Technische Universität Darmstadt
Fachbereich Informatik

Methoden zur Feature Subset Selection für Unsupervised Learning - Überblick und neuer Ansatz

Diplomarbeit

zur Erlangung des Grades eines Diplom-Informatikers

eingereicht von: Frank Steinmann

Betreuer: Prof. Dr. Johannes Fürnkranz

eingereicht am 21.04.05

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

Lindenfels, 20.04.05

Inhaltsverzeichnis

Kapitel 1

Einleitung

Im Bereich des Data Mining unterscheidet man zwei grundsätzliche Ansätze um Informationen aus Daten zu gewinnen, Supervised Learning und Unsupervised Learning. Zielsetzung und Voraussetzungen der beiden Ansätze sind verschieden. Beim Supervised Learning versucht man aus Daten, die eine Klasseneinteilung enthalten, ein Modell zu lernen, das eben diese Klasseneinteilung beschreibt, während beim Unsupervised Learning unklassifizierte Daten behandelt werden. Hier möchte man Gruppierungen innerhalb der Daten erkennen.

Ein Datensatz besteht üblicherweise aus einer Menge von Instanzen, die sich als Vektor von Attributwerten darstellen lassen. Beim Supervised Learning (supervised, weil dem Lernalgorithmus die Klassifizierung der Instanzen als Information mitgegeben wird) wird nun versucht, Zusammenhänge zwischen den Attributwerten und der Klassifizierung zu erkennen, so dass es möglich ist, neue, noch unklassifizierte Instanzen einer Klasse zuzuordnen. Beim Unsupervised Learning (unsupervised, weil keine Klasseninformation mitgegeben wird) versucht der Algorithmus anhand der Attributwerte ähnliche Instanzen in Gruppen (Clustern) zusammenzufassen. In der Regel stellt eine solche Gruppierung (im folgenden Clustering genannt) eine Partitionierung der Daten dar, d.h. jede Instanz gehört genau einem Cluster an.

Da nun nicht unbedingt alle Attribute (Features) relevant für die Klassifizierungen bzw. Gruppierungen sind, kann es sinnvoll sein eine Auswahl an Attributen zu treffen und irrelevante Attribute oder auch redundante Attribute zu entfernen. Zum einen erspart man den Lernalgorithmen damit Rechenzeit, zum anderen stellt bereits die Attributauswahl einen Informationsgewinn dar, da sie angibt, welche Attribute wichtig sind und welche nicht. Der Vorgang einer Attributauswahl wird *Feature Subset Selection* oder einfach *Feature Selection* genannt.

Im Bereich des Supervised Learning sind Feature Subset Selections bereits sehr weitläufig untersucht worden. Die allgemeine Strategie besteht dar-

in, solche Attribute auszuwählen, mit denen sich das Klassenattribut gut vorhersagen lässt. Schwieriger ist die Situation beim Unsupervised Learning, da hier keine Klasseninformation zur Verfügung steht. Feature Subset Selection für Unsupervised Learning ist vielleicht auch deshalb ein bisher noch wenig untersuchtes Gebiet. Diese Arbeit beschränkt sich auf die Problemstellung der Feature Subset Selection im Bereich Unsupervised Learning.

Die Arbeit ist folgendermaßen gegliedert: In Kapitel 2 werden Methoden zum Erstellen von Clusterings sowie Ansätze zur Bewertung von Clusterings beschrieben. In Kapitel 3 werden einige grundlegende Probleme und Vorgehensweisen erläutert. Kapitel 4 gibt einen Überblick über verschiedene Ansätze, die bisher in diesem Bereich entwickelt wurden. In Kapitel 5 wird eine neue Methode, die eine Weiterentwicklung bestehender Methoden darstellt, vorgestellt. Kapitel 6 zeigt die Wirksamkeit des neuen Algorithmus anhand von Testergebnissen. In Kapitel 7 werden abschließend Schlussfolgerungen aus den gewonnenen Erkenntnissen gezogen und ein Ausblick auf mögliche zukünftige Entwicklungen gegeben.

Kapitel 2

Clustering-Methoden

Es werden hier die wichtigsten Clustering-Methoden kurz vorgestellt und danach erläutert, wie die Qualität eines Clusterings bewertet werden kann, da dies von zentraler Bedeutung für eine Feature Subset Selection ist.

2.1 k-Means

Der k-Means-Algorithmus ist ein simpler und oft eingesetzter Clustering-Algorithmus. Gegeben die Clusteranzahl k, legt er zunächst k zufällige Clusterzentren fest. Nun wird jede Instanz des Datensatzes dem Clusterzentrum zugeordnet, welches der Instanz am nächsten ist. Daraufhin werden die Clusterzentren neu berechnet als Durchschnitt der ihnen zugewiesenen Instanzen. Durch die Verschiebung der Clusterzentren passiert es, dass manche Instanzen nun einem anderen Clusterzentrum näher sind als zuvor. Die Instanzen werden daher den Clusterzentren neu zugeordnet. Es wechseln sich die Berechnung der Clusterzentren und die Zuordnung der Instanzen zu Clustern ab, solange bis keine Änderung mehr eintritt.

2.2 Expectation-Maximization

Diese Methode basiert auf der Annahme, dass sich ein Clustering beschreiben lässt durch ein Gaussian Mixture Model, eine Summe von Normalverteilungen. Es sollen nun einerseits die Parameter der einzelnen Normalverteilungen gefunden werden und andererseits die Instanzen den Normalverteilungen zugeordnet werden. Das Problem dabei ist, dass man die Parameter kennen muss, um die Instanzen zuzuordnen oder umgekehrt die Zuordnungen kennen muss, um die Parameter zu berechnen. Hier behilft man sich mit dem Expectation-Maximization-Algorithmus (EM-Algorithmus). Dieser funktioniert, ähnlich wie k-Means, durch abwechselnde Ausführung von zwei Schritten. Zunächst werden die Parameter der Normalverteilungen zufällig

gewählt. Nun werden im sogenannten E-Schritt (Expectation) die Wahrscheinlichkeiten für die Zugehörigkeit der Instanzen zu den Normalverteilungen berechnet. Es folgt der M-Schritt (Maximization), bei dem die Parameter so angepasst werden, dass die Likelihood-Funktion maximiert wird. Die Schritte werden abwechselnd solange wiederholt, bis die Änderungen nur noch sehr gering sind.

2.3 Hierarchisches Clustering

Hierarchisches Clustering unterscheidet sich von den anderen Methoden darin, dass keine Partitionierung der Daten erzeugt wird, sondern eine Clusterhierarchie in Form eines (Binär-)Baumes. Jeder Knoten entspricht einem Cluster. Beim Erzeugen der Hierarchie kann man nach dem Top-Down-Prinzip oder nach dem Bottom-Up-Prinzip vorgehen.

Beim Top-Down-Prinzip startet man mit der Wurzel, die den gesamten Datensatz enthält. Schrittweise werden nun die bestehenden Cluster in zwei Teile aufgesplittet und diese als Kindknoten in den Baum eingefügt, solange bis man auf der untersten Ebene ausschließlich Cluster erhält, die genau eine Instanz enthalten. Die Schwierigkeit bei diesem Ansatz liegt darin zu entscheiden, auf welche Weise ein Cluster aufgeteilt wird.

Einfacher zu realisieren ist der Bottom-Up-Ansatz. Hier beginnt man bei den Blättern des Baumes, also bei den Clustern, die jeweils genau eine Instanz enhalten. Schritt für Schritt werden nun jeweils zwei Cluster zu einem neuen Cluster zusammengefügt, und dieser als Elternknoten der beiden Cluster in den Baum eingefügt. Dies wird solange fortgeführt, bis man bei der Wurzel angelangt, die alle Instanzen enthält. Beim Zusammenfügen kann man z.B. so vorgehen, dass man immer die beiden Cluster auswählt, deren Abstand am geringsten ist. Dafür sind allerdings sehr viele Abstandsberechnungen notwendig (im ersten Schritt $\frac{n(n-1)}{2}$ bei n Instanzen, da der Abstand jeder Instanz zu jeder anderen Instanz berechnet werden muss, in den folgenden Schritten muss jeweils der Abstand des neu zusammengefügten Clusters zu allen anderen Clustern berechnet werden).

2.4 Bewertung der Qualität von Clusterings

Hier stellt sich natürlich zunächst die Frage, was denn ein gutes Clustering ausmacht und wie man es von einem schlechten Clustering unterscheidet. Von einem guten Clustering wünscht man sich, dass es eine natürliche Gruppierung widerspiegelt, dass also Instanzen eines Clusters sich durch eine oder mehrere gemeinsame Eigenschaften auszeichnen und Instanzen verschiedener Cluster durch eben diese Eigenschaften voneinander zu unterscheiden sind. Es sollen zwei Instanzen, die im selben Cluster liegen, einander ähnlich sein, und zwei Instanzen, die in verschiedenen Clustern liegen, einander unähnlich

sein, so dass sich die Cluster möglichst klar voneinander abgrenzen lassen. Ähnlichkeit kann auf Grundlage der Attributwerte definiert werden als der Kehrwert des Abstandes, den die Instanzen im Attributraum einnehmen. Eine übliche Methode um Clusterings nach diesem Gesichtspunkt zu bewerten, ist, die Varianz innerhalb der Cluster sowie den Abstand der Clusterzentren zueinander zu berechnen und diese beiden Werte in eine Bewertungsfunktion einfließen zu lassen. Die Varianz innerhalb eines Clusters sollte möglichst gering sein, die Instanzen sollen also möglichst dicht beieinander liegen, während der Abstand zweier Clusterzentren möglichst groß sein sollte.

Um diese Werte überhaupt berechnen zu können, muss zunächst ein Abstandsmaß definiert werden. Bei numerischen Daten verwendet man üblicherweise die Euklidische Distanz, möglich ist auch die Manhattan-Distanz. Bei kategorischen Daten verwendet man die Hamming-Distanz (Summe der Nicht-Übereinstimmungen in den Attributwerten).

Probleme ergeben sich dadurch, dass Clusterings mit unterschiedlicher Clusteranzahl sowie Clusterings in verschiedenen Attribut-Unterräumen miteinander verglichen werden müssen. Denn oftmals haben Bewertungsmaße die Tendenz, Clusterings mit mehr Clustern zu bevorzugen (wobei das Optimum im ungünstigsten Fall beim trivialen Clustering liegt, bei dem jeder Cluster genau eine Instanz enthält), oder sie bevorzugen Clusterings in höherdimensionalen Räumen, was dem Wunsch nach möglichst minimaler Attributmenge entgegen steht. Solche Tendenzen müssen dann durch Korrekturfaktoren ausgeglichen werden. Es ist aber schwierig, sie völlig zu beseitigen ohne dabei umgekehrte Tendenzen zu erzeugen. Jedoch kann im Hinblick auf das Ziel, möglichst kleine Attributmengen zu finden, eine Tendenz zu geringerer Dimension positiv sein, gerade auch im Hinblick auf das Entfernen redundanter Attribute.

Kapitel 3

Grundlegendes zu Feature Subset Selections für Unsupervised Learning

Das Problem einer Feature Subset Selection (FSS) für Unsupervised Learning besteht darin, Attribute zu finden, die für ein Clustering, welches vorab noch nicht bekannt ist, relevant sind. Hinzu kommt, dass die Auswahl der Attribute das Aussehen des Clusterings entscheidend beeinflusst. So kann es z.B. sein, dass verschiedene Unterräume des Attributraums völlig verschiedene Clusterings enthalten. Das Ziel einer FSS ist daher, eine Attributmenge auszuwählen, mit der sich ein möglichst gutes Clustering erstellen lässt. Die gewählte Attributmenge sollte dabei möglichst klein sein, ohne jedoch relevante Informationen verloren zu haben. Das bedeutet, dass irrelevante Attribute, aber auch redundante Attribute entfernt werden sollen. Attribute sind irrelevant, wenn sie keine relevante Information enthalten. Zwei Attribute heißen redundant, wenn sie die gleiche Information enthalten. Man kann eines davon weglassen, ohne dass Information verloren geht.

Wenn man, statt sich auf bestimmte Attribute festzulegen und die anderen zu entfernen, eine Gewichtung nach der Relevanz der Attribute vornimmt, erhält man ein *Feature Weighting* (FW) als Verallgemeinerung der FSS. Diese Verallgemeinerung kann unter Umständen zu besseren Lösungen und aussagekräftigeren Erkenntnissen über die Relevanz der Attribute führen, allerdings hat man es nun mit einem Lösungsraum zu tun, der unendlich viele Lösungen enthält im Gegensatz zu endlich vielen Lösungsmöglichkeiten bei einer FSS. Zudem bietet ein FW nicht oder nur eingeschränkt die Möglichkeit zu einer Dimensionsreduktion der Daten, die bei großen Datensätzen mit ein Grund für die Durchführung einer FSS sein kann.

Man unterscheidet zwei Klassen von Algorithmen, *Wrapper* und *Filter*. Ein Wrapper bewertet die Qualität einer Attributauswahl anhand ei-

nes Clusterings, das er mit Hilfe eines Clustering-Algorithmus erzeugt. Ein Filter dagegen verfügt über ein Bewertungskriterium, das direkt auf die Attributauswahl angewandt wird. Er kommt somit ohne die Verwendung eines Clustering-Algorithmus aus. Beide Methoden haben Vor- und Nachteile. Während Wrapper meist etwas bessere Ergebnisse liefern, die zudem auch optimal an den verwendeten Clustering-Algorithmus angepasst sind, haben Filter normalerweise einen deutlichen Geschwindigkeitsvorteil, da das Erstellen von Clusterings in der Regel sehr zeitaufwändig ist im Vergleich zur Ausführung einer Bewertungsfunktion.

3.1 Wrapper

Wie erwähnt bedient sich ein Wrapper eines Clustering-Algorithmus, um eine Attributauswahl zu bewerten. Er bewertet also nicht die Attributauswahl direkt, sondern ein Clustering, das auf der Basis der Attributauswahl erstellt wurde. Entscheidend für den Erfolg eines Wrappers ist daher, dass er über eine gute Bewertungsfunktion verfügt, mit der sich gute Clusterings von schlechten unterscheiden lassen.

3.1.1 Finden der Clusteranzahl

Clustering-Algorithmen wie z.B. k-Means verlangen als Eingabe die Clusteranzahl k. Die wird jedoch in den wenigsten Fällen vorher bekannt sein. Man wird auch in verschiedenen Attribut-Unterräumen auf Clusterings mit unterschiedlicher Clusteranzahl kommen. Ein Wrapper muss also in der Regel k selbst finden, z.B. indem er den Clustering-Algorithmus mit verschiedenen Werten für k ausführt und dann das Ergebnis des besten Clusterings als Bewertung für eine Attributauswahl zurückgibt.

3.2 Filter

Da Filtern keine Clusterings zur Verfügung stehen, müssen Attributauswahlen nach anderen Gesichtspunkten bewertet werden. So kann man z.B. versuchen, durch Berechnung der Entropie Attribut-Unterräume, die gut separierte Cluster enthalten, von solchen zu unterscheiden, die keine Cluster enthalten. Da das Vorhandensein von Clustern eine Ordnung voraussetzt, sollte ein Unterraum, der Cluster enthält, eine niedrigere Entropie besitzen als einer, der keine Cluster enthält, in dem die Instanzen also mehr oder weniger gleichmäßig verteilt sind.

Ein anderer Ansatz ist die Überlegung, dass eine Abhängigkeit besteht zwischen den relevanten Attributen und einem versteckten Cluster-Attribut. Denn daraus lässt sich folgern, dass die relevanten Attribute auch untereinander abhängig sein müssen, was man nachprüfen kann. Irrelevante Attribute

sollten demnach unabhängig sein (was allerdings nicht immer zutreffen muss, da die irrelevanten Attribute voneinander abhängig sein könnten, auch ohne dass in ihnen Cluster vorhanden sind).

Redundante Attribute können aufgrund ihrer Korrelation erkannt werden. Deshalb eignen sich Filter sehr gut zum Entfernen redundanter Attribute, während Wrapper, gerade solche mit Tendenz zu höherer Dimensionalität, dafür oft weniger gut geeignet sind. Filter können z.B. die paarweise Korrelation zwischen allen Attributen der gewählten Attributmenge berechnen und die Attributmengen danach bewerten. Oder es werden explizit Attribute, die stark mit anderen Attributen korrelieren, aus der Menge herausgenommen.

3.3 Suchstrategien

Das Finden der besten Attributauswahl ist ein NP-schweres Optimierungsproblem. Da die Größe des Suchraums 2^n beträgt bei n Attributen, ist eine erschöpfende Suche in den meisten Fällen unpraktikabel. Es gibt jedoch eine Reihe von allgemeinen Suchstrategien für Optimierungsprobleme, die für das Problem geeignet sind. Bei diesen muss man allerdings hinnehmen, dass sie nicht garantieren, die optimale Lösung zu finden. Bei NP-schweren Problemen muss man sich aber ohnehin mit einer hinreichend guten Lösung zufrieden geben, das Finden der optimalen Lösung ist hier aller Voraussicht nach nicht in polynomieller Laufzeit garantierbar.

