LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

INSTITUT FÜR STATISTIK

Stephanie Thiemichen

Unverzerrtes rekursives Partitionieren: Ein empirischer Vergleich von CHAID und CTREE

Bachelor Thesis
Betreuer: Prof. Dr. Torsten Hothorn
Institut für Statistik – LMU München

03. Juli 2009

Bachelor-Thesis

Unverzerrtes rekursives Partitionieren: Ein empirischer Vergleich von CHAID und CTREE

Stephanie Thiemichen

03. Juli 2009

Betreuer: Prof. Dr. Torsten Hothorn
Ludwig-Maximilians-Universität München – Institut für Statistik

Abstract

Regressions- und Klassifikationsbäume sind eine sehr weit verbreitete Methode im Bereich der angewandten Datenanalyse. In der Forschung sind schon seit Längerem zwei wesentliche Probleme vieler dieser Verfahren bekannt: Überanpassung und eine Verzerrung der Variablenselektion hin zu Kovariablen mit vielen Splitmöglichkeiten. In der vorliegenden Arbeit werden zwei Ansätze miteinander verglichen, die es ermöglichen, Klassifikationsbäume zu erstellen und dabei die genannten Probleme zu lösen: CHAID und CTREE. Bei beiden Verfahren wird die Variablenselektion vom eigentlichen Splittingvorgang getrennt und die Stopp-Kriterien beider Algorithmen basieren auf formalen statistischen Hypothesentests. Ein Vergleich der Vorhersagegüte der beiden Ansätze erfolgt mittels verschiedener Benchmark-Experimente, wobei im Ergebnis CTREE meist einen geringeren Vorhersagefehler liefert als CHAID. Anhand zweier Simulationen wird gezeigt, dass die Variablenselektion bei CHAID im Gegensatz zu CTREE keineswegs unverzerrt erfolgt und die Ursache für die Verzerrung in der verwendeten Adjustierung der berechneten p-Werte zu suchen ist.

Keywords: Conditional inference trees; CHAID; Klassifikationsbäume; Rekursives Partitionieren; Variablenselektion; Vorhersagegüte

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Die so genannten Klassifikationsbäume entstammen ursprünglich dem Bereich der Diskriminanzanalyse. Die grundsätzliche Problemstellung ist dabei die selbe wie beispielsweise bei der klassischen linearen Diskriminanzanalyse: Eine Grundgesamtheit besteht aus mehreren Teilgesamtheiten (auch als Gruppen oder Klassen bezeichnet), so dass jedes Element (Objekt) genau einer dieser Teilgesamtheiten angehört. Ist die Klassenzugehörigkeit eines Objekts unbekannt, aber wurde an ihm ein Merkmalsvektor $\mathbf{X}$ beobachtet, soll anhand dieser Beobachtung dem Objekt die Klasse zugeordnet werden, aus der es stammt.[1] Neben der Erklärung der den Daten zugrunde liegenden Struktur, ist das hauptsächliche Ziel der Verfahren, die Anpassung eines Modells, das es ermöglicht, Vorhersagen zu treffen, also die Klassifikation neuer Beobachtungen.

Die Idee des CART-Ansatzes (Classification and Regression Trees) ist eine sukzessive Partitionierung des Merkmalsraums von $\mathbf{X}$. Dabei sollen die entstehenden Teilmengen in sich möglichst homogen, untereinander aber möglichst heterogen sein bezüglich des abhängigen metrischen Merkmals (Regressionsbäume) bzw. bezüglich der zugrunde liegenden Klasse (Klassifikationsbäume).
Die Ursprünge dieses Ansatzes liegen im AID-Algorithmus (Automatic Interaction Detection), welcher von Morgan und Sonquist (1963) beschrieben wurde. Neben der Methode bezeichnet CART auch noch ein Programm, das zusammen mit dem Buch „Classification & Regression Trees“ von Breiman

[1] Vergleiche hierzu Kapitel 8 bei Fahrmeir et al. (1996)

et al. (1984) entwickelt wurde. CART ist eine der am weitesten verbreiten Implementierung eines einfachen zwei-stufigen Algorithmus. Dabei werden zunächst die Beobachtungen rekursiv mittels binärer Splits partitioniert und anschließend wird in jeder Zelle ein konstantes Modell angepasst. Gesplittet wird dabei jeweils nach derjenigen Kovariable, die bezogen auf alle möglichen Splits den besten Split liefert. Als Informationsmaß dient dabei die Reinheit der Endknoten. Das bedeutet, je homogener die Beobachtungen in einem Knoten sind, umso besser.
Der gerade beschriebene Ansatz weist zwei wesentliche Probleme auf: Zum einen das Problem der so genannten Überanpassung (Overfitting) und zum anderen eine Verzerrung der Selektion hin zu Kovariablen mit vielen möglichen Splits.
Bei ersterem ist es möglich, dass so lange gesplittet wird, bis jeder Endknoten nur noch genau eine Beobachtung enthält. Dem Problem der Überanpassung kann durch so genannte Beschneidungsverfahren (Pruning) begegnet werden. Dabei wird zunächst der volle Baum berechnet und anschließend wird dieser dann meist mittels einer Form der Kreuz-Validierung verkleinert. Aufgrund des zweiten Problems der Selektionsverzerrung sind die Ergebnisse aber noch immer schlecht bzw. nur schwer interpretierbar.

Zwei Verfahren, die versuchen die gerade beschriebenen Probleme zu lösen, werden in dieser Arbeit miteinander verglichen: CHAID (Hothorn et al., 2006) und CTREE (Kass, 1980). In Kapitel 2 dieser Arbeit werden die beiden Verfahren im Einzelnen vorgestellt, bevor sie in Kapitel 3 mittels einer Simulation speziell im Hinblick auf Unverzerrtheit miteinander verglichen werden. In Kapitel 4 erfolgt ein Vergleich der Vorhersagegüte anhand realer Daten. Kapitel 5 beschäftigt sich mit den computationalen Details. Die Ergebnisse werden anschließend in Kapitel 6 diskutiert und Kapitel 7 enthält eine Zusammenfassung. Die wichtigsten Teile des R-Codes, welcher in dieser Arbeit verwendet wurde, sind im Anhang zu finden. Des Weiteren liegt eine CD-ROM bei, welche die entsprechenden R-Dateien enthält.

Kapitel 2

Rekursives Partitionieren

Zunächst soll kurz das allgemeine Verfahren zum Aufbau von Klassifikationsbäumen beschrieben werden, da diese im Fokus der vorliegenden Arbeit stehen. Die Beschreibung orientiert sich im Wesentlichen an Fahrmeir et al. (1996).
Einen Klassifikationsbaum erhält man bei m Merkmalen durch schrittweises Splitten des Merkmalsträgers $S \subset \mathbb{R}^m$. Ausgehend von $A_1 = S$ wird eine Kovariable $X_j,\ j = 1, \ldots, m$ gesucht, die es erlaubt, Gruppen bezüglich des Response Y möglichst gut voneinander zu trennen. Damit erhält man im Falle eines binären Splits eine erste Verzweigung in die Untermengen A_2 und A_3, wobei $A_2 \cap A_3 = \emptyset$ und $A_2 \cup A_3 = A_1$. Im Weiteren wird nun eine der beiden Untermengen, zum Beispiel A_2, wieder anhand einer Kovariable X_j weiter aufgeteilt in A_4 und A_5. Es wird dabei diejenige Kovariable X_j ausgewählt, die die meiste zusätzliche Information zur Trennung der Gruppen liefert, ausgehend von der Untermenge A_2. Dabei gilt wieder $A_4 \cap A_5 = \emptyset$ und $A_4 \cup A_5 = A_2$. Durch fortlaufende Anwendung dieses Prinzips der binären Rekursion erhält man einen Baum. Die Verzweigungsstellen des Baumes, auch Knoten genannt, entsprechen jeweils einer Untermenge A_n von A_1.

2.1 CHAID

χ^2 Automated Interaction Detection (kurz: CHAID) wurde von Kass (1980) vorgestellt und ist eine Erweiterung des AID-Verfahrens. CHAID ist ein Ansatz zur Lösung der bereits genannten Probleme der Überanpassung

und der Variablenselektion basierend auf statistischen Signifikanztests für Kontingenztafeln. Der Prozess der Variablenselektion wird hierbei vom eigentlichen Splitting-Vorgang getrennt. Der Zusammenhang zwischen dem nominalen Response Y und den Kovariablen $X_1, \ldots, X_m$ wird durch χ^2-Tests gemessen und diejenige Kovariable mit dem höchsten Wert der Teststatistik wird zum Splitten ausgewählt. Wird bei den χ^2-Tests ein bestimmtes Signifikanzniveau (in der Regel $\alpha = 0.05$) nicht mehr erreicht, endet der Algorithmus. CHAID ermöglicht somit die Implementierung eines Stopp-Kriteriums für den Algorithmus basierend auf formalen Hypothesentests.

2.1.1 Algorithmus

Die abhängige Variable Y sei kategorial und bestehe aus $d \geq 2$ Kategorien.

Schritt 1:
Für jeden Prädiktor $X_j,\ j = 1, \ldots, m$ wird die Kreuztabelle der Kategorien des Prädiktors und der Kategorien der abhängigen Variable Y berechnet und es werden die *Schritte 2* und *3* ausgeführt. Ist dies für jeden Prädiktor geschehen, erfolgt der Übergang zu *Schritt 4*.

Schritt 2:
Es werden die beiden Kategorien des Prädiktors X_j gesucht, deren $2 \times d$–Kontingenztafel den höchsten p-Wert aufweist, bezogen auf die anhängige Variable Y. Hierbei sind je nach Skalenniveau der Prädiktorvariable X_j nur bestimmte Kombinationen von Kategorien erlaubt, zum Beispiel können bei einer ordinalen Variable nur benachbarte Kategorien zusammengelegt werden. Bei einer nominalen Variable können es zwei beliebige Kategorien sein. Unterschreitet der ermittelte p-Wert nicht den kritischen Wert (zum Beispiel $\alpha = 0.05$) werden diese beiden Kategorien zusammengelegt und als eine neue Kategorie betrachtet. Der *Schritt 2* wird dann wiederholt.

Schritt 3:
Für jede zusammengelegte Kategorie, die aus drei oder mehr der Originalkategorien besteht, wird der signifikanteste binäre Split gesucht, welcher, wie im Schritt vorher, wiederum vom Typ der Prädiktorvariable X_j

abhängt. Erreicht die Signifikanz einen bestimmten kritischen Wert, wird der Split angewendet und man kehrt zu *Schritt 2* zurück.

Schritt 4:
Es wird die Signifikanz für jeden optimal zusammengelegten Prädiktor X_j, $j = 1, \ldots, m$ berechnet und der signifikanteste ausgewählt. Erreicht die Signifikanz ein bestimmten kritischen Wert, werden die Daten entsprechend der (zusammengelegten) Kategorien des ausgewählten Prädiktors aufgeteilt.

Schritt 5:
Für jeden Teilbereich der Daten, der noch nicht analysiert wurde, kehrt man zu *Schritt 1* zurück. Dabei können Bereiche mit nur sehr wenigen Beobachtungen ausgeschlossen werden.

2.1.2 Signifikanz der Prädiktoren

Im Schritt 4 des Algorithmus wird ein Signifikanztest für die reduzierte Kontingenztafel für jede der Prädiktorvariablen $X_j,\ j = 1, \ldots, m$ durchgeführt. Kass (1980) schlägt dazu die im Folgenden beschriebene Herangehensweise vor.
Falls keine Reduktion gegenüber der ursprünglichen Kontingenztafel in den vorhergehenden Schritten erfolgt ist, das bedeutet, falls keine Kategorien der Prädiktorvariable X_j zusammengelegt wurden, wird ein χ^2-Test in Abhängigkeit der Anzahl der Kategorien des entsprechenden Prädiktors angewendet.
Im Falle einer Reduktion der entsprechenden Kontingenztafel, also falls Kategorien der Prädiktorvariable X_j zusammengelegt wurden, wird ebenfalls ein χ^2-Test in Abhängigkeit der Anzahl der Kategorien des jeweiligen Prädiktors durchgeführt. Anschließend erfolgt hier aber zusätzlich eine Bonferroni-Adjustierung des berechneten p-Werts, indem dieser mit einem Korrekturfaktor multipliziert wird.

