Brazdil, P. and Gama, J., 1998

Constructive Induction on Continuous Spaces

In Liu, H./Motada, H.:
Feature Extraction Construction and Selection, A Data Mining Perspective.
Chapter 18, pages S.289-302; Kluwer Academic Publishers Boston, Dordrecht, London.

11.1.2000; M. Villwock
Seminar „Wissensentdeckung – Entdeckungswissenschaft“

Zum Beispiel: der Iris-Datensatz

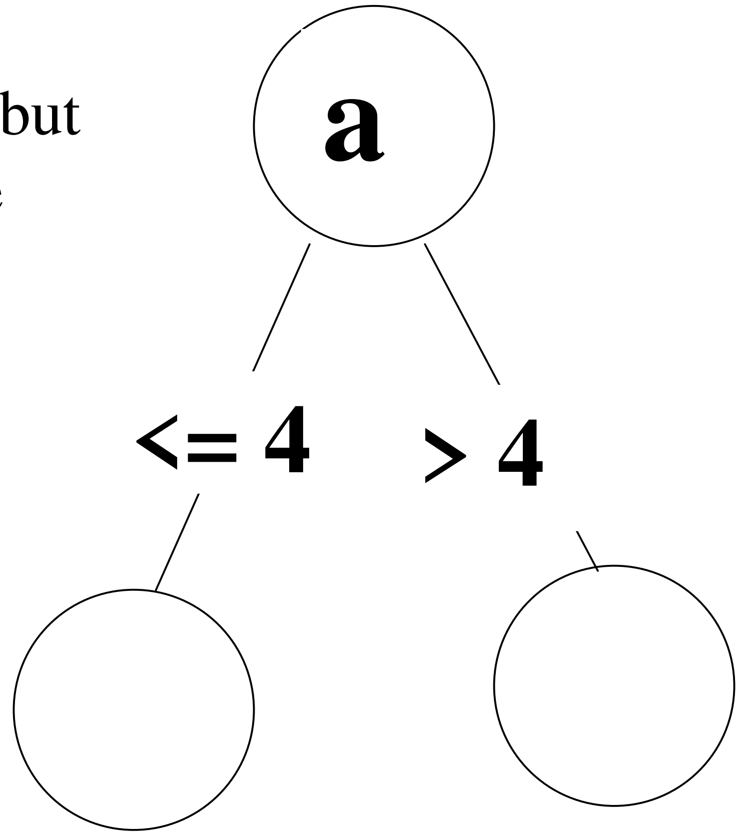
Nr.	Länge Kelchblatt	Breite Kelchblatt	Länge Blütenblatt	Breite Blütenblatt	Gattung
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	"
3	4.7	3.2	1.3	0.2	"
...					
51	7.0	3.2	4.7	1.4	versicolor
52	6.4	3.2	4.5	1.5	"
53	6.9	3.1	4.9	1.5	"
...					
101	6.3	3.3	6.0	2.5	virginica
102	5.8	2.7	5.1	1.9	"
103	7.1	3.0	5.9	2.1	"

(ftp://ftp.ics.uci.edu/pub/machine-learning-databases/iris)

Entscheidungsbaum über kontinuierlichen Daten

naives Vorgehen mit ID3 / C4.5:

- wähle in jedem Knoten ein geeignetes Attribut
- teste dieses gegen eine geeignete Konstante
- bilde zwei Söhne (<=, >)



Die geeignetste Kombination (Attribut, Konstante) ist jene mit maximalem *Gain Ratio*.

$$\text{Gain Ratio} := \frac{\text{Gain}}{\text{Split Info}}$$

Informationsgewinn einer Aufpaltung (Gain)
im Verhältnis zur
Aussagekraft der dadurch entstehenden Partitionsgrößen (Split Info).

[Quinlan, J.R. (1993). C4.5: Programs for Machine Learning. Morgan Kaufmann]

Auffrischung Entropie

$$I = -\sum_{m=1}^{n} p_m \log_2 p_m$$

bei n Klassen und Wahrscheinlichkeit p_m des m-ten Wertes

Bsp. für n=2

a	1/2	1/3	1/4	1/8	1/16	1/32
I(1-a, a)	1	0.9182	0.8112	0.5436	0.3373	0.2006

$$\mathrm{Gain}(T, A, a) \coloneqq$$

$$I(T, \mathrm{Klassen}) - \frac{|T_{<=}|}{|T|} I(T_{<=}, \mathrm{Klassen}) - \frac{|T_{>}|}{|T|} I(T_{>}, \mathrm{Klassen})$$

$$\mathrm{Split\ Info}(T, A, a) \coloneqq I(T, \mathrm{Partitionen\ nach}\ (A, a))$$

dabei:
T: Trainingsmenge
A: Attribut
a: ein Wert des Attributs A
$T_{<=}$: Teilmenge von T mit A <= a für alle Datentupel

