Die approbierte Originalversion dieser Diplom-/Masterarbeit ist an der Hauptbibliothek der Technischen Universität Wien aufgestellt (http://www.ub.tuwien.ac.at).

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology (http://www.ub.tuwien.ac.at/englweb/).

TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

Masterarbeit

zum Thema

Heuristiken zur Variablenselektion bei hochdimensionalen Daten

ausgeführt am
Institut für Statistik und Wahrscheinlichkeitstheorie
der Technischen Universität Wien

unter der Anleitung von
Ao. Univ. Prof. Dipl.-Ing. Dr. techn. Peter Filzmoser

durch
Leonhard Seyfang
Matr. Nr. 0227022
Heidestraße 6, 1140 Wien

Wien, am 18. Februar 2008

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Wien, am 18. Februar 2008

Danksagung

Ich möchte mich an dieser Stelle bei jenen Personen bedanken, die mich bei der Erabeitung dieser Masterarbeit und allgemein beim Studium unterstützt haben.

Mein Dank gilt Herrn Professor Dr. Peter Filzmoser für die persönliche Betreuung meiner Arbeit und für die zahlreichen Anregungen.

Außerdem bedanke ich mich bei meiner Familie und meinen Freunden für die mannigfaltige und nachhaltige Unterstützung.

Abstract

This work compares several subset selection techniques for large datasets. Simple, fast, as well as more complex and combined heuristics are considered. For all described methods and some subroutines the R-code is listed.

The different methods lead to very varying models regarding the involved variables. However, in terms of the optimised information criterion and the simulated prediction error the models are quite similar.

The robustness in relation to the used data portion was evaluated for several algorithms. It was found, that the results tend to depend on the used data portion rather than on the used selection technique.

Furthermore the generation of artificial datasets is discussed and a method for this purpose is introduced.

Subroutines, among other tools for downsizing datasets, for cross-validation and the standard method 'stepwise', are given.

Zusammenfassung

In dieser Arbeit werden einige Heuristiken zur Variablenselektion bei großen multivariaten Datensätzen verglichen. Dabei werden sowohl einfache, schnelle als auch aufwändigere und kombinierte Methoden betrachtet. Für alle behandelten Methoden und einige Hilfsfunktionen ist der R-Code angegeben.

Die verschiedenen Heuristiken führen zu sehr unterschiedlichen Modellen im Hinblick auf die beteiligten Variablen. Bezüglich des optimierten Gütekriteriums und des simulierten Prognosefehlers unterscheiden sich die Resultate aber nur wenig.

Einige Methoden wurden bezüglich ihrer Robustheit auf Veränderung der Trainingsdaten untersucht. Dabei wurde festgestellt, dass die resultierenden Modelle stärker von dem verwendeten Teil-Datensatz als von der verwendeten Methode abhängen.

Weiters wird die Erzeugung künstlicher Datensätze diskutiert und eine Funktion zu diesem Zweck vorgestellt.

Hilfsmethoden, u. a. zur Reduktion von großen Datensätzen, für die Kreuzvalidierung und für die Referenzmethode Stepwise sind ebenfalls angegeben.

Inhaltsverzeichnis

Inhaltsverzeichnis **4**

1 Einleitung **6**
1.1 Notation . . . 6
1.2 Einführung und Abgrenzung . . . 7
1.3 Gütekriterien . . . 10
1.3.1 R^2 . . . 11
1.3.2 Adjustiertes R^2 . . . 11
1.3.3 AIC . . . 11
1.3.4 BIC . . . 12
1.3.5 Prognosefehler . . . 12
1.4 Verwendete Datensätze . . . 16
1.5 Referenzmethode *Stepwise* . . . 20
1.5.1 Lokale Optima bei *Stepwise* . . . 21

2 Erzeugen künstlicher Datensätze **24**
2.1 Motivation . . . 24
2.2 Umsetzung . . . 24
2.2.1 Eingabeparameter . . . 25
2.2.2 Rückgabewerte . . . 25
2.2.3 R-Code . . . 25
2.3 Anwendungsbeispiel . . . 26

3 Heuristiken **30**
3.1 Einfache Heuristiken . . . 30
3.1.1 Schrittweises Hinzunehmen nach vorher bestimmter Reihenfolge . . . 31
3.1.2 ResCor: Hinzunehmen nach max. Korrelation mit den Residuen . . . 32
3.2 Methoden zum Austauschen einzelner Variablen . . . 34
3.3 STAP kombiniert Stepwise und Swap . . . 38
3.4 RASAR: Vorwärts-, Rückwärts- und Tauschschritte . . . 41
3.4.1 Untersuchung der Ähnlichkeit verschiedener Modelle . . . 46
3.4.2 RASAR in Kombination mit anderen Methoden . . . 48
3.5 FASS . . . 51

4 Vergleich der Ergebnisse **55**
4.1 Stabilität der Variablenauswahl . . . 55
4.1.1 Interpretation der Stabilität . . . 56
4.2 BIC und Prognosefehler . . . 62
4.2.1 Interpretation BIC und Prognosefehler . . . 63

5 Hilfsmethoden **66**
5.1 Generieren der Formelnotation mit `form` 66
5.2 Schrittweise Variablenauswahl mit `stepwise` 66
5.3 Kreuzvalidierung mit `cv` 68
5.4 Wiederholte Kreuzvalidierung mit `rep.cv` 71
5.5 Prognosefehler mit oder ohne Kreuzvalidierung 71
5.6 Leave-One-Out-Kreuzvalidierung mit `loo` 74
5.7 Entfernen von Variablen mit `rmAlmostConst` 75
5.8 Entfernen linear abhängiger Variablen mit `rmLinDep` 76

Abbildungsverzeichnis **77**

Tabellenverzeichnis **78**

Listingsverzeichnis **78**

Literaturverzeichnis **79**

1 Einleitung

1.1 Notation

CV Kreuzvalidierung (*Cross Validation*)

H *Hamming-Distanz*: Maß für die Unterschiedlichkeit gleich langer Zeichenketten. Fasst man ein Modell als 0-1-Vektor der Länge p auf, kann die H als Maß für die Verschiedenheit von zwei Modellen (A und B) verwendet werden. Sie gibt die Anzahl von *unterschiedlichen* Einträgen in binären Vektoren an, wobei die Vektoren gleiche Länge haben müssen.

n Anzahl der *Beobachtungen* eines Datensatzes.

NA Fehlender Wert (*Not Available*)

O *Overlap*: Maß für die Übereinstimmung binärer Vektoren (siehe Kap. 4, Seite 55).

p Anzahl der *erklärenden* (Prädiktor-) *Variablen* eines Datensatzes.

$\dot{p}$ Anzahl der erklärenden Variablen eines konkreten Modells. Es gilt: $0 \leq \dot{p} \leq p$. Da die Modelle in der Regel auch noch einen Intercept haben, ist die Zahl der zu schätzenden Parameter (bzw. die Zahl der verbrauchten Freiheitsgrade) $\dot{p} + 1$.

R ist eine Sprache und Programmierumgebung für die Analyse und Darstellung von Daten. Für eine Einführung siehe: [R Development Core Team, 2007]. R ist Open Source Software und kann unter `http://www.R-project.org` bezogen werden.

X Matrix der erklärenden Variablen. X hat die Dimensionen $n \times p$.

y *Zielvariable* (Response). y ist ein Vektor der Länge n.

$\ll$ „viel kleiner als“

$\gg$ „viel größer als“

R-Befehle oder -Objekte, z. B. Variablen- oder Funktionsnamen werden in `Schreibmaschinenschrift` wiedergegeben. *Funktionsnamen* werden manchmal zur leichteren Erkennbarkeit mit (eventuell leeren) Klammern dargestellt. Mit *Stepwise* ist z. B. der Algorithmus (also das geistige Konzept) gemeint, während `stepwise()` eine konkrete R-Funktion ist.

Die Wörter *Funktion* und *Methode* haben in dieser Arbeit die gleiche Bedeutung und bezeichnen ein (Unter-) Programm. Funktionen werden in der Regel mit Eingabewerten (Parameter, Argumente) aufgerufen und können ein beliebiges R-Objekt (häufig ist das eine Liste) als Rückgabewert retournieren.

In längeren Code-Abschnitten (Listings) sind Schlüsselwörter wie z. B. `plot()`, `#Kommentare` und `"Zeichenketten"` zur leichten Lesbarkeit farbig hervorgehoben.

Oft wird im Zusammenhang mit den verschiedenen Heuristiken die *Rechenzeit* (auch einfach als *Zeit* bezeichnet) angegeben. Diese ist natürlich von der verwendeten Hardware abhängig. Wesentlich ist aber der relative Vergleich der Methoden, nicht die absolute Zeit. (Verwendet wurde ein AMD Athlon XP 1600+ Prozessor mit 1,40 GHz.)

1.2 Einführung und Abgrenzung

Im Folgenden wird die Ausgangslage und Problemstellung kurz umrissen und abgegrenzt.

Gegeben sei ein Datensatz mit einer Zielvariable y mit n Beobachtungen und mehreren erklärenden Variablen, die in einer Matrix X zusammengefasst werden. X hat n Zeilen (Beobachtungen) und p Spalten (Variablen) die als $x_i \quad i = 1, \ldots, p$ bezeichnet werden. Diese Arbeit befasst sich mit großen Datensätzen d.h. $p > 100$. Zwischen y und X wird ein linearer Zusammenhang angenommen:

$$y = \alpha + X\beta + \varepsilon \qquad \text{mit} \qquad \varepsilon \sim N(0, \sigma^2 I)$$

mit dem Vektor 0 bestehend aus n Nullen und der n-dimensionalen Einheitsmatrix I. σ^2 gibt die Varianz der Residuen an.

Etwaige zu untersuchende Kreuzeffekte sind als eigene Variablen in X enthalten. Kategorielle Einflussgrößen können mit Dummy-Variablen codiert werden. Linear abhängige Variablen sind zu entfernen (siehe auch Kapitel 5.8).

Gesucht ist eine *Teilmenge* der p erklärenden Variablen, die sich für die Modellbildung eignet.

Von den vielen verschiedenen Motiven für Variablenselektion (bzw. überhaupt für Modellbildung) sind zwei Hauptaspekte zu nennen:

- *Interpretation*: Man möchte aus den Daten lernen, nämlich vor allem über die Art und Stärke des Zusammenhangs von y und X.
- *Prognose*: Man möchte Modelle für eine möglichst gute Prognose finden.

Leider werden diese beiden Intentionen mit unterschiedlichen Methoden optimal bedient.