Zur Ermittlung des benötigten Korrekturfaktors, wird die Anzahl der Möglichkeiten bestimmt, einen Prädiktor eines bestimmten Typs mit c Kategorien auf einen Prädiktor mit r Kategorien zu reduzieren:

(1) *Monotoner Prädiktor*
Ein monotoner Prädiktor ist ein geordneter Faktor, das heißt, die Kategorien sind ordinal und es dürfen nur aufeinander folgende Kategorien zusammengelegt werden. Der Bonferroni-Korrekturfaktor entspricht dann dem folgenden Binomial-Koeffizienten:

$$B_{\text{monoton}} = \binom{c-1}{r-1}$$

(2) *Freier Prädiktor*
Ein freier Prädiktor ist ein ungeordneter Faktor, das heißt, die Kategorien sind nominal und es ist möglich, beliebige Kategorien zusammenzulegen. In diesem Fall ergibt sich der Bonforroni-Korrekturfaktor über die folgende Formel:

$$B_{\text{frei}} = \sum_{i=0}^{r-1} (-1)^i \frac{(r-i)^c}{i!(r-i)!}$$

(3) *Floating Predictor*
Der dritte Typ, der so genannte Floating Predictor, entspricht im Wesentlichen einem geordneten Faktor, allerdings mit einer zusätzlichen so genannten gleitenden Kategorie, welche entweder nicht zum Rest gehört oder auf der Ordinalskala nicht eingeordnet werden kann. Dieser Fall soll an dieser Stelle nicht weiter betrachtet werden, da das Konzept im R-Paket `CHAID` (The FoRt Student Project Team, 2009), das in Kapitel 5 noch genauer vorgestellt wird, zum jetzigen Zeitpunkt noch nicht implementiert wurde.

Für weitere Details zu CHAID sei auf Kass (1980) verwiesen.

2.2 CTREE

Das Konzept der Conditional Inference Trees (kurz: CTREE) wurde von Hothorn et al. (2006) entwickelt. Baum-basierte Regressionsmodelle werden dabei mit einer wohldefinierten Theorie für bedingte Inferenzprozeduren verknüpft. Die implementierten Stopp-Kriterien basieren auf multiplen Testverfahren. Die vorgeschlagene Methode ist dabei auf jegliche Art von Regressionsproblemen anwendbar, das bedeutet, die Responsevariablen können neben nominal und ordinal, beispielsweise auch numerisch, zensiert oder

multivariat sein und auch die Kovariablen können ein beliebiges Skalenniveau aufweisen. Ziel ist es, einen einheitlichen Rahmen für rekursives Partitionieren zur Verfügung zu stellen.
Das Verfahren ist somit wesentlich universeller einsetzbar als CHAID, bei dem sowohl der Response als auch die Kovariablen kategorial sein müssen und daher lediglich Klassifikationsbäume erstellt werden können.

2.2.1 Algorithmus

Jeder Knoten eines Baumes kann durch einen Gewichtsvektor $\mathbf{w}$ mit nichtnegativen ganzzahligen Einträgen repräsentiert werden. Ist die entsprechende Beobachtung Element des Knotens, ist der Eintrag an der zugehörigen Stelle des Gewichtsvektors $\mathbf{w}$ von Null verschieden und ansonsten Null. Der grundlegende Algorithmus von CTREE, welcher rekursives binäres Splitten umsetzt, gestaltet sich dabei folgendermaßen. Wie auch bei CHAID wird hierbei der Prozess der Variablenselektion vom Splitting-Vorgang selbst getrennt.

Schritt 1:
Für den Gewichtsvektor $\mathbf{w}$ wird die globale Nullhypothese der Unabhängigkeit zwischen irgendeiner der m Kovariablen $X_j, j = 1, \ldots, m$ und der Responsevariable Y getestet. Kann diese nicht verworfen werden, stoppt der Algorithmus. Andernfalls wird die Kovariable X_{j^*} mit der stärksten Assoziation zu Y ausgewählt.

Schritt 2:
Es wird eine Menge $A^* \subset X_{j^*}$ ausgewählt, um X_{j^*} in zwei disjunkte Mengen zu zerlegen, A^* und $X_{j^*} \setminus A^*$. Die Gewichtsvektoren $\mathbf{w}_{\text{links}}$ und $\mathbf{w}_{\text{rechts}}$ bestimmen die beiden Untergruppen mit $\mathbf{w}_{\text{links},i} = w_i I(X_{j^*i} \in A^*)$ und $\mathbf{w}_{\text{rechts},i} = w_i I(X_{j^*i} \notin A^*)$ für alle $i = 1, \ldots, n$. $I(\cdot)$ ist hierbei die Indikatorfunktion.

Schritt 3:
Die *Schritte 1* und *2* werden mit den veränderten Gewichtsvektoren $\mathbf{w}_{\text{links}}$ bzw. $\mathbf{w}_{\text{rechts}}$ rekursiv wiederholt.

Der Algorithmus stoppt, wenn die globale Nullhypothese der Unabhängigkeit zwischen irgendeiner der m Kovariablen $X_j, j = 1, \ldots, m$ und der Responsevariable Y zu einem vorher definierten Signifikanzniveau α nicht mehr verworfen werden kann. Das hier verwendete Stopp-Kriterium ist somit intuitiv verständlich und statistisch motiviert.

2.2.2 Stopp- und Selektions-Kriterum

Der Test der globalen Nullhypothese in Schritt 1 des Algorithmus erfolgt über Mittelwerte der bedingten Verteilung linearer Statistiken im Rahmen von Permutationstests, wie sie von Strasser und Weber (1999) vorgeschlagen wurden. Der beste Split einer ausgewählten Kovariable wird ebenfalls im Rahmen solcher Tests basierend auf standardisierten linearen Statistiken bestimmt.
In jedem Knoten, bestimmt durch den Gewichtsvektor $\mathbf{w}$, kann die globale Hypothese der Unabhängigkeit über m partielle Hypothesen

$$H_0^j : \; F(Y|X_j) = F(Y)$$

formuliert werden mit der globalen Nullhypothese

$$H_0 = \bigcap_{j=1}^{m} H_0^j.$$

$F(\cdot)$ bezeichnet dabei die Verteilungsfunktion. Kann H_0 zu einem vorher festgelegten Level α nicht verworfen werden, endet die Rekursion. Kann die globale Nullhypothese hingegen abgelehnt werden, wird der Zusammenhang zwischen Y und jeder der Kovariablen $X_j, \; j = 1, \ldots, m$ über Teststatistiken bzw. p-Werte berechnet, welche die Abweichung von der partiellen Hypothese H_0^j erfassen.
Die gemeinsame Verteilung von Y und X_j ist in der Realität meistens unbekannt. Allerdings gilt unter der Nullhypothese der Unabhängigkeit zwischen Y und X_j, wenn die Kovariable fixiert ist, dass Permutationen des Responses die gemeinsame Verteilung nicht beeinflussen. Dieses Prinzip führt letztendlich zu den sogenannten Permutationstests (Strasser und Weber, 1999).

Um eine Vergleichbarkeit zwischen Kovariablen mit verschiedenen Skalenniveaus zu ermöglichen, werden nicht die zugehörigen Teststatistiken der Permutationstests direkt miteinander verglichen, sondern es werden die entsprechenden p-Werte verwendet. In Schritt 1 des Algorithmus wird diejenige Kovariable X_{j^*} ausgewählt, welche den kleinsten p-Wert aufweist, also

$$j^* = \underset{j=1,\ldots,m}{\operatorname{argmin}} \; P_j$$

mit P_j als Bezeichnung des p-Werts des bedingten Tests für H_0^j.
Es ist ausreichend, die einzelnen Hypothesen H_0^j zu testen, um eine unverzerrte Variablenselektion zu erreichen. Es wäre auch möglich, einen Test für die globale Hypothese H_0 zu konstruieren. Dies kann aber bei fehlenden Werten problematisch werden. Daher werden in CTREE multiple Testprozeduren basierend auf den p-Werten $P_1, \ldots, P_m$ verwendet. Das ist im einfachsten Fall eine Bonferroni-Adjustierung. Die globale Nullhypothese H_0 wird abgelehnt, wenn das Minimum der adjustierten p-Werte unterhalb eines vorher definierten Werts α liegt. Ist das nicht der Fall und H_0 kann nicht verworfen werden, stoppt CTREE an dieser Stelle. α kann in diesem Sinne als Parameter zur Bestimmung der Baumgröße betrachtet werden.

In Schritt 2 wird die Güte eines Splits ebenfalls über lineare Teststatistiken für alle möglichen Teilmengen A von X_{j^*} ermittelt. Die Teststatistiken erfassen dabei den Unterschied zwischen den beiden Stichproben

$$\{Y_i | w_i > 0 \text{ und } X_{ji} \in A; i =, 1 \ldots, n\} \text{ und}$$
$$\{Y_i | w_i > 0 \text{ und } X_{ji} \notin A; i =, 1 \ldots, n\}.$$

Es wird diejenige Teilmenge A^* von X_{j^*} gewählt mit der größten Teststatistik. Zur Verhinderung von pathologischen Splits kann die Anzahl der möglichen Teilmengen A, die evaluiert wird, begrenzt werden, zum Beispiel mittels Restriktionen für den Stichprobenumfang.

Weitere Details zur genauen Berechnung der Teststatistiken und der p-Werte in Schritt 1 und 2 des Algorithmus finden sich bei Hothorn et al. (2006).

2.3 RPART

RPART ist im Wesentlichen eine Umsetzung des bereits in der Einleitung erwähnten CART-Algorithmus in R im Paket `rpart` (Therneau und Atkinson, 1997). Dieser Algorithmus dient in dieser Arbeit als Referenz im Kapitel 4 beim Vergleich der Vorhersagegüte von CHAID und CTREE anhand realer Daten.

RPART gehört zu den sogenannten Exhaustionsmethoden (Exhaustive search procedures). Über allen möglichen binären Splits wird der beste Split gesucht. Gütekriterium ist dabei die Reinheit der Endknoten. Die entsprechende Kovariable, die den besten Split liefert, wird ausgewählt. Das Ganze wird rekursiv wiederholt. Am Ende wird in jeder Zelle der entstandenen Partition ein konstantes Modell angepasst.
Um das Problem der Überanpassung zu lösen, ist in RPART die sogenannte Kosten-Komplexitäts-Beschneidung (Cost-complexity pruning) basierend auf Kreuz-Validierung implementiert. Für die genaue Funktionsweise dieser Beschneidungsprozedur siehe zum Beispiel Fahrmeir et al. (1996).

Kapitel 3

Unverzerrtheit

Ein Algorithmus für rekursives Partitionieren ist unverzerrt, wenn unter der Nullhypothese der Unabhängigkeit zwischen Response Y und den Kovariablen $X_1, \ldots, X_m$ die Wahrscheinlichkeit, die Kovariable X_j auszuwählen, $\frac{1}{m}$ beträgt, für alle $j = 1, \ldots, m$, ungeachtet der Skalenniveaus der Kovariablen oder der Anzahl fehlender Werte.[1]

Im Folgenden soll durch ein einfaches Simulationsexperiment überprüft werden, ob neben CTREE [2] auch CHAID unverzerrt ist.
Die entsprechenden R-Codes für die Simulation sind in Anhang A zu finden.

3.1 Simulationsaufbau

Es wurden sechs kategoriale Kovariablen $X_1, \ldots, X_6$ erzeugt. Sie stellen zufällige Permutationen von Faktoren mit vier, sechs bzw. acht Faktorstufen dar, wobei die ersten drei Faktoren geordnet und die anderen drei ungeordnet sind. Eine zusätzliche Kovariable X_7 ist ebenfalls ein Faktor mit zwei Levels. Die Responsevariable Y ist binomial-verteilt mit Wahrscheinlichkeit p_0 bzw. $(1-p_0)$ in Abhängigkeit der beiden Gruppen, die durch X_7 definiert werden.

$$Y \sim B(1, p) \text{ mit } p = \begin{cases} p_0 & \text{, falls } X_7 = 1 \\ 1 - p_0 & \text{, falls } X_7 = 2 \end{cases}$$

[1] Vergleiche Hothorn et al. (2006).
[2] Zur Unverzerrtheit von CTREE siehe Hothorn et al. (2006).

Für $p_0 = 0.5$ ist der Response unabhängig von allen Kovariablen X_j, $j = 1, \ldots, 7$.
Mittels des eben beschriebenen Modells wurden Lernstichproben vom Umfang $n = 100$ gezogen. Die Wahrscheinlichkeit für die Auswahl von X_j, $j = 1, \ldots, 7$ wurde sowohl für CHAID, als auch für CTREE über die Mittelwerte von 10 000 Simulationsdurchläufen geschätzt. In dieser Simulation wurde kein Stopp-Kriterium angewendet und der erste Split (Root-Split) auf diese Weise erzwungen.