Beispiel aus dem Iris-Datensatz:

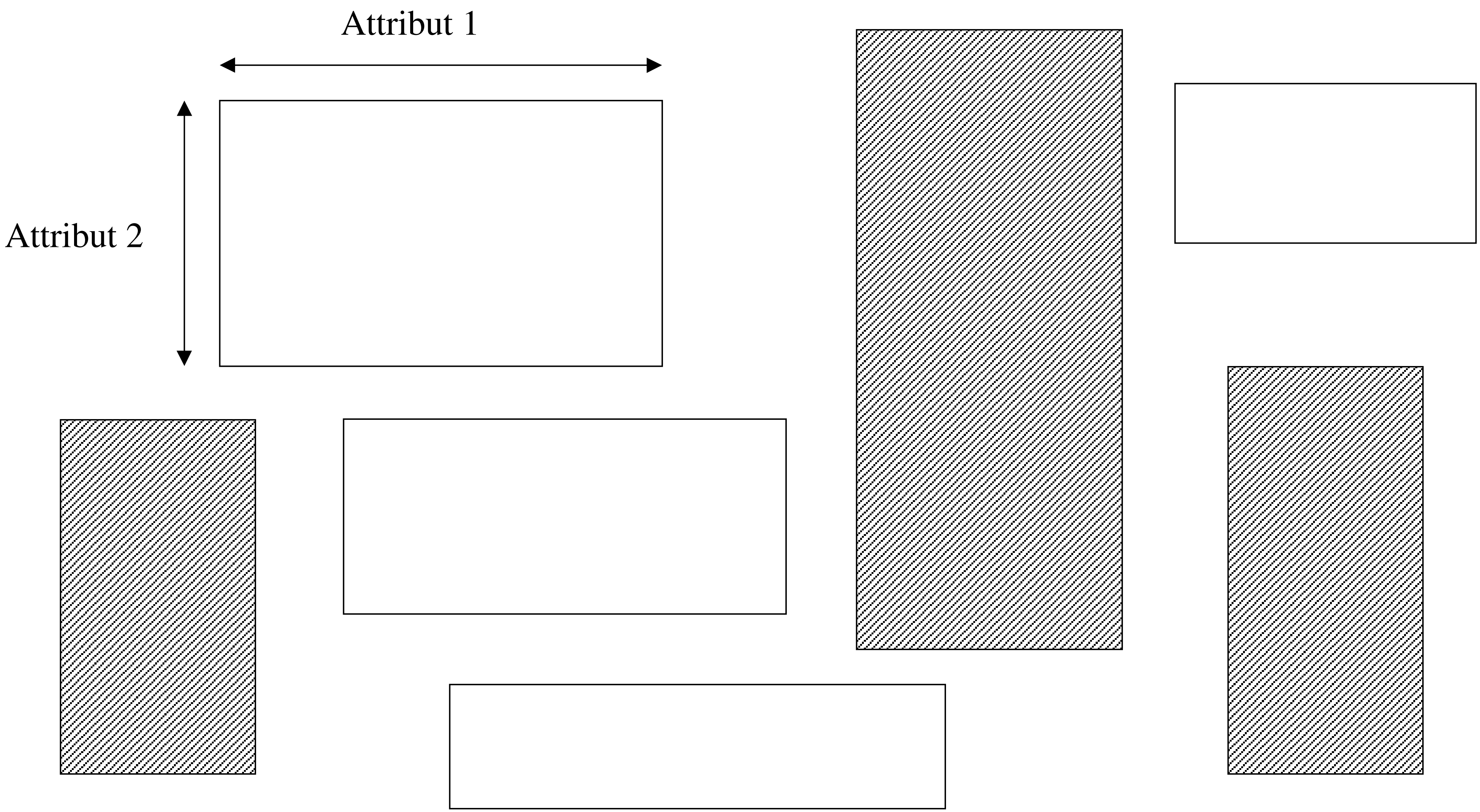
Bestimme das Gain Ratio für "Länge Kelchblatt" <= / > 5.1:

- I (T, Klassen) = – 3/9 * $\log_2$ (3/9) – 3/9 * $\log_2$ (3/9) – 3/9 * $\log_2$ (3/9) = 1.584963
- 6/9 * I ($T_{LKB <= 5.1}$, Klassen) = 6/9 * (– 0 – 3/6 * $\log_2$ (3/6) – 3/6 * $\log_2$ (3/6)) = 0.666667
- 3/9 * I ($T_{LKB > 5.1}$, Klassen) = 3/9 * (– 3/9 * $\log_2$ (3/3) – 0 – 0) = 0
- Gain (T, LKB, 5.1) = 1.584963 – 0.6667 – 0 = 0.918296

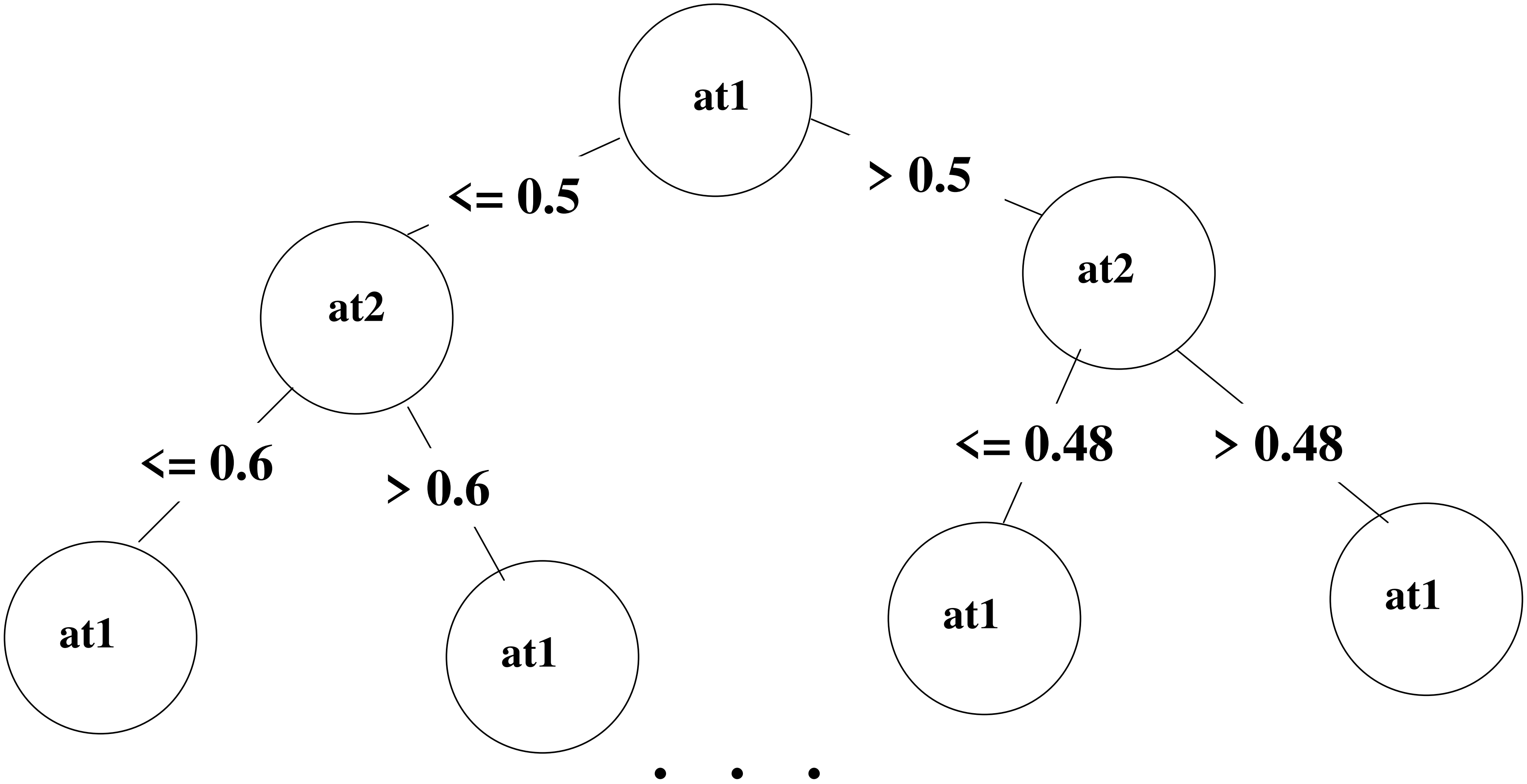
- Split Info (T, LKB, 5.1) = – 3/9 * $\log_2$ (3/9) – 6/9 * $\log_2$ (6/9) = 0.918296