Ein paar typische Probleme der Thematik sollen kurz skizziert werden:

- Wenn man keine Präferenzen für eine bestimmte Modellgröße hat, gibt es 2^p mögliche Modelle. Ein naheliegender Ansatz wäre, alle Modelle auszuprobieren, nach einem geeigneten Kriterium zu bewerten, und dann das beste auszuwählen. Für kleine p ist das tatsächlich eine gute Lösung, die z. B. in R im Paket `leaps` [Thomas Lumley und Alan Miller, 2007] realisiert ist. Für große p verlängert sich die Rechenzeit exponentiell: Bei $p = 400$ und einer geschätzten Rechenzeit von einer Sekunde pro Modell ergibt sich z. B. eine Gesamtdauer von ca. $8 \cdot 10^{112}$ Jahren. Aus diesem Grund wird dieser Ansatz hier nicht weiter verfolgt – was allerdings schade ist, denn gerade zur Beurteilung der Heuristiken wäre es sehr nützlich, das beste Modell zu kennen. (Tatsächlich müsste man nicht alle 2^p Modelle berechnen: Mit einem *Branch-and-Bound*-Algorithmus werden Gruppen von Lösungen ausgeschieden, die offensichtlich das Gütekriterium nicht optimieren können. Leider bleiben bei großen p noch immer viel zu viele Möglichkeiten übrig.)
- Wenn schon nicht jedes Modell berechnet werden kann, so ist doch die Idee verlockend, zumindest zu verhindern, dass die verwendete Heuristik das gleiche Modell

unnötigerweise öfters berechnet. Zu diesem Zweck bräuchte man einen logischen Vektor der Länge 2^p. Wenn die Information über jedes Modell (gerechnet, oder noch nicht gerechnet) in genau einem Bit (kleinste mögliche Speichereinheit) gespeichert werden kann, werden dafür immer noch $\sim 9 \cdot 10^{108}$ GB benötigt[1], was trotz rasantem Fortschritt in der Computer-Technologie dem gewöhnlichen Anwender noch längere Zeit nicht zur Verfügung stehen wird.

- Wenn die Modellgröße $\dot{p}$ fest vorgegeben wird (warum auch immer), bleiben noch immer $\binom{p}{\dot{p}}$ z. B. $\binom{400}{30} \approx 1,4 \cdot 10^{45}$ verschiedene Möglichkeiten. Als Trost ist allerdings die Wahl des Gütekriteriums bei Modellen derselben Größe unkritisch.

- Im Allgemeinen gibt es viele gleich oder ähnlich gute Modelle (im Sinne von Kapitel 1.3), die sich bezüglich der beteiligten Variablen stark unterscheiden (siehe Kapitel 3.4.1). Das ist vor allem unangenehm, wenn es um die *Interpretation* des Modells geht.

- Die für die Variablenauswahl verwendeten Heuristiken besitzen keine *Gütegarantie*. Von einer Heuristik mit Gütegarantie spricht man, wenn angegeben werden kann, um wie viel die gefundene Lösung im ungünstigsten Fall schlechter ist als die tatsächlich beste Lösung.

- Oft ist $n < p$. Viele Verfahren, insbesonders solche, bei denen das *volle Modell* geschätzt werden muss, sind dann nicht anwendbar. Überhaupt ist eine zuverlässige Schätzung der $\dot{p} + 1$ Parameter nur möglich, wenn $(\dot{p} + 1) \ll n$.

- Heuristiken neigen dazu, in einem lokalen Optimum *hängen zu bleiben*, d.h. bessere Lösungen werden nicht gefunden, wenn sie von der aktuellen Lösung zu weit entfernt (zu verschieden) sind (siehe z. B. Kapitel 1.5.1).

- *Fehlende Werte* in den Daten sind unbedingt zu vermeiden, weil sonst die Methoden zum Vergleichen verschiedener Modelle nicht mehr korrekt funktionieren würden.

- Im Falle von hoch korrelierten Variablen kommt es zu *instabilen* Schätzungen von β: minimale Änderungen in den Daten können zu großen Änderungen bei den geschätzten Koeffizienten führen. Als Indikator für Multikollinearität dient der VIF (*variance-inflation factor*): $VIF_i = (1 - R_i^2)^{-1}$ wobei R_i^2 das Bestimmtheitsmaß der Regression von x_i auf X_{-i} ist. In R steht der Befehl `vif()` aus dem Paket `car` [Fox, 2007] zur Identifizierung hoch korrelierter Variablen zur Verfügung.

Die meisten hier behandelten Methoden zur Modellbildung können als *schrittweise* Verfahren aufgefasst werden. Derartige Verfahren können vollständig beschrieben werden, indem die folgenden Punkte spezifiziert werden:

[1] 1 Byte $= 2^8$ Bit; 1 GB $= 2^{30}$ Byte $= 2^{38}$ Bit; $2^{400}/2^{38} = 2^{362} \approx 9,39 \cdot 10^{108}$

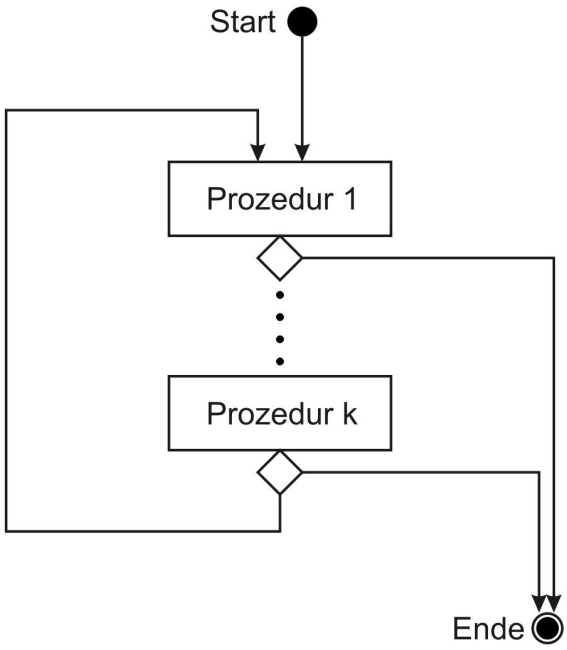

Abbildung 1: Schrittweise Verfahren können allgemein als wiederholte Abfolge von einer oder mehreren Prozeduren aufgefasst werden. Die Rauten in der Grafik symbolisieren die Überprüfung der Abbruchbedingung(en).

- Welches Gütekriterium wird verwendet? Es können in verschiedenen Phasen unterschiedliche Kriterien verwendet werden. Zum Beispiel ist es denkbar, für sehr häufig durchzuführende Berechnungen schnelle Kriterien zu verwenden, und an anderen Stellen rechenintensivere, genauere. Dies ist aber nur dann sinnvoll, wenn die Ersteren eine gute Abschätzung der Zweiteren darstellen.
- Welches Startmodell wird verwendet? In Frage kommen z. B.: das leere Modell, das volle Modell, zufällig gewählte Modelle und Modelle, die mit einer anderen (schnelleren) Heuristik gefunden wurden.
- Was sind die Abbruchbedingungen? Beispiele für Abbruchbedingungen sind:
  - Es wird (in einer bestimmten Zeit oder Schrittanzahl) keine Verbesserung mehr gefunden.
  - Die gefundene Verbesserung unterschreitet eine gegebene Schranke.
  - Eine bestimmte maximale Rechenzeit wird überschritten.
  - Eine bestimmte Anzahl an Durchläufen wird erreicht.
- Die Definition der eigentlichen Schritte, wobei jeder Schritt auch aus mehreren Teilschritten (Prozeduren) bestehen kann (siehe Abb. 1). Dies impliziert auch die *Schrittweite* d. h. die Umgebung im Lösungsraum, die in einem Schritt erreichbar ist. Große Schritte verringern die Gefahr, in einem lokalen Optimum hängen zu

bleiben. Gleichzeitig vergrößert sich aber auch die Anzahl der Möglichkeiten und somit die Rechenzeit.

1.3 Gütekriterien

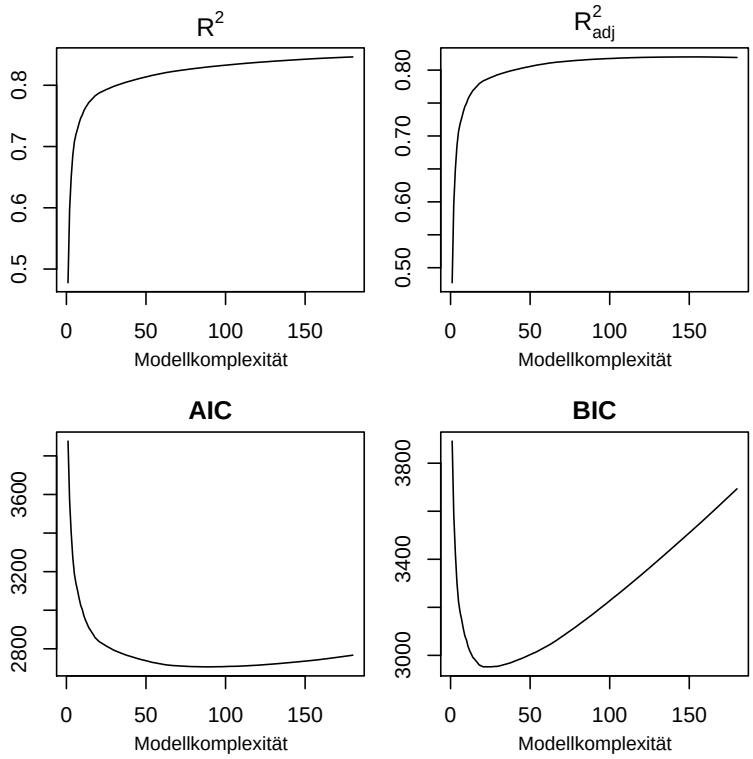

Abbildung 2: Ein paar Kriterien im Überblick: R^2 nimmt mit steigender Modellkomplexität stets zu, R^2_{adj} kann dagegen auch wieder fallen. AIC bestraft die Modellgröße stärker als R^2_{adj} und BIC nochmals deutlich stärker als AIC. Die absolute Höhe von AIC und BIC ist nicht zu interpretieren, sie dient nur dem Vergleich von verschiedenen, mit denselben Beobachtungen geschätzten, Modellen.