Einfache Suchstrategien sind die sequenzielle Vorwärts- und Rückwärtssuche. Bei der sequenziellen Vorwärtssuche startet man mit einer leeren Menge und wählt zunächst ein Attribut aus, das am besten bewertet wird. Dann testet man alle Mengen, die durch Hinzufügen eines weiteren Attributs entstehen. Ergibt sich eine Verbesserung, nimmt man das entsprechende Attribut hinzu und wiederholt den Vorgang solange, bis sich keine Verbesserung mehr durch Hinzufügen eines Attributs erreichen lässt. Entsprechend funktioniert die sequenzielle Rückwärtssuche, mit dem Unterschied, das man hier mit der kompletten Attributmenge startet und solange Attribute entfernt, bis sich keine Verbesserung mehr ergibt. Diese Suchstrategien sind sehr schnell und einfach zu implementieren, garantieren jedoch nicht einmal das Finden eines lokalen Optimums (bei der Vorwärtssuche könnte das Entfernen eines früher hinzugefügten Attributs zu einer Verbesserung führen).

Bei der Verwendung der gewöhnlichen lokalen Suche startet man mit einer beliebigen Attributmenge und verändert diese durch Hinzufügen oder Entfernen einzelner Attribute solange, bis keine Veränderung mehr möglich ist, die zu einer Verbesserung führt. Diese Suchstrategie findet garantiert ein lokales Optimum.

Die obigen lokalen Suchstrategien sind schnell, garantieren jedoch höchstens ein lokales Optimum, welches beliebig schlecht sein kann im Vergleich

zum globalen Optimum. Trotzdem funktionieren sie offenbar recht gut beim Problem der FSS. Wer bessere Ergebnisse erzielen möchte, muss auf fortschrittlichere Algorithmen zurückgreifen, die mit verschiedenen Mitteln ein Hängenbleiben im nächsten lokalen Optimum zu verhindern suchen. Diese Algorithmen geben zwar im Allgemeinen keinerlei Garantien bzgl. der Qualität der gefundenen Lösungen, erzielen jedoch meist bessere Ergebnisse als die einfachen Suchstrategien. Der Preis, den man dafür bezahlt, ist eine längere Laufzeit.

Häufig verwendete Algorithmen dieser Art sind evolutionäre Suchverfahren. Sie beruhen auf dem Prinzip der genetischen Evolution. Lösungen des Optimierungsproblems stellen dabei Individuen dar und werden als Genom kodiert. Üblich ist dabei eine Darstellung des Genoms als reeller Vektor oder als Bitvektor. Beim FSS-Problem bietet sich z.B. eine Kodierung an, bei der für jedes Attribut ein Bit zur Verfügung gestellt wird, welches anzeigt, ob sich das Attribut in der Auswahl befindet oder nicht. Zusätzlich kann man (bei Wrappern) noch die Clusteranzahl k suchen lassen, indem man $k_{max} - 2$ Bits zur Verfügung stellt, deren Summe plus 2 die Clusteranzahl angibt, wobei k_{max} eine vorgegebene maximale Clusteranzahl ist ($k - 2$ Bits, weil man in der Regel minimal zwei Cluster haben möchte).

Eine evolutionäre Suche läuft folgendermaßen ab: Zunächst wird eine Population von zufälligen Individuen erzeugt. Nun wird rundenweise nach einem Fitnesskriterium eine Selektion durchgeführt, so dass die schlechtesten Individuen aussortiert werden und die besten Individuen Nachkommen erzeugen. Ein Nachkomme ist entweder eine Kopie eines bestehenden Individuums oder eine Rekombination zweier Individuen. In beiden Fällen wird außerdem eine Mutation durchgeführt, die den Nachkommen leicht verändert. Mutation bedeutet hier, dass zufällig ein oder mehrere Bits invertiert werden.

Die gebräuchlichsten Strategien zur Rekombination zweier Genome sind *One-Point-Crossover*, *Two-Point-Crossover* und *Uniform Crossover*, wobei die beiden ersteren häufig bevorzugt werden. Beim One-Point-Crossover wird eine Stelle im Genom zufällig gewählt. Die Genome der Eltern werden an dieser Stelle getrennt und die Teile jeweils neu zusammengesetzt, wodurch zwei Nachkommen entstehen. Beim Two-Point-Crossover wählt man zwei zufällige Stellen, trennt die Genome dort und tauscht die Mittelteile aus. Beim Uniform Crossover wird jedes Gen des Nachkommen per Zufall entweder vom ersten oder vom zweiten Elternteil ausgewählt.

Es existiert eine Vielzahl von evolutionären Algorithmen, die sich hauptsächlich durch ihre Selektions- und Rekombinationsmechanismen unterscheiden.

3.4 Normalisierung

Auch wenn dieses Thema nicht direkt etwas mit FSS zu tun hat, soll es hier kurz angesprochen werden.

Bei numerischen Daten sollte vor der Ausführung einer FSS eine Normalisierung aller Attribute durchgeführt werden. Andernfalls kann es passieren, dass Attribute aufgrund ihres Wertebereichs bevorzugt oder benachteiligt werden, insbesondere bei stark unterschiedlichen Wertebereichen (Beispiel: ein Attribut gibt das Alter von Personen in Jahren an, ein anderes ihr jährliches Einkommen in Euro). Wenn sich beispielsweise die Werte eines Attributs über ein sehr großes Intervall erstrecken, ergeben sich viel größere Abstände zwischen den Instanzen als wenn sich die Werte über ein sehr kleines Intervall erstrecken. Hat man nun sowohl Attribute mit sehr großen Wertebereichen als auch Attribute mit sehr kleinen Wertebereichen, so werden die Abstände der Instanzen fast ausschließlich durch die Attribute mit den großen Wertebereichen bestimmt. Die anderen Attribute fallen kaum ins Gewicht und könnten daher fälschlicherweise als irrelevant angesehen werden.

Eine Möglichkeit der Normalisierung ist, die Wertebereiche aller Attribute auf das Einheitsintervall [0,1] zu transformieren. Eine andere Möglichkeit ist, die Attributwerte so zu verschieben, dass der Mittelwert bei 0 liegt und danach durch eine Division durch die Standardabweichung eine Standardabweichung von 1 zu erreichen.

Kapitel 4

Überblick bisheriger Ansätze

Erst in den letzten Jahren wurde dem Thema Feature Subset Selection im Bereich des Unsupervised Learning eine größere Beachtung geschenkt, so dass die Anzahl der Forschungsarbeiten sich noch in einem überschaubaren Rahmen hält. Die folgende Auflistung erhebt keinen Anspruch auf Vollständigkeit, bietet aber eine gute Übersicht über die bisher entwickelten Ansätze.

4.1 Wrapper-Methoden

4.1.1 Ansätze mit Hauptaugenmerk auf Bewertungsfunktionen für Clusterings

Dy & Brodley, Feature Selection for Unsupervised Learning [11]

Dy und Brodley sehen als Ziel einer Feature Subset Selection das Finden der kleinsten Attributmenge, die am besten eine „interessante und natürliche“ Gruppierung (Clustering) innerhalb der Daten aufdeckt. Um ein interessantes Clustering zu finden bedarf es eines Bewertungskriteriums, mit dem die Qualität eines Clusterings bewertet wird. Dy und Brodley stellen zwei solche Kriterien vor, Scatter Separability und Maximum Likelihood.

Scatter Separability: Bei diesem Bewertungskriterium wird davon ausgegangen, dass solche Clusterings interessant sind, bei denen erstens die Cluster gut voneinander trennbar sind und zweitens die Streuung der Instanzen innerhalb eines Clusters gering ist. Dazu wird zunächst die intra-Cluster Scatter-Matrix S_w und die inter-Cluster Scatter-Matrix S_b berechnet:

$$S_w = \sum_{j=1}^{k} \pi_j \Sigma_j$$

$$S_b = \sum_{j=1}^{k} \pi_j (\mu_j - M_0)(\mu_j - M_0)^T$$

Dabei ist Σ_j die Kovarianzmatrix des Clusters ω_j, π_j die Wahrscheinlichkeit, dass eine Instanz zu Cluster ω_j gehört, k die Anzahl der Cluster, μ_j der Mittelwert von Cluster ω_j und M_0 der Mittelpunkt aller Instanzen.

S_w gibt an, wie stark die Instanzen innerhalb eines Cluster gestreut sind, S_b, wie stark die Clusterzentren gestreut sind. Aus den beiden Matrizen wird schließlich als Bewertungskriterium $trace(S_w^{-1} S_b)$ berechnet.

Maximum Likelihood: Bei der Verwendung des Expectation-Maximization (EM) Algorithmus wird eine *Gaussian Mixture* als Modell für die Daten verwendet, da Cluster als normalverteilt angenommen werden. Das Maximum-Likelihood-Kriterium gibt an, wie gut dieses Modell auf die Daten passt. In diesem Fall sind die „interessanten Gruppierungen“ normalverteilte Gruppierungen.

Um die optimale Attributauswahl zu finden wird in diesem Ansatz eine sequenzielle Vorwärtssuche verwendet, d.h. man startet mit einer leeren Attributmenge und nimmt solange einzelne Attribute hinzu, die gemäß dem Bewertungskriterium am geeignetsten erscheinen, bis keine Verbesserung mehr erzielt werden kann.

Der Algorithmus

1. F = Menge aller Attribute des Datensatzes.
   Die Menge der ausgewählten Attribute S wird initialisiert als $S = \{\}$.
   $bestScore = 0$
2. ***for all*** $f_i \in F$
   - Suchen der Clusteranzahl k
   - $clustering = EM(k, S \cup \{f_i\})$
   - $score_i = criterion(clustering, S \cup \{f_i\})$
3. Wähle das Attribut f_i, für das $score_i$ maximal ist.
   if $score_i > bestScore$
   - $S = S \cup \{f_i\}$
   - $F = F \backslash \{f_i\}$

- ***goto*** 2.

else

- ***goto*** 4.

4. ***return*** S.

criterion ist hierbei entweder Scatter Separability oder Maximum Likelihood, *EM* ist der Expectation-Maximization Clustering-Algorithmus.

4.1.2 Genetische und evolutionäre Algorithmen

Morita, Sabourin, Bortolozzi und Suen, Unsupervised Feature Selection Using Multi-Objective Genetic Algorithms for Handwritten Word Recognition [24]

Dieser Ansatz verwendet einen genetischen Algorithmus um eine Menge pareto-optimaler Attributmengen bzgl. mehrerer Bewertungskriterien zu finden. Hierbei muss, um zu verhindern, dass der Algorithmus auf eine einzige Lösung zusteuert und um die Vielfalt der Population zu erhalten, für die Selektion ein besonderes Prinzip angewandt werden. Morita et al. verwenden den von Goldberg in [15] vorgeschlagenen und von Srinivas und Deb in [28] umgesetzten *Nondominated Sorting Genetic Algorithm* (NSGA), der vom Prinzip des sog. *Fitness Sharing* Gebrauch macht. Dabei wird zur Selektion eine Rangfolge der Individuen aufgrund ihrer *Nichtdominiertheit* erstellt. Ein Individuum I_1 *dominiert* ein anderes Individuum I_2 genau dann, wenn I_1 bei keiner der Zielfunktionen (Bewertungskriterien) einen schlechteren Wert erreicht als I_2 und bei mindestens einer Zielfunktion einen besseren. Individuen, die von keinem anderen Individuum dominiert werden, heißen *nichtdominiert* und bekommen einen hohen Fitnesswert zugewiesen.

In diesem Ansatz werden zwei Bewertungsfunktionen verwendet, zum einen der Davies-Bouldin (DB) Index [8], und zum anderen die Anzahl der gewählten Attribute, weil der DB-Index die Tendenz hat, Lösungen mit vielen Attributen zu bevorzugen.

DB-Index: In die Funktion fließen zum einen die Streuung der Instanzen innerhalb eines Clusters und zum anderen die Abstände der Cluster voneinander ein. Die Streuung innerhalb des Clusters C_i wird folgendermaßen definiert:

$$S_i = \frac{1}{|C_i|} \sum_{X \in C_i} \|X - Z_i\|$$

Die Distanz zwischen zwei Clustern C_i und C_j berechnet sich als als Distanz der Clusterzentern Z_i und Z_j:

$$d_{ij} = \|Z_i - Z_j\|$$

Der DB-Index ist definiert als

$$I_{DB} = \frac{1}{D}\frac{1}{K}\sum_{i=1}^{K} R_i$$

mit

$$R_i = \max_{j,j\neq i}\{\frac{S_i + S_j}{d_{ij}}\}$$

wobei K die Anzahl der Cluster und D die Anzahl der Attribute ist.

Zum Erzeugen der Clusterings wird der k-Means-Algorithmus verwendet. Das Genom eines Individuums ist kodiert als Bit-String, bestehend aus $d + k - 2$ Bits, wobei d die Anzahl der Attribute des Datensatzes und k eine vorgegebene maximal mögliche Anzahl von Clustern ist.

Der Algorithmus

1. Gegeben sind:
   n = Größe der Population
   generations = Anzahl der Generationen
   $p_{crossover}$ = Wahrscheinlichkeit eines Crossover
   $p_{mutation}$ = Wahrscheinlichkeit einer Mutation
   niche distance

2. Erzeuge eine zufällige Population P von n Individuen.

3. ***for*** i=1 ***to*** *generations*

   - Zuweisung der Fitnesswerte nach dem Prinzip des Nondominated Sorting. Berechnung der ersten Zielfunktion: Erzeuge ein Clustering mit k-Means und bewerte es mit dem DB-Index. Berechnung der zweiten Zielfunktion: Anzahl der Attribute.
   - Reproduktion auf Basis der Fitnesswerte

4. ***return*** $\{I \in P | I$ ist nichtdominiert$\}$.

Kim, Street und Menczer, Feature Selection in Unsupervised Learning via Evolutionary Search [18]

Im Unterschied zum Ansatz von Morita et al. [24] verwenden Kim et al. einen evolutionären Algorithmus ohne Crossover. Veränderungen ergeben sich also ausschließlich durch Mutationen. Auch mit diesem Ansatz wird versucht, eine Menge pareto-optimaler Lösungen bzgl. mehrerer Bewertungsfunktionen zu finden.

In jedem Durchgang erhalten die Individuen Energiewerte aus verschiedenen Energiequellen. Pro Bewertungsfunktion gibt es eine bestimmte Anzahl von Energiequellen, wobei jede Energiequelle für einen bestimmten Wertebereich der Bewertungsfunktion steht. Es steht für jeden Durchgang in jeder Energiequelle eine bestimmte Energiemenge zur Verfügung, und zwar in Energiequellen, die einem hohen Wertebereich zugeordnet sind mehr Energie als in solchen, die einem niedrigen Wertebereich zugeordnet sind. Die Energiequellen werden nach jedem Durchgang wieder aufgefüllt.

Jedes Individuum erhält nun einen Energiewert von den Quellen, zu denen es gemäß seiner Bewertungen, die es durch die Bewertungsfunktionen erhält, Zugang hat. Außerdem wird ihm eine konstante Energiemenge abgezogen. Individuen, die einen Energiewert kleiner null haben, werden aus der Population herausgenommen. In jedem Durchgang wird von jedem Individuum, wenn es einen bestimmten Energiewert überschreitet, ein mutierter Clon erstellt und der Population hinzugefügt. Je mehr Individuen sich im Bereich einer Energiequelle aufhalten, desto weniger Energie steht für ein einzelnes Individuum zur Verfügung, da die zur Verfügung stehende Energie unter den Individuen im Bereich der Energiequelle aufgeteilt wird. Damit soll sichergestellt werden, dass die Individuen sich möglichst auf alle Energiequellen verteilen und so eine Vielfalt erhalten bleibt.

Kim et al. verwenden vier Bewertungsfunktionen:

F_{within}: Diese Funktion bestimmt die Dichte der Cluster aufgrund der Abstände der Instanzen innerhalb eines Clusters:

$$F_{within} = 1 - \frac{1}{Z_{within}} \frac{1}{d} \sum_{k=1}^{K} \sum_{i=1}^{n} \alpha_{ik} \sum_{j \in J} (x_{ij} - \gamma_{kj})^2$$

J ist die gewählte Attributmenge, K ist die Menge der Cluster, γ_{ij} ist die j-te Komponente des Zentrums des k-ten Clusters. α_{ik} ist 1, wenn die i-te Instanz im k-ten Cluster liegt und 0 sonst, d ist die Dimension der gewählten Attributmenge und Z_{within} ist eine Konstante, die für jeden Datensatz empirisch bestimmt wird um den Wertebereich der Funktion auf das Einheitsintervall zu normieren.

$F_{between}$: Diese Funktion bevorzugt gut separierte Cluster, als Grundlage dienen hier die Abstände von Instanzen, die nicht im gleichen Cluster liegen:

$$F_{between} = \frac{1}{Z_{between}} \frac{1}{d} \frac{1}{k-1} \sum_{k=1}^{K} \sum_{i=1}^{n} (1 - \alpha_{ik}) \sum_{j \in J} (x_{ij} - \gamma_{kj})^2$$

$Z_{between}$ ist wieder eine Normierungskonstante, die empirisch ermittelt wird.

$F_{clusters}$: Mit dieser Funktion soll die Tendenz der beiden ersten Funktionen zu Lösungen mit vielen Clustern kompensiert werden:

$$F_{clusters} = 1 - \frac{K - K_{min}}{K_{max} - K_{min}}$$

Hierbei ist K_{max} (K_{min}) die maximal (minimal) mögliche Anzahl von Clustern, die im Genom der Individuen kodiert werden kann.

$F_{complexity}$: Mit dieser Funktion sollen Lösungen bevorzugt werden, die wenige Attribute verwenden:

$$F_{complexity} = 1 - \frac{d - 1}{D - 1}$$

D ist die Gesamtzahl der Attribute, d die Anzahl der gewählten Attribute.

Das Genom wird in diesem Algorithmus genauso wie im vorigen als Bitstring mit $D + k - 2$ Bits kodiert.

