Diplomarbeit

Effizientes Retrieval
und
Extraktion von Entscheidungswissen
mit k-d-Bäumen

Guido Derwand

April 1994

Betreuer:
Dipl.-Inform. Stefan Weß

Universität Kaiserslautern
Fachbereich Informatik
AG Künstliche Intelligenz – Expertensysteme

Prof. Dr. Michael M. Richter

SFB 314, Künstliche Intelligenz und wissensbasierte Systeme

Hiermit bestätige ich, daß ich die vorliegende Arbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe.

Kaiserslautern, den ____________________________ ____________________________

Zusammenfassung

In dieser Arbeit wird eine Kombinationsmöglichkeit von fallbasiertem und induktivem Schließen, basierend auf k-d- und Entscheidungsbäumen, entwickelt. Dabei wurde versucht, die Vorteile des induktiven Mechanismus, wie z. B. die sehr effiziente Klassifizierung und automatische Generierung, in den fallbasierten Mechanismus zu integrieren. Die Aufgabe zerfällt dabei in zwei Teilaufgaben, die im folgenden zusammengefaßt werden.

In fallbasierten Systemen ist eine der elementaren Aufgaben das ähnlichkeitsbasierte Retrieval von Fällen aus einer Fallbasis. Die in dieser Arbeit verwendeten Fälle haben die Form von Attribut-Werte-Paaren. Für diese steht im Zusammenhang mit den hier verwendeten Ähnlichkeitsfunktionen – gewichtete Summen von lokalen Ähnlichkeiten – noch kein effizienter Retrieval-Mechanismus bereit, welcher auch verschiedene Arten von Wertebereichen der Attribute unterstützt. Auf der einen Seite existieren Verfahren, welche Fälle direkt indizieren, während auf der anderen Seite Verfahren existieren, welche Fälle als Punkte in einem geometrischen Raum betrachten und dessen Struktur zum Retrieval ausnutzen. Da die erstgenannten Verfahren nur diskrete, begrenzte Attributwertebereiche unterstützen, während die letztgenannten nur geordnete Attributwertebereiche effizient unterstützen, sind Fälle, die sowohl ungeordnete Attribute mit begrenztem Wertebereich als auch geordnete Attribute mit eventuell unbegrenztem Wertebereich enthalten, durch keines der bekannten Verfahren vollständig handhabbar. Aus diesem Grund wird in dieser Arbeit der bekannte Retrieval-Mechanismus k-d-Baum so erweitert, daß mit ihm auch Fälle mit ungeordnetem Wertebereich handhabbar werden. Dabei werden die Informationen über die ungeordneten Wertebereiche in das Generierungs- und Retrieval-Verfahren integriert. Desweiteren wurde das Problem der unbekannten Werte, welche in der Praxis oft in Fällen auftreten können, angegangen. Wie gezeigt wird können auch Fälle mit teilweise unbekannten Attributwerten effizient in das Retrieval miteinbezogen werden, ohne daß sie, wie bei Entscheidungsbäumen, zu einem stark erhöhten Generierungsaufwand des k-d-Baums führen. Schließlich wird gezeigt, wie auch mehrere klassenspezifische Ähnlichkeitsmaße im k-d-Baum-Retrieval verwendbar sind.

Es werden mehrere Generierungsheuristiken für so erweiterte k-d-Bäume angegeben, die anhand eines realistischen Beispiels in ihrer Leistungsfähigkeit verglichen werden.

In einem zweiten Teil der Arbeit wird unter Verwendung von k-d-Bäumen ein Verfahren zur automatischen Generierung von leistungsfähigen Ähnlichkeitsmaßen vorgestellt, welche z. B. zur Klassen- oder Konzepterkennung mit Ähnlichkeitsmaßen eingesetzt werden können. Diese Verfahren beruhen u. a. auf einer Analyse der Inter- und Intra-Klassen-Ähnlichkeit und setzen die so erzielten Ergebnisse in ein Verfahren zur Generierung von Gewichten der einzelnen Attribute um. Da oft jedoch die Leistungsfähigkeit von gewichteten Summen als Ähnlichkeitsmaß nicht ausreicht, weil eine Klasse beispielsweise nur mit Hilfe eines disjunktiven Konzeptes beschreibbar ist, wird desweiteren ein Verfahren vorgestellt, welches es ermöglicht, solche disjunktiv beschriebenen Konzepte zu erkennen und in das Ähnlichkeitsmaß einzubetten. Dazu werden nicht nur mehrere Gewichtsvektoren für ein klassenspezifisches Ähnlichkeitsmaß generiert, sondern auch die verwendeten lokalen, attributspezifischen Ähnlichkeitsmaße werden bei diesem Ansatz optimiert. Die so generierten Ähnlichkeitsmaße können an die Stelle von normalen Konzepterkennungsfunktionen treten und ermöglichen somit eine unscharfe Zugehörigkeitsaussage eines Falls zu einem Konzept. Desweiteren ist damit auch eine sinnvolle Einbindung von unbekannten Attributwerten möglich. Die so generierten Ähnlichkeismaße sind natürlich auch mit den erweiterten k-d-Bäumen verwendbar.

Abschließend werden in dieser Arbeit noch einige, diesem Ansatz verwandte Ansätze vorgestellt und analysiert.

Der vorgestellte Ansatz wurde vollständig in Smalltalk implementiert und wird innerhalb des Inreca-Projektes verwendet.

Inhaltsverzeichnis

Kapitel 1

Einführung

Die in dieser Diplomarbeit entwickelten und implementierten Methoden werden im Rahmen des INRECA[1]-Projektes verwendet. Der Inhalt des Projektes sowie die Aufgaben und der Aufbau dieser Arbeit werden im folgenden beschrieben.

1.1 Motivation : Das INRECA-Projekt

Der Inhalt des INRECA-Projektes ist die Kombination und Integration von induktivem und fallbasiertem Schließen. Induktion ist schon ein fast klassisches Mittel zur Extraktion von Entscheidungswissen aus Beispielfällen, während fallbasiertes Schließen es ermöglicht, Bezüge zwischen einem aktuellen Problem und Vergangenheitserfahrungen herzustellen.

Ziel des Projektes ist die Bereitstellung von Werkzeugen und Methoden für die Entwicklung, Validierung und Wartung von entscheidungsunterstützenden Systemen.

Das INRECA-Projekt wird als ESPRIT-Projekt durch die Europäische Union gefördert und besitzt neben der Universität Kaiserslautern Partner in drei europäischen Ländern.

1.2 Aufgaben dieser Arbeit

In dieser Arbeit wird der fallbasierte Teil des INRECA-Systems bearbeitet, in welchem ein Modul entwickelt werden soll, welches es ermöglicht, zu einer gegebenen Problemsituation einen möglichst ähnlichen Fall aus einem durch Fälle repräsentierten Erfahrungswissen zu extrahieren. Das bisher verwendete Modul baute auf der Arbeit von F. Goebel [14] auf und verwendete k-d-Bäume zum effizienten Fallretrieval. Dieses Modul setzte selbst auf einem für das System CABPLAN entwickelten k-d-Baum-Retrieval von H. Öchsner [5] auf.

Auch in dieser Arbeit sollen k-d-Bäume zum Fallretrieval verwendet werden. Diese verwenden bisher nur geordnete Attribute und haben daher den Nachteil, daß sie nur einen kleinen Teil der in den Fällen enthaltenen Information zum Retrieval benutzen und deshalb unnötig viel Zeit zum Retrieval benötigen. In dieser Arbeit wird aufgezeigt, wie auch Werte ungeordneter Attribute sowie Attribute mit unbekanntem Wert in den k-d-Baum integriert werden können und zur Effizienzsteigerung beim Retrieval verwendet werden können. Dazu müssen zum einen die Erzeugungsheuristiken wie auch die Retrieval-Algorithmen erweitert bzw. komplett erneuert werden. Desweiteren muß aufgezeigt werden, wie mehrere klassenspezifische Ähnlichkeitsmaße in das k-d-Baum-Retrieval eingebaut werden können [10].

Da INRECA sich mit der Integration von induktivem und fallbasiertem Schließen beschäftigt, wird desweiteren ein Ansatz zur Integration beider Mechanismen vorgestellt. Er beruht darauf, aus Entscheidungsbaum-ähnlichen k-d-Bäumen Wissen über die Ähnlichkeit von Fällen zu extrahieren und in das verwendete Ähnlichkeitsmaß einzubetten. Dies ermöglicht es, in INRECA bisher nur

[1] Induction and reasoning from cases

durch Entscheidungsbäume repräsentierbares Wissen in den Prozeß des fallbasierten Schließens miteinzubauen. Dieser Teil der Arbeit baut sehr stark auf Erfahrungen mit dem an der Universität Kaiserslautern entwickelten System PATDEX/2 [11, 10] auf, einem System zur fallbasierten Diagnostik.

Die vorgestellten Konzepte und Methoden sind vollständig in SMALLTALK implementiert worden und werden im INRECA-Projekt verwendet.

1.3 Aufbau der Arbeit

In Kapitel 2 wird eine kurze Einführung in das fallbasierte Schließen und die verwendeten Ähnlichkeitsmaße gegeben. Daran schließt sich eine Zusammenfassung der bisher verwendeten k-d-Bäume an. Abgeschlossen wird das Kapitel durch eine kurze Einführung in den Bereich der Entscheidungsbäume.

In Kapitel 3 wird der Begriff der k-d-Bäume so erweitert, daß ungeordnete und unbekannte Attribute effizient in das k-d-Baum-Retrieval miteingebaut werden können. Dazu werden bekannte Erzeugungsverfahren erweitert und auch vollständig neue Verfahren entwickelt. Desweiteren wird das Retrieval-Verfahren so erweitert, daß die Erweiterung zum effizienten Retrieval genutzt werden kann und auch mehrere klassenspezifische Ähnlichkeitsmaße im Retrieval verwendet werden können.

In Kapitel 4 wird ein Verfahren zur automatischen Erzeugung von Ähnlichkeitsmaßen vorgestellt, das es ermöglicht, Relevanzen der einzelnen Attribute zu erkennen und in das verwendete Ähnlichkeitsmaß einzubetten. Desweiteren wird ein Verfahren zur Extraktion von Entscheidungswissen und dessen Einbettung in das verwendete Ähnlichkeitsmaß vorgestellt.

In Kapitel 5 wird eine Zusammenfassung mit Ausblick gegeben.

Letztlich werden in Kapitel 6 einige Arbeiten, die ähnliche Aufgabenstellungen bearbeiten, analysiert und bewertet.

Kapitel 2

Grundlagen

2.1 Fallbasiertes Schließen

Zur Lösung komplexer Aufgaben verwendet man in Expertensystemen explizit angegebenes Wissen gemeinsam mit Operationen zu dessen Anwendung. Dieses Wissen muß möglichst vollständig und korrekt aufgenommen werden und dem operationalen Teil des Expertensystems in adäquater Weise zur Verfügung gestellt werden. Als Formalismen der Wissensrepräsentation werden deshalb oft Regeln und verwandte Konstrukte verwendet, da in vielen Aufgabenbereichen das Expertenwissen schon in dieser Form vorliegt. Neben dieser gängigen Repräsentation liegt jedoch oft das verwendbare Wissen nur als Erfahrungswissen, sogenanntes episodisches Wisssen, vor, d. h. ausgehend von der Beschreibung eines gelösten Problems ist noch die passende Problemlösung, eventuell auch noch der Lösungsfindungsweg, bekannt. Wir nennen das Paar (Tripel) bestehend aus Problem und Lösung (sowie Lösungsfindungsweg) auch einen *Fall*. Diese Form des Wissens in eine regelähnliche, effizient handhabbare Form zu konvertieren ist im allgemeinen mit dem Verlust an Detailwissen behaftet und deshalb oft nicht tolerierbar. Deshalb liegt die Forderung nahe, vorhandenes Wissen möglichst ohne Detailverluste in die Wissenbasis eines Expertensystems zu integrieren, was der Forderung nach Korrektheit Rechnung trägt, aber die gewünschte Vollständigkeit nahezu unmöglich werden läßt, da im Prinzip jede nur mögliche Problembeschreibung inklusive Lösungswissen in der Wissensbasis vorhanden sein müßte. Der Ausweg aus diesem scheinbaren Dilemma ist die Erkenntnis, daß die Kenntnis von einigen ausgezeichneten Fällen genügt, wenn man durch Anpassung von deren Lösungen an *ähnliche* Problemstellungen Lösungen für eben diese generieren kann. Ein Teil des Wissens wird also als *Ähnlichkeitsmaß* und ein weiterer Teil als *Adaptionswissen* kodiert und kann somit einen Teil der Fälle ersetzen. Ähnlichkeitsmaße können während der Anwendung des so strukturierten Expertensystems erlernt werden (ein interessanter Ansatz findet sich in [10]). Adaptionswissen ist im allgemeinen einfacher strukturiert als Problemlösungswissen und wird in einigen Anwendungsfällen, wie beispielsweise der Klassifikation, gar nicht erst benötigt. Läßt man nun die Erweiterung der Fallmenge, auch *Fallbasis* genannt, um neu erzeugte Fälle zu, so kann man einen Großteil des benötigten Wissens erlernen und erhält ein dynamisches Expertensystem, welches ohne den Aufwand der Bereitstellung von vollständigen Wissen funktionsfähig ist.
Der Mechanismus des *Fallbasierten Schließens (Case-Based Reasoning, CBR)* läßt sich als Prozeßmodell formulieren, wobei hier das Modell von Rissland (siehe [7] und [1]) vorgestellt werden soll:

Eingabe: Aktuelle Problembeschreibung P, Fallbasis FB mit den Fällen f_i, $FB = \{f_i | i \in I\}$
Ausgabe: Lösung S für P

1. **Bereitstellung:** Es wird eine Vorauswahl geeigneter Fälle aus FB getroffen, die zumeist auf Eigenschaften von P beruht, welche sich in der Vergangenheit für die Lösungsfindung als relevant herausgestellt haben.

2. **Auswahl:** Da der vorhergehende Schritt nur eine Vorauswahl getroffen hat, muß in diesem Schritt die Menge der Fälle auf eine kleine Anzahl von guten Kandidaten $f_{i_1}, \ldots, f_{i_k}$ reduziert werden, um die Anzahl der benötigten Adaptionsvorgänge einzuschränken.

3. **Anpassung & Interpretation:** Die Lösungen von $f_{i_1}, \ldots, f_{i_k}$ werden an P angepasst, d. h. es werden potentielle Lösungen $S'_{i_1}, \ldots, S'_{i_k}$ generiert. Dieser Schritt kann auf Adaptions- oder problemspezifischem Wissen beruhen.

4. **Test & Kritik:** $S'_{i_1}, \ldots, S'_{i_k}$ werden *intern* vom System auf Realisierbarkeit und Erfolg geprüft. Dies kann die Lösungen auf $S'_{j_1}, \ldots, S'_{j_n}$ einschränken ($\{j_1, \ldots, j_n\} \subseteq \{i_1, \ldots, i_k\}$), oder es wird schon die beste Problemlösung S nach internen Kriterien ausgewählt.

5. **Ergebnisüberprüfung:** $S'_{j_1}, \ldots, S'_{j_n}$ (bzw. S) wird nun *extern* in der *realen* Welt überprüft. Dazu wird das positive oder negative Feedback vom Benutzer benutzt und es wird die beste Problemlösung S ausgewählt, falls dies nicht schon im vorhergehenden Schritt geschehen ist.

6. **Lernphase:** Das Feedback wird ausgewertet und kann in einer Änderung des Systems münden. Dabei kann beispielsweise der neu erzeugte Fall (P, S) in FB aufgenommen werden oder das verwendete Ähnlichkeitsmaß kann geändert werden. Auch die internen Beurteilungskriterien wie auch das Adaptionswissen könnten angepaßt werden.

Die oben unter 1 und 2 aufgeführten Schritte sind auch unter dem Namen *MAC/FAC*[1] (siehe [13]) bekannt, und werden zusammengefaßt als *Retrieval* bezeichnet, wobei in dieser Arbeit zusätzlich zu diesen beiden Schritten noch Schritt 6 betrachtet werden soll.

Die Retrieval-Aufgabe läßt sich auch als *nearest-neighbour-problem* interpretieren, wenn man die Lösungen der Fälle aus der Fallbasis während des Retrievals ignoriert und das zu lösende Problem dann ebenfalls als Fall betrachtet, welcher dann als *Anfragefall (query case)* bezeichnet wird. Rechnet man dann noch das zu verwendende Ähnlichkeitsmaß in ein Distanzmaß um, so kann man die Suche nach den n ähnlichsten Problemen auf die Suche nach den n nächsten Nachbarfällen zum Anfragefall zurückführen.

Die Effizienz eines fallbasierten Systems hängt stark von der Komplexität des Retrievals ab, welche im schlechtesten Fall $O(m)$ ist, wenn m die Anzahl der Fälle in der Fallbasis ist. Das ist auf den ersten Blick noch ein respektabeler Zustand, ist aber in dieser Größenordnung nicht akzeptabel, da erfahrungsgemäß eine sehr große Anzahl von Fällen benötigt wird, um gute Lösungen zu erzielen. Deshalb wird auf vielerlei Weise versucht, die Komplexität des Retrievals zu vermindern, wobei hier zwei grundsätzlich verschiedene Ansätze verfolgt werden:

- Parallele Hardware-Architekturen mit je einem Prozessorelement *(PE)* für eine kleine Anzahl von Fällen. Diese verringern zwar nicht die Komplexität, verkleinern aber je nach Anzahl der PEs den konstanten Faktor.

- Datenstrukturen, die den Zugriff zumindest mit *mittlerer* Komplexität $O(\log m)$ erlauben.

Die in dieser Arbeit vorgestellten Strukturen der k-d-Bäume sowie der Entscheidungsbäume gehören zum zweiten Ansatz und werden in ihrer bisher verwendeten und implementierten Form (siehe [14]) im Anschluß vorgestellt. Desweiteren werden Verbesserungsmöglichkeiten für k-d-Bäume aufgezeigt und es wird der Versuch der Integration beider Ansätze unternommen. Es sollen nun einige Begriffe festgelegt werden:

Definition 2.1 (Fall) *Seien $A_1, \ldots, A_k$ Attribute und $W_1, \ldots, W_k$ die zugehörigen Wertebereiche. Wir setzen $\hat{W}_i := W_i \cup \{\perp\}$ und interpretieren $\perp$ als "unbekannten Wert". Weiter sei $\hat{\mathcal{W}} := \hat{W}_1 \times \ldots \times \hat{W}_k$ der zugehörige Datenraum und $\mathcal{S}$ die Menge aller Lösungen, sowie $\hat{\mathcal{S}}$ analog zu oben. Dann heißt $F \in \hat{\mathcal{W}} \times \hat{\mathcal{S}}$ ein Fall. Für F schreiben wir auch in Komponentenschreibweise $<(w_1, \ldots, w_k), s>$.*

[1] many are choosen, but few are called

Ein Fall ist also das Paar aus Problembeschreibung und Lösung, wobei die Problembeschreibung ein Attributvektor mit eventuell unbekannten Attributen ist, und die Lösung eventuell auch unbekannt sein kann. Nun kann der Begriff der Fallbasis definiert werden:

Definition 2.2 (Fallbasis) *$FB \subseteq \hat{\mathcal{W}} \times \mathcal{S}$ heißt* <u>*Fallbasis*</u>.

Hierbei ist zu beachten, daß für Fälle aus der Fallbasis eine unbekannte Lösung ausgeschlossen wird. Wie man sieht reicht eine Menge als Fallbasisstruktur aus. Dies würde jedoch eine lineare Suche oder eine parallele Suche mit Hardware-Unterstützung erfordern und ist dementsprechend langsam bzw. aufwendig. Deshalb werden in 2.3 Algorithmen und Datenstrukturen vorgestellt werden, die einen schnellen Zugriff auf die n nächsten Nachbarn eines gegebenen Anfragefalles erlauben. Zuvor soll allerdings der Begriff der Ähnlichkeit genauer festgelegt werden.

2.2 Ähnlichkeit und Ähnlichkeitsmaße

Zur Lösung des nearest-neighbour-Problems wird eine Formalisierung des Begriffs des nächsten Nachbarn benötigt. Dazu kann man die Relation R verwenden, welche mit $R(x, y, u, v)$ festlegt, daß *x ähnlicher zu y ist als u zu v*. Die gewünschte Relation S mit $S(x, y, z)$ gdw *x ist ähnlicher zu y als zu z* ergibt sich dann mit

$$S(x, y, z) \Leftrightarrow R(x, y, x, z) \tag{2.1}$$

Mit Hilfe der Relation S kann man nun einen zu einem Fall F_q *ähnlichsten Fall* $f \in FB$ bestimmen. Dieser genügt dann der Bedingung $\forall f' \in FB : S(F_q, f, f')$. Die Relation $S(x, y, z)$ liefert also eine Partialordnung $\succeq$ auf Paaren von Fällen:

$$S(x, y, z) \Leftrightarrow R(x, y, x, z) \Leftrightarrow (x, y) \succeq (x, z) \Leftrightarrow y \succeq_x z \tag{2.2}$$

Wir nennen solch eine Ordnung *Präferenzordnung* [18, 12]. Wie man sieht induziert jeder Fall x eine spezifische Präferenzordnung $\succeq_x$, welche als eine Ordnung der Nachbarn bezüglich ihrer Ähnlichkeit zu x angesehen werden kann. Dies legt die Vorstellung einer numerischen Fassung von "mehr oder weniger ähnlich (unähnlich)“ nahe. Diese werden als Ähnlichkeitsmaße sim bzw. Distanzfunktionen d bezeichnet. Diese induzieren durch $R_{sim}(x, y, u, v) \Leftrightarrow sim(x, y) \geq sim(u, v)$ bzw. $R_d(x, y, u, v) \Leftrightarrow d(x, y) \leq d(u, v)$ eine Ähnlichkeitsrelation. sim und d heißen *kompatibel*, wenn $R_{sim} \equiv R_d$.

Ähnlichkeitsmaße dienen also zur *Erkennung* und *Bewertung* von Gemeinsamkeiten zweier Fälle und können auch als eine Art *geometrisches Modell* der Ähnlichkeit aufgefaßt werden. Mit Hilfe von *Voronoi-Diagrammen* (s. Kapitel 6) läßt sich dieses Modell sehr anschaulich visualisieren.

Wir unterscheiden zwischen zwei verschiedenen Arten von Ähnlichkeitsmaßen, den *globalen* und den *lokalen* Ähnlichkeitsmaßen.

2.2.1 Lokale Ähnlichkeitsmaße

Lokale Ähnlichkeitsmaße sind spezielle Ähnlichkeitsmaße, welche nur Werteausprägungen jeweils *eines* Attributs A_i miteinander vergleichen können.

Definition 2.3 (Lokales Ähnlichkeitsmaß) *Die Funktion sim_i heißt* <u>*lokales Ähnlichkeitsmaß*</u> *für Attribut A_i, wenn sie wie folgt festgelegt ist:*

1. $sim_i : \hat{W}_i \times \hat{W}_i \longrightarrow [0, 1]$
2. $\forall w \in W_i : sim_i(w, w) = 1$

Wichtig an obiger Definition ist noch, daß für den unbekannten Wert $\perp$ nicht $sim_i(\perp, \perp) = 1$ verlangt wird.

Lokale Ähnlichkeitsmaße bieten nach obiger Definition also noch sehr viel Freiraum für ihre Festlegung. So könnte für $W_i = \Re$ das Maß sim_i auf die folgenden, sehr verschiedenen Weisen festgelegt werden:

$$sim_i(u,v) = \begin{cases} 1 & \text{falls } u = v \\ 0 & \text{sonst} \end{cases}$$

oder auch

$$sim_i(u,v) = 1 - \frac{|u-v|}{1+|u-v|}$$

Das zweite Maß schöpft also den ganzen möglichen Ergebnisbereich aus, während das erste nur einfaches Matching realisiert.

Ist ein Attribut A_i geordnet, d. h. existiert auf dem Wertebereich W_i eine Totalordnung $<_i$, so wird bei der Verwendung von sim_i und $<_i$ noch folgende **Verträglichkeitsbedingung** verlangt:

$$\boxed{\textbf{Forderung 1: } \forall u,v,w \in W_i : u <_i v <_i w \Rightarrow sim_i(u,v) \geq sim_i(u,w)}$$

Dies heißt nichts anderes, als daß, wenn man mit $d_i(u,v) = 1 - sim_i(u,v)$ eine Abstandfunktion definiert, der Abstand von u und v in einer aufsteigenden Kette $u <_i \ldots <_i v$ monoton wächst. Von dieser Eigenschaft wird bei den später vorgestellten k-d-Bäumen (implizit) Gebrauch gemacht, in welchen eine Abgrenzung mit $<_i$ umgesetzt wird in eine Abschätzung für sim_i.

2.2.2 Globale Ähnlichkeitsmaße

Mit Hilfe globaler Ähnlichkeitsmaße werden nicht nur einzelne Attributwerte, sondern ganze Fälle miteinander verglichen.

Definition 2.4 (Globales Ähnlichkeitsmaß) *Die Funktion sim heißt globales Ähnlichkeitsmaß, wenn sie wie folgt festgelegt ist:*

1. $sim : (\hat{\mathcal{W}} \times \hat{\mathcal{S}})^2 \longrightarrow [0,1]$
2. $\forall f \in \mathcal{W} \times \hat{\mathcal{S}} : sim(f,f) = 1$

Für Fälle f mit unbekannten Attributwerten wird also nicht $sim(f,f)$ verlangt. Dies liegt an der analogen Bedingung bei den lokalen Ähnlichkeitsmaßen.

Globale Ähnlichkeitsmaße lassen sich für Fälle im $\Re^k$ durch Umrechnung einer p-Norm in ein Ähnlichkeitsmaß finden. Im folgenden wird jedoch eine etwas andere Art von Maßen verwendet: Wir konstruieren mit Hilfe der lokalen Ähnlichkeitsmaße sim_i für jedes Attribut und sog. *Gewichten* $\omega_i \in [0,1]$ ein globales Maß:

$$sim(<(w_1^{(1)},\ldots,w_k^{(1)}),s>,<(w_1^{(2)},\ldots,w_k^{(2)}),s>) = \sum_{i=1}^{k} \omega_i sim_i(w_i^{(1)},w_i^{(2)}) \tag{2.3}$$

Desweiteren wird verlangt, daß

$$\omega_i \in [0,1] \tag{2.4}$$

und

$$\sum_{i=1}^{k} \omega_i = 1 \tag{2.5}$$

was die Bedingung $sim(f,f) = 1$ aus obiger Definition sichert. Zur Konstruktion eines globalen Ähnlichkeitsmaßes kann man sich also auf die Bestimmung der lokalen Maße und auf die Festlegung der Gewichte zurückziehen, wobei letztere als eine Art "Relevanz" für die Ähnlichkeit interpretiert werden können.

Für globale Ähnlichkeitsmaße fordern wir noch die **Monotoniebedingung**, d. h. wenn die drei Fälle $f_j =<(w_1^{(j)},\ldots,w_k^{(j)}),s>$ für $j = 1,2,3$ gegeben sind, muß gelten

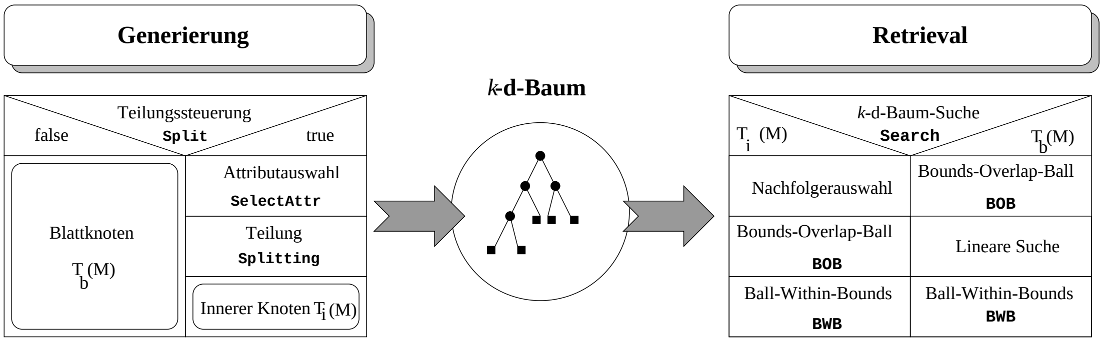
Forderung 2: $(\forall i \in \{1,\ldots,k\} : sim_i(w_i^{(1)}, w_i^{(2)}) \geq sim_i(w_i^{(1)}, w_i^{(3)}))$ $\Rightarrow sim(f_1, f_2) \geq sim(f_1, f_3)$

Die Monotoniebedingung sichert, daß sich eine *lokale* Verbesserung oder Verschlechterung der Ähnlichkeit auch als *globale* Verbesserung bzw. Verschlechterung der Ähnlichkeit auswirkt. Dies ergibt die Möglichkeit, sich bei der Abschätzung von globalen Ähnlichkeiten auf die Betrachtung lokaler Ähnlichkeiten zurückziehen zu können. Da später versucht wird, mit Hilfe lokaler Veränderungen Erkenntnisse über globale Veränderungen zu gewinnen, ist diese Forderung unerläßlich.

Im allgemeinen erfüllt aber nicht jedes Ähnlichkeitsmaß die Monotoniebedingung. Es wird also in dieser Arbeit nur eine bestimmte Klasse von Ähnlichkeitsmaßen verwendet. Wenn man globale Ähnlichkeitsmaße mit Hilfe von Gewichten und lokalen Maßen wie oben gezeigt konstruiert, so ist die Monotoniebedingung, wie man leicht sieht, garantiert.

2.3 k-d-Bäume

Der k-d-Baum [5, 9, 15] ist ein $\underline{k}$-$\underline{\text{d}}$imensionaler binärer Suchbaum, d. h. es wird nicht nur die Suche nach einem Primärschlüssel unterstützt, sondern es ist vielmehr die Suche nach bis zu k Suchschlüsseln gleichzeitig möglich[2]. Desweiteren ist nicht nur die Suche nach exakt zum Anfragefall passenden Fällen aus FB möglich, sondern es können auch die bezüglich des verwendeten Ähnlichkeitsmaß n nächsten Nachbarn gesucht werden.



Der Begriff "k-d-Baum" untergliedert sich in den Bereich der Erzeugung *(Generierung)* und den Bereich der Verwendung *(Retrieval)* des Baumes. Diese werden nun im folgenden beschrieben.

2.3.1 Generierung

Es folgt nun die zugehörige Definition, wobei diese etwas allgemeiner als in den oben angegebenen Referenzen gehalten ist, um die noch folgenden Erweiterungen des Begriffs "k-d-Baum" einbetten zu können. In Abbildung 2.1 ist der Zusammenhang der Definitionen aus diesem und dem folgenden Kapitel angegeben.

Definition 2.5 (Allgemeiner k-d-Baum) *Sei $M \subseteq FB$ eine Menge Fällen und* `Split`(M) *ein Prädikat, das genau dann gilt, wenn eine Unterteilung von M erfolgen soll.* `SelectAttr`(M) *sei eine Funktion, welche das zu verwendende Diskriminatorattribut A_j auswählt, also die Dimension des Datenraums, in welcher M partitioniert werden soll, d. h. es gelte* `SelectAttr`$(M) = A_j$.

Im folgenden bezeichnet $T_b(M)$ einen Blattknoten (diese werden auch als "Bucket" bezeichnet), welcher alle Fälle von M beinhaltet, und $T_i(M)$ einen inneren Knoten, welcher Verweise auf Nachfolgeknoten sowie auf A_j enthält. Wir benötigen noch eine Funktion `Splitting`$_j(M)$, *welche M in p_j Teilmengen ($p_j \leq |M|$) partitioniert, wobei p_j nur vom verwendeten Diskriminatorattribut A_j abhängt. Es heißt dann $T(M)$ allgemeiner k-d-Baum, wenn gilt:*

[2] Im weiteren bezeichnet k die Anzahl der Attribute eines Falls, die hier auch die Funktion von Schlüsseln haben

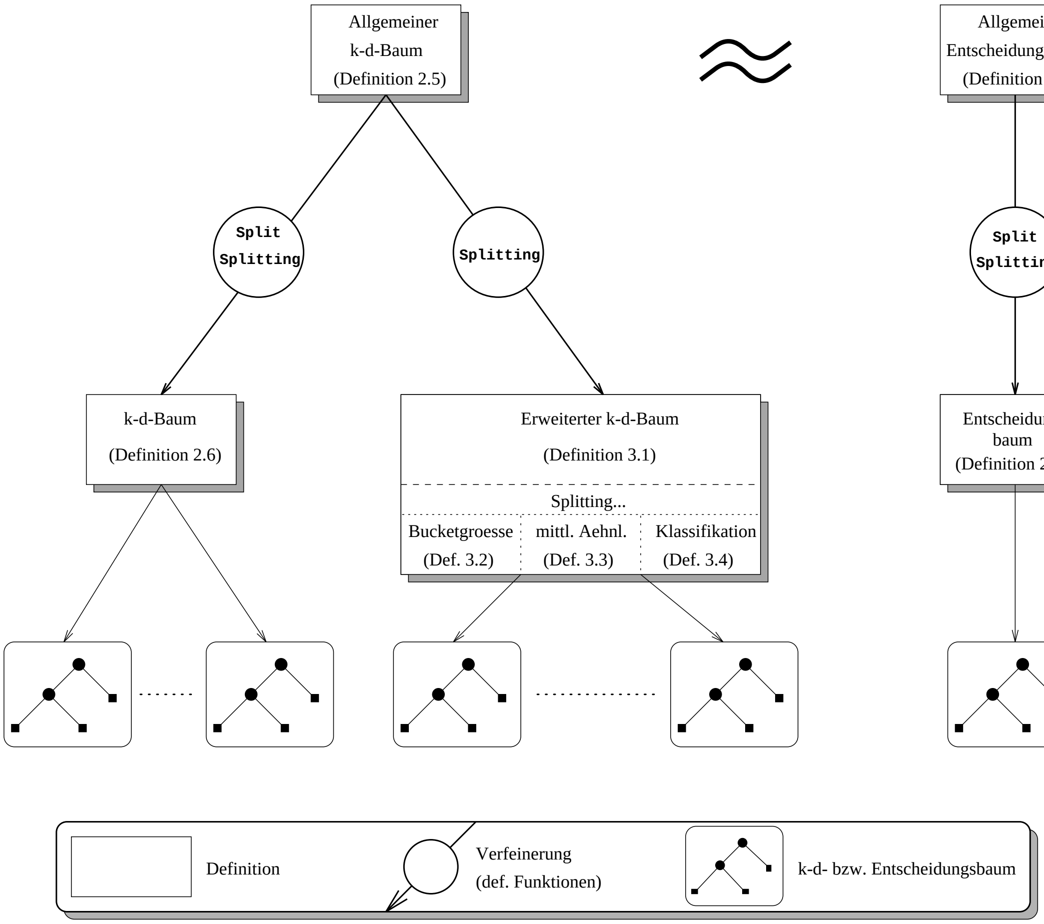


Abbildung 2.1: Zusammenhang der Definitionen

- *Falls* $\neg$`Split`(M)*:* $T(M) = T_b(M)$
- *Falls* `Split`(M)*:* $T(M) = T_i(M)$ *mit*
  - $T_i(M)$ *enthält das verwendete Diskriminatorattribut* A_j
  - $\texttt{Splitting}_j(M) = (M_1, \ldots, M_{p_j})$*, wobei* $M = M_1 \cup \ldots \cup M_{p_j}$*, sowie* $M_u \cap M_v = \emptyset$ *und* $M_u|_{A_j} \cap M_v|_{A_j} = \emptyset$ *(d. h.* M_u *und* M_v *enthalten nur verschiedene Ausprägungen von Attribut* A_j*) für* $1 \leq u < v \leq p_j$*, sowie* $\forall u : M_u \neq M$
  - $T_i(M)$ *verweist auf die Unterbäume* $T(M_1), \ldots, T(M_{p_j})$

Anhand obiger Definition ist noch nicht zu erkennen, daß es sich bei k-d-Bäumen um binäre Bäume handelt, denn diese Eigenschaft ist in den Funktionen `Splitting` verborgen. Sie wird bei den Erweiterungen in Kapitel 3 aufgegeben, ist aber in der ursprünglichen Definition [5, 9, 15] noch enthalten. In dieser existieren noch mehrere Freiheitsgrade zur Berechnung eines k-d-Baumes (s. a. Def. 2.6). Diese sind im einzelnen:

1. Die Bucketgröße b
2. Die Funktion `SelectAttr` zur Auswahl des in jedem Partitionierungsschritt verwendeten Diskriminatorattributs A_j

3. Die Funktion `SelectPartitionValue` zur Auswahl des Partitionswertes P

Für 1 gibt es keine allgemeingültigen Größenvorschläge. Kleine Werte für b haben den Nachteil, daß der k-d-Baum sehr groß werden kann, während große Werte dazu führen, daß bei der Suche wenig Fälle ausgespart werden können, da beim Durchsuchen eines Buckets immer alle darin enthaltenen Fälle betrachtet werden müssen. Später werden weitere Möglichkeiten der Definition des Prädikats `Split` vorgestellt, die Buckets unterschiedlicher Größen erzeugen können. Für 2 existieren mehrere Methoden, die nach der folgenden Definition vorgestellt werden. Doch nun zuerst zur "klassischen" Definition von k-d-Bäumen:

Definition 2.6 (k-d-Baum) *Ein allgemeiner k-d-Baum $T(M)$ zu $M \subseteq FB$ heißt $\underline{k\text{-}d\text{-}Baum}$ zur Bucketgröße b, wenn die Datenräume W_i $(1 \leq i \leq k)$ geordnet sind*[3]*, und das Prädikat* `Split` *sowie die Funktionen* `Splitting` *wie folgt festgelegt sind, wobei* `SelectPartitionValue` *eine neu eingeführte Funktion ist:*

- `Split`$(M) \Leftrightarrow |M| > b$
- `SelectPartitionValue`$(M, A_j) = P$ *mit* $P \in (\Theta_j - \{\max(\Theta_j)\})$, *wobei*

$$\Theta_j := \{w_j | <(w_1, \ldots, w_k), s> \in M, w_j \neq \perp\}$$

  P wird auch als "Partitionswert" bezeichnet. A_j ist das mit `SelectAttr` *ausgewählte Diskriminatorattribut.*

- `Splitting`$_j(M) = (M_{\leq}, M_{>})$ *mit*
  - $M_{\leq} = \{<(w_1, \ldots, w_k), s> \in M | w_j \leq P\}$
  - $M_{>} = \{<(w_1, \ldots, w_k), s> \in M | w_j > P\}$

Obige Definition ist also eine Erweiterung von Def. 2.5. In ihr werden, wie auch in Abb. 2.1 zu ersehen ist, die noch offenen Definitionen der Funktionen `Split` und `Splitting` angegeben. Letztere partitioniert die Fallbasis in zwei disjunkte Teile mit Hilfe der $<$-Relation.

Wahl des Diskriminatorattributs (`SelectAttr`)

Die richtige Auswahl des Diskriminatorattributs ist ebenso wie die Wahl des Partitionswertes von entscheidender Bedeutung für den Effizienzgewinn, welcher durch die Benutzung eines k-d-Baumes erzielt werden soll. Die Intention muß also hier sein, ein Attribut auszuwählen, welches bezüglich des zu verwendenden Partitionswertes die aktuelle Fallmenge möglichst gut trennt, d. h. daß die so durch Teilung entstehenden Teilmengen der Fallmenge bezüglich des verwendeten Ähnlichkeitsmaßes innerhalb einer Teilmenge die enthaltenen Fälle große Ähnlichkeiten aufweisen und Fälle verschiedener Teilmengen relativ geringe Ähnlichkeiten aufweisen. Ziel ist es also, untereinander ähnliche Fälle innerhalb eines möglichst kleinen gemeinsamen Teilbaumes unterzubringen, um eine Anfrage nach den n nächsten Nachbarn eines Falles in einem möglichst kleinen Teilbaum erfüllen zu können, wobei sich "klein" vor allen Dingen auf die Anzahl der zu durchsuchenden Fälle bezieht. Hier gibt es verschiedene Ansätze die entweder

- sich auf die Untersuchung der in der aktuellen Fallmenge enthaltenen Ausprägungen jeweils *eines* Attributs beschränken und die dadurch erzielten Bewertungen zur Auswahl eines Attributes der k untersuchten Attribute verwenden, oder aber
- zwar anhand jeweils eines Attributes, also einer Dimension, die aktuelle Fallmenge teilen, aber die so entstandene Teilung unter Berücksichtigung *aller* Attribute bewerten, und diese Bewertung zur Auswahl eines Attributes verwenden.

[3] Dies ist eine erhebliche Einschränkung, die aber in späteren Kapiteln wieder aufgehoben wird

In die erste Kategorie der Ansätze passen Maße, welche die Streuung der Attributsausprägungen erfassen können. Da aber sowohl qualitative wie auch quantitative Werte möglich sind, wird in [5] der sogenannte *Interquartilsabstand* (siehe auch [16]) verwendet, welcher für beide Arten von Werteausprägungen verwendet werden kann. Zur Berechnung dieses Maßes werden die vorhandenen Werteausprägungen eines Attributes mit Hilfe des Medians zuerst in zwei und dann in insgesamt vier Teile (*Quartile*) zerlegt, wie dies in Abbildung 2.2 angedeutet ist. Sind nun q_1, q_2

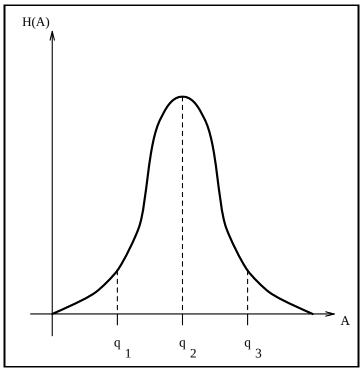


Abbildung 2.2: Unterteilung der Werteverteilung in Quartile

und q_3 die Grenzwerte der Quartile, dann berechnet sich der Interquartilsabstand D_i für die in der aktuellen Fallmenge M vorhandenen Werteausprägungen von Attribut A_i mit Hilfe des lokalen Ähnlichkeitsmaßes $sim_i(w_1, w_2)$ wie folgt:

$$D_i := 1 - sim_i(q_1, q_3) \text{ mit } sim_i \text{ wie in Def. 2.3} \tag{2.6}$$

Hier wurde die Voraussetzung aus Abschnitt 2.2 verwendet, daß das lokale Ähnlichkeitsmaße sim_i auf $[0, 1]$ normiert sind, damit die Subtraktion gerade die notwendige Umrechnung der Ähnlichkeit in eine Distanz bewirkt. Die auf diesem Ansatz beruhende Funktion `SelectAttr` gibt als Resultat nun gerade das Attribut A_i zurück, welches den größten Interquartilsabstand D_i bezüglich M hat, da dies als Indiz für die größte Streuung der durch q_2 bildbaren Partitionen von M bezüglich A_i angesehen werden kann.

Ein weiterer in diese Kategorie passender Ansatz ist der in [14] vorgeschlagene *Abstand in der größten Lücke*. Der Abstand D_i für ein Attribut A_i läßt sich sehr einfach mit Hilfe des lokalen Ähnlichkeitsmaßes sim_i berechnen, wenn die in der aktuellen Fallmenge M vorkommenden Werteausprägungen als geordnete Menge $\Theta_i = \{w_1, \ldots, w_l\}$ mit $w_1 < \ldots < w_l$ vorliegen:

$$D_i := \max \{1 - sim_i(w_u, w_{u+1}) | 1 \leq u < l\} \tag{2.7}$$

Hier wird wieder wie oben davon ausgegangen, daß sim_i auf $[0, 1]$ normiert ist. Die Auswahl des Diskriminatorattributs geschieht analog zu oben durch die Wahl des Attributs A_i mit maximalem D_i.

In die zweite Kategorie der auf Seite 13 beschriebenen Ansätze gehört das Maß der *mittleren Ähnlichkeit*. In [9, 14] werden noch weitere in diese Kategorie passende Ansätze untersucht, welche dort als *a-posteriori Abschätzung* bezeichnet werden. Das Ergebnis der Untersuchung war jedoch, daß es sich bei der mittleren Ähnlichkeit um die mit Abstand beste Abschätzung handelt. Diese wird nun im folgenden vorgestellt.