Um verschiedene Modelle vergleichen zu können, muss die Qualität eines Modells gemessen werden. Zu diesem Zweck dienen die verschiedenen Gütekriterien (*Informationskriterien*), die aber ihrerseits auch wieder nach verschiedenen Gesichtspunkten kritisch zu beurteilen sind. Die unterschiedlichen Gütemaße führen nämlich zu durchaus stark abweichenden Ergebnissen, und es hängt vom Zweck der Variablenselektion ab, welche Kriterien günstig sind.

Die meisten in dieser Arbeit besprochenen Algorithmen können ohne weiteres mit verschiedenen Gütekriterien arbeiten, wobei sich allerdings für rechenintensive Kriterien zeitlich bedingte Einschränkungen ergeben. Der Schwerpunkt der Arbeit wird sich mit den verschiedenen Heuristiken, nicht mit den konkret verwendeten Kriterien befassen.

Wenn Modelle als *gut* oder *schlecht* bezeichnet werden, so ist das immer hinsichtlich des jeweils verwendeten Kriteriums zu verstehen.

Einige Gütekriterien, die in dieser Arbeit verwendet werden, sind nachfolgend kurz beschrieben.

1.3.1 R²

Das *Bestimmtheitsmaß* R^2 ist definiert als

$$R^2 = \frac{ESS}{TSS} = \frac{Var(\hat{y})}{Var(y)} = \frac{\sum(\hat{y}_i - \bar{y})^2}{\sum(y_i - \bar{y})^2} \tag{1}$$

wobei ESS für *Explained Sum of Squares* und TSS für *Total Sum of Squares* steht. Falls $\bar{y} = 0$ oder das Modell einen Intercept-Term enthält, gilt

$$R^2 \in [0, 1]$$

Vorteil dieses Maßes ist neben der Normierung vor allem die anschauliche Interpretation als *Anteil der erklärten Varianz an der Gesamtvarianz*. Da R^2 mit zunehmender Modellgröße immer zunimmt, eignet es sich nicht zum Vergleich unterschiedlich großer Modelle (es würde immer das größere Modell gewählt werden).

1.3.2 Adjustiertes R²

Das adjustierte R^2 ist gegeben mit:

$$R^2_{adj} = 1 - (1 - R^2)\frac{n-1}{n-\dot{p}-1} \tag{2}$$

Größere Modelle werden also „bestraft“. $R^2_{adj} \leq R^2$.

1.3.3 AIC

Das *Akaike Informationskriterium* (*Akaike's Information Criterion*) ist definiert als

$$-2L + 2(p+1) \tag{3}$$

wobei L die maximierte Log-Likelihood-Schätzung bezeichnet.

Die Zeilen 6, 7 und 8 in Listing 1 sind also äquivalent.

```
1 lmF <- lm(y ~ ., data=x)
2 rss <- deviance(lmF)
3 n   <- nrow(x)
4 p   <- ncol(x) + 1
5
6 AIC(lmF)
7 -2 * logLik(lmF) + 2*(p+1)
8 n + n * log(2 * pi) + n * log(rss/n) + 2*(p+1)
```

Listing 1: AIC

1.3.4 BIC

Das BIC (*Bayesian Information Criterion*) ist ähnlich dem AIC, wobei aber die Komplexität des Modells stärker bestraft wird, nämlich mit dem Faktor $log(n)$ statt 2. Deshalb führt die Verwendung von BIC (statt AIC) zu kleineren Modellen.

$$-2L + log(n)(p + 1) \tag{4}$$

Die Zeilen 1, 2, 3 und 4 in Listing 2 sind äquivalent.

```
1 AIC(lmF, k=log(n))
2 -2 * logLik(lmF) + log(n)*(p+1)
3 n + n * log(2 * pi) + n * log(rss/n) + log(n)*(p+1)
4 BIC(lmF)    # library(nlme)
```

Listing 2: BIC

Im Gegensatz zu R^2 und R^2_{adj} sind AIC und BIC zu *minimieren*.

1.3.5 Prognosefehler

Ein scheinbar nahe liegendes Kriterium ist der mittlere, quadrierte (ev. auch absolute) Prognosefehler. Die in dieser Arbeit verwendeten Methoden zur Berechnung des Prognosefehlers sind in den Kapiteln 5.3 bis 5.6 beschrieben.

Anders als bei den zuvor beschriebenen Kriterien ist der Prognosefehler nicht eindeutig, also vom Zufall abhängig (außer bei *Leave-One-Out*-Kreuzvalidierung). Es stellt sich daher die Frage, ob und unter welchen Bedingungen der Prognosefehler ein zuverlässiges Mittel zur Beurteilung eines Modells darstellt.

Dies ist offensichtlich nur dann der Fall, wenn ähnliche Modelle auch zu ähnlichen Werten in der Beurteilung führen. Um dies zu testen, braucht man Modelle, über deren Reihung hinsichtlich der Prognosequalität möglichst viel im Vorhinein bekannt ist. Das ist bei einer mittels Vorwärtsselektion (siehe Kapitel 1.5) erzeugten Modellsequenz der Fall: Die Variablen jedes Modells sind eine Teilmenge der Variablen aller nachfolgenden Modelle. Zwei benachbarte Modelle unterscheiden sich nur durch genau eine Variable und sollten daher in der Beurteilung der Modellgüte ähnlich abschneiden (siehe auch Abbildung 1.3). Wie man in den Abbildungen 3, 4 und 43 sieht, ist das nicht unbedingt der Fall:

Nur bei Kreuzvalidierung mit sehr häufiger Wiederholung und nicht zu großen Test-Sets entsteht ein einigermaßen glatter Verlauf der Kurve des Prognosefehlers. Die Verteilung des Prognosefehlers hängt also außer von der Modellgüte, die man eigentlich abbilden möchte, von folgenden Faktoren ab:

- davon ob Kreuzvalidierung durchgeführt wurde oder nicht
- von der Anzahl der Wiederholungen

- vom Verhältnis zwischen der Größe des Trainings-Sets und der Modellgröße $\dot{p}$ (nur wenn das Trainings-Set wesentlich größer ist als $\dot{p}$, ist eine stabile Schätzung der Parameter möglich)
- vom Zufall

Der mittlere Prognosefehler nimmt ab, wenn die Größe des Test-Sets abnimmt. Die Varianz nimmt ebenfalls mit der Größe der Test-Sets ab und außerdem mit der Anzahl der Wiederholungen.

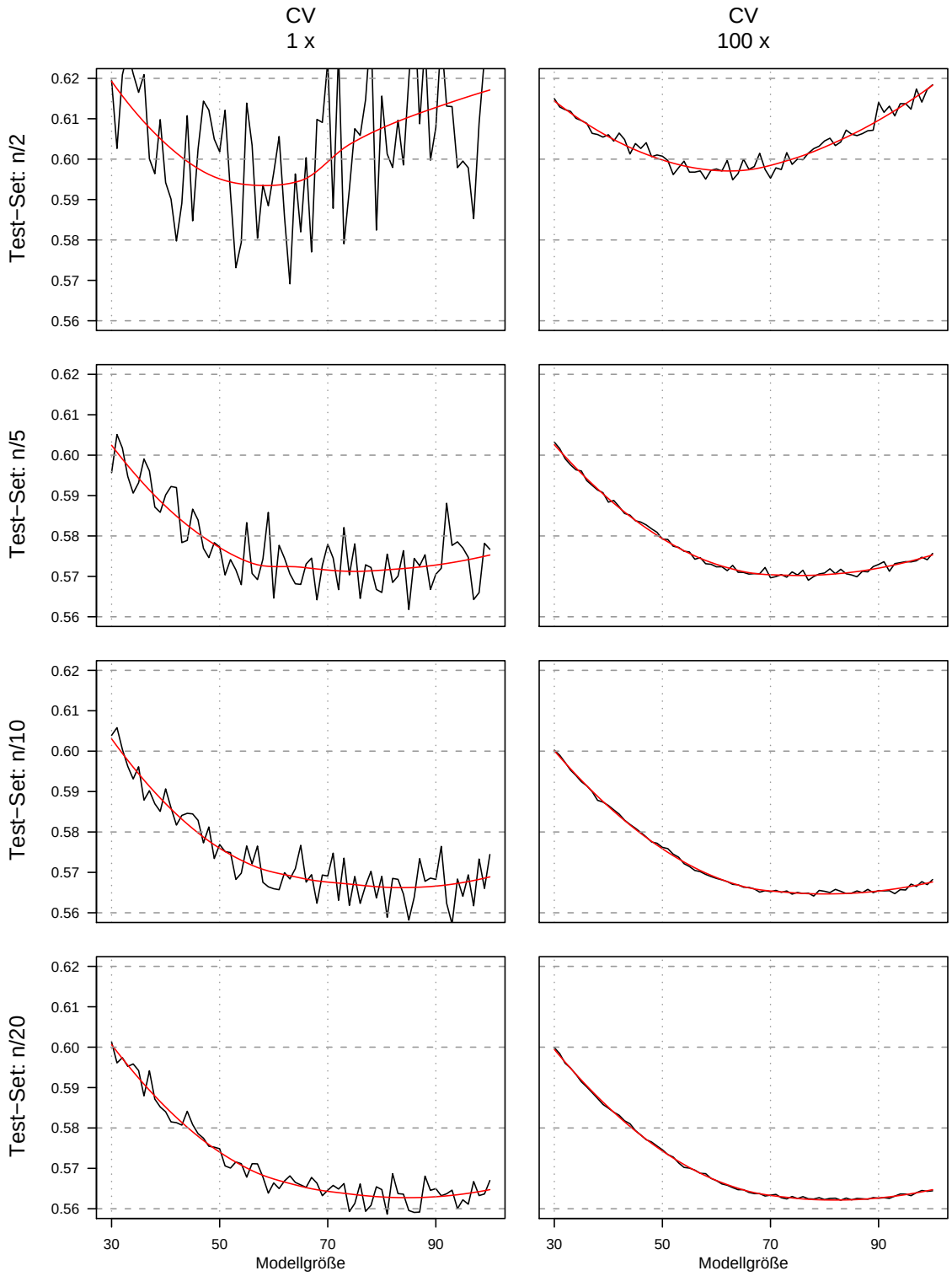

Abbildung 3: Der mittels CV berechnete Prognosefehler einer Stepwise-Modellsequenz (Der hier verwendete ART-Datensatz wird in Kapitel 1.4 beschrieben): In der linken Spalte wurde der Prognosefehler für jedes Modell mit einfacher Kreuzvalidierung berechnet, in der rechten Spalte wurde dieser Vorgang 100-mal wiederholt und gemittelt. Die Zeilen unterscheiden sich durch die Größe des Trainings- und Test-Sets: Für das Test-Set wurden $\frac{1}{2}$, $\frac{1}{5}$, $\frac{1}{10}$ und $\frac{1}{20}$ der Daten genommen. Die rote Linie ist eine (mit der Methode `loess` erzeugte) Glättung der Reihe und soll die Interpretation erleichtern.