Der Algorithmus

1. Initialisiere die Population P mit zufälligen Individuen und weise allen Individuen den Energiewert $\theta/2$ zu.
2. ***for*** i=1 ***to*** T
   (a) ***for each*** Energiequelle c
   for each v $(0 \ldots 1)$ *//Schrittweite wird individuell festgelegt*
   $E^c_{envt}(v) = 2vE^c_{tot}$
   (b) ***for each*** $a \in P$
   $a' = mutate(clone(a))$
   for each Energiequelle c
   $v = Fitness(a', c)$
   $\Delta E = min(v, E^c_{envt}(v))$
   $E^c_{envt}(v) = E^c_{envt}(v) - \Delta E$
   $E_a = E_a + \Delta E$
   (c) $E_a = E_a - E_{cost}$
   (d) ***if*** $E_a > \theta$
   $P = P \cup \{a'\}$
   $E_{a'} = E_a/2$
   $E_a = E_a - E_{a'}$
   else if $E_a < 0$
   $P = P \backslash \{a\}$
3. ***return*** P.

4.1.3 Besondere Algorithmen

Devaney & Ram, Efficient Feature Selection in Conceptual Clustering [10]

Dieser Ansatz baut auf dem COBWEB System [12] für hierarchisches Clustering auf. COBWEB ist ein iteratives Verfahren, das nacheinander alle Instanzen eines Datensatzes in eine Clusterhierarchie einfügt und die Hierarchie dabei gegebenenfalls durch Aufsplitten oder Zusammenfügen von Knoten (Clustern) anpasst. Dazu werden Cluster mit einem Kriterium namens Category Utility [14] bewertet.

Category Utility: Dieses Maß ist definiert als

$$CU(C_k) = P(C_k) \sum_i \sum_j \left[P(A_i = V_{ij}|C_k)^2 - P(A_i = V_{ij})^2 \right]$$

wobei A_i die Attribute und V_{ij} die Werte der Attribute sind. Das Kriterium bewertet, wie gut die Werte der Attribute im Cluster C_k vorhergesagt werden können im Vergleich zur Vorhersagegenauigkeit der Attributwerte im gesamten Datensatz. Um eine Partitionierung, also eine Ebene der Hierarchie zu bewerten, wird

$$PU(\{C_1, C_2, \ldots, C_n\}) = \frac{\sum_{k=1}^{n} CU(C_k)}{n}$$

berechnet.

Der Algorithmus von Devaney und Ram mit dem Namen AICC (Attribute-Incremental Concept Creator) führt eine Feature Subset Selection basierend auf COBWEB aus. Dabei wird eine sequenzielle Vorwärts- oder Rückwärtssuche verwendet.

Die Idee bei diesem Ansatz ist, nicht für jede gewählte Attributmenge einmal den COBWEB-Algorithmus auszuführen, sondern eine einmal erstellte Cluster-Hierarchie für die nächste Attributmenge anzupassen. Bei der Vorwärtssuche wird dabei, wenn ein neues Attribut hinzugenommen wird, die Hierarchie genau wie beim Einfügen von Instanzen mittels Aufsplitten und Zusammenfügen von Clustern angepasst (analog bei der Rückwärtssuche, wenn ein Attribut entfernt wird). Als Clustering bewertet wird immer die erste Ebene unterhalb der Wurzel.

Der Algorithmus

1. F = Menge aller Attribute des Datensatzes.
   Die Menge der ausgewählten Attribute S wird initialisiert als $S = \{\}$.
   $bestScore = 0$

2. ***for all*** $f_i \in F$
   - Cluster-Hierarchie = AICC(Cluster-Hierarchie)
   - clustering = erste Ebene der Cluster-Hierarchie unter der Wurzel
   - $score_i = predictive - accuracy(clustering)$
3. Wähle das Attribut f_i, für das $score_i$ maximal ist.
   if $score_i > bestScore$
   - $S = S \cup \{f_i\}$
   - $F = F \backslash \{f_i\}$
   - ***goto*** 2.

   else
   - ***goto*** 4.
4. ***return*** S.

Der AICC-Algorithmus:

AICC(Cluster-Hierarchie)

1. Passe die Instanz-Beschreibungen an.
2. ***for each*** Partitionierung P der Hierarchie
   (a) ***do*** /*splits*/
      i. $CU_0 = category\_utility(P)$
      ii. ***for each*** $n_i \in P$
         $P' = split(P, n_i)$
         $CU_i = category\_utility(P')$
      iii. $CU_{max} = max(CU_i)$
      iv. ***if*** $CU_{max} > CU_0$
         $P = split(P, n_i)$

   while $CU_{max} > CU_0$

   (b) ***do*** /*merges*/
      i. $CU_0 = category\_utility(P)$
      ii. ***for each*** $n_i, n_j \in P, i \neq j$
         $P' = merge(P, n_i, n_j)$
         $CU_{ij} = category\_utility(P')$
      iii. $CU_{max} = max(CU_{ij})$
      iv. ***if*** $CU_{max} > CU_0$
         $P = merge(P, n_i, n_j)$
3. ***return*** die Hierarchie.

Agrawal, Gehrke, Gunopulos und Raghavan, Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications [1]

Im Unterschied zu den bisherigen Ansätzen werden bei dem von Agrawal et al. vorgestellten CLIQUE-Algorithmus (**CL**ustering **I**n **QUE**st) keine Clusterings im Sinne von Partitionierungen gesucht, sondern es wird versucht, Regionen mit hoher Dichte auszumachen und von solchen Clustern eine möglichst einfache Beschreibung zu erzeugen.

Zunächst wird der Wertebereich jedes Attributs in gleichlange Intervalle unterteilt. Danach wird versucht, in beliebigen Unterräumen des durch die Attributmenge aufgespannten Raumes Einheiten („Hyperwürfel“) mit hoher Dichte zu identifizieren.

Eine erschöpfende Suche wäre hier schon bei wenigen Dimensionen impraktikabel. Der Algorithmus macht sich deshalb die Erkenntnis zunutze, dass eine Einheit mit hoher Dichte in einem k-dimensionalen Raum auch in allen $(k-1)$-dimensionalen Unterräumen eine hohe Dichte haben muss. So lassen sich (nach dem Prinzip des Apriori-Algorithmus [2]) schrittweise aus Einheiten niedriger Dimension solche höherer Dimension herleiten. Es werden also zunächst alle eindimensionalen Einheiten mit hoher Dichte ermittelt. Diese setzt man zu zweidimensionalen Einheiten zusammen, bildet daraus wiederum, wenn möglich, dreidimensionale Einheiten usw., bis sich für eine Dimension keine Einheiten hoher Dichte mehr finden lassen.

Hat man alle Einheiten mit hoher Dichte identifiziert, versucht man im nächsten Schritt, daraus Cluster zu bilden, indem man Gruppen zusammenhängender Einheiten ermittelt. Jede solche Gruppe ist ein Cluster. Hat man dies erledigt, so versucht man in einem letzten Schritt, aus diesen Beschreibungen der Cluster als Gruppen von Einheiten mit hoher Dichte kürzere Beschreibungen zu konstruieren. Man sucht (Hyper-)Quader, die möglichst viele Einheiten der Gruppe umschließen und dabei keine Einheiten auserhalb der Gruppe enthalten. Die minimale Beschreibung erhält man, wenn man die minimale Anzahl von Quadern findet, die die Gruppe der Einheiten insgesamt vollständig enthalten. Dieses Optimierungsproblem ist jedoch NP-schwer. Mit einem einfachen Algorithmus wird deshalb eine möglichst gute Lösung gesucht. Man beginnt mit einem Quader, der aus genau einer Einheit der Gruppe besteht und dehnt den Quader durch Hinzunahme benachbarter Einheiten nacheinander in allen Dimensionen soweit wie möglich aus. Dann sucht man eine noch nicht abgedeckte Einheit der Gruppe und wiederholt die Prozedur. Dies führt man solange durch, bis alle Einheiten abgedeckt wurden. Anschließend entfernt man noch eventuelle Quader, die vollständig von anderen Quadern umschlossen werden, beginnend mit dem kleinsten.

Der Algorithmus

1. Identifizierung von Unterräumen, die Cluster enthalten:
   (a) Teile den Raum in gleichgroße Einheiten auf.
   (b) D_1 = Menge der 1-dimensionalen Einheiten mit hoher Dichte.
   (c) ***for*** i=2 ***to*** d
   D_i = Menge der i-dimensionalen Einheiten, deren (i-1)-dimensionale Komponenten in D_{i-1} enthalten sind.
   if $D_i = \{\}$
   goto (d).
   (d) $D = \{D_1, D_2, \ldots, D_d\}$
   return D.
2. Identifizierung von Clustern:
   (a) ***for all*** $D_i \in D$
   Finde zusammenhängende Einheiten (=Cluster) mittels Depth-First-Suche, solange bis alle Einheiten zu einem Cluster gehören.
   (b) ***return*** die Menge C aller Cluster (Cluster = Menge zusammenhängender Einheiten).
3. Generierung einer minimalen Beschreibung für die Cluster:
   (a) ***for all*** $c \in C$
   i. ***do***
   Finde eine nicht abgedeckte Einheit u.
   Bilde, von u ausgehend, einen Quader maximaler Größe, der nur Einheiten aus c enthält.
   while es gibt noch nicht abgedeckte Einheiten in c.
   ii. ***do***
   Finde die kleinste Region, deren sämtliche Einheiten von anderen Regionen abgedeckt werden.
   Falls eine solche Region gefunden wurde, entferne sie.
   while Eine Veränderung wurde vorgenommen.
4. ***return*** die Menge aller Clusterbeschreibungen.

Modha & Spangler, Feature Weighting in k-Means Clustering [23]

In diesem Ansatz wird keine Feature Subset Selection im eigentlichen Sinn vorgestellt, sondern eine Methode, mit der Gruppen von Attributen (*Attributräume*) Gewichte zugewiesen bekommen, entsprechend ihrer Relevanz. Zunächst werden die Attribute eines Datensatzes in Attributräume eingeteilt.

Dies kann z.B. aufgrund des Typs (nominal, numerisch) oder der Bedeutung der Attribute erfolgen. Die Einteilung muss so vorgenommen werden, dass sich für jeden Attributraum ein Distanzmaß angeben lässt. Die Distanzmaße werden schließlich zu einem gewichteten Distanzmaß verrechnet. Für einen Datensatz mit m Attributräumen und Instanzen $x = (F_1, F_2, \ldots, F_m)$, $\widetilde{x} = (\widetilde{F}_1, \widetilde{F}_2, \ldots, \widetilde{F}_m)$ gilt:

$$D^{\alpha}(x, \widetilde{x}) = \sum_{l=1}^{m} \alpha_l D_l(F_l, \widetilde{F}_l)$$

Dabei ist D_l das Distanzmaß des l-ten Attributraumes, $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_m)$ ist eine Gewichtung derart, dass $\sum_{l=1}^{m} \alpha_l = 1$ und $\alpha_l \geq 0$ für $l = 1, \ldots, m$.

Es soll nun eine Gewichtung gefunden werden, die in irgendeiner Hinsicht die beste ist. Bei der Entscheidung, wie die Qualität einer Gewichtung zu berechnen ist, gilt es zu berücksichtigen, dass unterschiedliche Gewichtungen zu unterschiedlichen Distanzmaßen führen. Zunächst wird ein Clustering definiert als Partitionierung der Daten mit den entsprechenden Clusterzentren:

$$\Pi(\alpha) = \{\pi_u^{\dagger}(\alpha), c_u^{\dagger}(\alpha)\}_{u=1}^{k}$$

Nun definiert man die durchschnittliche intra-Cluster Distanz und die durchschnittliche inter-Cluster Distanz bzgl. dem l-ten Attributraum als

$$\Gamma_l(\alpha) \equiv \Gamma_l(\Pi(\alpha)) = \sum_{u=1}^{k} \sum_{x \in \pi_u^{\dagger}(\alpha)} D_l(F_l, c_{(u,l)}^{\dagger}(\alpha))$$

$$\Lambda_l(\alpha) \equiv \Lambda_l(\Pi(\alpha)) = \sum_{i=1}^{n} \left[D_l(F_{(i,l)}, \overline{c}_l) - \Gamma_l(\alpha) \right]$$

Um gut separierte Cluster zu erhalten, muss

$$\Phi_l(\alpha) \equiv \Phi_l(\Pi(\alpha)) = \left(\frac{\Gamma_l(\alpha)}{\Lambda_l(\alpha)} \right)^{n_l/n}$$

minimiert werden, wobei n_l die Anzahl der Instanzen ist, die einen von null verschiedenen l-ten Attribut-Vektor besitzen.

Da eine gute Unterscheidung bzgl. aller Attributräume erwünscht ist, wählt man die optimale Gewichtung $\alpha^{\dagger}$ als

$$\alpha^{\dagger} = \arg\min_{\alpha \in \Delta} [\Phi(\alpha)]$$

mit

$$\Phi(\alpha) = \Phi_1(\alpha) \times \Phi_2(\alpha) \times \cdots \times \Phi_m(\alpha).$$

Um die optimale Gewichtung zu finden, wird in diesem Ansatz eine erschöpfende Suche mit einem feinen Raster verwendet. Diese wird für verschiedene Werte für die Clusteranzahl k wiederholt. Bewertet wird jeweils ein mit k-Means erstelltes Clustering.

Der Algorithmus

1. $K = \{2, 4, 6, 8, 16\}$
   Menge der möglichen Gewichtungen $A = \{\alpha \in \Delta | (\alpha_i \cdot 10) \in \mathbb{Z}, i = 1, \ldots, m\}$

2. ***for all*** $k \in K$
   for all $\alpha \in A$
   Erstelle zufälliges Clustering.
   do
   Ordne jede Instanz dem nächsten Clusterzentrum zu bzgl. D^α.
   Berechne die Clusterzentren neu.
   while Änderungen.
   Bewerte das Clustering nach $\Phi(\alpha)$.

3. ***return*** Konfiguration (k, α), für die $\Phi(\alpha)$ maximal ist.

4.2 Filter-Methoden

4.2.1 Methoden zum Entfernen irrelevanter Features

Søndberg-Madsen, Thomsen und Peña, Unsupervised Feature Subset Selection [27]

Die Idee zu diesem Ansatz basiert auf der Grundannahme, dass jeder Cluster einen physikalischen Prozess repräsentiert, der durch eine unbekannte Wahrscheinlichkeitsverteilung definiert wird. Es wird die Existenz einer Zufallsvariablen $\mathbf{X} = (X_1, \ldots, X_{n+1})$ vorausgesetzt, die aufgeteilt werden kann in $\mathbf{X} = (C, \mathbf{Y})$, wobei C eine eindimensionale versteckte Zufallsvariable ist, die die Clusterzugehörigkeit widerspiegelt, und $\mathbf{Y} = (Y_1, \ldots, Y_n)$ den Beobachtungen, also den Attributen des Datensatzes, entspricht.

Da nun alle relevanten Attribute abhängig von C sein müssen, müssen sie auch alle paarweise voneinander abhängig sein. Diejenigen Attribute, die unabhängig von den anderen Attributen sind, können daher als irrelevant betrachtet werden (der Ansatz setzt voraus, dass es wenigstens zwei relevante Attribute gibt), wobei allerdings nicht auszuschließen ist, dass es auch irrelevante Attribute gibt, die voneinander abhängig sind und auf diese Weise nicht erkannt werden können.

Søndberg-Madsen et al. stellen zwei Abhängigkeitsmaße vor, die geeignet sind, die Abhängigkeit zweier Attribute Y_i und Y_j zu messen. Das erste ist das *Mutual Information* Maß, das definiert ist als

$$DS(Y_i, Y_j) = MI(Y_i, Y_j) = H(Y_i) - H(Y_i|Y_j)$$

wobei $H(Y_i)$ die Entropie von Y_i und $H(Y_i|Y_j)$ die bedingte Entropie von Y_i, gegeben Y_j, ist.

Das zweite Abhängigkeitsmaß basiert auf der *Mutual Prediction*. Es ist definiert als

$$DS(Y_i, Y_j) = MP(Y_i, Y_j) = 1 - \frac{1}{2}\left(\frac{PA(Y_i)}{PA(Y_i|Y_j)} + \frac{PA(Y_j)}{PA(Y_j|Y_i)}\right)$$

wobei $PA(Y_i)$ und $PA(Y_i|Y_j)$ definiert sind als

$$PA(Y_i) = \max_{y_i} p(y_i)$$

$$PA(Y_i|Y_j) = \sum_{y_j} p(y_j) \max_{y_i} p(y_i|y_j).$$

$PA(Y_i)$ ist die Wahrscheinlichkeit, mit der der Zustand von Y_i korrekt vorhergesagt werden kann (durch Vorhersage des wahrscheinlichsten Zustands). $PA(Y_i|Y_j)$ ist entsprechend die Wahrscheinlichkeit, mit der der Zustand von Y_i vorhergesagt werden kann, wenn der Zustand von Y_j gegeben ist.

Die Wahrscheinlichkeitsverteilungen, die für die Berechnungen der beiden Maße benötigt werden, werden durch das Maximum-Likelihood-Kriterium geschätzt.

Als Maß für die Relevanz eines Attributs Y_i wird nun die durchschnittliche Abhängigkeit des Attributs zu allen anderen Attributen berechnet:

$$RS(Y_i) = \frac{1}{n-1} \sum_{j=i, j\neq i}^{n} DS(Y_i, Y_j)$$

Um nun letztendlich festzustellen, ob ein Attribut relevant ist oder nicht, wird ein Hypothesentest durchgeführt mit der Nullhypothese, dass Y_i irrelevant ist. $RS(Y_i)$ wird dabei als statistischer Wert betrachtet. Die Wahrscheinlichkeitsverteilung der Statistik wird empirisch geschätzt mit Hilfe der Werte einer ausreichenden Anzahl von Attributen, deren Irrelevanz bekannt ist (diese Attribute können konstruiert sein).