Ist M die aktuelle Fallmenge und $M_1, \ldots, M_l$ eine Partitionierung von M in l disjunkte Teilmengen mit $M_i = \{f_1^{(i)}, \ldots, f_{m_i}^{(i)}\}$, so berechnet sich die mittlere Ähnlichkeit $\overline{sim_{M_i}}$ jeder Partition wie folgt:

$$\overline{sim_{M_i}} := \frac{2}{|M_i|(|M_i| + 1)} \sum_{u=1}^{|M_i|} \sum_{v=u}^{|M_i|} sim(f_u^{(i)}, f_v^{(i)}) \tag{2.8}$$

Diese Formel gilt nur für symmetrische Ähnlichkeitsmaße. Für unsymmetrische Maße muß die Formel wie folgt abgeändert werden:

$$\overline{sim_{M_i}} := \frac{1}{|M_i|^2} \sum_{f_u^{(i)}, f_v^{(i)} \in M_i} sim(f_u^{(i)}, f_v^{(i)}) \tag{2.9}$$

Die mittlere Ähnlichkeit einer Partitionierung $\overline{sim_{M_1,\ldots,M_l}}$ kann nun auf zweierlei Weise berechnet werden:

1. ohne Gewichtung der einzelnen Partitionen:

$$\overline{sim_{M_1,\ldots,M_l}} := \frac{1}{l} \sum_{i=1}^{l} \overline{sim_{M_i}} \tag{2.10}$$

2. mit Gewichtung der einzelnen Partitionen anhand ihrer Mächtigkeit:

$$\overline{sim_{M_1,\ldots,M_l}} := \frac{1}{|M|} \sum_{i=1}^{l} |M_i| \overline{sim_{M_i}} \tag{2.11}$$

Als Diskriminatorattribut wird nun das Attribut gewählt, welches den größten mittleren Ähnlichkeitswert $\overline{sim_{M_1,\ldots,M_l}}$ mit der bezüglich des betreffenden Attributs gewählten Partitionierung erzielt. Eine noch offene Frage ist nun, wie eine gute Partitionierung für ein gewähltes Attribut zu finden ist. Möglichkeiten dazu werden im folgenden Abschnitt angegeben. Ein etwas anderer Ansatz, welcher die Wahl des Diskriminatorattributs und die Suche nach einer optimalen Partitionierung zusammenfaßt wird in Kapitel 3 eingeführt.

Wahl der Partitionierung bzw. des Partitionswertes (`SelectPartitionValue`)

Die Wahl der Partitionierung bzw. des Partitionswertes bestimmt die Struktur des k-d-Baumes. Sie ist also neben der Wahl des Diskriminatorattributs entscheidend für die Effizienz des Retrievals. Hier sollen nun Ansätze vorgestellt werden, welche ausschließlich geordnete Werteausprägungen von Attributen bearbeiten können, da nur diese Art von Attributen in den ursprünglichen k-d-Bäumen verarbeitet werden konnte. Ungeordnete Werteausprägungen werden später als Erweiterung des k-d-Baum-Begriffs behandelt.

Der Partitionswert, wie er in Definition 2.6 (s. Seite 13) verwendet wird, definiert auf natürliche Weise eine Partitionierung der aktuellen Fallmenge M. Ist P der Partitionswert bezüglich Attribut A_i, ist also $P =$ `SelectPartitionValue(`M`)`, so ist $(M_\leq, M_>)$ die zugehörige Partitionierung mit:

- $M_\leq = \{<(w_1, \ldots, w_k), s> \in M | w_i \leq P\}$
- $M_> = \{<(w_1, \ldots, w_k), s> \in M | w_i > P\}$

Die Frage ist nun, wie P gewählt werden soll, d. h. wie die Funktion `SelectPartitionValue` realisiert werden kann. In [5] und [15] wird der *Median* als Partitionswert in Verbindung mit dem Interquartilsabstand vorgeschlagen. Der Median bezüglich des Attributs A_i ist der Wert, welcher in der *geordneten* Aufzählung[4] $\Theta_i = \{w_1, \ldots, w_l\}$ mit $w_1 < \ldots < w_l$ der Werteausprägungen aller Fälle aus M als mittlerer Wert von Θ_i aufgezählt wird, d. h. der Index I_{Θ_i} des Medians läßt sich wie folgt berechnen:

$$I_{\Theta_i} := \left\lfloor \frac{|\Theta_i| + 1}{2} \right\rfloor \tag{2.12}$$

Der Partitionswert P ist dann $w_{I_{\Theta_i}}$. Die Wahl des Medians als Partitionswert soll gewährleisten, daß die zu bearbeitende Fallmenge M in zwei möglichst gleichgroße Hälften aufgeteilt wird, um die Tiefe des k-d-Baumes möglichst gering zu halten, und ein Retrieval mit mittlerer Komplexität

[4] ohne Duplikate

$O(\log m)$ zu ermöglichen, wobei m wieder die Anzahl der Fälle der Fallbasis bezeichnet. Diese Komplexität kann aber nur dann erreicht werden, wenn (fast) kein Backtracking bei der Suche durch den Baum benötigt wird. In [14] wird jedoch gezeigt, daß gerade die Wahl des Medians ein hohes Maß an Backtracking erfordert, weshalb sich die gewünschte mittlere Komplexität nur selten einstellt. Deshalb wurde dort *der größere Wert beim maximalen Abstand* als Partitionswert eingeführt. Liegt Θ_i wieder wie oben geordnet vor, so berechnet sich der Partitionswert P wie folgt:

$$P = w_{u+1} \text{ mit } 1 - sim_i(w_u, w_{u+1}) = D_i \tag{2.13}$$

mit D_i wie in Gleichung 2.7. Bei dieser Variante kommt es nicht mehr zu balancierten k-d-Bäumen wie beim Median, allerdings hat sich die Balancierung als nicht entscheidend erwiesen, da es vielmehr darauf ankommt, daß die Fälle optimal getrennt werden.

2.3.2 Retrieval

Im vorangegangenen Abschnitt wurden verschiedene Möglichkeiten der k-d-Baum Generierung vorgestellt. Nun soll gezeigt werden, wie der Baum zum effizienten Retrieval von Fällen genutzt werden kann. Dazu wird eine Suchprozedur vorgestellt, welche es nicht nur erlaubt, identische Fälle aus der zum k-d-Baum gehörigen Fallbasis zu finden, sondern die es vielmehr ermöglicht, die zum Anfragefall n *ähnlichsten* Fälle zu finden. Da der Baum im allgemeinen nicht jede Anfrage mit einem Durchlauf von seiner Wurzel zu einem Blatt (Bucket) befriedigen kann, wird, falls nötig, *Backtracking* eingesetzt, um jede Anfrage bewältigen zu können. Das Backtracking wird jedoch mit Hilfe diverser Steuerungsmechanismen stark eingeschränkt, um nah an der mittleren Komplexität $O(\log m)$ zu bleiben.

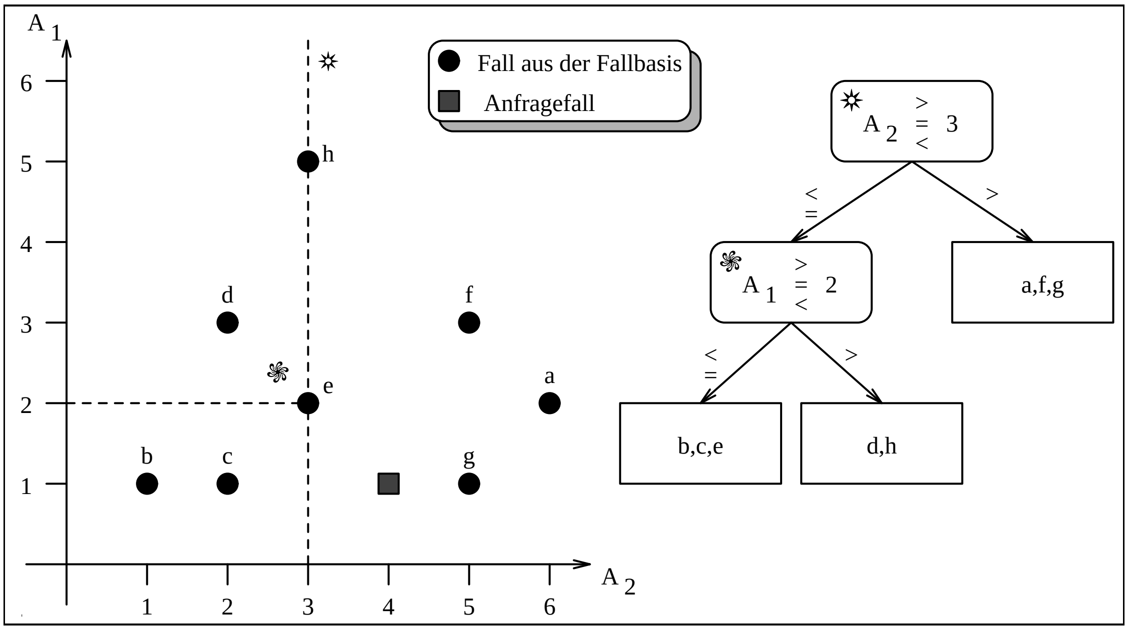


Abbildung 2.3: Beispiel einer Fallbasis mit k-d-Baum

Betrachten wir nun anhand des Beispiels aus Abbildung 2.3, wie ein k-d-Baum zur Suche der n ähnlichsten Fälle benutzt werden kann, wobei hier von Fällen mit zwei Attributen A_1 und A_2 jeweils mit dem Wertebereich $N = \{1, 2, 3, \ldots\}$ ausgegangen werden soll. Es ist also im Beispiel $k = 2$. Der angegebene Baum sei dabei für die maximale Bucketgröße $b = 3$ mit nicht näher spezifizierten Heuristiken für die Auswahl des Diskriminatorattributs und des Partitionswertes wie angegeben konstruiert worden. Die Aufgabe sei nun, die $n = 1$ ähnlichsten Fälle zum angegebenen

Anfragefall zu finden. Wir verwenden dabei ein Ähnlichkeitsmaß *sim*, daß sich wie folgt auf den euklidischen Abstand d in N^k stützt:

$$d((w_{1_1},\ldots,w_{1_k}),(w_{2_1},\ldots,w_{2_k})) := \sqrt{\sum_{i=1}^{k} |w_{1_i} - w_{2_i}|^2} \tag{2.14}$$

$$sim((w_{1_1},\ldots,w_{1_k}),(w_{2_1},\ldots,w_{2_k})) := 1 - \frac{d((w_{1_1},\ldots,w_{1_k}),(w_{2_1},\ldots,w_{2_k}))}{1 + d((w_{1_1},\ldots,w_{1_k}),(w_{2_1},\ldots,w_{2_k}))} \tag{2.15}$$

Daraus ergeben sich für das angegebene Beispiel folgende Werte:

Fall	A_1	A_2	d	sim
g	1	5	1	0.5
e	2	3	1.4142	0.4142
c	1	2	2	$0.\overline{3}$
a	2	6	2.2361	0.3090
f	2	6	2.2361	0.3090
d	3	2	2.8284	0.2612
b	1	1	3	0.25
Anfrage	1	4		

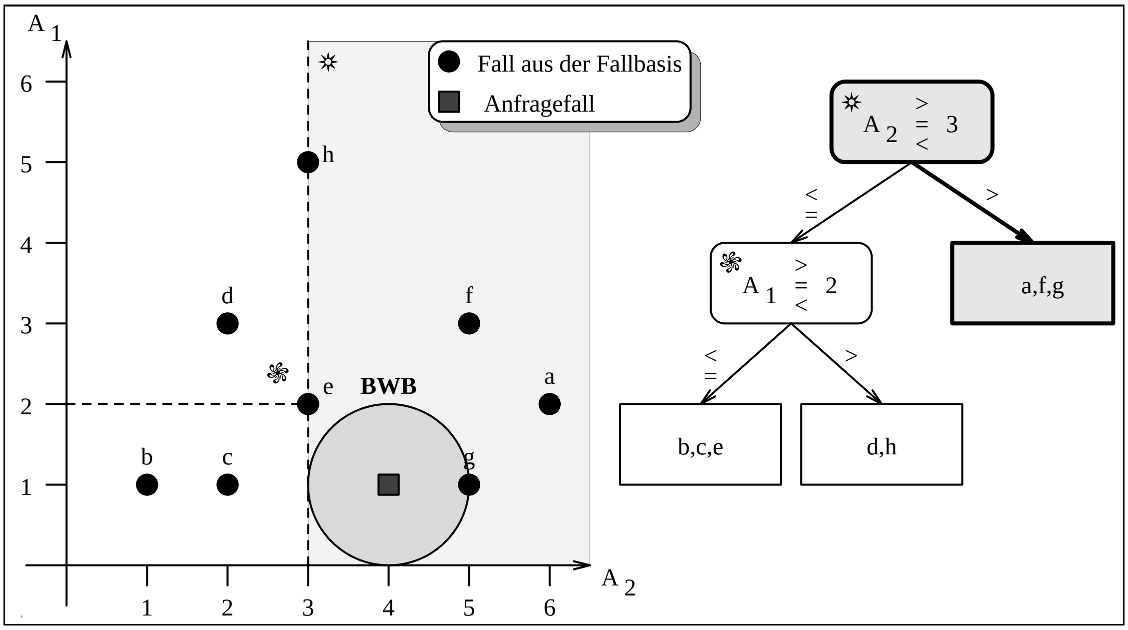


Abbildung 2.4: Suchen des nächsten Nachbarn

Wie man leicht sieht ist Fall g der ähnlichste Fall zum Anfragefall. Dieser ist auch sehr einfach im k-d-Baum zu finden, wenn man das Bucket des Baums bestimmt, in welches der Anfragefall gehört, d. h. man folgt den beschrifteten Kanten von der Wurzel bis zu einem Bucket wie in Abbildung 2.4 angedeutet. Dies führt zu dem Bucket mit den Fällen a, b und g. Nun muß noch innerhalb des Bucket *lineare Suche* angewendet werden, um den ähnlichsten der vorgefundenen Fälle a,f und g zu bestimmen. Dies führt dann zum gewünschten nächsten Nachbarn Fall g.

Der oben suggerierte Algorithmus zur Bestimmung des nächsten Nachbarn zu einem Anfragefall ist jedoch trügerisch, denn nicht immer findet man in dem Bucket, in welches der Anfragefall gehören würde, auch den nächsten Nachbarn vor. Man kann sich das sehr leicht verdeutlichen, wenn man den bisherigen nächsten Nachbarn, Fall g, aus der Fallbasis und dem k-d-Baum (also dem Bucket) entfernt. Dann verändert sich der Baum nun in dem rechtem Bucket, in welchem Fall g entfernt wird. Der Anfragefall würde dann natürlich immer noch in das rechte Bucket gehören, obgleich der nächste Nachbar, sprich Fall e, in einem ganz anderen Bucket und Teilbaum zu finden ist.

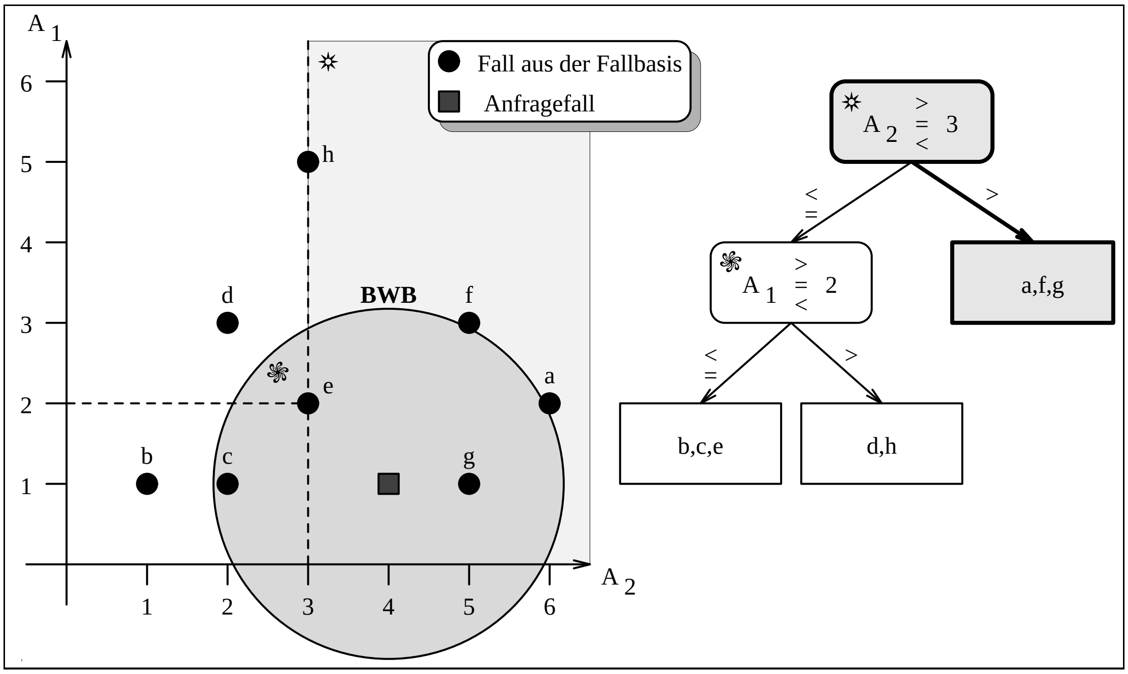


Abbildung 2.5: Suchen der 2 nächsten Nachbarn (1. Schritt)

Eine ganz ähnliche Problematik ergibt sich, wenn man mehr als nur einen Nachbarn bestimmen möchte. Es seien also beispielsweise die $n = 2$ ähnlichsten Fälle zu finden. Wie oben schon angedeutet ist der zweitähnlichste Fall der Fall e und damit nicht in dem rechten Bucket enthalten. Es ist also noch nötig den restlichen Baum nach dem zweitbesten Fall zu durchsuchen. An dieser Stelle könnte man nun so vorgehen, daß man den gesamten restlichen Baum durchsucht, bis man alle n ähnlichsten Fälle gefunden hat. Dies ist allerdings sehr ineffizient, da in diesem Fall alle Fälle mit dem Anfragefall verglichen werden müssen und zusätzlich noch der Aufwand des Baumdurchlaufs entsteht. Ausserdem erreicht man so nur eine Komplexität von $O(m)$, d. h. man hätte die Retrieval-Aufgabe doch besser gleich mit *linearer Suche* gelöst und hätte sich die Zeit für die Generierung des k-d-Baums sowie den zusätzlichen Overhead, welcher durch den Baumdurchlauf entsteht, gespart. Eine Lösungsidee könnte nun sein, daß man die Bucketgröße so groß wählt, daß die Notwendigkeit des Suchens über ein Bucket hinaus relativ selten besteht. Diese Idee hat jedoch in der Praxis keinen Wert, da apriori nicht bekannt ist, wieviele Fälle man zu einem beliebigen Anfragefall finden muß, um die dahinter stehende Aufgabe zufriedenstellend zu lösen. Man müßte also die Bucketgröße b so groß wählen, daß auch umfassende Anfragen mit *einem* Bucket-Zugriff erledigt werden könnten. Da aber innerhalb eines Buckets lineare Suche zur Bestimmung des/der besten Falles/Fälle benutzt wird, hat man damit nicht mehr als eine lineare Suche auf eine etwas verkleinerte Fallbasis erreicht. Glücklicherweise ist es aber nicht notwendig, nach dem ersten Bucket und noch nicht befriedigter Anfrage den kompletten Rest des k-d-Baumes zu durchsuchen. Vielmehr kann man mittels zweier einfacher Tests einen großen Teil des k-d-Baums von der Suche ausschließen.

Steuerung des Retrievals

Wie beim obigen Beispiel angedeutet wurde, ist das Bucket mit dem *wahrscheinlich* ähnlichsten Fall zum Anfragefall relativ leicht zu finden, indem man das Bucket bestimmt, in welches der Anfragefall gehören würde. Die Hoffnung auf eine diesbezüglich große Wahrscheinlichkeit relativiert sich jedoch, wenn man sich vor Augen führt, daß beim Durchlaufen von der Wurzel bis zum Bucket nur eine geringe Anzahl von Attributen getestet wird[5], und nur für diese die Hoffnung auf eine gewisse Ähnlichkeit mit den Attributen des Anfragefalles besteht. Verschärfend kann noch hinzukommen, daß einige Attribute in mehreren inneren Knoten eines Wurzel-Bucket-Pfades getestet werden. Außerdem geben innere Knoten, wie z. B. nach Definition 2.6 nur eine Tendenz für den Attributwert in Richtung "größer" oder "kleiner" an.

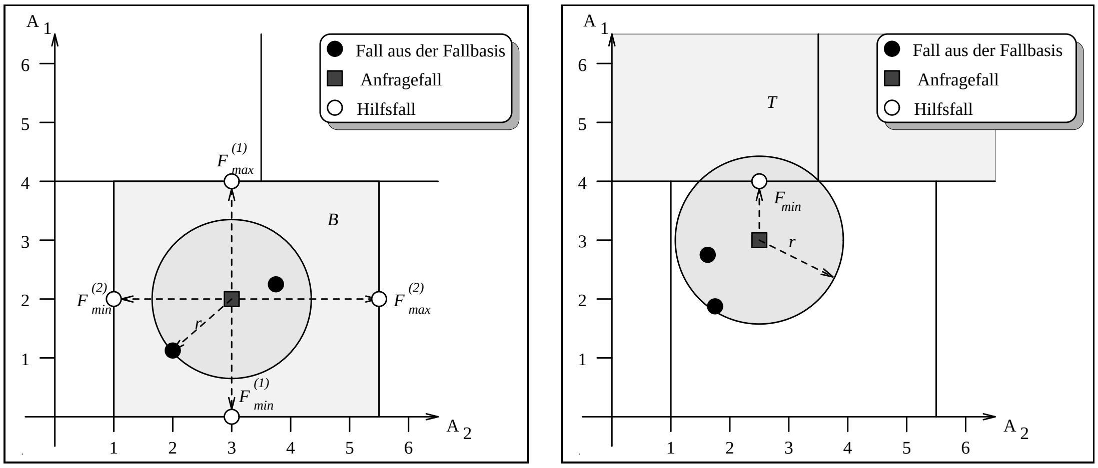


Abbildung 2.6: Ball-Within-Bounds-Test (l) und Bounds-Overlap-Ball-Test (r)

Ist man also mit Hilfe des Anfragefalls in einem Bucket angekommen, so ist vor allen Dingen erst einmal nötig festzustellen, ob man die n ähnlichsten Fälle überhaupt schon gefunden hat oder vielleicht nur einen Teil von n oder u. U. auch gar keinen der n ähnlichsten Fälle. Dies wird mit dem sogenannten *Ball-Within-Bounds*-Test, kurz BWB-Test, geprüft. Dieser Test resultiert aus einer geometrischen Vorstellung des Datenraums $\mathcal{W}$ (s. Definition 2.1), in dem man sich die Fälle als Punkte vorstellen kann. Nun denkt man sich den Anfragefall als Mittelpunkt einer k-dimensionalen Hyperkugel mit dem Radius r. Wählt man nun den Radius r so[6], daß er dem zur Zeit n-größten Ähnlichkeitswert entspricht, so muß die in dieser Weise konstruierte Kugel alle Punkte, sprich Fälle, enthalten, welche einen mindestens genauso großen Ähnlichkeitswert zum Anfragefall haben. Betrachtet man die durch den k-d-Baum herleitbaren geometrischen Beschreibungen des Datenraums, den ein Bucket repräsentiert, so kann man testen, ob außerhalb des betreffenden Buckets kein ähnlicherer als der bisher n-ähnlichste Fall gefunden werden kann. Dies ist nämlich genau dann der Fall, wenn die Hyperkugel vollständig innerhalb der Bucketbeschreibung liegt. Genau dieser Fall ist in Abbildung 2.4 zu ersehen, in welcher die Aufgabe lautete, nur $n = 1$ nächsten Nachbarn zu finden. Hier ist die Hyperkugel vollständig innerhalb des Buckets, wenn man $r = d(\text{Anfragefall}, \text{Fall } g)$ setzt.

Betrachtet man jedoch Abbildung 2.5, in welcher $n = 2$ nächste Nachbarn gesucht werden sollen, so reicht die Inspizierung des rechten Buckets nicht aus, da dort als zweitnächster Nachbar nur (gemeinsam) die Fälle a und f gefunden werden können. Das ergibt für die Hyperkugel den Radius $r = d(\text{Anfragefall}, \text{Fall } a \text{ oder } f)$. Innerhalb dieser Hyperkugel liegt aber noch Fall c und Fall e, was einen Abbruch der Suche an dieser Stelle verbietet. Wie man auch sieht liegt die Hyperkugel nicht vollständig innerhalb der Grenzen des Buckets. Genau dieses Kriterium wird zu Definition

[5] In der Praxis hat sich gezeigt, daß oft recht ausgeglichene k-d-Bäume generiert werden. D. h. aber, daß in einem Wurzel-Bucket-Pfad nur ungefähr $\log_2 m$ Attribute getestet werden, wobei m die Anzahl der Fälle beschreibt

[6] Für eine Umrechnung von Ähnlichkeiten in Distanzen und umgekehrt siehe [17]

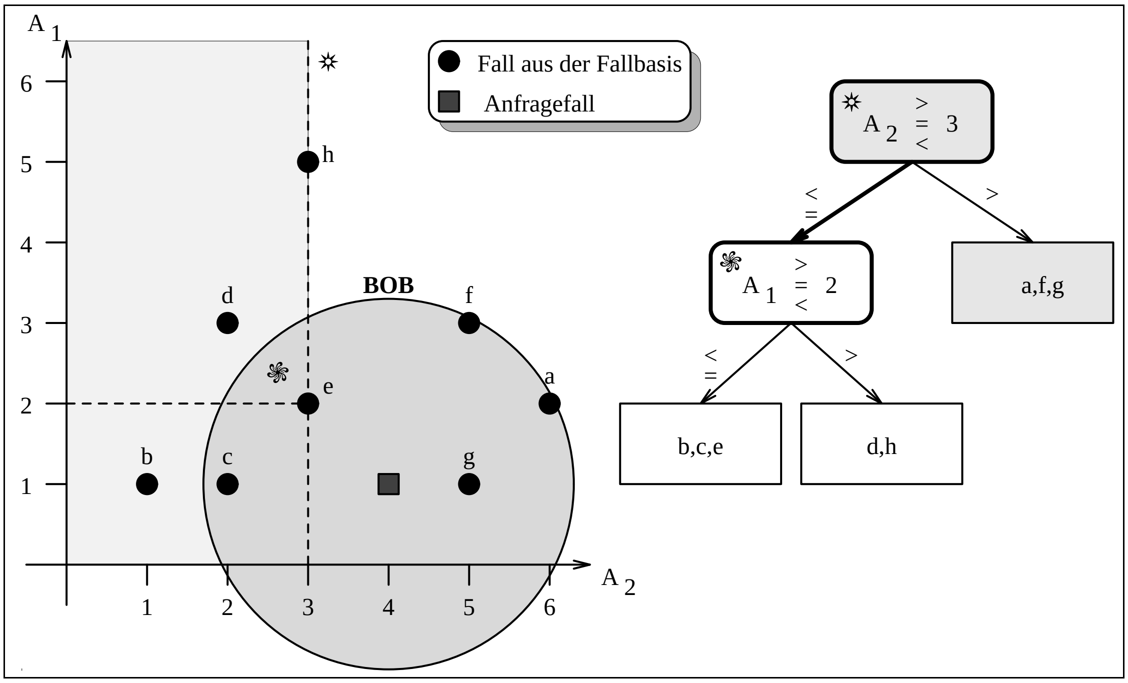


Abbildung 2.7: Suchen der 2 nächsten Nachbarn (2. Schritt)

des BWB-Test herangezogen. Dies berücksichtigt zwar einerseits nicht, ob tatsächlich noch Fälle innerhalb der Kugel liegen und ist damit recht pessimistisch ausgelegt, ist aber andererseits sehr viel leichter zu realisieren.

Der BWB-Test läßt sich wie folgt definieren (s. Abbildung 2.6):

Definition 2.7 (Ball-Within-Bounds-Test) *Seien n ähnliche Fälle zu finden*[7]*, und B der aktuell betrachtete Teilbaum, der durch die Dimensionsgrenzen $((l_1, u_1), \ldots, (l_k, u_k))$ beschrieben werden kann, wobei l_i die untere und u_i die obere Schranke bezeichnet. Sei weiter $F_q = (w_{q_1}, \ldots, w_{q_k})$ der Anfragefall und r der Ähnlichkeitsradius, d. h.*

$$r = \begin{cases} sim(F_q, F') & \textit{wenn } F' \textit{ der } n\textit{-ähnlichste der bisher gefundenen Fälle ist} \\ -\infty & \textit{wenn erst} < n \textit{ verschieden ähnliche Fälle gefunden wurden} \end{cases}$$

Wir konstruieren nun Hilfsfälle $F_{min}^{(i)} = (h_1^{(i)}, \ldots, h_k^{(i)})$ und $F_{max}^{(i)} = (v_1^{(i)}, \ldots, v_k^{(i)})$ für $i, j = 1, \ldots, k$ wie folgt:

$$h_j^{(i)} = \begin{cases} l_j & \textit{falls } i = j \\ w_{q_j} & \textit{sonst} \end{cases}$$

$$v_j^{(i)} = \begin{cases} u_j & \textit{falls } i = j \\ w_{q_j} & \textit{sonst} \end{cases}$$

Das Prädikat $\mathtt{BWB}(F_q, r, B)$ läßt sich nun wie folgt beschreiben:

$$\mathtt{BWB}(F_q, r, B) \Leftrightarrow \forall i \in \{1, \ldots, k\} : sim(F_q, F_{min}^{(i)}) < r \wedge sim(F_q, F_{max}^{(i)}) < r \wedge l_i \leq_i w_{q_i} \leq_i u_i$$

[7] Fälle mit gleicher Ähnlichkeit zum Anfragefall werden zusammengefaßt, d. h. es werden n verschiedene "Ähnlichkeiten" gesucht

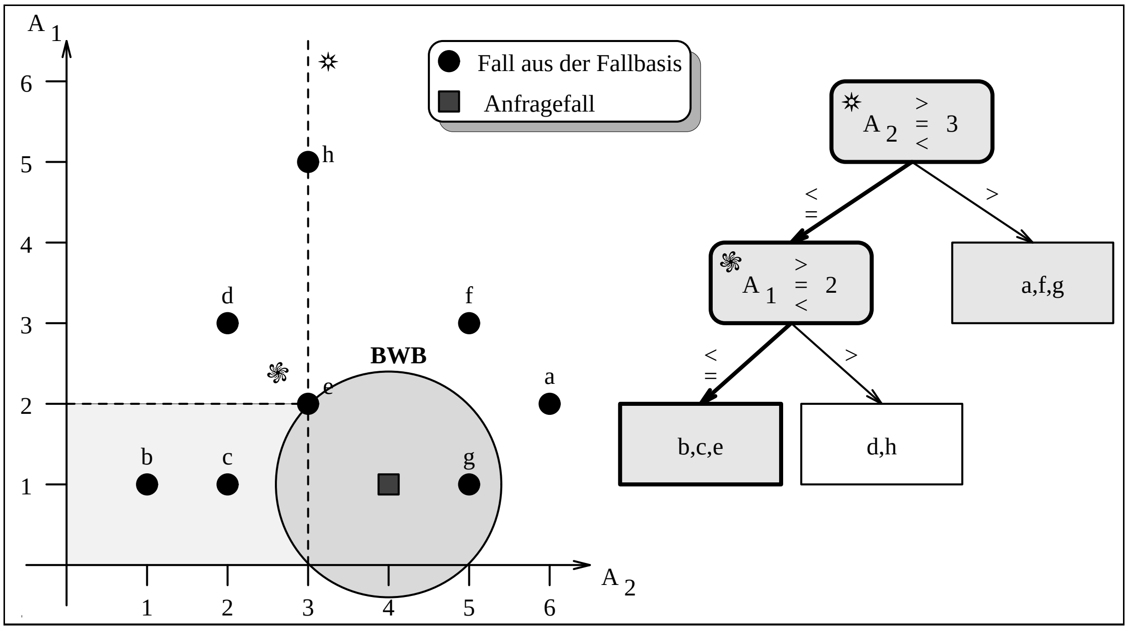


Abbildung 2.8: Suchen der 2 nächsten Nachbarn (3. Schritt)

Obige Definition des Prädikats läßt sich noch vereinfachen, denn es genügt, den an F_q näher liegenden Hilfsfall von $F^{(i)}_{max}$ und $F^{(i)}_{min}$ zu betrachten. Setzt man also

$$F^{(i)} = \begin{cases} F^{(i)}_{max} & \textit{falls } sim_i(w_{q_i}, v_i^{(i)}) > sim_i(w_{q_i}, h_i^{(i)}) \\ F^{(i)}_{min} & \textit{sonst} \end{cases}$$

so ergibt sich:

$$\texttt{BWB}(F_q, r, B) \Leftrightarrow \forall i \in \{1, \ldots, k\} : sim(F_q, F^{(i)}) < r \wedge l_i \leq_i w_{q_i} \leq_i u_i$$

Setzen wir nun das auf Seite 19 unterbrochene Beispiel für $n = 2$ fort. Wie man in Abbildung 2.5 sieht, mißlingt der BWB-Test und die Suche muß im restlichen Baum fortgesetzt werden. Es wird nun das Backtracking gestartet, in dem wir die Suche beim vorletzten Knoten fortsetzen. Dies ist aber die Wurzel des Baumes, so daß wir nur die Möglichkeit haben, in den noch nicht besuchten Teilbaum zu gehen, sprich dem linken Ast zu folgen. Da wir allerdings nicht wissen können, ob sich innerhalb dieses Teilbaumes auch wirklich bessere, d. h. zum Anfragefall ähnlichere Fälle befinden können, wird jetzt der zweite Test, der sogenannte *Bounds-Overlap-Ball*-Test, kurz BOB-Test, angewendet. Dieser bildet analog zum BWB-Test die aktuelle Hyperkugel und untersucht, ob diese die zum betrachteten Teilbaum gehörige Beschreibung des Datenraums schneidet, wie das z. B. in Abbildung 2.7 der Fall ist.

Der BOB-Test läßt sich dann wie folgt definieren (s. Abbildung 2.6):

Definition 2.8 (Bounds-Overlap-Ball-Test) *Sei T der zu untersuchende Teilbaum mit den Dimensionsgrenzen $((l_1, u_1), \ldots, (l_k, u_k))$ sowie F_q und r wie in Definition 2.7. Wir konstruieren einen Hilfsfall F_{min}, welcher am nahesten an F_q aber noch innerhalb von T liegt. Liegt dieser Fall innerhalb der Hyperkugel, so überlappt diese in den Teilbaum T. Der Hilfsfall $F_{min} = (v_1, \ldots, v_k)$ wird wie folgt konstruiert:*

$$v_i = \begin{cases} w_{q_i} & \textit{falls } l_i \leq w_{q_i} \leq u_i \\ l_i & \textit{falls } w_{q_i} \leq l_i \\ u_i & \textit{falls } u_i \leq w_{q_i} \end{cases}$$

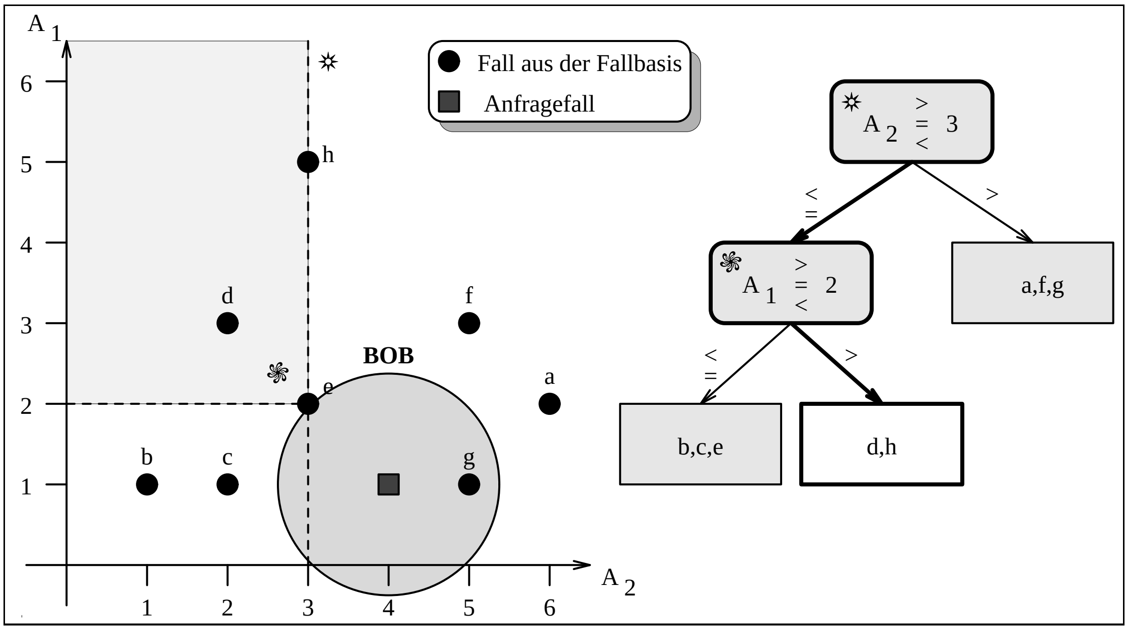


Abbildung 2.9: Suchen der 2 nächsten Nachbarn (4. Schritt)

Das Prädikat $\texttt{BOB}(F_q, r, T)$ *läßt sich nun wie folgt beschreiben:*

$$\texttt{BOB}(F_q, r, T) \Leftrightarrow sim(F_q, F_{min}) \geq r$$

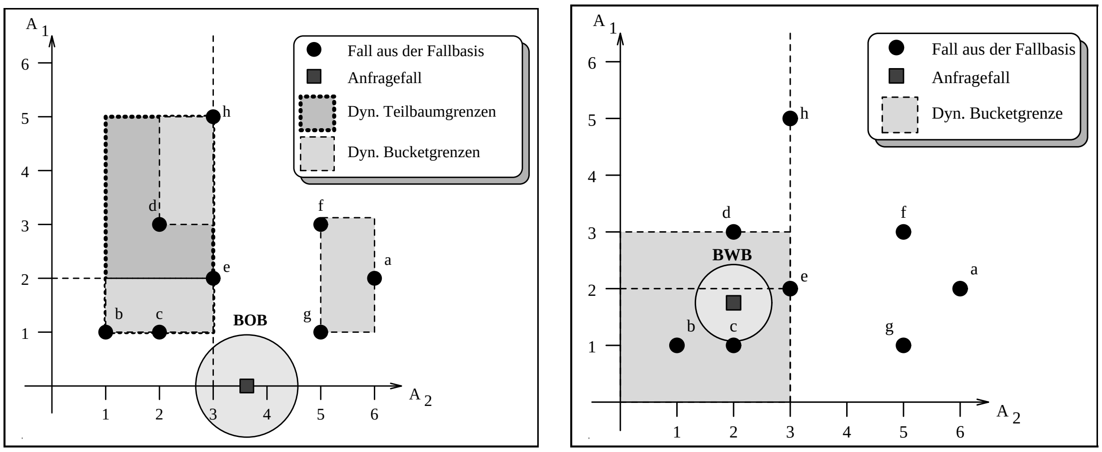


Abbildung 2.10: Minimale Grenzen (l) und maximale Grenzen (r)

In der Fortsetzung des Beispiels aus Abbildung 2.7 ist der BOB-Test erfüllt, d. h. der linke Teilbaum muß durchsucht werden, indem der Suchmechanismus rekursiv auf diesen Teilbaum ausgeführt wird. Wir beginnen also wieder damit, das Bucket zu suchen, in welches der Anfragefall (bezüglich des betrachteten Teilbaums) gehören würde. Dies ist – wie in Abbildung 2.8 zu sehen – das linke Bucket mit den Fällen b, c und e. Dort finden wir als zweitähnlichsten Fall den Fall e, was zu einer Verkleinerung des Radius der Hyperkugel führt. Nun testen wir wieder mit Hilfe des BWB-Tests (s. Abb. 2.8), ob die Suche in diesem Bucket abgebrochen werden kann. Das mißlingt, weshalb es wieder notwendig ist, zum Vaterknoten zurückzugehen.

Dort gibt es wieder die Möglichkeit, in dem noch nicht besuchten Teilbaum, sprich dem mittleren Bucket zu suchen. Zuvor wenden wir jedoch wieder den BOB-Test an, um festzustellen, ob dies überhaupt erfolgsversprechend ist. Dieser ist jedoch wie in Abbildung 2.9 zu ersehen nicht erfüllt und wir können ohne Inspizierung des mittleren Buckets die Suche beenden. Als Ergebnis haben wir also korrekterweise die Fälle g und e gefunden.

Retrieval-Beschleunigung

Wie im vorangegangenen Abschnitt schon erwähnt wurde, sind die BWB- und BOB-Test im allgemeinen sehr pessimistisch, da sie nur die Dimensionsgrenzen der Buckets und Teilbäume zur Entscheidung verwenden. Da diese aber innerhalb eines gewissen Rahmens frei wählbar sind, ist hier noch Raum für Verbesserung vorhanden. In [9] und [14] wurde der Begriff der *dynamischen Grenzen* eingeführt, was anhand von Abbildung 2.10 erläutert werden soll. Die Idee dabei ist, daß man je nach Art des Testes, ob BWB- oder BOB-Test, die Grenzen eines Bucket innerhalb des Möglichen so verändert, daß jeder Test möglichst viele Fälle von einer Untersuchung ausschließt. So kann man beispielsweise die Grenzen des Buckets soweit ausdehnen, bis sie Fälle in anderen Buckets erreichen, wie dies in Abbildung 2.10 rechts zu sehen ist. Dies ist ohne Verlust der Korrektheit möglich, da im hinzugewonnenen Raum sowieso keine Fälle liegen und die Grenze ohne Änderung des zugehörigen k-d-Baumes auch in dieser Weise hätte gelegt werden können. So erweiterte Grenzen ergeben dann die bestmögliche Ausbeute, d. h. Retrieval-Abbruch-Chance, beim BWB-Test. Wie man im Beispiel sieht kann dann der BWB-Test ein Abbruch des Retrievals verursachen, obwohl er eine Bucketgrenze überlappt.

Analog kann man beim BOB-Test verfahren. Hier ist es nur notwendig, daß sich die zum Test-Bucket benachbarten Buckets auf ihre minimale Ausdehnung zurückziehen. Im Beispiel in Abbildung 2.10 auf der linken Seite kann man sehen, daß dies einen frühzeitigen Ausschluß von Teilbäumen aus der Suchprozedur ermöglicht.

In der Praxis hat sich gezeigt, daß insbesondere der BOB-Test durch diese Erweiterung gewinnt. Der BWB-Test leidet ein wenig daran, daß der Mittelpunkt der Hyperkugel immer in dem Bucket liegt, in welchem der Anfragefall liegen würde. Sind die Grenzen dieses Buckets aber nicht sonderlich vergrößerbar, so kann der BWB-Test die Suche nur bei sehr großen Ähnlichkeitswerten abbrechen.

2.4 Induktion mit Entscheidungsbäumen

In [23] beschreibt Quinlan *Induktion* als

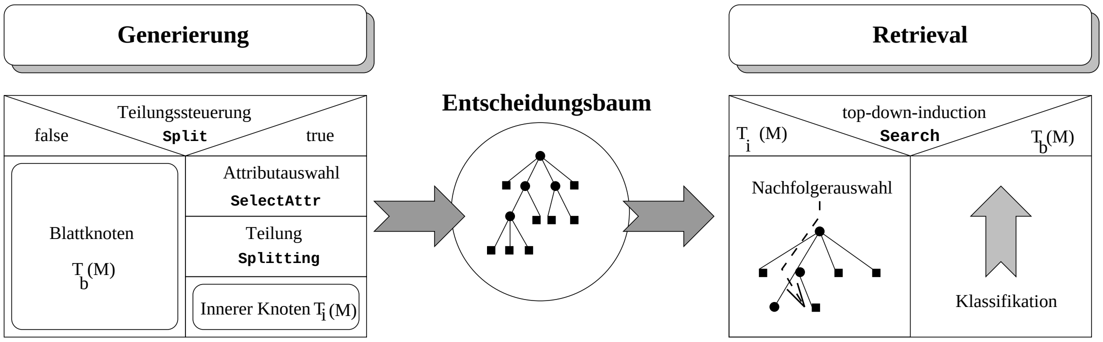
> ...the process by which structures underlying and explaining a chaotic agglomeration of examples are discovered through analysis of examples themselves.