- Gain Ratio (T, LKB, 5.1) = 0.918296 / 0.918296 = 1   (das ist maximal)

Zwei Klassen von Rechtecken



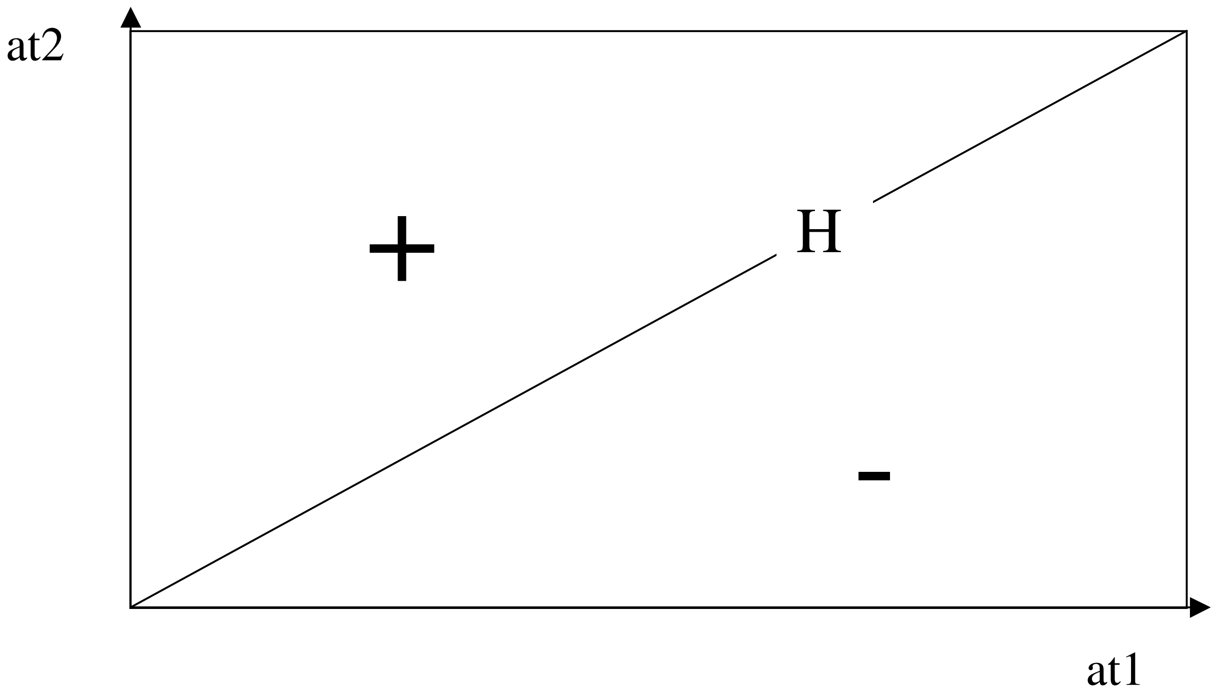
Ein Entscheidungsbaum zum Rechteck-Problem



Problem

Der Baum wird sehr groß, weil dieses Problem in der Sprache der C4.5-Bäume schlecht beschreibbar ist.

Das gilt leider für alle schiefen (engl. „oblique“) Probleme.