Der Algorithmus

1. F = Menge aller Attribute des Datensatzes.
   Die Menge der ausgewählten Attribute S wird initialisiert als $S = \{\}$.
2. ***for all*** $Y_i \in F$
   - $RS_i = \frac{1}{n-1} \sum_{j=1, j\neq i}^{n} DS(Y_i, Y_j)$

     (dabei ist $DS(Y_i, Y_j) = H(Y_i - H(Y_i)|Y_j)$)
3. Sortiere die Y_i nach der Werten von RS_i in absteigender Reihenfolge.

4. ***for*** i=1 ***to*** $|F|$

   - Führe einen Hypothesentest durch:
     $H_0 : Y_i$ ist irrelevant.
     $H_1 : Y_i$ ist relevant.
   - ***if*** Y_i ist relevant
     $S = S \cup \{Y_i\}$
     else goto 5.

5. ***return*** S.

Peña, Lozano, Larrañaga und Inza, Dimensionality Reduction in Unsupervised Learning of Conditional Gaussian Networks [25]

Wie beim vorigen Ansatz wird auch hier davon ausgegangen, dass die relevanten Attribute alle paarweise voneinander abhängig sind und unabhängige Attribute somit als irrelevant entfernt werden können.

Zur Berechnung der Relevanz wird hier ein simples Relevanzmaß verwendet:

$$relevance(Y_i) = \sum_{j=1, j\neq i}^{n} \frac{-N \log(1 - r_{ij|rest}^2)}{n-1}$$

Hierbei ist n die Anzahl der Attribute, N die Anzahl der Instanzen und $r_{ij|rest}$ die partielle Korrelation von Y_i und Y_j, die sich auch als Maximum-Likelihood-Schätzung der Elemente der inversen Varianzmatrix ausdrücken lässt als

$$r_{ij|rest} = -\hat{w}_{ij}(\hat{w}_{ii}\hat{w}_{jj})^{-\frac{1}{2}}.$$

Nach der Berechnung der Relevanz aller Attribute können diese entsprechend ihrer Werte geordnet werden. Um nun bestimmen zu können, welche dieser Attribute als irrelevant gelten, benötigt man einen Grenzwert, es sei denn, man kennt die Anzahl der relevanten Attribute, was aber im Allgemeinen nicht der Fall ist. Peña et al. bedienen sich hierbei einer Heuristik. Der Relevanz-Grenzwert wird berechnet als Lösung der folgenden Gleichung:

$$0.95 = G_\chi(x) - \frac{1}{2}(2n+1)x\frac{1}{\sqrt{2\pi}}x^{-\frac{1}{2}}e^{-\frac{1}{2}x}N^{-1}$$

Hierbei ist $G_\chi(x)$ die Verteilungsfunktion einer χ_1^2 Zufallsvariablen. Peña et al. verwenden die Newton-Raphson-Methode zur Lösung der Gleichung.

Alle Attribute, deren Relevanzwerte den Grenzwert überschreiten, werden als relevant betrachtet, der Rest als irrelevant.

Der Algorithmus

1. F = Menge aller Attribute des Datensatzes.
2. ***for all*** $Y_i \in F$
   $relevance(Y_i) = \sum_{j=1,j\neq i}^{n} \frac{-Nlog(1-r^2_{ij|rest})}{n-1}$
3. Berechnung des Thresholds: Finde die Lösung der Gleichung
   $$0.95 = G_\chi(x) - \frac{1}{2}(2n+1)x\frac{1}{\sqrt{2\pi}}x^{-\frac{1}{2}}e^{-\frac{1}{2}x}N^{-1}$$
   (z.B. durch Anwendung der Newton-Raphson-Methode)
4. Attributauswahl $S = \{Y_i | relevance(Y_i) > Threshold\}$
5. ***return*** S.

Talavera, Feature Selection as a Preprocessing Step for Hierarchical Clustering [29]

Auch in diesem Ansatz wird versucht, die Relevanz von Attributen durch deren Abhängigkeit zu anderen Attributen zu ermitteln. Um zu überprüfen, ob zwei Attribute voneinander abhängig sind, wird hier festgestellt, ob die Werte des einen Attributs besser vorausgesagt werden können, wenn man die Werte des zweiten Attributs kennt.

Die erwartete Anzahl von Werten, die für ein Attribut A_M korrekt vorhergesagt werden kann, wenn die Werte eines Attributs A_i bekannt sind, ist

$$E(A_M, A_i) = \sum_{j_M} P(A_M = V_{Mj_M} | A_i = V_{ij_i})^2.$$

Die erwartete Anzahl von Werten, die vorhergesagt werden kann, wenn keine Werte eines anderen Attributs bekannt sind, ist

$$E(A_M) = \sum_{j_M} P(A_M = V_{Mj_M})^2.$$

Damit lässt sich nun die Abhängigkeit des Attributs A_M vom Attribut A_i berechnen als

$$dependency(A_M, A_i) = \sum_{j_i} P(A_i = V_{j_i}) E(A_M, A_i) - E(A_M).$$

Als Abhängigkeitsmaß wird schließlich die durchschnittliche Abhängigkeit zu allen anderen Attributen berechnet:

$$dependency(A_M) = \frac{\sum_{i=1,i\neq M}^{n} dependency(A_M, A_i)}{n-1}$$

wobei n die Anzahl der Attribute ist. Dieser Wert wird nun für alle Attribute berechnet. Mit Hilfe eine Grenzwertes τ, der vorher gewählt wird, wird entschieden, welche Attribute als relevant und welche als irrelevant gelten.

Der Algorithmus

1. ***for all*** $A_i \in Features$
   $score_i = dependency(A_i)$
2. $F_{relevant} = \{A_i \in Features | score_i \geq \tau \cdot max\{score_i | i = 1 \dots n\}\}$
3. ***return*** $F_{relevant}$.

Dash, Choi, Scheuermann und Liu, Feature Selection for Clustering - A Filter Solution [5]

Einen anderen Weg als den der bisherigen Filter-Methoden gehen Dash et al. In ihrem Ansatz werden nicht einzelne Attribute bewertet, sondern Attributmengen. Mit Hilfe eines Entropiemaßes solchen Attributmengen, die Cluster, also gut separierbare Gruppierungen von Instanzen, enthalten, von solchen unterschieden werden, die keine Cluster enthalten. Mit einer sequenziellen Vorwärtssuche wird dann die optimale Attributmenge gesucht.

Da die Entropie die Unordnung eines Systems angibt, sollte sie maximal sein, wenn die Instanzen des Datensatzes bzgl. der ausgewählten Attribute gleichverteilt sind. Wenn dagegen gut voneinander abgrenzbare Cluster existieren, sollte die Entropie niedrig sein. Die Entropie wird auf Basis der Distanzen zwischen den Instanzen berechnet. Dash et al. erklären als Ziel ihrer Methode, Intra- und Inter-Cluster-Distanzen eine niedrige Entropie und anderen, verrauschten Distanzen eine höhere Entropie zuzuweisen.

Als erster Ansatz lässt sich die Entropie berechnen als

$$E = -\sum_{X_i} \sum_{X_j} [D_{ij} \log D_{ij} + (1 - D_{ij}) \log(1 - D_{ij})]$$

wobei D_{ij} die auf das Einheitsintervall normierte Distanz zwischen den Instanzen X_i und X_j ist. Die Entropie ist 0 für minimale und maximale Distanzen und 1 für mittlere Distanzen. Dieser Ansatz hat jedoch noch zwei Nachteile. Zum einen kann die mittlere Distanz 0.5, die die höchste Entropie erhält, eine Inter-Cluster-Distanz sein, für die eine niedrige Entropie gewünscht ist. Zum anderen steigt die Entropie bei kleinen Distanzen sehr schnell an, wodurch Intra-Cluster-Distanzen sehr unterschiedliche Entropiewerte erhalten.

Um diese unerwünschten Effekte zu beseitigen, werden zwei Änderungen durchgeführt. Ein Parameter μ dient dazu, die mittlere Distanz zu verschieben, optimalerweise auf einen Wert, der zwischen den Intra- und den Inter-Cluster-Distanzen liegt. Mit einem Koeffizienten β und der Verwendung der Exponentialfunktion anstelle des Logarithmus soll das zweite Problem behoben werden. Die neue Definition der Entropie lautet:

$$E = \sum_{X_i} \sum_{X_j} E_{ij}$$

mit

$$E_{ij} = \begin{cases} \frac{exp(\beta \cdot D_{ij}) - exp(0)}{exp(\beta \cdot \mu) - exp(0)} & : 0 \leq D_{ij} \leq \mu \\ \frac{exp(\beta \cdot (1.0 - D_{ij})) - exp(0)}{exp(\beta \cdot (1.0 - \mu)) - exp(0)} & : \mu \leq D_{ij} \leq 1.0 \end{cases}$$

Für β wurde in Experimenten ein Wert von 10 als geeignet befunden, μ wird für jede Attributmenge berechnet. Dazu wird innerhalb der angenommenen Intra-Cluster-Distanzen (5 - 30 % des gesamten Distanzumfangs scheinen geeignet) der Distanzbereich mit der höchsten Frequenz gesucht und dann μ nach der obigen Formel für $0 \leq D_{ij} \leq \mu$ berechnet, wobei für E ein kleiner Wert E_T und für D_{ij} die ermittelte Distanz eingesetzt wird.

Der Algorithmus

1. F = Menge aller Attribute des Datensatzes.
   Menge der ausgewählten Attribute $S = \{\}$
   $\beta = 10$
   $E = \infty$

2. ***for each*** $f_i \in F$
   $S_i = S \cup \{f_i\}$
   Berechne μ.
   $E_i = entropy(S', \beta, \mu)$

3. $E_{max} = max\{E_i\}$

4. ***if*** $E_{max} > E$
   $S = S_i$
   goto 2.

5. ***return*** S.

4.2.2 Methoden zum Entfernen redundanter Features

Mitra, Murthy und Pal, Unsupervised Feature Selection Using Feature Similarity [22]

Ziel dieses Ansatzes ist es, redundante Attribute aus einem Datensatz zu entfernen. Genauer gesagt sollen so viele Attribute wie möglich bei minimalem Informationsverlust entfernt werden. Dafür wird ein Maß benötigt, das die Ähnlichkeit zweier Attribute bestimmt. Mitra et al. beschreiben zwei solche Maße, die auf linearer Abhängigkeit zwischen Attributen basieren und stellen ein neues Maß, den *Maximal Information Compression Index*, vor. Die anderen Maße sind:

Korrelationskoeffizient ρ: Der Korrelationskoeffizient zwischen zwei Zufallsvariablen x und y ist definiert als

$$\rho(x,y) = \frac{cov(x,y)}{\sqrt{var(x)var(y)}}$$

wobei $var()$ die Varianz einer Variablen angibt und $cov()$ die Kovarianz zwischen zwei Variablen. Bei exakter linearer Abhängigkeit von x und y ist $\rho(x,y)$ 1 oder -1. Als Ähnlichkeitsmaß eignet sich also $1 - |\rho(x,y)|$.

Least Square Regression Error: Hierbei handelt es sich um ein Maß, das den Fehler bei der Vorhersage von y aus dem linearen Modell $y = a + bx$ berechnet. a und b sind die Regressionskoeffizienten, die durch die Minimierung des mittleren quadratischen Fehlers

$$e(x,y)^2 = \frac{1}{n}\sum (e(x,y)_i)^2$$

mit

$$e(x,y)_i = y_i - a - bx_i$$

ermittelt werden. Die Koeffizienten werden berechnet als $a = \overline{y}$ und $b = \frac{cov(x,y)}{var(x)}$ und $e(x,y)$ wird berechnet als $e(x,y) = var(y)(1 - \rho(x,y)^2)$. Es ist $e(x,y) = 0$, falls x und y in linearer Beziehung stehen und $e(x,y) = var(y)$, falls x und y komplett unabhängig voneinander sind.

Mitra et al. wünschen sich als Eigenschaften für ein gutes Ähnlichkeitsmaß u.a. Symmetrie (vom *Least Square Regression Error* nicht erfüllt), Sensitivität gegenüber Skalierung (vom *Korrelationskoeffizienten* nicht erfüllt) sowie Invarianz gegenüber Rotation (von beiden Maßen nicht erfüllt). All diese Eigenschaften erfüllt der *Maximal Information Compression Index*.

Maximal Information Compression Index (λ_2): Dieses Maß ist definiert als der kleinste Eigenwert der Kovarianzmatrix der Zufallsvariablen x und y:

$$\begin{aligned} 2\lambda_2(x,y) = {} & var(x) + var(y) \\ & - \sqrt{(var(x) + var(y))^2 - 4var(x)var(y)(1 - \rho(x,y)^2)} \end{aligned}$$

λ_2 ist 0, falls x und y linear abhängig sind und ist umso größer, je geringer die Abhängigkeit von x und y ist.

Der Algorithmus zur Feature Subset Selection basiert auf dem k-nearest-Neighbour Prinzip. Zu einem anfänglich gewählten k wird von jedem Attribut das k-nächste Attribut bestimmt. Das Attribut, dessen k-nächster

Nachbar die höchste Ähnlichkeit hat, wird ausgewählt und die k nächsten Attribute zu diesem Attribut werden herausgenommen. Der Abstand zum k-nächsten Attribut dient nun als Grenzwert. k wird solange reduziert, bis bei einem Attribut wieder der k-nächste Nachbar eine Ähnlichkeit größer als der Grenzwert hat. Dies wird solange wiederholt, bis $k = 1$ ist.

Der Algorithmus

1. F = Menge aller Attribute des Datensatzes.
   Wähle einen Wert für $k \leq D - 1$.
   Intialisiere die Auswahlmenge S als $S = F$.

2. ***for each*** $f_i \in S$
   Berechne r_i^k.

3. Finde Attribut $f_{i'}$, für das $r_{i'}^k$ minimal ist.

4. Entferne die k nächsten Attribute von $F_{i'}$ aus S. $\epsilon = r_{i'}^k$.

5. ***if*** $k > |S| - 1$
   $k = |S| - 1$

6. ***if*** $k = 1$
   goto 9.

7. ***while*** $r_{i'}^k > \epsilon$
   $k = k - 1$
   $r_{i'}^k = min(r_i^k)$
   if $k = 1$
   goto 9.

8. ***goto*** 2.

9. ***return*** S.

Kapitel 5

Ein neuer Ansatz

Der im Folgenden vorgestellte Ansatz ist eine Wrapper-Methode, basierend auf einem genetischen Algorithmus. Ein Problem bei genetischen Algorithmen ist im Allgemeinen ihre lange Laufzeit. Die vorgestellte Methode setzt an diesem Problem an. Es soll im Vergleich zu einem einfachen genetischen Algorithmus eine deutlich kürzere Laufzeit erreicht werden, ohne jedoch an Leistungsfähigkeit einzubüßen. Als Clustering-Algorithmus verwendet dieser Wrapper k-Means, es kann jedoch jeder Algorithmus eingesetzt werden, der auf dem Prizip der iterativen Optimierung basiert. Zunächst wird ein simpler Basisalgorithmus vorgestellt und danach gezeigt, wie durch einfache Modifikationen eine deutliche Verkürzung der Laufzeit erreicht werden kann.

5.1 Der Basisalgorithmus

Die Grundidee ist, einen genetischen Algorithmus zur Auswahl der Attribute sowie zum Finden der Clusteranzahl k zu verwenden. Für n Attribute und k_{max} als maximaler Clusteranzahl besteht das Genom eines Individuums aus $n + k_{max} - 2$ Bits. Die ersten n Bits geben für jedes Attribut an, ob es in die Auswahl aufgenommen werden soll oder nicht. Die letzten $k_{max} - 2$ Bits bestimmen die Clusteranzahl. Sie wird berechnet als

$$k = 2 + \sum_{i=n+1}^{n+k_{max}-2} bit_i.$$

Zuerst wird eine zufällige Population von p Individuen erzeugt, wobei p die vorher festgelegte Populationsgröße ist. Der Algorithmus läuft nun für eine ebenfalls vorher festgelegte Rundenzahl von r Runden. Am Anfang jeder Runde wird für jedes Individuum ein Fitnesswert berechnet. Dazu wird jeweils ein Clustering erzeugt, wobei nur die im Genom des Individuums gesetzten Attribute verwendet werden und die Clusteranzahl k wie oben angegeben aus dem Genom berechnet und für den Clustering-Algorithmus

festgelegt wird. Das Clustering wird dann mit einer Bewertungsfunktion bewertet und dieser Wert als Fitnesswert festgelegt.

Danach werden die Individuen in absteigender Reihenfolge nach ihren Fitnesswerten geordnet. Nun werden die letzten c Individuen aus der Population herausgenommen. c ist hierbei ein Wert, der vorher festgelegt wird. Die Population wird daraufhin mit c Nachkommen wieder aufgefüllt. Dabei werden jeweils zwei Eltern aus der verbliebenen Population zufällig ausgewählt, wobei die Wahrscheinlichkeit ausgewählt zu werden für jedes Individuum entsprechend seines Fitnesswertes festgelegt wird, d.h. Individuen mit hohem Fitnesswert erzeugen mit größerer Wahrscheinlichkeit Nachkommen als Individuen mit niedrigem Fitnesswert. Die Nachkommen werden nach dem Uniform-Crossover-Verfahren erzeugt und jedes Bit mit der Wahrscheinlichkeit $p_{mutation}$ invertiert. Tests mit dem Two-Point-Crossover-Verfahren fielen schlechter aus, weshalb Uniform Crossover bevorzugt wurde.