Induktion ist also die Exktraktion von Wissen aus Beispielfällen, welche wieder wie in Definition 2.1 definiert werden. Es ist aber noch mehr, denn es wird nicht nur (sicheres) Wissen extrahiert, sondern es werden auch (unsichere) Hypothesen aufgestellt.

In der üblichen Anwendung ist die Menge von Lösungen $\mathcal{S}$ der Fälle einfach eine Menge von Klassen und die Aufgabe ist somit die Suche nach Regeln, welche eine Klasse anhand der in den Fällen auftretenden Attribute identifizieren. Die Intention ist nun, daß mit den so erzielten Regeln auch bisher ungesehene Fälle klassifiziert werden können. Als Hilfsmittel zur Erzeugung von Regeln wird deshalb Generalisierung von Fallbeschreibungen anwendet, welche die verwendeten Fälle[8] korrekt klassifizieren und zusätzlich noch (möglichst viele) ungesehene Fälle klassifizieren können. Nun ist aber letztlich unerheblich, ob als Konklusion einer so erzeugten Regel der Name einer Klasse oder aber eine komplexere Datenstruktur eingesetzt wird. Deshalb ist es durchaus legitim, daß man sich unter der Menge $\mathcal{S}$ statt einfacher Klassennamen auch komplexe Lösungsbeschreibungen vorstellt. Wir können also den Begriff Fall wieder konform zur bisherigen Bedeutung verwenden.

[8] In der Literatur werden diese häufig auch *Trainingsfälle* genannt

Im folgenden werden nur noch Regeln betrachtet, welche mit Hilfe von Entscheidungsbäumen dargestellt werden können. Diese haben den Vorteil, daß sie zwar eine z. B. gegenüber semantischen Netzen eingeschränkte Aussagekraft haben, aber eine effiziente Abarbeitung der Regeln ermöglichen, welche selbsterklärend mit dem Namen *TDIDT*[9] bezeichnet wird (s. [19]).



Die Vorstellung der Entscheidungsbäume gliedert sich wieder, analog zu den k-d-Bäumen, in die Bereiche Generierung und Retrieval.

2.4.1 Generierung

Es folgt nun die Definition der Datenstruktur Entscheidungsbaum, wobei diese im ersten Schritt wieder etwas allgemeiner gehalten ist.

Definition 2.9 (Allgemeiner Entscheidungsbaum) *Sei $M \subseteq FB$ eine Menge Fällen und* `Split`(M) *ein Prädikat, das genau dann gilt, wenn eine Unterteilung von M erfolgen soll.*

`SelectAttr`(M) *sei eine Funktion, welche das zu verwendende Diskriminatorattribut A_j auswählt, also die Dimension des Datenraums, in welcher M partitioniert werden soll, d. h.* `SelectAttr`$(M) = A_j$.

Im folgenden bezeichnet wieder $T_b(M)$ einen Blattknoten , welcher alle Fälle von M beinhaltet, und $T_i(M)$ einen inneren Knoten, welcher Verweise auf Nachfolgeknoten sowie auf A_j enthält. Wir benötigen noch eine Funktion `Splitting`$_j(M)$, *welche M in p_j Teilmengen ($p_j \leq |M|$) partitioniert, wobei p_j nur vom verwendeten Diskriminatorattribut A_j abhängt. Es heißt dann $T(M)$ allgemeiner Entscheidungsbaum, wenn gilt:*

- *Falls* $\neg$`Split`(M): $T(M) = T_b(M)$
- *Falls* `Split`(M): $T(M) = T_i(M)$ *mit*
  - $T_i(M)$ *enthält das verwendete Diskriminatorattribut* A_j
  - `Splitting`$_j(M) = (M_1, \ldots, M_{p_j})$, *wobei* $M = M_1 \cup \ldots \cup M_{p_j}$, *sowie* $M_u|_{A_j} \cap M_v|_{A_j} \subseteq \{\perp\}$ *(d. h. M_u und M_v enthalten nur verschiedene Ausprägungen von Attribut A_j außer eventuell den "Wert unbekannt") für* $1 \leq u < v \leq p_j$, *sowie* $\forall u : M_u \neq M$
  - $T_i(M)$ *verweist auf die Unterbäume* $T(M_1), \ldots, T(M_{p_j})$

Obige Definition ist sehr ähnlich zu Definition 2.5 bis auf folgende Unterschiede[10]:

- Es wird nicht $M_u \cap M_v = \emptyset$ verlangt
- $M_u|_{A_j} \cap M_v|_{A_j}$ kann den Platzhalter für den "unbekannten Wert" $\perp$ enthalten

[9] Dieses Palindrom steht für top-down induction with decision trees

[10] für $1 \leq u < v \leq p_j$

Diese beiden Unterschiede resultieren aus der unterschiedlichen Behandlung von $\perp$ beim Aufbau von k-d- und Entscheidungsbäumen. Die Behandlung von unbekannten Werteausprägungen ist in Def. 2.5 wie auch in obiger Definition noch nicht näher beschrieben worden und wird in Kapitel 3 vertieft. In diesem Abschnitt wollen wir ausschließlich auf Fälle mit bekannten Werteausprägungen für jedes Attribut beschränken. Damit wird Def. 2.9 identisch zu Def. 2.5. Dies bildet die Basis für die Hoffnung auf die Integrierbarkeit beider Ansätze. Es soll nun die Definition von Entscheidungsbäumen angegeben werden, wie sie in der Literatur verwendet wird (s. [19]).

Definition 2.10 (Entscheidungsbaum) *Ein allgemeiner Entscheidungsbaum $T(M)$ zu $M \subseteq FB$ heißt Entscheidungsbaum, wenn das Prädikat* `Split`(M) *sowie die Funktionen* `Splitting` *wie folgt festgelegt sind:*

- `Split`$(M) \Leftrightarrow \neg(\exists \tilde{s} : \forall <(w_1, \ldots, w_k), s> \in M : s = \tilde{s})$
- `Splitting`$_j(M) = (M_{j_1}, \ldots, M_{j_q})$, *falls*

$$\{w_{j_1}, \ldots, w_{j_q}\} = \{w_j \mid <(w_1, \ldots, w_k), s> \in M\}$$

und

$$M_{j_i} = \bigcup <(w_1, \ldots, w_{j-1}, w_{j_i}, w_{j+1}, \ldots, w_k), s> \in M$$

Vergleicht man obige Definition mit Def. 2.6, so erkennt man zwei Unterschiede:

1. Das Prädikat `Split` veranlaßt eine Teilung nicht bei Überschreiten einer gewissen Größe von M, sondern wenn mehrere verschiedene Problemlösungen oder Klassen in M vorkommen.
2. Die Funktionen `Splitting` unterteilen nach *jeder* Werteausprägung des verwendeten Diskriminatorattributs. Es können somit weit mehr als nur zwei Partitionen von M entstehen.

Es gibt noch einen Freiheitsgrad bei der Berechnung eines Entscheidungsbaumes. Dieser besteht wieder in der Funktion `SelectAttr` zur Auswahl des in jedem Partitionierungsschritts verwendeten Diskriminatorattributs A_j. Eine Methode zu deren Implementierung wird im folgenden vorgestellt.

Wahl des Diskriminatorattributs (`SelectAttr`)

Die richtige Wahl des Diskriminatorattributs bestimmt hier wieder die Effizienz[11] der Benutzung eines Entscheidungsbaumes. Da die Retrievalprozedur allerdings sehr einfach ausfällt (siehe unten), ist dies nicht von entscheidender Bedeutung. Vielmehr werden Entscheidungsbäume in dem Sinne benutzt, daß ein Retrievaldurchlauf eine *Abfragereihenfolge der Attribute* vorgibt, d. h. daß bei jedem Knoten des Baumes, der nicht gerade ein Blatt ist, entschieden werden muß, in welche Richtung weiter gesucht wird. Aus diesem Grund wird hier ein informationstheoretischer Ansatz zur Auswahl des Diskriminatorattributs benutzt. Dieser versucht, den Informationsgewinn, welcher durch Testen eines neuen Attributs entstehen kann, zu maximieren.

Gilt für die Menge aller Problemlösungen[12] $\mathcal{S}$, daß $\mathcal{S}=\{s_1, \ldots, s_m\}$, so lassen sich relative Häufigkeiten $\tilde{s_i}$ von Lösung s_i in einer betrachteten Trainingsmenge M angeben. Mit $s = \sum_{i=1}^{m} \tilde{s_i}$ ergibt sich der *Informationsgehalt* einer Lösung s_i wie folgt:

$$I(s_i) = -\log_2(\frac{\tilde{s_i}}{s}) \tag{2.16}$$

Der Informationsgehalt ist also eine auf $[0, \infty[$ normierte Funktion, die man sich auch als Gradmesser des "Überraschungseffekts" vorstellen kann, wobei der Wert 0 das sichere Ereignis beschreibt

[11] Die Effizienz bezieht sich auf die Anzahl der abzufragenden Attribute des Anfragefalls. Optimiert wird hier der Erwartungswert der Anzahl der Abfragen zur Bestimmung der Klasse eines Falls

[12] Diese stellen in der üblichen Anwendung von Entscheidungsbäumen nur Klassenbezeichner dar

und größere Werte ein höheres Maß an Überraschung beschreiben. Damit ergibt sich die *Entropie*, daß ist der Erwartungswert des Informationsgehalts, der Menge M zu:

$$H(M) = \sum_{i=1}^{m} \frac{\tilde{s}_i}{s} I(s_i) \tag{2.17}$$

Die Entropie ist eine auf $[0,1]$ normierte Funktion, wobei große Werte einen hohen mittleren Informationsgehalt, also eine Art Unentschiedenheit bezüglich der Klassifikation der in M enthaltenen Fälle beschreibt, und Werte nahe 0 eine (fast) eindeutige Klassifikation der Fälle aus M in Richtung einer einzigen Problemlösung/Klasse bedeutet. Daraus läßt sich dann auch die Anforderung an das auszuwählende Diskriminatorattribut formulieren: Es muß das Attribut ausgewählt werden, daß die Entropie am stärksten verringert, denn das Ziel ist ja an den Blättern die Entropie 0 zu haben. Dazu wird nun der Begriff des *Informationsgewinns* eingeführt. Dieser gibt an, wie stark das betrachtete Attribut die Entropie reduziert. Ist nun A_j ein Attribut mit den (diskreten) Werteausprägungen $a_1, \ldots, a_l$ und setzen wir $M_j^{(i)} = M|_{A_j=a_i}$, so ist $M_j^{(1)}, \ldots, M_j^{(l)}$ eine disjunkte[13] Zerlegung von M. Die Rückschlußentropie $H(M|A_j)$ ergibt sich nun wie folgt:

$$H(M|A_j) = \sum_{i=1}^{l} \frac{|M_j^{(i)}|}{|M|} H(M_j^{(i)}) \tag{2.18}$$

Es muß nun das Attribut A_j gewählt werden, für das die Rückschlußentropie am geringsten ist, d. h. für das $H(M) - H(M|A_j)$ maximal wird, denn diese Differenz gibt gerade den Informationsgewinn an, welchen man durch Testen des betreffenden Attributs erzielt. Es ist also

$$\texttt{SelectAttr}(M) = A_j \in \{A_1, \ldots, A_k\} \text{ mit } H(M|A_j) \text{ minimal} \tag{2.19}$$

Obige Auswahl des Diskriminatorattributs hat den kleinen Nachteil, daß sie Attribute mit großen Wertebereichen im allgemeinen bevorzugt. Dies ist dann von Nachteil, wenn die so generierten Entscheidungsbäume von menschlichen Experten nachvollzogen werden sollen. Auch die Klassifikationsgenauigkeit von Fällen, die nicht in der Trainingsmenge lagen, kann dadurch beeinträchtigt werden. Aus diesem Grund schlägt Quinlan in [21] eine "Normalisierung" des Informationsgewinns vor. Dazu verwendet er den Informationsgehalt eines Attributs A_j mit den Werteausprägungen $a_1, \ldots, a_l$ bezüglich der Fallmenge M wie folgt:

$$I(M, A_j) = -\sum_{i=1}^{l} \frac{|M_j^{(i)}|}{|M|} \log_2\left(\frac{|M_j^{(i)}|}{|M|}\right) \tag{2.20}$$

mit $M_j^{(i)}$ wie oben. $I(M, A_j)$ ist völlig unabhängig von den Klassen bzw. Lösungen der Fälle in M und ist sehr groß, wenn in den einzelnen Fallmengen $M_j^{(i)}$ die Anzahl der Fälle zu den vorhandenen Klassen/Lösungen fast gleich ist. Um Attribute mit solchen Eigenschaften nicht als Wurzel zu bekommen, kann man den Informationsgewinn zu

$$G(M, A_j) = \frac{H(M) - H(M|A_j)}{I(M, A_j)} \tag{2.21}$$

definieren. Da $G(M, A_j)$ nicht immer definiert ist und Attribute mit kleinem $I(M, A_j)$ bevorzugt, schlägt Quinlan folgendes Auswahlkriterium vor: Für alle Attribute $A_{i_1}, \ldots, A_{i_p}$ $(p \leq k)$ mit *überdurchschnittlichem* Informationsgewinn $H(M) - H(M, A_i)$ verfahre wie folgt:

$$\texttt{SelectAttr}(M) = A_j \in \{A_{i_1}, \ldots, A_{i_p}\} \text{ mit } G(M, A_j) \text{ maximal} \tag{2.22}$$

[13] Dies gilt natürlich nur, wenn, wie hier vorausgesetzt, keine unbekannten Werteausprägungen auftreten

2.4.2 Retrieval

Im vorangegangenen Abschnitt wurden Möglichkeiten der Generierung von Entscheidungsbäumen angegeben. Nun soll angegeben werden, wie diese zum Retrieval von Fällen benutzt werden können. Der verwendete Algorithmus ist sehr einfach: Man beginnt bei der Wurzel und unterscheidet folgende zwei Fälle:

- Befindet man sich in einem Blatt, so sind die in diesem enthaltenen Fälle das Retrieval-Ergebnis und das Retrieval ist beendet.
- Befindet man sich in einem inneren Knoten mit Diskriminatorattribut A_j, so folgt man der Kante, die mit der Werteausprägung des Anfragefalls bezüglich A_j beschriftet ist.

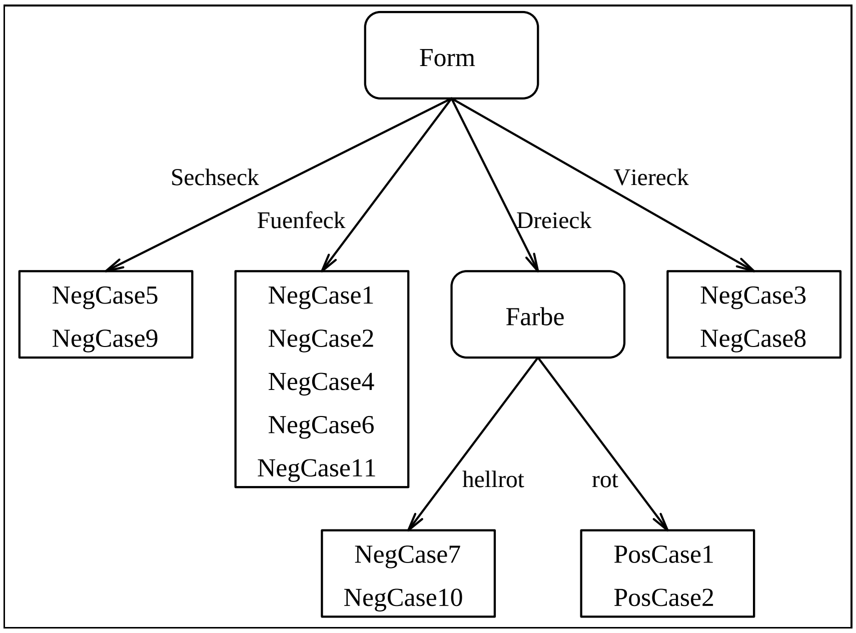


Abbildung 2.11: Entscheidungsbaum zum angegebenen Beispiel

Dazu soll ein kleines Beispiel betrachtet werden. Die Fälle haben die Attribute *Form, Farbe* und *Größe* mit folgenden Wertebereichen:

- Form $\in$ {Dreieck,Viereck,Fünfeck,Sechseck},
- Farbe $\in$ {dunkelrot,rot,hellrot} und
- Größe $\in$ {klein,normal,groß}.

Desweiteren sei die Menge der Problemlösungen $\mathcal{S}$ = {positiv,negativ}, wobei "positiv" und "negativ" der Einfachheit halber zwei Klassenbezeichner sind. Die Fallbasis und der Anfragefall sind wie folgt gegeben:

Fall	Farbe	Form	Größe	Klasse
PosCase1	rot	Dreieck	klein	positiv
PosCase2	rot	Dreieck	normal	positiv
NegCase1	dunkelrot	Fünfeck	groß	negativ
NegCase2	rot	Fünfeck	klein	negativ
NegCase3	rot	Viereck	normal	negativ
NegCase4	rot	Fünfeck	normal	negativ
NegCase5	rot	Sechseck	normal	negativ
NegCase6	hellrot	Fünfeck	klein	negativ
NegCase7	rot	Fünfeck	klein	negativ
NegCase8	hellrot	Sechseck	normal	negativ
NegCase9	hellrot	Sechseck	normal	negativ
NegCase10	hellrot	Dreieck	groß	negativ
NegCase11	hellrot	Fünfeck	groß	negativ
Anfragefall 1	rot	Dreieck	groß	
Anfragefall 2	hellrot	Dreieck	groß	

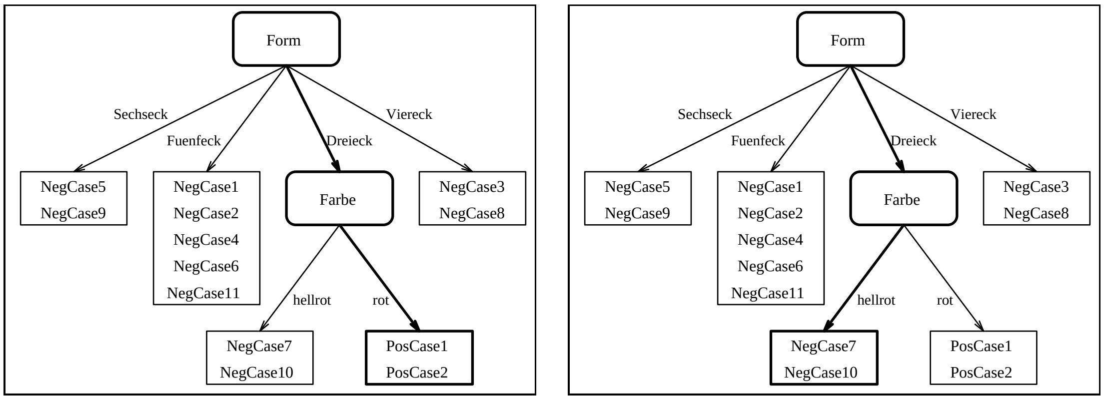


Abbildung 2.12: Suche nach Anfragefall 1 (l) und 2 (r)

Zu dieser Fallbasis ergibt sich der Entscheidungsbaum[14] wie in Abb. 2.11. Das Retrieval nach den Anfragefällen 1 und 2 ist in Abb. 2.12 dargestellt. Im Vergleich zum Retrieval in k-d-Bäumen (s. Abschnitt 2.3.2) zeigen sich dabei folgende Unterschiede:

1. Es wird (in Entscheidungsbäumen) kein Backtracking verwendet.
2. Es werden nicht die n ähnlichsten Fälle, sondern nur eine Problembeschreibung gefunden.
3. Die Fälle werden zum Retrieval nicht benötigt.
4. Der Anfragefall muß nicht vollständig (aber hinreichend komplett) vorliegen.

Punkt 1 ist ein zu begrüßender Umstand, denn auch beim Retrieval in k-d-Bäumen wird dieser Zustand mit Hilfe der Ball-Tests zu erreichen versucht, um die Retrieval-Geschwindigkeit zu erhöhen. Desweiteren leiten viele existierende Systeme, die Entscheidungsbäume zur Klassifikation verwenden, aus der Retrieval-Prozedur eine Abfragereihenfolge für die Attribute ab, indem gerade das Attribut (vom Benutzer) abgefragt wird, welches im aktuellen inneren Knoten benötigt wird. Dies macht allerdings nur Sinn mit Punkt 4. Hier wird also der Anfragefall nicht konkret "am Stück" angegeben. Diese implizite Abfragereihenfolge der Attribute kann allerdings auch sehr lästig für den Benutzer werden, wenn zusammengehörige Attributtests nicht en bloc sondern beliebig permutiert abgefragt werden.

[14] Mit `SelectAttr` wie in Gleichung 2.19

Punkt 2 ist absolut inakzeptabel, denn auch für die gefundene Problembeschreibung oder Klasse kann nicht garantiert werden, daß sie mit der des ähnlichsten Falls übereinstimmt[15]. Das ist auch nicht verwunderlich, denn zum Aufbau wie zum Retrieval wurde nirgends die verwendete Ähnlichkeitsfunktion verwendet. In den folgenden Kapiteln wird gezeigt, wie die Vorteile des schnellen Retrievals in Entscheidungsbäumen mit der Eigenschaft des korrekten und vollständigen Retrievals in k-d-Bäumen kombiniert werden kann.

Punkt 3 ist eine direkte Konsequenz aus Punkt 2 und im Sinne des Fallbasierten Schließens nicht unbedingt eine erstrebenswerte Eigenschaft.

Punkt 4 ist aus dem verwendeten Retrieval-Algorithmus ersichtlich. Wie man auch am angegebenen Beispiel sieht, werden vom Anfragefall nur die Attribute benötigt, welche auf dem Weg von der Wurzel zum Blatt auftreten. So werden im Beispiel nur die Attribute Form und Farbe benötigt. Das Attribut Größe wird nie benötigt. Würden wir nach einem großen roten Sechseck suchen, so würde sogar nur das Attribut Form benötigt. Diese Eigenschaft gewinnt dann an Bedeutung, wenn bspw. wie in der Diagnose (s. [11] und [10]) mit der Ermittlung jedes Attributwertes Kosten verbunden sind. Dann muß natürlich versucht werden, so wenig Kosten wie möglich zum Retrieval entstehen zu lassen.

2.5 Gegenüberstellung: Entscheidungsbaum – k-d-Baum

Es sollen nun noch die Eigenschaften beider Bäume gegenübergestellt werden. Dabei werden auch mögliche Forderungen für die Integration beider Ansätze mit angegeben.

Aspekt	Entscheidungsbaum	k-d-Baum	Integrationsforderung
Ähnlichkeitsmaß	unnötig	notwendig, beliebig unter Beachtung von Forderung 1 und 2	mehrere Ähnlichkeitsmaße mit Klassifikationsfähigkeiten
nearest-neighbour-Klassifikation mit Ähnlichkeitsmaß	unmöglich	möglich	wie k-d-Baum
Retrieval-Ergebnis	Klassifikation	n nächste Nachbarn	wie k-d-Baum
Fälle im Retrieval	unnötig	notwendig	wie k-d-Baum
Abfragereihenfolge der Attribute	vorgegeben	Anfragefall muß vollständig vorliegen	beliebige Vorgabe eines Anfragefalls mit evtl. unbekannten Attributen
Diskriminierung nach ungeordneten Attributen	möglich	unmöglich	wie Entscheidungsbaum
Diskriminierung nach geordneten Attributen	unmöglich	möglich	wie k-d-Baum
Unbekannte Attributwerte	aufwendiges Kopieren von Fällen	unmöglich	effiziente Erweiterung des k-d-Baums zur Integration unbekannter Attribute

Obige Gegenüberstellung läßt eigentlich nur eine Integrationsmöglichkeit offen: die Erweiterung des k-d-Baums und des verwendeten Ähnlichkeitsmaßes, um die Eigenschaft der schnellen Klassifikation, auch für Fälle mit ungeordneten, diskreten Attributen, zu ermöglichen. Dies wird in den folgenden Kapiteln angegangen.

[15] Diese Eigenschaft kann nur garantiert werden, wenn die betrachtete Domäne *vollständig* durch die gegebenen Fallbeispiele beschrieben ist. Dies beruht auf der Forderung an alle verwendeten Ähnlichkeitsmaße sim, daß $sim(f, f) = 1$ ist für jeden Fall f. Aber auch hier kann es Ausnahmen geben, wenn Unbekannte ins Spiel kommen

2.6 Zusammenfassung

In diesem Kapitel wurde das grundlegende Schema des Fallbasierten Schließens vorgestellt. Als eine der zentralen Aufgaben enthält dieses das ähnlichkeitsbasierte Retrieval von Fällen aus einer Fallbasis.

In dieser Arbeit wird eine einfach zu berechnende aber trotzdem leistungsfähige Klasse von Ähnlichkeitsmaßen – gewichtete Summen von lokalen Ähnlichkeiten – verwendet. Diese sichern trotz ihrer Flexibilität im Retrieval benötigte Verträglichkeits- und Monotoniebedingungen und sind über dies noch für den Benutzer nachvollziehbar.

Mit den k-d-Bäumen wurde eine Strukturierung der Fallbasis eingeführt, welche die Suche nach den n nächsten Nachbarn unter Verwendung eines mit einer Ordnung verträglichen und monotonen Ähnlichkeitsmaß effizient unterstützt. Die Effizienz des auf k-d-Bäumen verwendeten Retrieval-Verfahrens stützt sich auf eine geometrische Vorstellung der Fallbasis und leitet aus dieser Mechanismen zum Ausschluß von Fällen aus der aktuellen Suchanfrage ab. Diese Vorstellung ist allerdings auch ein großes Handicap der k-d-Bäume, da sich Attribute von Fällen mit nicht zu ordnenden Werten nicht in diese Vorstellungswelt einpassen lassen und somit nicht in k-d-Bäumen verwendbar sind, was zu unnötigen Falluntersuchung führen kann. Auch die Einbettung mehrerer Ähnlichkeitsmaße wie z. B. in PATDEX/2 (s. [10]) ist in k-d-Bäumen in der vorgestellten Form noch nicht möglich.

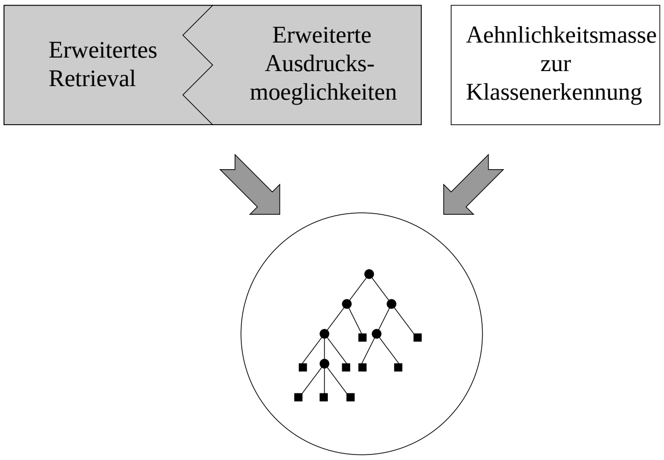
Entscheidungsbäume sind bekannt als eine effiziente Darstellungs- und Abarbeitungsform von Klassifikationsregeln. Da das Konstruktionsprinzip der Entscheidungsbäume auf der Präsentation von Fallbeispielen beruht, liegt die Idee nahe, Entscheidungsbäume auch zum Fallretrieval zu verwenden. Sie bieten den Vorteil eines sehr schnellen Retrievals, da kein Backtracking verwendet wird. Ein Problem bilden jedoch die so gefundenen Fälle, da diese i. a. nicht die ähnlichsten zum Anfragefall sind. Der Grund dafür ist sehr simpel: Weder bei der Konstruktion noch beim Retrieval wird ein Ähnlichkeitsmaß verwendet. Entscheidungsbäume sind also in höchstem Maß unflexibel und disqualifizieren sich somit für das Fallretrieval.

Kapitel 3

Erweiterung des ähnlichkeitsbasierten Fallretrievals mit k-d-Bäumen

3.1 Integration, Teil I

Die Idee der Integration von k-d- und Entscheidungsbaum, sprich induktivem und fallbasiertem Schließen, ist neben der Verbesserung und Generierung der verwendeten Ähnlichkeitsmaße die Erweiterung des k-d-Baums, sodaß mit diesem auch Fälle mit ungeordneten oder unbekannten Attributen ähnlich effizient gehandhabt werden können wie mit Entscheidungsbäumen.



Integrierter k-d- und Entscheidungsbaum

Desweiteren muß für die Verwendung leistungsfähiger, klassenspezifischer Ähnlichkeitsmaße [11, 10] die Möglichkeit geschaffen werden, daß diese beim Retrieval mit dem k-d-Baum verwendet werden können. Diese Aufgaben werden in den folgenden Abschnitten schrittweise angegangen.

3.2 Erweiterte Ausdrucksmöglichkeiten

Wie wir schon in Definition 2.6 gesehen haben, eignen sich k-d-Bäume in der ursprünglichen Fassung nur für Fälle mit Attributen, deren Wertebereich mit einer Totalordnung $<$ verträglich ist. Zwar wurden in [14] schon erste Ansätze zur Bearbeitung von *ungeordneten Attributen* aufgezeigt. Diese wurden vor allen Dingen zur Bewertung der Auswahl des Diskriminatorattributs verwendet[1].

[1] also innerhalb der Funktion `SelectAttr` verwendet

Letztlich ergaben sie aber nicht die Möglichkeit, daß nach einem ungeordneten Attribut diskriminiert und partitioniert wurde. Bei Fällen mit nicht vernachlässigbarem Anteil an ungeordneten Attributen hat das zur Konsequenz, daß die Retrieval-Struktur k-d-Baum letztlich nur auf einem eventuell kleinen Anteil der Attribute (Dimensionen), nämlich gerade den geordneten Attributen, beruhen kann. Bei Fällen, welche nur ungeordnete Attribute enthielten, war der k-d-Baum sogar nicht anwendbar. Wie Entscheidungsbäume zeigen, haben gerade ungeordnete, diskrete Attribute mit relativ großem Wertebereich eine sehr gute Diskriminierungseigenschaft, denn gerade diese werden vom Erzeugungsalgorithmus bevorzugt als Diskriminatorattribut verwendet.

Eine weitere unangenehme Eigenschaft von Fällen, die auch schon in Definition 2.1 zu entdecken war, ist, daß Fälle im allgemeinen nur unvollständig vorhanden sind, d. h. die Werte einiger Attribute sind *unbekannt*. Die Frage ist also, wie man Fälle in den "Fallraum" einordnen soll, wenn sie in einigen Dimensionen gar nicht einzuordnen sind. Eine einfache Möglichkeit wäre, wenn man, analog zu den Entscheidungsbäumen, Fälle mit unbekannten Wert im Diskriminatorattribut A in jede der gebildeten Partitionen der Fallmenge kopiert. Dieser Ansatz ist sicherlich korrekt, vergrößert jedoch die Fallmenge und den k-d-Baum enorm. Ausserdem wird auf den k-d-Baum ja auch ein intelligenterer Retrieval-Algorithmus als auf einen Entscheidungsbaum angewendet, sodaß dort ja noch Raum für Innovation besteht.

Desweiteren bietet es sich bei den bevorstehenden Änderungen an, mögliche Effizienzsteigerungen des k-d-Baums gleich mit einzubauen. Dies betrifft vor allen Dingen die in [30] vorgeschlagene Implementierung des k-d-Baums mit drei statt zwei Ästen, nämlich einem zusätzlichen Ast für die Gleichheit des Diskriminatorattributwerts mit dem Partitionswert. Das führt zu nichts anderem, als daß in Unterbäumen dieses "Gleichheitsastes" sich die Anzahl der zu bearbeitenden Attribute, sprich Dimensionen, um eins verringert.

Im folgenden wird nun die Struktur des k-d-Baumes festgelegt, wie er im weiteren benutzt wird.

Definition 3.1 (Erweiterter k-d-Baum) *Ein allgemeiner k-d-Baum*[2] $T(M)$ *zu* $M \subseteq FB$ *heißt* *erweiterter k-d-Baum*, *wenn die Funktionen* `Splitting` *wie folgt festgelegt sind:*

$$\texttt{Splitting}_j(M) = \begin{cases} (M_<, M_=, M_>, M_\perp) & \textit{falls } W_j \textit{ mit der Totalordnung } <_j \textit{ verträglich ist, d. h. } A_j \textit{ ist (total) geordnet} \\ (M_{v_1}, \ldots, M_{v_m}, M_\perp) & \textit{falls } A_j \textit{ nicht geordnet ist und } W_j = \{v_1, \ldots, v_m\} \end{cases}$$

mit

- $<_j$ *ist eine auf* W_j *definierte Totalordnung, die die Verträglichkeitsbedingung*

$$\forall v, w, z \in W_j : v <_j w <_j z \Rightarrow sim_j(v, w) \geq sim_j(v, z)$$

  erfüllt.

- $M_< = \{<w_1, \ldots, w_k, s> \in M | w_j <_j P\}$
- $M_= = \{<w_1, \ldots, w_k, s> \in M | w_j = P\}$
- $M_> = \{<w_1, \ldots, w_k, s> \in M | P <_j w_j\}$
- $M_\perp = \{<w_1, \ldots, w_k, s> \in M | w_j = \perp\}$
- `SelectPartitionValue`$(M, A_j) = P$ *mit* $P \in \Theta_j$, *wobei*

$$\Theta_j := \{w_j | <(w_1, \ldots, w_k), s> \in M, w_j \neq \perp\}$$

bzw.

[2] siehe Definition 2.5

- $\forall v \in W_j : M_v = \{<w_1, \ldots, w_k, s> \in M | w_j = v\}$
- $M_\perp = \{<w_1, \ldots, w_k, s> \in M | w_j = \perp\}$

Dabei ist A_j das mit `SelectAttr` *gewählte Diskriminatorattribut.*

Obige Definition baut wieder auf den allgemeinen k-d-Bäumen nach Def. 2.5 (s. a. Abb. 2.1 auf Seite 12) auf, wobei oben die Funktionen `Splitting` festgelegt werden. Ein Unterschied zu den "klassischen" k-d-Bäumen nach Def. 2.6 steckt genau in diesen Funktionen: Es können statt des einen Typs innerer Knoten nun zwei verschiedene Typen von inneren Knoten erzeugt werden:

1. Knoten für geordnete Diskriminatorattribute mit zusätzlichem Gleichheits- und *unbekannt*-Ast.

2. Knoten für ungeordnete Diskriminatorattribute, die ähnlich wie in Def. 2.10 eine Partitionierung nach allen möglichen Werteausprägungen des Diskriminatorattributs erlauben. Zusätzlich wird auch eine Partition für unbekannte Werte erzeugt.

Es gibt wieder mehrere Freiheitsgrade zur Berechnung des oben vorgestellten k-d-Baumes, die in den folgenden Abschnitten festgelegt werden:

1. Das Prädikat `Split`, welches entscheidet, ob ein weiterer Teilungschritt erfolgen soll

2. Die Funktion `SelectAttr` zur Auswahl des Diskriminatorattributs A_j

3. Im Falle von geordneten Attributen A_j:

   (a) Die Funktion `SelectPartitionValue` zur Wahl des Partitionswertes P

   (b) Die Totalordnung $<_j$. Auf deren Erzeugung wird im folgenden nicht näher eingegangen. Sie muß im allgemeinen mit der Modellierung der Wertetypen erzeugt werden.

   Die Entscheidung, ob ein Attribut geordnet oder ungeordnet ist, muß im allgemeinen schon bei der Modellierung der Domäne getroffen werden.

Angefügt werden soll noch, daß die in Kapitel 2 angeführten Prädikate bzw. Funktionen für `Split`, `SelectAttr` und `SelectPartitionValue` auch für k-d-Bäume nach obiger Definition verwendbar sind. Diese verwenden jedoch nur geordnete Attribute und schöpfen damit nicht alle Möglichkeiten der Baumerzeugung aus. Prädikate und Funktionen, die dies ermöglichen werden nun vorgestellt.

3.2.1 Generierung

Für die Generierung von k-d-Bäumen gibt es analog zu Kapitel 2 wieder eine ganze Reihe von Heuristiken, die sich grob in zwei Kategorien einteilen lassen:

- **Kategorie 1:** Heuristiken, die analog zu *Conceptual Clustering*-Systemen [32, 33, 34] versuchen, zusammengehörige (d. h. ähnliche) Fälle zu erkennen und diese in einem möglichst kleinen Teilbaum zusammenzufassen. Hier wird die in den Fällen angegebene Prolemlösung oder Klasse nicht berücksichtigt.

- **Kategorie 2:** Heuristiken, welche analog zu Entscheidungsbäumen auf eine schnelle Klassifikation des Anfragefalls zielen. Diese verwenden keine Ähnlichkeitsmaße.

Es werden nun einige neue Prädikate und Funktionen in Ergänzung zu Kapitel 2 eingeführt, welche die Generierung von k-d-Bäumen steuern.

Teilungssteuerung (`Split`)

Das Prädikat `Split` entscheidet abhängig von der noch zu bearbeitenden Menge von Fällen M, ob eine weitere Zergliederung von M nötig oder sinnvoll ist, oder ob alle Fälle aus M in *einem* Blattknoten (Bucket) zusammengefaßt abgespeichert werden sollen. Es bestimmt somit,

- wie fein die Zergliederung und Analyse der Fallbasis erfolgt, und
- anhand welches Kriteriums die Entscheidung für oder gegen eine Zergliederung fällt.

Eine einfache Realisierung wurde ja schon in Def. 2.6 aufgezeigt. Weitere ergeben sich aus der ähnlichkeitsbasierten Analyse der Fallmenge M bzw. aus der Analyse der Problemlösungs- oder Klassifizierungsinformation innerhalb der Fälle. Diese werden nun im folgenden angegeben.

Definition 3.2 (Splitting zur Bucketgröße b) *Zur Realisierung des Splitting zur Bucketgrösse b bezüglich der Fallmenge M wird* `Split` *wie folgt festgelegt:*

$$\texttt{Split}(M) \Leftrightarrow \texttt{Split}_b(M)$$

mit

$$\texttt{Split}_b(M) \Leftrightarrow |M| > b$$

Dies ist die aus Kapitel 2 (s. Def. 2.6) hinlänglich bekannte, einfache Realisierung. Sie ist vor allen Dingen dann von Nutzen, wenn

1. i. a. eine feste Anzahl n von Fällen gesucht wird mit $n \approx b$, und
2. wenig Information über die Struktur der Fallbasis in das zum Retrieval verwendete Ähnlichkeitsmaß kodiert wurde.

Definition 3.3 (Splitting zum mittleren Ähnlichkeitswert μ) *Zur Realisierung des Splitting zum mittleren Ähnlichkeitswert μ bezüglich der Fallmenge M wird* `Split` *wie folgt festgelegt:*

$$\texttt{Split}(M) \Leftrightarrow \texttt{Split}_{\overline{sim},\mu}(M)$$

mit[3]

$$\texttt{Split}_{\overline{sim},\mu}(M) \Leftrightarrow \frac{1}{|M|^2} \sum_{f_1, f_2 \in M} sim(f_1, f_2) \; < \mu$$

Diese Definition von `Split` eignet sich hervorragend für die ähnlichkeitsbasierte Erkennung von Fallclustern in der Fallbasis. Grundvorausetzung dazu ist natürlich, daß das Ähnlichkeitsmaß so gewählt ist, daß es die "interessanten" Eigenschaften der Fälle erkennen kann. Wie ein solches Ähnlichkeitsmaß zu einer gegebenen Menge von Fälle gefunden werden kann wird in Kapitel 4 vertieft.

Definition 3.4 (Klassifikationssplitting) *Zur Realisierung eines Klassifikationssplittings bezüglich der Fallmenge M wird* `Split` *wie folgt festgelegt:*

$$\texttt{Split}(M) \Leftrightarrow \texttt{Split}_\kappa(M)$$

mit

$$\texttt{Split}_\kappa(M) \Leftrightarrow \nexists \tilde{s} : \forall < w_1, \ldots, w_k, s > \in M : s = \tilde{s}$$

[3] Die folgende Definition von $\texttt{Split}_{\overline{sim},\mu}(M)$ ist für symmetrische und unsymmetrische Ähnlichkeitsmaße geeignet

Dies ist die Definition von `Split`, welche auch bei Entscheidungsbäumen verwendet wird. Diese zielt auf eine schnelle Klassifikation des Anfragefalls. Da sich die Retrievalalgorithmen von k-d- und Entscheidungsbäumen aber grundlegend unterscheiden (s. Kap. 2) macht die oben vorgeschlagene Einstellung von `Split` nur Sinn, wenn das verwendete Ähnlichkeitsmaß "kompatibel" zur klassengesteuerten Generierung von k-d-Bäumen ist. Die Generierung eines kompatiblen Ähnlichkeitsmaßes wird in Kapitel 4 angegangen.

Die oben angeführten Möglichkeiten für `Split` können natürlich auch miteinander kombiniert werden. Einige sinnvolle Kombinationen, deren Bedeutungen leicht ersichtlich sind, wären:

$$\mathtt{Split}_{\overline{sim},\mu,b}(M) \Leftrightarrow \mathtt{Split}_{\overline{sim},\mu}(M) \wedge \mathtt{Split}_{b}(M) \tag{3.1}$$

oder auch

$$\mathtt{Split}_{\kappa,b}(M) \Leftrightarrow \mathtt{Split}_{\kappa}(M) \wedge \mathtt{Split}_{b}(M) \tag{3.2}$$

Wahl des Diskriminatorattributs (`SelectAttr`) und Partitionswertes (`SelectPartitionValue`)

Die Funktionen `SelectAttr` und `SelectPartitionValue` wählen das Diskriminatorattribut und den Partitionswert aus und "erkennen" somit die Struktur der Fallbasis, d. h. in ihnen wird festgelegt, nach welcher Heuristik die Fallbasis strukturiert wird. Stellt man sich die Fallbasis als einen Raum mit je einem Punkt pro Fall vor, so geben obige Funktionen die Strategie der Zerteilung des Raums in Unterräume an. In ihnen ist also (implizit) festgelegt, welche Fälle zusammen gehören, also z. B. ähnlich sind oder die gleiche Problemlösung/Klasse haben.

Die Funktion `SelectAttr` bestimmt das Attribut A_j, nach welchem die zu bearbeitende Fallmenge M zergliedert wird. Bei der Wahl von A_j müssen zwei verschiedene Situationen unterschieden werden:

1. Das Attribut A_j ist mit einer Totalordnung $<_j$ verträglich. Hier bietet sich eine Unterteilung von M in die vier Teile $M_<$, $M_=$, $M_>$ und $M_\perp$ wie in Def. 3.1 bezüglich eines Partitionswerts P an, wobei P von der Funktion `SelectPartitionValue`(M, A_j) geliefert wird. Dieses unterstützt in idealer Weise die Idee der BOB- und BWB-Tests (s. Def. 2.8 & 2.7).

2. Das Attribut A_j ist ungeordnet. Eine Unterteilung von M muß jede Werteausprägung von A_j erfassen, d. h. es werden die Partitionen $M_{v_1}, \ldots, M_{v_m}$ gebildet, wenn $W_j = \{v_1, \ldots, v_m\}$ ist. Das heißt aber auch, daß die Funktion `SelectPartitionValue` in diesem Fall nicht benötigt wird. Dies erfordert aber auch eine Anpassung der BOB- und BWB-Tests sowie des gesamten Retrieval-Algorithmuses, da er nach [5] nur geordnete Attribute unterstützt. Dies wird im nachfolgenden Abschnitt erledigt.