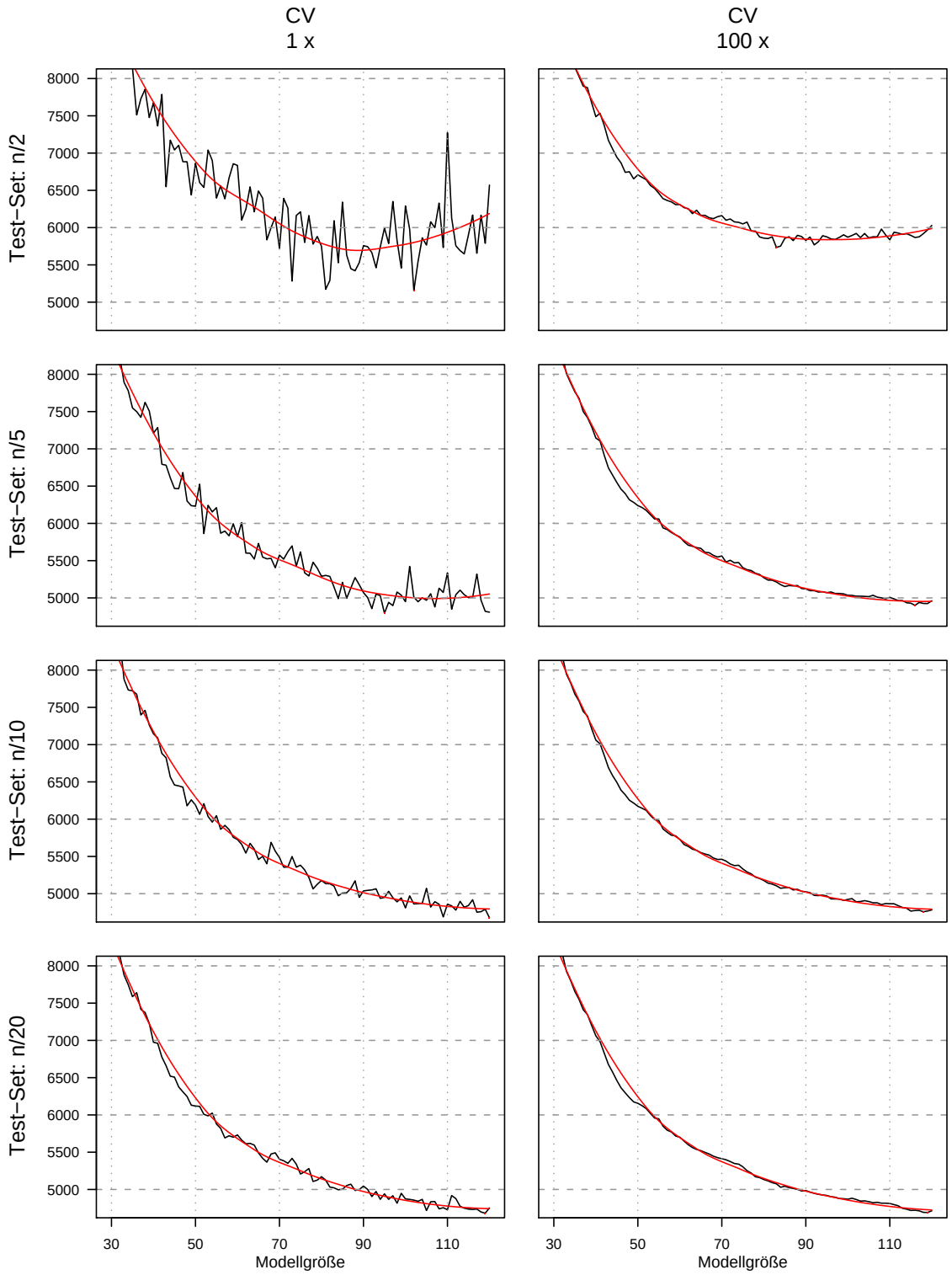

Abbildung 4: Analog zur Abb. 3 ist hier der Prognosefehler für die Stepwise-Modellsequenz mit den TOX-Daten (siehe Kapitel 1.4) dargestellt.

1.4 Verwendete Datensätze

Zur Veranschaulichung der hier besprochenen Methoden werden vor allem zwei Datensätze verwendet:

ART ist ein künstlich erzeugter Datensatz (siehe Kapitel 2) mit $n = 1200$ und $p = 400$.

In Abbildung 5 ist die Varianz der ersten 10 Hauptkomponenten dargestellt. Durch die ersten 5 Hauptkomponenten werden fast 63% der Gesamtvarianz erklärt.

Der χ^2-Plot (Abb. 6) ist ein grafisches Verfahren zur Beurteilung von multivariater Normalverteilung. In diesem Fall bilden die Punkte annähernd eine Gerade, man kann die Daten also als normalverteilt betrachten.

Der VIF (siehe Kapitel 1.2) dient als Indikator für Multikollinearität. In Abb. 7 sind die VIFs für alle 400 Variablen als Boxplot dargestellt. Alle Werte sind im unkritischen Bereich (< 10).

TOX ist ein chemometrischer Datensatz mit $n = 812$ und $p = 400$.

Abbildung 8 gibt die ersten 10 Hauptkomponenten als Barplot wieder. Durch die ersten 5 Hauptkomponenten werden knapp 39% der Varianz erklärt.

Dem χ^2-Plot (Abb. 9) ist zu entnehmen, dass die Daten nicht gut in die Theorie der Normalverteilung passen. Außerdem ist Abb. 10 zu entnehmen, dass die VIFs durchwegs große Werte annehmen. Es bestehen also sehr hohe Korrelationen zwischen den verschiedenen Variablen.

Zusammenfassend kann man sagen, dass ART und TOX bezüglich der Anzahl der Variablen gleich groß sind, ART hat aber um die Hälfte mehr Beobachtungen und erfüllt die theoretischen Voraussetzungen (Unkorreliertheit der erklärenden Variablen, Normalverteilung der Fehler) besser.

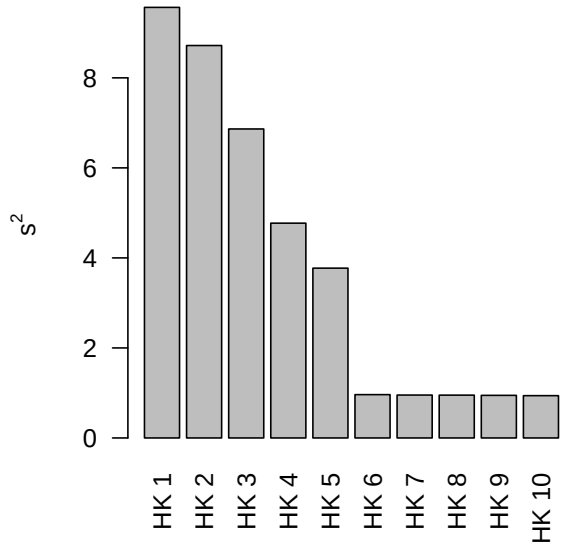

Abbildung 5: s^2 der ersten 10 Hauptkomponenten der ART-Daten

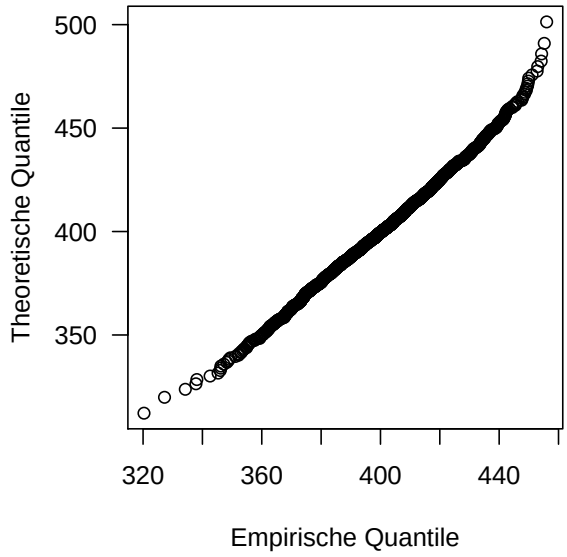

Abbildung 6: χ^2-Plot der ART-Daten zur Beurteilung der multivariaten Normalverteilung

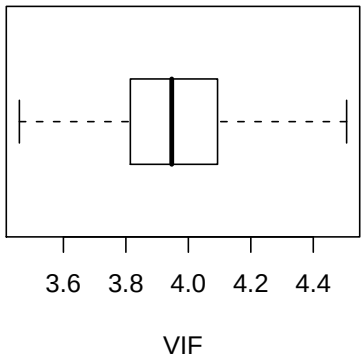

Abbildung 7: Verteilung der VIFs der ART-Daten

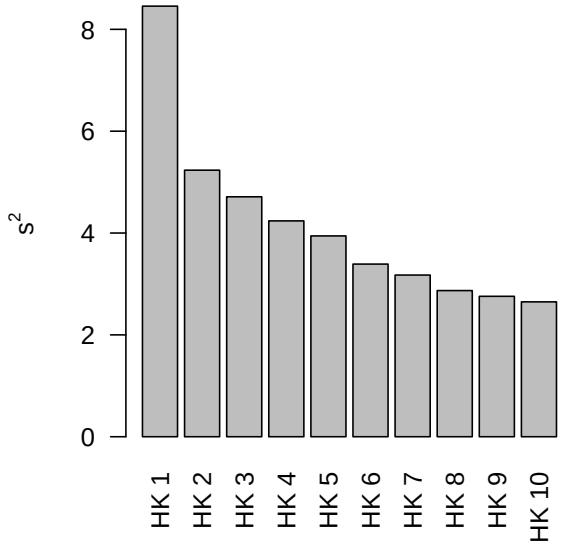

Abbildung 8: s^2 der ersten 10 Hauptkomponenten der TOX-Daten

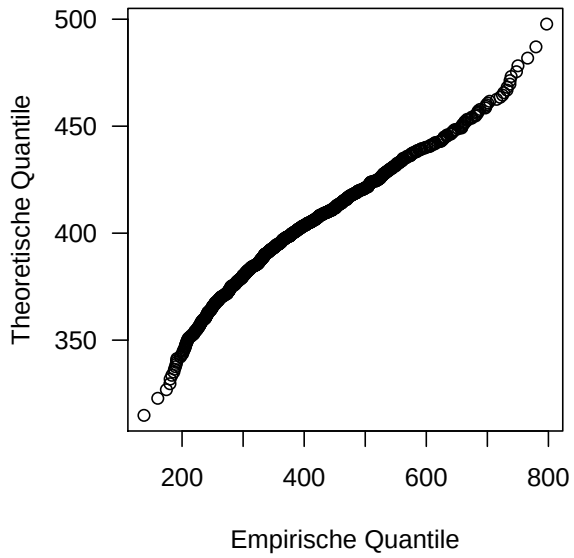

Abbildung 9: χ^2-Plot der TOX-Daten zur Beurteilung der multivariaten Normalverteilung