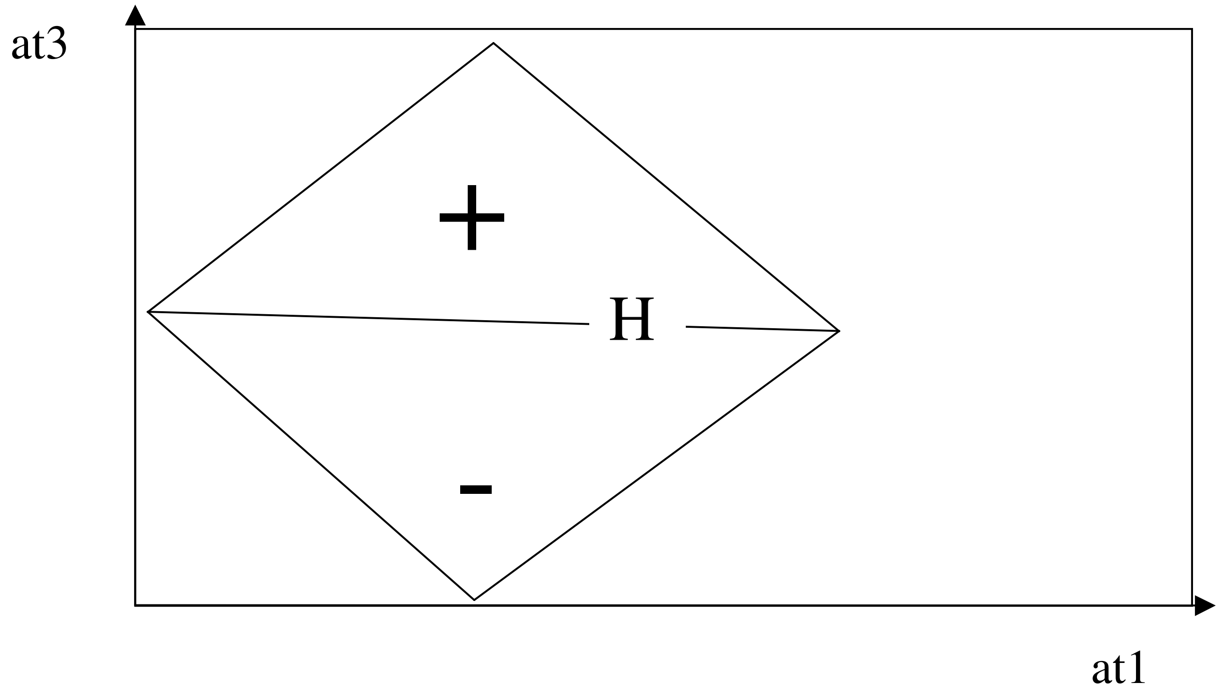
Constructive Induction:

“the application
of a set of constructive operators
to a set of existing features
resulting in
the construction
of one or more new features
intended for use in describing the target concept.”

(Definition aus [Matheus and Rendell, 1989], zitiert nach [Brazdil and Gama, 1998])

Ein neues Attribut eröffnet eine neue Sicht auf die Beispiele im Merkmalsraum

wähle at3 := at1 – at2



Beobachtungen im erweiterten Merkmalsraum:

- Das die Klassen trennende Lernergebnis H wäre eine Hyperebene.
- Die senkrecht auf diese Hyperebene stehende Gerade bezeichnet die Achse des neuen Attributs.

Aufgaben von Systemen zur Konstruktiven Induktion:

- Feststellen des Bedarfs neuer Attribute
- Konstruktion der neuen Attribute
- Finden der Hypothese (hier: Hyperebene)

Der Algorithmus *C4.5Oblique*:

- Feststellen des Bedarfs neuer Attribute,
  Konstruktion der neuen Attribute:
  - erzeuge vorab lineare Diskriminanzfunktionen