Betrachten wir dazu ein (sehr) kleines Beispiel mit zwei Fällen mit jeweils zwei Attributen A_1 und A_2 und den Wertebereichen $W_1 = \{a, b\}$ und $W_2 = \Re$. Dabei sei A_1 ungeordnet und A_2 (wie üblich) geordnet. Möchten wir nun für die Menge $M = \{(a, 1), (b, 2)\}$ ein Diskriminatorattribut auswählen, so muß eine Entscheidung getroffen werden, ob A_1 oder A_2 besser ist. Es muß also eine "ist besser"-Relation durch die Funktion `SelectAttr` implementiert werden. Dies läßt sich analog zu Ähnlichkeiten und Ähnlichkeitsmaßen wieder durch eine Bewertungsfunktion realisieren, die *vergleichbare* Bewertungen für geordnete und ungeordnete Attribute berechnet. Für das Beispiel müssen also die Bewertungen des Vergleichs von a und b vergleichbar sein mit Bewertungen des Vergleichs von 1 und 2. Die Struktur der Funktion `SelectAttr` läßt sich also wie folgt beschreiben:

Algorithmus 3.1 (SelectAttr(M))

```
max := −∞
FOR i = 1 TO k DO
    IF isOrdered(A_i) THEN
        /* A_i ist geordnet */
        P := SelectPartitionValue(M, A_i)
        score := CalculateScore_ordered(M, A_i, P)
```

```
    ELSE
        /* A_i ist nicht geordnet */
        score := CalculateScore_unordered(M, A_i)
    ENDIF
    IF score > max THEN
        max := score
        best := i
    ENDIF
ENDFOR
RETURN A_best
```

Das Prädikat `isOrdered` testet, ob das übergebene Attribut geordnet ist, d. h. ob zum zugrundeliegenden Wertetyp eine bekannte, mit dem zugehörigen lokalen Ähnlichkeitsmaß verträgliche Totalordnung $<$ existiert. Deshalb kann dieses Prädikat schon bei der Definition der Fallstruktur definiert werden.

Die Funktionen $\texttt{CalculateScore}_X$ berechnen eine Bewertung der durch die Parameter definierten Partitionierung der Fallmenge M, anhand derer das Diskriminatorattribut ausgewählt wird. Dabei ist es natürlich wichtig, daß die Ergebnisse von $\texttt{CalculateScore}_{ordered}$ mit denen von $\text{CalculateScore}_{unordered}$ vergleichbar sind, wenn in einem Fall geordnete und ungeordnete Attribute vorkommen können. Um diese Vergleichbarkeit zu gewährleisten basieren beide Funktionen i. a. auf der gleichen Bewertungsmethodik. Einige Methodiken, z. B. die in Kapitel 2 vorgestellten (Interquartilsabstand und Abstand in der größten Lücke), erlauben jedoch nur geordnete Attribute.

Es werden nun einige Möglichkeiten der Definition von `CalculateScore` angegeben, welche in der Regel in die Teile für geordnete und für ungeordnete Attribute zerfällt. Die in Kapitel 2 angeführten Möglichkeiten werden der Vollständigkeit halber nochmals aufgeführt. Im folgenden gelte immer

$$\begin{array}{rcl} \Theta_i & = & \{w_i | <w_1, \ldots, w_k, s> \in M, w_i \neq \bot\} \text{ mit } \Theta_i = \{v_1, \ldots, v_m\} \text{ sowie} \\ \psi_\bot & = & \begin{cases} 1 & \text{falls } \exists f = <w_1, \ldots, w_k, s> \in M \text{ mit } w_i = \bot \\ 0 & \text{sonst} \end{cases} \end{array} \tag{3.3}$$

Falls Attribut A_i geordnet ist, sei o. B. d. A. $v_1 < \ldots < v_m$. Desweiteren werden gleich die passenden Definitionen von `SelectPartitionValue` mitangegeben.

Definition 3.5 (Attributselektion durch Interquartilsabstand & Median) *Für die Fallmenge M, das Attribut A_i und $\Theta_i, \psi_\bot$ wie in Gl. 3.3 ist die Attributselektion durch Interquartilsabstand und Median durch die Funktionen* $\texttt{CalculateScore}^{(iqd)}_{ordered}$ *und* $\texttt{SelectPartitionValue}^{(iqd)}$ *wie folgt definiert*[4]:

$$\texttt{CalculateScore}^{(iqd)}_{ordered} = \begin{cases} 1 - sim_i(v_{q_1}, v_{q_3}) & \textit{falls } |\Theta_i| + \psi_\bot \geq 2 \\ -\infty & \textit{sonst} \end{cases}$$

mit $q_2 = \lfloor \frac{1+|\Theta_i|}{2} \rfloor$, $q_1 = \lfloor \frac{1+q_2}{2} \rfloor$ *und* $q_3 = q_1 + q_2 - 1$. *Dabei ist* sim_i *das lokale Ähnlichkeitsmaß von Attribut* A_i. *Die Funktion* $\texttt{CalculateScore}^{(iqd)}_{unordered}$ *ist nicht definiert. Desweiteren ist*

$$\texttt{SelectPartitionValue}^{(iqd)} = v_{q_2} \in \Theta_i$$

Die Eigenschaften obiger Heuristik sind ebenso wie die der folgenden schon in Kapitel 2 erläutert worden.

[4] Die Argumente der zu definierenden Funktionen wurden der Einfachheit halber nicht notiert. Sie verbleiben wie in Alg. 3.1

Definition 3.6 (Attributselektion durch maximalen Abstand) *Für die Fallmenge M, das Attribut A_i und $\Theta_i, \psi_\perp$ wie in Gl. 3.3 ist die* <u>*Attributselektion durch maximalen Abstand*</u> *durch die Funktionen* $\texttt{CalculateScore}^{(maxd)}_{ordered}$ *und* $\texttt{SelectPartitionValue}^{(maxd)}$ *wie folgt definiert*[4]*:*

$$\texttt{CalculateScore}^{(maxd)}_{ordered} = \begin{cases} 1 - sim_i(v_{p-1}, v_p) & \textit{falls } |\Theta_i| + \psi_\perp \geq 2 \textit{ und } v_p = P \\ -\infty & \textit{sonst} \end{cases}$$

Dabei ist sim_i das lokale Ähnlichkeitsmaß von Attribut A_i. Die Funktion $\texttt{CalculateScore}^{(maxd)}_{unordered}$ *ist nicht definiert. Desweiteren ist*

$$\texttt{SelectPartitionValue}^{(maxd)} = v_{t+1} \in \Theta_i, \textit{falls } sim_i(v_t, v_{t+1}) = \min\{sim_i(v_u, v_{u+1}) | 1 \leq u < m\}$$

Obige zwei Heuristiken können also nur geordnete Attribute verarbeiten. Sie können jedoch so abgewandelt werden, daß sie auch ungeordnete Attribute verarbeiten können:

Definition 3.7 (Attributselektion durch Median & mittlere Ähnlichkeit) *Für die Fallmenge M, das Attribut A_i und $\Theta_i, \psi_\perp$ wie in Gl. 3.3 ist die* <u>*Attributselektion durch Median und mittlere Ähnlichkeit*</u> *durch d. Funktionen* $\texttt{CalculateScore}^{(medsim)}_{ordered}$, $\texttt{CalculateScore}^{(medsim)}_{unordered}$ *und* $\texttt{SelectPartitionValue}^{(medsim)}$ *wie folgt definiert*[4]*:*

$$\texttt{CalculateScore}^{(medsim)}_{ordered} = \begin{cases} \frac{\overline{sim}_{<_i} + \overline{sim}_{=} + \overline{sim}_{>_i} + \overline{sim}_{\perp}}{4} & \textit{falls } |\Theta_i| + \psi_\perp \geq 2 \\ -\infty & \textit{sonst} \end{cases}$$

mit

$$\overline{sim}_{\bigcirc} = \frac{1}{|M_{\bigcirc}|^2} \sum_{f_1, f_2 \in M_{\bigcirc}} sim(f_1, f_2) \qquad \textit{für } \bigcirc \in \{<_i, >_i, =, \perp\} \qquad (\star)$$

und

- $M_{\bigcirc} = \{< w_1, \ldots, w_k, s > \in M | w_i \bigcirc P\}$ *für* $\bigcirc \in \{<_i, >_i, =\}$
- $M_\perp = \{< w_1, \ldots, w_k, s > \in M | w_i = \perp\}$

Die Funktion $\texttt{CalculateScore}^{(medsim)}_{unordered}$ *ist wie folgt definiert:*

$$\texttt{CalculateScore}^{(medsim)}_{unordered} = \frac{1}{1 + |\Theta_i|} (\overline{sim}_\perp + \sum_{v \in \Theta_i} \overline{sim}_v)$$

mit $\overline{sim}_\perp$ *wie in* $(\star)$ *und*

$$\overline{sim}_v = \frac{1}{|M_v|^2} \sum_{f_1, f_2 \in M_v} sim(f_1, f_2) \qquad \textit{für } M_v = \{< w_1, \ldots, w_k, s > \in M | w_i = v\}$$

Desweiteren ist

$$\texttt{SelectPartitionValue}^{(medsim)} = P = v_{median} \textit{ mit } median = \lfloor \tfrac{1 + |\Theta_i|}{2} \rfloor$$

Definition 3.8 (Attributselektion durch maximalen Abstand & mittlere Ähnlichkeit) *Für die Fallmenge M, das Attribut A_i und $\Theta_i, \psi_\perp$ wie in Gl. 3.3 ist die Attributselektion durch maximalen Abstand und mittlere Ähnlichkeit durch die Funktionen* $\texttt{CalculateScore}^{(maxdsim)}_{ordered}$ *sowie* $\texttt{CalculateScore}^{(maxdsim)}_{unordered}$ *und* $\texttt{SelectPartitionValue}^{(maxdsim)}$ *wie folgt definiert*[4]*:*

- $\texttt{CalculateScore}^{(maxdsim)}_{ordered} = \texttt{CalculateScore}^{(medsim)}_{ordered}$ *aus Def. 3.7*
- $\texttt{CalculateScore}^{(maxdsim)}_{unordered} = \texttt{CalculateScore}^{(medsim)}_{unordered}$ *aus Def. 3.7*

- $\texttt{SelectPartitionValue}^{(maxdsim)} = v_{t+1} \in \Theta_i$ *so, daß* $sim_i(v_t, v_{t+1})$ *minimal ist. Dabei ist* sim_i *das lokale Ähnlichkeitsmaß von Attribut* A_i.

Definition 3.7 und 3.8 unterscheiden sich nur in der Funktion `SelectPartitionValue`. Sie sind die abgewandelten Versionen von Def. 3.5 bzw. 3.6.

Dies waren nun die schon in Kapitel 2 vorgestellten Auswahlheuristiken, wie sie prinzipiell auch in [5, 9, 15, 14] vorgestellt wurden. In der obigen Version wurden diese aber um die Bewertung ungeordneter Attribute und damit um die Möglichkeit der Erzeugung von inneren Knoten im k-d-Baum bereichert, welche nach einem ungeordneten Attribut diskriminieren. Es sollen nun zwei neue Auswahlheuristiken vorgestellt werden.

Definition 3.9 (Attributselektion durch optimale Fallcluster) *Für die Fallmenge* M*, das Attribut* A_i *und* $\Theta_i, \psi_\perp$ *wie in Gl. 3.3 ist die* <u>*Attributselektion durch optimale Fallcluster*</u> *durch die Fkt.* $\texttt{CalculateScore}^{(opt)}_{ordered}$, $\texttt{CalculateScore}^{(opt)}_{unordered}$ *und* $\texttt{SelectPartitionValue}^{(opt)}$ *wie folgt definiert*[4]*:*

- $\texttt{CalculateScore}^{(opt)}_{unordered} = \texttt{CalculateScore}^{(medsim)}_{unordered}$ *aus Def. 3.7*

- $\texttt{CalculateScore}^{(opt)}_{ordered}$ *und* $\texttt{SelectPartitionValue}^{(opt)}$ *sind wie folgt definiert:*

$$\texttt{CalculateScore}^{(opt)}_{ordered} = \begin{cases} \max\{\frac{\overline{sim}^v_{<_i} + \overline{sim}^v_{=} + \overline{sim}^v_{>_i} + \overline{sim}_\perp}{4} | v \in \Theta_i\} & \textit{falls } |\Theta_i| + \psi_\perp \geq 2 \\ -\infty & \textit{sonst} \end{cases}$$

mit $\overline{sim}_\perp$ *wie in* $(\star)$ *von Def. 3.7 und*

$$\overline{sim}^v_{\bigcirc} = \frac{1}{|M^v_{\bigcirc}|^2} \sum_{f_1, f_2 \in M^v_{\bigcirc}} sim(f_1, f_2) \qquad \textit{für } \bigcirc \in \{<_i, >_i, =\}$$

sowie $M^v_{\bigcirc} = \{< w_1, \ldots, w_k, s > \in M | w_i \bigcirc v\}$ *für* $\bigcirc \in \{<_i, >_i, =\}$*. Desweiteren ist*

$$\texttt{SelectPartitionValue}^{(opt)} = P \in \Theta_i \textit{ so, daß}$$

$\frac{\overline{sim}^P_{<_i} + \overline{sim}^P_{=} + \overline{sim}^P_{>_i} + \overline{sim}_\perp}{4}$ *maximal ist.*

Der Begriff der *optimalen Cluster* bezieht sich also auf die größte mittlere Ähnlichkeit einer Fallmenge. Obiges Selektionskriterium unterscheidet sich an nur einem, aber entscheidenden Punkt von den in Definition 3.7 und 3.8 angegebenen Kriterien: Die Suche nach einem Partitionswert (durch die Funktion `SelectPartitionValue`) wird *nicht* durch eine zusätzliche Heuristik – wie Median oder Randwert an der größten Lücke – bestimmt. Vielmehr wird der Partitionswert so gewählt, daß die mittlere Ähnlichkeit der (maximal) vier entstehenden Fallmengen *(Cluster)* maximal wird. In der Praxis hat sich nämlich gezeigt, daß sich die Zweiteilung Partitionswertauswahl – Bewertung der Partitionen, nur dann lohnt, wenn beide Teile eine gemeinsame Heuristik als Grundlage haben. Dies ist jedoch bei Median & mittlerer Ähnlichkeit sowie maximalem Abstand & mittlerer Ähnlichkeit nicht der Fall.

Die letzten drei Auswahlheuristiken werden auch als *a-posteriori*-Abschätzungen bezeichnet, weil sie, im Gegensatz zu den in Definition 3.5 und 3.6 vorgestellten, die Güte des Attributs als Diskriminator anhand der (bezüglich des gewählten Partitionswerts) entsprechenden Partitionen der Fallmenge M beurteilen. Es ist also immer nötig, meist mehrere Partitionierungen der Fallmenge zu berechnen, auch wenn nur immer eine benötigt wird. Dies legt nahe, die Funktion $\texttt{Splitting}_j(M, A_j)$ (s. Def. 3.1) nicht mehr separat zu implementieren, da man sich ja die Partitionierung des gewählten Diskriminatorattributs während des Auswahlprozesses merken kann.

Ein Handicap der letzten drei Heuristiken ist der erheblich gestiegene Rechenzeitbedarf, da eine große Anzahl von Ähnlichkeitsberechnungen zwischen Fällen durchgeführt wird. Naheliegend ist also die Berechnung einer Ähnlichkeitsmatrix wie in [14] vorgeschlagen wurde. Der Rechenzeitbedarf des Ansatzes läßt sich jedoch noch weiter verringern, wenn man bedenkt, daß für die Berechnung der Ähnlichkeitsmatrix alle Fälle miteinander verglichen werden, obwohl bei der Berechnung von `SelectAttr` immer nur ein Teil der Fälle miteinander verglichen wird. Ein besserer Ansatz ist also, die Ähnlichkeitsmatrix nicht explizit zu berechnen, sondern nur "nach Bedarf" zu berechnen. Sie wird somit als eine Art Cache für Ähnlichkeitswerte benutzt.

Die Attributselektion durch optimale Fallcluster ist im Vergleich zu allen anderen angegebenen Selektionskriterien die mit Abstand aufwendigste Methode, da z. B. im Vergleich zur Selektion durch maximalen Abstand & mittlere Ähnlichkeit nicht nur eine Partitionierung der Fallmenge M berechnet werden muß, sondern derer m berechnet werden müssen[5] mit $m = |\Theta_i|$ (s. Gl. 3.3). Wenn man bedenkt, daß dies für bis zu k Attribute geschehen muß, sieht man leicht ein, daß hier ein nicht unerheblicher Rechenaufwand entsteht. Deshalb wird nun im folgenden ein Algorithmus angegeben, welcher durch geschicktes Speichern von Ähnlichkeitswerten und Teilpartitionen den Rechenzeitbedarf erheblich senkt.

Der Algorithmus berechnet für ein geordnetes Attribut A_i das optimale Fallcluster, implementiert also die Funktionen $\texttt{CalculateScore}^{(opt)}_{ordered}$ und $\texttt{SelectPartitionValue}^{(opt)}$. Desweiteren berechnet der Algorithmus auch die zugehörige Partitionierung der Fallmenge M, implementiert also (für geordnete Attribute) die Funktion $\texttt{Splitting}_i$. Der Algorithmus benötigt folgende Eingabeparameter[6]:

- $\Theta_i = \{v_1, \ldots, v_m\} \subseteq W_i$ mit $v_1 < \ldots < v_m$ analog zu Gl. 3.3
- $M_{v_1}, \ldots, M_{v_m}$ mit $M_{v_t} = \{<w_1, \ldots, w_k, s>\in M | w_i = v_t\}$
- $M_\perp = \{<w_1, \ldots, w_k, s>\in M | w_i = \perp\}$
- $sim_{v_1}, \ldots, sim_{v_m}, sim_\perp \in \Re_0^+$ mit

$$sim_v = \sum_{f_1, f_2 \in M_v} sim(f_1, f_2) \qquad \text{für } v \in \{v_1, \ldots, v_t, \perp\}$$

Die im Algorithmus berechneten Fallmengen $M_t^<$ und $M_t^>$ enthalten die Fälle $<w_1, \ldots, w_k, s>\in M$ mit $w_i < v_{t+1}$ bzw. $w_i > v_{t-1}$. $sim_t^<$ und $sim_t^>$ sind die zugehörigen akkumulierten Ähnlichkeitswerte der enthaltenen Fälle, wenn man diese innerhalb der Menge mit dem Ähnlichkeitsmaß sim vergleicht. Als Rückgabe liefert der Algorithmus

1. Die mittlere Ähnlichkeit max der besten Fallcluster von Attribut A_i
2. Den zugehörigen Partitionswert P
3. Das Splitting der Menge M, falls A_i als Diskriminatorattribut ausgewählt wird

Algorithmus 3.2 (OptimalCluster($\Theta_i, M_{v_1}, \ldots, M_{v_m}, M_\perp, sim_{v_1}, \ldots, sim_{v_m}, sim_\perp$))

```
M_0^< := ∅
M_{m+1}^> := ∅
sim_0^< := 0
sim_{m+1}^> := 0
FOR t = 1 TO m DO
    u := m − t + 1 /* Herabzählender Index */
    M_t^< := M_{t−1}^< ∪ M_{v_t} /* <-Partitionen */
    M_u^> := M_{u+1}^> ∪ M_{v_u} /* >-Partitionen */
    sim_t^< := sim_{t−1}^< + sim_{v_t} /* <-Intra-Partitions-Ähnlichkeit */
    sim_u^> := sim_{u+1}^> + sim_{v_u} /* >-Intra-Partitions-Ähnlichkeit */
```

[5]Natürlich nur bei geordneten Attributen. Bei ungeordneten Attributen ist der Aufwand gleich denen in Def. 3.7 & 3.8 angegebenen Selektionskriterien

[6]M bezeichnet wieder aktuell zu bearbeitende Fallmenge $\subseteq FB$

```
    FOR f_1 ∈ M_{v_t} DO
        FOR f_2 ∈ M^<_{t-1} DO
            /* Berechne <-Inter-Partitions-Ähnlichkeit */
            sim^<_t := sim^<_t + sim(f_1, f_2) + sim(f_2, f_1)
        ENDFOR
    ENDFOR
    FOR f_1 ∈ M_{v_u} DO
        FOR f_2 ∈ M^>_{u+1} DO
            /* Berechne >-Inter-Partitions-Ähnlichkeit */
            sim^>_u := sim^>_u + sim(f_1, f_2) + sim(f_2, f_1)
        ENDFOR
    ENDFOR
ENDFOR
/* Bestimmung der optimalen Cluster */
max := -∞
FOR t = 1 TO m DO
    score := (sim^<_{t-1} + sim_{v_t} + sim^>_{t+1} + sim_⊥) / (|M^<_{t-1}|^2 + |M_{v_t}|^2 + |M^>_{t+1}|^2 + |M_⊥|^2)
    IF score > max THEN
        max := score
        /* Merke auch Splitting & Partitionswert */
        splitting := (M^<_{t-1}, M_{v_t}, M^>_{t+1}, M_⊥)
        P := v_t
    ENDIF
ENDFOR
RETURN max, P, splitting
```

Die bisher vorgestellten Selektionskriterien basierten alle auf einem der lokalen Ähnlichkeitsmaße sim_i oder auf dem globalen Ähnlichkeitsmaß sim, welche zur Implementierung von verschiedenen Strukturerkennungsheuristiken verwendet wurden. Wenn auch die Vorgehensweisen verschieden waren, waren trotzdem die *Grundannahme* und das *Ziel* immer das Gleiche:

- Grundannahme ist immer, daß es das verwendete lokale oder globale Ähnlichkeitsmaß ermöglicht, zusammengehörige Fälle, d. h. ähnliche Fälle, zu erkennen.
- Ziel ist immer, ähnliche Fälle in der k-d-Baumstruktur umso näher[7] im Baum zu speichern, je ähnlicher sich die Fälle sind.

Das Selektionskriterium *Auswahl durch Entropie*, welches weiter unten vorgestellt wird, unterscheidet sich jedoch grundlegend von den bisher vorgestellten Kriterien:

- Grundannahme ist nun, daß Fälle, die in *ausgewählten* Attributen die gleiche Werteausprägung vorweisen, auch die gleiche Problemlösung bzw. Klasse haben.
- Ziel ist es nun, die durchschnittliche Anzahl der zu betrachtenden[8] Attribute zur eindeutigen Bestimmung der Problemlösung oder Klasse so gering wie möglich zu halten.

Hier ergibt sich also ein entscheidender Unterschied: Es wird kein Ähnlichkeitsmaß verwendet bzw. berücksichtigt, d. h. daß i. a. der oder die zur Bestimmung der Problemlösung/Klasse herangezogenen Fälle nicht der oder die ähnlichsten zum Anfragefall sein müssen. Ein Ansatz, welcher diese Lücke zu schließen versucht wird im nachfolgenden Kapitel vorgestellt. Nun soll jedoch erst einmal das Attributauswahl mit Entropie festgelegt werden.

[7] Die Nähe bzw. Distanz bezieht sich auf die übliche Distanzbestimmung in Bäumen

[8] In der Begriffswelt der Entscheidungsbäume wird i. a. von *Test* gesprochen

Definition 3.10 (Attributselektion durch Entropie) *Für die Fallmenge M, das Attribut A_i und $\Theta_i, \psi_\perp$ wie in Gleichung 3.3 ist die Attributselektion durch Entropie durch die Funktionen* $\texttt{CalculateScore}^{(ent)}_{ordered}$, $\texttt{CalculateScore}^{(ent)}_{unordered}$ *und* $\texttt{SelectPartitionValue}^{(ent)}$ *wie folgt definiert*[9]*:*

$$\texttt{CalculateScore}^{(ent)}_{ordered} = \begin{cases} 1 - \min\{\frac{|M^v_{<_i}|H(M^v_{<_i})+|M^v_{=}|H(M^v_{=})+|M^v_{>_i}|H(M^v_{>_i})+|M_\perp|H(M_\perp)}{|M|} | v \in \Theta_i\} \\ \textit{falls } |\Theta_i| + \psi_\perp \geq 2 \\ \\ -\infty \textit{ sonst} \end{cases}$$

mit

- $M^v_{\bigcirc} = \{< w_1, \ldots, w_k, s > \in M | w_i \bigcirc v\}$ *für* $\bigcirc \in \{<_i, >_i, =\}$
- $M_\perp = \{< w_1, \ldots, w_k, s > \in M | w_i = \perp\}$

Die Funktion $CalculateScore^{(ent)}_{unordered}$ ist wie folgt definiert:

$$CalculateScore^{(ent)}_{unordered} = 1 - \frac{1}{|M|}(|M_\perp|H(M_\perp) + \sum_{v \in \Theta_i} |M_v|H(M_v))$$

für $M_v = \{< w_1, \ldots, w_k, s > \in M | w_i = v\}$*. Desweiteren ist*

$$\texttt{SelectPartitionValue}^{(ent)} = P \in \Theta_i \textit{ so, daß}$$

$\frac{|M^P_{<_i}|H(M^P_{<_i})+|M^P_{=}|H(M^P_{=})+|M^P_{>_i}|H(M^P_{>_i})+|M_\perp|H(M_\perp)}{|M|}$ *minimal ist.*

Es kann also sowohl nach geordneten wie auch nach ungeordneten Attributen diskriminiert werden. Für letztere ist die Bewertung der entstehenden Partitionen analog zu dem im Entscheidungsbaum verwendeten Maß, bis auf den Unterschied, daß auch die unbekannte "Werteausprägung" $\perp$ berücksichtigt wird. Die Fälle mit unbekanntem Wert beim gewählten Diskriminatorattribut werden also nicht in alle Partitionen der Fallmenge kopiert, sondern werden vielmehr in einer eigenen Partition gesammelt. Dies ist durch die unterschiedlichen Retrieval-Algorithmen von Entscheidungsbaum und k-d-Baum möglich, da bei letzterem durch das Backtracking auch ein Durchsuchen der unbekannt-Partition möglich ist, obwohl vielleicht der Wert des betreffenden Diskriminatorattributs bekannt ist. Dies steht im Gegensatz zum Entscheidungsbaum, in welchem ja nur top-down gesucht wird.

Für geordnete Attribute werden bis zu vier Teilpartitionen gebildet, welche dann analog zu den m Teilpartitionen von ungeordneten Attributen mit Hilfe der Rückschlußentropie bewertet werden. Die Beschränkung auf vier Teilpartitionen unterstützt wieder die Idee des Teilbaum-Ausschlusses mit Hilfe der BOB- und BWB-Tests, da ja immer noch mit Hilfe eines Ähnlichkeitsmaßes gesucht wird! Da aber das Entropie-Maß nichts über Ähnlichkeiten der Fälle weiß, ist zu erwarten, daß das Retrieval in dem so erzeugten Baum aufwendiger ist als in Bäumen, in welchem bewußt ähnliche Fälle zusammengelegt wurden. Im folgenden soll noch eine Übersicht über sinnvolle Parameter-Einstellungen bei der k-Baum-Generierung angegeben werden.
Tabelle 3.1 gibt die mögliche Wahl des Prädikats `Split` an, während Tabelle 3.2 die mögliche Wahl der Funktion `CalculateScore` angibt. Die Möglichkeiten sind jeweils in Abhängigkeit von den in den Fällen vorhandenen Attributtypen (geordnet, ungeordnet) aufgetragen. Desweiteren wird Rücksicht darauf genommen, ob ein *sinnvolles* globales Ähnlichkeitsmaß sim definiert wurde oder nicht. Die lokalen Ähnlichkeitsmaße sim_i sollten immer "sinnvolle" Werte liefern.

[9] Die Funktion H ist die in Gl. 2.17 festgelegte Entropie

Nr.	Fallstruktur				Teilungssteuerung		
	Attribute		globales	Klasse			
	geordnet	ungeordnet	*sim*	vorhanden	$\texttt{Split}_b$	$\texttt{Split}_\kappa$	$\texttt{Split}_{\overline{sim},\mu}$
1	$\times$			$(\times)$	$\surd$	$(\surd)$	
2		$\times$		$(\times)$	$\surd$	$(\surd)$	
3	$\times$	$\times$		$(\times)$	$\surd$	$(\surd)$	
4	$\times$		$\times$	$(\times)$	$\surd$	$(\surd)$	$\surd$
5		$\times$	$\times$	$(\times)$	$\surd$	$(\surd)$	$\surd$
6	$\times$	$\times$	$\times$	$(\times)$	$\surd$	$(\surd)$	$\surd$

$\times$= vorhanden bzw. sinnvoll definiert $\surd$= möglich

Tabelle 3.1: Teilungssteuerungs-Parameter bei der k-d-Baum-Generierung

Nr.	Retrievalzeiten					
	iqd	*maxd*	*medsim*	*maxdsim*	*opt*	*ent*
1	$\bigcirc$	$+$	$-$	$-$	$-$	$(\bigcirc)$
2	$-$	$-$	$-$	$-$	$-$	$(\bigcirc)$
3	$\bigcirc$	$\bigcirc/+$	$\bigcirc/+$	$\bigcirc/+$	$+$	$(\bigcirc)$
4	$-$	$-$	$+$	$+$	$+$	$(\bigcirc/+)$
	k-d-Baum Generierungszeit					
	$+$	$+$	$-$	$-$	$-$	$+$

$-$= schlecht $\bigcirc/+$= mittel $+$= gut

Tabelle 3.2: Diskriminator-Auswahlstrategien bei der k-d-Baum-Generierung

3.2.2 Retrieval

Nachdem wir im letzten Unterabschnitt gesehen haben, wie ungeordnete Attribute sowie Fälle mit unbekannten Werteausprägungen in die k-d-Baum-Struktur einfließen, ist es nun von Interesse, wie das Retrieval in der so modifizierten Baumstruktur vonstatten gehen soll. Speziell ist an folgenden Punkten eine Modifizierung der Retrieval-Prozedur nötig:

1. Die Bounds-Tests, welche für die Backtracking-Steuerung und den Teilbaum-Ausschluß verwendet wurden, verwenden (s. Def. 2.7 & 2.8) für jeden Teilbaum eine Beschreibung der Attribute, sprich Dimensionen, welche den jeweils kleinsten und größten Wert des Attributs enthält. Diese Beschreibung ist nun aber nur noch partiell vorhanden, da

   (a) Attributewerte unbekannt sein können und sich somit nicht mehr durch die Angabe einer unteren und oberen Schranke der Werteausprägungen beschreiben lassen. Desweiteren sind

   (b) nicht alle Attribute (total) geordnet. Der in dem Teilbaum enthaltene Wertebereich eines Attributs läßt sich also nicht mehr durch die Angabe von Schranken beschreiben. Desweiteren zerstören diese Attribute (Dimensionen) die "räumliche" Vorstellung des Fallraums, da der durch die geordneten Attribute beschriebene Fallraum sich mit jeder Werteausprägung eines ungeordneten Attributs dupliziert.

   Andererseits ist es natürlich möglich, zu jedem Teilbaum die enthaltenen Werteausprägungen der ungeordneten Attribute zu bestimmen. Das weckt die Hoffnung, daß man auch diese

Information zur Retrieval-Beschleunigung anwenden kann.

2. Da wir den k-d-Baum auf die Struktur der vier möglichen Sohnknoten $<, >, =$ und $\perp$ umgestellt haben, ist es nötig die Retrieval-Prozedur auf die geänderte Baumstruktur anzupassen. Desweiteren muß das Problem gelöst werden, daß im Anfragefall einige Werteausprägungen unbekannt sind. Ist eine dieser unbekannten Attribute ein Diskriminatorattribut im Baum, so ist keine Richtungsentscheidung in $<, >$ oder $=$[10] möglich. Hier muß also eine Reihenfolge der Sohnknoten generiert werden, in welcher diese zu durchsuchen sind.

Anpassung der Bounds-Tests

Wie wir schon in Abb. 2.6 auf Seite 19 gesehen haben, konstruieren die Bounds-Tests mit Hilfe der Dimensionsgrenzen $((l_1, u_1), \ldots, (l_k, u_k))$ Hilfsfälle F_{min} bzw. $F^{(i)}_{min}$ und $F^{(i)}_{max}$, für welche getestet wird, ob sie inner- oder außerhalb der Hyperkugel mit dem Ähnlichkeitsradius r liegen. Da wir nun aber wie oben schon erläutert wurde nur eine partielle Beschreibung $((l_{i_1}, u_{i_1}), \ldots, (l_{i_m}, u_{i_m}))$ mit $1 \leq i_1 < \ldots < i_m \leq k$ zur Verfügung haben, können die Hilfsfälle nicht in jedem Attribut/Dimension so konstruiert werden, daß sie "auf der Grenze liegen", weil diese ja gerade nicht bekannt ist. Um die Korrektheit und Vollständigkeit des Suchalgorithmuses nicht zu verlieren müssen wir also für die Dimensionen, über die wir keine konkreten Angaben haben, so pessimistisch wie möglich sein, wenn es darum geht die Suche abzubrechen (BWB-Test) bzw. einen Teilbaum nicht zu durchsuchen (BOB-Test). Dazu wird der Wertebereich W_j jedes Attributs A_j um einen Wert $*$ erweitert, welcher als *don't care* zu interpretieren ist, d. h. wir verlangen für jedes lokale Ähnlichkeitsmaß sim_j:

$$\forall v \in W_j \cup \{\perp, *\} : sim_j(v, *) = sim_j(*, v) = 1 \tag{3.4}$$

Wenn wir also für alle Attribute/Dimensionen, für die wir keine Grenzen angeben können, den Wert $*$ in die entsprechenden Attribute der Hilfsfälle einsetzen, dann sind die Bounds-Tests wieder verwendbar und die Korrektheit und Vollständigkeit des Suchalgorithmuses bleiben erhalten. Hier haben wir aber eine implizite Annahme gemacht, die noch zu einem Fehler führen kann: Wir haben angenommen, daß wir zumindest partiell die Hilfsfälle konstruieren können. Dies setzt aber voraus, daß für zumindest ein Attribut die Schranken angegeben werden können. Wenn dies aber nicht der Fall ist, so sind gar keine Hilfsfälle konstruierbar und die obige Überlegung ist hinfällig. Nun könnten wir in diesem Extremfall wieder pessimistisch sein und die Suche *nicht* abbrechen bzw. den Teilbaum *auf jeden Fall* durchsuchen. Doch glücklicherweise müssen wir auch in diesem Extremfall die Leistungsfähigkeit unseres Suchalgorithmuses nicht so radikal beschneiden, wenn man sich nochmals die Gründe für das Fehlen der Schranken ansieht:

1. Attribut A_j ist geordnet, jedoch enthält der betrachtete Teilbaum oder das betrachtete Bucket einen Fall, dessen Attributwert nicht definiert ist, d. h. $A_j = \perp$ für den betreffenden Fall.

2. Attribut A_j ist ungeordnet. Deshalb sind keine Unter- und Oberschranke für den in einem Teilbaum enthaltenen Wertebereich von A_j angebbar.

Punkt 1 wird dann interessant, wenn für alle Fälle in dem Teilbaum oder Bucket das betrachtete Attribut $A_j = \perp$ ist. Dies kann beim BOB-Test ausgenutzt werden, welcher ja einen Hilfsfall F_{min} konstruiert, welcher noch in dem betrachteten Teilbaum T, in welchen die Hyperkugel überlappt werden soll, liegt. Wenn wir aber nun wissen, daß für alle Fälle in T $A_j = \perp$ gilt, dann kann man auch in F_{min} das Attribut $A_j = \perp$ setzen. Sei F'_{min} der so aus F_{min} entstehende Fall. Wenn man dann verlangt, daß für jedes lokale Ähnlichkeitsmaß sim_j gilt

$$\forall v \in W_j \cup \{\perp\} : sim_j(v, \perp) = sim_j(\perp, v) = 0 \tag{3.5}$$

dann gilt, daß für alle möglichen Fälle F gilt: $sim(F, F'_{min}) \leq sim(F, F_{min})$. Dies kann man aber nun zur Verbesserung des BOB-Tests ausnutzen, wenn bedenkt, daß dieser für einen Anfragefall F_q und den Ähnlichkeitsradius r den betrachteten Teilbaum von der Suche *ausschließt*, falls

[10] $\perp$ als Partitionswert ist ja ausgeschlossen

$sim(F_q, F_{min}) < r$ ist. Verwendet man nun F'_{min} statt F_{min}, so hat man wegen $sim(F_q, F'_{min}) \leq sim(F_q, F_{min})$ den BOB-Test verbessert, d. h. er ist in der Lage, öfters Teilbäume von der Suche auszuschließen. In der Praxis hat sich gezeigt, daß bei Fällen mit einigen unbekannten Attributen dieser Mechanismus bis zu 50 % mehr Fälle von der Suche ausschließt. Außerdem profitiert diese Überlegung natürlich von den neu eingeführten $\perp$-Ästen im k-d-Baum, da diese ja genau dafür sorgen, daß Fälle, die in einem Attribut unbekannt sind, zusammengefaßt werden.

Punkt 2 führt zu der Überlegung, daß man zwar den im Teilbaum T enthaltenen Wertebereich eines Attributs A_j nicht durch zwei Grenzen angeben kann, jedoch ist es immer möglich, die enthaltenen Werteausprägungen aufzulisten. Das führt aber i. a. dazu, daß gerade in der Nähe der Wurzel des k-d-Baums fast alle enthaltenen Werteausprägungen jedes ungeordneten Attributs aufgelistet werden müßten, was zu einem erheblichen Speicherbedarf führen würde, da die Grenzen ja in jedem Knoten des Baums gespeichert werden sollen (s. u.). Wir beschränken uns daher auf den Spezialfall, daß alle Fälle in Teilbaum T im ungeordneten Attribut A_j den gleichen Wert haben. Dies kann wieder analog zu Punkt 1 im BOB-Test verwendet werden, wenn man statt den Wert $*$ für das Attribut A_j den für alle Fälle in T gleichen Wert v einsetzt, da nach Gl. 3.4 ja gilt:

$$\forall w \in W_j \cup \{*, \perp\} : sim_j(v, w) \leq sim_j(*, w) \wedge sim_j(w, v) \leq sim_j(w, *)$$

Obige Überlegung profitiert von der Möglichkeit, daß auch nach ungeordneten Attributen partitioniert werden kann. Das führt zu nichts anderem, als daß in den damit erzeugten Partitionen der bearbeiteten Fallmenge die Werte der Fälle im Diskriminatorattributs gleich sind.

Es soll nun noch einmal zusammengefaßt festgelegt werden, wie die Grenzen eines Teilbaums zu berechnen sind. Diese sind dann für jeden Teilbaum, sprich Knoten des k-d-Baums, zu berechnen und in dem Knoten zu speichern.

Definition 3.11 (Bounds) *Die <u>Bounds</u> B eines Teilbaums T werden für die in ihm enthaltene Fallmenge M_T wie folgt festgelegt: $b_T = (b_<, b_=)$ mit $b_< = ((l_{i_1}, u_{i_1}), \ldots, (l_{i_m}, u_{i_m}))$ falls die Attribute $A_{i_1}, \ldots, A_{i_m}$ geordnet sind $(1 \leq i_1 < \ldots < i_m \leq k)$, sowie $b_= = (v_{j_1}, \ldots, v_{j_p})$ falls die Attribute $A_{j_1}, \ldots, A_{j_p}$ ungeordnet sind $(1 \leq j_1 < \ldots < j_p \leq k)$. Die Elemente von $b_<$ und $b_=$ sind dabei wie folgt festgelegt:*

- *Für $i_t \in \{i_1, \ldots, i_m\}$ ist*

$$(l_{i_t}, u_{i_t}) = \begin{cases} (v_l, v_u) \in W_{i_t} \times W_{i_t} & \text{falls } \forall <(w_1, \ldots, w_k), s> \in M_T : v_l \leq_{i_t} w_{i_t} \leq_{i_t} v_u \\ (\perp, \perp) & \text{falls } \forall <(w_1, \ldots, w_k), s> \in M_T : w_{i_t} = \perp \\ (*, *) & \text{sonst} \end{cases}$$

- *Für $j_t \in \{j_1, \ldots, j_p\}$ ist*

$$v_{j_t} = \begin{cases} v \in W_{j_t} & \text{falls } \forall <(w_1, \ldots, w_k), s> \in M_T : v = w_{j_t} \\ * & \text{sonst} \end{cases}$$

Nun ist es möglich die erweiterten Bounds-Tests anzugeben. Begonnen werden soll mit dem etwas einfacheren BOB-Test.

Definition 3.12 (Erweiterter Bounds-Overlap-Ball-Test) *Sei T der zu untersuchende Teilbaum mit den Dimensionsgrenzen $b_T = (b_<, b_=)$ wie in Def. 3.11. Weiter sei $F_q = (w_1, \ldots, w_k)$ der Anfragefall, zu dem die n ähnlichsten Fälle zu finden seien[11], und r der Ähnlichkeitsradius, d. h.*

$$r = \begin{cases} sim(F_q, F') & \text{wenn } F' \text{ der } n\text{-ähnlichste der bisher gefundenen Fälle ist} \\ -\infty & \text{wenn erst} < n \text{ verschieden ähnliche Fälle gefunden wurden} \end{cases}$$

Wir konstruieren einen Hilfsfall F_{min}, welcher am nahesten an F_q aber noch innerhalb von T liegt. Liegt dieser Fall innerhalb der Hyperkugel, so überlappt diese in den Teilbaum T. Für die Dimensionsgrenzen gelte $b_< = ((l_{i_1}, u_{i_1}), \ldots, (l_{i_m}, u_{i_m}))$ und $b_= = (v_{j_1}, \ldots, v_{j_p})$. Der Hilfsfall $F_{min} = (x_1, \ldots, x_k)$ wird wie folgt konstruiert:

[11] Fälle mit gleicher Ähnlichkeit zum Anfragefall werden zusammengefaßt, d. h. es werden n verschiedene "Ähnlichkeiten" gesucht

- *Für $t \in \{i_1, \ldots, i_m\}$:*

$$x_t = \begin{cases} \perp & \textit{falls } l_t = u_t = \perp \\ * & \textit{falls } l_t = u_t = * \\ w_{q_t} & \textit{falls } l_t \leq w_{q_t} \leq u_t \\ l_t & \textit{falls } w_{q_t} \leq l_t \\ u_t & \textit{falls } u_t \leq w_{q_t} \end{cases}$$

- *Für $t \in \{j_1, \ldots, j_p\}$:*

$$x_t = v_t$$

Das Prädikat $\texttt{BOB}(F_q, r, T)$ *läßt sich nun wie folgt beschreiben:*

$$\texttt{BOB}(F_q, r, T) \Leftrightarrow sim(F_q, F_{min}) \geq r \qquad (\star)$$

In obiger Definition werden die Angaben zu den ungeordneten Attributen in $b_=$ zur Belegung des Hilfsfalls benutzt. Wenn diese Werte ungleich $*$ enthalten kann sich die Ähnlichkeit von F_q zu F_{min} vermindern, da $*$ ja immer zum optimalen lokalen Ähnlichkeitswert führt. Das heißt aber, daß das Prädikat BOB wahrscheinlich seltener eintritt, was dazu führt, das der korrespondierende Teilbaum wahrscheinlich seltener durchsucht werden muß. Hier hat sich also die Analyse der ungeordneten Attribute schon gelohnt. Kommen wir nun zum BWB-Test:

Definition 3.13 (Erweiterter Ball-Within-Bounds-Test) *Sei T der zu untersuchende Teilbaum mit den Dimensionsgrenzen $b_T = (b_<, b_=)$ wie in Def. 3.11. Weiter sei der Anfragefall F_q, der Ähnlichkeitsradius r und n wie in Def. 3.12.*
Für die Dimensionsgrenzen gelte $b_< = ((l_{i_1}, u_{i_1}), \ldots, (l_{i_m}, u_{i_m}))$ und $b_= = (v_{j_1}, \ldots, v_{j_p})$. Es werden nun getrennt für die geordneten und ungeordneten Dimensionen Hilfsfälle konstruiert.

1. **Geordnete Dimensionen:** *Sei $I_< = \{i | i \in \{i_1, \ldots, i_m\}, l_i \notin \{*, \perp\}\}$ die Indexmenge der beschränkten, geordneten Dimensionen und $I_\perp = \{i | i \in \{i_1, \ldots, i_m\}, l_i = \perp\}$. Für $i \in I_<$ konstruieren wir nun Hilfsfälle $F^{(i)}_{min} = (h^{(i)}_1, \ldots, h^{(i)}_k)$ und $F^{(i)}_{max} = (g^{(i)}_1, \ldots, g^{(i)}_k)$ mit $j = 1, \ldots, k$ wie folgt:*

$$h^{(i)}_j = \begin{cases} l_j & \textit{falls } i = j \\ w_{q_j} & \textit{sonst} \end{cases}$$

$$g^{(i)}_j = \begin{cases} u_j & \textit{falls } i = j \\ w_{q_j} & \textit{sonst} \end{cases}$$

   $F^{(i)}_{min}$ und $F^{(i)}_{max}$ werden wieder analog zu Def. 2.7 zu $F^{(i)}$ zusammengefaßt:

$$F^{(i)} = \begin{cases} F^{(i)}_{max} & \textit{falls } sim_i(w_{q_i}, g^{(i)}_i) > sim_i(w_{q_i}, h^{(i)}_i) \\ F^{(i)}_{min} & \textit{sonst} \end{cases}$$

2. **Ungeordnete Dimensionen:** *Sei $I_= = \{i | i \in \{j_1, \ldots, j_p\}, v_i \neq *\}$ die Indexmenge der definierten, ungeordneten Dimensionen. Für $i \in I_=$ konstruieren wir nun Hilfsfälle $F^{(i)} = (h^{(i)}_1, \ldots, h^{(i)}_k)$ mit $j = 1, \ldots, k$ wie folgt:*

$$h^{(i)}_j = \begin{cases} \texttt{next}_i(v_j) & \textit{falls } i = j \\ w_{q_j} & \textit{sonst} \end{cases}$$

   Dabei ist $\texttt{next}_i$ die Funktion, die zu einem Wert aus W_i den (bzw. einen der) nächstähnlichen Wert(e) liefert.