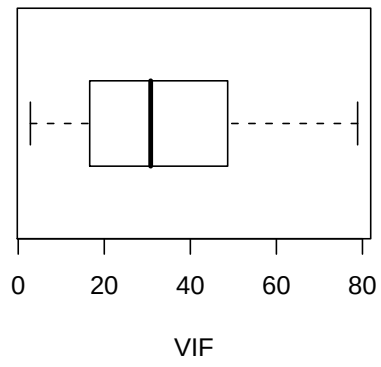

Abbildung 10: Verteilung der VIFs der TOX-Daten

1.5 *Stepwise* als Referenzmethode

Die *schrittweise Variablenauswahl* (*stepwise selection*) ist eine einfache, häufig benutzte Heuristik zur Variablenauswahl. Es existieren im Wesentlichen zwei Varianten:

Vorwärtsselektion (*step forward*) Beginnend mit dem leeren Modell wird in jedem Schritt genau die Variable dazugenommen, die nach dem verwendeten Gütekriterium zur größten Verbesserung führt. Wenn durch das Hinzunehmen einer Variable keine bzw. keine wesentliche Verbesserung bezüglich des verwendeten Kriteriums mehr erreicht werden kann oder bereits alle Variablen im Modell sind, ist der Algorithmus beendet.

Rückwärtselimination (*step backward*) Beginnend mit dem vollen Modell wird in jedem Schritt genau eine Variable entfernt, so dass, analog zur Vorwärtsselektion, das Gütekriterium optimiert wird. Der Algorithmus terminiert, wenn keine bzw. keine wesentliche Verbesserung bezüglich des verwendeten Kriteriums mehr gefunden wird oder alle Variablen entfernt wurden.

Beide Möglichkeiten führen zu einem eindeutigen Ergebnis. Eine einfache Erweiterung besteht darin, in jedem Schritt *beide* Richtungen zuzulassen, also *entweder* eine Variable zu entfernen *oder* dazuzunehmen. Gelegentlich wird empfohlen, mit dem vollen Modell

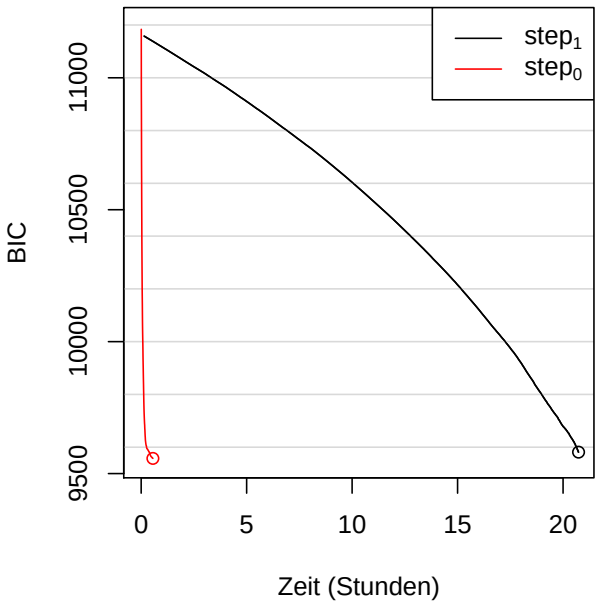

Abbildung 11: Vergleich der Stepwise-Methode mit dem leeren (step_0) und dem vollen (step_1) Modell als Startmodell (TOX-Daten).

zu beginnen. Diese Taktik hat bei den ART-Daten bei 300-facher Rechenzeit eine kleine Verbesserung gebracht. Bei den TOX-Daten ist das Ergebnis bei 37-facher Rechenzeit sogar schlechter als beim leeren Startmodell (Abb. 11). Interessant ist, dass die Modelle in beiden Fällen größer ausfallen, wenn mit dem vollen Modell begonnen wird.

Daten	Startmodell	$\dot{p}$	BIC	Rechenzeit (sek)
ART	Leeres Modell	22	2 948,757	364
	Volles Modell	26	2 939,145	115 321
TOX	Leeres Modell	69	9 557,573	2 002
	Volles Modell	98	9 581,351	74 648

Tabelle 1: Vergleich der schrittweisen Vorwärtsselektion und Rückwärtselimination

Die Ergebnisse sind in Tabelle 1 zusammengestellt. In dieser Arbeit wird in der Regel mit dem leeren Modell begonnen, und es sind immer beide Richtungen (Vorwärts- und Rückwärtsschritte) möglich.

In R steht für die schrittweise Variablenselektion der Befehl `step()` zur Verfügung. Außerdem wird in dieser Arbeit die Methode `stepwise` aus Kapitel 5.2 verwendet.

Die Rechenzeit der Stepwise-Methode hängt von der Anzahl der benötigten Schritte ab. In jedem Schritt werden p neue Modelle gerechnet, wobei die Dauer für jedes Modell von $\dot{p}$ und n abhängt.

1.5.1 Lokale Optima bei *Stepwise*

Wie schon in Kapitel 1.2 angemerkt, leiden schrittweise Heuristiken darunter, in lokalen Optima *hängen zu bleiben*. Das ist dann der Fall, wenn innerhalb der *Schrittweite* des Algorithmus keine bessere Lösung mehr gefunden wird. Bei Stepwise ist die Schrittweite das Entfernen *oder* Dazunehmen *einer* Variable. (Fasst man ein Modell als binäres Wort der Länge p auf, entspricht die Schrittweite genau der *Hamming-Distanz* 1.)

Zur Veranschaulichung ist in Listing 3 ein winzig kleines Beispiel gegeben.

In den Zeilen 3 bis 6 werden drei erklärende und eine Zielvariable erzeugt. Es ist klar, dass für ein aus einer Variable bestehendes Modell x_3 am besten geeignet ist, denn x_3 ist genauso wie y: $1 \cdot x_1 + 1 \cdot x_2 + \varepsilon$. Stepwise wählt daher x_3 als erstes aus. Das global beste Modell wäre $x_1 + x_2$ (siehe Zeilen 28 bis 31). Mit Stepwise kann es aber nicht erreicht werden, weil sowohl $x_3 + x_1$ als auch $x_3 + x_2$ schlechter sind als das aktuelle Modell $y \sim x_3$ (siehe Zeilen 24 und 25).

```
1  n <- 20
2  set.seed(4)
3  x1 <- rnorm(n, 0, 1)
4  x2 <- rnorm(n, 0, 1)
5  x3 <- x1 + x2 + rnorm(n, 0, 0.5)
6  y <- x1 + x2 + rnorm(n, 0, 2)
7  dat <- data.frame(y, x1, x2, x3)
8  lm1 <- lm(y ~ 1, data=dat)
9  lm2 <- step(lm1, scope=list(lower= y ~ 1, upper=y ~ x1 + x2 + x3))
10 # Start:  AIC=29.19
11 # y ~ 1
```

```
12
13 #        Df Sum of Sq    RSS    AIC
14 # + x3    1    14.805 63.085 26.975
15 # + x2    1    13.117 64.773 27.503
16 # + x1    1    12.677 65.213 27.638
17 # <none>              77.890 29.191
18
19 # Step:  AIC=26.98
20 # y ~ x3
21
22 #        Df Sum of Sq    RSS    AIC
23 # <none>              63.085 26.975
24 # + x1    1     3.488 59.597 27.838
25 # + x2    1     1.545 61.540 28.479
26 # - x3    1    14.805 77.890 29.191
27
28 AIC(lm(y ~ x3))
29 # 85.73255
30 AIC(lm(y ~ x1 + x2))
31 # 83.15901
```

Listing 3: Stepwise findet das lokale Optimum $y \sim x_3$, während das globale Optimum $y \sim x_1 + x_2$ ist.

Abbildung 13 kann als Graph im Sinne der Graphentheorie gesehen werden: die Knoten sind die möglichen Modelle, die Kanten stellen die bei Stepwise erlaubten Übergänge dar (in beide Richtungen). Von jedem Knoten sind p andere Knoten erreichbar. Die restlichen $2^p - p - 1$ sind nicht direkt, sondern höchstens über Umwege erreichbar.

Man kann die Grafik aber auch als Abbildung eines dreidimensionalen Würfels sehen: Jede der (Raum-) Dimensionen steht für eine Variable ($x_1 \dots x_3$), wobei allerdings entlang jeder Achse nur zwei Werte angenommen werden können: 0 oder 1. Jede Ecke dieses Würfels entspricht einem Modell. Diese Anschauung kann auf den p-dimensionalen Raum übertragen werden: Das Ausführen der Stepwise-Methode entspricht dem Wandern über die Ecken des p-dimensionalen Würfels, wobei grundsätzlich nur auf *benachbarte* Ecken gewechselt werden kann. Genau genommen wird immer die Ecke gewählt, die von allen benachbarten Lösungen die größte Verbesserung darstellt.

Immerhin ist gewährleistet, dass man keinen Punkt ein zweites Mal aufsucht.