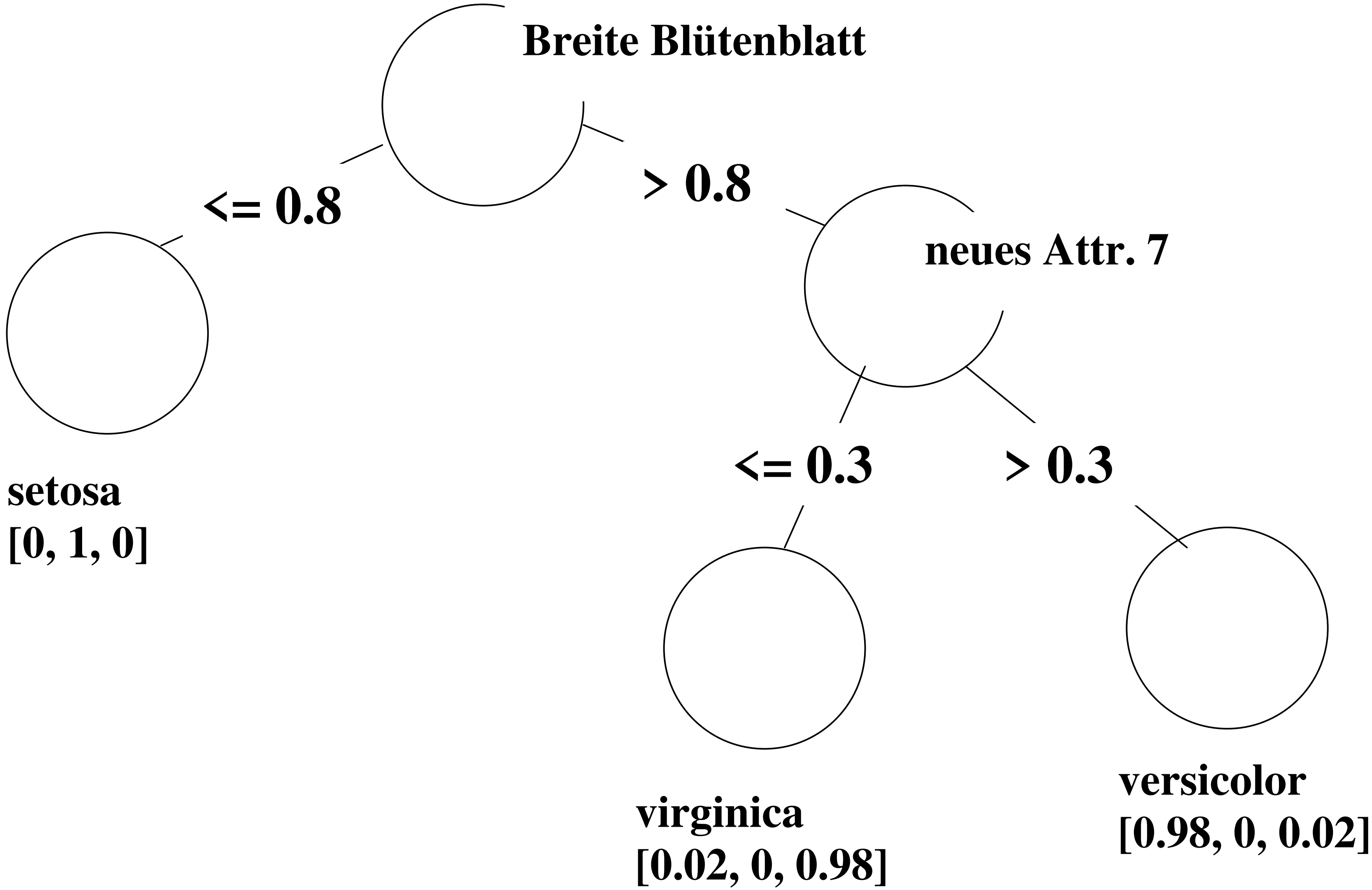
- Finden der Hypothese:

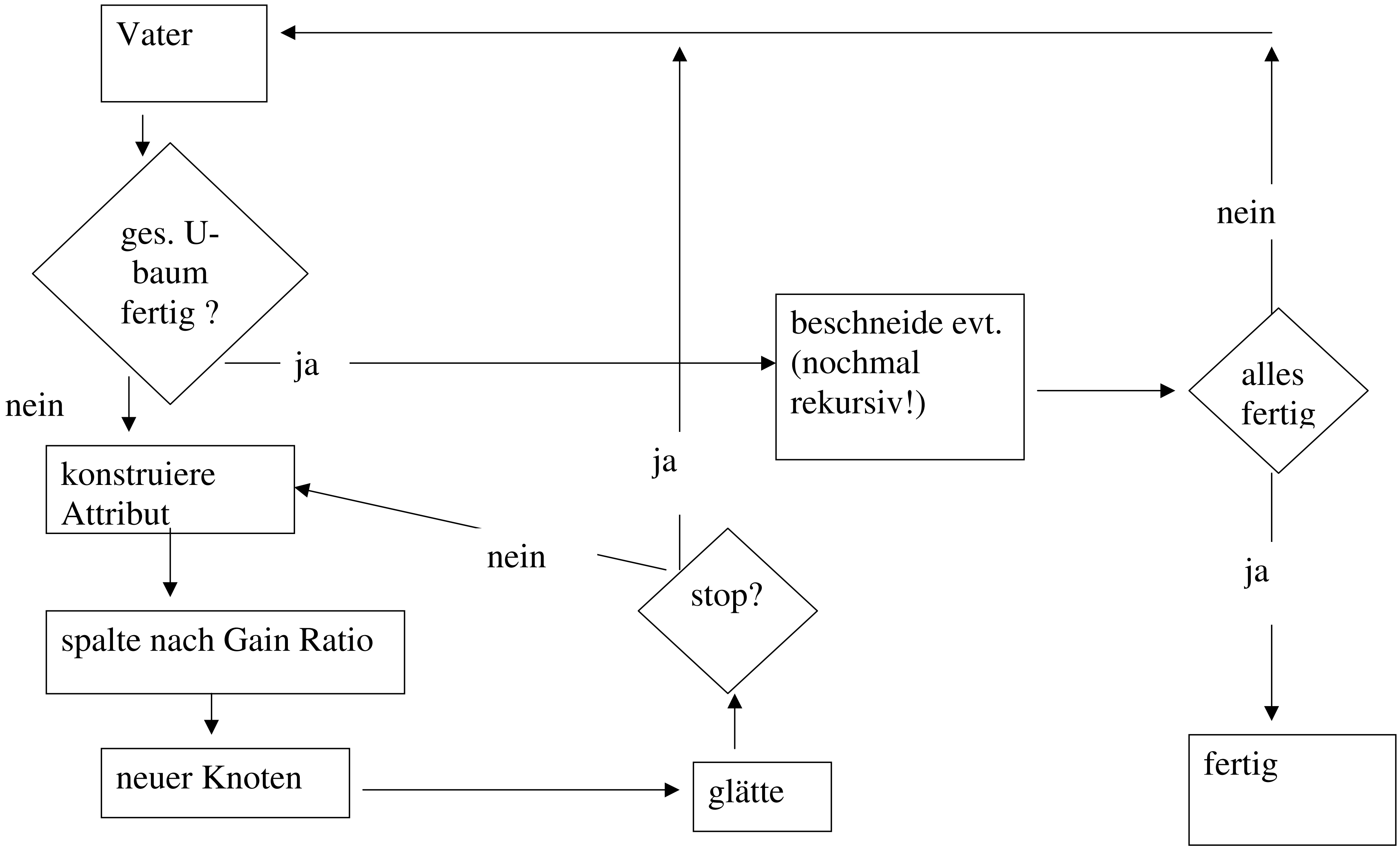
  wende C4.5 an (maximiere Gain Ratio)