Das Prädikat $\texttt{BWB}(F_q, r, T)$ *läßt sich nun wie folgt beschreiben:*

$$\begin{array}{lll} \texttt{BWB}(F_q, r, T) \Leftrightarrow & \forall i \in I_< \cup I_= : sim(F_q, F^{(i)}) < r & \wedge \qquad (\star) \\ & \forall i \in I_< : l_i \leq_i w_{q_i} \leq_i u_i & \wedge \\ & \forall i \in I_= : v_i = w_{q_i} & \wedge \\ & I_\perp = \emptyset & \wedge \\ & |I_<| + |I_=| > 0 & \end{array}$$

Die Analyse des BWB-Tests gestaltet sich im Vergleich zu der in Kapitel 2 vorgestellten Version erheblich schwieriger, da nun nicht mehr von einer geometrischen Interpretation ausgegangen werden kann. So steht in einigen Dimensionen keine Information über die "Ausdehnung" des Teilbaums im "Raum" zur Verfügung. Für diese Dimensionen können dann natürlich keine Hilfsfälle konstruiert werden. Diese Situation kann nur dann auftreten, wenn bisher nicht nach diesen Attributen/Dimensionen im Baum diskriminiert wurde[12]. Das heißt aber doch, daß für diese Attribute noch jeder Wert aus dem zugehörigen Wertebereich sowie der Wert $\perp$ auftreten kann. Also kann sich die Hyperkugel mit dem Radius r in einer dieser Dimensionen doch beliebig "breit machen", da wir für diese Dimensionen noch deren ganze "Breite" ausschöpfen können.

Die Konstruktion von Hilfsfällen für geordnete Dimensionen, deren Grenzen *unbekannt*, also gleich $(\perp, \perp)$, sind, kann man sich sparen, da sich die Hilfsfälle nur dann vom Anfragefall unterscheiden würden, wenn dieser in dem entsprechenden Attribut einen Wert ungleich $\perp$ vorweisen könnte. Da aber jeder andere Wert aus dem entsprechenden Wertebereich der Dimension mindestens gleichgute lokale Ähnlichkeitswerte wie $\perp$ erzielt, kann BWB(...) nicht zutreffen, da es möglicherweise einen Teilbaum gibt, in welchem gerade dieses mindestens gleichgut (bzgl. der lokalen Ähnlichkeit) mit dem Anfragefall übereinstimmt. Da wir aber genau genommen nicht n Fälle, sondern Fälle mit n verschiedenen Ähnlichkeiten suchen, kann die Suche an dieser Stelle nicht durch den BWB-Test abgebrochen werden.

Für die ungeordneten Dimensionen wird die Hilfsfall-Konstruktion der geordneten Dimensionen nachgeahmt, in dem eine einfache Ordnung $<_w \in W_j$ bezüglich dem Wert w wie folgt definiert wird:

$$\text{Für } u, v \in: u <_w v \Leftrightarrow sim_j(u, w) > sim_j(v, w) \tag{3.6}$$

Diese Ordnung wird innerhalb $\texttt{next}_j$ lediglich dazu benutzt, den nächstähnlichen Wert zu w zu bestimmen, welcher dann in den Hilfsfall für Attribut A_j eingesetzt wird.

Ein spezieller Fall tritt dann ein, wenn für überhauptkeine Dimension, geordnet oder ungeordnet, eine Angabe vorliegt. Dann können natürlich keinerlei Hilfsfälle konstruiert werden und der BWB-Test hat keinerlei Information, aufgrund derer er die Suche abbrechen könnte. Wie man sich aber leicht überlegt, kann diese Situation nur in der Wurzel des k-d-Baums eintreten, da ja ansonsten nach mindestens einem Attribut hätte diskriminiert werden müssen, sodaß zumindest für dieses eine Angabe in den Bounds enthalten sein müßte. Da der BWB-Test aber nur dann in der Wurzel aufgerufen wird, wenn schon der komplette k-d-Baum durchsucht worden ist, bleibt dieses Phänomen ohne Auswirkungen.

Anpassung der dynamischen Bounds

Der in [14] und [9] vorgeschlagene Beschleunigungs-Ansatz der *dynamischen Bounds* (s. a. Kap. 2) läßt sich auch auf die nach Definition 3.11 konstruierten Grenzen eines Teilbaums anwenden. Relevant für die Betrachtung von minimalen und maximalen Grenzen ist jedoch immer nur der Teil, welcher die geordneten Attribute/Dimensionen beschreibt, und auch nur an den Stellen, an welchen Werte ungleich $*$ und $\perp$ für die jeweilige Ober- und Untergrenze zur Verfügung stehen. Für den so eingeschränkten Teilbereich der Grenzen eines Teilbaums laßen sich dann beispielsweise die in obiger Literatur angegebenen Berechnungsalgorithmen für dynamische Bounds anwenden.

Die Anwendung der minimalen und maximalen Grenzen bleibt natürlich wie zuvor, d. h. der BWB-Test verwendet die maximalen Grenzen, während der BOB-Test die minimalen Grenzen verwendet.

Auswahl des nächsten zu durchsuchenden Teilbaums

Wie wir schon Kapitel 2 gesehen haben läßt sich das Retrieval von Fällen im k-d-Baum in zwei Aufgaben unterteilen:

- Lineares Retrieval von Fällen innerhalb von Blattknoten (Buckets).

[12] Die Umkehrung dieses Satzes gilt jedoch i. a. nicht

- Retrieval innerhalb von Teilbäumen, also inneren Knoten.

Während die erste Aufgabe sehr einfach zu lösen ist, verdient die zweite Aufgabe mehr Aufmerksamkeit. Im vorigen Abschnitt wurden die dazu verwendeten Hilfsmittel der Bounds-Test an die geänderte Struktur der Fälle und des k-d-Baums angepaßt. Ist nun die Retrieval-Prozedur innerhalb eines inneren Knotens, so stellt sich die Frage, welcher der Sohnknoten als nächster zu durchsuchen ist. Dabei sind zwei grundlegend verschiedene Situationen zu unterscheiden:

1. Der innere Knoten wird zum ersten mal während des aktuellen Retrievals besucht.

2. Der innere Knoten wird im Rahmen des Backtrackings wiederholt konsultiert.

In Kapitel 2 war diese Aufgabe noch sehr leicht zu lösen, da jeder Fall (auch der Anfragefall) voll spezifiziert war, d. h. keine unbekannten Werte enthielt, und jeder innere Knoten nur die zwei Sohnknoten zu $\leq$ und $>$ enthielt. Hier war die Lösung sehr einfach: Zuerst wurde der "passende Sohn", d. h. der, der den passenden Wertebereich zum Wert des Diskriminatorattributs im Anfragefall enthielt, durchsucht, und bei einem möglichen Backtracking wurde der noch nicht besuchte Sohnknoten durchsucht. Wie man in Abb. 3.1 ersehen kann ist die Aufgabe nun aber um einiges aufwendiger geworden. Wie man oben schon sehen kann, bietet sich eine Zweiteilung der

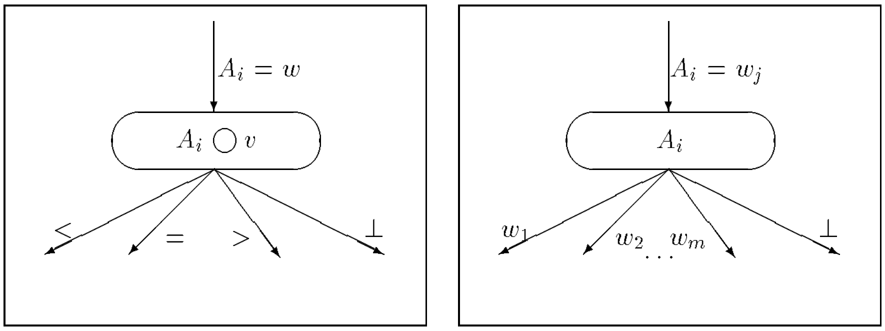


Abbildung 3.1: Auswahl des nächsten Teilbaums für geordnete (l) und ungeordnete Attribute (r)

Selektions-Prozedur an:

1. Eine Funktion `bestSon` liefert den zu Attributwert w_j am besten passenden Sohnknoten. Diese Funktion wird während der Vorwärtsphase, d. h. beim ersten Besuchen des Knoten aufgerufen.

2. Eine Funktionen `nextSon` liefert nacheinander die nächsten zu w_j passenden Sohnknoten. Diese Funktion wird während der Backtrackingphase, d. h . beim wiederholten Besuch des Knotens aufgerufen.

Desweiteren müssen diese beiden Prozeduren natürlich getrennt für Knoten, die nach geordneten Attributen, und für Knoten, die nach ungeordneten Attributen diskriminieren, festgelegt werden, da bei Knoten für geordnete Attribute ja nur *ein* Vergleichswert zum Attributwert des Anfragefalls, nämlich gerade der Partitionswert, zur Verfügung steht, während bei Knoten, welche nach ungeordneten Attributen diskriminieren, ja die ganze Palette der möglichen Werte für das Diskriminatorattribut zur Verfügung steht. Ausserdem muß natürlich bei beiden Knotentypen noch festgelegt werden, wie $\perp$ zu behandeln ist, daß ja zum einem als Attributwert des Anfragefalls auftreten kann und zum anderen die Werteausprägung des Diskriminatorattributs in einem der Sohnknoten, nämlich dem $\perp$-Sohn, welcher in beiden Typen von inneren Knoten immer vorhanden ist, beschreiben kann.

- **Innere Knoten für geordnete Attribute:** Die Idee für `bestSon` ist ja schon oben beschrieben worden. Die Idee für `nextSon` ist folgende (w bezeichne im folgenden den Wert des Diskriminatorattributs im Anfragefall und $T_{\bigcirc}$ den Sohn zur Menge $M_{\bigcirc}$ für $\bigcirc \in \{<, >, =, \perp\}$):

- Falls $w \neq \perp$: Vorgegeben sei folgende Ordnung von Sohnknoten: $T_<$,$T_=$,$T_>$. Diese wird dazu verwendet, eine sinnvolle Reihenfolge von Sohnknoten zu generieren, indem nacheinander die nächsten Nachbarn in der vorgegebenen Reihenfolge generiert werden, d. h. wenn der durch `bestSon` generierte erste Sohnknoten $T_>$ war, wird danach $T_=$ und danach $T_<$ angeboten. War jedoch $T_=$ der erste Sohnknoten, so wird danach $T_<$ und anschließend $T_>$ angeboten. Zu guter letzt wird immer der Sohnknoten $T_\perp$ angeboten, da in diesem Teilbaum der lokale Ähnlichkeitswert für das Diskriminatorattribut am niedrigsten ist.
- Falls $w = \perp$: Als erster Sohnknoten wird von `bestSon` der Knoten $T_\perp$ generiert. Danach werden die restlichen Sohnknoten in einer beliebigen Reihenfolge $T_<$, $T_=$, $T_>$ angeboten. Diese Vorgehensweise resultiert daher, daß der lokale Ähnlichkeitswert für das Diskriminatorattribut in allen Fällen aller Sohnknoten gleich ist. Das Erstangebot von $T_\perp$ kann mit der Hoffnung begründet werden, daß ein zum Anfragefall ähnlicher Fall eventuell auch im Diskriminatorattribut unbekannt ist. Für die restlichen Knoten kann dann zumindest aus dieser lokalen Sichtweise keine sinnvolle Reihenfolge generiert werden.

- **Innere Knoten für <u>ungeordnete</u> Attribute:** Die Idee für `bestSon` ist oben schon beschrieben worden, während die Idee für `nextSon` folgende ist (w_j bezeichne im folgenden den Wert des Diskriminatorattributs A_i im Anfragefall und T_v den Sohn zur Menge M_v für $v \in W_i$):
  - Falls $w_j \neq \perp$: Als erster Sohnknoten wird von `bestSon` natürlich T_{w_j} angeboten. Für die Generierung der folgenden Sohnknoten gibt es zwei mögliche Ansätze:
    1. Die noch nicht besuchten Sohnknoten werden in einer beliebigen Reihenfolge angeboten. Dies macht Sinn bei lokalen Ähnlichkeitsmaßen sim_i, die nur einen einfachen Match abtesten, d. h. für die gilt:
    $$sim_i(u,v) = \begin{cases} 1 & \text{falls } u = v \\ 0 & \text{sonst} \end{cases}$$
    2. Die noch nicht besuchten Sohnknoten werden mit sinkendem lokalen Ähnlichkeitswert zu w_j angeboten, d. h. T_u wird vor T_v angeboten, falls $sim_i(w_j,u) > sim_i(w_j,v)$ ist. Dieser Ansatz macht Sinn für lokale Ähnlichkeitsmaße, die mehr als nur einen $\{0,1\}$-Vergleich durchführen und so eine nicht triviale Partialordnung bezüglich w_j über W_i definieren. Natürlich ist dieser Ansatz aufwendiger als der erste, da eventuell zu jedem Wert eines ungeordneten Attributs eine Anordnung der restlichen Werte erzeugt werden muß. Dies ist allerdings nur einmal nötig (z. B. nach der Festlegung der lokalen Ähnlichkeitsmaße) und bleibt dann bei jedem Retrieval-Vorgang fest.

    Als letzter Knoten wird mit der gleichen Begründung wie bei geordneten Attributen der Knoten $T_\perp$ angeboten.
  - Falls $w_j = \perp$: Als erster Sohnknoten wird von `bestSon` der Knoten $T_\perp$ generiert. Danach werden die restlichen Sohnknoten in einer beliebigen Reihenfolge angeboten. Diese Vorgehensweise resultiert wieder aus der gleichen Begründung wie bei geordneten Attributen.

Retrievalalgorithmus

Abschließend soll noch das Schema des Retrieval-Algorithmus angegeben werden. Dieser wurde im Vergleich zur in [14] angegebenen Version nochmals verfeinert, und zwar wurde der dort zusätzlich eingeführte BOB-Test[13] für alle inneren Knoten des k-d-Baums wieder weggelassen, da

[13] welcher bei der Suche des Buckets, welches den Anfragefall (theoretisch) enthalten könnte, vor dem Eintritt in jeden Unter-Teilbaum aufgerufen wurde

sich gezeigt hat, daß der Test, wenn überhaupt, erst kurz vor einem Bucket "zuschlägt". Da die Zeit zur Ausführung eines BOB-Tests in keinem Verhältnis zur minimalen Durchlaufzeit zu einem Sohnknoten steht, kann man sich die Zeit für den BOB-Test sparen. Sinn macht er jedoch, wenn man ihn vor dem Besuchen eines Buckets aufruft, da das Besuchen eines Buckets sehr "teuer" werden kann, da auf jeden Fall der Anfragefall mit allen im Bucket enthaltenen Fällen der Fallbasis verglichen wird. Wenn man zusätzlich noch davon ausgeht, daß die Fälle beim Besuchen eines Buckets aus einer Datenbank beschafft werden müssen, dann macht der *eine* zusätzliche BOB-Test sehr viel Sinn, da mit diesem eventuell sehr viel Zeit gespart werden kann, da dieser für die meist sehr engen Grenzen eines Buckets sehr genau bestimmen kann, ob sich ein Besuchen desselben lohnt oder auch nicht.
Der Algorithmus benutzt folgende Parameter:

- den Anfragefall F_q,
- den aktuellen Teilbaum T,
- die Ergebnisqueue Q, welche Elemente der Form $(sim, \{f_1, \ldots, f_m\})$ absteigend nach sim sortiert enthält, d. h. die Fälle $f_1, \ldots, f_m$ sind mit dem Ähnlichkeitswert sim in Q enthalten. Es werden also Fälle mit gleichem Ähnlichkeitswert zusammengefaßt. Q ist nach absteigenden Werten von sim geordnet. Letzter Parameter ist
- die Anzahl der zu suchenden Fälle n. Insbesondere müssen in Q nicht mehr als n Fälle gespeichert werden.

Das Ergebnis des Algorithmus wird ebenfalls in der Queue Q gespeichert. Das zurückgegebene Flag dient lediglich der Abbruchkontrolle der Suche.

An Funktionen benutzt der Algorithmus die weiter oben eingeführten `bestSon` und `nextSon` sowie die Funktion `insert`, welche lediglich ein Queue-Insert realisiert. Das verwendete Prädikat `isBucket` testet lediglich, ob das Argument ein Bucket ist oder nicht. Doch nun zum Algorithmus:

Algorithmus 3.3 (search(F_q, T, Q, n))

```
finished := false /* merkt sich, ob die Suche beendet wurde */
IF isBucket(T) THEN /* T ist ein Bucket */
    FOR f ∈ T.Cases DO
        /* Berechne zu allen enthaltenen Fällen die Ähnlichkeit zu F_q */
        insert(Q, f, sim(f, F_q))
    ENDFOR
ELSE /* T ist ein innerer Knoten */
    t := bestSon(T, F_q)
    IF ¬isBucket(t) OR BOB(F_q, Q[n].sim, t) THEN
        /* Durchsuche besten Sohn */
        finished := search(F_q, t, Q, n)
    ENDIF
    WHILE ¬finished AND (t := nextSon(T, F_q, t)) ≠ empty_node DO
        /* Durchsuche die restlichen Söhne */
        IF BOB(F_q, Q[n].sim, t) THEN
            finished := search(F_q, t, Q, n)
        ENDIF
    ENDWHILE
ENDIF
IF ¬finished THEN
    finished := BWB(F_q, Q[n].sim, T)
ENDIF
RETURN finished
```

3.3 Einbettung mehrerer Ähnlichkeitsmaße

Die Verwendung mehrerer verschiedener Ähnlichkeitsmaße im Fallbasierten Schließen ist schon in [10] zu finden. Dort wird vorgeschlagen, für jede Diagnose, welche in der Begriffswelt dieser Arbeit einer Problemlösung oder Klasse entsprechen, die Möglichkeit der Definition einer eigenen Ähnlichkeitsfunktion vorzusehen. Die Vorteile der Verwendung mehrerer Ähnlichkeitsmaße sind:

- Die Attribute der Fälle können je nach Problemlösung unterschiedliche Wichtigkeit haben. Dies kann in das Ähnlichkeitsmaß mit hineinfließen.
- Die Ähnlichkeitsmaße werden zur Klassen/Konzepterkennung benutzt. Dann ist aber die Mächtigkeit gewichteter globaler Ähnlichkeitsmaße, wie sie in Kapitel 2 vorgeschlagen wurden, viel zu beschränkt, da mit ihnen nur konjunktive Konzepte erkannt werden können. Dies läßt sich an einem Beispiel sehr einfach verdeutlichen. Nehmen wir also wieder Fälle mit den drei Attributen Farbe, Form und Größe. Wäre das betrachtete Konzept Farbe = rot$\wedge$Form = Dreieck, so ließe sich daß noch in einer Gewichtsverteilung von $\omega_{\text{Farbe}} = \omega_{\text{Form}} = 0.5$ und $\omega_{\text{Größe}} = 0$ wiederspiegeln. Würde das Konzept jedoch (Farbe = rot $\wedge$ Form = Dreieck) $\vee$ Größe = normal lauten, so gibt es keine einfache herleitbare Gewichtsverteilung. Um solche disjunktiven Konzepte mit Ähnlichkeitsmaßen darstellen zu können, muß man die Disjunktionsglieder auf mehrere Maße verteilen, d. h. im obigen Beispiel würde man ein Maß für Farbe & Form erzeugen, sowie ein Maß für die Größe, die dann mit Hilfe der Funktion *max* zusammengefaßt werden können.

3.3.1 Relevanzen & gewichtete globale Ähnlichkeitsmaße

Die Einführung mehrerer, lösungsspezifischer Ähnlichkeitsmaße macht vor allen Dingen dann einen Sinn, wenn man gewichtete globale Ähnlichkeitsmaße, wie sie in Kapitel 2 vorgeschlagen wurden, verwendet. Dann kann man für jede Lösung/Klasse die Attribute entsprechend ihrer Relevanz gewichten. Aber noch ein weiterer Vorteil entspringt der Beschränkung auf solcherlei Ähnlichkeitsmaße. So lassen sich die verschiedenen Formen sehr leicht in Form einer Matrix, der sogenannten Relevanzmatrix, notieren, was den Implementierungsaufwand erheblich einschränkt, da sich alle Ähnlichkeitsmaße mit Hilfe der lokalen Maße und einem einfachen Skalarprodukt berechnen lassen.

Definition 3.14 (Relevanzmatrix) *Sei $\mathcal{S} = \{s_1, \ldots, s_m\}$ die Menge der Problemlösungen bzw. Klassen. Die Matrix Ω heißt Relevanzmatrix, wenn gilt*[14]*:*

1.
$$\Omega = \begin{pmatrix} \omega_{11} & \omega_{12} & \cdots & \omega_{1m} \\ \omega_{21} & \omega_{22} & \cdots & \omega_{2m} \\ \vdots & \vdots & & \vdots \\ \omega_{k1} & \omega_{k2} & \cdots & \omega_{km} \end{pmatrix}$$

2.
$$\forall j \in \{1, \ldots, m\} : \sum_{i=1}^{k} \omega_{ij} \quad = 1$$

3.
$$\omega_{ij} \in [0,1] \text{ für } i \in \{1, \ldots, k\} \text{ und } j \in \{1, \ldots, m\}$$

Für jede Problemlösung/Klasse $s_j \in \mathcal{S}$ wird dann das zugehörige Ähnlichkeitsmaß sim^{s_j} wie folgt festgelegt:

$$sim^{s_j}(<(w_1^{(1)}, \ldots, w_k^{(1)}), s^{(1)}>, <(w_1^{(2)}, \ldots, w_k^{(2)}), s^{(2)}>) = \sum_{i=1}^{k} \omega_{ij} sim_i(w_i^{(1)}, w_i^{(2)}) \tag{3.7}$$

In [10] werden die Ähnlichkeitsmaße mit Hilfe der Forderung zusammengefaßt, daß das erste Argument von sim immer eine definierte Diagnose, sprich Problemlösung oder Klasse, hat, d. h. für die Fälle $<(w_1, \ldots, w_k), s>$ und f würde sim wie folgt festgelegt:

$$sim(<(w_1, \ldots, w_k), s>, f) = sim^s(<(w_1, \ldots, w_k), s>, f) \tag{3.8}$$

Dieser Ansatz wird hier, zumindestens in dieser strengen Form, nicht benutzt. Es wird vielmehr davon ausgegangen, daß es kein allgemein anwendbares Maß sim gibt. Es muß also immer festgelegt werden, welches Ähnlichkeitsmaß benutzt werden soll. Diese Festlegung wird im folgenden Abschnitt getroffen.

3.3.2 Auswahl der globalen Ähnlichkeitsmaße

Die Aufgabe der Auswahl eines globalen Ähnlichkeitsmaßes kann während des Fallretrievals in zwei Situationen auftreten:

1. Es wird innerhalb eines Buckets für alle enthaltenen Fälle ($\in FB$) die Ähnlichkeit zum Anfragefall F_q berechnet.
2. Es werden in inneren Knoten des k-d-Baums Bounds-Tests aufgerufen. Diese berechnen ebenfalls Ähnlichkeiten zum Anfragefall F_q, diesmal jedoch wird dieser nicht mit einem Fall aus der Fallbasis verglichen, sondern mit einem konstruierten Hilfsfall.

Diese beiden Situation erfordern eine unterschiedliche Vorgehensweise, da verschiedene Arten von Fällen mit dem Anfragefall verglichen werden.

[14] k bezeichnet wieder die Anzahl der Attribute

Ähnlichkeitsberechnung zwischen Anfragefall und Fall aus der Fallbasis

Gegeben sind also der Anfragefall F_q und ein Fall $f \in FB$. Die Ähnlichkeitsberechnung gestaltet sich in dieser Situation relativ einfach, da wir schon Definition 2.2 festgelegt hatten, daß jeder Fall aus der Fallbasis eine bekannte Problemlösung oder Klasse hat. Ist also s die Lösung von f, so wird die Ähnlichkeit zwischen F_q und f mit sim^s (s. Gl. 3.7 und 3.8) berechnet.

Ähnlichkeitsberechnung zwischen Anfragefall und Hilfsfall der Bounds-Tests

Gegeben sind in dieser Situation also der Anfragefall F_q und ein von einem der Bounds-Tests konstruierter Hilfsfall h. Hier ist die Situation wesentlich schwieriger als noch im vorigen Abschnitt, da weder F_q noch h eine Information über die Zugehörigkeit zu einer Klasse enthalten. Da F_q auf keinen Fall Informationen über eine Klassenzugehörigkeit liefern kann (diese soll ja gerade unter anderem im Rahmen des Retrievals näher bestimmt werden), kann nur im Umfeld von h nach Zugehörigkeiten zu Klassen geforscht werden. Dazu werden die Bounds-Tests getrennt betrachtet:

- **BOB-Test:** Der BOB-Test entscheidet, ob innerhalb des betrachteten Teilbaums T noch weitere, mindestens r-ähnliche[15] Fälle zu finden sind. Da diese aber die Ähnlichkeitsmaße sim^s verwenden (die Fälle liegen ja in Buckets), müssen zur Ähnlichkeitsbestimmung zwischen F_q und h *alle in T möglichen* Ähnlichkeitsmaße sim^s benutzt werden. Das heißt also, daß

  1. in jedem Teilbaum T bekannt sein muß, welchen Klassen die in dem Teilbaum enthaltenen Fällen angehören. Diese Information kann bspw. in den Knoten des Baums gespeichert werden.
  2. Die zu den in T enthaltenen Klassen/Problemlösungen $\{s_1, \ldots, s_m\}$ gehörenden Funktionen $sim^{s_1}, \ldots, sim^{s_m}$ müssen im BOB-Test bei der Berechnung der Ähnlichkeit zwischen F_q und h berücksichtigt werden, da T nur dann von der Suche ausgeschlossen werden darf, wenn für alle sim^{s_i} der Fall h aus dem Radius r ausgeschlossen werden kann. Die Zeile ($\star$) in Definition 3.12 muß also wie folgt erweitert werden:

  $$\texttt{BOB}(F_q, r, T) \Leftrightarrow \exists s \in \{s_1, \ldots, s_m\} : sim^s(F_q, F_{min}) \geq r$$

- **BWB-Test:** Der BWB-Test entscheidet, ob ausserhalb des aktuellen Teilbaums T noch Fälle existieren können, die bessere Ähnlichkeitswerte als r liefern[15]. Hier könnte man den zum BOB-Test komplementären Ansatz wählen, und sich alle außerhalb von T existierenden Problemlösungen/Klassen merken. Das dies i. a. aber sowieso fast immer alle vorhandenen sind macht dieser Aufwand wenig Sinn. Es reicht also einfach alle möglichen Klassen in Zeile ($\star$) von Definition 3.13 zu berücksichtigen[16]:

  $$\forall s \in \mathcal{S} \, \forall i \in I_< \cup I_= : sim^s(F_q, F^{(i)}) < r$$

3.4 Leistungsvergleich

Es soll nun die Leistungsfähigkeit des wie zuvor beschriebenen Retrieval-Mechanismus mit erweiterten k-d-Bäumen aufgezeigt werden. Dazu wird das implementierte System (im folgenden nur noch *erweitertes* oder *neues* System genannt), welches auch im Rahmen des INRECA-Projektes verwendet wird, verglichen mit einem älteren System (im folgenden nur noch *altes* System genannt) des Projektes, welches k-d-Bäume mit den Erweiterungen aus [14] verwendet. Diese Erweiterungen sind bis auf den leicht modifizierten Retrieval-Steuerungsalgorithmus auch im erweiterten System enthalten.

Die Testdomäne bildet wie in [14] wieder die öffentlich zugängliche PKW-Datenbank aus dem UCI REPOSITORY OF MACHINE LEARNING DATABASES AND DOMAIN THEORIES. Diese wird im Anhang näher beschrieben.

[15] r bezeichnet den aktuellen Ähnlichkeitsradius, s. Def. 3.12 und 2.7

[16] $\mathcal{S}$ bezeichnet die Menge aller Problemlösungen/Klassen

Im verwendeten Ähnlichkeitsmaß wurden die Gewichte der Attribute gleichverteilt, d. h. von den 23 diskriminanten Attributen erhielt jedes das Gewicht $\omega = 1/23$. Die verwendeten lokalen Ähnlichkeitsmaße wie auch der verwendete Anfragefall sind ebenfalls dem Anhang zu entnehmen. Zu bemerken ist noch, daß der Anfragefall *nicht* in der Fallbasis enthalten ist. Er erzielt zum ähnlichsten Fall der Fallbasis den Ähnlichkeitswert 0.58375540 und stellt somit eine durchaus realistische Anfrage dar.

Als erstes soll die Anzahl der betrachteten Fälle ein beiden Systemen untersucht werden. Es wurden dazu $n = 1, 5, 10, 15$ ähnlichste Fälle[17] gesucht. Die Anzahl der untersuchten Fälle ist der beste Indikator der Retrieval-Geschwindigkeit, da diese sich in etwa proportional zu dieser Anzahl verhält. In Abb. 3.2 ist die Anzahl der untersuchten Fälle jeweils für jede (im jeweiligen System) mögliche Generierungsmethode[18] für k-d-Bäume (s. Kap. 2 und 3) aufgetragen worden. Wie man sieht müssen im alten System bei n=1 im *günstigsten* Fall schon über 140 (= 68%) Fälle inspiziert werden, während es im erweiterten System im *schlechtesten* Fall gerade mal 44 (= 22%) Fälle sind[19]. Diese Zahl läßt sich noch durch Wahl einer besseren Generierungsmethode auf 6 (= 3%) Fälle drücken. Dieser Trend setzt sich auch für $n = 5, 10, 15$ fort. Während im erweiterten System sich die Zahl der Fälle nur sehr langsam mit n steigert und auch für $n = 15$ unter 100 rangiert, werden im alten System schon bei $n = 5$ über 175 (= 85%) Fälle inspiziert.

Wie leistungsfähig das erweiterte System ist, zeigt sich in Abb. 3.2 an der Kurve für den mit Entropie generierten Baum. Obwohl dieser ohne jeglichen Einfluß des Ähnlichkeitsmaßes generiert wurde, werden mit ihm bessere Retrieval-Leistungen erzielt als mit jedem k-d-Baum des alten Systems. Hier hat also die geänderte Datenraumaufteilung (zusätzlicher Gleichheitsast) sowie der verfeinerte Retrievalalgorithmus seine Wirkung gezeigt. Dies ist auch in Abb. 3.3 und 3.4 ersichtlich. Wie man sieht verwendet das erweiterte System wesentlich intensiver als das alte System den BOB-Test, um eventuell Teilbäume aus der Suche auszuschließen. Dies ist eine Folge der stärkeren Verästelung des k-d-Baumes, der somit mehr Möglichkeiten zum BOB-Test bietet. Umgekehrt wird die Verwendung des BWB-Tests im erweiterten System dadurch stark vermindert. Daraus läßt sich die stark gestiegene Retrieval-Geschwindigkeit (s. Abb. 3.6) erklären, denn zum einem erzeugt die stärkere BOB-Test-Verwendung einen starke Beschneidung des Baumes, während zum anderen der BWB-Test wie schon erläutert wurde sehr selten die Suche abbrechen kann und somit oft nutzlos ist. Wenn man dann noch bedenkt, daß der BOB-Test wesentlich "billiger“ als der BWB-Test ist, da der letztere bis zu k Ähnlichkeitsberechnungen erfordert, der BOB-Test aber mit maximal einer Ähnlichkeitsberechnung auskommt, dann ist die starke Geschwindigkeitssteigerung eine sich aufdrängende Konsequenz.

Zu den in Abb. 3.6 angegebenen Retrievalzeiten ist zu bemerken, daß diese unter UNIX während dem normalen Mehrbenutzer- und -prozeßbetrieb gemessen wurden. Da sie jedoch auf dem gleichen Rechner mit etwa gleichbleibender Auslastung durchgeführt wurden, sind sie durchaus noch partiell aussagekräftig.

Zusammengefaßt kann man das geänderte Retrievalverhalten für den in beiden Systemen mit Interquartilsabstand generierbaren k-d-Baum in Abb. 3.5 betrachten. Dieser liegt mit seiner Leistungsfähigkeit im Mittelfeld beider Systeme und verhält sich wie oben beschrieben.

3.5 Zusammenfassung

In diesem Kapitel wurden die k-d-Bäume so erweitert, daß mit ihnen auch ein effizientes Retrieval von Fällen möglich ist, welche Attribute mit nicht zu ordnendem Wertebereich enthalten. In der Original-Version der k-d-Bäume sind solche Fälle zwar auch (eingeschränkt) handhabbar, indem die Information der ungeordneten Attribute einfach ignoriert wird. Wie aber im Kapitel

[17] Fälle mit gleichem Ähnlichkeitswert zum Anfragefall werden zusammengefaßt, d. h. es werden n "Ähnlichkeiten“ gesucht

[18] bis auf die Methode *iqdsim*, da diese sich kaum von *maxdsim* unterscheidet

[19] Der mit Entropie generierte k-d-Baum ist dabei nicht vergleichbar, da dieser ja ohne Einbettung einer Information über das Ähnlichkeitsmaß generiert wird. Die mit ihm im erweiterten System erzielten Ergebnisse sind allerdings immer noch wesentlich besser als im alten System

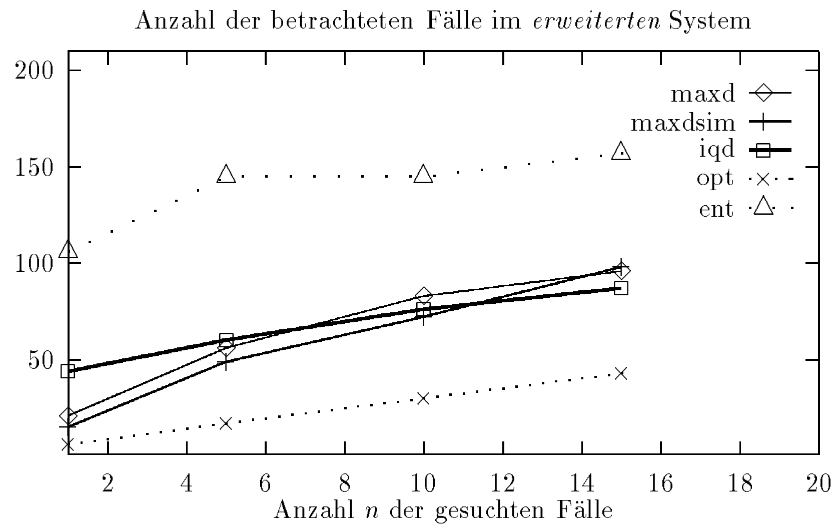
gezeigt wurde kann auch die Information der ungeordneten Attribute zum effizienten Retrieval miteinbezogen werden. Außerdem ermöglicht diese Erweiterung die Bearbeitung von Fällen, welche fast ausschließlich ungeordnete Attribute enthalten. Diese Klasse von Fällen war bisher Ansätzen vorbehalten, welche mit Hilfe direkter Indizierung die nächsten Nachbarn zu einem Anfragefall bestimmten. Diese bringen jedoch die starke Einschränkung mit sich, daß als lokale Ähnlichkeitsfunktionen nur simple matching-Funktionen verwendet werden können um korrekt zu sein. Dies hat jedoch mehr mit normalem Datenbank-Retrieval als mit leistungsfähiger nearest-neighbour-Klassifikation zu tun, welche mit leicht veränderbaren und leistungsfähigen Ähnlichkeitsmaßen ausgestattet ist, die dann z. B. auch verrauschte Daten handhaben können.

Als weiterer Punkt wurde die Behandlung von Attributen mit unbekanntem Wert angegangen. Dieser in der Praxis häufig auftretende Fall konnte so integriert werden, daß er zum einen den k-d-Baum nicht unnötig vergrößert (wie bspw. bei Entscheidungsbäumen) und zum anderen noch geschickt im Retrieval ausgenutzt werden kann.

Die von Mehlhorn (s. [30]) vorgeschlagene Implementierung von k-d-Bäumen wurde integriert und für alle obigen Erweiterungen wurden Generierungsverfahren entwickelt, welche je nach Generierungsaufwand k-d-Bäume erzeugen, die zu effizientem bis sehr effizientem Retrieval führen. Diese Effizienz beruht auch auf den erweiterten Retrieval-Steuerungsmechanismen, welche die Informationen über unbekannte und ungeordnete Attribute sowie die lokalen Ähnlichkeitsmaße bei der Auswahl des nächsten zu durchsuchenden Teilbaums zur Retrieval-Steuerung mitverwenden.

Letztlich wurde noch die Möglichkeit der Verwendung von klassenspezifischen Ähnlichkeitsmaßen eingebaut wie sie z. B. in der fallbasierten Diagnose auftreten.

Im Vergleich mit dem bisherigen k-d-Baum nach [14] konnte die Retrieval-Geschwindigkeit um etwa den Faktor 10 verbessert werden.



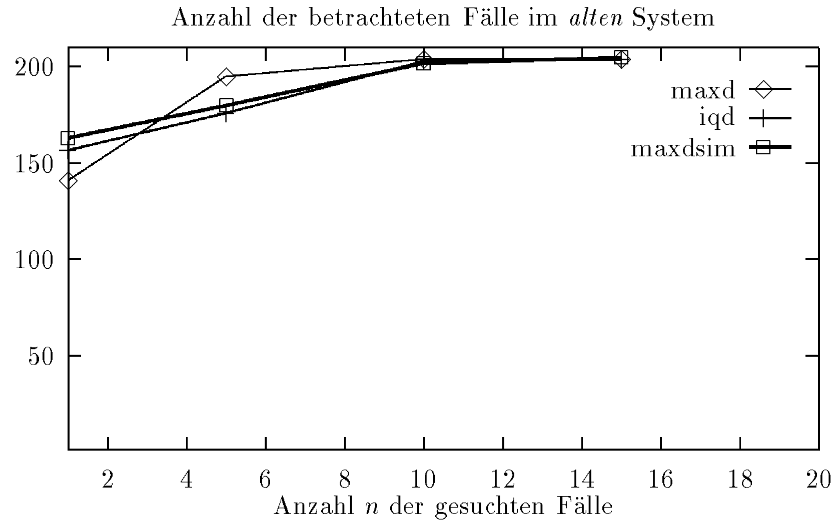


Abbildung 3.2: Anzahl der betrachteten Fälle (Ähnlichkeitsvergleiche)

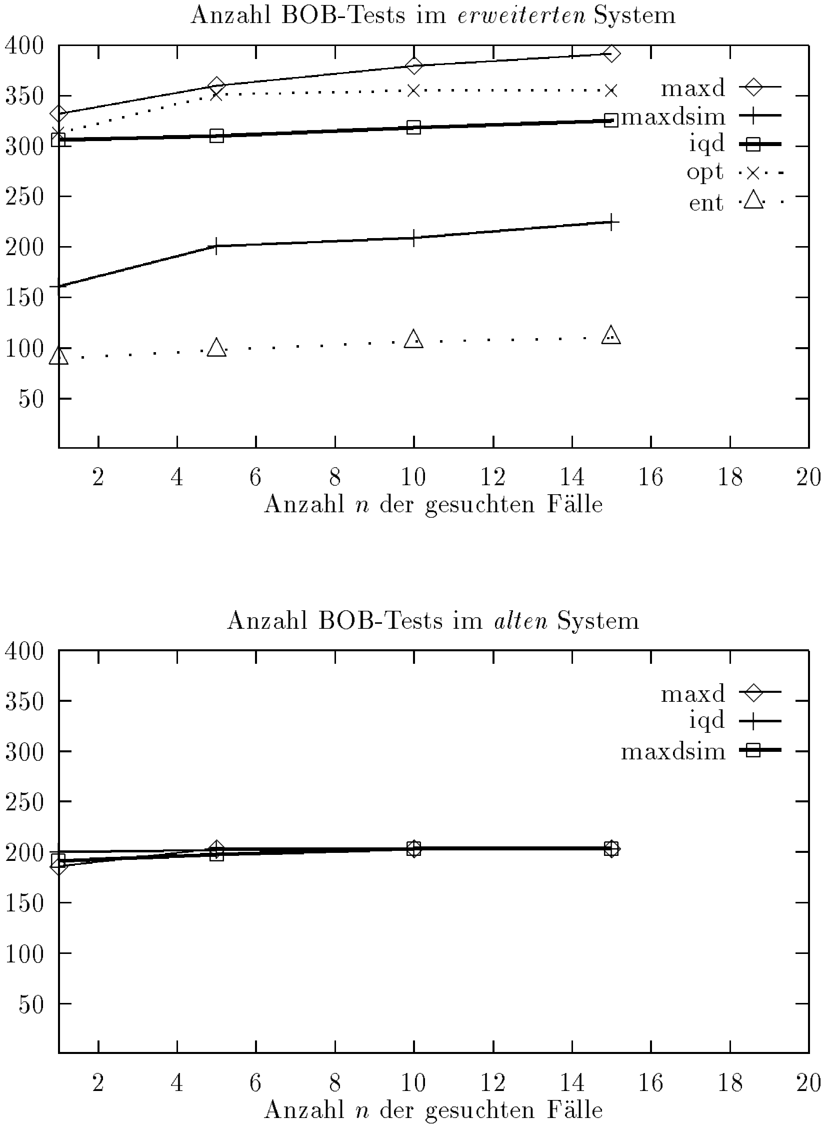
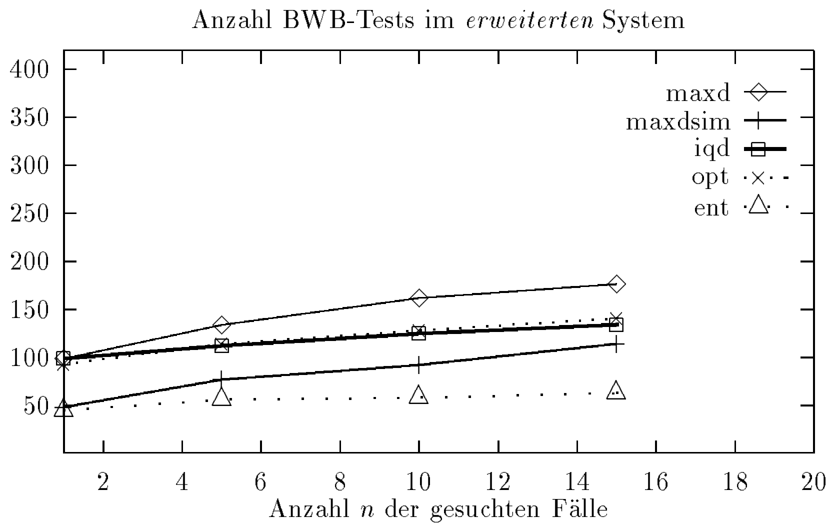


Abbildung 3.3: Anzahl der ausgeführten Bounds-Overlap-Ball-Tests



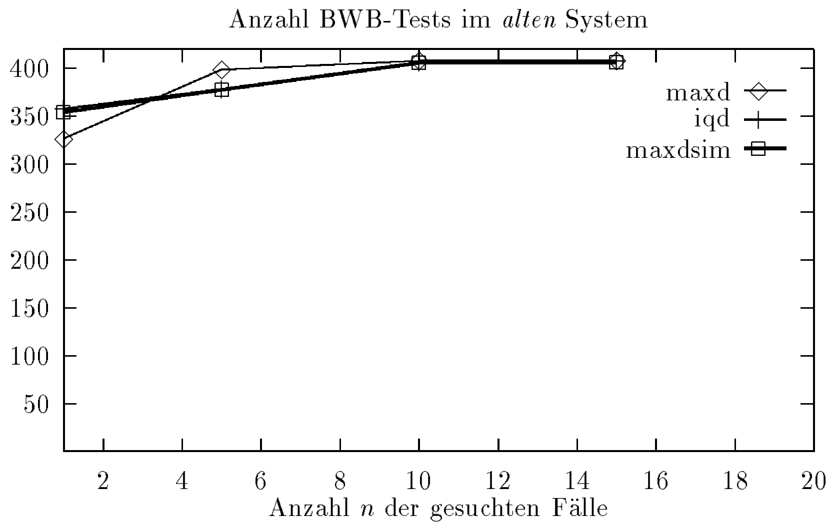


Abbildung 3.4: Anzahl der ausgeführten Ball-Within-Bounds-Tests

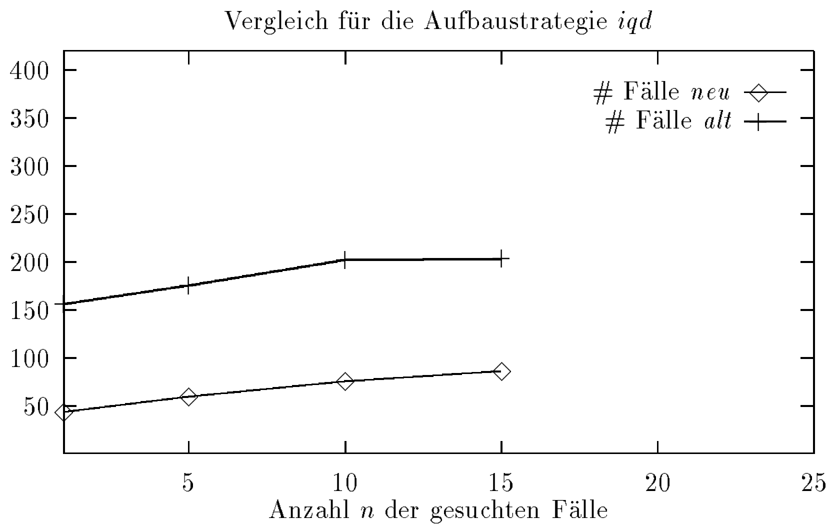
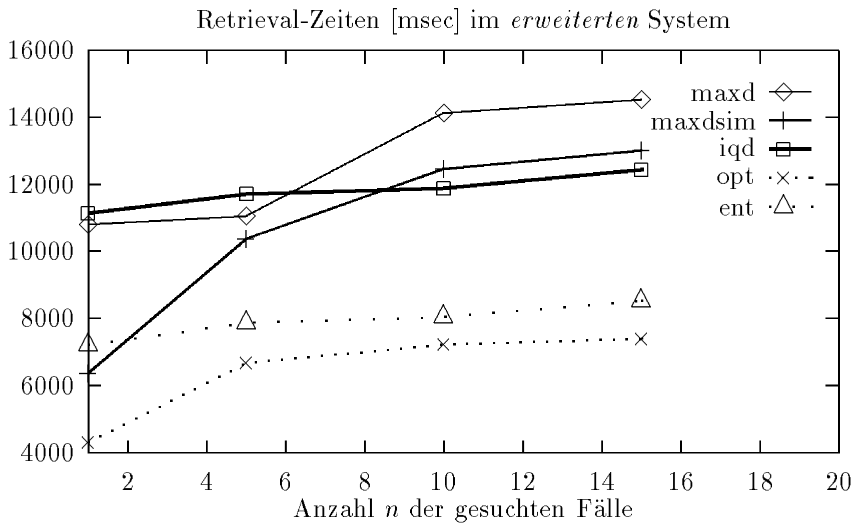


Abbildung 3.5: Retrievalverhalten des mit Interquartilsabstand erzeugten k-d-Baums



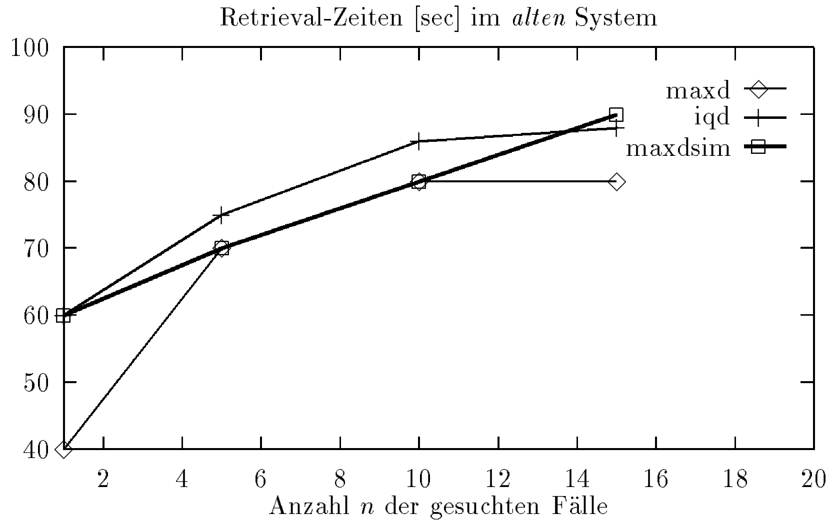


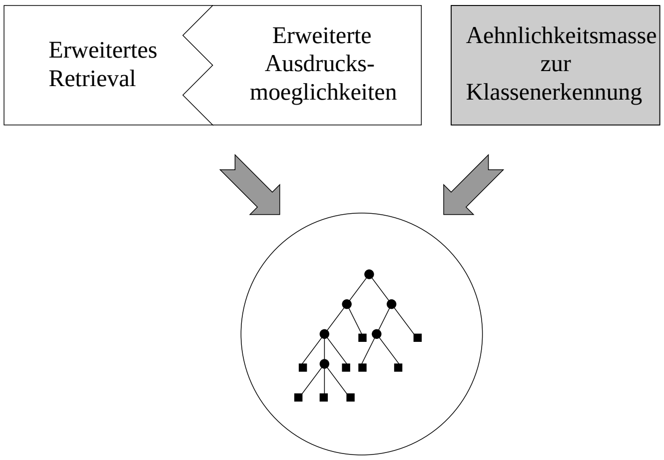
Abbildung 3.6: Retrievalzeiten

Kapitel 4

Erzeugung von Ähnlichkeitsmaßen zur Klassenerkennung

4.1 Integration, Teil II

Die Idee der Verbesserung der verwendeten Ähnlichkeitsmaße ist neben der Erweiterung des k-d-Baums der zweite Bestandteil der Integration von k-d- und Entscheidungsbäumen bzw. induktiven und fallbasiertem Schließen.