Nach der letzten Runde wird noch einmal für alle Individuen der Fitnesswert berechnet und die Attributauswahl des Individuums mit dem besten Fitnesswert zurückgegeben.

Der Algorithmus

1. Erzeuge eine Population P von p Individuen, wobei jedes Bit der Genome zufällig gesetzt wird.
2. ***for*** $j = 1$ ***to*** r
   (a) ***for all*** $i \in P$
      - Ermittele die Attributauswahl S und die Clusteranzahl k aus dem Genom von i.
      - $clustering = k\text{-}Means(S, k)$
      - $fitness(i) = evaluate(clustering)$

   (b) Ordne die Individuen nach ihrem Fitnesswert.

   (c) Entferne die c Individuen mit den schlechtesten Fitnesswert.

   (d) $sum_{fitness} = \sum_{i \in P} fitness(i)$

   (e) ***for all*** $i \in P$
      - $p_{crossover}(i) = \frac{fitness(i)}{sum_{fitness}}$

   (f) ***for*** $a = 1$ ***to*** c
      - Wähle zwei Individuen i_1 und i_2 zufällig, wobei jedes Individuum i mit Wahrscheinlichkeit $p_{crossover}(i)$ gewählt wird.
      - $child = mutation(crossover(i_1, i_2))$
      - $P = P \cup \{child\}$

3. ***for all*** $i \in P$

   - Ermittele die Attributauswahl S und die Clusteranzahl k aus dem Genom von i.
   - $clustering = k\text{-}Means(S, k)$
   - $fitness(i) = evaluate(clustering)$

4. Wähle das Individuum i mit dem höchsten Fitnesswert.

5. Ermittele die Attributauswahl S aus dem Genom von i.

6. ***return*** S.

5.2 Der verbesserte Algorithmus

Die Geschwindigkeit des Basisalgorithmus hängt ab zum einen von der Geschwindigkeit des Clustering-Algorithmus und zum anderen von der Geschwindigkeit der Bewertungsfunktion. Beide Komponenten werden wiederum in ihrer Geschwindigkeit beeinflusst von der Anzahl der Attribute, denn je höher die Dimensionalität des gewählten Attributraums, desto länger dauern die Abstandsberechnungen.

Um den Algorithmus mit einer Population von 20 Individuen 100 Runden laufen zu lassen, muss der Clustering-Algorithmus 2000 mal durchlaufen werden. Hierbei bietet sich ein großes Einsparpotential. Dazu muss man sich die Arbeitweise des Clustering-Algorithmus, in diesem Fall k-Means, vergegenwärtigen. Statt k-Means kann auch ein anderer Algorithmus, der nach dem Prinzip der iterativen Optimierung arbeitet, verwendet werden. Solche Algorithmen beginnen mit einem zufälligen Clustering und verbessern dieses iterativ solange, bis keine Veränderung oder nur noch eine Veränderung, die unter einem bestimmten Grenzwert liegt, auftritt. Es ist klar ersichtlich, unter der Voraussetzung, dass der Algorithmus zu einer Lösung konvergiert, dass der Algorithmus in den letzten Schritten nur noch kleine Veränderungen vornimmt. Es ist daher zu erwarten, dass er bereits nach wenigen Schritten ein Clustering erhält, welches sich nicht mehr grundsätzlich vom letztendlichen Clustering unterscheidet. Ebenso ist zu erwarten, dass sich die Bewertung des vorläufigen Clusterings durch die Bewertungsfunktion nicht mehr grundsätzlich von der Bewertung des letztendlichen Clusterings unterscheidet.

Diese Überlegungen kann man sich zunutze machen, indem man, statt den Clustering-Algorithmus jedes Mal komplett auszuführen, seine Ausführung auf eine bestimmte kleine Anzahl von Iterationen beschränkt. Dies wird im verbesserten Algorithmus umgesetzt. Die maximale Anzahl der k-Means-Iterationen pro Durchlauf wird auf einen kleinen Wert i (in den Tests wurde $i = 3$ verwendet) festgelegt.

Würde man es dabei belassen, wäre nun zwar ein Laufzeitvorteil des verbesserten Algorithmus im Vergleich zum Basisalgorithmus zu verzeichen, sehr wahrscheinlich müsste man aber Einbußen bei der Leistung hinnehmen. Besser wäre es, wenn der Algorithmus nur in den ersten Schritten mit den „unausgereiften“ Clusterings umgehen müsste, um dort eine grobe Selektion durchzuführen, bei der die wirklich schlechten Individuen aussortiert werden. Denn dafür dürften die zur Verfügung stehenden Clusterings ausreichend sein. Später jedoch, wenn es um geringere Unterschiede geht, wären Clusterings, die aus einem kompletten Durchlauf des Clustering-Algorithmus resultieren, für ein gutes Ergebnis der FSS erforderlich.

Die Lösung ist einfach. Jedem Individuum wird das auf Basis seines Genoms berechnete Clustering für die nächste Runde mitgegeben. In der nächsten Runde beginnt der Clustering-Algorithmus nun nicht mit einem zufällig erzeugten Clustering, sondern eben mit dem in der vorigen Runde erzeugten Clustering. Er setzt also seinen in der vorangegangen Runde abgebrochenen Optimierungsvorgang fort, wodurch nach einigen Runden das Clustering qualitativ dem eines kompletten Durchlaufs des Clustering-Algorithmus entspricht. Dies führt sogar zu einem nochmaligen Geschwindigkeitsgewinn, denn in den darauffolgenden Runden wird der Clustering-Algorithmus jeweils mit dem fertigen Clustering initialisiert und bricht bereits nach einer Iteration ohne Veränderung ab.

Nun stellt sich noch die Frage, ob und wie Clusterings an Nachkommen vererbt werden sollen. Berücksichtigt werden muss hier die Clusteranzahl, die sich bei Eltern und Nachkommen unterscheiden kann. Im Algorithmus ist die Weitergabe von Clusterings so geregelt, dass ein Nachkomme das Clustering eines Elternteils übernimmt, falls die Clusteranzahl bei beiden übereinstimmt. Gibt es zu beiden Elternteilen keine Übereinstimmung in der Clusteranzahl, so wird kein Clustering übernommen, sondern ein neues zufällig erzeugt. Tests mit einem Algorithmus, der aus den Clusterings der Eltern ein neues Clustering mit passender Clusteranzahl erzeugte, zeigten keine Verbesserung, weshalb diese Modifizierung wieder herausgenommen wurde, da sie natürlich auch die Laufzeit erhöhte.

Eine weitere Verbesserung des Algorithmus zur nochmaligen Verkürzung der Laufzeit setzt an der Dimensionalität an. Beim Basisalgorithmus wird die Population initialisiert, indem für jedes Individuum das Genom zufällig erzeugt wird, d.h. jedes Bit wird zufällig auf 1 oder 0 gesetzt. Das führt dazu, dass von den Individuen der Anfangspopulation im Mittel jeweils etwa die Hälfte der Attribute ausgewählt wird. Bei Datensätzen mit sehr vielen Attributen bedeutet das, dass zumindest in den ersten Runden die Abstandsberechnungen recht lange dauern im Vergleich zu den Berechnungen bei Datensätzen mit nur wenigen Attributen. Viel schneller wären die Berechnungen, wenn man die meisten der Attribut-Bits auf 0 setzen und nur wenige Attribute auswählen würde.

Die Frage dabei ist, ob eine so initialisierte Population im Schnitt weiter entfernt ist von der optimalen Lösung als eine völlig zufällig initialisierte Population. Denn wäre das der Fall, so wäre der Algorithmus in den Anfangsrunden zwar schneller, müsste aber länger suchen, um eine gute Lösung zu erreichen. Eine Abschätzung hierzu kann die Hamming-Distanz (also die Summe der bitweisen Unterschiede) zweier Genome liefern. Die Bits für die Clusteranzahl werden hier von der Betrachtung ausgenommen, da sie in beiden Initialisierungsvarianten auf die gleiche Weise erzeugt werden und daher im Bezug auf die Ähnlichkeit zur optimalen Lösung keine Rolle spielen. Im folgenden werden also Genome betrachtet, die ausschließlich Bits zur Attributauswahl enthalten.

Der Erwartungswert der Hamming-Distanz eines völlig zufällig erzeugten Genoms zum optimalen Genom liegt immer bei $\frac{n}{2}$ für einen Datensatz mit n Attributen, unabhängig von der Zahl der relevanten Attribute. Denn für jedes Bit besteht eine 50%-Chance, dass es mit dem entsprechenden Bit der Optimallösung übereinstimmt, womit im Schnitt die Hälfte der Bits richtig gesetzt sein wird.

Wie sieht nun die erwartete Hamming-Distanz für ein Individuum aus, bei dem z.B. genau zwei Attribute ausgewählt wurden? Für einen Datensatz mit n Attributen, von denen r Attribute relevant sind ($r \geq 2$), berechnet sich die erwartete Hamming-Distanz als

$$E(HD) = \frac{\binom{r}{2}(r-2) + r^2(n-r) + \binom{n-r}{2}(r+2)}{\binom{n}{2}}.$$

$\binom{r}{2}(r-2)$ ist die Anzahl der Möglichkeiten, wenn alle ausgewählten Attribute relevant sind, multipliziert mit der Hamming-Distanz für diesen Fall.

$r^2(n-r)$ ist die Anzahl der Möglichkeiten, wenn ein relevantes und ein irrelevantes Attribut gewählt wurden, multipliziert mit der Hamming-Distanz für diesen Fall.

$\binom{n-r}{2}(r+2)$ ist die Anzahl der Möglichkeiten, wenn beide gewählten Attribute irrelevant sind, wiederum multipliziert mit der Hamming-Distanz für diesen Fall.

$\binom{n}{2}$ ist die Summe aller Möglichkeiten.

Die erwartete Hamming-Distanz ist $< \frac{n}{2}$ für $r < \frac{n}{2}$ und $> \frac{n}{2}$ für $r > \frac{n}{2}$. Für $r < \frac{n}{2}$ ergibt sich also bei der Auswahl von nur zwei Attributen im Schnitt eine größere Ähnlichkeit zur optimalen Lösung im Vergleich zur Initialisierungsvariante mit völlig zufälliger Auswahl von Attributen. Auch wenn die Hamming-Distanz nur bedingt Aufkunft über die Erreichbarkeit der optimalen Lösung geben kann (hier spielt z.B. die Bewertungsfunktion eine große Rolle), so ist doch durch die Änderung zumindest kein Nachteil zu erwarten, wenn weniger als die Hälfte aller Attribute relevant ist.

Der Algorithmus

1. Erzeuge eine Population P von p Individuen, wobei von den ersten d Bits der Genome zwei zufällig gewählte auf 1 und die restlichen auf 0 gesetzt werden. Die letzten $k-2$ Bits werden zufällig gesetzt.
2. ***for*** $j = 1$ ***to*** r
   (a) ***for all*** $i \in P$
      - ***if*** $clustering(i) = null$
        - Ermittele die Attributauswahl S und die Clusteranzahl k aus dem Genom von i.
        - $clustering = k\text{-}Means(S, k, iterations)$
        - $clustering(i) = clustering$
      - ***else***
        - $clustering = k\text{-}Means(S, k, iterations, clustering(i))$
        - $clustering(i) = clustering$
      - $fitness(i) = evaluate(clustering)$

   (b) Ordne die Individuen nach ihrem Fitnesswert.

   (c) Entferne die c Individuen mit den schlechtesten Fitnesswert.

   (d) $sum_{fitness} = \sum_{i \in P} fitness(i)$

   (e) ***for all*** $i \in P$
      - $p_{crossover}(i) = \frac{fitness(i)}{sum_{fitness}}$

   (f) ***for*** $a = 1$ ***to*** c
      - Wähle zwei Individuen i_1 und i_2 zufällig, wobei jedes Individuum i mit Wahrscheinlichkeit $p_{crossover}(i)$ gewählt wird.
      - $child = mutation(crossover(i_1, i_2))$
      - $P = P \cup \{child\}$
3. ***for all*** $i \in P$
   - Ermittele die Attributauswahl S und die Clusteranzahl k aus dem Genom von i.
   - $fitness(i) = evaluate(clustering(i))$
4. Wähle das Individuum i mit dem höchsten Fitnesswert.
5. Ermittele die Attributauswahl S aus dem Genom von i.
6. ***return*** S.

5.3 Die Bewertungsfunktion

Das Herzstück eines FSS-Algorithmus ist die Bewertungsfunktion für Attributauswahlen, denn sie ist dafür verantwortlich, dass der Suchalgorithmus überhaupt nach einer gewünschten Attributauswahl sucht. Im Falle eines Wrappers wird eine Funktion benötigt, die Clusterings bewertet. Wie in 2.4 beschrieben, fließen in diese Funktion häufig die Varianz innerhalb der Clusters und die Abstände der Cluster zueinander ein.

Für den hier beschriebenen FSS-Algorithmus wurde eine Bewertungsfunktion entwickelt, die nur auf der Varianz bzw. der Standardabweichung innerhalb der Cluster basiert. Der Grund für den Verzicht auf die Cluster-Abstände ist die folgende Beobachtung: Normalerweise würde man erwarten, dass der Abstand zwischen zwei benachbarten Clusterzentren mit zunehmender Clusteranzahl sinkt, da sich mehr Cluster den gleichen Raum teilen. Dies ist jedoch oftmals nicht der Fall. Betrachtet man z.B. einen Datensatz mit zwei gleichverteilten Attributen, deren Wertebereich jeweils das Einheitsintervall ist, so wird man bei zwei Clustern einen Abstand von ca. 0.5 berechnen. Bei drei Clustern wird der durchnittliche Minimalabstand, also der durchschnittliche Abstand der Clusterzentren zum nächsten benachbarten Clusterzentrum, leicht größer sein, bei vier Clustern liegt er wieder bei 0.5. Bei mehr als vier Clustern ist er schließlich niedriger. Noch stärker ist dieser Effekt in höherdimensionalen Räumen, wie man in Tabelle 5.1 sieht. Der Effekt führt dazu, dass Clusterings mit mehr Clustern bevorzugt werden, was nicht erwünscht ist.

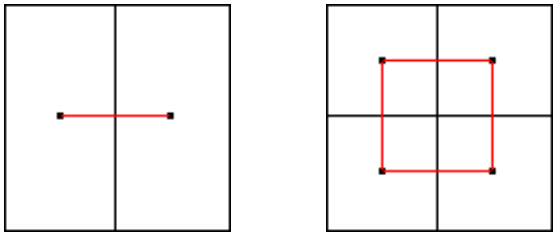

Abbildung 5.1: Minimalabstände der Clusterzentren bei zwei und vier Clustern

Außerdem zeigt sich, dass allein aufgrund der Varianz Clusterings mit gut separierten Clustern bevorzugt werden, zumindest, wenn sich durchdringende Cluster von vornherein ausgeschlossen sind, wie es bei k-Means der Fall ist, und unter der Voraussetzung, dass die Instanzen der Cluster normalverteilt sind.

Dimension\Cluster	2	3	4	6	10	20
1	0.512	0.336	0.249	0.162	0.097	0.047
2	0.517	0.549	0.504	0.365	0.289	0.201
3	0.511	0.554	0.570	0.564	0.403	0.332
5	0.508	0.563	0.576	0.627	0.620	0.537
10	0.493	0.565	0.615	0.652	0.679	0.708

Tabelle 5.1: Gemessene Minimalabstände von Clusterzentren bei gleichverteilten Instanzen

Als Basisfunktion dient also die Standardabweichung der Clusterinstanzen, und zwar als gewichteter Durchschnittswert aller Cluster:

$$score = \frac{1}{n} \sum_{i=1}^{k} n_i \cdot s_i = \frac{1}{n} \sum_{i=1}^{k} \left(n_i \cdot \sqrt{\frac{1}{n_i - 1} \sum_{j=1}^{n_i} d(x_{ij}, c_i)^2} \right)$$

Hierbei ist n die Anzahl der Instanzen des Datensatzes, n_i die Anzahl der Instanzen in Cluster i und s_i die Standardabweichung der Instanzen von Cluster i. x_{ij} ist die j-te Instanz von Cluster i, c_i das Clusterzentrum von Cluster i und d ein Abstandsmaß.

Diese Funktion muss jedoch noch angepasst werden um Clusterings unabhängig von der Dimension des Attribut-Unterraums und unabhängig von der Anzahl der Cluster bewerten zu können, denn beides ist bei der FSS notwendig. Clusterings mit unterschiedlicher Clusteranzahl und unterschiedlicher Dimension müssen vergleichbar sein.

Konkret ergeben sich zwei Probleme. Zum einen haben kleine Cluster (also Cluster, die aus wenigen Instanzen bestehen) im Allgemeinen eine kleinere Standardabweichung als große Cluster. Da bei Clusterings mit vielen Clustern die Cluster natürlich kleiner sind als bei Clusterings mit wenigen Clustern, bevorzugt die Bewertungsfunktion daher Clusterings mit vielen Clustern. Zum anderen sind die Distanzen in einem Raum mit hoher Dimension größer als in einem Raum mit niedriger Dimension, was sich auf die Standardabweichungen auswirkt. Somit werden Räume mit niedriger Dimension bevorzugt.

Zuerst soll nun das Problem mit der Clustergröße gelöst werden. Dazu muss untersucht werden, wie sich die Standardabweichung bei unterschiedlicher Clusteranzahl bzw. Clustergröße verhält. Hierbei hilft wieder die Betrachtung eines Datensatzes, dessen Instanzen in allen Attributen gleichverteilt sind.