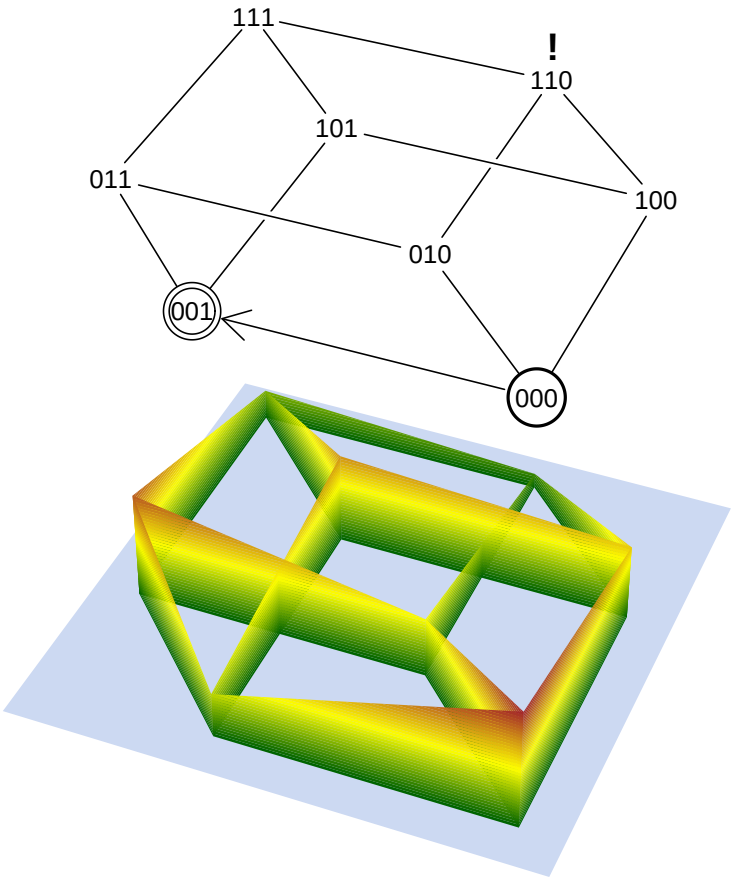

Abbildung 12: *Hängen bleiben* im lokalen Optimum (siehe Listing 3): Vom Ausgangspunkt $(0, 0, 0)$ wird nach $(0, 0, 1)$ die größte mit einem Schritt mögliche Verbesserung erreicht. Von dort ist keine weitere Verbesserung mehr möglich, obwohl das globale Minimum $(1, 1, 0)$ wäre.

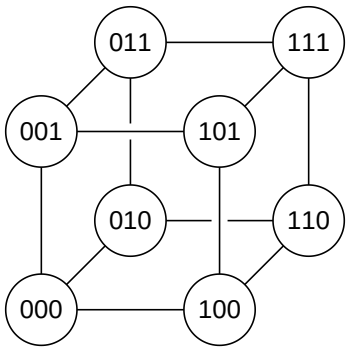

Abbildung 13: Graph der möglichen Modelle: Die binären Zahlen in den Knoten repräsentieren jeweils ein Modell, wobei 1 an der Stelle i bedeutet, dass x_i im Modell enthalten ist.

2 Erzeugen künstlicher Datensätze

2.1 Motivation

Für das Entwickeln und Testen neuer Methoden werden oft Datensätze benötigt, die in der gewünschten Größe, Qualität und Menge nicht zur Verfügung stehen oder nur schwierig zu beschaffen sind. Hier können künstlich erzeugte Daten Abhilfe schaffen, wobei sich die Eigenschaften der Datensätze wesentlich besser kontrollieren lassen als bei „echten“ Daten. Eigenschaften, die von besonderem Interesse sein können, sind z. B.:

- die Anzahl der Beobachtungen (n)
- die Anzahl der Variablen (p)
- die Verteilungen der Variablen
- das Auftreten von *fehlenden Werten*
- die Stärke und Art des Zusammenhangs zwischen den erklärenden und der Zielvariable
- das Auftreten bzw. die Häufigkeit von Ausreißern
- die Varianz des Fehlerterms ε in $y = X\beta + \varepsilon$
- der Anteil der *Störvariablen* in X (Variablen, die in keinem Zusammenhang mit der Zielvariable stehen)
- die Anzahl der relevanten Hauptkomponenten von X.

Abgesehen von diesen gezielt beeinflussten Eigenschaften sollen sich die künstlichen Datensätze natürlich möglichst wie „echte“ Daten verhalten.

2.2 Umsetzung

Hauptkomponenten im Sinne der Hauptkomponentenanalyse sind (vereinfacht ausgedrückt) die jeweils orthogonalen „Hauptrichtungen“ der durch X gegebenen p-dimensionalen Punktwolke.

Faktoren im Sinne der Faktorenanalyse sind nicht direkt beobachtbare (latente) Variablen, auf die durch die beobachteten Variablen (X) geschlossen werden kann. Hauptkomponenten *können* unter Umständen Faktoren entsprechen und umgekehrt.

Bei der im Folgenden vorgestellten Methode `artdata()` werden zunächst einige Komponenten erzeugt, von denen dann sowohl y als auch X abgeleitet werden.

Die Komponenten und die Koeffizienten werden gespeichert, so dass der „wirkliche“ Zusammenhang zwischen y und X rekonstruiert werden kann.

Für die folgende Erläuterung sei:

Abkürzung	Bezeichung im R-Code	Erklärung
H	`hk`	Matrix mit den Komponenten
β_y	`yCoeff`	Koeffizienten für y
$\beta_{x.}$	`xCoeff`	Koeffizienten für X

Die Zielvariable y wird

$$y = H\beta_y + \varepsilon_y \quad \text{wobei} \quad \varepsilon_y \sim N(0, \sigma_y^2) \quad \text{und} \quad \sigma_y^2 = epsVar \cdot var(H\beta_y)$$

und die erklärenden Variablen x_i

$$x_i = H\beta_{x_i} + \varepsilon_{x_i} \quad \text{wobei} \quad \varepsilon_{x_i} \sim N(0, \sigma_{x_i}^2) \quad \text{und} \quad \sigma_{x_i}^2 = xErr_i \cdot var(H\beta_{x_i})$$

gebildet.

2.2.1 Eingabeparameter

Alle Argumente haben Defaultwerte (Voreinstellungen). Im Sinne von Kapitel 2.1 ist es aber angebracht, sich je nach Bedarf sinnvolle Werte zu überlegen und eventuell verschiedene Einstellungen auszuprobieren.

`n`	die Anzahl der *Beobachtungen* im Datensatz ($n > 1$)
`p`	die Anzahl der *erklärenden* Variablen ($p \in \mathbb{N}^+$)
`hk`	eine $n \times k$ Matrix mit den k Hauptkomponenten (Faktoren)
`yCoeff`	ein Vektor der Länge k mit den Koeffizienten für y
`xCoeff`	eine $p \times k$ Matrix mit den Koeffizienten für X
`epsVar`	die Varianz des Fehlerterms in der Relation zur Varianz von y ($epsVar \in]0, \infty[$)
`xErr`	Vektor mit den Störgrößen der X-Variablen

2.2.2 Rückgabewerte

Das Rückgabeobjekt der Funktion `artdata()` ist eine Liste mit folgenden Elementen: `y`, `x`, `hk`, `yCoeff`, `xCoeff`, `epsVar` und `xErr`. Die ersten beiden Elemente enthalten y und X, die restlichen dienen der Aufbewahrung der übergebenen Argumente (bzw. der Defaultwerte). `hk` unterscheidet sich insofern vom übergebenen Argument, als es innerhalb der Methode skaliert wurde.

2.2.3 R-Code von `artdata`

```
1  artdata <- function(n=200, p=100, hk, yCoeff, xCoeff, epsVar, xErr) {
2      if(missing(hk)) hk <- matrix(rnorm(n*5), n, 5)
3      hk  <- scale(hk)
4      nhk <- ncol(hk)
5      if(missing(yCoeff)) {
```

```
6           tmp    <- ceiling(nhk/2)
7           yCoeff <- sample(c(runif(tmp, min=-1), rep(0, nhk-tmp)))
8       }
9       if(missing(xCoeff)) {
10          xCoeff <- matrix(0, p, nhk)
11          for(i in 1:p) {
12              ncoeffs    <- sample(1:max(1,nhk-1), 1)
13              xCoeff[i,] <- sample(c(runif(ncoeffs, min=-1), rep(0,nhk-
                    ncoeffs)))
14              xCoeff[i,] <- xCoeff[i,] / sum(abs(xCoeff[i,]))
15          }
16      }
17      if(missing(epsVar)) epsVar <- 0.1
18      if(missing(xErr)) xErr <- seq(0.1, 1, length=p)
19
20      y   <- as.vector(hk %*% yCoeff)
21      eps <- rnorm(n, 0, sqrt(var(y)*epsVar))
22      y   <- y + eps
23      if(min(y) < 0) y <- y - min(y) + runif(1)
24
25      x <- matrix(0, n, p)
26
27      for(i in 1:p) {
28          x[,i] <- hk %*% xCoeff[i,]
29          x[,i] <- x[,i] + rnorm(n, 0, max(sqrt(xErr[i]*var(x[,i])),0.01))
30          x[,i] <- x[,i] - min(x[,i]) + runif(1)
31      }
32
33      x        <- as.data.frame(x)
34      names(x) <- paste("x", 1:ncol(x), sep="")
35      y        <- as.vector(y)
36      hk       <- as.data.frame(hk)
37
38      return(list(y=y, x=x, hk=hk, yCoeff=yCoeff, xCoeff=xCoeff, epsVar=
            epsVar, xErr=xErr))
39  }
```

Listing 4: Die Methode `artdata` für die Erzeugung künstlicher Datensätze.

2.3 Anwendungsbeispiel der Methode `artdata`

Die Verwendung von `artdata()` wird an einem kleinen Beispiel vorgeführt. Der erzeugte Datensatz „`art200x96`“ wird anschließend überblicksmäßig betrachtet.

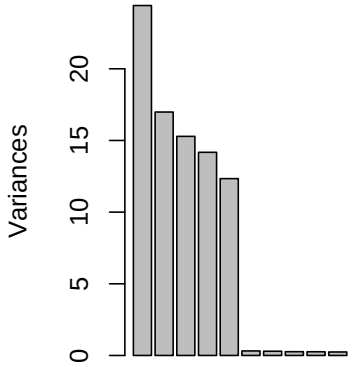

Abbildung 14: Die ersten 10 Hauptkomponenten von X

```
1  library(e1071)
2  set.seed(123)
3  n  <- 200
4  hk <- data.frame(rnorm(n), runif(n), rbinom(n, 50, 0.5), rlnorm(n),
       rbinom(n, 1, 0.4))
5  names(hk) <- paste("k", 1:ncol(hk), sep="")
6  yC <- c(1, 0.75, 0.5, 0, 0)
7  b  <- bincombinations(5)
8  xC <- rbind(b, b, b)
9  xC <- xC * runif(length(xC), min=-1)
10 xE <- rep(0.5, 96)
11
12 art200x96 <- artdata(n=n, p=96, hk=hk, yCoeff=yC, xCoeff=xC, epsVar
       =0.5, xErr=xE)
13 x <- art200x96$x
14 y <- art200x96$y
```

Listing 5: Beispiel-Anwendung von `artdata`

Es wurden 5 Hauptkomponenten (Zeile 4 im Listing 5) erzeugt, diese sind mittels Hauptkomponentenzerlegung (in R: `princomp(x)`) klar erkennbar (siehe Abbildung 14).