Integrierter k-d- und Entscheidungsbaum

Die Aufgabe ist nun die Generierung von Ähnlichkeitsmaßen, welche effiziente Konzepterkennung ähnlich den induktiv generierten Konzepterkennungsfunktionen erlauben ohne deren Nachteile (s. u.) zu übernehmen. Deren Verwendung im k-d-Baum ermöglicht dann eine fallbasierte Klassifikation von relativ komplexen Fällen, ohne daß der Benutzer des Systems ein *sinnvolles* Ähnlichkeitsmaß vorgeben muß. Es ist also nicht wie allgemein üblich nötig, daß der Benutzer eines *nearest-neighbour*-Klassifizierers *a-priori* mit Hilfe des Ähnlichkeitsmaßes vorgeben muß, wie klassifiziert werden soll, sondern diese Aufgabe kann ähnlich wie bei Entscheidungsbäumen automatisiert werden, ohne daß die erzeugten Ähnkeitsmaße ihre *dynamische Veränderbarkeit* verlieren, welche z. B. bei Entscheidungsbäumen gar nicht vorhanden ist.

4.2 Einführung

Eine häufig anzutreffende Aufgabe z. B. in der Diagnostik ist die, daß anhand einiger bekannter Symptome eine den Zustand beschreibende Diagnose gestellt werden muß. In der Begriffswelt dieser Arbeit kann man sich Symptome als Attribute mit Werten und Diagnosen als Klassenbezeichner oder auch ausführlicherere Problemlösungsbeschreibungen vorstellen. Mit dem Entscheidungsbaum wurde schon in Kapitel 2 ein Hilfsmittel dafür vorgestellt. Dieser repräsentiert ja u. a. auch eine Menge von Klassifikationsregeln der Form $A_{i_1} = v_1 \wedge \ldots \wedge A_{i_m} = v_m \rightarrow s$ für Attribute A_{i_j}, Werte v_j und Klassenbezeichner s, welche zur Bestimmung der Klasse eines Falles benutzt werden können. Diese durch die Benutzung des Baumes implizit benutzten Regeln haben jedoch einige Nachteile:

- Es wird nur *eine* Klasse für jeden Anfragefall vorgeschlagen. Kann diese aber nicht bestätigt werden, z. B. durch Nachprüfen oder durch zusätzliche Informationen, so kann kein weiterer Vorschlag für eine Klasse generiert werden.

- Ist der Anfragefall *verrauscht*, also in einigen Attributen falsch oder auch nur geringfügig verfälscht, so kann die zum unverrauschten Fall gehörende Klasse meist nicht gefunden werden. Da keine Alternativen generiert werden können und die korrekt oder falsch vorgeschlagene Klasse noch von der Struktur des Entscheidungsbaumes abhängt, sind verrauschte Daten i. a. nur schwierig handhabbar.

- Unbekannte Werte sind in Entscheidungsbäumen schwierig zu behandeln. Treten sie in Fällen der Beispielmenge (Fallbasis) auf, so müssen Fälle dupliziert werden. Treten sie im Anfragefall auf, so können sie zum Abbruch des Klassifizierungsalgorithmus führen. Dies ist immer dann der Fall, wenn in einem Knoten des Baums nach Attribut A_i diskriminiert wird, dieses aber gerade im Anfragefall unbekannt ist.

- Oft wird neben der Klasse eines Anfragefalles auch ein Fall gesucht, welcher zu dieser Klasse gehört und ähnlich zum Anfragefall ist. I. a. wird sogar verlangt, daß der ähnlichste Fall zu der Klasse gehört, welche mit Hilfe des Entscheidungsbaumes generiert wurde. Dies stellt jedoch ein durchaus großes Problem dar, weil im Entscheidungsbaum-Verfahren überhaupt keine Ähnlichkeitsmaße verwendet werden (können).

Als Abhilfe zu diesen Nachteilen wird ein in seinem Grundmuster sehr einfaches Verfahren vorschlagen:

- Zum Anfragefall werden die n ähnlichsten Fälle in der Fallbasis FB gesucht. Als Klasse für den Anfragefall wird die des ähnlichsten Falls vorgeschlagen. Diese Aufgabe läßt sich natürlich sehr gut mit k-d-Bäumen erledigen. Sie entspricht im Kern ja nichts anderem als ähnlichkeitsbasiertem Fallretrieval.

- Vorschläge für weitere Klassen ergeben sich in der Reihenfolge absteigender Ähnlichkeitswerte.

Da Ähnlichkeitsmaße recht einfach so ausgelegt werden können, daß sie stabil gegenüber Verrauschung sind, schließt dieser Ansatz alle Nachteile des obigen aus. Er kann diesen auch Ersetzen, wenn man folgende Forderung an das Ähnlichkeitsmaß erfüllen kann (s. a. [12]):

Das Ähnlichkeitsmaß muß die gleiche Ausdrucksmächtigkeit wie eine Klassen- oder Konzepterkennungsfunktion haben.

Das soll z. B. heißen, daß man mit einem Ähnlichkeitsmaß nicht nur einfache konjunktive Konzepte (s. o.) erkennen kann, sondern auch bspw. etwas kompliziertere disjunktive Konzepte. Dazu sind folgende Aufgaben zu bewältigen:

1. Der für jede Klasse spezifische Gewichtsvektor $(\omega_1, \ldots, \omega_k)$ muß durch Analyse der Fallbasis generiert werden können.
2. Die gegebene Klassifizierung der Fälle muß gegebenenfalls verfeinert werden, d. h. in Unterklassen aufgeteilt werden, um disjunktive Konzepte in einfachere, zumeist konjunktive Konzepte zu zerlegen.
3. Die verwendeten *lokalen* Ähnlichkeitsmaße müssen so verändert werden, daß sie Fälle einer Unterklasse nicht mehr unterscheiden. Dazu sind i. a. unterklassen-spezifische lokale Ähnlichkeitsmaße notwendig.

Diese Aufgaben werden nach den folgenden zwei Abschnitten angegangen.

4.3 Ähnlichkeitsmaße

Der allgemeine Begriff des Ähnlichkeitsmaßes wurde schon in Kapitel 2 eingeführt. Wir wollen uns hier auf die schon in Kapitel 3 eingeführten klassenspezifischen gewichteten Ähnlichkeitsmaße einschränken, da diese schon stark parametrisiert sind.

4.3.1 Ähnlichkeitsmaß-Parameter

Die einstellbaren Parameter der gewählten Klasse von Ähnlichkeitsmaßen sind im einzelnen:

- Das Gewicht ω_j für jedes Attribut A_j ist innerhalb $[0,1]$ und unter Berücksichtigung der Normierungsbedingung (s. Kap. 2) frei wählbar. Dabei gibt jedes Gewicht an, wie relevant dieses Attribut für die betrachtete Klasse ist. Stimmen die Fälle in der Klasse in diesem Attribut sehr stark (nicht) überein, so kann das durch ein hohes (niedriges) Gewicht zum Ausdruck gebracht werden.
- Die lokale Ähnlichkeitsfunktion sim_j für A_j ist eine beliebige Abbildung in $[0,1]$, welche lediglich für alle v aus dem Wertebereich die Bedingung $sim_j(v,v) = 1$ erfüllen muß. Sie bestimmt die Nähe der Werte aus dem Wertebereich zueinander und ergibt somit die Möglichkeit, nicht nur Werteübereinstimmungen, wie oben gefordert, zu erkennen, sondern *differenzierter* als mit simpler Gleichheit zu entscheiden, ob Werte ähnlich sind.

Jetzt ergibt sich die Frage, wie Gewichte und lokale Ähnlichkeitsfunktionen zu finden sind. Das Generieren von Gewichten wird im folgenden Abschnitt behandelt. Das Finden von lokalen Ähnlichkeitsfunktionen ist i. a. nicht steuerbar, da diese natürlich stark vom Typ und Wertebereich des jeweiligen Attributs abhängen. Das Problem ist für quantitative Attribute jedoch relativ leicht lösbar, da ein Abstand i. a. einfach zu berechnen ist, und dieser (s. [17]) in ein Ähnlichkeitsmaß umgerechnet werden kann. Für qualitative Attribute ist jedoch keine generelle Verfahrensweise vorgebbar. In einem späteren Abschnitt wird jedoch ein halbautomatisches Verfahren zur Erweiterung von lokalen Ähnlichkeitsmaßen vorgestellt, daß es ermöglicht, trotz einfacher Maße eine große Ausdrucksmächtigkeit zu erzielen.

4.3.2 Bewertungsmaßstab für Ähnlichkeitsmaße

Durch die große Anzahl an einstellbaren Parametern ist es i. a. sehr schwierig, ein Ähnlichkeitsmaß zu finden, welches die gewünschte Aufgabe erledigt, welches es also ermöglicht, "gute" Ähnlichkeiten für Fälle innerhalb einer Klasse zu erzielen. Die Frage ist dabei insbesondere, wann ein Maß "gut" ist und wann nicht. Dazu wird nun ein Bewertungsmaßstab vorgeschlagen. Die Idee dazu ist wie folgt:

1. Es wird eine Bewertung benötigt, welche angibt, wie genau Fälle innerhalb *einer* Klasse klassifziert werden, d. h. wie ähnlich der Fall zu den Fällen seiner Klasse ist. Als Maß bietet sich hier die mittlere Intra-Klassen-Ähnlichkeit. an. Diese wird unten definiert. Sie ist wieder eine auf $[0,1]$ abbildende Funktion, wobei 1 ein sehr guter Wert ist, der i. a. bedeutet, daß das Ähnlichkeitsmaß die den Fällen in einer Klasse gemeinsamen Features erkannt hat. 0 ist natürlich der denkbar schlechteste Wert.

2. Es wird eine weitere Bewertung benötigt, welche angibt, wie gut Klassen unterschieden werden können. Als Maß dazu bietet sich die mittlere Inter-Klassen-Ähnlichkeit an (s. u.). Auch sie ist eine auf $[0,1]$ abbildende Funktion. 0 ist hier der beste Wert, denn er bedeutet, daß die Fälle unterschiedlicher Klassen keine Ähnlichkeit zueinander aufweisen.

Obiger Ansatz weißt einige Analogien zu Konzepterkennungsfunktionen im symbolischen Lernen auf, in welcher Konzepte dann als gelernt betrachtet werden, wenn sie *konsistent* und *vollständig* sind (s. [17]). So ist der Begriff der Intra-Klassen-Ähnlichkeit mit dem Begriff der Vollständigkeit vergleichbar, während die Inter-Klassen-Ähnlichkeit mit der Konsistenz vergleichbar ist. Weitere Analogien ergeben sich zur Bildverarbeitung, in welcher die Segmentierung von Bildern in ähnlicher Weise erfolgt, wobei man sich hier die Pixel eines Bildes als Fälle vorstellen kann.

Definition 4.1 (Intra- und Inter-Klassen-Ähnlichkeit) *Sei M eine Menge von Fällen mit den Problemlösungen/Klassen $\{s_1, \ldots, s_m\}$, welche sich anhand der in den Fällen enthaltenen Klasseninformation in die disjunkten Mengen $M_{s_1}, \ldots, M_{s_m}$ partitionieren läßt. Dann sind Inter- und Intra-Klassen-Ähnlichkeit wie folgt definiert[1] für $1 \leq i \leq m$:*

$$\overline{sim}^{intra}_{M_{s_i}} = \frac{1}{|M_{s_i}|^2} \sum_{f_1, f_2 \in M_{s_i}} sim(f_1, f_2)$$

$$\overline{sim}^{inter}_{M_{s_i}} = \frac{1}{|M_{s_i}|\,|M - M_{s_i}|} \sum_{f_1 \in M_{s_i}, f_2 \in M - M_{s_i}} sim(f_1, f_2)$$

Diese Maße lassen sich auch wie folgt zur *kombinierten mittleren Ähnlichkeit* zusammenfassen:

$$\overline{sim}_{M_{s_i}} = \overline{sim}^{intra}_{M_{s_i}} (1 - \overline{sim}^{inter}_{M_{s_i}}) \tag{4.1}$$

$\overline{sim}_{M_{s_i}}$ ist wieder auf $[0,1]$ normiert, wobei 1 für ein optimales und 0 für ein schlechtes Ähnlichkeitsmaß steht.

4.4 Beispieldomäne und -fälle

Alle in diesem Kapitel vorgestellen Konzepte und Begriffe sollen anhand einer Beispieldomäne erläutert und exemplarisch nachgeprüft werden. Die betrachteten Fälle bestehen aus folgenden drei Attributen:

- Farbe mit dem Wertebereich $W_{\text{Farbe}} = \{$ dunkelrot, rot, hellrot$\}$
- Form mit dem Wertebereich $W_{\text{Form}} = \{$ Dreieck, Viereck, Fünfeck, Sechseck$\}$
- Größe mit dem Wertebereich $W_{\text{Größe}} = \{$ klein, normal, groß$\}$

Dabei sei das Attribut Form wie folgt geordnet: Dreieck $<_{\text{Form}}$ Viereck $<_{\text{Form}}$ Fünfeck $<_{\text{Form}}$ Sechseck. Die lokalen Ähnlichkeitsmaße werden wie folgt vorgegeben:

[1] Diese Definition gilt streng genommen nur für symmetrische Ähnlichkeitsmaße, kann jedoch durch $(sim(f_1, f_2) + sim(f_2, f_1))/2$ statt $sim(f_1, f_2)$ auf unsymmetrische erweitert werden

sim_{Farbe}	W_{Farbe}	dunkelrot	rot	hellrot	
	dunkelrot	1	0.5	0	
	rot	0.5	1	0.5	
	hellrot	0	0.5	1	
sim_{Form}	W_{Form}	Dreieck	Viereck	Fünfeck	Sechseck
	Dreieck	1	$\frac{2}{3}$	$\frac{1}{3}$	0
	Viereck	$\frac{2}{3}$	1	$\frac{2}{3}$	$\frac{1}{3}$
	Fünfeck	$\frac{1}{3}$	$\frac{2}{3}$	1	$\frac{2}{3}$
	Sechseck	0	$\frac{1}{3}$	$\frac{2}{3}$	1
$sim_{\text{Größe}}$	$W_{\text{Größe}}$	klein	normal	groß	
	klein	1	0.5	0	
	normal	0.5	1	0.5	
	groß	0	0.5	1	

Es sollen folgende drei Konzepte gleichzeitig gelernt werden:

1. K_1: Farbe = rot $\wedge$ Form = Dreieck
2. K_2: Form = Fünfeck $\vee$ Form = Sechseck
3. K_3: Form = Viereck $\vee$ (Form = Dreieck $\wedge$ (Farbe = dunkelrot $\vee$ Farbe = hellrot))

Die 36 Fälle, welche die Domäne vollständig beschreiben, können der Tabelle 4.1 entnommen werden.

4.5 Gewichtsgenerierung

Für das in Abschnitt 4.4 eingeführte Beispiel ergeben sich für gleichverteilte Gewichte (in diesem Fall also jeweils $\frac{1}{3}$ pro Attribut und Klasse) folgende Werte:

Klasse	K_1	K_2	K_3
$\overline{sim}^{intra}_{M_{K_i}}$	0.851852	0.648148	0.638518
$\overline{sim}^{inter}_{M_{K_i}}$	0.548822	0.481481	0.508995
$\overline{sim}_{M_{K_i}}$	0.384337	0.336077	0.313516

Wie man leicht sieht sind die Werte für diese noch sehr weit von dem Optimum 1 bzw. 0 entfernt. Dies liegt natürlich an der willkürlichen Wahl der Gewichte. Ein erster Ansatz zur automatischen Generierung von Gewichten ist im System PATDEX/2 [10] zu finden. Dort werden Gewichte über Häufigkeiten des Auftretens von Werteausprägungen eines Attributs bestimmt. Diese werden anschließend mit einem Korrektur-Algorithmus so angepaßt, daß alle Fälle der Trainings-Fallmenge korrekt klassifiziert werden. Da der Algorithmus jedoch unter Umständen die Gewichte zyklisch ändert, kann weder ein korrektes noch ein verbessertes Klassifikationsverhalten mit den so erzeugten Ähnlichkeitsmaßen garantiert werden. Desweiteren benutzt PATDEX/2 keine lokalen Ähnlichkeitsmaße zur Generierung von Gewichten (außer implizit im Korrektur-Algorithmus bei der Test-Klassifikation der Trainingsfälle) und verzichtet somit auf eine Informationsquelle über differenziertere Ähnlichkeiten von Attributwerten.

Im vorigen Abschnitt wurde als Bewertungsmaßstab für die Güte eines Ähnlichkeitsmaßes die Intra- und Inter-Klassen-Ähnlichkeit vorgeschlagen. Die Idee ist nun, dieses Kriterium direkt zur Berechnung eines Ähnlichkeitsmaßes zu verwenden. Dazu betrachten wir die Inter- und Intra-Klassen-Ähnlichkeit nicht komplett für alle Attribute, sondern jeweils getrennt für ein Attribut:

Definition 4.2 (Lokale Intra- und Inter-Klassen-Ähnlichkeit) *Sei M wieder eine Menge von Fällen mit den Problemlösungen/Klassen $\{s_1, \ldots, s_m\}$, welche sich anhand der in den Fällen enthaltenen Klasseninformationen in die disjunkten Mengen $M_{s_1}, \ldots, M_{s_m}$ partitionieren läßt.*

Fall	Farbe	Form	Größe	Klasse
Fall_{11}	rot	Dreieck	klein	K_1
Fall_{12}	rot	Dreieck	normal	K_1
Fall_{13}	rot	Dreieck	groß	K_1
Fall_{21}	dunkelrot	Fünfeck	klein	K_2
Fall_{22}	dunkelrot	Fünfeck	normal	K_2
Fall_{23}	dunkelrot	Fünfeck	groß	K_2
Fall_{24}	rot	Fünfeck	klein	K_2
Fall_{25}	rot	Fünfeck	normal	K_2
Fall_{26}	rot	Fünfeck	groß	K_2
Fall_{27}	hellrot	Fünfeck	klein	K_2
Fall_{28}	hellrot	Fünfeck	normal	K_2
Fall_{29}	hellrot	Fünfeck	groß	K_2
Fall_{2A}	dunkelrot	Sechseck	klein	K_2
Fall_{2B}	dunkelrot	Sechseck	normal	K_2
Fall_{2C}	dunkelrot	Sechseck	groß	K_2
Fall_{2D}	rot	Sechseck	klein	K_2
Fall_{2E}	rot	Sechseck	normal	K_2
Fall_{2F}	rot	Sechseck	groß	K_2
Fall_{2G}	hellrot	Sechseck	klein	K_2
Fall_{2H}	hellrot	Sechseck	normal	K_2
Fall_{2I}	hellrot	Sechseck	groß	K_2
Fall_{31}	dunkelrot	Viereck	klein	K_3
Fall_{32}	dunkelrot	Viereck	normal	K_3
Fall_{33}	dunkelrot	Viereck	groß	K_3
Fall_{34}	rot	Viereck	klein	K_3
Fall_{35}	rot	Viereck	normal	K_3
Fall_{36}	rot	Viereck	groß	K_3
Fall_{37}	hellrot	Viereck	klein	K_3
Fall_{38}	hellrot	Viereck	normal	K_3
Fall_{39}	hellrot	Viereck	groß	K_3
Fall_{3A}	dunkelrot	Dreieck	klein	K_3
Fall_{3B}	dunkelrot	Dreieck	normal	K_3
Fall_{3C}	dunkelrot	Dreieck	groß	K_3
Fall_{3D}	hellrot	Dreieck	klein	K_3
Fall_{3E}	hellrot	Dreieck	normal	K_3
Fall_{3F}	hellrot	Dreieck	groß	K_3

Tabelle 4.1: Fälle der Beispielfallbasis

Dann sind <u>*lokale Inter- und Intra-Klassen-Ähnlichkeit*</u> *für das Attribut* A_j *und* $1 \leq i \leq m$ *wie folgt definiert:*

$$\overline{sim}^{intra}_{M_{s_i},A_j} = \frac{1}{|M_{s_i}|^2} \sum_{f_1,f_2 \in M_{s_i}} sim_j(f_1|_{A_j}, f_2|_{A_j})$$

$$\overline{sim}^{inter}_{M_{s_i},A_j} = \frac{1}{|M_{s_i}||M - M_{s_i}|} \sum_{f_1 \in M_{s_i}, f_2 \in M - M_{s_i}} sim_j(f_1|_{A_j}, f_2|_{A_j})$$

Dabei ist $f|_{A_j}$ *die Werteausprägung von Attribut* A_j *in Fall* f.

Die lokale Intra- und Inter-Klassen-Ähnlichkeit ist also die Übertragung des Begriffs der mittleren Ähnlichkeit auf ein Attribut mit dem zugehörigen lokalen Ähnlichkeitsmaß. Die lokale Intra-Klassen-Ähnlichkeit gibt dabei an, wie ähnlich im Mittel sich die Werteausprägungen des betreffenden Attributs innerhalb einer Klasse sind. Ein niedriger Wert nahe 0 läßt darauf schließen, daß das betrachtete Attribut keine Relevanz für die betrachtete Klasse hat, da sich die in den Fällen der Klasse enthaltenen Werte des Attributs doch stark unterscheiden. Einen großer Wert nahe 1 könnte man nun als Indiz dafür verbuchen, daß das betreffende Attribut eine große Relevanz für die betrachtete Klasse hat. Dies ist jedoch nicht ganz richtig, da noch betrachten muß, ob die Attributwerte der Klasse auch für diese *typisch* sind, d. h. man muß die Werteausprägungen des Attributs innerhalb der Klasse noch mit denen der anderen vorhandenen Klassen vergleichen. Dazu dient die lokale Inter-Klassen-Ähnlichkeit. Erst wenn diese niedrig (nahe 0) ist, kann man davon ausgehen, daß das Attribut relevant für die betrachtete Klasse ist. Für das in Abschnitt 4.4 angegebene Beispiel ergeben sich folgende Werte:

- Intra-Klassen-Ähnlichkeit:

	K_1	K_2	K_3
Farbe	1	0.5	0.52
Form	1	$0.8\overline{3}$	0.84
Größe	0.5	0.5	0.5

- Inter-Klassen-Ähnlichkeit:

	K_1	K_2	K_3
Farbe	$0.\overline{63}$	0.5	0.542857142
Form	$0.\overline{45}$	$0.\overline{6}$	0.428571428
Größe	0.5	0.5	0.5

Mit Hilfe der lokalen Inter- und Intra-Klassen-Ähnlichkeit kann man nun ein Gewicht für das jeweilige Attribut berechnen. Die Idee dazu ist in Fortsetzung obiger Interpretation, daß die Differenz aus Intra- und Inter-Klassen-Ähnlichkeit gerade den für die betrachtete Klasse besonders relevanten Anteil an der mittleren Ähnlichkeit beschreibt. Dieser werde mit $\Delta\overline{sim}_{M_{s_i},A_j}$ bezeichnet und berechnet sich für Attribut A_j und Klasse s_i wie folgt:

$$\Delta\overline{sim}_{M_{s_i},A_j} = \max\{0, \overline{sim}^{intra}_{M_{s_i},A_j} - \overline{sim}^{inter}_{M_{s_i},A_j}\} \tag{4.2}$$

Damit läßt sich nun für die Klasse s_i sehr einfach ein Gewichtvektor $(\omega_1, \ldots, \omega_k)$ durch Normalisierung der Summe auf 1 errechnen $(1 \leq j \leq k)$:

$$\omega_j = \frac{\Delta\overline{sim}_{M_{s_i},A_j}}{\sum_{l=1}^{k} \Delta\overline{sim}_{M_{s_i},A_l}} \tag{4.3}$$

Für das betrachtete Beispiel ergibt sich somit:

A_l	K_1 $\Delta\overline{sim}_{M_{K_1},A_l}$	K_1 ω_l	K_2 $\Delta\overline{sim}_{M_{K_2},A_l}$	K_2 ω_l	K_3 $\Delta\overline{sim}_{M_{K_3},A_l}$	K_3 ω_l
Farbe	$0.\overline{36}$	0.4	0	0	0 (-0.022857142)	0
Form	$0.\overline{54}$	0.6	$0.1\overline{6}$	1	0.411428572	1
Größe	0	0	0	0	0	0

Wie man obiger Tabelle entnehmen kann ist mit dieser Gewichtsgenerierungs-Methode für jede Klasse erkannt worden, daß die Werteausprägungen des Attributs Größe irrelevant sind, da diese ja auch nicht in den zugehörigen Konzepten festgelegt werden.

Für die Klasse K_2 ist ebenfalls erkannt worden, daß lediglich die Werteausprägungen des Attributs Form eine Rolle spielen, da das Konzept ja lediglich alle Fünfecke und Sechsecke umfaßt.

Die höhere Gewichtung von Attribut Form in Klasse K_1 läßt sich dadurch erklären, daß die Werteausprägung "Dreieck" seltener (nur noch in Klasse K_3) als "rot" für Farbe auftritt.

Das Konzept von Klasse K_3 kann jedoch noch nicht korrekt erkannt worden sein, da das Attribut Farbe nicht irrelevant für das Konzept ist, da ja im zweiten Disjunktionsglied auf Farbe=hellrot oder Farbe=dunkelrot getestet wird. Während sich, wie in Klasse K_1, konjunktive Konzepte oder, wie in Klasse K_2, disjunktive Konzepte mit pro Disjunktionsglied gleichen enthaltenen Attributen schon relativ gut erkennen lassen, gibt es noch einen Mangel bei disjunktiven Konzepten, welche verschiedene Attribute pro Disjunktionsglied enthalten. Dieses Problem wird im folgenden Abschnitt angegangen. Zum Abschluß soll nochmals die Verbesserung des Ähnlichkeitsmaßes gegenüber gleichverteilter Gewichtung dargestellt werden:

Klasse	K_1	K_2	K_3
$\overline{sim}^{intra}_{M_{K_i}}$	1	$0.8\overline{3}$	0.84
$\overline{sim}^{inter}_{M_{K_i}}$	0.527273	$0.\overline{3}$	0.4285071
$\overline{sim}_{M_{K_i}}$	0.472727	$0.\overline{5}$	0.480054
Verbesserung[2]	23 %	65 %	53 %

Als Ergebnis läßt sich für die in diesem Abschnitt vorgestellte Gewichtsgenerierungsmethode anführen, daß gegenüber gleichverteilter Gewichtung vor allen Dingen die (globale) Intra-Klassen-Ähnlichkeit verbessert wird, während sich die Inter-Klassen-Ähnlichkeit nur geringfügig verbessert oder sogar ein wenig verschlechtert. Insgesamt gesehen hat sich das Ähnlichkeitsmaß jedoch um mindestens 23 % verbessert.

4.6 Klassenerkennung

Es ist nun die Frage, wie auch "schwierige" Konzepte, wie z. B. für Klasse K_3, mit obiger recht einfachen Methode ausdrückbar sind. Das Problem ist hier, daß das Konzept in DNF[3] aus den drei Disjunktionsgliedern

1. Form=Viereck
2. Form = Dreieck $\wedge$ Farbe = dunkelrot
3. Form = Dreieck $\wedge$ Farbe = hellrot

besteht. Diese enthalten nur zum Teil eine Bedingung für das Attribut Farbe, was dazu führt, daß es keinen gemeinsam gültigen Gewichtsvektor für alle drei Disjunktionsglieder geben kann.

Für konjunktive Konzepte genügt *ein* Gewichtsvektor. Die Idee zur Bewältigung disjunktiver Konzepte muß also sein, daß man für jedes Disjunktionsglied einen eigenen Gewichtsvektor zur Verfügung stellt. Das heißt aber in der bisherigen Zuordnung von Ähnlichkeitsmaßen und Klassen nichts anderes, als daß man die bestehende Klassifikation verfeinert in dem Sinne, daß man pro

[2] gegenüber gleichverteilter Gewichtung
[3] disjunktiver Normalform

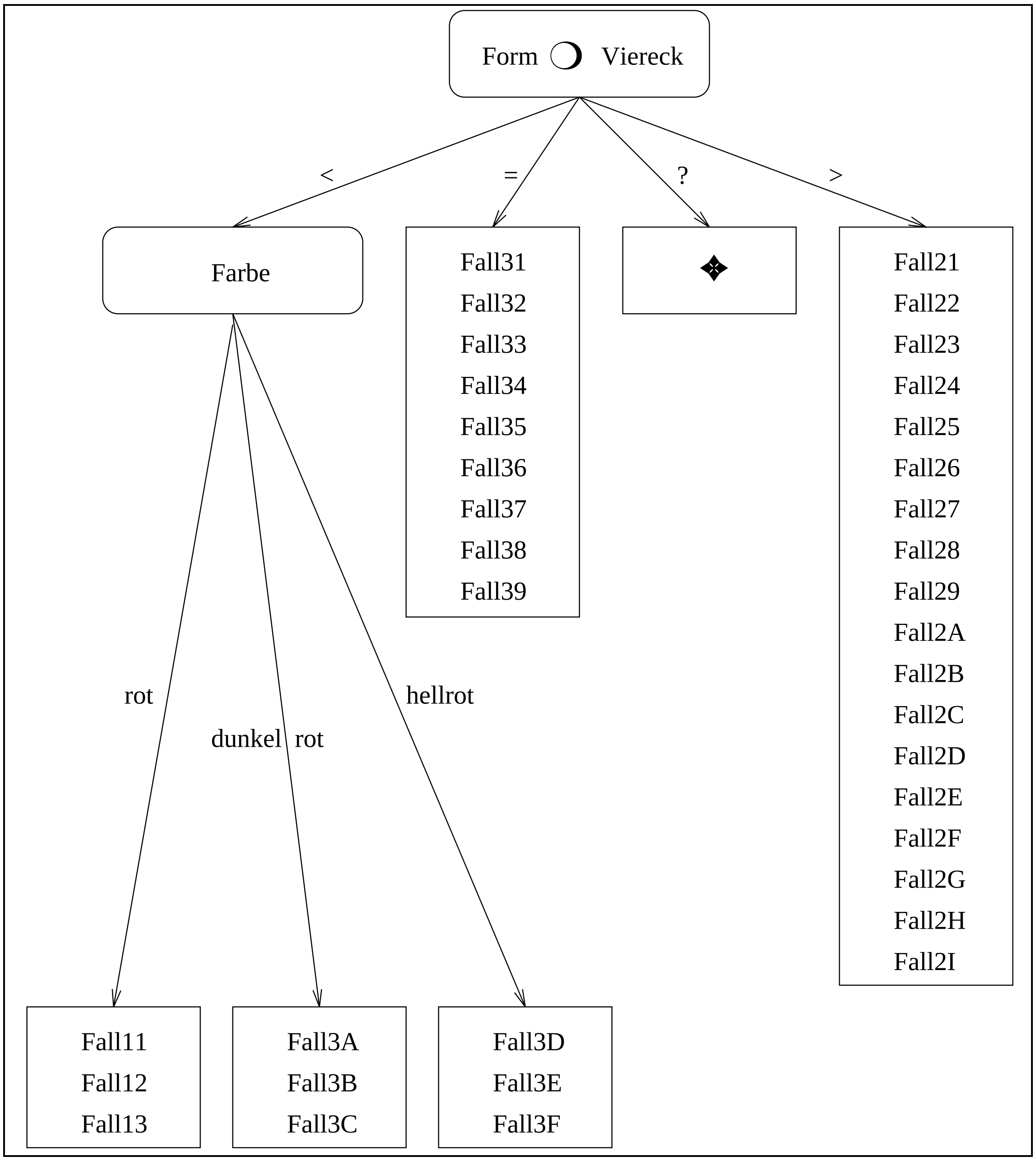


Abbildung 4.1: Mit Entropie generierter k-d-Baum zur Fallbasis

Klasse mehrere "Unterklassen" einführt, welchen man ein eigenes Ähnlichkeitsmaß zuordnet. Die Frage ist nun, wie man solche Unterklassen auffinden kann. Eine erste Idee wäre es, mit Hilfe eines Entscheidungsbaumes solche Unterklassen aufzuspüren, denn diese entsprechen ja gerade den Blättern des Entscheidungsbaums. Diese Beobachtung ist jedoch nur dann richtig, wenn keine unbekannten Werte in den Fällen der Fallbasis auftreten. Ist dies jedoch der Fall, dann müssen Fälle in den Blättern des Entscheidungsbaumes nicht unbedingt zu der entsprechend zugeordneten Unterklasse gehören, da sie auch durch Kopieren des Falls wegen eines unbekannten Wertes in einem Diskriminatorattribut dorthing gelangt sein können. Diese kopierten Fälle können also keiner Unterklasse richtig zugeordnet werden, was diesen ersten Ansatz untauglich macht, da die Intention ja ist, jedem Fall genau ein Ähnlichkeitsmaß – nämlich das zu seiner Unterklasse gehörige – zuzuordnen.

4.6.1 Unterklassenerzeugung

Wie wir oben gesehen haben, eignet sich der Entscheidungsbaum nur bedingt zur Unterklassenerkennung. Wir haben jedoch in unserem Repertoire einen Baum mit ähnlichen Eigenschaften: den k-d-Baum. Wenn dieser mit Entropie aufgebaut wird und Buckets dann erzeugt werden, wenn alle Fälle der aktuellen Fallmenge M die gleiche Klasse haben, dann kann der k-d-Baum wie auch der Entscheidungsbaum disjunktive Konzepte erkennen. Zusätzlich werden jedoch auch Fälle mit unbekannten Attributwerten durch die $\perp$-Äste korrekt eingeordnet, d. h. auch Konzepte mit unbekannten Attributwerten können korrekt erkannt werden. Die Vorgehensweise ist also nun im einzelnen:

1. Erzeuge zur Fallbasis FB den k-d-Baum mit `Split` $\equiv$ $\texttt{Split}_\kappa$ und `CalculateScore` $\equiv$ $\texttt{CalculateScore}^{(ent)}$.
2. Erzeuge für jedes Bucket des k-d-Baums eine Unterklasse U. Diese enthält die Fälle des Buckets.
3. Ordne jeder Unterklasse ein eigenes Ähnlichkeitsmaß sim^U zu.

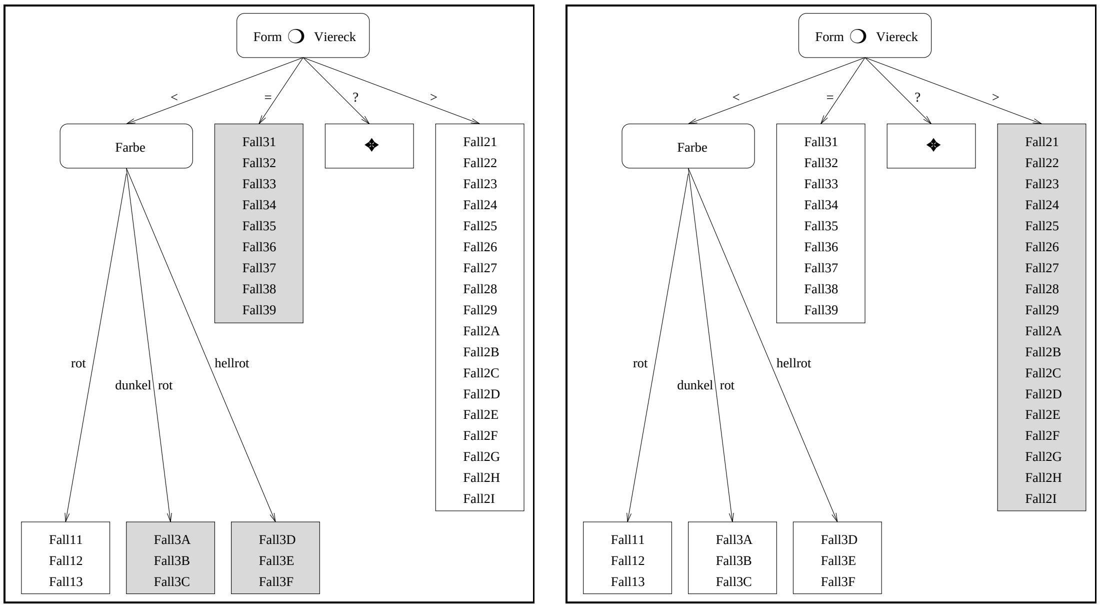


Abbildung 4.2: Unterklassenzerlegung von K_3 (l) und -nichtzerlegung von K_2 (r)

Für das in Abschnitt 4.4 eingeführte Beispiel ist der entsprechende k-d-Baum in Abbildung 4.1 zu sehen. Mit Hilfe von diesem Baum lassen sich fünf Unterklassen erzeugen, wobei zwei von diesen mit Klassen übereinstimmen. Das "schwierige" disjunktive Konzept von Klasse K_3 wird jedoch, wie in Abb. 4.2 zu sehen, in seine drei konjunktiven Teilkonzepte zerlegt. Es ergibt sich im einzelnen:

- Die (Unter-)Klassen $K_1 = \{\text{Fall}_{11}, \ldots, \text{Fall}_{13}\}$ und $K_2 = \{\text{Fall}_{21}, \ldots, \text{Fall}_{2I}\}$.
- Die Unterklassen $K_{31} = \{\text{Fall}_{3D}, \ldots, \text{Fall}_{3F}\}$, $K_{32} = \{\text{Fall}_{3A}, \ldots, \text{Fall}_{3C}\}$ sowie $K_{33} = \{\text{Fall}_{31}, \ldots, \text{Fall}_{39}\}$.

Wie wir am Beispiel erkennen können erlaubt diese Methode also das Erkennen und zerlegen von disjunktiven Konzepten in einfachere, zumeist konjunktive Konzepte. Es ist jedoch zu beobachten, daß nicht unbedingt jedes disjunktive Konzept zerlegt wird. So wird z. B. das Konzept von Klasse K_2 (Form=Fünfeck $\vee$ Form=Sechseck) nicht zerlegt (s. Abb. 4.2). Dies liegt jedoch daran, daß die zwei konjunktiven Teilkonzepte sehr ähnlich sind, da sie Bedingungen für die gleichen Attribute angeben. In diesem Fall kann also ein Gewichtsvektor ausreichen, da die gleichen Attribute relevant sind für jedes konjunktive Teilkonzept.