3.2 Simulationsergebnisse

Aus praktischer Sicht sind vor allem zwei Aspekte für den Vergleich der beiden Methoden interessant: zum einen die Wahrscheinlichkeit, überhaupt eine Kovariable zum Splitten auszuwählen für ein $p_0 \geq 0.5$ und somit den Root-Split durchzuführen. Dies bezeichnet man als Power des Verfahrens. Zum anderen die bedingte Wahrscheinlichkeit, den „korrekten“ Split in Kovariable X_7 auszuwählen, gegeben, es wurde eine Kovariable zum Splitten ausgewählt.
Abbildung 3.1 zeigt die geschätzten Wahrscheinlichkeiten in Abhängigkeit von p_0. Für $p_0 = 0.5$, das bedeutet bei Unabhängigkeit zwischen dem Response und den Kovariablen, beträgt die Wahrscheinlichkeit, den Root-Split durchzuführen, für CTREE 0.0425 und für CHAID 0.2532. Das heißt, bei CHAID wird trotz Unabhängigkeit zwischen dem Response und den Kovariablen in einem Viertel der Fälle dennoch eine Kovariable zum Splitten ausgewählt. Bei CTREE ist die Wahrscheinlichkeit einer solchen inkorrekten Entscheidung durch $\alpha = 0.05$ nach oben begrenzt. Der hohe Anteil falscher Splitentscheidungen bei CHAID ist darauf zurückzuführen, dass die berechneten p-Werte für die Auswahl der Kovariable zum Splitten in Schritt 4 des Algorithmus, wie in Abschnitt 2.1.2 beschrieben, lediglich bezüglich der Anzahl der Kategorien des entsprechenden Prädiktors adjustiert werden, nicht aber bezüglich der Anzahl der Kovariablen, die zum Splitten in Frage kommen.
Unter der Alternative der Abhängigkeit zwischen Response und Kovariablen, also für $p_0 \geq 0.5$ liegt die Power von CHAID etwas über der von CTREE.
Die Wahrscheinlickeit, Kovariable X_7 zum Splitten auszuwählen, gegeben,

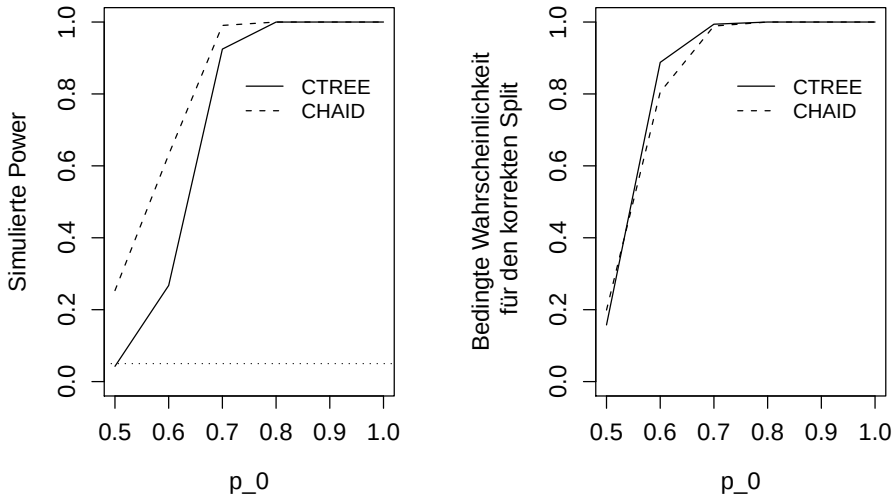

Abbildung 3.1 Simulierte Power, das heißt die Wahrscheinlichkeit eines Root-Splits (links), und simulierte bedingte Wahrscheinlichkeit eines korrekten Splits in der Variable X_7, gegeben, es wurde ein Root-Split vorgenommen (rechts). Die horizontale gestrichelte Linie steht für $\alpha = 0.05$. Die Ergebnisse basieren auf 10 000 Simulationswiederholungen.

es wurde überhaupt eine Kovariable zum Splitten ausgewählt, ist bei beiden Verfahren ähnlich (rechter Teil in Abbildung 3.1).

Tabelle 3.1 zeigt die Ergebnisse des oben beschriebenen Simulationsexperiments für den Fall, dass der Response von den Kovariablen unabhängig ist, also für $p_0 = 0.5$. Die 95%-Konfidenzintervalle (nach Goodman, 1965) der geschätzten Wahrscheinlichkeiten bzw. Anteile enthalten für CTREE jeweils $\frac{1}{7} \approx 0.1429$, wie es für eine unverzerrte Variablenselektion, unabhängig von der Skala der Kovariablen, auch zu erwarten ist.
Für CHAID enthält lediglich das 95%-Konfidenzintervall für X_3 den Wert $\frac{1}{7}$. Am häufigsten wird in der Kovariable X_7 gesplittet, die, wie oben beschrieben, ein binärer Faktor ist. Die Wahrscheinlichkeiten für Splits in den geordneten Faktoren X_1, X_2 und X_3 sind höher als die für Splits in den ungeordneten Faktoren X_4, X_5 und X_6 und jeweils abnehmend mit zunehmender Levelanzahl. Es scheint somit eine deutliche Verzerrung hin zu Variablen

Tabelle 3.1 Simulierte Wahrscheinlichkeiten für die Variablenselektion von sieben voneinander unabhängigen Variablen, wenn der Response unabhängig ist von $X_1, \ldots, X_7$, das heißt $p_0 = 0.5$. Die Resultate basieren auf 10 000 Simulationswiederholungen.

	CHAID		CTREE	
	Schätzer	95% -KI	Schätzer	95%-KI
X_1	0.181	(0.169, 0.194)	0.144	(0.134, 0.156)
X_2	0.155	(0.145, 0.167)	0.145	(0.135, 0.157)
X_3	0.139	(0.128, 0.150)	0.144	(0.134, 0.155)
X_4	0.131	(0.121, 0.141)	0.146	(0.135, 0.157)
X_5	0.086	(0.079, 0.095)	0.140	(0.130, 0.151)
X_6	0.056	(0.050, 0.063)	0.140	(0.130, 0.152)
X_7	0.252	(0.237, 0.267)	0.140	(0.130, 0.151)

mit wenigen Splitmöglichkeiten vorzuliegen. Dieses Ergebnis ist zunächst überraschend, da von Exhaustionsmethoden wie RPART beispielsweise genau die gegenteilige Problematik bekannt ist, nämlich eine Verzerrung der Variablenselektion hin zu Variablen mit möglichst vielen Splitmöglichkeiten.

3.3 Korrekturfaktoren als mögliche Ursache der Verzerrung von CHAID

Wie in Kapitel 2 beschrieben, erfolgt in Schritt 4 des CHAID-Algorithmus eine Bonferroni-Korrektur der berechneten p-Werte. Diese Korrekturfaktoren stellen eine mögliche Ursache des im vorhergehenden Abschnitt beschriebenen Verzerrungsproblems dar. Je mehr Splitmöglichkeiten es für eine Variable gibt, umso größer sind die entsprechenden Korrekturfaktoren aus Abschnitt 2.1.2 für die p-Werte.
Zur Überprüfung dieser Vermutung wurde eine weitere Simulation durchgeführt.

3.3.1 Simulationsaufbau

Es wurden drei untereinander unabhängige kategoriale Kovariablen X_1, X_2 und X_3 erzeugt. X_1 und X_2 stellen zufällige Permutationen von Faktoren

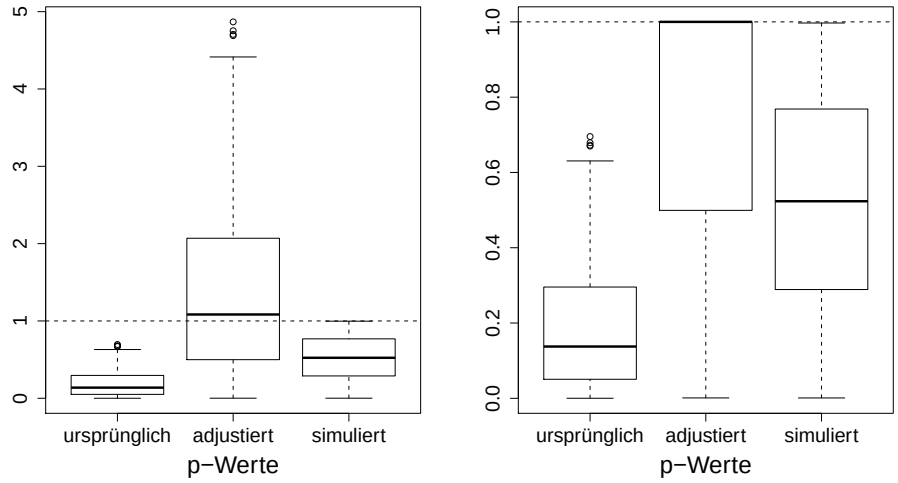

Abbildung 3.2 Ergebnisse der Simulation zur p-Wert-Berechnung für den geordneten Faktor X_1 mit acht Levels. Im rechten Boxplot wurden Werte größer 1 für die adjustierten p-Werte auf 1 gesetzt. Die Ergebnisse basieren auf 500 Simulationsdurchläufen.

mit acht Faktorstufen dar, wobei X_1 geordnet ist und X_2 ungeordnet. X_3 ist binär. Der Response Y ist binomial-verteilt mit $p = 0.5$ und unabhängig von den drei Kovariablen.

Auf diese Weise wurde im ersten Simulationsschritt ein Datensatz mit $n = 500$ Beobachtungen erzeugt. Für jede der drei Kovariablen und den Response wurden die Schritte 1 bis 4 (wie in 2.1.1 beschrieben) von CHAID ausgeführt, allerdings wurden in Schritt 4 sowohl der unkorrigierte p-Wert p_{org} als auch der adjustierte p-Wert p_{adj} ausgegeben.

Im zweiten Simulationsschritt wurde der Response festgehalten und die Kovariablen zufällig permutiert. Wiederum wurden die Schritte 1 bis 4 von CHAID durchgeführt und diesmal nur der unkorrigierte p-Wert p_i berechnet. Dies wurde 1 000-mal für jede der drei Kovariablen wiederholt. Daraus ergibt sich jeweils ein simulierter p-Wert

$$p_{\text{sim}} = \frac{1}{1000} \sum_{i=1}^{1000} I(p_i < p_{\text{org}}).$$

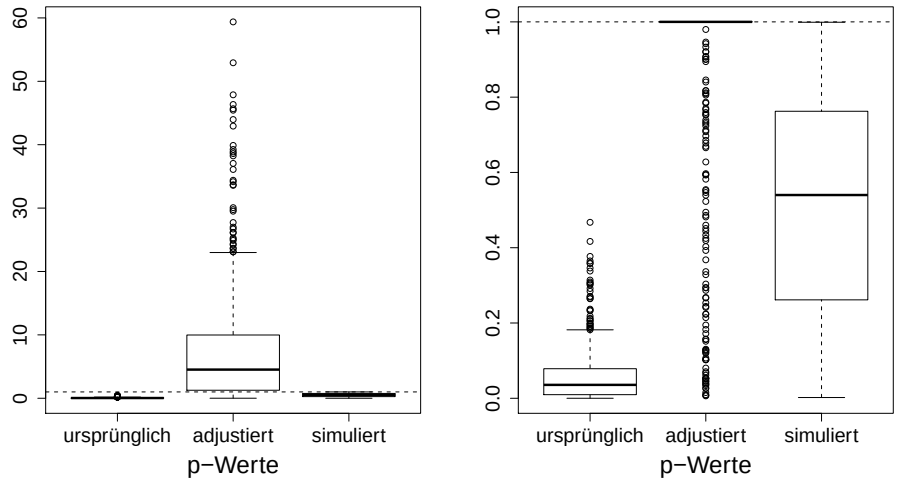

Abbildung 3.3 Ergebnisse der Simulation zur p-Wert-Berechnung für den ungeordneten Faktor X_2 mit acht Levels. Im rechten Boxplot wurden Werte größer 1 für die adjustierten p-Werte auf 1 gesetzt. Die Ergebnisse basieren auf 500 Simulationsdurchläufen.

Diese beiden Simulationsschritte wurden insgesamt 500-mal wiederholt.