Als Störgröße der Zielvariable (`epsVar`) wurde 0,5 übergeben. Sei die Varianz von y ohne ε A, dann ist die Varianz von ε $A/2$ und die Varianz von $y+\varepsilon$ ist erwartungsgemäß $1{,}5A$. Davon kann aber ohne Überanpassung (*overfitting*) nur A durch X erklärt werden, der theoretisch maximal erreichbare Wert für R^2 ist also $\frac{1A}{1{,}5A} = 0{,}\dot{6}$.

In Abbildung 15 ist der R^2-Wert für eine Reihe von Modellen der Größe 1 bis 21 dargestellt. Die Modelle wurden mit der Stepwise-Methode gefunden, wobei beim leeren Modell begonnen wurde. Als Kriterium zur Modellbewertung wurde AIC (schwarze Linie) und BIC (rote Linie) verwendet. Das BIC-Kriterium führt zu einem Modell mit $\dot{p} = 6$ und $R^2 = 0{,}69$, bei der Verwendung von AIC ist $\dot{p} = 21$ und $R^2 = 0{,}76$. Im letzteren Fall handelt es sich also um *overfitting*. Abbildung 16 zeigt die mittels `rep.cv` (siehe

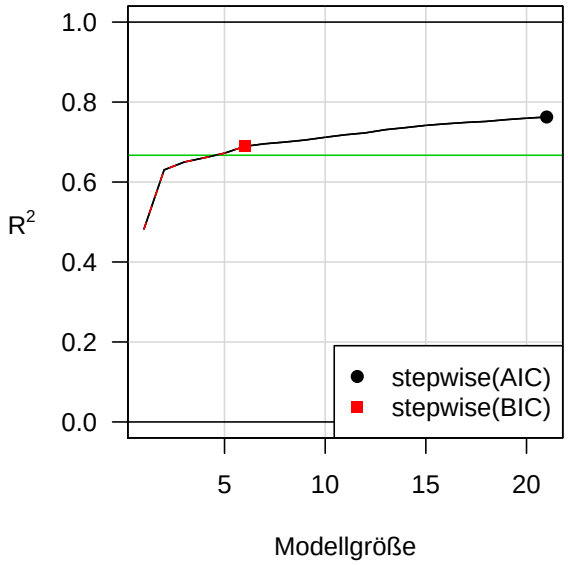

Abbildung 15: R^2 verschiedener Stepwise-Modelle.

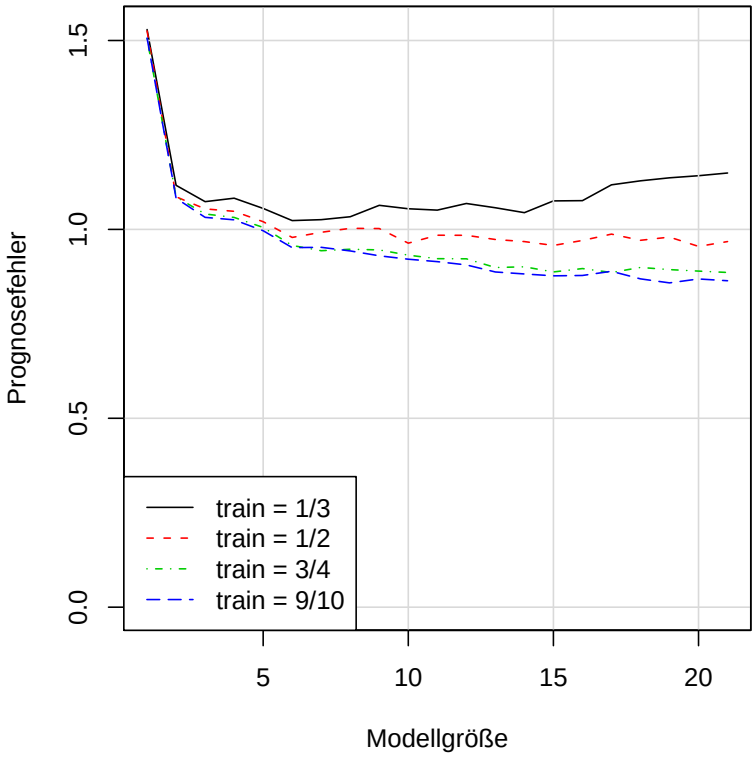

Abbildung 16: Der Prognosefehler, abhängig von der Modellgröße. Für die Ermittlung des Prognosefehlers wurde die Methode `rep.cv` von Kapitel 5.4 verwendet.

Kapitel 5.4) ermittelten Prognosefehler abhängig von der Modellgröße und der Größe des Trainings-Sets. Es stellt sich heraus, dass der Prognosefehler zumindest bei ausreichend großen Trainings-Datensätzen tatsächlich mit der zunehmenden Modellgröße sinkt. Das AIC-Kriterium hat also anscheinend trotz *overfitting* das bessere Prognosemodell gebracht.

3 Heuristiken

3.1 Einfache, schnelle Heuristiken

Außer dem bereits in Kapitel 1.5 vorgestellten Stepwise-Algorithmus sind noch weitere einfache aber schnelle Heuristiken denkbar. Beispiele dafür werden hier kurz besprochen und verglichen (Abbildung 17 und 18).