4.6.2 Ähnlichkeitsmaße für Klassen und Unterklassen

Wie oben schon angeführt wurde wird für jede Unterklasse ein eigenes Ähnlichkeitsmaß zur Verfügung gestellt, d. h. jede Unterklasse erhält einen eigenen Gewichtsvektor. Dieser kann dann mit den in Abschnitt 4.5 vorgestellten Methoden berechnet werden. Für die Einbindung der Unterklassen in die Ähnlichkeitsberechnung $sim(f_1, f_2)$ gibt es nun zwei Möglichkeiten:

1. Es wird die Unterklasse von Fall f_1 bestimmt. Mit Hilfe deren Gewichtsvektor wird die Ähnlichkeit von f_1 zu Fall f_2 berechnet. Die Unterklasse tritt also fast vollständig an die Stelle der Klasse. Lediglich im Klassifizierungs- oder Diagnose-Prozeß wird als Klasse bzw. Diagnose die zur Unterklasse gehörige Klasse vorgeschlagen. Dieses Verfahren wird auch im implementierten System verwendet.
2. Es wird die Klasse K von Fall f_1 bestimmt. Für alle Unterklassen U von K wird $sim^U(f_1, f_2)$ berechnet. Ergebnis ist das Maximum dieser Berechnungen. Diese Vorgehensweise bietet sich an, wenn die Zugehörigkeit der Fälle zu Unterklassen nicht gespeichert werden soll. Da im Laufe des Retrieval-Prozesses aber implizit eine Maximumsbestimmung geschieht, ist diese Vorgehensweise im Retrievalprozeß nicht sinnvoll, da nicht zwischen einzelnen Unterklassen differenziert wird und somit auch schlecht zum Anfragefall passende Fälle mit Hilfe der Maximumsbildung einen hohen Ähnlichkeitswert zum Anfragefall erzielen können.

Für das Beispiel aus 4.4 ergeben sich mit dem Verfahren aus Abschnitt 4.5 für die (Unter)Klassen K_1 und K_2 die gleichen Gewichte wie zuvor. Für die Unterklassen K_{31}, K_{32} und K_{32} ergeben sich folgende Gewichte:

Gewichte	ω_{Farbe}	ω_{Form}	$\omega_{\text{Größe}}$
K_{31}	0.5	0.5	0
K_{32}	0.5	0.5	0
K_{33}	0	1	0

In dem Beispiel ist zu beobachten, daß in den konjunktiven Teilkonzepten von Klasse K_3 die relevanten Attribute (Form & Farbe bzw. nur Form) erkannt wurden. Interessant ist nun noch zu beobachten, wie sich die Inter- und Intra-Klassen-Ähnlichkeit verändert haben:

Klasse	K_1	K_2	K_3
$\overline{sim}^{intra}_{M_{K_i}}$	1	$0.8\overline{3}$	0.78
$\overline{sim}^{inter}_{M_{K_i}}$	0.527273	$0.\overline{3}$	0.471429
$\underline{sim}_{M_{K_i}}$	0.472727	$0.\overline{5}$	0.41228538
Verbesserung[4]	23 %	65 %	31 %

Es scheint so, als hätten wir uns gegenüber der Gewichtsgenerierung für die vorhandenen Klassen nicht verbessert, sonder verschlechtert. Dieser Schein trügt jedoch, wenn man sich die Analyse für jede Unterklasse ansieht. Hier haben sich die Werte erheblich verbessert:

(Unter)Klasse	K_1	K_2	K_{31}	K_{32}	K_{33}
$\overline{sim}^{intra}_{M_{K_i}}$	1	$0.8\overline{3}$	1	1	1
$\overline{sim}^{inter}_{M_{K_i}}$	0.527273	$0.\overline{3}$	$0.\overline{45}$	$0.\overline{45}$	$0.\overline{5}$
$\overline{sim}_{M_{K_i}}$	0.472727	$0.\overline{5}$	$0.\overline{54}$	$0.\overline{54}$	$0.\overline{4}$
Verbesserung	23 %	63 %	74 %	74 %	42 %

4.6.3 Einbindung in das k-d-Baum-Retrieval

Zum Abschluß soll noch geklärt werden, wie Ähnlichkeitsmaße für Unterklassen in die Retrievalprozedur für k-d-Bäume eingebaut werden können. Wie wir schon in Kapitel 3 gesehen haben, ist es möglich innerhalb des k-d-Baums für jede Klasse, Diagnose oder Problemlösung ein eigenes Ähnlichkeitsmaß zu verwenden. Die Vorgehensweise ist nun straightforward, indem die Unterklassen an die Stelle der Klassen treten. Im einzelnen geschieht also folgendes:

- Während der Suche in einem Bucket (lineare Suche) bestimmt die Unterklasse des Falls aus dem Bucket das zu verwendende Ähnlichkeitsmaß.
- In den Bounds-Tests bestimmen die im Teilbaum enthaltenen Unterklassen die zu verwendenden Ähnlichkeitsmaße. Diese Unterklassen treten hier ebenso wie oben an die Stelle der Klassen. Dazu ist es bei der Erzeugung des k-d-Baums natürlich nötig, daß sich die in einem Teilbaum enthaltenen Unterklassen (statt Klassen) gemerkt werden.

4.7 Erweiterung lokaler Ähnlichkeitsmaße mit Unterklassen-Randbedingungen

In den vorigen Abschnitten wurden einige Möglichkeiten der Erweiterung und Berechnung von Ähnlichkeitsmaßen vorgestellt. Dazu wurden Unterklassen generiert, um für diese separat Gewichtsvektoren bestimmen zu können und somit disjunktive Konzepte von Klassen besser repräsentieren zu können. Es gibt aber immer noch einige Defizite:

- Konjunktive Teilkonzepte von disjunktiven Konzepten werden nicht unbedingt in unterschiedliche Unterklassen aufgeteilt. Dies geschieht vor allen Dingen dann nicht, wenn die Teilkonzepte nur zum Teil *unterschiedliche* Bedingungen an die *gleichen* Attribute stellen. Im Beispiel ist das bei Klasse K_2 der Fall. Hier lautet die Bedingung an das Attribut Form zum einen "= Fünfeck" und zum anderen "= Sechseck".
- Es wurden bisher ausschließlich Gewichte berechnet. Es ist jedoch nicht zu verschweigen, daß diese nur Relevanzen des jeweiligen Attributs für die Ähnlichkeit bestimmen können. Der zweite Teil jedes Ähnlichkeitsmaßes, die lokalen Ähnlichkeitsmaße, waren jedoch fest vorgegeben und konnten wider besseres Wissen nicht verändert werden. Das ganze Manko wird deutlich, wenn man zwei Fälle aus der gleichen (Unter)Klasse vergleicht. Nimmt man z. B. die Fälle Fall_{21} und Fall_{2A} aus Klasse K_2 (s. Tabelle 4.1) und vergleicht diese mit dem im vorigen Abschnitt generierten Ähnlichkeitsmaß ($\omega_{\text{Farbe}} = \omega_{\text{Größe}} = 0$, $\omega_{\text{Form}} = 1$), so ergibt dies den Wert 2/3. Wünschenswert wäre hier natürlich der Wert 1, insbesondere wenn man in Betracht zieht, daß sich beim Vergleich von Fall_{21} und Fall_{31} aus K_3 mit dem Ähnlichkeitsmaß von K_2 ebenfalls der Wert 2/3 ergibt. Wenn Fall_{21} nicht in der Fallbasis enthalten wäre und als Anfragefall gestellt würde, so könnte er nicht einmal eindeutig klassifiziert werden! Dies ist natürlich nicht hinnehmbar. Bei genauerem Hinsehen entpuppt sich das

[4] gegenüber gleichverteilter Gewichtung

lokale Ähnlichkeitsmaß sim_{Form} als der Verursacher. So lautet zwar das Konzept von K_2 Form = Fünfeck $\vee$ Form = Sechseck, das lokale Ähnlichkeitsmaß sim_{Form} macht jedoch einen Unterschied zwischen Fünf- und Sechsecken, der jedoch für Klasse K_2 irrelevant ist. Dieses Manko gilt es zu beheben.

Obige Probleme werden nun mit (Unter)Klassen-Randbedingungen angegangen. Die Vorgehensweise ist dabei wie folgt: Die lokalen Ähnlichkeitsmaße sim_i werden zu (unter)klassen-spezifischen lok. Ähnlichkeitsmaßen sim'_i so erweitert, daß sie für das betreffende Attribut A_i *innerhalb* der zugehörigen (Unter)Klasse keinen Unterschied zwischen den in dieser Klasse enthalten Werteausprägungen von A_i machen. Dazu werden sogenannte *Randbedingungen* für Unterklassen verwendet.

4.7.1 Randbedigungen für Klassen und Unterklassen

Es wird nun ein Weg beschrieben, wie man die oben benötigte Erkenntnis der "Irrelevanz" von Form=Fünf- oder Sechseck für Klasse K_2 erkennen kann. Dazu betrachten wir uns wieder den mit Entropie generierten k-d-Baum in Abb. 4.1. Wir können diesen dazu benutzen, ähnlich wie bei Entscheidungsbäumen, Konzeptbeschreibungen für die Buckets – also die weiter oben beschriebenen Unterklassen – zu generieren. Für die (Unter)Klassen des Beispiels ergibt sich somit:

- K_1 : Form < Viereck $\wedge$ Farbe = rot
- K_2 : Form > Viereck
- K_{31} : Form < Viereck $\wedge$ Farbe = hellrot
- K_{32} : Form < Viereck $\wedge$ Farbe = dunkelrot
- K_{33} : Form = Viereck

Ersetzt man noch $<$ durch $\leq$, so ergibt sich:

- K_1 : Form $\leq$ Dreieck $\wedge$ Farbe = rot
- K_2 : Form $\geq$ Fünfeck
- K_{31} : Form $\leq$ Dreieck $\wedge$ Farbe = hellrot
- K_{32} : Form $\leq$ Dreieck $\wedge$ Farbe = dunkelrot
- K_{33} : Form = Viereck

Interessant ist, daß wir mit Hilfe der auf dem Attribut Form definierten Ordnung das Konzept Form=Fünf- oder Sechseck einfacher durch Form$\geq$Fünfeck ausdrücken können.

Die aus k-d-Bäumen generierten Bedingungen sind im Prinzip nichts anderes als die Beschreibung des in dem zur betreffenden Unterklasse gehörenden Buckets enthaltenen Wertebereichs für das jeweilige Attribut. Wie man leicht sieht haben die so generierten Bedingungen folgende Form:

Definition 4.3 ((Unter)Klassenbedingung) *Für die Attribute $A_1, \ldots, A_k$ haben die Bedingungen für jede zu einem Bucket in einem k-d-Baum gehörende Unterklasse folgende Form: $\beta_{i_1} \wedge \ldots \wedge \beta_{i_m}$ mit $1 \leq i_j \leq k$ und $i_j \neq i_l$ für $j \neq l$. Die β_i sind Bedingungen an die Attribute A_i und haben folgende Form:*

$$\beta_i = \begin{cases} A_i \leq v & \textit{mit } v \in W_i & \textit{oder} \\ A_i \geq v & \textit{mit } v \in W_i & \textit{oder} \\ A_i = v & \textit{mit } v \in W_i & \textit{oder} \\ A_i \in [v..w] & \textit{mit } v, w \in W_i \textit{ und } v \leq_i w & \end{cases}$$

Die einzelnen β_i geben also eine Bedingung für Attribut A_i an. Diese kann nun verwendet werden, um das lokale Ähnlichkeitsmaß sim_i wie folgt in sim'_i zu überführen. Der zweite Fall von sim'_i sichert dabei eine Art *Irrelevanz des gewählten Repräsentanten der zugehörigen Unterklasse* beim Ähnlichkeitsvergleich.

$$sim'_i(u_1, u_2) = \begin{cases} 1 & \text{falls } \beta_i \equiv A_i \bigcirc v \text{ und } u_2 \bigcirc v \text{ mit } \bigcirc \in \{\geq, \leq, =\} \text{ oder} \\ & \beta_i \equiv A_i \in [v..w] \text{ und } v \leq_i u_2 \leq_i w \\ sim_i(v, u_2) & \text{falls } \beta_i \equiv A_i \bigcirc v \text{ und } \neg(u_2 \bigcirc v) \text{ mit } \bigcirc \in \{\geq, \leq, =\} \text{ oder} \\ & \beta_i \equiv A_i \in [v..w] \text{ und } u_2 <_i v \text{ oder} \\ & \beta_i \equiv A_i \in [w..v] \text{ und } u_2 >_i v \end{cases} \tag{4.4}$$

Die so definierten sim'_i ersetzen nun die sim_i in der Ähnlichkeitsberechnung, allerdings ist zu beachten, daß die sim'_i jetzt nicht mehr für *jede* (Unter)Klasse gültig sind, sondern jetzt abhängig von der *betrachteten* (Unter)Klasse sind. Diese wird wie üblich durch die (Unter)Klasse des ersten Arguments der Funktion *sim* festgelegt. Zusammengefaßt berechnet sich für zwei Fälle $f_1 =< (v_1, \ldots, v_k), s_1 >$ und $f_2 =< (w_1, \ldots, w_k), s_2 >$ die Ähnlichkeit wie folgt:

1. Bestimme die Unterklasse U von f_1.
2. Bestimme zu U den Gewichtsvektor $(\omega_1, \ldots, \omega_k)$.
3. Bestimme zu U die Unterklassenbedingungen $\beta_{i_1}, \ldots, \beta_{i_m}$.
4. Setze

   $$sim(f_1, f_2) = \sum_{i=1}^{k} \omega_i g_i(v_i, w_i)$$

   mit

   $$g_i(v_i, w_i) = \begin{cases} sim'_i(v_i, w_i) & \text{falls } i \in \{i_1, \ldots, i_m\} \\ sim_i(v_i, w_i) & \text{sonst} \end{cases}$$

Die Vorgehensweise soll nun an einem Beispiel verdeutlicht werden. Vergleichen wir dazu Fall_{2E} und Fall_{31}. Diese sind wie folgt festgelegt:

- Fall_{2E}: Farbe=rot, Form=Sechseck, Größe=normal, (Unter)Klasse=K_2
- Fall_{31}: Farbe=dunkelrot, Form=Viereck, Größe=klein, Unterklasse=K_{33}

Obige Schritte ergeben sich nun wie folgt:

1. $U = K_2$
2. $\omega_{\text{Farbe}} = 0$, $\omega_{\text{Form}} = 1$, $\omega_{\text{Größe}} = 0$
3. $\beta_{\text{Form}} \equiv \text{Form} \geq \text{Fünfeck}$
4.

   $$sim'_{\text{Form}}(u_1, u_2) = \begin{cases} 1 & \text{falls } u_2 \geq \text{Fünfeck} \\ sim_{\text{Form}}(\text{Fünfeck}, u_2) & \text{sonst} \end{cases}$$

   Damit ergibt sich:

   $$\begin{aligned} sim(\text{Fall}_{2E}, \text{Fall}_{31}) &= \omega_{\text{Farbe}} \cdot sim_{\text{Farbe}}(\text{rot}, \text{dunkelrot}) + \\ &\quad \omega_{\text{Form}} \cdot sim'_{\text{Form}}(\text{Sechseck}, \text{Viereck}) + \\ &\quad \omega_{\text{Größe}} \cdot sim_{\text{Größe}}(\text{normal}, \text{klein}) \\ &= 0 + 1 \cdot sim_{\text{Form}}(\text{Fünfeck}, \text{Viereck}) + 0 \\ &= \tfrac{2}{3} \end{aligned}$$

Soweit das Beispiel. Die Frage ist nun, ob mit dieser Konstruktion die weiter oben angeführten Defizite ausgeräumt sind. Das erste Defizit war, daß mehrere konjunktive Teilkonzepte noch *einer* Unterklasse zugeordnet werden können, sodaß für diese nur ein gemeinsamer Gewichtsvektor zur Verfügung steht. Dieser Effekt ensteht durch die Verwendung von $<$ und $>$ im Attributsvergleich. Würde lediglich mit $=$ verglichen, so würde dieser Effekt nicht entstehen. Da aber in den Unterklassenbedingungen ebenfalls $<$ und $>$ (bzw. $\leq$ und $\geq$) erlaubt sind haben wir dieses Manko mit Hilfe der Definition von sim_i' aus Gl. 4.4 in den Griff bekommen, da für Werte von A_j die innerhalb des durch β_j beschriebenen Wertebereichs liegen immer eine 1 als loklaler Ähnlichkeitswert erzeugt wird.

Das zweite Defizit, daß Fälle innerhalb einer Unterklasse beim Vergleich schlechte Ähnlichkeitswerte erzielen, ist i. a. durch die Unterklassenbedingungen auch ausgeräumt. Dies gilt umso mehr, wenn Gewichte von Attributen, für die keine Bedingung angebbar ist, gleich Null sind. Dann tritt nämlich der Idealfall ein, daß Fälle innerhalb einer Unterklasse den Ähnlichkeitswert 1 erzielen. Bei Fällen, die zwar innerhalb einer Klasse, aber nicht innerhalb einer gemeinsamen Unterklasse liegen, gilt obige Aussage i. a. jedoch nicht. Sie gilt erst dann wieder, wenn man eine Maximumsbildung der Unterklassen-Ähnlichkeitsmaße einführt. Da diese Maximumsbildung jedoch durch den Retrieval-Prozeß ersetzbar ist, wird bei einer Anfrage an das System i. a. ein sehr hoher Ähnlichkeitswert zwischen Anfragefall und gefundenem Fall erzielt. Dies trifft natürlich nur auf vollständig angegebene Anfragefälle zu. Bei Fällen mit unbekannten reduziert sich die Ähnlichkeit immer.

Zum Abschluß soll wieder der Bewertungsmaßstab Inter- und Intra-Klassen-Ähnlichkeit zur Bewertung herangezogen werden, um das mit allen neu eingeführten Möglichkeiten (Unterklassen, Gewichtsgenerierung & Unterklassenbedingungen) erzeugte Ähnlichkeitsmaß mit dem einfachen, gleichgewichteten Ähnlichkeitsmaß zu vergleichen. Im einzelnen ergibt sich:

Klasse	K_1	K_2	K_3
$\overline{sim}^{intra}_{M_{K_i}}$	1	1	1
$\overline{sim}^{inter}_{M_{K_i}}$	0.527273	0.5	0.471429
$\overline{\overline{sim}}_{M_{K_i}}$	0.472727	0.5	0.528529
Verbesserung	23 %	49 %	69 %

Die Ähnlichkeitsmaße haben sich also stark verbessert. Dieser Trend wird auch durch die Einzelanalyse der Unterklassen-Ergebnisse bestätigt. Fälle, die mit Fällen derselben Klasse verglichen werden, erzielen nach obiger Intra-Klassen-Ähnlichkeit also immer den Wert 1, während sie beim Vergleich mit Fällen anderer Klassen nach der Inter-Klassen-Ähnlichkeit sehr schlechte Ergebnisse liefern. Aber dies ist ja genau das, was i. a. gewünscht wird.

(Unter)Klasse	K_1	K_2	K_{31}	K_{32}	K_{33}
$\overline{sim}^{intra}_{M_{K_i}}$	1	1	1	1	1
$\overline{sim}^{inter}_{M_{K_i}}$	0.527273	0.5	$0.\overline{45}$	$0.\overline{45}$	$0.\overline{5}$
$\overline{\overline{sim}}_{M_{K_i}}$	0.472727	0.5	$0.\overline{54}$	$0.\overline{54}$	$0.\overline{4}$
Verbesserung	23 %	49 %	74 %	74 %	42 %

4.7.2 Generierungsprozedur für Unterklassenbedingungen

Es soll nun noch der Algorithmus zur Generierung von Unterklassenbedingungen vorgestellt werden. Dieser verwendete folgende Parameter:

- T: Dies ist der aktuelle Teilbaum. Mit $T.A_j$ wird das in dem betreffenden Knoten enthaltene Diskriminatorattribut angesprochen. Ist T kein Bucket so sind $T_<, T_>, T_=$ und $T_\perp$ bzw. T_v für $v \in W_j$ die Söhne, falls es sich bei A_j um ein geordnetes bzw. nicht geordnetes Attribut handelt. Bei inneren Knoten zu geordneten Attributen ist $T.P$ der Partitionswert.

Zu Anfang ist T der durch `CalculateScore`$\equiv$ `CalculateScore`$^{(ent)}$ und `Split`$\equiv$ `Split`$_\kappa$ bestimmte k-d-Baum zur Fallbasis.

- β_j: Aktuelle Unterklassenbedingung für Attribut A_j oder $\perp$.

An Funktionen werden neben den schon bekannten `isBucket` und `isOrdered` noch die folgenden verwendet:

- `join`$(\beta_j, A_j \bigcirc v)$: Diese Funktion fügt zwei Bedingungen an ein Attribut zusammen. Sie läßt sich wie folgt definieren:

$$\mathtt{join}(\beta_j, A_j \bigcirc v) = \begin{cases} A_j \bigcirc v & \text{falls } \beta_j \equiv \perp \text{ oder} \\ & \beta_j \equiv A_j \bigcirc w \text{ und } v \bigcirc w \text{ oder} \\ & \bigcirc \equiv = \\ [v..w] & \text{falls } \bigcirc \equiv \geq \text{ und } \beta_j \equiv A_j \leq w \text{ oder} \\ & \beta_j \equiv [u..w] \\ [w..v] & \text{falls } \bigcirc \equiv \leq \text{ und } \beta_j \equiv A_j \geq w \text{ oder} \\ & \beta_j \equiv [w..u] \\ \perp & \text{sonst} \end{cases}$$

  Dabei ist $\bigcirc \in \{\leq, \geq, =\}$.

- `succ` und `pred`: Diese geben den Nachfolger bzw. Vorgänger des Arguments zurück. Bei nicht abzählbaren Wertebereichen (z. B. $\Re$) wird das Argument zurückgegeben.

Aufgerufen wird der Algorithmus mit **CreateConstraints**$(T, (\perp, \ldots, \perp))$. Als Ergebnis liefert der Algorithmus eine Menge von Tupeln $(T, (\beta_1, \ldots, \beta_k))$. T ist dabei ein Bucket – enthält also die Fälle genau einer Unterklasse – und die β_i sind Bedingungen (oder $\perp$ falls keine existiert) für Attribut A_i für die dem Bucket entsprechende Unterklasse.

Algorithmus 4.1 (CreateConstraints$(T, (\beta_1, \ldots, \beta_k))$)

IF `isBucket`(T) THEN
    $result := \{(T, (\beta_1, \ldots, \beta_k))\}$
ELSE
    IF `isOrdered`$(T.A_j)$ THEN
        /* A_j *ist ein geordnetes Attribut* */
        $result :=$
            CreateConstraints$(T_<, (\beta_1, \ldots, \beta_{j-1}, \mathtt{join}(\beta_j, A_j \leq \mathtt{pred}(T.P)), \beta_{j+1}, \ldots, \beta_k)) \cup$
            CreateConstraints$(T_=, (\beta_1, \ldots, \beta_{j-1}, \mathtt{join}(\beta_j, A_j = T.P), \beta_{j+1}, \ldots, \beta_k)) \cup$
            CreateConstraints$(T_>, (\beta_1, \ldots, \beta_{j-1}, \mathtt{join}(\beta_j, A_j \geq \mathtt{succ}(T.P)), \beta_{j+1}, \ldots, \beta_k)) \cup$
            CreateConstraints$(T_\perp, (\beta_1, \ldots, \beta_{j-1}, \perp, \beta_{j+1}, \ldots, \beta_k))$
    ELSE
        /* A_j *ist ein ungeordnetes Attribut* */
        $result := \emptyset$
        FOR $v \in W_j$ DO
            $result := result \cup$
                CreateConstraints$(T_v, (\beta_1, \ldots, \beta_{j-1}, \mathtt{join}(\beta_j, A_j = v), \beta_{j+1}, \ldots, \beta_k))$
        ENDFOR
    ENDIF
ENDIF
RETURN $result$

4.7.3 Retrievalverhalten von Ähnlichkeitsmaßen mit Unterklassenbedingungen

Zum Abschluß dieses Abschnitts soll noch das Verhalten des Retrieval-Algorithmuses mit einem Ähnlichkeitsmaß *mit* Unterklassenbedingungen und *generierten* Gewichten verglichen werden mit einem *gleichgewichteten* Ähnlichkeitsmaß *ohne* Unterklassen und Bedingungen. Als Beispiel wird wieder die aus Kapitel 3 bekannte PKW-Datenbank verwendet. Diese, wie auch der Anfragefall, ist im Anhang näher beschrieben. Der verwendete k-d-Baum wird natürlich mit Entropie generiert, da aus diesem ja die Unterklassen und -bedingungen extrahiert wurden, und dieser ohne angepaßtes Ähnlichkeitsmaß nur mäßige Retrievaleffizienz erbrachte. Die Ergebnisse sind in Abb. 4.3 auf Seite 79 zu ersehen. Mit dem gleichgewichteten Ähnlichkeitsmaß müssen schon bei $n = 20$ gesuchten Fällen rund 160 ($\approx 75\%$) Fälle betrachtet werden. Mit dem bedingten Ähnlichkeitsmaß müssen in diesem Fall nur rund 70 ($= 34\%$) Fälle betrachtet werden. Auch wenn bei $n = 30$ die Anzahl auf rund 125 stark ansteigt, bleibt sie jedoch rund 30 Fälle unter dem Niveau des gleichverteilten Ähnlichkeitsmaß.

Bei den Anzahlen der BWB- und BOB-Tests setzt sich dieser Trend fort. Auch bei diesen bleibt die Anzahl mit dem bedingten Ähnlichkeitsmaß für $n \leq 20$ bei etwa der Hälfte der Anzahl mit gleichgewichtetem Ähnlichkeitsmaß.

4.8 Lernverfahren

Fallbasiertes Schließen ist i. a. ein dynamischer Prozeß insofern, als daß das in der Fallbasis FB gespeicherte Wissen nicht statisch ist, sondern erweiterbar sein muß. Eine typische Situation entsteht zum Beispiel bei der fallbasierten Diagnose oder Klassifikation. Hier wird dem Benutzer zum Anfragefall ein ähnlichster Fall und dessen Diagnose bzw. Klasse als Lösung vorgeschlagen. Dieser Vorschlag kann aber mit den Wünschen des Benutzers nicht vereinbar sein. Wird beispielsweise der Defekt eines Bauteils diagnostiziert, welches der Benutzer aber schon überprüft und für funktionsfähig befunden hat, so muß das System zwei Leistungen vollbringen können:

- **Leistung 1**: Es müssen weitere Lösungsvorschläge generiert werden können bis der Benutzer zufriedengestellt ist, oder das System jeden ihm möglichen Vorschlag angeboten hat.
- **Leistung 2**: Wird zu einem späteren Zeitpunkt die gleiche Anfrage wieder gestellt, so sollen die "Fehler" der derzeitigen Anfragesitzung nicht wiederholt werden. Das System muß also aus seinen Fehlern lernen können.

4.8.1 Inkrementeller Lösungsvorschlag durch Fallausschluß

Es soll nun kurz dargestellt werden, wie Leistung 1 realisiert werden kann. Die Idee hier ist dual zum MAC/FAC-Vorschlag (s. [13]). So werden in einem ersten Arbeitsgang alle auf keinen Fall in Frage kommenden Fälle aus der Retrieval-Struktur k-d-Baum (*nicht* FB) ausgeblendet, sodaß sie im zweiten Arbeitsgang, dem eigentlichen Retrieval, nicht mehr gefunden werden können. Damit ist garantiert, daß immer die n ähnlichsten Fälle gefunden werden, die die unerwünschten Eigenschaften nicht haben. Dazu müssen folgende Erweiterungen im System erfolgen:

- Es müssen Fälle in den Buckets des k-d-Baums aktivierbar und deaktivierbar sein, um sie für die Suche ein- und ausschließen zu können. Dies ist eine sehr einfache Erweiterung.
- Es werden einige Index-Listen über die Fälle benötigt, die z. B. enthalten, welche Fälle einer bestimmten Klasse angehören, oder welche Fälle einen bestimmten Wert in einem Attribut haben. Je nach Leistungsfähigkeit des Ausschlußmechanismus ist hier einiger Speicherbedarf vonnöten.

Im implementierten System wurde ein sehr umfangreicher Ein- und Ausschluß-Mechanismus eingebaut, mit dem der Benutzer schon vor einer Diagnosesitzung Anforderungen an die Attribute der zu findenden Fälle stellen kann. Vom System wird intern der Ausschluß-Mechanismus nur dazu verwendet, um vom Benutzer abgelehnte Diagnosen von der Suche auszuschließen.

4.8.2 Lernen von Ähnlichkeitsmaßen und Fallbasiserweiterung

Es soll nun dargestellt werden, wie Leistung 2 von oben realisiert werden kann. Die Situation ist dabei folgende: Es wurde vom Benutzer ein Anfragefall spezifiziert. Zu diesem ist der ähnlichste Fall mit Hilfe des k-d-Baums gesucht worden. Dessen Diagnose, Klasse oder Problemlösung ist dem Benutzer vorgeschlagen worden. Dieser Vorschlag wird jedoch vom Benutzer als unmöglich eingestuft. Die Aufgabe ist nun einen neuen Vorschlag zu generieren. Dazu ist es notwendig, alle Fälle mit den bisher abgelehnten Klassen/Diagnosen von der Suche wie oben schon angedeutet auszuschließen. Dieser Vorgang kann sich mehrmals wiederholen bis eine dem Benutzer genehme Klasse/Diagnose vorgeschlagen wurde oder keine weiteren Vorschläge mehr möglich sind. Nach einer erfolgreich beendeten Sitzung kann dann ein Lernverfahren angestoßen werden. Dies soll nun skizziert werden.

Eingabe: Anfragefall F_q mit, wie oben beschrieben, akzeptierter Klasse/Diagnose D.

1. Füge Fall F_q mit D in eine Menge M_{gen} ein. Falls M_{gen} *genügend* Fälle enthält, dann fahre fort mit dem nächsten Punkt, sonst STOP.

2. Erzeuge zu $FB \cup M_{gen}$ einen mit Entropie erzeugten k-d-Baum wie es in den vorigen Abschnitten beschrieben wurde. Extrahiere aus diesem neue Unterklassen, Unterklassen-Bedingungen und Gewichte wie beschrieben. Das Neuerzeugen des k-d-Baums kann auch durch ein inkrementelles Einfügen (s. [14]) ersetzt werden. Fahre fort mit dem nächsten Punkt.

3. Die Fälle aus M_{gen} können nun in die Fallbasis FB übernommen werden. Dies ist besonders sinnvoll für die Fälle, die im ersten Anlauf des Systems falsch klassifiziert bzw. diagnostiziert wurden. Diese Fälle können dann auch in den aktuellen k-d-Baum übernommen werden, um sie auch finden zu können.

   Ist die Fallbasis jedoch schon sehr umfangreich, so genügt es auch die Fälle in M_{gen} zu belassen.

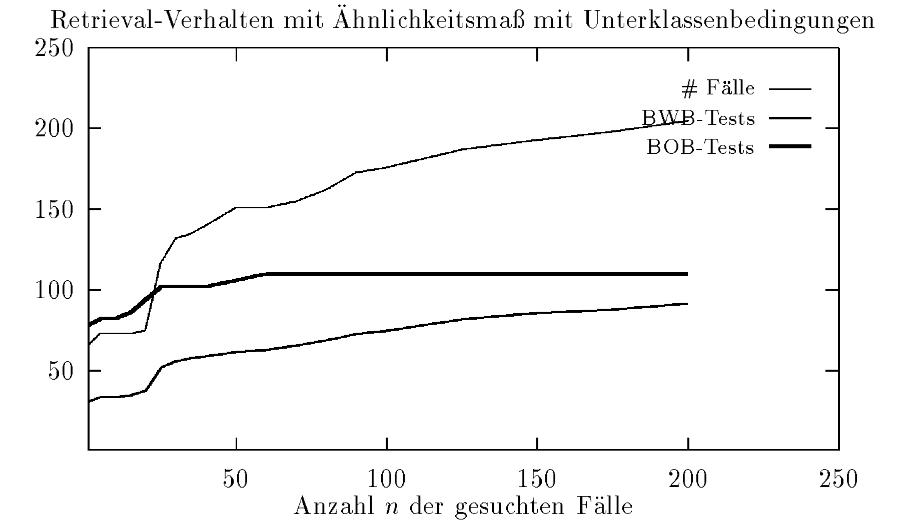
   STOP.

Obiger Algorithmus hat noch einen Parameter, welcher bestimmt, wann *genügend* Parameter in M_{gen} gesammelt worden sind. Dieser sollte so ausgelegt werden, daß die Neuberechnung der Ähnlichkeitsmaße und k-d-Bäume das System nicht zu stark beansprucht. Es kann auch so verfahren werden, daß das Lernverfahren vom Benutzer angestoßen wird.

4.9 Zusammenfassung

In diesem Kapitel wurde eine Möglichkeit zur automatischen Generierung von leistungsfähigen Ähnlichkeitsmaßen aufgezeigt, welche zur Klassenerkennung eingesetzt werden können. Dazu wurden die Inter- und Intra-Klassen-Ähnlichkeit analysiert und in ein Verfahren zur Generierung von Gewichten umgesetzt.

Oft reicht jedoch die Leistungsfähigkeit von gewichteten Summen als Ähnlichkeitsmaß nicht aus, da eine Klasse z. B. nur mit Hilfe eines disjunktiven Konzeptes beschreibbar ist. Für diesen Fall sind Verfahren vorgestellt worden, die es ermöglichen, solche disjunktiv beschriebenen Konzepte zu erkennen und in das Ähnlichkeitsmaß einzubetten. Diese basieren zum einen auf der Verwendung von mehreren Gewichtsvektoren pro Klasse und zum anderen auf einer Veränderung der lokalen Ähnlichkeitsmaße mit Hilfe von Bedingungen an die Attribute. Die so veränderten lokalen Ähnlichkeitsmaße verwenden die Bedingungen allerdings nicht so restriktiv wie normale Konzepterkennungsfunktionen, welche i. a. mit Gleichheit oder mit Prädikaten die Zugehörigkeit des Falles zu einem Konzept entscheiden, sondern sie setzen auf den vorgegebenen lokalen Ähnlichkeitsmaßen auf und ermöglichen so eine unscharfe Zugehörigkeitsaussage des Falles zum Konzept. Die Bedingungen werden ähnlich zu der Regelextraktion aus Entscheidungsbäumen aus einem mit Entropie generierten k-d-Baum generiert. Sie haben dadurch jedoch ein größeres Repertoire an Vergleichsoperatoren, da zusätzlich zu $=$ noch $<$ und $>$ verwendet werden können. Außerdem können so auch Fallbasen mit in einigen Attributen unbekannten Fällen verarbeitet werden. Die so generierten Ähnlichkeitsmaße wirken sich i. a. positiv auf das Retrieval-Verhalten des k-d-Baums aus.



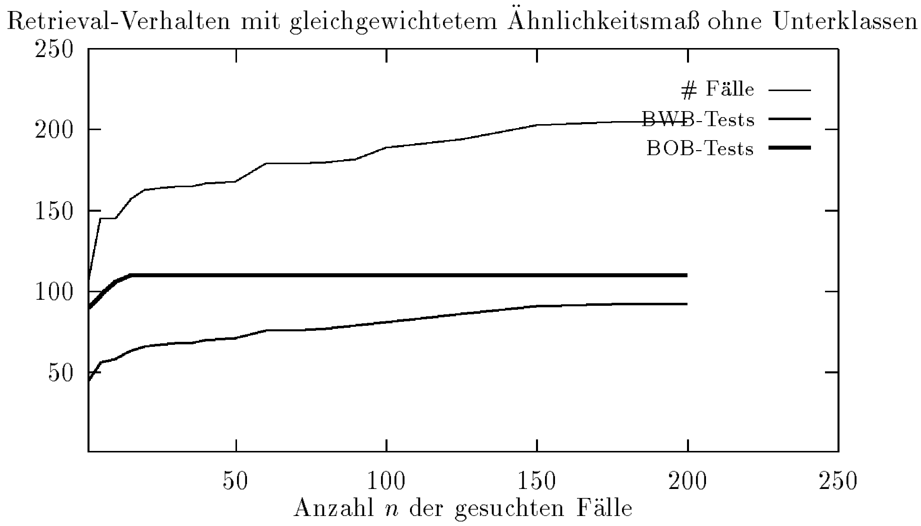


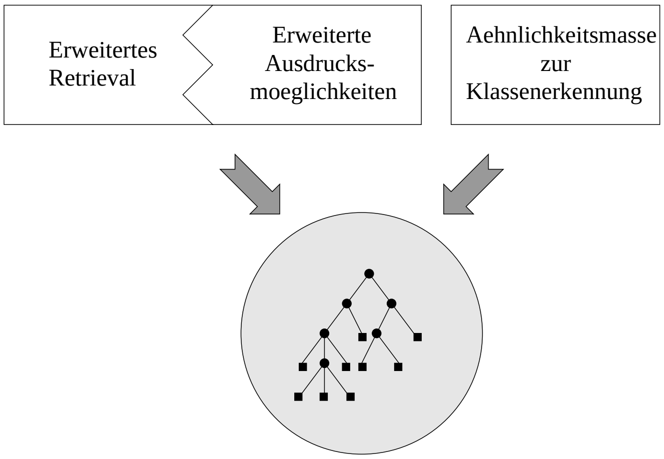
Abbildung 4.3: Retrievalverhalten im mit Entropie generierten k-d-Baum

Kapitel 5

Ausblick

In dieser Arbeit ist vorgestellt worden, wie fallbasiertes und induktives Schließen in Verwendung von k-d- und Entscheidungsbäumen sinnvoll kombiniert werden kann. Die Idee dabei war die Erweiterung des fallbasierten Ansatzes um das Feature der induktiven Konzepterkennung. Dazu mußte als erstes die "Infra-Struktur" k-d-Baum so erweitert werden, daß auch komplexere Fälle mit geordneten und ungeordneten Attributen verwendet werden können, um den Bereich der Konzepterkennung auf Fällen mit diskreten Attributen auch abdecken zu können. Diese Aufgabe wurde gleich so ausgeführt, daß auch unbekannte Attribute sicher und effizient handhabbar sind, was bisher keiner der beiden Ansätze effizient ermöglichte. Als angenehmer Nebeneffekt trägt die Erweiterung der k-d-Bäume zu einer erheblich gesteigerten Retrieval-Geschwindigkeit bei.

Die wichtigste Aufgabe war danach sicherlich die Entwicklung von automatischen Generierungsmethoden für Ähnlichkeitsmaße, welche effiziente Konzepterkennung ähnlich den induktiv generierten Konzepterkennungsfunktionen erlauben, ohne deren Nachteile z. B. bei verrauschten Daten zu übernehmen.



Integrierter k-d- und Entscheidungsbaum

Diese Kombination von erweitertem k-d-Baum-Retrieval und leistungsfähiger Ähnlichkeitsmaße ergibt trotz automatischer Generierung ein sehr dynamisches System, ganz im Gegensatz zu TDIDT-Verfahren, da daß generierte Ähnlichkeitsmaß im Prinzip als Vorschlag gesehen werden kann und sich natürlich beliebig verändern läßt.

Diese Dynamik setzt sich in dem recht einfache Lernverfahren des Systems fort, welches auf Fallbasiserweiterung und Neugenerierung der Ähnlichkeitsmaße beruht. An dieser Stelle sind noch zu wenige Erfahrungen zu dessen Beurteilung vorhanden. Eine Erweiterung dieses Verfahrens könnte durchaus auch die partielle Entfernung von Fällen aus der Fallbasis umfassen, wenn ein genügend gutes Ähnlichkeitsmaß gepaart mit guten Repräsentanten in der Fallbasis vorhanden ist.

Das vorgestellte System ist in dieser Form sicherlich schon in der Lage, einen einfachen, auf Entscheidungsbäumen basierenden Klassifikationsmechanismus zu ersetzen und bietet alle bekannten Vorteile eines fallbasierten Systems.

Kapitel 6

Verwandte Arbeiten

6.1 Octrees und direkte Indizierung

Wie wir in Kapitel 2 gesehen haben, eignen sich die ursprünglichen k-d-Bäume gut für das Retrieval von Fällen mit Attributen, deren Wertebereich geordnet ist, während sich Entscheidungsbäume eher für Attribute mit endlichem, diskreten Wertebereich[1] eignen. Stottler, Henke und King unterscheiden in [26] den Wertebereich in ähnlicher Weise. Sie unterscheiden quantitative, also numerische, und qualitive Wertebereiche, wie bspw. Farbnamen. Für beide Attributtypen stellen sie in einem ersten Schritt separate Retrieval-Algorithmen vor, welche dann in einem zweiten Schritt miteinander kombiniert werden, um Fälle mit beiden Sorten von Attributen bearbeiten zu können.

6.1.1 Retrieval mit Octrees

Octrees sind eine Datenstruktur, welche ursprünglich aus der Computergraphik stammt und dort u. a. zur Strukturierung von 3D-Graphikszenen verwendet wird (s. [29]). Dazu wird eine Szene im dreidimensionalen Raum rekursiv in 8 achsenparallele Würfel geteilt, bis maximal eine vorgebare Anzahl von primitiven Elementen (Punkte, Dreiecke, etc.) in einem Würfel liegt. Da die Grenzen der Würfel achsenparallel sind, ist relativ leicht feststellbar, in welchem Würfel ein vorgebarer Punkt liegt. Genau diese Eigenschaft wird beim Retrieval von Fällen ausgenutzt, wenn man sich auf Fälle mit quantitativen Wertebereichen von Attributen beschränkt. Denn dann entsprechen die Fälle Punkten im $\Re^k$, wobei k wieder die Anzahl der Attribute angibt. Da i. a. die Fälle mehr als 3 Attribute haben, werden verallgemeinerte Würfel, sogenannte Hypercubes, zur Raumstrukturierung verwendet, wobei der Raum in 2^k achsenparallele Hypercubes eingeteilt wird. Abb. 6.1 zeigt einige Fälle im $\Re^2$ mit zugehörigem Octree, welcher im 2D-Fall *Quadtree* heißt, und korrespondierendem k-d-Baum. Wie man sieht sind einige Rechtecke leer. Jedoch ist mindestens ein Nachbar eines leeren Rechtecks nicht leer.

Das Retrieval eines Anfragefalles besteht nun aus der Suche des den Anfragefall enthaltenden Hyperwürfels. Die darin enthaltenen Fälle sind gute Kandidaten für die Rolle der nächsten Nachbarn des Anfragefalls. Es kann jedoch auch nötig werden (wie auch im angegebenen Beispiel), daß die angrenzenden Würfel durchsucht werden müßen.

Die Ähnlichkeit von zwei Fällen (Punkten) wird mit Hilfe einer p-Norm berechnet. Da der oben beschriebene einfache Retrieval-Mechanismus nicht mit einer unterschiedlichen Gewichtung der einzelnen Attribute (Dimensionen) funktioniert, wird in einer Preprozessing-Phase jeder Attributwert mit der entprechenden Gewichtung multipliziert, bevor über den so modifizierten Fällen der Octree erzeugt wird.

[1] Diese Unterscheidung der Wertebereiche ist natürlich nicht ganz disjunkt, denn auch diskrete Wertbereiche können geordnet sein. Von dieser Eigenschaft wird dann bei Entscheidungsbäumen allerdings kein Gebrauch gemacht

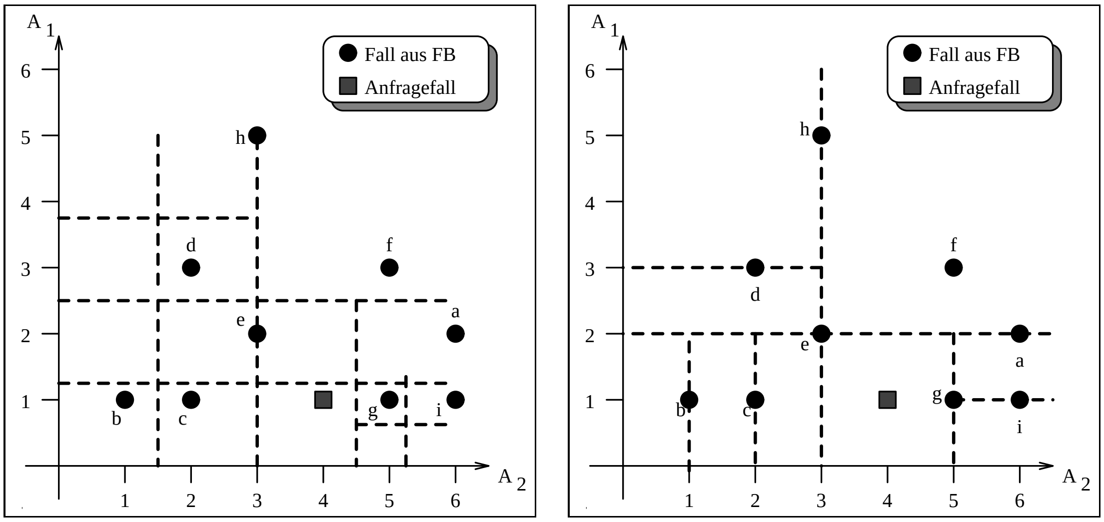


Abbildung 6.1: Partitionierung des Fallraums mit einem Quadtree (l) und k-d-Baum (r)

6.1.2 Retrieval mit direkter Indizierung

Nun soll der in [26] vorgeschlagene Retrieval-Algorithmus für Fälle mit qualitativen Attributen vorgestellt werden. Ähnliche Ansätze finden sich in [11].

