on Machine Learning*, S. 190–198, 1994.

[51] NG, R., HAN, J.: *Efficient and effective clustering methods for spatial data mining*. In: *Proceedings of the* 20^{th} *Conference on VLDB*, S. 144–155, 1994.

[52] QUIAN, W.N., ZHOU, A.Y.: *Analyzing Popular Clustering Algorithms from different Vewpoints*. Journal of Software, 13(8), 2002.

[53] RUSSEL, S., NORVIG, P.: *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2. Aufl., December 2002.

[54] SAITTA, L., ET AL.: *Informed Parameter Setting*. Mining Mart, Dezember 2000.

[55] SALZBERG, S.L.: *A nearest hyperrectangle learning method*. Machine Learning, 6:251–276, 1991.

[56] SCHEIKHOLESLAMI, G., CHATTERJEE, S., ZHANG, A.: *WaveCluster: A multi-resolution clustering approach for very large spatial databases*. In: *Proceedings of the* 24^{rd} *Conference on VLDB*, S. 428–439, 1998.

[57] SCHIKUTA, E., ERHART, M.: *The BANG-clustering system: grid-based data analysis*. In: *Proceedings of Advances in Intelligent Data Analysis, Reasoning about Data,* 2^{nd} *International Symposium*, S. 513–524, 1997.

[58] SCHOELKOPF, B., SMOLA, A.J.: *Learning with Kernels*. MIT Press, 2002.

[59] SIBSON, R.: *SLINK: An optimally efficient algorithm for the single link cluster method*. The Computer Journal, 16:30–34, 1973.

[60] SKALAK, D.: *Prototype and feature selection by sampling and random mutation hill climbing algorithms*. In: *Proceedings of the Eleventh International Conference on Machine Learning*, S. 293–301, 1994.

[61] STANFILL, C., WALTZ, D: *Toward memory-based reasoning*. Communications of the Association for Computing Mashinery, 29:1213–1228, 1986.

[62] THOMOPOULOS, S., BOUGOULIAS, D., WANN, C.D.: *DIGNET: an unsuperviced learning clustering algorithm for clustering and data fusion*. IEEE Trans. on Aerospace and Electr. Systems, 31(1):1–38, 1995.

[63] TISHBY, N., PEREIRA, F., BIALEK, W.: *The informational bottleneck method*. In: *37th Allerton Conference of Communication, Control and Computing*, 1999.

[64] TSAI, W.H., WANG, H.M., RODGERS, D., CHENG, S.S., YU, H.M.: *Blind Clustering of Popular Music Recordings Based on Singer Voice Characteristics*, 2002.

[65] TSANETAKIS, G., ESSL, G., COOK, P.: *Automatic Musical Genre Classification Of Audio Signals*, 2001.

[66] TSANG, I.W., KWOK, J.T.: *Distance Metric Learning with Kernels*. In: *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*, S. 126–129, 2003.

[67] VAN RIJSBERGEN, C.J.: *Information Retrieval*. Butterworths, London, Second Aufl., 1979.

[68] VAPNIK, V.: *The Nature of Statistical Learning Theory*. Springer Verlag, New York, 1995.

[69] VITANYI, P., CILIBRASI, R., DE WOLF, R.: *Algorithmic Clustering of Music*, März 2003.

[70] WAGSTAFF, K., CARDIE, C., ROGERS, S., SCHROEDL, S. (Hrsg.): *Constrained K-means Clustering with Background Knowledge*. Eighteenth International Conference on Machine Learning, 2001.

[71] WALLACE, C., DOVE, D.: *Intrinsic Classification by MML - the Snob program*. In: *Proceedings of the* 7^{th} *Australian Joint Conference on Artificial Intelligence*, S. 37–44, 1994.

[72] WANG, W., YANG, J., MUNTS, R.: *STING: a statistical information grid approach to spatial data mining*. In: *Proceedings of the* 23^{rd} *Conference on VLDB*, S. 186–195, 1997.

[73] WEGENER, I.: *Skript: Effiziente Algorithmen*, SS 2002.

[74] WESTON, J., MUKHERJEE, S., CHAPELLE, O., PONTIL, M., POGGIO, T., V. VAPNIK: *Feature Selection for SVMs*. In: *Advances in Neural Information Processing Systems 13*, 2001.

[75] WETTSCHERECK, D, AHA,D.W., MOHRI, T.: *A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms.*. Artificial Intelligence Review, 11:273–314, 1997.

[76] WILSON, D.R., MARTINEZ, T.R.: *Improved Heterogeneous Distance Functions*. Journal of Artificial Intelligence Research, 6:1–34, 1997.

[77] WURST, M. und J. NOVAK: *Knowledge Sharing im Heterogeneous Expert Communities based on Personal Taxonomies*. In: *ECAI Workshop on Agent Mediated Knowledge Management*, 2004.

[78] XING, E.P., NG, A.Y., JORDAN, M.I., RUSSEL, S.: *Distance metric learning with application to clustering with side information*. In: *Advances in Neural Informational Processing Systems*, 2003.

[79] XU, X., ESTER, M., KRIEGEL, H.-P., SANDER, J.: *A distribution-based clustering algorithm for mining in large databases*. In: *Proceedings of the* 14^{th} *ICDE Conference*, S. 324–331, 1998.

[80] YOKOO, M., HIRAYAMA, K.: *Algorithms for Distributed Constraint Satisfaction: A Review*. Autonomous Agents and Multi-Agent Systems, 3(2):198–212, 2000.

[81] ZADEH, L.A.: *Fuzzy Sets*. Inf. Control, 8:338–353, 1965.

[82] ZHANG, T., RAMAKRISHNAN, R., LIVNY, M.: *BIRCH - A new data clustering algorithm and its applications*. In: *Proceedings of the ACM SIGMOD Conference*, Bd. 1, S. 103–114, 1996.

Index