Der Algorithmus *Ltree*:

- Konstruktion *neuer Attribute* während des Baumaufbaus
- neue *neue Attribute* können aus alten *neuen Attributen* zusammengesetzt sein
- probabilistischer Baum, alle Klassenverteilungen auf dem Pfad werden berücksichtigt
- daher Klassifikation immer möglich, auch bei fehlenden Attributen

Bsp.: Der Ltree-Baum für das Iris-Problem

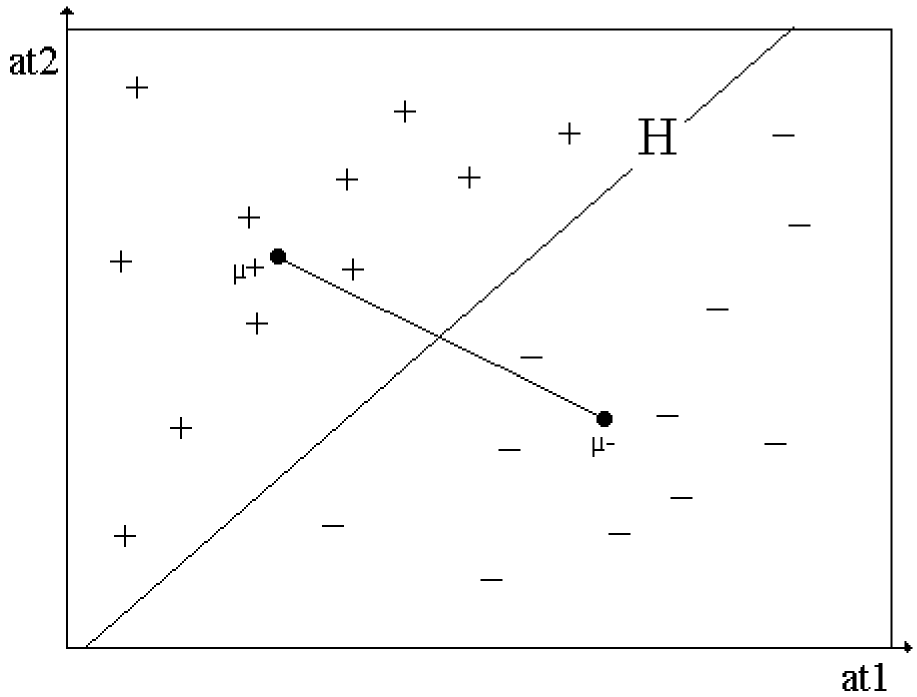



Vater
ges. U-
baum
fertig ?
ja
nein
beschneide evt.
(nochmal
rekursiv!)
alles
fertig
nein
ja
konstruiere
Attribut
ja
nein
stop?
spalte nach Gain Ratio
neuer Knoten
glätte
fertig