3.3.2 Simulationsergebnisse

Die Abbildungen 3.2, 3.3 und 3.4 zeigen die Ergebnisse der eben beschriebenen Simulation. Für die binäre Einflussgröße X_3 entsprechen die adjustierten p-Werte genau den ursprünglichen p-Werten, das heißt $p_{\text{adj}} = p_{\text{org}}$. Das ist dadurch zu erklären, dass bei binären Faktoren keine Kategorien zusammengelegt werden können, da sonst die Kovariable für alle Beobachtungen identisch wäre. Somit müssen die berechneten p-Werte letztlich auch nicht mittels der Korrekturfaktoren, wie in Abschnitt 2.1.2 beschrieben, adjustiert werden. In Abbildung 3.2 und 3.3 ist deutlich zu erkennen, dass die adjustierten p-Werte für X_1 und X_2 in vielen Fällen sehr groß sind und die entsprechenden Variablen selbst für einen erzwungenen Root-Split, bei dem in Schritt 4 $\alpha = 1$ gesetzt wurde, nicht mehr in Betracht kommen. Für den geordneten Faktor X_1 mit acht Levels liegen die korrigierten p-Werte in mehr als der Hälfte der Fälle über 1, für den ungeordneten Faktor X_2

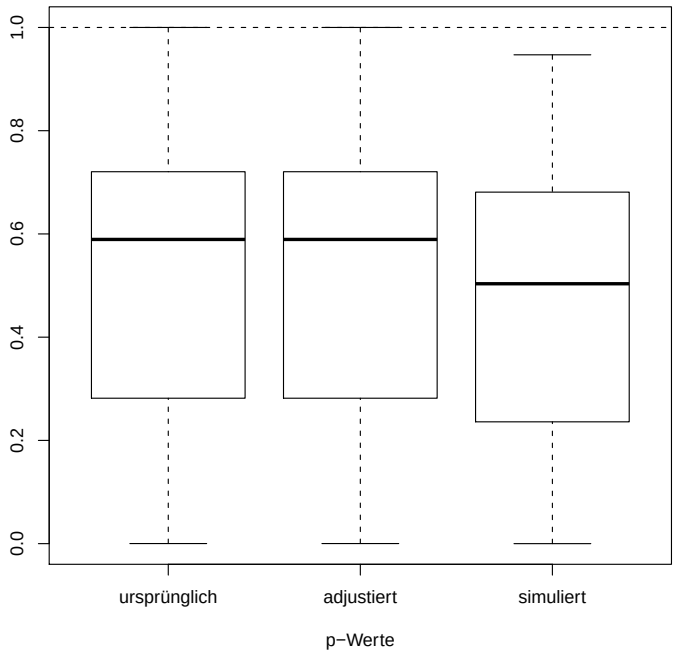

Abbildung 3.4 Ergebnisse der Simulation zur p-Wert-Berechnung für den binären Faktor X_3. Die Ergebnisse basieren auf 500 Simulationsdurchläufen.

ebenfalls mit acht Faktorstufen ist dies sogar in mehr als 75% der Simulationsdurchläufe der Fall.

Das Ergebnis dieser Simulation bestätigt die Vermutung, dass die Verzerrung der Variablenselektion von CHAID hin zu Variablen mit wenigen Splitmöglichkeiten durch die verwendeten Korrekturfaktoren in Schritt 4 des Algorithmus zu Stande kommt.

Kapitel 4

Vorhersagegüte

Die Vorhersagegüte von CHAID und CTREE wurde anhand der Vorhersagefehler für acht Datensätze miteinander verglichen. Zusätzlich wurde jeweils auch der Vorhersagefehler von RPART ermittelt. Hothorn et al. (2006) hatten bereits gezeigt, dass die Vorhersagegüte von CTREE und die Vorhersagegüte von RPART in vielen Fällen äquivalent sind.
Die meisten der verwendeten Datensätze stammen aus dem UCI Repository (Newman et al., 1998) und sind auch im R-Paket `mlbench` (Leisch und Dimitriadou, 2009) zu finden. Darüber hinaus wurde der Datensatz `Glaucoma` zur Berechnung herangezogen. Dieser ist im R-Paket `ipred` (Peters et al., 2002) enthalten. Die hier durchgeführten Benchmark-Experimente orientieren sich im Wesentlichen an den bei Hothorn et al. (2006) betrachteten Benchmark-Problemen.[1]

Der zugehörige R-Code für die Berechnungen ist in Anhang B zu finden.

4.1 Datenaufbereitung

Tabelle 4.1 gibt einen Überblick über die vorliegende Datensituation für die acht verwendeten Datensätze.

Die Responsevariablen sind jeweils kategorial. Da CHAID in der derzeitigen Implementierung (siehe The FoRt Student Project Team, 2009) nur mit kate-

[1] Zur Konstruktion von Benchmark-Experimenten siehe Hothorn et al. (2005).

Tabelle 4.1 Überblick über die Datensätze. J ist dabei die Anzahl der Klassen der Responsevariable, n die Anzahl der Beobachtungen, NA die Anzahl der Beobachtungen mit mindestens einem fehlenden Wert und m die Anzahl der Kovariablen. Außerdem ist die Skala der Kovariablen angegeben.

	J	*n*	*NA*	*m*	*nominal*	*ordinal*	*stetig*
`BreastCancer`	2	699	16	9	4	5	–
`Diabetes`	2	768	–	8	–	–	8
`Glass`	6	214	–	9	–	–	9
`Glaucoma`	2	196	–	62	–	–	62
`Ionosphere`	2	351	–	33	1	–	32
`Sonar`	2	208	–	60	–	–	60
`Soybean`	19	683	121	35	30	5	–
`Vehicle`	4	846	–	18	–	–	18

gorialen Kovariablen umgehen kann, wurden die metrischen Kovariablen für die Berechnungen kategorisiert. Zu diesem Zweck wurden die 10%-Quantile der empirischen Verteilungsfunktionen berechnet und mittels dieser geordnete Faktoren erzeugt.
Desweiteren wurden zur Berechnung mit CHAID, CTREE und RPART nur vollständige Beobachtungen (über die R-Funktion `complete.cases()`) herangezogen, da in der Funktion `chaid()` fehlende Werte (NAs) zum jetzigen Zeitpunkt nicht weiter berücksichtigt werden.

4.2 Berechnung der Vorhersagefehler

Der Vorhersagefehler bzw. die Missklassifikationsrate gibt den Anteil an Beobachtungen des Testdatensatzes an, für die die Vorhersage und der tatsächliche Response nicht übereinstimmen.
Es wurden pro Datensatz 500 zufällige Stichproben mittels Bootstrap[2] gezogen. Die in der jeweiligen Bootstrap-Stichprobe enthaltenen Beobachtungen stellten dann den Lerndatensatz für die Berechnung dar. Der Testdatensatz für die Berechnung der jeweiligen Vorhersagefehler beziehungsweise Missklassifikationsraten wurde über die so genannte Out-of-Bag-Methode bestimmt, das heißt, alle Beobachtungen des ursprünglichen Datensatzes, die

[2]Die Funktionsweise von Bootstrap ist in Abschnitt 5.2 näher beschrieben.

nicht in der entsprechenden Bootstrap-Stichprobe enthalten waren, bildeten den Testdatensatz.

Die Verteilungen der Vorhersagefehler für zwei Verfahren werden als äquivalent angesehen, wenn sich die Vorhersagefehler nicht um mehr als 10% voneinander unterscheiden. Die Nullhypothese nicht-äquivalenter Vorhersagefehler wird dann definiert über das Verhältnis der Erwartungswerte der Verteilungen der Vorhersagefehler. Die Nullhypothese wird hier zum 5%-Niveau verworfen, wenn das zweiseitige 90%-Konfidenzintervall für das Verhältnis der Erwartungswerte der Vorhersagefehler nach Fieller (1940) komplett im Äquivalenzbereich $(0.9, 1.1)$ liegt.[3]

4.3 Ergebnisse

Die Abbildungen 4.1 bis 4.8 zeigen die Boxplots der Vorhersagefehler für CHAID, CTREE und RPART bei jeweils 500 Durchläufen für jeden der acht Datensätze.
Für die meisten Datensätze, mit Ausnahme von `Soybean` (Abbildung 4.7), liegt der Vorhersagefehler von CHAID leicht über dem von CTREE und RPART.

[3] Siehe hierzu Hothorn et al. (2006).

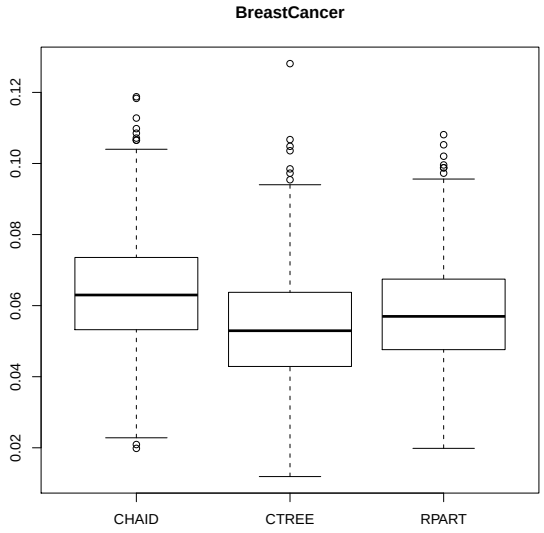

Abbildung 4.1 Vorhersagefehler von CHAID, CTREE und RPART für den Datensatz `BreastCancer`

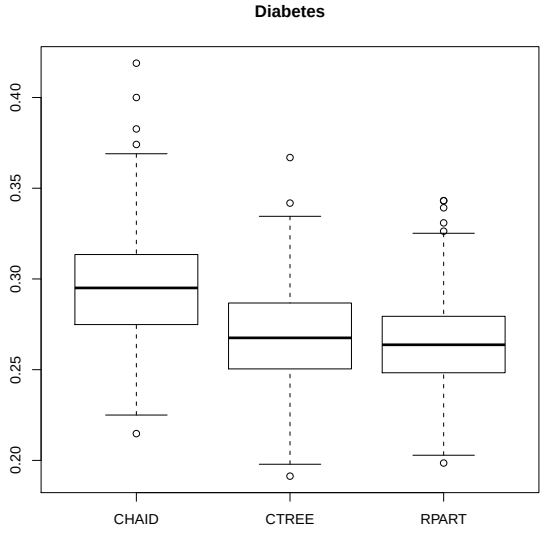

Abbildung 4.2 Vorhersagefehler von CHAID, CTREE und RPART für den Datensatz `Diabetes`

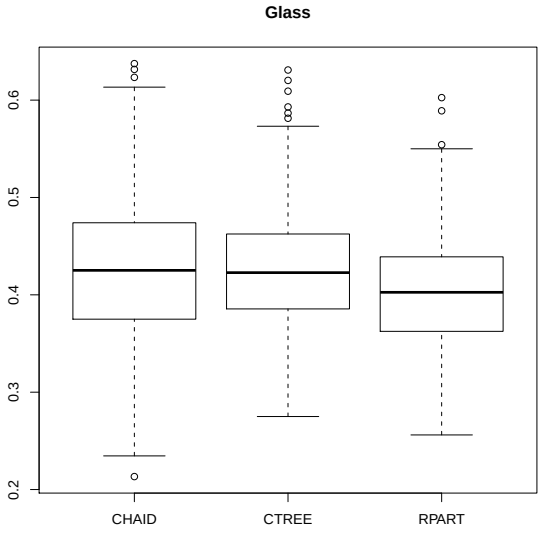

Abbildung 4.3 Vorhersagefehler von CHAID, CTREE und RPART für den Datensatz `Glass`

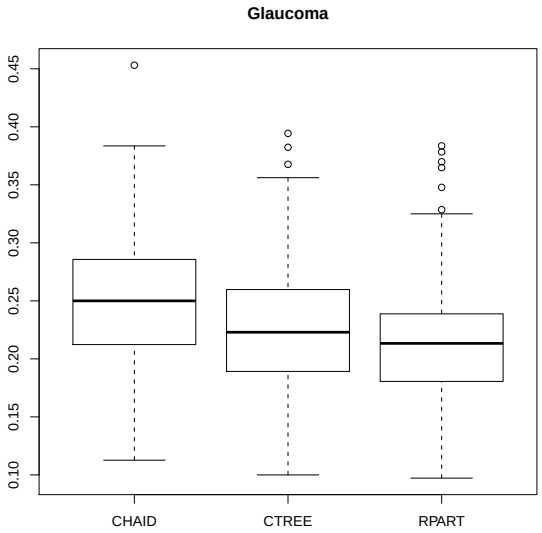

Abbildung 4.4 Vorhersagefehler von CHAID, CTREE und RPART für den Datensatz `Glaucoma`

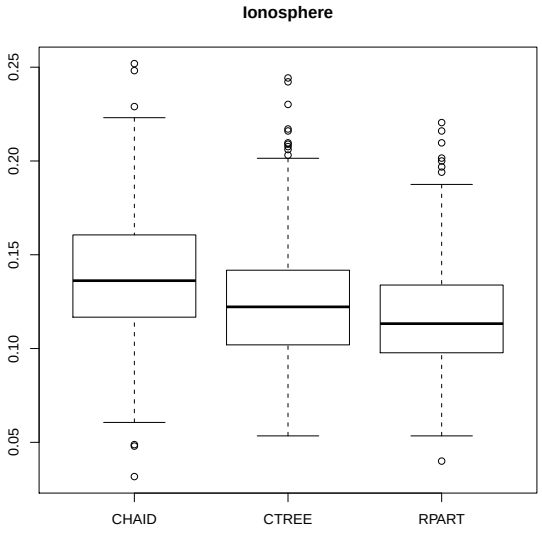

Abbildung 4.5 Vorhersagefehler von CHAID, CTREE und RPART für den Datensatz `Ionosphere`