Die Standardabweichung einer stetigen Gleichverteilung lässt sich berechnen und gilt praktisch als Erwartungswert für die Standardabweichung einer diskreten Gleichverteilung. Man berechnet im eindimensionalen Fall die Va-

rianz im Intervall $[a, b]$ als

$$\begin{aligned}
v &= \frac{1}{b-a}\int_a^b \left(x - \frac{a+b}{2}\right)^2 dx \\
&= \frac{1}{b-a}\int_a^b x^2 - (a+b)x + \frac{(a+b)^2}{4} dx \\
&= \frac{1}{b-a}\left[\frac{1}{3}x^3 - \frac{a+b}{2}x^2 + \frac{(a+b)^2}{4}x\right]_a^b \\
&= \frac{1}{b-a}\left[\frac{4x^3 - 6(a+b)x^2 + 3(a+b)^2 x}{12}\right]_a^b \\
&= \frac{1}{b-a}\frac{(b-a)^3}{12} \\
&= \frac{(b-a)^2}{12}
\end{aligned}$$

Somit ist die Standardabweichung

$$s = \sqrt{v} = \frac{b-a}{\sqrt{12}}$$

Man sieht leicht, dass eine Verdopplung der Clusteranzahl (= Halbierung der Clustergröße) zu einer Halbierung der Standardabweichungen und damit zur Halbierung des Ergebnisses der Bewertungsfunktion führt, da im Eindimensionalen eine Halbierung der Clustergröße gleichbedeutend mit der Halbierung des Intervalls ist. Im Zweidimensionalen sieht es jedoch anders aus. Die Varianz berechnet sich hier als

$$\begin{aligned}
v &= \frac{1}{b_2-a_2}\int_{a_2}^{b_2}\left[\frac{1}{b_1-a_1}\int_{a_1}^{b_1}\left(x_1 - \frac{a_1+b_1}{2}\right)^2 + \left(x_2 - \frac{a_2+b_2}{2}\right)^2 dx_1\right] dx_2 \\
&= \frac{1}{b_2-a_2}\int_{a_2}^{b_2}\frac{(b_1-a_1)^2}{12} + \left(x_2 - \frac{a_2+b_2}{2}\right)^2 dx_2 \\
&= \frac{1}{b_2-a_2}\left[\frac{(b_1-a_1)^2}{12}x_2 + \frac{4x_2^3 - 6(a_2+b_2)x_2^2 + 3(a_2+b_2)^2 x_2}{12}\right]_{a_2}^{b_2} \\
&= \frac{(b_1-a_1)^2}{12} + \frac{(b_2-a_2)^2}{12} \\
&= \frac{(b_1-a_1)^2 + (b_2-a_2)^2}{12}
\end{aligned}$$

wobei a_1 und b_1 die Intervallgrenzen der ersten Dimension und a_2 und b_2 die Intervallgrenzen der zweiten Dimension sind. Die Standardabweichung ist hier also

$$s = \frac{\sqrt{(b_1-a_1)^2 + (b_2-a_2)^2}}{\sqrt{12}}$$

Wenn wir davon ausgehen, dass die Attribute normalisiert sind auf gleiche Wertebereiche, können wir $a_1 = a_2 = a$ und $b_1 = b_2 = b$ setzen und erhalten

$$s = \frac{\sqrt{2}(b-a)}{\sqrt{12}}$$

Wie man leicht sieht, ist bei Dimension dim

$$s = \frac{\sqrt{dim}(b-a)}{\sqrt{12}}$$

Im Zweidimensionalen führt eine Vervierfachung der Clusteranzahl zur Halbierung der Standardabweichungen. Im Allgemeinen braucht man die 2^{dim}-fache Clusteranzahl zur Halbierung der Standardabweichungen.

Es sei hier jedoch bemerkt, dass es bei der Korrektur des Effekts nicht in erster Linie darum gehen soll, die geringeren Standardabweichungen bei größerer Clusteranzahl auszugleichen. Dies ist nur eine Folge des eigentlichen Problems, welches darin besteht, dass kleine Cluster allgemein eine geringere Standardabweichung besitzen als große Cluster. Benötigt wird also ein Korrekturfaktor für die Standardabweichungen der Cluster, der es erlaubt, Cluster unabhängig von ihrer Größe miteinander zu vergleichen.

2^{dim}-fache Clustergröße führt bei Gleichverteilung zu doppelter Standardabweichung. Dies kann ausgeglichen werden, indem die Standardabweichung durch den Wert $\sqrt[dim]{n_i}$ dividiert wird, wobei n_i die Anzahl der zum Cluster gehörenden Instanzen ist. Denn angenommen, alle Cluster haben die gleiche Größe und Standardabweichung und diese seien bei k Clustern n_{c_k} und s_{c_k}, dann gilt

$$\begin{aligned}
\frac{1}{n}\sum_{i=1}^{2^{dim}k}\left(\frac{n_{c_k}}{2^{dim}}\cdot\frac{s_{c_k}}{2\sqrt[dim]{\frac{n_{c_k}}{2^{dim}}}}\right) &= \frac{1}{n}\cdot 2^{dim}k\cdot\frac{n_{c_k}}{2^{dim}}\cdot\frac{s_{c_k}}{2\sqrt[dim]{\frac{n_{c_k}}{2^{dim}}}} \\
&= \frac{1}{n}\cdot k\cdot n_{c_k}\cdot\frac{s_{c_k}}{2\sqrt[dim]{\frac{n_{c_k}}{2^{dim}}}} \\
&= \frac{1}{n}\cdot k\cdot\frac{n_{c_k}}{\sqrt[dim]{n_{c_k}}}\cdot s_{c_k} \\
&= \frac{1}{n}\sum_{i=1}^{k}\left(\frac{n_{c_k}}{\sqrt[dim]{n_{c_k}}}\cdot s_{c_k}\right)
\end{aligned}$$

Damit berechnet sich die Funktion als

$$score = \frac{1}{n}\sum_{i=1}^{k}\left(\frac{n_i}{\sqrt[dim]{n_i}}\cdot s_i\right).$$

Die Funktion liefert nun bei Gleichverteilung gleiche Werte unabhängig von der Clusteranzahl, jedoch noch nicht unabhängig von der Dimension.

Zunächst muss beachtet werden, dass der eben eingefügte Korrekturfaktor abhängig von der Dimension ist. Dies gilt es auszugleichen, natürlich ohne dabei die gewünschte Wirkung wieder zu neutralisieren. Zieht man den Korrekturfaktor aus der Summe heraus, so ergibt sich

$$\frac{1}{n}\sum_{i=1}^{k}\left(\frac{n_i}{\sqrt[dim]{n_i}}\cdot s_i\right) = \frac{k}{\sqrt[dim]{n_i}}\cdot\frac{1}{n}\sum_{i=1}^{k}\left(n_i\cdot s_i\right).$$

Im Fall $k = 1$ wird aus dem Faktor $\frac{1}{\sqrt[dim]{n}}$. Hier kommt nun k nicht mehr vor und auch nicht die Clustergröße n_i. Fügt man nun also den Faktor $\sqrt[dim]{n}$ ein, hat man für $k = 1$ wieder den ursprünglichen Wert. Da $\sqrt[dim]{n}$ eine Konstante ist, ändert sich auch nichts an der erreichten Korrektur bzgl. der Clustergröße. Die Funktion sieht nun folgendermaßen aus:

$$score = \frac{1}{n}\sqrt[dim]{n}\sum_{i=1}^{k}\left(\frac{n_i}{\sqrt[dim]{n_i}}\cdot s_i\right).$$

Schließlich muss noch der Einfluss der Dimension herausgerechnet werden. Das geschieht, indem alle berechneten Distanzen mit dem Faktor $\frac{1}{\sqrt{dim}}$ normalisiert werden:

$$\begin{aligned}
score &= \frac{1}{n}\sqrt[dim]{n}\sum_{i=1}^{k}\left(\frac{n_i}{\sqrt[dim]{n_i}}\cdot\sqrt{\frac{1}{n_i-1}\sum_{j=1}^{n_i}\frac{d(x_{ij},c_i)^2}{dim}}\right)\\
&= \frac{1}{n}\sqrt[dim]{n}\sum_{i=1}^{k}\left(\frac{n_i}{\sqrt[dim]{n_i}}\cdot\sqrt{\frac{1}{n_i-1}\frac{1}{dim}\sum_{j=1}^{n_i}d(x_{ij},c_i)^2}\right)\\
&= \frac{1}{n}\sqrt[dim]{n}\sum_{i=1}^{k}\left(\frac{n_i}{\sqrt[dim]{n_i}}\cdot\frac{s_i}{\sqrt{dim}}\right)\\
&= \frac{1}{n}\sqrt[dim]{n}\frac{1}{\sqrt{dim}}\sum_{i=1}^{k}\left(\frac{n_i}{\sqrt[dim]{n_i}}\cdot s_i\right)
\end{aligned}$$

Es wurde nun erreicht, dass Clusterings in gleichverteilten Daten immer gleich bewertet werden. Dabei ist es irrelevant, ob die Clusterings aus vielen oder wenigen Clustern bestehen und ob viele oder wenige Attribute ausgewählt wurden. Die Funktion erfüllt nun also die Bedingung einer Bewertung unabhängig von der Dimension und von der Anzahl der Cluster.

Kapitel 6

Testergebnisse

Um die Leistung des neuen Algorithmus unter Beweis zu stellen, wurden ausführliche Tests an synthetischen und an realen Daten durchgeführt. Getestet wurden der oben vorgestellte Algorithmus, im folgenden FastGeneticFSS genannt, eine vereinfachte Version des Algorithmus (SimpleGeneticFSS), die zur Fitnessberechnung den Clustering-Algorithmus immer vollständig durchlaufen lässt und keine Clusterings von einer Runde zur nächsten übergibt, und zwei der Algorithmen, die in Kap. 4 vorgestellt wurden, nämlich der auf einem Entropiemaß basierende Filter von Dash et al. [5] (DashFSS) sowie der auf einem evolutionären Algorithmus basierende Wrapper von Kim et al. [18] (KimFSS). Die getesteten Algorithmen sind eigene Implementierungen, Abweichungen des DashFSS- und des KimFSS-Algorithmus von den originalen Algorithmen können nicht ausgeschlossen werden.

6.1 Die Testdaten

Getestet wurde mit verschiedenen synthetischen Datensätzen sowie vier realen Datensätzen aus dem UCI Machine Learning Repository [4].

6.1.1 Synthetische Daten

Die Datensätze enthalten verschiedene Arten von Attributen: relevante Attribute, irrelevante Attribute mit gleichverteilten Werten und irrelevante Attribute mit normalverteilten Werten.

Relevante Attribute: Die relevanten Attribute sind die Attribute, die die Cluster enthalten. Sie werden folgendermaßen erzeugt:

Eine vorgegebene Anzahl von Clusterzentren wird durch zufällig gewählte Werte für die relevanten Attribute festgelegt. Die Attributwerte der Clusterzentren liegen dabei immer zwischen 0 und 100. Um gut separierte Cluster zu erhalten, ist ein Mindestabstand zwischen den Clusterzentren vorgegeben.

Die Festlegung der Clusterzentren wird solange wiederholt, bis die Clusterzentren so gewählt wurden, dass der Mindestabstand eingehalten wird.

Jede Instanz wird nun zufällig einem der Cluster zugeordnet. Die Attributwerte der Instanzen werden so bestimmt, dass sie normalverteilt um die Clusterzentren liegen, mit einer vorher festgelegten Standardabweichung.

Es gibt also in einem Datensatz genau ein vorgegebenes Clustering, das sich über alle relevanten Attribute erstreckt. Um die Zugehörigkeit einer Instanz zu einem Cluster erkennen zu können, wird ein Klassenattribut erzeugt, das eben diese Clusterzugehörigkeit anzeigt. Während der Tests wird dieses Attribut selbstverständlich herausgenommen. Es dient ausschließlich zur Bewertung der Ergebnisse.

Irrelevante gleichverteilte Attribute: Bei den gleichverteilten Attributen wird jeder Instanz ein zufälliger Wert zwischen 0 und 100 zugewiesen.

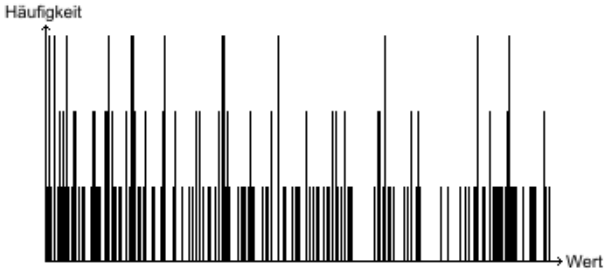

Abbildung 6.1: Gleichverteiltes Attribut

Irrelevante normalverteile Attribute: Von den normalverteilen Attributen gibt es zwei Arten. Bei der ersten Art wird als Erwartungswert 50 gewählt und die Attributwerte der Instanzen so bestimmt, dass sie normalverteilt um den Erwartungswert liegen, mit einer vorher festgelegten Standardabweichung.

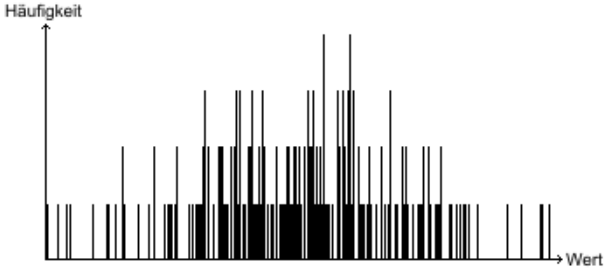

Abbildung 6.2: Normalverteiltes Attribut, erste Variante

Bei der zweiten Art wird als Erwartungswert 0 gewählt und die Werte der Instanzen normalverteilt gewählt, mit einer vorher festgelegten Standardab-

weichung. Jedoch wird den Instanzen dann jeweils der Betrag des gewählten Wertes zugewiesen, so dass also nur positive Werte vorkommen.

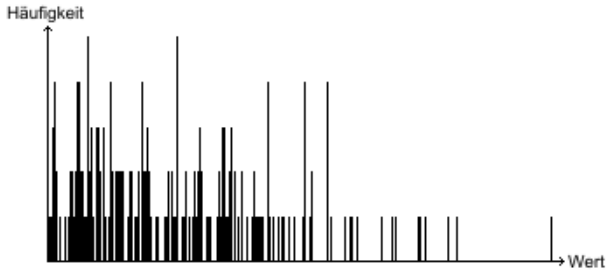

Abbildung 6.3: Normalverteiltes Attribut, zweite Variante

Es wurden Datensätze mit verschiedenen Konfigurationen erzeugt, die sich in vier Gruppen einteilen lassen.

Gruppe D1: Diese Gruppe besteht aus Datensätzen mit zwei relevanten Attributen, die vier Cluster enthalten. Dabei wurde für die Cluster eine Standardabweichung von 6 gewählt. Diese Datensätze enthalten außerdem jeweils zwei irrelevante gleichverteilte Attribute und je zwei Attribute der beiden Arten von irrelevanten normalverteilten Attributen. Die Datensätze enthalten also insgesamt acht Attribute, von denen zwei relevant sind.

Gruppe D2: Die Datensätze dieser Gruppe entsprechen denen der Gruppe D1, mit dem Unterschied, dass für die Cluster eine Standardabweichung von 10 gewählt wurde.

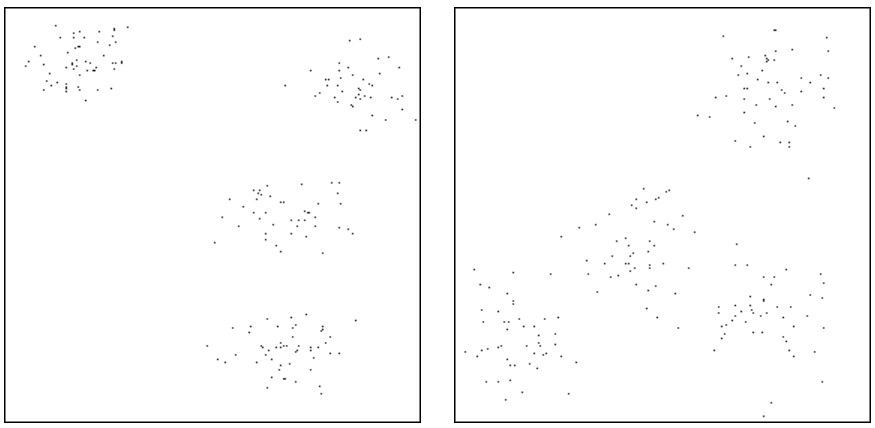

Abbildung 6.4: Beispiele für Cluster mit Standardabweichung 6 (links) und 10 (rechts)

Gruppe D3: Diese Gruppe enthält Datensätze mit vier relevanten Attributen, die vier Cluster enthalten, wobei eine Standardabweichung von 6 gewählt wurde. Außerdem enthalten die Datensätze je zehn Attribute der verschiedenen irrelevanten Attributarten. Die Gesamtzahl der Attribute beträgt also 34.

Gruppe D4: In dieser Gruppe befinden sich Datensätze, die denen der Gruppe D3 entsprechen, mit dem Unterschied, dass für die Cluster eine Standardabweichung von 10 gewählt wurde.