Konstruktive Induktion bei *Ltree*

- Feststellung des Bedarfs neuer Attribute:
  - betrachte nur q_{node} Klassen mit mehr Vertretern, als es bis jetzt Attribute gibt
  - stelle fest, daß es q_{node}-1 neuer Attribute bedarf

- Konstruktion der neuen Attribute,
  Finden der Hypothese:
  - nehme an:
    Attributwerte in allen Klassen normalverteilt,
    gleiche Varianzen in allen Klassen
  - stelle lineare Diskriminanzfunktion auf



Beispiel für das Rechteck-Problem

$$H(x) = \frac{(\mu_+ - \mu_-)}{\sigma^2} x - \frac{(\mu_+ + \mu_-)(\mu_+ - \mu_-)}{2\sigma^2}$$

Lineare Diskriminanzfunktion für den allgemeinen Fall:

$$S = \sum_i \frac{n_i * S_i}{n - q}$$

$$\vec{\alpha}_i = -\frac{1}{2}\vec{\mu}_i^T S^{-1}\vec{\mu}_i \qquad \vec{\beta}_i = S^{-1}\vec{\mu}_i$$

$$H_i = \vec{\alpha}_i + \sum_j \vec{\beta}_{ij} * \vec{x}_j$$

i: Laufvariable über die Klassen 1 bis q_{node}-1
j: Laufvariable über die Attribute
S_i: Kovarianzmatrix für die i-te Klasse
H_i: Hyperebene, die die i-te Klasse von einer der anderen Klassen trennt

Aufspaltung

- verwende Gain Ratio (genau wie bei C4.5), um festzustellen, welches Attribut zu verwenden ist.
- das Finden des Schwellwerts ist nicht mehr erforderlich

Abbruchkriterium

- mehr als 95% (default) der Beispiele an diesem Knoten aus derselben Klasse

Nachglätten der Klassenverteilung

- Motivation:
  Divide-and-conquer, daher Gefahr von hoher Varianz und Overfitting
- rekursiv

Wahrsch. am Vorgängerknoten, daß Beispiel aus Klasse H in diesen Konten wandert

$$P(H \mid e_n, e) = P(H \mid e_n) \frac{P(e \mid e_n, H)}{P(e \mid e_n)}$$

dieser Knoten

Vorgänger

Wahrsch. am Vorgängerknoten, daß Beispiel in diesen Knoten wandert

Stutzen des Baums

- *static error*:
  Anzahl der falsch klassifizierten Beispiele nach der Klassenverteilung an diesem Knoten
- *backed-up error*:
  Summe der static errors aller Blätter im Unterbaum

Schneide alles darunterliegende ab, wenn
backed-up error > static error

Hier zeigt sich der Vorteil des Nachglättens:
es kann mehr zurückgeschnitten werden

Algorithmen im Vergleich

Datensatz	**Ltree**	LMDT	OC1	C4.5	**C4.5Oblique**
Australian	**13.9±4**	35.9±12	14.8±6	15.3±6	14.9±5
	14.2	72.6	13.2	35.5	21.1
Balance	10.4±4	12.0±4	10.4±3	34.6±4	**9.0±4**
	6.9	59.6	20.8	43.5	7.6
...					
Waveform	17.8±2	20.3±3	22.1±1	23.9±2	**16.9±2**
	167.2	115.8	**61.8**	309.2	128.0
Wine	**2.8±3**	3.4±4	9.0±6	6.7±8	2.8±5
	5.0	**4.0**	9.2	9.6	5.0
∅	**13.9**	20.2	17.9	17.9	14.6
(ohne Letter)	32	49	**26**	65	35

1. Zeile: Klassifikationsfehler mit Standardfehlerabweichung
2. Zeile: Baumgröße in Knoten

Ergebnisse der *Constructive Induction* – Algorithmen:

- gut bzgl. der Genauigkeit (durch constructive induction)
- gut bzgl. der Baumgröße (durch constructive induction)
- gut bzgl. der Laufzeit (durch divide and conquer)
- Ltree schneidet etwas besser ab als C4.5Oblique

Ausblick

Man kann auch Regressionsprobleme angehen, gemäß der Klassenverteilung können verschiedene Funktionen gewichtet zusammengeführt werden.

auch andere Trennfunktionen als Linearkombinationen möglich:

- quadratische
- logistische
- ...
- Kernfunktionen! (vgl. Support Vector Machine)

mein Vorschlag:
direktes Finden der Hypothese durch *Genetic Programming* in jedem Baumknoten