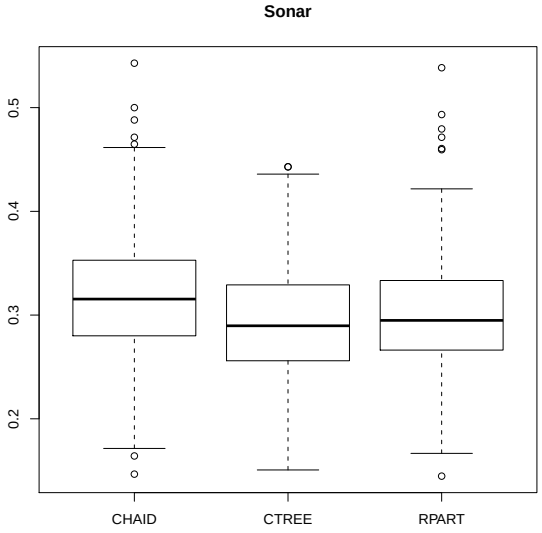

Abbildung 4.6 Vorhersagefehler von CHAID, CTREE und RPART für den Datensatz `Sonar`

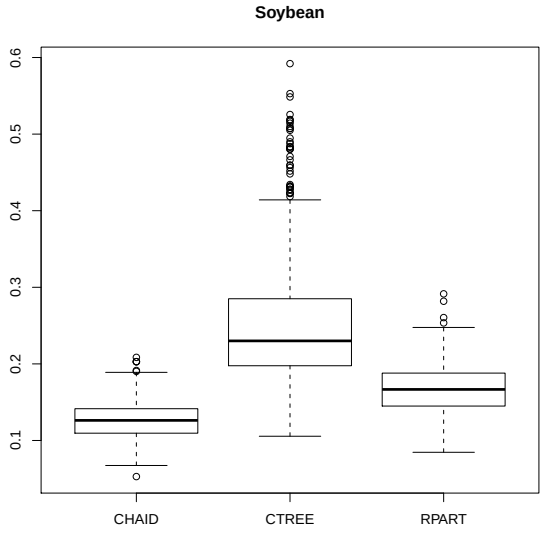

Abbildung 4.7 Vorhersagefehler von CHAID, CTREE und RPART für den Datensatz `Soybean`

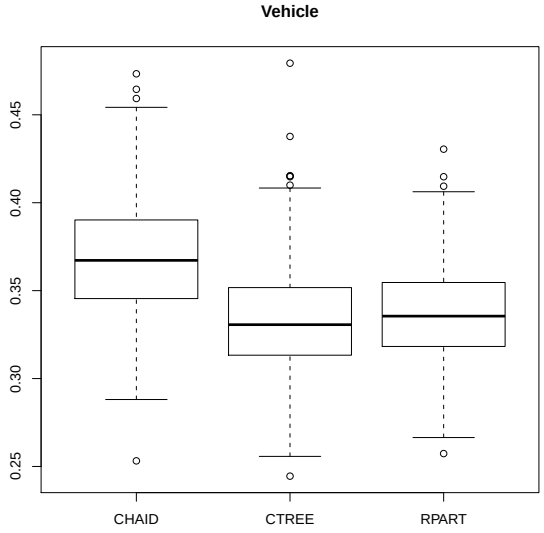

Abbildung 4.8 Vorhersagefehler von CHAID, CTREE und RPART für den Datensatz `Vehicle`

Tabelle 4.2 Ergebnisse des Kruskal-Wallis Rangsummen-Tests.

Datensatz	Test-Statistik	p-Wert
`BreastCancer`	84.18	0.00
`Diabetes`	313.28	0.00
`Glass`	39.12	0.00
`Glaucoma`	137.13	0.00
`Ionosphere`	126.20	0.00
`Sonar`	49.60	0.00
`Soybean`	883.09	0.00
`Vehicle`	306.74	0.00

Zum Vergleich der Verteilungen der Vorhersagefehler von CHAID, CTREE und RPART wurde jeweils ein Kruskal-Wallis Rangsummen-Test durchgeführt. Tabelle 4.2 zeigt die Testergebnisse. Die Nullhypothese, dass die Verteilungsparameter der Vorhersagefehler für alle drei Verfahren identisch sind, wird für jeden Datensatz zu einem Niveau $\alpha = 0.05$ verworfen. Das bedeutet, dass die Verteilungen der Vorhersagefehler jeweils für mindestens zwei der drei Verfahren unterschiedlich sind.

Abbildung 4.9 enthält neben den Boxplots der Verteilungen der paarweisen Verhältnisse der Vorhersagefehler bzw. Missklassifikationsraten für CHAID und CTREE, auch Schätzer für das Verhältnis der erwarteten Vorhersagefehler sowie die zugehörigen 90%-Fieller-Konfidenzintervalle. Beispielsweise bedeutet ein Schätzer für das Verhältnis der Missklassifikationsraten von CHAID und CTREE in Höhe von 1.080 für den `Sonar`-Datensatz, dass die Missklassifikationsrate von CHAID im Mittel um 8% höher ist als die Missklassifikationsrate von von CTREE. Da aber das zugehörige Konfidenzintervall von $(1.062, 1.098)$ im vorher definierten Äquivalenzbereich von $\pm 10\%$ liegt, sind die Vorhersagefehler von CHAID und CTREE für den `Sonar`-Datensatz als gleichwertig anzusehen.
Für den Datensatz Glass sind die Missklassifikationsraten von CHAID und CTREE ebenfalls äquivalent.
Für die restlichen Datensätze bestätigt sich das Bild aus den vorhergehenden Abbildungen 4.1 – 4.8, dass CHAID meistens ein etwas schlechteres Ergebnis als CTREE liefert. Zum Beispiel liegt der Schätzer für das Verhältnis der

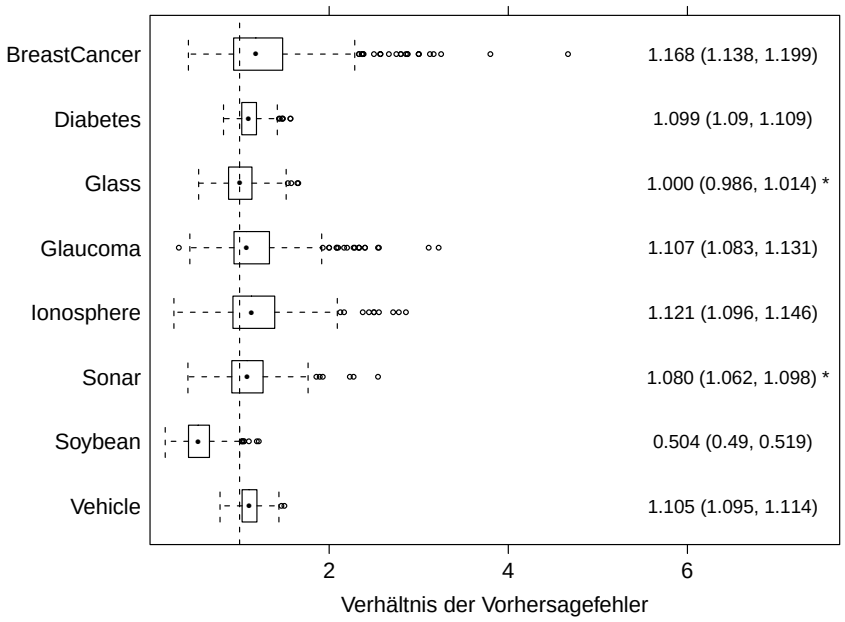

Abbildung 4.9 Verteilungen der paarweisen Verhältnisse der Vorhersagefehler für CHAID und CTREE zusammen mit Schätzungen und 90%-Fieller-Konfidenzintervallen für das Verhältnis der Erwartungswerte der Vorhersagefehler. Die Sternchen markieren äquivalente Vorhersagefehler, das heißt, das Konfidenzintervall liegt im Äquivalenzbereich (0.9,1.1).

Missklassifikationsraten beim Datensatz `BreastCancer` bei 1.168, das bedeutet, die Missklassifikationsrate von CHAID ist hier im Mittel um 16.8% höher als die Missklassifikationsrate von CTREE. Das zugehörige Fieller-Konfidenzintervall $(1.138, 1.199)$ liegt ebenfalls nicht mehr im Äquivalenzbereich.

Allerdings gibt es eine deutliche Ausnahme, den Datensatz `Soybean`. Bereits in Abbildung 4.7 ist zu erkennen, das CHAID für diesen Datensatz eine deutlich niedrigere Missklassifikationsrate als CTREE aufweist und auch der Schätzer für das Verhältnis der Missklassifikationsraten in Höhe von 0.504 bestätigt dieses Ergebnis, das heißt, die Missklassifikationsrate von CHAID für `Soybean` ist im Mittel nur etwa halb so groß wie die Missklassifikationsrate von CTREE.

Kapitel 5

Computationale Details

5.1 Die verwendeten R-Pakte

5.1.1 CHAID

Das Paket `CHAID` (The FoRt Student Project Team, 2009) stellt eine Implementierung des CHAID-Algorithmus (Kass, 1980), wie in Abschnitt 2.1 beschrieben, zur Verfügung. Der Funktionsaufruf selbst erfolgt über die folgende Funktion:

```
chaid(formula, data, subset, weights, na.action = na.omit,
      control = chaid_control())
```

Tabelle 5.1 Überblick über die Parameter der Funktion `chaid()`.

Parameter	Details
`formula`	symbolische Beschreibung des anzupassenden Modells in der Form `Response` $\sim$ `Kovariablen`
`data`	ein optionaler Data Frame, der die Variablen des Modells enthält
`subset`	ein optionaler Vektor, der eine Teilmenge von Beobachtungen spezifiziert, die zur Anpassung des Modells verwendet werden sollen
`weights`	ein optionaler Gewichtsvektor für die Beobachtungen
`na.action`	eine Funktion, die angibt, wie mit fehlenden Werten umgegangen werden soll, die Default-Einstellung ist `na.omit`, d.h. Beobachtungen, die NAs enthalten, werden entfernt
`control`	Hyperparameter des Algorithmus, die von der Funktion `chaid.control()` zurückgegeben werden

Tabelle 5.1 gibt einen Überblick über die Eingabewerte dieser Funktion. Sowohl die Responsevariable als auch die Kovariablen bzw. Prädiktorvariablen müssen kategorial sein.

Tabelle 5.2 Überblick über die Parameter der Funktion `chaid_control()`.

Parameter	Details
`alpha2`	Signifikanzniveau, das zum Zusammenlegen von Kategorien der Prädiktorvariable(n) benutzt wird (Schritt 2), der Defaultwert ist 0.05
`alpha3`	wenn ein positiver Wert $\leq$ 1 zugewiesen wird: Signifikanzniveau, das zum Splitten bereits zusammengelegter Kategorien der Prädiktorvariable(n) benutzt wird (Schritt 3), andernfalls wird Schritt 3 nicht ausgeführt (die Defaulteinstellung)
`alpha4`	Signifikanzniveau, das zum Splitten eines Knotens im signifikantesten Prädiktor verwendet wird (Schritt 4), der Default ist 0.05
`minsplit`	Anzahl an Beobachtungen im gesplitteten Response, bei der nicht mehr weiter gesplittet wird, der Default liegt bei 20 Beobachtungen
`minbucket`	Minimum der Anzahl an Beobachtungen in jedem Endknoten, der Default ist 7 Beobachtungen
`minprob`	minimale Häufigkeit der Beobachtungen in Endeknoten, der Default liegt bei 0.01
`stump`	logischer Wert, wenn auf TRUE gesetzt, werden nur Root-Splits (der erste Split) angewendet, der Default ist FALSE

Die Funktion `chaid_control()` erlaubt die Steuerung der Hyperparameter des Algorithmus, Tabelle 5.2 zeigt die zugehörigen Eingabe- und Defaultwerte. In der vorliegenden Arbeit wurden in der Regel die Defaultwerte benutzt. Die hauptsächliche Ausnahme stellt hierbei die Simulation in Kapitel 3 dar, bei der nur der Root-Split durchgeführt, aber erzwungen wurde. Dies geschah mittels `chaid_control(alpha4 = 1, stump = TRUE)`.
Vorhersagen sind über die Funktion `predict()` mit dem gefitteten Modell und einem Data Frame, der die neuen Beobachtungen enthält, als Eingabewerte möglich.