	D1	D2	D3	D4
relevante Attribute	2	2	4	4
irrelevante Attribute	6	6	30	30
Std.-Abw. der Cluster	6	10	6	10
Anzahl der Instanzen	200	200	200	200

Tabelle 6.1: Übersicht über die Konfiguration der Datengruppen

Für jede Gruppe wurden 30 verschiedene Datensätze erzeugt, alle mit jeweils 200 Instanzen. Jeder Algorithmus führte auf jedem der Datensätze genau einmal eine FSS durch. Dies soll eine objektive Beurteilung der Ergebnisse ermöglichen. Bei der Verwendung eines einzelnen Datensatzes pro Konfiguration bestünde die Gefahr, dass die Ergebnisse aufgrund der Besonderheiten dieses einzelnen Datensatzes anders ausfielen als es bei anderen Datensätzen mit gleicher Konfiguration der Fall wäre. Bei 30 Datensätzen kann erwartet werden, dass die Ergebnisse in etwa die wirklichen Leistungen der Algorithmen auf Datensätzen der jeweiligen Konfiguration widerspiegeln.

6.1.2 UCI-Daten

Aus dem UCI Repository wurden die Datensätze iris, wine, glass und segmentation verwendet. Diese Datensätze wurden ausgewählt, weil sie nicht zu groß waren um mit vertretbarem Zeitaufwand die Tests durchzuführen, und weil sich allgemein mit Clusterings auf diesen Datensätzen die Klasseneinteilung mehr oder weniger gut vorhersagen ließ, was bei anderen Datensätzen teilweise fast gar nicht möglich war.

iris Die *Iris Plants Database* ist ein sehr bekannter und oft verwendeter Datensatz aus dem Repository. Er besteht aus 150 Instanzen in drei Klassen, wobei jede Klasse einem Typ von Iris-Pflanzen entspricht. Der Datensatz enthält neben dem Klassenattribut vier numerische Attribute, von denen die beiden letzten als besonders relevant eingeschätzt werden [5].

wine Die *Wine Recognition Database* enthält Ergebnisse von chemischen Analysen dreier Weinsorten, die alle in der selben Region in Italien gewachsen sind. Der Datensatz besteht aus 178 Instanzen, die durch 13 numerische Attribute und das Klassenattribut, das die Zugehörigkeit zur Sorte angibt, beschrieben werden.

glass Die *Glass Identification Database* enthält Daten zu verschiedenen Glas-Sorten. Der Datensatz besteht aus 214 Instanzen, neun numerische Attribute beschreiben die chemische Zusammensetzung des Glases, das Klassenattribut gibt den Verwendungszweck des Glases an. Es gibt sieben Klassen.

segmentation Die *Image Segmentation Database* enthält Daten, die aus sieben digitalen Fotos erstellt wurden. Jede der 210 Instanzen beschreibt eine 3x3 Pixel Region eines der Fotos. Sie zeigen Ausschnitte von einer Ziegelmauer, vom Himmel, von Laub, Zement und Gras sowie von einem Fenster und von einem Fußpfad. 19 numerische Attribute beschreiben die Instanzen. Die Attribute 3,4 und 5 wurden für die Tests herausgenommen, weil ihre Werte konstant bzw. nahezu konstant sind. Das Klassenattribut gibt an, was der Ausschnitt zeigt.

	iris	wine	glass	segmentation
Instanzen	150	178	214	210
Attribute	4	13	9	16
Klassen	3	3	7	7

Tabelle 6.2: Übersicht über die UCI-Datensätze

Alle Datensätze wurden mit den Algorithmen FastGeneticFSS, SimpleGeneticFSS und KimFSS je zehn mal getestet, für den DashFSS-Algorithmus reichte je ein Testlauf aus, da er deterministisch arbeitet. Als zu findendes Clustering wurde jeweils die Klassifizierung angenommen. Das Klassenattribut wurde für die Tests herausgenommen.

Sowohl die synthetischen als auch die realen Datensätze wurden vor den Tests normalisiert. Der Wertebereich aller Attribute wurde auf das Einheitsintervall transformiert.

6.2 Bewertungskriterien

Die Ergebnisse der Algorithmen wurden nach verschiedenen Kriterien bewertet.

Wahl der richtigen Attribute: Da bei den synthetischen Daten bekannt ist, welche Attribute relevant sind, kann die Qualität einer Attributauswahl danach beurteilt werden, wie viele relevante Attribute nicht gewählt wurden und wie viele irrelevante Attribute gewählt wurden (0 ist in beiden Fällen das Optimum). Bei den realen Daten ist diese Beurteilung nicht möglich, da nicht bekannt ist, welche Attribute relevant und welche irrelevant sind.

Vorhersagegenauigkeit (Predictive Accuracy): Es ist durchaus möglich, dass nicht alle relevanten Attribute benötigt werden, um das korrekte Clustering zu erzeugen. Da die Clusterzugehörigkeit der Instanzen bekannt ist, kann die Qualität einer Attributauswahl danach beurteilt werden, wie gut ein mit der Auswahl erzeugtes Clustering das korrekte Clustering widerspiegelt. Bei den getesteten Wrapper-Algorithmen, die alle auch die Clusteranzahl ermitteln, wurden sowohl Clusterings mit der ermittelten Clusteranzahl als auch Clusterings mit der korrekten Clusteranzahl erzeugt und bewertet. Bei dem Filter-Algorithmus wurden nur Clusterings mit der korrekten Clusteranzahl erzeugt und bewertet. Zur Bewertung diente folgendes Maß:

$$PA = \frac{quote_{same} + quote_{diff}}{2}$$

mit

$$quote_{same} = \frac{\sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} right\_same(x_i, x_j)}{\sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} stated\_same(x_i, x_j)}$$

$$quote_{diff} = \frac{\sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} right\_diff(x_i, x_j)}{\sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} stated\_diff(x_i, x_j)}$$

und

$$right\_same(x_i, x_j) = \begin{cases} 1 & : x_i \text{ und } x_j \text{ im selben Cluster korrekt} \\ 0 & : x_i \text{ und } x_j \text{ im selben Cluster inkorrekt} \end{cases}$$

$$stated\_same(x_i, x_j) = \begin{cases} 1 & : x_i \text{ und } x_j \text{ im selben Cluster} \\ 0 & : x_i \text{ und } x_j \text{ in verschiedenen Clustern} \end{cases}$$

$$right\_diff(x_i, x_j) = \begin{cases} 1 & : x_i \text{ und } x_j \text{ in verschiedenen Clustern korrekt} \\ 0 & : x_i \text{ und } x_j \text{ in verschiedenen Clustern inkorrekt} \end{cases}$$

$$stated\_diff(x_i, x_j) = \begin{cases} 1 & : x_i \text{ und } x_j \text{ in verschiedenen Clustern} \\ 0 & : x_i \text{ und } x_j \text{ im selben Cluster} \end{cases}$$

6.3 Konfiguration der Algorithmen

Die Tabellen 6.3 bis 6.6 zeigen die Konfiguration der Algorithmen für die verschiedenen Daten. Der KimFSS-Algorithmus berechnet nicht eine beste Lösung, sondern eine Menge von Lösungen. Als Ergebnis wurde hier jeweils die Lösung verwendet, die das Produkt $F_{within} \cdot F_{between}$ maximierte.

Die Konfigurationen des DashFSS- und des KimFSS-Algorithmus wurden größtenteils so gewählt wie von den Autoren vorgeschlagen bzw. wie in deren Tests verwendet. Die Werte von Z_{within} und $Z_{between}$ wurden zu den synthetischen Daten so bestimmt, dass sie F_{within} und $F_{between}$ für alle Datensätze einer Gruppe so gut wie möglich normalisieren. Die Konfigurationen des SimpleGeneticFSS- und des FastGeneticFSS-Algorithmus wurden so gewählt, dass sie am besten zu funktionieren schienen. Natürlich ließen sich durch eine Erhöhung der Rundenzahl möglicherweise bessere Ergebnisse erzielen, jedoch wurde beobachtet, dass z.B. bei den Datengruppen D3 und D4 die Population sich entweder recht früh in Richtung eines guten Ergebnisses bewegte, welches nach 200 Runden fast immer stabil war, oder aber die Entwicklung in eine völlig falsche Richtung ging (keine relevanten Attribute ausgewählt), wodurch auch bei einer größeren Rundenzahl im Allgemeinen kein besseres Ergebnis erreicht worden wäre.

SimpleGeneticFSS	D1	D2	D3	D4	iris	wine	glass	segm.
Population	20	20	20	20	20	20	20	20
Runden	100	100	200	200	100	100	100	150
k_{max}	10	10	10	10	10	10	10	10
Nachkommen	5	5	5	5	5	5	5	5
$p_{mutation}$	0.1	0.1	0.03	0.03	0.1	0.1	0.1	0.06

Tabelle 6.3: Konfiguration des SimpleGeneticFSS-Algorithmus

FastGeneticFSS	D1	D2	D3	D4	iris	wine	glass	segm.
Population	20	20	20	20	20	20	20	20
Runden	100	100	200	200	100	100	100	150
k_{max}	10	10	10	10	10	10	10	10
Nachkommen	5	5	5	5	5	5	5	5
$p_{mutation}$	0.1	0.1	0.03	0.03	0.1	0.1	0.1	0.06
k-Means-Iterationen	3	3	3	3	3	3	3	3

Tabelle 6.4: Konfiguration des FastGeneticFSS-Algorithmus

KimFSS	D1	D2	D3	D4	iris	wine	glass	segm.
Population Anfang	20	20	20	20	20	20	20	20
Population maximal	100	100	100	100	100	100	100	100
Runden	400	400	800	800	400	400	400	600
k_{max}	10	10	10	10	10	10	10	10
θ	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
$p_{mutation}$	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Bins $F_{clusters}$	9	9	9	9	9	9	9	9
Bins $F_{complexity}$	10	10	10	10	10	10	10	10
Bins $F_{between}$	10	10	10	10	10	10	10	10
Bins F_{within}	10	10	10	10	10	10	10	10
$Z_{between}$	90	70	90	70	60	45	110	210
Z_{within}	12	12	12	12	3.5	6.5	6	10

Tabelle 6.5: Konfiguration des KimFSS-Algorithmus

DashFSS	D1	D2	D3	D4	iris	wine	glass	segm.
β	10	10	10	10	10	10	10	10
E_T	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
R_I	20%	20%	20%	20%	20%	20%	20%	20%

Tabelle 6.6: Konfiguration des DashFSS-Algorithmus

6.4 Ergebnisse

Tabelle 6.7 zeigt die Testergebnisse der Algorithmen für die synthetischen Daten. In der Spalte A_i ist die durchschnittliche Anzahl der ausgewählten irrelevanten Attribute angegeben, in der Spalte A_r die durchschnittliche Anzahl der nicht ausgewählten relevanten Attribute. $PA_{k_{gef}}$ bezeichnet die Vorhersagegenauigkeit, wenn ein Clustering auf den ausgewählten Attributen mit der vom FSS-Algorithmus gefundenen Clusteranzahl erstellt wird. $PA_{k_{korr}}$ ist die Vorhersagegenauigkeit, die ein Clustering erreicht, welches ebenfalls auf den ausgewählten Attributen erzeugt wurde, wobei aber der Clustering-Algorithmus die korrekte Clusteranzahl (also $k = 4$ an Stelle des von der FSS gefundenen k) als Eingabe erhielt. Als Vergleichswert ist jeweils noch die Vorhersagegenauigkeit, die ein Clustering mit der korrekten Clusteranzahl, das auf Basis aller Attribute erstellt wurde, erreicht. Dieser Wert zeigt, wie gut die Clusterzugehörigkeit vorhergesagt werden kann, wenn keine FSS durchgeführt wird. Alle Clusterings wurden mit k-Means durchgeführt.

Man erkennt, dass bei allen synthetischen Daten der SimpleGeneticFSS- und der FastGeneticFSS-Algorithmus deutlich bessere Ergebnisse erzielen als die Vergleichsalgorithmen.

D1	A_i	A_r	$PA_{k_{gef}}$	$PA_{k_{korr}}$
alle Attribute				0.7749 (0.087)
SimpleGeneticFSS	0	0	0.9342 (0.050)	0.9998 (0.001)
FastGeneticFSS	0	0	0.9719 (0.042)	0.9998 (0.001)
KimFSS	0.0667	1.0667	0.8114 (0.108)	0.8214 (0.098)
DashFSS	0.4667	1	-	0.7998 (0.171)
D2				
alle Attribute				0.6386 (0.076)
SimpleGeneticFSS	0.7	0.4667	0.8323 (0.189)	0.8561 (0.194)
FastGeneticFSS	0.6333	0.2667	0.8729 (0.170)	0.9053 (0.166)
KimFSS	0.7333	1.7333	0.5800 (0.136)	0.5706 (0.121)
DashFSS	1.5667	1.6667	-	0.5760 (0.110)
D3				
alle Attribute				0.8656 (0.102)
SimpleGeneticFSS	0.8667	0.5	0.8376 (0.126)	0.9463 (0.153)
FastGeneticFSS	0.3	0.1667	0.9140 (0.072)	1.0000 (0)
KimFSS	3.03	2.6	0.6590 (0.147)	0.7103 (0.188)
DashFSS	0.8	2.9667	-	0.7411 (0.187)
D4				
alle Attribute				0.7313 (0.094)
SimpleGeneticFSS	1.2	0.8667	0.8155 (0.143)	0.9013 (0.181)
FastGeneticFSS	1.7	1.1333	0.8271 (0.194)	0.8782 (0.214)
KimFSS	2.4667	3.3667	0.5709 (0.086)	0.6000 (0.116)
DashFSS	1.3667	3.5	-	0.5983 (0.131)

Tabelle 6.7: Testergebnisse für die synthetischen Daten (in Klammern die Standardabweichungen)

Bei den Datensätzen der D1-Gruppe wird immer die korrekte Attributauswahl gefunden. Diese ermöglicht mit der korrekten Clusteranzahl stehts eine korrekte Vorhersage der Clusterzugehörigkeiten. Die Vorhersagegenauigkeit mit dem gefundenen k liegt etwas niedriger, was bedeutet, dass das gefundene k nicht immer korrekt war. Der FastGeneticFSS-Algorithmus schneidet hier etwas besser ab als der SimpleGeneticFSS-Algorithmus. Die Vorhersagegenauigkeit mit allen Attributen und korrektem k wird jedoch von beiden Algorithmen immer noch deutlich übertroffen.

Die Vergleichsalgorithmen können mit ihrer Attributauswahl bei der D1-Gruppe die Vorhersagegenauigkeit kaum steigern in Bezug auf die Genauigkeit mit allen Attributen. Das liegt daran, dass sie beide fast immer nur eins der relevanten Attribute auswählen. Der DashFSS-Algorithmus schneidet leicht schlechter ab als der KimFSS-Algorithmus, weil er manchmal ein irrelevantes Attribut in die Auswahl nimmt.

Die D2-Gruppe zeigt, dass die Vergleichsalgorithmen sehr gut separierte Cluster brauchen um die relevanten Attribute zu finden. Durch die größere Standardabweichung von 10 im Vergleich zu 6 bei D1 versagen sie hier fast völlig. Die relevanten Attribute werden kaum gefunden, was zu einer Vorhersagegeneuigkeit von etwa 57% führt, was nur wenig über den zu erwartenden 50% für komplettes Raten und unter den 64% für die Vorhersage mit allen Attribute liegt. SimpleGeneticFSS und FastGeneticFSS schneiden hier deutlich besser ab. Die relevanten Attribute werden meistens gefunden und gelegentlich hinzugenommene irrelevante Attribute stören die Vorhersage nur wenig. Der FastGeneticFSS-Algorithmus kann sich auch hier wieder, diesmal noch etwas deutlicher, durchsetzen. Der Unterschied ist jedoch, wie auch bei D1, zwischen den beiden Algorithmen nicht signifikant.

Bei der D3-Gruppe sind nun mehr Attribute, vier relevante und 30 irrelevante, im Spiel, mit der geringen Standardabweichung von 6 bei den Clustern. Die beiden genetischen Algorithmen haben auch hier die Nase vorn, wobei der FastGeneticFSS-Algorithmus sowohl weniger irrelevante Attribute auswählt als auch weniger relevante Attribute weglässt und dadurch mit der korrekten Clusteranzahl immer auf eine korrekte Vorhersage kommt. SimpleGeneticFSS ist hier signifikant schlechter. Der DashFSS-Algorithmus wählt meistens wieder nur ein relevantes Attribut aus und kommt deshalb nur auf 74% Vorhersagegenauigkeit und ist damit noch etwas besser als der KimFSS-Algorithmus, der im Schnitt zwar mehr relevante Attribute auswählt, aber auch deutlich mehr irrelevante Attribute und deshalb nur 71% Vorhersagegenauigkeit erreicht.