Das verwendete Ähnlichkeitsmaß betrachtet nur Übereinstimmungen in den jeweiligen Attributwerten (engl. *exact matches*) und verwendet zwar auch unterschiedliche Gewichtungen der Attribute, verwendet aber keine lokalen Ähnlichkeitsmaße, wie sie bspw. schon in Kapitel 2 verwendet wurden. Genau daraus bezieht der Algorithmus seine Effizienz, da mit Hilfe der Attributwerte die Fälle indiziert werden. Dies wird nun im folgenden detailiert vorgestellt.

Definition 6.1 (Attributindizierung) *Seien $A_{j_1}, \ldots, A_{j_m}$ $(1 \leq j_1 < \ldots < j_m \leq k)$ quantitative Attribute mit den Wertebereichen $W_{j_1}, \ldots, W_{j_m}$ und $M \subseteq FB$ eine Menge von Fällen mit diesen Attributen. Dann heißt $\mathcal{I}^M_{j_1,\ldots,j_m} \subseteq (W_{j_1} \times \ldots \times W_{j_m}) \times M$ Attributindizierung für M und $A_{j_1}, \ldots, A_{j_m}$, wenn gilt:*

$$\forall f \in M \text{ mit } f = <(w_1, \ldots, w_k), s> : (w_{j_1}, \ldots, w_{j_m}, f) \in \mathcal{I}^M_{j_1,\ldots,j_m}$$

In der praktischen Anwendung müssen Attributindizierungen für verschiedene Kombinationen von Attributen berechnet werden (siehe unten). Deshalb kommt ein Fall im allgemeinen in mehreren Indizierungen vor. Um nicht viele verschiedene Duplikate ein und desselben Falls speichern zu müssen, ist es vorteilhaft, wenn in $\mathcal{I}^M_{j_1,\ldots,j_m}$ statt dem Fall nur eine Referenz auf diesen gespeichert wird. Dann kann der Fall bspw. in einer Datenbank gespeichert werden und nur wenn nötig in den Arbeitsspeicher geladen werden.

Fahren wir nun fort mit dem Indizierungs-Algorithmus. Dieser berechnet für jede der $2^k - 1$ möglichen Kombination von Attributen die zugehörige Attributindizierung und ordnet diese in einem Indizierungsbaum an, der nun definiert werden soll.

Definition 6.2 (Indizierungsbaum) *T heißt Indizierungsbaum zu den Attributen $A_1, \ldots, A_k$ und der Fallbasis FB, wenn $T = T^{ind}(\{A_1\}, k)$ ist, mit $T^{ind}(\{A_{j_1}, \ldots, A_{j_m}\}, x)$ wie folgt:*

- *Falls $x = k$: Blattknoten T^{ind}_b, welcher $\mathcal{I}^{FB}_{j_1,\ldots,j_m}$ beinhaltet.*
- *Falls $x < k$: Innerer Knoten T^{ind}_i, welcher $\mathcal{I}^{FB}_{j_1,\ldots,j_m}$ beinhaltet, sowie Verweise auf*
  - *den linken Sohn $T^{ind}(\{A_{j_1}, \ldots, A_{j_m}, A_{j_{m+1}}\}, x + 1)$ und*

– *den rechten Sohn* $T^{ind}(\{A_{j_1}, \ldots, A_{j_{m-1}}, A_{j_{m+1}}\}, x+1)$

enthält.

Ein Indizierungsbaum enthält also alle möglichen Kombinationen von Attributindizierungen. Die Kombination der Attribute kann auch Abb. 6.2 für das Beispiel von 4 Attributen $A_1, \ldots, A_4$ entnommen werden.

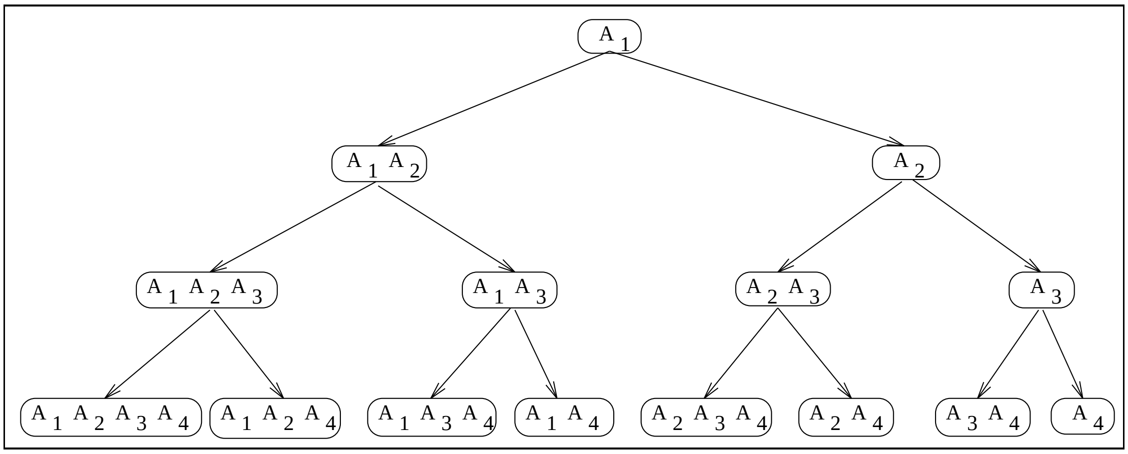


Abbildung 6.2: Attributkombinationen im Indizierungsbaum

Nun soll betrachtet werden, wie der Indizierungsbaum zum Retrieval von Fällen benutzt werden kann.

1. Ist $<(w_1, \ldots, w_k), \perp>$ der Anfragfall, so ist der einfachste Ansatz, indem man alle Knoten des Baumes durchläuft und mit Hilfe von $\mathcal{I}^{FB}_{j_1,\ldots,j_m}$ des Knotens für alle Fälle f mit $(w_{j_1}, \ldots, w_{j_m}, f) \in \mathcal{I}^{FB}_{j_1,\ldots,j_m}$ die Ähnlichkeit mit dem Anfragefall berechnet. Ein Vergleich der Ergebnisse ergibt dann die n ähnlichsten Fälle (bezüglich des wie oben eingeschränkten Ähnlichkeitsmaßes) zum Anfragefall.

2. Eine geschicktere Methode vermeidet das Durchsuchen aller Knoten des Indizierungsbaums, indem der Baum in einer bestimmten Reihenfolge durchsucht wird. Gestartet wird an der Wurzel des Baums. Dabei wird in jedem Knoten wie unter Punkt 1 versucht, *einen* Fall per Indizierung zu finden. Ist dieser Versuch erfolgreich, so geht es rekursiv genauso mit dem linken Sohn des Knotens weiter. Im Falle des Mißerfolgs geht es rekursiv mit dem rechten Sohn weiter. Dieser Prozeß wird solange fortgesetzt, bis ein Blatt erreicht wird. Dann werden zum letzten erfolgreichen Knoten wie unter Punkt 1 die Ähnlichkeiten der indizierbaren Fälle zum Anfragefall berechnet und gespeichert. Dann muß zum letzten erfolgreichen Knoten zurückgekehrt werden *(Backtracking)* und wir fahren mit dem rechten Sohn wie oben beschrieben fort, bis letztlich kein Backtracking zu einem zuletzt erfolgreichen Knoten mehr möglich ist. Dann werden aus den gespeicherten Fällen und Ähnlichkeitswerten die n ähnlichsten berechnet. Der Effizienzgewinn dieser Methode beruht also auf dem möglichen Auschluß einer Reihe von Teilbäumen.

3. Eine weitere Verfeinerung zu Punkt 2 ergibt sich, wenn man zusätzlich Teilbäume wie folgt von der Suche ausschließen kann: Speichert man in jedem Blatt den erreichbaren Ähnlichkeitswert, der sich über die Summe der Gewichte der enthaltenen Attribute[2] berechnen läßt, und speichert in den inneren Knoten das Maximum dieser Werte der beiden Söhne, so kann

[2] Dies ist natürlich nur möglich, weil in diesem Ansatz nur *exact matches* betrachtet werden

beim Retrieval der Teilbaum ab dem betrachteten Knoten ausgeschlossen werden, wenn der bisher n-beste Fall schon einen besseren Ähnlichkeitswert als der wie oben beschrieben berechnete Wert des betrachteten Knoten hat.

6.1.3 Kombiniertes Retrieval

Um Fälle mit q qualitativen und $k-q$ quantitativen Attributen bearbeiten zu können, schlagen Stottler, Henke und King eine einfache Kombination der beiden oben beschriebenen Retrieval-Algorithmen vor. Dazu wird jeder Fall getrennt nach einem der beiden Typen von Attributen betrachtet. Mit dem qualitativen "Anteil" der Fälle wird ein Indizierungsbaum wie oben beschrieben erzeugt. Für jeden Knoten dieses Baumes wird dann für die von ihm in $\mathcal{I}^{FB}_{j_1,\ldots,j_m}$ referenzierten Fälle ein $(k-q)$-dimensionaler Octree erzeugt.

Um die n ähnlichsten Fälle zu finden, muß *jeder* Knoten des Indizierungsbaums durchsucht werden. Dazu wird in dem betrachteten Knoten zuerst der Anteil der qualitativen Attribute am Ähnlichkeitswert berechnet. Anschließend wird das Retrieval im Octree fortgesetzt und ergibt den Anteil der quantitativen Attribute am Ähnlichkeitswert für jeden der in dem Knoten referenzierten Fälle. Zu beachten ist, daß auch die Knoten des Indizierungsbaums betrachten werden müssen, welche in keiner der jeweils enthaltenen qualitativen Attribute mit dem Anfragefall übereinstimmen.

6.1.4 Bewertung

Der in [26] vorgeschlagene und oben beschriebene Ansatz des Retrievals bietet interessante Ansätze, wie z. B. den Indizierungsbaum, geht jedoch von erheblichen Einschränkungen aus:

- Es können keine lokalen Ähnlichkeitsmaße verwendet werden. Für die qualitativen Attribute ist nur der "exact match" und für quantitative Attribute die p-Norm vorgesehen.
- Die Anzahl der Attribute muß sehr klein sein, da der Speicherbedarf des Indizierungsbaums exponentiell von der Anzahl der (qualitativen) Attribute abhängt. Ebenso hängt der Speicherbedarf des Octrees exponentiell von der Anzahl der (quantitativen) Attribute ab.
- Der kombinierte Ansatz benutzt den Indizierungsbaum nur zur Zerteilung der Fallbasis[3]. Es wird beim Retrieval kaum Vorteil aus der aufwendigen Indizierung gezogen.

Beschränkt man sich auf wenige, qualitative Attribute und exact matches, so ist der Ansatz des Indizierungsbaums recht interessant, auch wenn der Unterschied zu leistungsfähigem Datenbankretrieval dann sehr klein wird.

Der Octree leidet sehr stark an der starren Unterteilung des Raumes in gleichgroße Hyperwürfel. Dies führt, wie in Abb. 6.1 zu sehen ist, zu vielen unnötigen leeren Hyperwürfeln. Auch fehlt ein Ansatz beim Retrieval, der Teile des Octrees vom Retrieval ausschließt.

6.2 Induktives Lernen aus wenigen klassifizierten Fällen

Ein Ansatz, der partiell Parallelen zu dem in dieser Arbeit vorgestellten Verfahren und Methoden aufzeigt, wird in [31] von W. Emde vorgestellt. Dieser hat sich in seiner Arbeit folgender Aufgabe gewidmet: Es sollen möglichst gute Konzeptbeschreibungen aus einer sehr kleinen Menge von *klassifizierten* Fällen generiert werden. Dieser Ansatz geht somit auf den ersten Blick in die entgegengesetzte Richtung wie übliche fallbasierte Generierungsmethoden für Konzeptbeschreibungen, welche im allgemeinen eine große Zahl an klassifizierten Fällen benötigen.

[3] in nicht unbedingt disjunkte Teile

6.2.1 Vorgehensweise

Die Vorgehensweise ist dabei recht einfach: Die kleine Menge der klassifizierten Fälle wird um eine große Anzahl an *unklassifzierten* Fällen erweitert. Auf diese wird ein Conceptual Clustering Algorithmus zur Erkennung und Beschreibung "natürlicher" Klassen angewendet. Die Entdeckung dieser Klassen kann z. B. mit einem Ähnlichkeitsmaß geschehen. Die so erzeugten Klassen können dann anhand der wenigen klassifizierten Fälle partiell identifiziert, d. h. nachklassifiziert werden. Somit wird die Anzahl der klassifizierten Fälle durch Einsatz von Conceptual Clustering erhöht und die dazu eingesetzten Verfahren erzeugen auch für jede Klasse eine Beschreibung, wobei i. a. nicht disjunkte Klassen, sondern eine Klassenhierarchie erzeugt wird und somit auch mehrere Beschreibungen für eine Klasse zur Verfügung stehen, die sich zwischen der allgemeinsten und der speziellsten Beschreibung bewegen. Die beste Beschreibung wird dann mit Hilfe einer Bewertungsfunktion ausgewählt, welche auf den Anzahlen der durch die Klasse abgedeckten und insgesamt vorhandenen positiven und negativen Fallbeispielen beruht. Dieses Vermehrungsverfahren von klassifizierten Fällen kann u. U. zu einem verbesserten Klassifizierungsverhalten der nach obigem Schema erzeugten Konzeptbeschreibungen führen.

6.2.2 Bewertung

Auch wenn der erste Eindruck etwas anderes suggeriert: Auch dieses Lernverfahren beruht auf einer möglichst großen Zahl von Fallbeispielen. Von Vorteil ist jedoch, daß diese nur zu einem kleinen Teil klassifiziert sein müssen, da mit Hilfe eines Conceptual Clustering Verfahrens die fehlende Klasseninformation nachgereicht werden kann. Zwar widmet sich Emde sehr genau der Auswahl der mit den CC-Verfahren erzeugten Klassenbeschreibungen, jedoch ist auch zu beachten, daß das Verfahren insgesamt sehr stark von den verwendeten Clustering-Heuristiken abhängt, welche das eigentliche "Wissen" über die Klassifizierung enthalten. An deren Stelle kann im übrigen auch ein k-d-Baum treten, da dieser mit Hilfe der Bucketgenerierung mit mittlerer Ähnlichkeit auch "natürliche" Klassen(hierarchien) erkennen kann. Bei Verwendung des Partitionierungsverfahrens Optimale Fallcluster ist das verwendete Ähnlichkeitsmaß der einzigste Parameter im so konstruierten Clustering-Verfahren, und ist deshalb relativ leicht kontrollierbar und nachvollziehbar. Es fehlt lediglich noch die Generierung von Klassenbeschreibungen. Aber dazu kann man durchaus den in Kapitel 4 vorgestellten Generierungsmechanismus für (Unter)Klassenbedigungen anwenden.

Eine Frage drängt sich jedoch noch auf: Woher stammen die unklassifizierten Fälle? In den in [31] angeführten Beispielen wurde die vorhandene Klassifizierungsinformation der zusätzlichen Beispiele einfach ignoriert. Das macht natürlich in der Praxis wenig Sinn, da bei vorhandener Klassifizierung die bekannten Verfahren eingesetzt werden können. Im übrigen liegen die Fälle sowieso als Erfahrungswissen, also klassifiziert, vor. Die Frage ist also, ob ein normales CBR-System, welches Fälle inkrementell einbinden kann und somit auch aus (ehemals) unklassifizierten Fällen lernt, nicht das einfachere Verfahren ist, da hier realistische Fälle im Umgang mit dem System erzeugt werden und direkt eingebunden werden können.

6.3 Voronoi-Diagramme

Voronoi-Diagramme sind mit Sicherheit die eindeutigste Darstellung von Nachbarschaftsbeziehungen im Datenraum. Sie sollen nun kurz vorgestellt werden.

6.3.1 Allgemeine Definition

Die Retrieval-Aufgabe erfordert, daß man zu einem Anfragefall einen (evtl. mehrere) nächsten Nachbarfall findet, welcher am ähnlichsten zum Anfragefall ist. Stellen wir uns nun vor, jeder Fall wäre durch einen Punkt im $\Re^2$ beschreibbar und die verwendete Ähnlichkeitsfunktion wäre aus einer p-Norm ableitbar, z. B. der üblichen euklidischen Norm. Dann hat jeder Fall eine Art Einzugsbereich von Anfragefällen (die ja dann auch einfach Punkte im $\Re^2$ sind), für die der betreffende

Fall der nächste Nachbar, also der ähnlichste Fall ist. Grenzt man für jeden Fall der Fallbasis seinen Bereich ab, in dem er der nächste Nachbar ist, so ergeben sich unter obigen Voraussetzungen konvexe Polygone. Diese (im $\Re^2$) wunderbar anschauliche Darstellung von Nachbarschaftsbeziehungen ist nichts anderes als ein *Voronoi-Diagramm*. Voronoi-Diagramme sind also kurz gesagt nichts anderes als die Ansammlung von Einzugsbereichen jedes Falls der Fallbasis. Diese Einzugsbereiche werden auch als *Voronoi-Region* bezeichnet. Sie lassen sich für ein Ähnlichkeitsmaß sim und eine Fallbasis FB wie folgt festlegen:

Definition 6.3 (Voronoi-Region) *Für jeden Fall $f \in FB$ ergibt sich die zugehörige Voronoi-Region $R(f)$ im Datenraum $\mathcal{W} = W_1 \times \ldots \times W_k$ wie folgt:*

$$R(f) = \{g \in \mathcal{W} | \forall h \in FB - \{f\} : sim(f,g) \geq sim(h,g)\}$$

Voronoi-Diagramme lassen sich also sehr gut zum Fallretrieval einsetzen, wenn man zu jedem Fall des Datenraums die Voronoi-Region bestimmen kann, da dann nur noch die Voronoi-Region gefunden werden muß, in welcher der Anfragefall liegt. Der zugehörige Fall der Fallbasis ist dann der nächste Nachbar.

Voronoi-Diagramme stützen sich in der Literatur (s. [2] und [3]) noch sehr stark auf eine geometrische Veranschaulichung der Fälle, welche als Punkte im $\Re^k$ vorausgesetzt werden. Auch als Ähnlichkeits- bzw. Abstandsmaß wird eine Metrik vorausgesetzt. Dies ist eine sehr starke Einschränkung der Fallstruktur und ist i. a. in dieser Form nicht gegeben. Zwar gibt es Möglichkeiten zur Transformation diskreter Wertebereiche, diese können jedoch die Berechnungskomplexität stark erhöhen. Die Einschränkung auf metrische Abstandsmaße wird in der Literatur aber immer vorausgesetzt. Wie man anhand obiger Definition sieht, ist der Begriff Voronoi-Region bzw. -Diagramm jedoch unabhängig von der Wahl des Datenraums und der Ähnlichkeitsfunktion. Das Problem ist lediglich, daß die bekannten Berechnungs- und Retrieval-Algorithmen sehr stark auf die geometrische Veranschaulichung des Datenraums und der Ähnlichkeitsfunktion abgestimmt sind. Für allgemeine Voronoi-Diagramme existieren noch keine Berechnungs- und Retrieval-Algorithmen.

6.3.2 Voronoi-Diagramme und k-d-Bäume

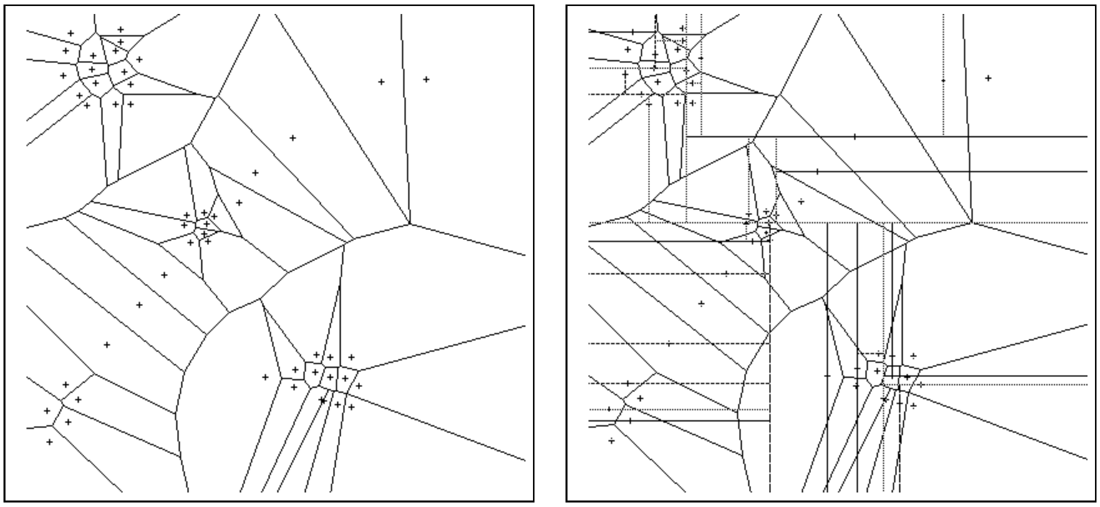


Abbildung 6.3: Voronoi-Diagramm (l) mit überlagertem k-d-Baum (r)

Vergleicht man Voronoi-Diagramme mit k-d-Bäumen (s. Abb. 6.3), so sieht man, daß k-d-Bäume im Hinblick auf die Nachbarschaftsbeziehungen den Datenraum wesentlich ungeschickter aufteilen. Dieser Nachteil erwächst aber gleichzeitig wieder zu einem Vorteil, da man durch die achsenparallele Aufteilung des Raums ein sehr einfaches Retrieval ermöglicht, während bei Voronoi-Diagrammen noch ein zusätzliches Preprocessing zur Bestimmung der Voronoi-Region nötig ist.

Die Eleganz der Voronoi-Diagramme im Vergleich zu k-d-Bäumen ist nicht verwunderlich, wenn man beachtet, daß sie im Prinzip genau die Struktur der Fallbasis festlegen, welche beim Retrieval erkannt werden muß. Betrachtet man sich jedoch die Definition, so sieht man, daß sie im allgemeinen Fall mit Abstand den höchsten Berechnungsaufwand erfordern werden, da alle möglichen Nachbarschaftsbeziehungen schon vorberechnet werden müssen, die dann natürlich im Retrieval elegant verwendet werden können.

6.3.3 Bewertung

Voronoi-Diagramme erkennen die Struktur der Fallbasis, welche beim Retrieval auch erkannt werden muß und bieten somit den höchsten Grad an Strukturierung. Ein großer Nachteil ist jedoch, daß für eine beliebige Domäne mit qualitativen und quantitativen, geordneten und ungeordneten Attributen kein allgemeines Berechnungs- und Retrieval-Verfahren zur Verfügung steht. Nur für Fälle aus dem $\Re^k$ sind Verfahren bekannt, die jedoch einige starke Restriktionen (z. B. an k) erfordern. Eine genauere Diskussion findet sich dazu in [8].

Weitere Nachteile ergeben sich daraus, daß Voronoi-Diagramme nur Anfragen nach einer *festen* Anzahl n von nächsten Nachbarn befriedigen können. Auch das Vorhandensein von Unbekannten in Fällen der Fallbasis ist schwierig zu handhaben.

6.4 Kombiniertes Retrieval mit Indizierung und Ähnlichkeitsvergleich

In [25] schlagen Tanaka und Sueda einen Retrieval-Mechanismus vor, welcher sich in den MAC/-FAC-Ansatz (s. [13]) einordnen läßt. So unterteilt sich das Retrieval in zwei Phasen:

1. Mit Hilfe *direkter Indizierung* werden uninteressante Fälle vom Retrieval ausgeschlossen.
2. Die übrig bleibenden Fälle werden mit Hilfe eines gewichteten Ähnlichkeitsmaßes mit dem Anfragefall verglichen.

Beide Phasen sind in einen Steuerungsalgorithmus eingebunden, welcher sichert, daß in Phase 1 nicht Fälle ausgeschlossen werden, welche in Phase 2 vielleicht höher bewertet würden, als andere Fälle, die diese Phase erreichen.

6.4.1 Steuerungsalgorithmus

Der Steuerungsalgorithmus hat neben der Fallbasis FB und dem Anfragefall F_q noch eine minimale Anzahl n_{min} und eine maximale Anzahl n_{max} von zu findenden Fällen als Eingabe. Er liefert als Ergebnis eine Menge von Fällen und deren jeweiligen Ähnlichkeitswert zum Anfragefall.

Algorithmus 6.1 (retrieve_case($F_q, FB, n_{min}, n_{max}$))

```
δ := δ_0
REPEAT
    I := generate_min_str_index(F_q, δ)
    n := get_population(I, FB)
    IF n_max < n THEN
        increase(δ)
    ENDIF
    IF n < n_min THEN
        decrease(δ)
    ENDIF
UNTIL n_min ≤ n ≤ n_max
M' := strict_matching(I, FB)
RETURN similarity_assessment(δ, M')
```

Die Idee des Algorithmus ist also folgende: Es wird iterativ eine Mindest-Ähnlichkeitsschwelle δ berechnet, sodaß ungefähr n Fälle mit $n_{min} \leq n \leq n_{max}$ mit Hilfe des Index I aus der Fallbasis extrahiert werden können. Dabei schätzt die Funktion `get_population`, wieviele Fälle ein Index extrahiert. Sie kann auf bekannten statistischen Daten der verwendeten Datenbank beruhen, welche i. a. zur Indizierung und Speicherung der Fälle benutzt wird. Die Berechnung von δ kann bei bekannter Verteilung der Fälle im Datenraum aber eventuell auch direkt erfolgen. Bei passendem δ werden mit dem zugehörigen Index I die interessanten Fälle der Fallbasis aus der Datenbank extrahiert und anschließend mit dem Anfragefall verglichen.

6.4.2 Indexgenerierung

Dreh- und Angelpunkt des obigen Algorithmus bildet die korrekte Generierung eines Index, der zum einen gute Fallkandidaten nicht vom Ähnlichkeitsvergleich ausschließt, zum anderen aber so wenige Fälle wie mögich zum Ähnlichkeitsvergleich zuläßt. Es wird dabei für jedes Attribut eine Menge von möglichen Werten generiert, die in guten Kandidaten vorkommen können. Dabei wird zwischen qualitativen und quantitativen Attributen wie folgt unterschieden (sei $(w_1, \ldots, w_k)$ der Anfragefall):

- Ist A_i ein qualitatives Attribut, so wird davon ausgegangen, daß der Wertebereich taxonomisch geordnet ist. Generalisiere oder spezialisiere w_i auf eine nicht näher spezifizierte Weise und gebe diese(n) Wert(e) als Möglichkeiten für A_i an.
- Ist A_i ein quantitatives Attribut, so gebe das Interval $[l, u]$ als Möglichkeiten für A_i an. Dabei sind l und u so gewählt, daß $sim_i(w_i, l) = \delta = sim_i(w_i, u)$.

6.4.3 Bewertung

Der Ansatz von Tanaka und Sueda zeigt starke Ähnlichkeit zum bekannten MAC/FAC-Ansatz auf, wobei er versucht, für die MAC-Phase eine korrekte und effiziente Auswahl der Fälle zu treffen. Diese Auswahl wird mit üblichem DB-Retrieval erledigt, sodaß für dieses Indizes zur Auswahl generiert werden müssen. Diese Generierung bleibt zumindest in [25] für qualitative Attribute noch sehr schleierhaft. Die Frage ist, ob sich für qualitative Attribte, die nicht taxonomisch geordnet sind, überhaupt effiziente Indizes generieren lassen, da im allgemeinen doch sehr viele Werteausprägungen eines Attributs in Erwägung gezogen werden müssen. Da scheint der Indizierungsansatz von Stottler, Henke und King (s. Abschnitt 6.1) doch vielversprechender zu sein. Die Generierung von Wertebereichen für quantitative Attribute hat eine gewisse Ähnlichkeit zu den Bounds-Tests der k-d-Bäume und ist sicherlich sinnvoll.

Letzlich soll noch darauf hingewiesen werden, daß der MAC/FAC-Ansatz mit k-d-Bäumen sehr effizient implementiert werden kann, da sich der Ausschluß von Fällen aus dem Baum und Retrieval (s. Abschnitt 4.8.1) sehr einfach realisieren läßt. Der weitere Vorteil ist dann, daß in der FAC-Phase nicht lineare Suche verwendet werden muß, sondern der k-d-Baum zur Suche benutzt werden kann.

Anhang A

PKW-Datenbank

Über das UCI Repository of Machine Learning Databases and Domain Theories sind Fallbasen über Internet frei zugänglich. Die enthaltenen Daten stammen von echten Anwendungen oder aus Literaturbeispielen. Die verwendete PKW-Datenbank ist folgendermaßen erreichbar:

Zugang	:	anonymous-ftp
Adresse	:	ics.uci.edu
Pfad	:	pub/machine-learning-databases/imports-85

Die PKW-Datenbank enthält 205 Fälle, welche die Typen der im Jahr 1985 in die USA importierten PKWs beschreiben. Die Fälle werden durch folgende 25 Attribute beschrieben:

Nr.	Attribut	Wertebereich
1	symboling	-3,-2,-1,0,1,2,3
2	make	Alfa-Romeo, Audi, BMW, Chevrolet, Dodge, Honda, Isuzu, Jaguar, Mazda, Mercedes-Benz, Mercury, Mitsubishi, Nissan, Peugeot, Plymouth, Porsche, Renault, Saab, Subaru, Toyota, Volkswagen, Volvo
3	fuel type	diesel, gas
4	aspiration	std, turbo
5	number of doors	2,4
6	body style	hardtop, wagon, sedan, hatchback, convertible
7	drive wheels	4wd, fwd, rwd
8	engine location	front, rear
9	wheel base	$[86.6, 120.9]$
10	length	$[141.1, 208.1]$
11	width	$[60.3, 72.3]$
12	height	$[47.8, 59.8]$
13	curb weight	$[1488, 4066]$
14	engine type	dohc, dohcv, l, ohc, ohcf, ohcv, rotor
15	number of cylinders	2,3,4,5,6,8,12
16	engine size	$[61, 326]$
17	fuel system	1bbl, 2bbl, 4bbl, idi, mfi, mpfi, spdi, spfi
18	bore	$[2.54, 3.94]$
19	stroke	$[2.07, 4.17]$
20	compression ratio	$[7, 23]$
21	horsepower	$[48, 288]$
22	peak rpm	$[4150, 6600]$
23	city mpg	$[13, 49]$
24	highway mpg	$[16, 54]$
25	price	$[5118, 45400]$

Anhang B

Messungsparameter

Im folgenden werden die in Kapitel 3 und z. T. auch in Kapitel 4 verwendeten Einstellungen, Eingaben und Parameter angegeben.

B.1 Feste Einstellungen beim k-d-Baum-Aufbau

Alle mit dem neuen System generierten k-d-Bäume erzeugen Buckets, wenn die betrachtete Fallmenge nur noch eine Problemlösung/Klasse enthält, es gilt also $\texttt{Split} \equiv \texttt{Split}_\kappa$. Da dies im alten System noch nicht möglich war, wurden dort Buckets mit jeweils einem Fall erzeugt. Es gilt also dort $\texttt{Split} \equiv \texttt{Split}_1$

B.2 Anfragefall

Als Anfragefall wurde immer folgender, nicht in der Fallbasis enthaltene Fall verwendet:

Nr.	Attribut	Wert	Nr.	Attribut	Wert
1	symboling	$\perp$	14	engine type	dohc
2	make	$\perp$	15	number of cylinders	4
3	fuel type	gas	16	engine size	130
4	aspiration	std	17	fuel system	spdi
5	number of doors	4	18	bore	3.5
6	body style	convertible	19	stroke	2.68
7	drive wheels	rwd	20	compression ratio	9.5
8	engine location	front	21	horsepower	115
9	wheel base	89	22	peak rpm	5400
10	length	170	23	city mpg	20
11	width	65	24	highway mpg	28
12	height	50	25	price	$\perp$
13	curb weight	2500			

Von den 25 Attributen wurden die Attribute 1 (symboling) und 2 (make) als nicht diskriminant betrachtet. Desweiteren wurde der Wert von Attribut 1 als Klasse des Falls verwendet.

B.3 Lokale Ähnlichkeitsmaße

Für die Attribute 3, 4, 5, 6, 7, 8, 14, und 17 wurde ein einfaches $\{0,1\}$-Ähnlichkeitsmaß verwendet, d. h. $sim_i(u,v) = 1 \Leftrightarrow u = v$ und $sim_i(u,v) = 0 \Leftrightarrow u \neq v$.

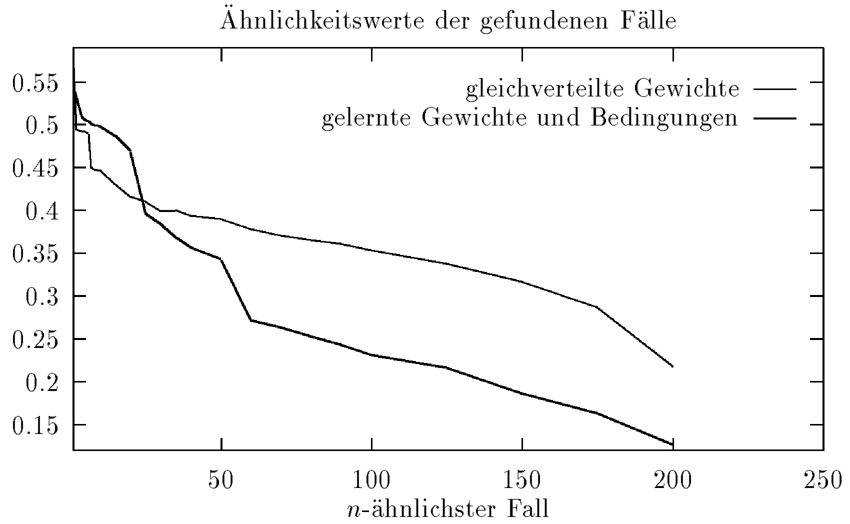
Für die quantitativen Attribute $9, \ldots, 13$, 16, $18, \ldots, 25$ wurde der Betrag der Distanz zur Festlegung eines Ähnlichkeitsmaßes verwendet, d. h. $sim_i(u,v) = 1 - \frac{|u-v|}{1+|u-v|}$.
Für Attribut 15 wurde folgendes lokale Ähnlichkeitsmaß verwendet:

number of doors	2	3	4	5	6	8	12
2	1	$\frac{5}{6}$	$\frac{2}{3}$	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{6}$	0
3	$\frac{5}{6}$	1	$\frac{5}{6}$	$\frac{2}{3}$	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{6}$
4	$\frac{2}{3}$	$\frac{5}{6}$	1	$\frac{5}{6}$	$\frac{2}{3}$	$\frac{1}{2}$	$\frac{1}{3}$
5	$\frac{1}{2}$	$\frac{2}{3}$	$\frac{5}{6}$	1	$\frac{5}{6}$	$\frac{2}{3}$	$\frac{1}{2}$
6	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{2}{3}$	$\frac{5}{6}$	1	$\frac{5}{6}$	$\frac{2}{3}$
8	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{2}{3}$	$\frac{5}{6}$	1	$\frac{5}{6}$
12	0	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{2}{3}$	$\frac{5}{6}$	1

Die Attribute 5, $9, \ldots, 13$, 15, 16, $18, \ldots, 25$ wurden mit üblichen Ordnung $<$ auf $\Re$ geordnet. Die restlichen Attribute sind ungeordnet.

B.4 Berechnete Ähnlichkeitswerte

Im folgenden sind die errechneten Ähnlichkeitswerte aufgetragen, welche mit gleichgewichtetem bzw. bedingtem (s. Kap. 4) Ähnlichkeitsmaß erzielt wurden. Angegeben ist dabei der Ähnlichkeitswert des n-ähnlichsten Falls zum Anfragefall. Zu bemerken ist, daß alle Werte unter 0.59, also recht niedrig, liegen. Dies ist ein Indiz für eine realistische Anfrage, da kein Anfragefall verwendet wurde, der "fast gleich" in der Fallbasis vorkommt.



Literaturverzeichnis

[1] Klaus-Dieter Althoff, Stefan Weß, Brigitte Bartsch-Spörl, Dietmar Janetzko, Frank Maurer & Angi Voß. *Fallbasiertes Schließen in Expertensystemen: Welche Rolle spielen Fälle für wissensbasierte Systeme?* In *Künstliche Intelligenz* KI(4/92), 14–21, FBO-Verlag, Baden-Baden, (1992)

[2] F. Aurenhammer. *Gewichtete Voronoi-Diagramme – Geometrische Deutung und Konstruktions-Algorithmen.* Institut für Informationsverarbeitung, Technische Universität Graz,(1984)

[3] F. Aurenhammer. *Voronoi-Diagrams – A Survey.* Institut für Informationsverarbeitung, Technische Universität Graz,(1988)

[4] Michel Manago, Eric Auriol, Ralph Traphöner, Stefan Weß & Noël Contruyt. *Report on Integration of Induction and Case-Based Reasoning (INRECA).* ESPRIT project 6322, (1993)

[5] Hannes Öchsner. *Mehrdimensionale Zugriffspfadstrukturen für das ähnlichkeitsbasierte Retrieval von Fällen.* Diplomarbeit Universität Kaiserslautern, (1992)

[6] Hannes Öchsner & Stefan Weß. *Ähnlichkeitsbasiertes Retrieval von Fällen durch assoziative Suche in einem mehrdimensionalen Datenraum.* In K.-D. Althoff, S. Weß, B. Bartsch-Spörl & D. Janetzko (Eds.) *Proceedings: Ähnlichkeit von Fällen beim fallbasierten Schließen*, SEKI-Report, Universität Kaiserslautern, (1992)

[7] Edwina L. Rissland, Janet L. Kolodner & David Waltz. *Cased-Based Reasoning from darpa: Machine Learning program plan.* In J. Hammond *Proceedings: Case-Based Reasoning Workshop, San Mateo, California, 1989*, 1–13. Pensacola Beach, Florida, USA, May 31 - June 1, (1989)

[8] Werner Schirp. *Einsatz von Voronoi-Diagrammen für das Ähnlichkeits-Retrieval in fallbasierten Systemen.* Projektarbeit Universität Kaiserslautern, (1993)

[9] Stefan Weß, Klaus-Dieter Althoff & Guido Derwand. *Using k-d-Trees to Improve the Retrieval Step in Case-Based Reasoning.* In M. M. Richter, S. Weß, K.-D. Althoff, F. Maurer (Eds.). *First European Workshop for Case-Based Reasoning – EWCBR-93*Kaiserslautern, Germany (1994)

[10] Stefan Weß. PATDEX/2 – *Ein System zum fallfokusierenden Lernen in technischen Diagnosesituationen.* Diplomarbeit Universität Kaiserslautern (1990) und SEKI-Working Paper SWP-91-01 (1991)

[11] Stefan Weß. PATDEX – *Inkrementelle und wissensbasierte Verbesserung von Ähnlichkeitsurteilen in der fallbasierten Diagnostik.* In *Tagungsband 2. deutsche Expertensystemtagung XPS-93.* Springer Verlag, Hamburg, (1993)

[12] Stefan Weß & Christoph Globig. *Case-Based and Symbolic Classification Algorithms – A Case Study using Version Space –.* In M. M. Richter, S. Weß, K.-D. Althoff, F. Maurer (Eds.). *First European Workshop for Case-Based Reasoning – EWCBR-93*Kaiserslautern, Germany (1994)

[13] Dedre Gentner & Kenneth D. Forbus. MAC/FAC*: A model of similarity-based retrieval.* In *Proceedings of the 13th Annual Conference of the Cognitive Science Society*, Seite 504–509, (1991)

[14] Frank Goebel. *Optimiertes Retrieval von Fallbeispielen mit k-d-Bäumen.* Diplomarbeit Universität Kaiserslautern, (1993).

[15] J. H. Friedman, J. L. Bentley & R. A. Finkel. *An algorithm for finding best matches in logarithmic expected time.* In *ACM Transactions on math. Software*, 3, Seite 209–226, (1977).

[16] L. H. Koopmans. *Introduction to Contemporary Statistical Methods.* Second Edition, Duxbury Press, Boston, (1987)

[17] Michael M. Richter & Oliver Wendel. *Lernende Systeme, Teil I: Symbolische Methoden.* Vorlesungsskript Universität Kaiserslautern, (1993)

[18] Michael M. Richter. *Classification and Learning of Similarity Measures.* In *Proceedings der 16. Jahrestagung der Gesellschaft für Klassifikation e. V.*, Springer Verlag, (1992)

[19] J. R. Quinlan. *Induction of decision trees.* In *Machine Learning* 1, 1, Seite 81–106, (1986)

[20] J. R. Quinlan. *Simplifying decision trees.* International Journal of Man-Machine Studies 27, 4, Seite 221–234, (1987)

[21] J. R. Quinlan. *Decision trees and multivalued attributes.* In J. E. Hayes, D. Michie, und J. Richards (Eds.) *Machine Intelligence* 11, Oxford University Press, (1988)

[22] J. R. Quinlan. *Unknown attribute values in induction.* In A. M. Segre (Ed.) *Proceedings of the 6th International Machine Learning Workshop.*, San Mateo, CA, Morgan Kaufmann, (1989)

[23] J. R. Quinlan. *Discovering rules by induction from large collections of examples.* In D. Michie (Ed.) *Expert Systems in the Micro Electronic Age*, Edinburgh University Press, (1979)

[24] J. R. Quinlan. *Probablistic decision trees.* In Y. Kodratoff, R. Michalski (Eds.) *Machine Learning* 3, Seite 140–152

[25] T. Tanaka & N. Sueda. *Combining strict matching and similarity assessment for retrieving appropriate cases efficiently.* In *Working Notes, AAAI Spring Symposium Series*, Symposium: *Case-based reasoning and information retrieval– Exploring the opportunities for technology sharing*, Seite 112–119, (März 1993)

[26] R. Stottler, A. L. Henke & J. A. King. *Rapid retrieval algorithms for case-based reasoning.* In N. S. Sridharan (Ed.) *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, Volume 1, Seite 233–237, (1989)

[27] P. E. Utgof & C. E. Brodley. *An incremental method for finding multivariate splits for decision trees.* In *Proceedings of the 7th International Conference of Machine Learning*, Seite 58–65, Morgan Kaufmann Publishers, San Mateo, CA

[28] P. E. Utgof. ID5 – *An incremental* ID3. In *Proceedings of the 5th International Conference of Machine Learning*, Seite 107–120, Morgan Kaufmann Publishers, San Mateo, CA

[29] J. D. Foley, A. van Dam, S. K. Feiner & J. F. Hughes. *Computer Graphics: Principles and Practice.* 2nd Edition, Addison-Wesley, (1990)

[30] K. Mehlhorn. *Data Strutures and Algorithms 3: Multi-dimensional Searching and Computational Geometry.* Springer Verlag, (1984)

[31] W. Emde. *Inductive Learning of Characteristic Concept Descriptions from Small Sets of Classified Examples.* In F. Bergamo, L. de Raedt (Eds.) *Proceedings of the 7th European Conference on Machine Learning*, Springer Verlag, LNAI, (1994)

[32] D. H. Fisher. *Knowledge Acquisition via Incremental Conceptual Clustering.* In *Machine Learning* 2, Seite 139–172, (1987)

[33] D. H. Fisher. *Knowledge Acquisition via Incremental Conceptual Clustering.* Tech. Report No. 87-22 (doctoral dissertation), Department of Information and Computer Science, University of California, Irvine, CA, (1987)

[34] M. Lebowitz. *Concept Learning in a Rich Domain: Generalisation-Based Memory.* In R.S. Michalski, J. G. Carbonell, T. M. Mitchell (Eds.) *Machine Learning: An Artificial Intelligence Approach* (Vol. 2), Los Altos, CA, Morgan Kaufmann, (1986)

[35] J. Rahmel & A. von Wangenheim. *Case-based Diagnosis with Kohonen Networks.* In *Proceedings of the Workshop on Neural Networks: Techniques and Applications*, Liverpool Workshops on Computer Science, (September 1993)