Gemeinsam mit `CHAID` wird auch das R-Paket `partykit` (Hothorn und Zeileis, 2009) geladen, welches die benötigten Routinen zur Darstellung von

inneren und terminalen Knoten von Bäumen bereitstellt.

5.1.2 party

Das R-Paket party enthält die Implementierung des CTREE-Algorithmus (Hothorn et al., 2006). Der Funktionsaufruf des CTREE-Algorithmus selbst erfolgt in diesem Paket über die folgende Funktion:

```
ctree(formula, data, subset = NULL, weights = NULL,
      controls = ctree_control(), xtrafo = ptrafo,
      ytrafo = ptrafo, scores = NULL)
```

Details zu den Eingabewerten dieser Funktion sind in Tabelle 5.3 zu finden.

Tabelle 5.3 Überblick über die Parameter der Funktion `ctree()`.

Parameter	Details
`formula`	symbolische Beschreibung des anzupassenden Modells in der Form `Response` $\sim$ `Kovariablen`
`data`	ein Data Frame, der die Variablen des Modells enthält
`subset`	ein optionaler Vektor, der eine Teilmenge von Beobachtungen spezifiziert, die zur Anpassung des Modells verwendet werden sollen
`weights`	ein optionaler Gewichtsvektor für die Beobachtungen, nur nicht-negative ganzzahlige Werte sind zugelassen
`controls`	ein Objekt vom Typ `TreeControl`, das man mittels `ctree_control()` erhält
`xtrafo`	eine Funktion, die auf alle Prädiktorvariablen angewendet wird, per Default wird `ptrafo()` angewendet
`ytrafo`	eine Funktion, die auf alle Responsevariablen angewendet wird, per Default wird `ptrafo()` angewendet
`scores`	eine optionale Liste mit Scores, die geordneten Faktoren angehängt werden soll

Für eine genaue Beschreibung der Parameter von `ctree_control()` sei auf Hothorn et al. (2006) verwiesen. Genau wie bei CHAID wurden in dieser Arbeit mit Ausnahme der Simulation in Kapitel 3, die Defaultwerte verwendet. Bei der Simulation erfolgte das Erzwingen des Root-Splits über `party::ctree_control(stump = TRUE, mincriterion = 0)`. Der Parameter `mincriterion` gibt hierbei den Wert der Teststatistik bzw. 1 – p-Wert

an, der erreicht werden muss, damit ein Split angewendet wird. Das Argument `stump` sorgt dafür, dass nach dem Root-Split abgebrochen wird.

Es ist zu beachten, dass, wenn sowohl `party`, als auch das Paket `partykit` geladen sind, der Funktionsaufruf in der Form

`party::ctree()` bzw. `party::ctree_control()`

erfolgen muss, da `partykit` gleichnamige Funktionen enthält.

Vorhersagen erhält man wiederum über die Funktion `predict()`.

5.1.3 `rpart`

Wie bereits in Abschnitt 2.3 erwähnt, enthält das Paket `rpart` (Therneau und Atkinson, 1997) eine Implementierung des CART-Algorithmus. Der Funktionsaufruf erfolgt über

```
rpart(formula, data, weights, subset, na.action = na.rpart,
      method, model = FALSE, x = FALSE, y = TRUE, parms,
      control, cost, ...).
```

Tabelle 5.4 gibt einen Überblick über die Eingabewerte dieser Funktion.

Für weitere Details zu den einzelnen Parametern der Funktion sei auf Therneau und Atkinson (1997) verwiesen. Zur Berechnung der Vorhersagefehler in Kapitel 4 wurden die Defaulteinstellungen von `rpart()` verwendet.

Mittels der im R-Paket enthaltenen Funktion `prune()` ist es möglich, die Bäume zu beschneiden und somit dem Problem der Überanpassung zu begegnen. Die Funktion wurde dabei auf folgende Weise aufgerufen:

```
prune(tree, cp = tree$cptable[which.min(tree$cptable[,4]),1])
```

Dabei ist `tree` der mittels `rpart()` berechnete Baum und `cp` ist ein Komplexitätsparameter, auf den der Baum „zurückgeschnitten“ wird. Auch hier können über die Funktion `predict()` wie für CHAID und CTREE Vorhersagen berechnet werden.

Für weitere Details zum Paket `rpart` sei wiederum auf Therneau und Atkinson (1997) verwiesen.

Tabelle 5.4 Überblick über die Parameter der Funktion `rpart()`.

Parameter	Details
`formula`	symbolische Beschreibung des anzupassenden Modells in der Form `Response` $\sim$ `Kovariablen`
`data`	ein optionaler Data Frame, der die Variablen des Modells enthält
`weights`	ein optionaler Gewichtsvektor für die Beobachtungen
`subset`	ein optionaler Vektor, der eine Teilmenge von Beobachtungen spezifiziert, die zur Anpassung des Modells verwendet werden sollen
`na.action`	in der Defaulteinstellung werden alle Beobachtungen mit fehlendem Response entfernt, aber Beobachtungen mit NAs in den Kovariablen werden beibehalten
`method`	'anova', 'poisson', 'class' oder 'exp', wird keine Angabe gemacht, so versucht die Funktion anhand des Response, eine Entscheidung zu treffen, beispielsweise wird bei kategorialem Response `method = 'class'` gesetzt, alternativ kann method auch eine Liste von Funktionen sein
`model`	falls logisch: Angabe, ob eine Kopie des Modell Frame im Ergebnis beibehalten werden soll, falls ein Modell Frame: wird anstelle der Konstruktion neuer Daten verwendet
`x`	Beibehalten einer Kopie der x-Matrix im Ergebnis
`y`	Beibehalten einer Kopie der abhängigen Variable, falls fehlend, wird ein Modell zur Verfügung gestellt
`parms`	optionale Parameter der Split-Funktion
`control`	Optionen, die die Details des RPART-Algorithmus kontrollieren
`cost`	ein Vektor nicht-negativer Kosten für die Variablen im Modell

5.2 Bootstrap

Die Bootstrap-Methode ist ein generelles Werkzeug, das es ermöglicht, die Genauigkeit statistischer Verfahren abzuschätzen.
Man betrachte die folgende Situation: Es liegt ein Lerndatensatz vom Umfang n vor. Die grundsätzliche Idee ist es, zufällig Datensätze mit Zurücklegen aus den Lerndaten zu ziehen (Resampling), wobei jede Stichprobe wieder den Umfang n des ursprünglichen Lerndatensatzes hat. Das wird B-mal wiederholt (zum Beispiel war für die Berechnungen in Kapitel 4 $B = 500$) und

somit erhält man B Bootstrap-Stichproben. Da mit Zurücklegen gezogen wird, können einzelne Beobachtungen mehrmals in eine Stichprobe gelangen. Die jeweilige Bootstrap-Stichprobe lässt sich über einen Gewichtsvektor $\mathbf{w}_b$, $b = 1, \ldots, B$ für den ursprünglichen Lerndatensatz mit nicht-negativen ganzzahligen Einträgen darstellen. Der entsprechende Eintrag für eine Beobachtung im Gewichtsvektor $\mathbf{w}_b$ ist Null, wenn die Beobachtung nicht Teil der Bootstrap-Stichprobe ist und ansonsten gibt der Eintrag an, wie oft die Beobachtung in der Bootstrap-Stichprobe enthalten ist. Für jede der B Bootstrap-Stichproben wird dann ein Modell unter Verwendung des erhaltenen Gewichtsvektors $\mathbf{w}_b$ für den ursprünglichen Lerndatensatz angepasst und jeweils für einen Testdatensatz der entsprechende Vorhersagefehler von CHAID, CTREE und RPART ermittelt.
In dieser Arbeit wurden die Testdatensätze über die so genannte Out-of-Bag-Methode bestimmt. Alle Beobachtungen die nicht in der jeweiligen Bootstrap-Stichprobe enthalten waren, das heißt, deren Eintrag im Gewichtsvektor $\mathbf{w}_b$ gleich Null war, bildeten den entsprechenden Testdatensatz zur Ermittlung der Vorhersagefehler für die drei Verfahren.
Generell gelangen durchschnittlich etwa 63.2% der ursprünglichen Beobachtungen in die Bootstrap-Stichprobe, was sich einfach zeigen lässt:

$$\begin{aligned} P\{\text{Beobachtung } i \in \text{Bootstrap-Stichprobe } b\} &= 1 - \left(1 - \frac{1}{n}\right)^n \\ &\approx 1 - e^{-1} \\ &= 0.632 \end{aligned}$$

Die restlichen 36.8% (durchschnittlicher Wert) der Daten bilden dann den entsprechenden Test-Datensatz.

Eine generelle Einführung sowie weitere Details zu Bootstrap sind bei Hastie et al. (2008) zu finden.

Kapitel 6

Diskussion

CHAID und CTREE sind zwei Ansätze, die rekursives Partitionieren ermöglichen und dabei versuchen, die beiden klassischen Probleme der Überanpassung und der verzerrten Variablenselektion zu lösen. In beiden Algorithmen wird dabei der Prozess der Variablenselektion vom eigentlichen Splittingprozess getrennt. Die implementierten Stopp-Kriterien basieren in beiden Verfahren auf formalen Hypothesentests und sind somit zum einen statistisch motiviert und zum anderen intuitiv verständlich.
Bereits in Kapitel 2 wurde erwähnt, dass CTREE wesentlich universeller einsetzbar ist als CHAID, da die Anwendung im Gegensatz zu CHAID nicht auf Klassifikationsbäume mit kategorialem Response und kategorialen Kovariablen beschränkt ist. Im Fokus dieser Arbeit standen aber genau diese Klassifikationsbäume und damit verbunden der Vergleich der beiden Verfahren mittels simulierter Daten und anhand realer Daten bezüglich ihrer Vorhersagegüte.

Die Benchmark-Experimente in Kapitel 4 zum Vergleich der Vorhersagegüte zeigen, dass CHAID in der Regel etwas größere Vorhersagefehler als CTREE liefert. Das Beispiel des Datensatzes `Soybean` macht deutlich, dass dies allerdings nicht in jeder Situation der Fall ist. Generell ist es so, dass sich fast immer Situationen finden lassen, in denen ein Algorithmus, der oft besser ist als konkurrierende Verfahren, plötzlich schlechtere Ergebnisse liefert und umgekehrt. Man wird kaum ein Verfahren finden, das generell besser ist als sämtliche Alternativen, da die Anwendbarkeit und somit auch die Güte eines Verfahrens immer auch von der vorliegenden Datensituation abhängt.

Die wesentlichen Unterschiede der beiden Verfahren werden vor allem durch die Simulation im ersten Teil von Kapitel 3 deutlich. Bei CHAID wird trotz Unabhängigkeit zwischen Response und Kovariablen sehr häufig eine Kovariable zum Splitten ausgewählt. In der durchgeführten Simulation war dies in ca. einem Viertel der Durchläufe der Fall. Im Gegensatz dazu ist die Wahrscheinlichkeit für solch eine falsche Entscheidung bei CTREE durch α (in der Regel $\alpha = 0.05$) nach oben begrenzt. Ursache für den hohen Anteil falscher Entscheidungen bei CHAID ist die fehlende Adjustierung der berechneten p-Werte bezüglich der Anzahl an Kovariablen, die zum Splitten in Frage kommen.
Außerdem hat die Simulation ein weiteres gravierendes Problem bei der Anwendung von CHAID aufgezeigt. Es liegt eine deutliche Verzerrung der Variablenselektion hin zu Variablen mit möglichst wenigen Splitmöglichkeiten vor. Klassische Ansätze, wie beispielsweise CART, weisen genau die gegenteilige Problematik auf. Bei diesen Verfahren wird man eher mit dem Problem konfrontiert, dass die Variablenselektion dahingehend verzerrt ist, dass Variablen mit möglichst vielen Splitmöglichkeiten bevorzugt werden.
Die im zweiten Teil von Kapitel 3 durchgeführte Simulation zum Vergleich der ursprünglichen mit den adjustierten p-Werten hat deutlich gemacht, dass bei CHAID die Verzerrung auf die verwendeten Korrekturfaktoren für die berechneten p-Werte zurückzuführen ist. Beispielsweise waren die adjustierten p-Werte für einen ungeordneten Faktor mit acht Faktorstufen unter Unabhängigkeit in gut 75% der Simulationsdurchläufe größer 1. Auch bei $\alpha = 1$, was bedeutet, dass eigentlich in jedem Fall gesplittet werden müsste, kommt dann die entsprechende Variable zum Splitten nicht mehr in Betracht. Binäre Kovariablen sind von der Problematik nicht betroffen, da für sie keine p-Wert-Korrektur bezüglich der Anzahl an Variablenlevels vorgenommen werden muss, der entsprechende p-Wert somit auch nicht größer sein kann als 1 und bei erzwungenem Root-Split die binären Variablen noch immer potentiell zum Splitten in Frage kommen.