Bei der D4-Gruppe kommt es zu ähnlichen Ergebnissen wie bei D2. Interessanterweise schneidet hier der SimpleGeneticFSS-Algorithmus etwas besser ab als der FastGeneticFSS-Algorithmus, allerdings nicht signifikant. Die Vergleichsalgorithmen sind hier wieder weit abgeschlagen und kommen auf etwa 60% Vorhersagegenauigkeit.

iris	Anzahl gew. Attr.	$PA_{k_{gef}}$	$PA_{k_{korr}}$
alle Attribute	4		0.823 (0.059)
SimpleGeneticFSS	2.3 (0.483)	0.719 (0.047)	0.925 (0.029)
FastGeneticFSS	2.3 (0.483)	0.710 (0.036)	0.925 (0.029)
KimFSS	1 (0)	0.844 (0.061)	0.943 (0)
DashFSS	1	-	0.578 (0.009)
wine			
alle Attribute	13		0.920 (0.013)
SimpleGeneticFSS	5.2 (0.632)	0.653 (0.019)	0.891 (0.036)
FastGeneticFSS	5.2 (0.632)	0.642 (0.019)	0.871 (0.05)
KimFSS	1 (0)	0.609 (0.027)	0.657 (0.04)
DashFSS	4	-	0.776 (0.086)
glass			
alle Attribute	9		0.593 (0.016)
SimpleGeneticFSS	2.9 (0.876)	0.597 (0.025)	0.596 (0.029)
FastGeneticFSS	2.1 (0.316)	0.609 (0.008)	0.611 (0.005)
KimFSS	1 (0)	0.601 (0.032)	0.579 (0.014)
DashFSS	1	-	0.607 (0.005)
segmentation			
alle Attribute	16		0.772 (0.023)
SimpleGeneticFSS	4.8 (0.632)	0.637 (0.035)	0.677 (0.049)
FastGeneticFSS	4.8 (1.317)	0.600 (0.05)	0.636 (0.075)
KimFSS	2.5 (1.269)	0.716 (0.094)	0.682 (0.08)
DashFSS	1	-	0.540 (0)

Tabelle 6.8: Testergebnisse für die UCI-Daten (in Klammern die Standardabweichungen)

Wie man in Tabelle 6.8 sieht, ergibt sich bei den realen Daten ein etwas anderes Bild. Beim iris-Datensatz schneidet der KimFSS-Algorithmus am besten ab, obwohl er immer nur ein Attribut auswählt. FastGeneticFSS und SimpleGeneticFSS liegen knapp dahinter, sie wählen zwei bis drei Attribute aus. Die Vorhersagegenauigkeit ist bei der Verwendung der korrekten Clusteranzahl deutlich höher als bei der vom Algorithmus ermittelten Anzahl, da diese sich hier meist stark von der korrekten Anzahl unterscheidet. Der DashFSS-Algorithmus erreicht beim iris-Datensatz ein schwaches Ergebnis, da er mit Attribut 2 ein eher irrelevantes Attribut auswählt. Es sei hier darauf hingewiesen, dass sich dieses Ergebnis nicht mit dem von Dash et al. in [5] dokumentierten Ergebnis für den iris-Datensatz deckt (das Ergebnis von Dash et al. ließ sich auch mit verschiedenen anderen Konfigurationen des Algorithmus nicht herbeiführen).

Beim wine-Datensatz können alle Algorithmen nicht die Vorhersagegenauigkeit erreichen, die man mit einem Custering, das auf allen Attributen basiert, erhält. FastGenetivFSS und SimpleGeneticFSS kommen jedoch auf eine nur wenig (aber dennoch signifikant) geringere Genauigkeit. Etwas schlechter ist der DashFSS-Algorithmus, der dafür aber mit einem Attribut weniger auskommt. Der KimFSS-Algorithmus schneidet hier am schlechtesten ab und kommt mit nur einem ausgewählten Attribut auf 65 % Vorhersagegenauigkeit.

Die Vorhersagegenauigkeit beim glass-Datensatz ist allgemein recht niedrig. Die Ergebnisse aller Algorithmen sind hier recht ähnlich und liegen in etwa bei der Genauigkeit, die man mit allen Attributen erreicht. KimFSS und DashFSS reicht allerdings jeweils ein Attribut, während die beiden anderen Algorithmen zwei bis drei Attribute auswählen.

Beim segmentation-Datensatz lässt sich die beste Vorhersagegenauigkeit wieder mit allen Attributen erzielen. KimFSS und SimpleGeneticFSS erreichen die höchsten Genauigkeiten der Algorithmen (KimFSS mit weniger Attributen), FastGeneticFSS liegt etwas dahinter, signifikant schlechter ist jedoch nur der DashFSS-Algorithmus.

↓ besser als →	alle Attr.	S.G.FSS	F.G.FSS	KimFSS	DashFSS
alle Attribute	-	2 (2)	2(2)	6 (6)	6 (6)
SimpleGeneticFSS	6 (5)	-	3 (0)	6 (5)	7 (7)
FastGeneticFSS	6 (6)	3 (1)	-	6 (6)	8 (8)
KimFSS	2 (2)	2 (1)	2 (1)	-	4 (2)
DashFSS	2 (1)	1 (0)	0	4 (2)	-

Tabelle 6.9: Anzahl der Testreihen, bei denen der Algorithmus bei $PA_{k_{korr}}$ besser abgeschnitten hat als der Vergleichsalgorithmus, in Klammern die Anzahl der Testreihen, bei denen das Abschneiden signifikant (nach dem t-Test für zwei unabhängige Stichproben) besser war.

	D1	D2	D3	D4
SimpleGeneticFSS	14705.70	18109.27	31439.80	34415.43
FastGeneticFSS	3009.47	3119.97	6016.37	6114.37

Tabelle 6.10: Durchschnittliche Anzahl der k-Means-Iterationen für eine FSS

Tabelle 6.9 zeigt, wie die Algorithmen insgesamt im direkten Vergleich abschneiden. Man erkennt, dass der FastGeneticFSS- und der SimpleGeneticFSS-Algorithmus etwa gleich gut zu funktionieren scheinen. Dies war nicht von vornherein so zu erwarten. Einsparungen bei der Laufzeit werden normalerweise mit schlechteren Ergebnissen bezahlt und der SimpleGeneticFSS-Algorithmus sollte dadurch, dass er bereits von Anfang an auf fertige Clusterings zurückgreifen kann, einen Vorteil haben. Jedoch kann durch die Neuberechnung des Clusterings in jeder Runde ein Nachteil entstehen, dann nämlich, wenn k-Means durch eine zufällig ungünstige Wahl der Clusterzentren am Anfang zu einer eigentlich guten Attributauswahl ein schlechtes Clustering berechnet. Dann kann nämlich eine gute Lösung verloren gehen, weil das Clustering schlecht bewertet und das Individuum möglicherweise aus der Population entfernt wird. Dies kann beim FastGeneticFSS-Algorithmus nicht passieren, da immer die Clusterzentren des eine Runde zuvor berechneten Clusterings als Startwerte für die neue Berechnung eingesetzt werden. Dadurch bleibt ein einmal erreichtes gutes Ergebnis immer erhalten, solange es nicht durch hinreichend viele noch bessere Ergebnisse verdrängt wird.

Die Tabelle zeigt auch, dass die Vergleichsalgorithmen meist etwas schwächer abschneiden, wobei erwähnt werden muss, dass dies bei den synthetischen Daten deutlicher erkennbar ist als bei den realen.

In Tabelle 6.10 zeigt sich anhand der Anzahl der k-Means-Iterationen der deutliche Geschwindigkeitsvorteil des FastGeneticFSS-Algorithmus im Vergleich zum SimpleGeneticFSS-Algorithmus. Je nach Datensatz braucht der FastGeneticFSS-Algorithmus nur ein Fünftel bis ein Sechstel der k-Means-Iterationen.

Jedoch ist auch der SimpleGeneticFSS-Algorithmus noch deutlich schneller als der KimFSS-Algorithmus, der wegen seiner größeren Population und Rundenzahl etwa 20 mal so viele k-Means-Iterationen benötigt wie SimpleGeneticFSS. Der Einsatz einer Crossover-Operation würde bei diesem Algorithmus sicher die Suche beschleunigen.

Was die Laufzeit angeht, ist jedoch erwartungsgemäß der DashFSS-Algorithmus am besten. Als Filter mit einer einfachen sequenziellen Suche ist er den Wrappern da deutlich überlegen. Auch wenn die verwendeten Implementierungen der Algorithmen möglicherweise nicht überall die optimale Laufzeit erbringen und ein Vergleich der reinen Laufzeiten deshalb nicht sehr aussagekräftig ist, lässt sich dies mit Sicherheit sagen.

Kapitel 7

Schlussfolgerungen und Aussichten

Es wurde gezeigt, dass es möglich ist, einen von Haus aus eher langsamen genetischen Algorithmus durch einige einfache Modifikationen so anzupassen, dass sich seine Laufzeit deutlich verkürzt, ohne dass er dabei an Leistung einbüßt. Mit der beschriebenen Bewertungsfunktion, die in der Lage ist, unabhängig von Dimension und Clusteranzahl Cluster zu bewerten, erzielt er insgesamt bessere Ergebnisse als die Vergleichsalgorithmen.

Die Algorithmen lieferten allgemein bessere Ergebnisse auf den synthetischen Daten als auf den realen. Dafür kann es mehrere Gründe geben. Zum einen wurde bei den realen Daten zur Beurteilung der Clusterings das Klassenattribut herangezogen. Damit wurde vorausgesetzt, dass die Daten Cluster enthalten, die die Klassifizierung widerspiegeln. Das muss jedoch nicht unbedingt auch in deutlichem Maß der Fall sein. Es könnten z.B. wichtige Informationen fehlen, so dass die Cluster nur schwach ausgeprägt und schlecht separierbar sind. Vor allem aber ist es möglich, dass sich weitere Cluster in den Daten befinden, die andere Gruppierungen repäsentieren. Es könnten z.B. die einzelnen Klassen in mehrere Cluster unterteilt werden oder es könnte sein, dass sich Cluster bilden lassen, die einen gänzlich anderen Sachverhalt widerspiegeln und somit ganz anders zusammengesetzt sind als die Klassen. Die Algorithmen könnten dann durchaus ein gutes Clustering finden, welches aber eine geringe Vorhersagegenauigkeit bezüglich des Klassenattributs hätte. Zum anderen sind die Cluster in den synthetischen Daten recht gut separierbar, wenn man genau die relevanten Attribute gefunden hat, und man kann genau zwischen relevanten und irrelevanten Attributen unterscheiden. Bei realen Daten ist eine solche Unterscheidung in der Regel nicht klar zu treffen. Neben Attributen, von denen man klar sagen kann, dass sie relevant sind, und Attributen, die man ebenso klar als irrelevant identifizieren kann, wird es häufig eine Reihe von Attributen geben, die einen gewissen Beitrag, aber keinen entscheidenden Beitrag zum Clustering liefern können.

Feature Weighting könnte deshalb ein Ansatz sein, den es sich lohnt weiter zu verfolgen. Interessant wäre dabei z.B. eine Verbindung mit einem evolutionären Algorithmus. Statt eines Bitvektors könnte ein reeller Vektor zur Kodierung einer Lösung verwendet werden.

Feature Subset Selections sind Optimierungsprobleme. Um sie zufriedenstellend zu lösen, braucht man einen guten Suchalgorithmus sowie eine gute Bewertungsfunktion. Wie im Überblick der bisherigen Ansätze zu sehen ist, hat man bisher oft das Hauptaugenmerk auf die Bewertungsfunktion gerichtet und als Suchalgorithmus eine einfach Vorwärts- oder Rückwärtssuche verwendet.

Optimierungsprobleme sind jedoch ein gut erforschtes Gebiet. Eine Vielzahl von Suchalgorithmen steht zur Verfügung. Durch die Auswahl eines geeigneten, performanten Algorithmus könnte eine deutliche Leistungssteigerung im Vergleich zur Verwendung einer sequenziellen Suche erreicht werden. Evolutionäre Algorithmen bieten sich an, weil eine Kodierung der Lösungen als Bitvektor leicht zu realisieren ist. Vor allem für Wrapper muss jedoch die lange Laufzeit eines evolutionären Algorithmus berücksichtigt werden. Wie der hier vorgestellte Ansatz zeigt, ist es lohnenswert, den Suchalgorithmus an das gegebene Problem anzupassen. Schon kleine Modifikationen können zu deutlich besserem Laufzeitverhalten oder besseren Ergebnissen führen.

Dabei darf die Wichtigkeit der Bewertungsfunktion nicht vernachlässigt werden. Mit einer schlechten Bewertungsfunktion wird auch ein guter Suchalgorithmus niemals eine gute Attributauswahl finden. In der Bewertungsfunktion steckt noch ein großes Potential. Bisherige Entwicklungen funktionieren gut bei sehr gut separierten Clustern mit geringer Standardabweichung. Diese werden jedoch in der Realität eher selten auftreten. Bei Clustern mit großer Standardabweichung versagen aber die Bewertungsfunktionen häufig. Clusterings auf gleichverteilten Attributen erhalten hier oft kaum schlechtere Bewertungen als das eigentlich zu findende Clustering. Es gilt also, Bewertungsfunktionen zu entwickeln, die auch in solchen Situationen Attributmengen mit Clustern von solchen ohne Cluster unterscheiden können.

Literaturverzeichnis

[1] R. Agrawal, J. Gehrke, D. Gunopulos und P. Raghavan: Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications, in: Proceedings of the ACM SIGMOD Conference on Management of Data, 1998, S. 94 - 105

[2] R. Agrawal und R. Srikant: Fast Algorithms for Mining Association Rules, in: Proceedings of the 20th International Conference on Very Large Data Bases, 1994, S. 487 - 499

[3] J. Bilmes: A Gentle Tutorial on the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models, in: Technical Report, University of Berkeley, ICSI-TR-97-021, University of Berkeley, 1998

[4] C. L. Blake und C. J Merz, UCI Repository of Machine Learning Databases, http://www.ics.uci.edu/∼mlearn/MLRepository.html, 1998

[5] M. Dash, K. Choi, P. Scheuermann und H. Liu: Feature Selection for Clustering - A Filter Solution, in: Proceedings of IEEE International Conference on Data Mining (ICDM), 2002, S. 115 - 122

[6] M. Dash, H. Liu und J. Yao: Dimensionality Reduction for Unsupervised Data, in: Proceedings of 9th IEEE International Conference on Tools with Artificial Intelligence (ICTAI), 1997

[7] M. Dash und H. Liu: Feature Selection for Clustering, in: Proceedings of Fourth Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2000, S. 110 - 121

[8] D.L. Davies, D.W. Bouldin: A Cluster Separation Measure, in: IEEE Transactions on Pattern Analysis and Machine Intelligence 1 (2), 1979, S. 224 - 227

[9] F. Dellaert: The Expectation Maximization Algorithm, in: Technical Report GIT-GVU-02-20, College of Computing GVU Center, Georgia Institute of Technology, 2002

[10] M. Devaney und A. Ram: Efficient Feature Selection in Conceptual Clustering, in: Machine Learning: Proceedings of the Fourteenth International Conference, 1997, S. 92 - 97

[11] J. Dy und J. Brodley:, Feature Selection for Unsupervised Learning, in: Journal of Machin Learning Research 5 (2004), S. 845 - 889

[12] D. Fisher: Knowledge Acquisition via Incremental Conceptual Clustering, in: Machine Learning, 2, 1987, S. 139 - 172

[13] D. Fisher: Iterative Optimization and Simplification of Hierarchical Clusterings, in: Technical report, Vanderbilt University, TR CS-95-01, 1995

[14] M. A. Gluck und J. E. Corter: Information, Uncertainty, and the Utility of Categories, in: Proceedings of the Seventh Annual Conference of the Cognitive Science Society, 1985, S. 283 - 287

[15] D. E. Goldberg: Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Publishing Company Inc., 1989

[16] J. Hu, K. Seo, S. Li, Z. Fan, R. C. Rosenberg und E. D. Goodman: Structure Fitness Sharing (SFS) for Evolutionary Design by Genetic Programming, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2002, S. 780 - 787

[17] A. J. Jones: Genetic Algorithms and their Applications to the Design of Neural Networks, in: Neural Computing & Applications, 1(1), 1993, S. 32 - 45

[18] Y. Kim, W. N. Street und F. Menczer: Feature selection in Unsupervised Learning via Evolutionary Search, in: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2000, S. 365 - 369

[19] M. Law, M. Figueiredo und A. Jain: Feature Saliency in Unsupervised Learning, in: Tech. Rep., Dept. Computer Science and Eng., Michigan State Univ., 2002

[20] F. Menczer, M. Degeratu und W. N. Street: Effcient and Scalable Pareto Optimization by Evolutionary Local Selection Algorithms, in: Evolutionary Computation, 8(2), 2000, S. 223 - 247

[21] T. P. Minka: Expectation-Maximization as lower bound maximization, Tutorial veröffentlicht im Internet unter:
http://research.microsoft.com/∼minka/papers/em.html

[22] P. Mitra, C. A. Murthy und S. K. Pal: Unsupervised Feature Selection Using Feature Similarity, in: IEEE Transactions on Pattern Analysis and Machine Intelligence 24(3), 2002, S. 301 - 312

[23] D. Modha und W. Spangler: Feature Weighting in k-Means Clustering, in: Machine Learning, 52 (2003), S. 217 - 237

[24] M. Morita, R. Sabourin, F. Bortolozzi und C. Y. Suen: Unsupervised Feature Selection Using Multi-Objective Genetic Algorithms for Handwritten Word Recognition, in: Proceedings of the Seventh International Conference on Document Analysis and Recognition, 2003

[25] J. M. Peña, J. A. Lozano, P. Larrañaga und I. Inza: Dimensionality Reduction in Unsupervised Learning of Conditional Gaussian Networks, in: IEEE Transactions on Pattern Analysis and Machine Intelligence 23(6), 2001, S. 590 - 603

[26] K. Schwinger: Exponential Family, Maximum Likelihood, EM Algorithmus und Gaussian Mixture Models, 2003

[27] N. Søndberg-Madsen, C. Thomsen und J. M. Peña: Unsupervised Feature Subset Selection, in: Proceedings of the Workshop on Probabilistic Graphical Models for Classification (within ECML/PKDD 2003), S. 71 - 82

[28] N. Srinivas und K. Deb: Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms, in: Evolutionary Computation, 2(3), 1994, S. 221 - 248

[29] L. Talavera: Feature Selection as a Preprocessing Step for Hierarchical Clustering, in: Proceedings of the Sixteenth International Conference on Machine Learning, 1999, S. 389 - 397

[30] S. Vaithyanathan und B. Dom: Generalized Model Selection for Unsupervised Learning in High Dimensions, in: Proceedings of Advances in Neural Information Processing Systems (NIPS 1999), Vol. 12, MIT Press, 2000