Aufgrund der verzerrten Variablenselektion, sind die Ergebnisse von CHAID eher mit Vorsicht zu betrachten. Da die Variablenselektion bei CTREE unverzerrt ist, was bereits von Hothorn et al. (2006) gezeigt wurde, ist es in der Regel ratsam, CTREE anstelle von CHAID anzuwenden.

Kapitel 7

Zusammenfassung

Klassifikations- und Regressionsbäume stellen ein immer beliebteres Werkzeug für die angewandten Datenanalyse dar. Im Wesentlichen möchte man mit Hilfe dieser Verfahren zwei Ziele erreichen: Erklärung und Vorhersage. Neben der Erklärung der den Daten zugrunde liegenden Struktur möchte man häufig gleichzeitig auch in der Lage sein, Vorhersagen für neue Beobachtungen zu treffen. Zwar liefern beispielsweise Methoden wie Support Vector Machines (SVM) oft niedrigere Vorhersagefehler. Möchte man aber die Struktur verstehen, die den Daten zugrunde liegt, sind die Ergebnisse von Klassifikations- und Regressionsbäumen in der Regel aus Anwendersicht intuitiver und somit einfacher zu verstehen bzw. zu interpretieren.
Ziel dieser Arbeit war der Vergleich zweier Verfahren, die es ermöglichen, Klassifikationsbäume zu erstellen, basierend auf formalen statistischen Hypothesentests. CHAID und CTREE stellen Lösungsansätze für die zwei fundamentalen Probleme rekursiver Partitionsalgorithmen dar: der Überanpassung und der verzerrten Variablenselektion hin zu Kovariablen mit vielen Splitmöglichkeiten. Diese Problematiken sind beispielsweise von Exhaustionsmethoden wie CART bekannt.
Die Simulation aus Kapitel 3 dieser Arbeit macht deutlich, dass im Gegensatz zu CTREE bei CHAID eine deutliche Verzerrung der Variablenselektion hin zu Variablen mit möglichst wenigen Splitmöglichkeiten gegeben ist. Darüber hinaus erfolgt bei Unabhängigkeit zwischen dem Response und den Kovariablen bei CHAID in etwa einem Viertel der Fälle dennoch ein Root-Split. Bei CTREE ist diese Wahrscheinlichkeit deutlich niedriger und mit α nach oben begrenzt. Im zweiten Teil von Kapitel 3 hat sich gezeigt, dass diese

Probleme bei CHAID im Wesentlichen auf die Adjustierung der berechneten p-Werte und die dafür verwendeten Korrekturfaktoren zurückzuführen sind. Die Benchmark-Experimente in Kapitel 4 zeigen, dass CHAID für viele Datensätze einen schlechteren Vorhersagefehler liefert als CTREE, aber dies wiederum nicht immer der Fall ist, was am Datensatz `Soybean` deutlich wird. Dennoch ist es in jedem Fall ratsam, eher CTREE zu verwenden als CHAID, da die Verzerrung der Variablenselektion sich natürlich auf die Interpretierbarkeit der Ergebnisse auswirkt und diese letztendlich mit etwas Skepsis betrachtet werden sollten.

Anhang A

R-Code – Simulationen

A.1 Simulation zur Unverzerrtheit

Dieser Abschnitt enthält den R-Code für die Simulation aus Abschnitt 3.1 zum Vergleich von CHAID und CTREE bezüglich der Unverzerrtheit. Der Code basiert in weiten Teilen auf dem R-Code, welcher der Simulation in der Arbeit von Hothorn et al. (2006) zu Grunde lag.

Teil 1 (sim_bin.R):
Dieser Teil enthält die Funktionen, die für die Simulation selbst notwendig sind.

```
## Simulation y ~ B(1,p)

library("party")
library("CHAID")

Nsim <- 10000
set.seed(290875)

foo <- function(n, p = 0.5) {
    df <- as.data.frame(matrix(0, nrow = n, ncol = 6))

    df[1] <- sample(gl(4,n,ordered=TRUE))[1:n]
    df[2] <- sample(gl(6,n,ordered=TRUE))[1:n]
    df[3] <- sample(gl(8,n,ordered=TRUE))[1:n]
    df[4] <- sample(gl(4,n))[1:n]
    df[5] <- sample(gl(6,n))[1:n]
    df[6] <- sample(gl(8,n))[1:n]
```

```
    df$V7 <- sample(gl(2, n/2))
    df$y <- rep(0, n)

    df$y[df$V7 == "1"] <- rbinom(n/2, 1, prob = p)
    df$y[df$V7 == "2"] <- rbinom(n/2, 1, prob = (1 - p))

    df$y <- as.factor(df$y)

    df
}

loop <- function(Nsim, p = 0.5) {

    sctree <- vector(mode = "character", length = Nsim)
    pvalue_ctree <- vector(mode = "numeric", length = Nsim)
    schaid <- vector(mode = "character", length = Nsim)
    pvalue_chaid <- vector(mode = "numeric", length = Nsim)

    for (i in 1:Nsim) {
        print(i)
        df <- foo(100, p = p)

        ctrl <- party::ctree_control(stump = TRUE,
                                     mincriterion = 0)
        mod <- party::ctree(y ~ ., data = df, controls = ctrl)
        sctree[i] <- names(df)[which.max(mod@tree[[3]][[2]])]
        pvalue_ctree[i] <- (1 - mod@tree[[3]][[3]])

        ctrl <- CHAID::chaid_control(alpha4 = 1, stump = TRUE)
        mod <- chaid(y ~ ., data = df, control = ctrl)
        schaid[i] <- names(node_party(mod)$split$varid)
        pvalue_chaid[i] <- min(node_party(mod)$info$adjpvals)
    }

    ret <- data.frame(sctree = factor(sctree, levels =
                                      paste("V", 1:7, sep = "")),
                      pvalue_ctree = pvalue_ctree,
                      schaid = factor(schaid, levels =
                                      paste("V", 1:7, sep = "")),
                      pvalue_chaid = pvalue_chaid)
```

```
    attr(ret, "p") <- p
    return(ret)
}
```

Teil 2 (run_0.5.R):
Dies ist ein Beispiel-Code für $p_0 = 0.5$ (Die Simulation wurde für $p_0 = 0.5, 0.6, \ldots, 1$ durchgeführt.).

```
p <-  0.5
source("sim_bin.R")

ret <- loop(Nsim, p)

save(ret, file = paste("ret_", p, ".rda", sep = ""))
```

Teil 3 (goodman.R):
Die folgende Funktion dient zur Berechnung der Konfidenzintervalle für Anteilswerte nach Goodman (1965).

```
goodman <- function(freq, alpha = 0.05){
    k <- length(freq)
    int <- matrix(nrow = k, ncol = 2)
    n <- sum(freq)
    B <- qchisq(p = 1 - alpha / k, df = 1)

    for(i in 1:k){
        a <- (1 - B / n)
        b <- (- 2 * as.numeric(freq[i]) - B) / n
        c <- (as.numeric(freq[i]) / n)^2

        int[i,1] <- ((-b) - sqrt(b^2 - 4 * a * c)) / (2 * a)
        int[i,2] <- ((-b) + sqrt(b^2 - 4 * a * c)) / (2 * a)
    }
    return(int)
}
```

Teil 4:
Dieser Teil fasst die Ergebnisse der Simulation zusammen.

```
source("goodman.R")

p <- seq(from = 0.5, to = 1, by = 0.1)
try(rm(ret))
Nsim <- 10000
out <- c()
fctree <- c()
fchaid <- c()

for (m in p) {
    load(paste("ret_", m, ".rda", sep =""))
    out <- rbind(out, c(mean(ret$pvalue_ctree < 0.05),
                        if(sum(ret$pvalue_ctree < 0.05) == 0) {0}
                        else { sum(ret$sctree == "V7" &
                                   ret$pvalue_ctree < 0.05) /
                               sum(ret$pvalue_ctree < 0.05) },
                        mean(ret$pvalue_chaid < 0.05),
                        if(sum(ret$pvalue_chaid < 0.05) == 0) {0}
                        else { sum(ret$schaid == "V7" &
                               ret$pvalue_chaid < 0.05) /
                               sum(ret$pvalue_chaid < 0.05) }))
    fctree <- rbind(fctree, table(ret$sctree) / Nsim)
    fchaid <- rbind(fchaid, table(ret$schaid) / Nsim)
    rm(ret)
}

rownames(out) <- p
colnames(out) <- c("split_ctree", "correct_ctree",
                   "split_chaid", "correct_chaid")
out <- as.data.frame(out)

pdf("power_split.pdf", width = 8, height = 5)

cex <- 1.15

layout(matrix(1:2, nc = 2))
par(mar=c(5, 6, 4, 2) + 0.1)
plot(p, out[["split_ctree"]], type = "l", lty = 1, ylim = c(0,1),
     ylab = "Simulierte Power", xlab = "p_0", cex.axis = cex,
     cex.lab = cex)
lines(p, out[["split_chaid"]], lty = 2)
```

```
abline(h = 0.05, lty = 3)
legend(0.71, 0.89, lty = c(1, 2), legend = c("CTREE", "CHAID"),
       cex = 1, bty = "n")

plot(p, out[["correct_ctree"]], type = "l", lty = 1,
     ylim = c(0,1), ylab = "Bedingte Wahrscheinlichkeit\n
                            für den korrekten Split", xlab = "p_0",
     cex.axis = cex, cex.lab = cex)
lines(p, out[["correct_chaid"]], lty = 2)
legend(0.71, 0.89, lty = c(1, 2), legend = c("CTREE", "CHAID"),
       cex = 1, bty = "n")

dev.off()

print(out)

print(fctree)
print(fchaid)

c1 <- round(goodman(fctree[1,] * Nsim), 3) ## fuer p = 0.5
c2 <- round(goodman(fchaid[1,] * Nsim), 3)

selprob <- cbind(fchaid[1,], c2[,1], c2[,2],
                 fctree[1,], c1[,1], c1[,2])

library("xtable")
xtable(selprob, digits = rep(3, 7))
```

A.2 Simulation zur Verzerrung bei CHAID

Dieser Abschnitt enthält den R-Code zu der Simulation aus Abschnitt 3.3, bei der untersucht werden sollte, ob die Korrekturfaktoren als mögliche Ursache der Verzerrung von CHAID in Frage kommen.

Teil 1 (chaid_temp.R):
Dieser Teil des R-Codes beinhaltet eine Abänderung der Funktionen aus dem R-Paket `CHAID` und sorgt dafür, dass zum einen nur die Schritte 1 bis 4 des Algorithmus ausgeführt werden und zum anderen, dass neben dem adjustierten p-Wert in Schritt 4 auch der unkorrigierte p-Wert ausgegeben wird.

```
step4internal_temp <- function(response, x, weights, index) {
```

```
    if (nlevels(response[, drop = TRUE]) < 2) return(0)
    mx <- mergex(x, index)
    c_levels <- nlevels(x[weights > 0, drop = TRUE])
    r_levels <- nlevels(mx)
    xytab <- xtabs(weights ~ response + mx)

    if (sum(colSums(xytab) > 0) < 2) return(0)

    logp <- logchisq.test(xytab)

    ret <- vector(mode="numeric", length = 2)
    ret[1] <- exp(logp) ## original p-Wert
    if (is.ordered(x)) {
        ### formula (3.1) in Kass (1980)
        ret[2] <- exp(logp + lchoose(c_levels - 1, r_levels - 1))
    } else {
        i <- 0:(r_levels - 1) ### formula (3.2)
        fact <- sum((-1)^i * ((r_levels - i)^c_levels) /
                (factorial(i) * factorial(r_levels - i)))
        ret[2] <- exp(logp + log(fact))
    }
    attr(logp, "Chisq") <- attr(logp, "Chisq")
    return(ret)
}

step4_temp <- function(response, xvars, weights, indices) {
    p <- numeric(length(xvars))
    p_org <- numeric(length(xvars))
    X2 <- rep(NA, length(xvars))
    for (i in 1:length(xvars)) {
        tmp <- step4internal_temp(response, xvars[[i]],
                                  weights, indices[[i]])
        p[i] <- tmp[2]
        p_org[i] <- tmp[1]
        if (!is.null(attr(tmp, "Chisq"))) X2[i] <- attr(tmp, "Chisq")
    }
    names(p) <- names(xvars)
    attr(p, "Chisq") <- X2
    return(cbind(p,p_org))
}
```

```
step5_temp <- function(id = 1L, response, x, weights = NULL,
                       indices = NULL, ctrl = chaid_control()) {
    if (is.null(weights)) weights <- rep.int(1, length(response))
    if (sum(weights) < ctrl$minsplit) return(partynode(id = id))

    if (ctrl$stump && id > 1) return(partynode(id = id))

    indices <- step1(response, x, weights, indices = indices, ctrl)

    pvals <- step4_temp(response, x, weights, indices)
    return(pvals)
}

chaid_temp <- function(formula, data, subset, weights,
                       na.action = na.omit,
                       control = chaid_control()){
      mf <- match.call(expand.dots = FALSE)
      m <- match(c("formula", "data", "subset", "weights",
                   "na.action"), names(mf), 0)
      mf <- mf[c(1, m)]
      mf$drop.unused.levels <- FALSE
      mf[[1]] <- as.name("model.frame")
      m <- eval.parent(mf)
      y <- model.response(m)
      x <- m[, c(-1, -which(names(m) == "(weights)")), drop = FALSE]
      w <- model.weights(m)
      chaid_pvals <- step5_temp(1L, y, x, weights = w,
                                ctrl = control)

      return(chaid_pvals)
}
```

Teil 2:
Dies ist der benötigte Code für die eigentliche Simulation.

```
## Simulation bzgl. der Verzerrung von CHAID

library("CHAID")

## Umgehen des NAMESPACE, um auf die internen
## Funktionen aus CHAID zuzugreifen zu koennen
```

```
attach(asNamespace("CHAID"))

n_total <- 500  ## Simulationsdurchlaeufe
Nsim <- 1000    ## Simulationsdurchlaeufe p-Wert
n_dat <- 500    ## Umfang des gezogenen Datensatzes

p_all <- as.data.frame(matrix(0, nrow = n_total, ncol = 9))
names(p_all) <- c("p1_org", "p1_adj", "p1_sim",
  "p2_org", "p2_adj", "p2_sim",
  "p3_org","p3_adj","p3_sim")

source("chaid_temp.R")

set.seed(290875)

for (j in 1:n_total) {
    ## Daten aus Simulationsmodell ziehen
    df <- as.data.frame(matrix(0, nrow = n_dat, ncol = 3))

    ## x1 ist ein geordneter Faktor
    df[1] <- sample(gl(8,n_dat,ordered=TRUE))[1:n_dat]
    ## x2 ist ein ungeordneter Faktor
    df[2] <- sample(gl(8,n_dat))[1:n_dat]
    ## x3 ist binaer
    df[3] <- sample(gl(2, n_dat/2))

    names(df) <- c("x1","x2","x3")

    df$y <- as.factor(rbinom(n_dat/2, 1, prob = 0.5))

    ## p-Wert berechnen
    p1 <- chaid_temp(y~x1, data = df)
    p2 <- chaid_temp(y~x2, data = df)
    p3 <- chaid_temp(y~x3, data = df)

    ## original p-Werte
    p_all[j,1] <- p1[2]
    p_all[j,4] <- p2[2]
    p_all[j,7] <- p3[2]

    ## adjustierte p-Werte
```

```
    p_all[j,2] <- p1[1]
    p_all[j,5] <- p2[1]
    p_all[j,8] <- p3[1]

    ## y festhalten und x permutieren

    p1_neu <- vector(mode = "numeric", length = Nsim)
    p2_neu <- vector(mode = "numeric", length = Nsim)
    p3_neu <- vector(mode = "numeric", length = Nsim)

    for (i in 1:Nsim){
        x1temp <- sample(df$x1)
        x2temp <- sample(df$x2)
        x3temp <- sample(df$x3)

        ## mittels CHAID p-Werte berechnen
        p1_neu[i] <- chaid_temp(y~x1temp, data = df)[2]
        p2_neu[i] <- chaid_temp(y~x2temp, data = df)[2]
        p3_neu[i] <- chaid_temp(y~x3temp, data = df)[2]
    }

    ## Vergleich mit p_org
    p_all[j,3] <- sum(p1_neu < p1[2])/Nsim
    p_all[j,6] <- sum(p2_neu < p2[2])/Nsim
    p_all[j,9] <- sum(p3_neu < p3[2])/Nsim
}

p_all
```

Anhang B

R-Code – Vergleich der Vorhersagegüte

Der verwendete R-Code zum Vergleich der Vorhersagegüte von CHAID, CTREE und RPART in Kapitel 4 orientiert sich in weiten Teilen an dem Code, der den Benchmark-Experimenten bei Hothorn et al. (2006) zugrunde lag.

Teil 1 (Datenaufbereitung):
Am Beispiel des Datensatzes `Diabetes` wird das Vorgehen zur Kategorisierung stetiger Kovariablen gezeigt. Die stetigen Kovariablen wurden in allen Datensätzen in geordnete Faktoren umgewandelt.

```
## Diabetes

library("mlbench")
data(PimaIndiansDiabetes)
Diabetes <- PimaIndiansDiabetes
names(Diabetes)[9] <- "y" ## Response

## alle Einflussgroessen kategorisieren
for(i in 1:8) {
    Diabetes[[i]] <- cut(Diabetes[[i]], breaks = unique(
                                quantile(Diabetes[[i]],
                                         probs = seq(0,1,0.1))),
                         ordered_result = TRUE,
                         include.lowest = TRUE)
}
```

```
save(Diabetes, file = "data/Diabetes.rda")
```

Teil 2 (cmp.R):
Der folgende R-Code stellt die Hauptfunktion für die Berechnung und den Vergleich der Vorhersagefehler bereit.

```
## cmp.R

library("party")
library("rpart")
library("CHAID")

B <- 500
set.seed(290875)

plrpbench <- function(dat, B = 10) {

    dat <- dat[complete.cases(dat),]
    ## complete cases, da CHAID Beobachtungen
    ## mit NAs sowieso entfernt

    n <- nrow(dat)

    bs = rmultinom(B, n, rep.int(1, n)/n) ## Bootstrap

    perf <- matrix(0, ncol = 3, nrow = B)

    ptree <- vector(length = B, mode = "list")
    rtree <- vector(length = B, mode = "list")
    chtree <- vector(length = B, mode = "list")

    for (i in 1:B) {
        print(i)
        oob = (bs[,i] == 0)

        ## CTREE
        pt <- party::ctree(y ~ ., data = dat, weight = bs[,i])
        pr <- predict(pt, newdata = dat)
        ptree[[i]] <- pt
        perf[i,2] <- mean((pr != dat$y)[oob])
```

```
        ## RPART
        tree = rpart(y ~ ., data = dat, weights = bs[,i])
        rtree[[i]] <- tree
        perf[i,3]  <- mean(predict(prune(tree,
                              cp = tree$cptable[which.min(
                                     tree$cptable[,4]),1]),
                              newdata = dat[oob,], type = "class")
                            != dat$y[oob])

        ## CHAID
        cht <- chaid(y ~ ., data = dat, weights = bs[,i])
        chtree[[i]] <- cht
        perf[i,1] <- mean(predict(cht, newdata = dat[oob,])
                          != dat$y[oob])

        cat(perf[i,1], " ", perf[i,2], " ", perf[i,3], "\n");
    }
    list(perf = perf, bs = bs)
}
```

Teil 3:
Der folgende R-Code zeigt am Beispiel des Datensatzes `Diabetes` den Aufruf der Berechnung.

```
source("cmp.R")
load("data/Diabetes.rda")
perf <- plrpbench(Diabetes, B = B)
apply(perf$perf, 2, summary)
perf$name = "Diabetes"
save(perf, file = "perf/perfDiabetes.rda")
```

Teil 4:
Dieser Teil beinhaltet den benötigten R-Code zur Erstellung der Boxplots und der Kruskal-Wallis Rangsummen-Tests zum Vergleich der Vorhersagefehler für CHAID, CTREE und RPART.

```
sets <- c("BreastCancer", "Diabetes", "Glass", "Glaucoma",
          "Ionosphere", "Sonar", "Soybean", "Vehicle")

# Boxplots
```

```
for (m in sets) {
load(paste("perf/perf", m, ".rda", sep = ""))
pdf(file = paste("pdf/bp_", m, ".pdf", sep = ""))
boxplot(list(perf$perf[,1],perf$perf[,2],perf$perf[,3]),
            names=c("CHAID","CTREE","RPART"), main = paste(m))
dev.off()
}

# KW-Test

total <- data.frame()

for (m in sets) {
    load(paste("perf/perf", m, ".rda", sep = ""))

    ch <- data.frame(rep(paste(m), length(perf$perf[,1])),
                     rep("chaid", length(perf$perf[,1])),
                     perf$perf[,1])
    ct <- data.frame(rep(paste(m), length(perf$perf[,2])),
                     rep("ctree", length(perf$perf[,2])),
                     perf$perf[,2])
    rp <- data.frame(rep(paste(m), length(perf$perf[,3])),
                     rep("rpart", length(perf$perf[,3])),
                     perf$perf[,3])
    names(ch) <- c("dataset","algorithm","error")
    names(ct) <- c("dataset","algorithm","error")
    names(rp) <- c("dataset","algorithm","error")

    verf <- rbind(ch,ct,rp)

    ## Gesamtdatensatz zusammensetzen
    total <- rbind(total,verf)
}

kwresult <- data.frame()

for (m in sets) {
    kwt <- kruskal.test(total$error[total$dataset == paste(m)]
                ~ total$algorithm[total$dataset == paste(m)])
```

```
    ## Ergebnisse in Tabelle schreiben
    kwres <- data.frame(paste(m), kwt$statistic, kwt$p.value)
    names(kwres) <- c("dataset", "statistic", "p.value" )
    kwresult <- rbind(kwresult, kwres)
}

library("xtable")
xtable(kwresult)
```

Anhang C

Beiliegende CD-ROM

Die beiliegende CD-ROM enthält den gesamten verwendeten R-Code für die vorliegende Arbeit (Stand: 03. Juli 2009).

Inhalt:

- *Simulation:* Code für die Simulationen aus Kapitel 3
- *Vorhersage:* Code für den Vergleich der Vorhersagefehler in Kapitel 4

Literaturverzeichnis

L. Breiman, J. Friedman, C. J. Stone, und R. A. Olshen. *Classification and Regression Trees*. Chapman & Hall/CRC, 1984.

L. Fahrmeir, A. Hamerle, und G. Tutz. *Multivariate statistische Verfahren*. Berlin, New York, de Gruyter, 1996.

E. C. Fieller. The biological standardization of insulin. *Journal of the Royal Statistical Society*, 7:1–64, 1940. Supplement.

L. A. Goodman. On simultaneous confidence intervals for multinomial proportions. *Technometrics*, 7(2):247–254, 1965.

T. Hastie, R. Tibshirani, und J. Friedman. *The Elements of Statistical Learning, Second Edition: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer New York, 2 edition, 2008.

T. Hothorn und A. Zeileis. *partykit: A Toolkit for Recursive Partytioning*, 2009. R Paket Version 0.0-1.

T. Hothorn, F. Leisch, A. Zeileis, und K. Hornik. The design and analysis of benchmark experiments. *Journal of Computational and Graphical Statistics*, 14(3):675–699, 2005.

T. Hothorn, K. Hornik, und A. Zeileis. Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical Statistics*, 15(3):651–674, 2006.

G. V. Kass. An exploratory technique for investigating large quantities of categorial data. *Applied Statistics*, 29(2):119–127, 1980.

F. Leisch und E. Dimitriadou. *mlbench: Machine Learning Benchmark Problems*, 2009. R Paket Version 1.1-6.

J. N. Morgan und J. A. Sonquist. Problems in the analysis of survey data, and a proposal. *Journal of the American Statistical Association*, 58:415–434, 1963.

D. J. Newman, S. Hettich, C. L. Blake, und C. J. Merz. UCI Repository of machine learning databases, 1998. Online erhältlich unter `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

A. Peters, T. Hothorn, und B. Lausen. ipred: Improved predictors. *R News*, 2(2):33–36, June 2002. URL `http://CRAN.R-project.org/doc/Rnews/`.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. URL `http://www.R-project.org`.

H. Strasser und C. Weber. On the asymptotic theory of permutation statistics. *Mathematical Methods of Statistics*, 8:220–250, 1999.

The FoRt Student Project Team. *CHAID: CHi-squared Automated Interaction Detection*, 2009. R Paket Version 0.1-0.

T. M. Therneau und E. J. Atkinson. An introduction to recursive partitioning using the rpart routines. *Technical Report 61, Section of Biostatistics, Mayo Clinic, Rochester*, 1997. Online erhältlich unter `http://www.mayo.edu/hsr/techrpt/61.pdf`.

Eigenständigkeitserklärung

Hiermit erkläre ich, dass es sich bei der vorliegenden Bachelor Thesis um eine selbständig verfasste Arbeit handelt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt wurden.

München, den 03. Juli 2009

Stephanie Thiemichen