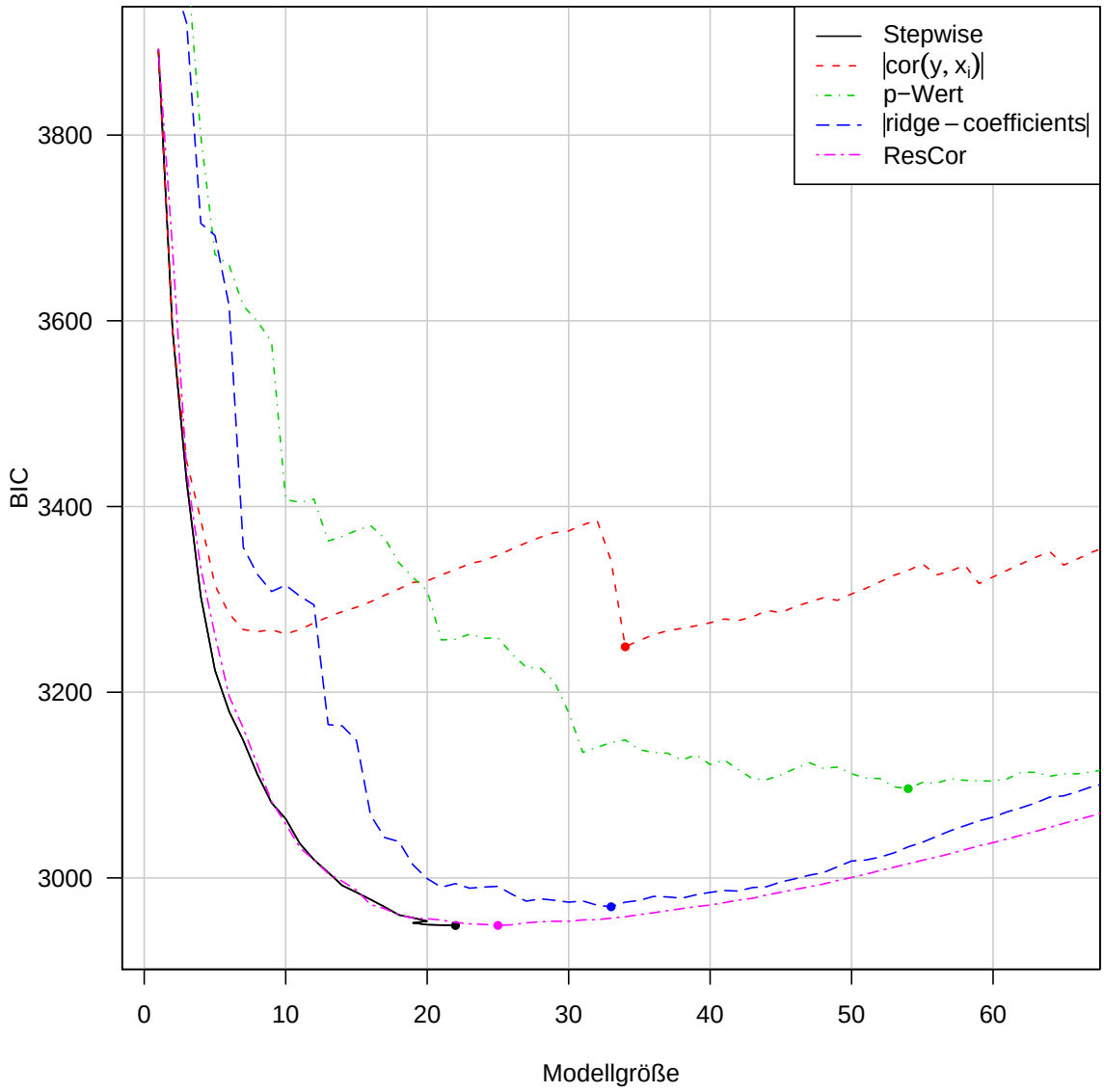

Abbildung 17: Vergleich einfacher Methoden mit den ART-Daten

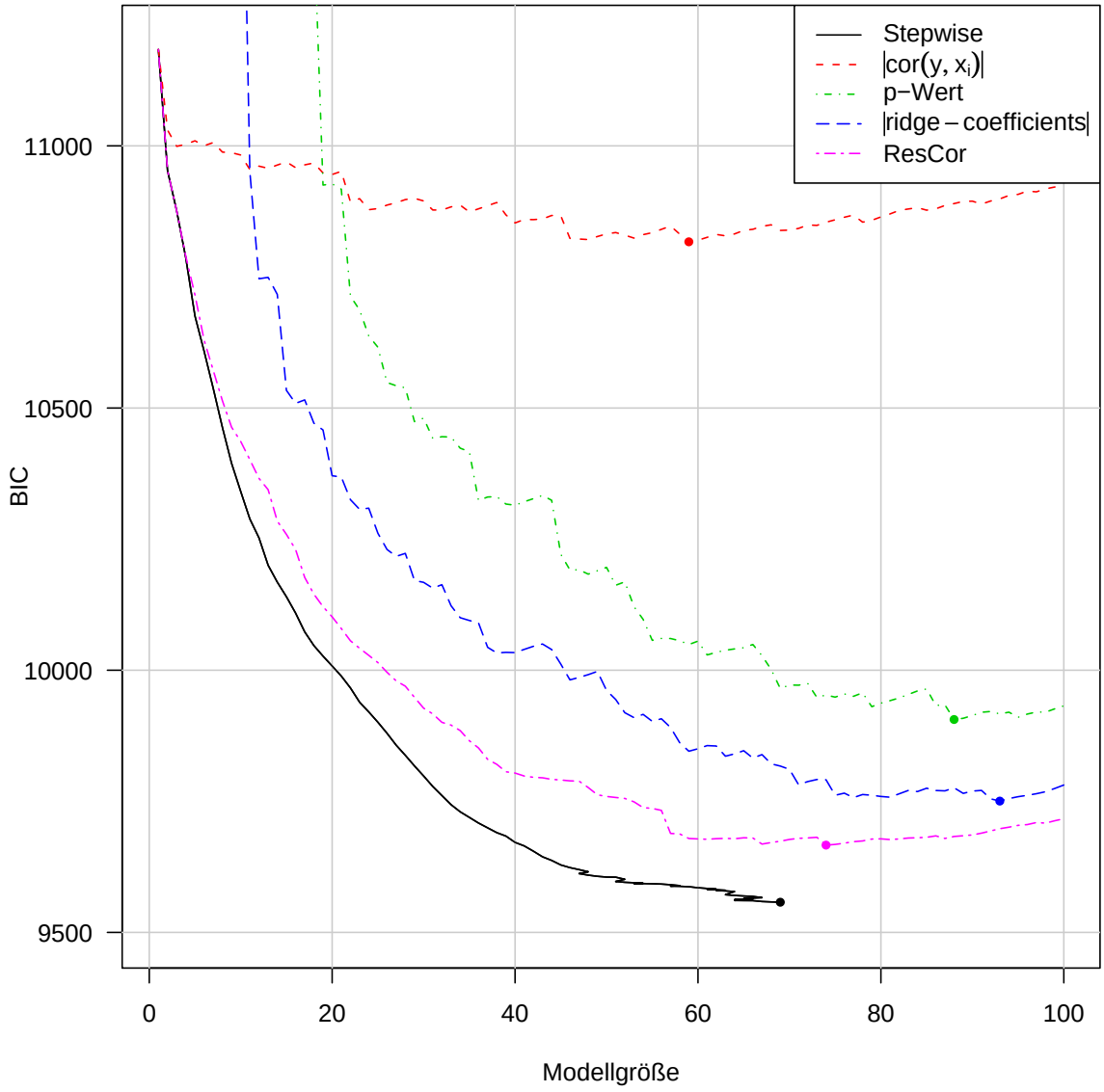

Abbildung 18: Vergleich einfacher Methoden mit den TOX-Daten

3.1.1 Schrittweises Hinzunehmen nach einer vorher bestimmten Reihenfolge

Eine sehr einfache und schnelle Möglichkeit besteht darin, sich ein Kriterium zu überlegen, von dem man annimmt, dass es die Nützlichkeit der Variablen repräsentieren könnte. Die Variablen werden dann nach diesem Kriterium (absteigend) geordnet und der Reihe nach ins Modell aufgenommen. Für jede denkbare Modellgröße ergibt sich somit eine eindeutige Lösung. Für jede dieser Lösungen kann dann ein beliebiges Gütekriterium berechnet werden, und von diesen p Modellen wird das beste ausgewählt. Für die Reihung der Variablen könnten unter anderem folgende Möglichkeiten herangezogen werden:

- Die Korrelation zwischen den Variablen und der Zielgröße:

$$max(|cor(y, x_i)|)$$

  Dieses Kriterium führt bei den hier verwendeten Datensätzen zu keinem befriedigenden Ergebnis. Bei anderen Daten kann das aber anders sein, vor allem, wenn y nur von *einem* Faktor im Sinne der Faktorenanalyse abhängt.

- Der P-Wert vom Test ob die einzelnen Regressionskoeffizienten des vollen Modells 0 sind:

$$min(\text{P-Wert})$$

  Dieser steht natürlich nur zur Verfügung wenn $n > p$. Gute Resultate wird diese Methode bringen wenn:

  - $n \gg p$
  - die erklärenden Variablen untereinander nicht hoch korrelieren.

- Die Ridge-Regression-Koeffizienten:

$$max(|\beta^{Ridge}|)$$

  mit $\hat{\beta}^{Ridge} = \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}$.

  Diese Methode bringt bei beiden Datensätzen gute Ergebnisse. Allerdings muss zuvor ein günstiges λ ermittelt werden.

3.1.2 ResCor: Schrittweises Hinzunehmen nach der maximalen Korrelation mit den Residuen

Etwas intelligenter als die in Kapitel 3.1.1 vorgestellten Heuristiken, aber wesentlich schneller als Stepwise, ist die Methode *ResCor* (*Residuals Correlation*).

Dabei wird, beginnend mit dem leeren Modell, immer diejenige Variable zum Modell genommen, die die größte absolute Korrelation mit den Residuen des aktuellen Modells hat. Es ergibt sich also für jede Modellgröße $\dot{p}$ eine eindeutige Lösung.

- Die Methode führt zu einem eindeutigen Ergebnis.
- Die Methode ist schnell: Bei jedem Schritt muss nur ein Modell geschätzt werden (bei Stepwise müssen für jeden Schritt p Modelle gerechnet werden).
- Bei den ART-Daten führt die Methode bei Evaluierung mit BIC zu einem ungefähr gleich guten Ergebnis wie Stepwise, bei den TOX-Daten ist es etwas schlechter.

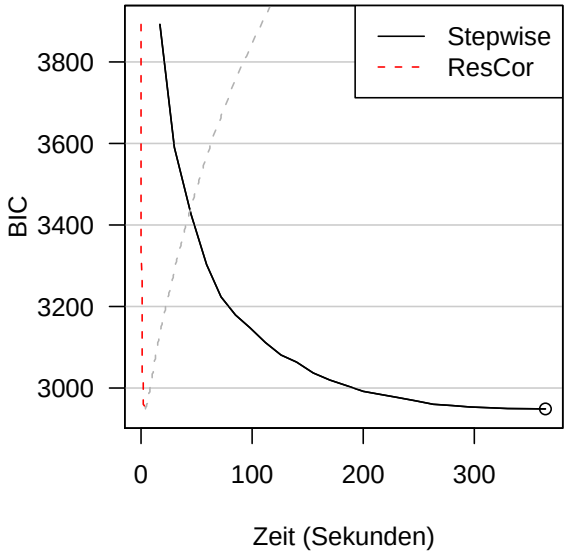

Abbildung 19: Die Entwicklung des BIC-Kriteriums abhängig von der Rechenzeit bei der Methode *ResCor* mit dem ART-Datensatz

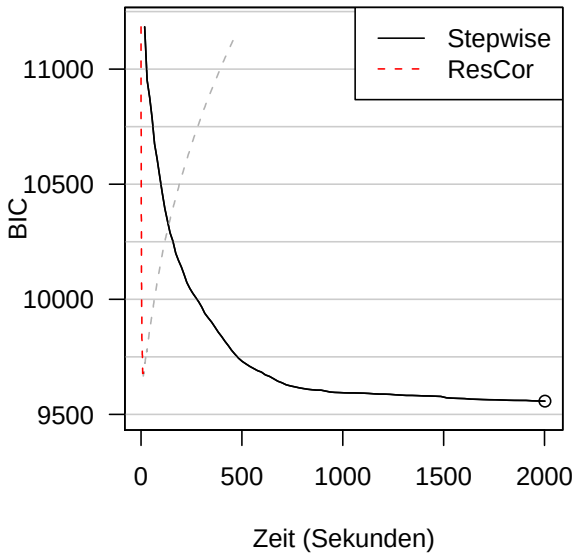

Abbildung 20: Die Entwicklung des BIC-Kriteriums abhängig von der Rechenzeit bei der Methode *ResCor* mit dem TOX-Datensatz

```
1 resCor <- function(y, x, maxSteps) {
2     startTime <- as.integer(Sys.time())
3     p    <- ncol(x)
4     if(missing(maxSteps)) maxSteps <- p - 1
5     k    <- log(nrow(x))
6     m    <- matrix(0, maxSteps+2, p)
7      bics <- rep(NA, maxSteps)
```

```
8       usedTime <- rep(NA, maxSteps)
9
10      for(i in 1:(maxSteps+1)) {
11          lm1     <- lm(form(m[i,], names(x)), data=x)
12          if(i > 1) {
13              bics[i-1] <- AIC(lm1, k=k)
14              usedTime[i-1] <- as.integer(Sys.time())-startTime
15          }
16          m[i+1,] <- m[i,]
17          tmpcor  <- as.vector(abs(cor(x=x[,!m[i,]], y=residuals(lm1),
               method="pearson")))
18          m[i+1, which(!m[i,])[which.max(tmpcor)]] <- 1
19      }
20      m <- m[-c(1,maxSteps+2),]
21      return(list(usedTime=usedTime, bic=bics, models=m))
22  }
```

Listing 6: Die Methode `resCor` nimmt immer die Variable zum Modell, die die absolut größte Korrelation zu den Residuen des bestehenden Modells hat.

3.2 Methoden zum Austauschen einzelner Variablen

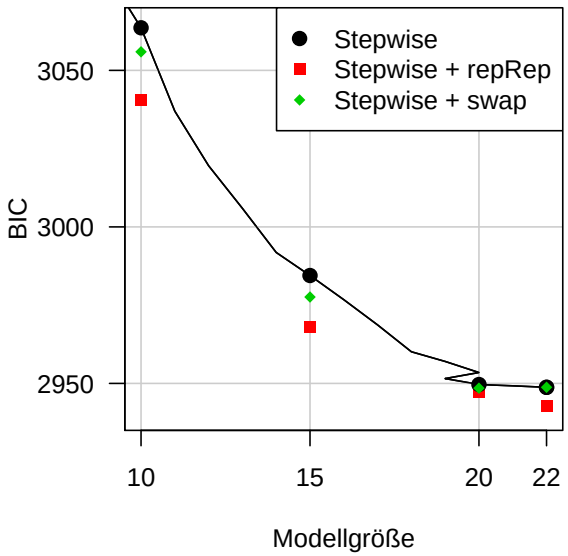

Abbildung 21: Verbesserung der Stepwise-Modelle mit den Methoden `repRep` und `swap` mit den ART-Daten

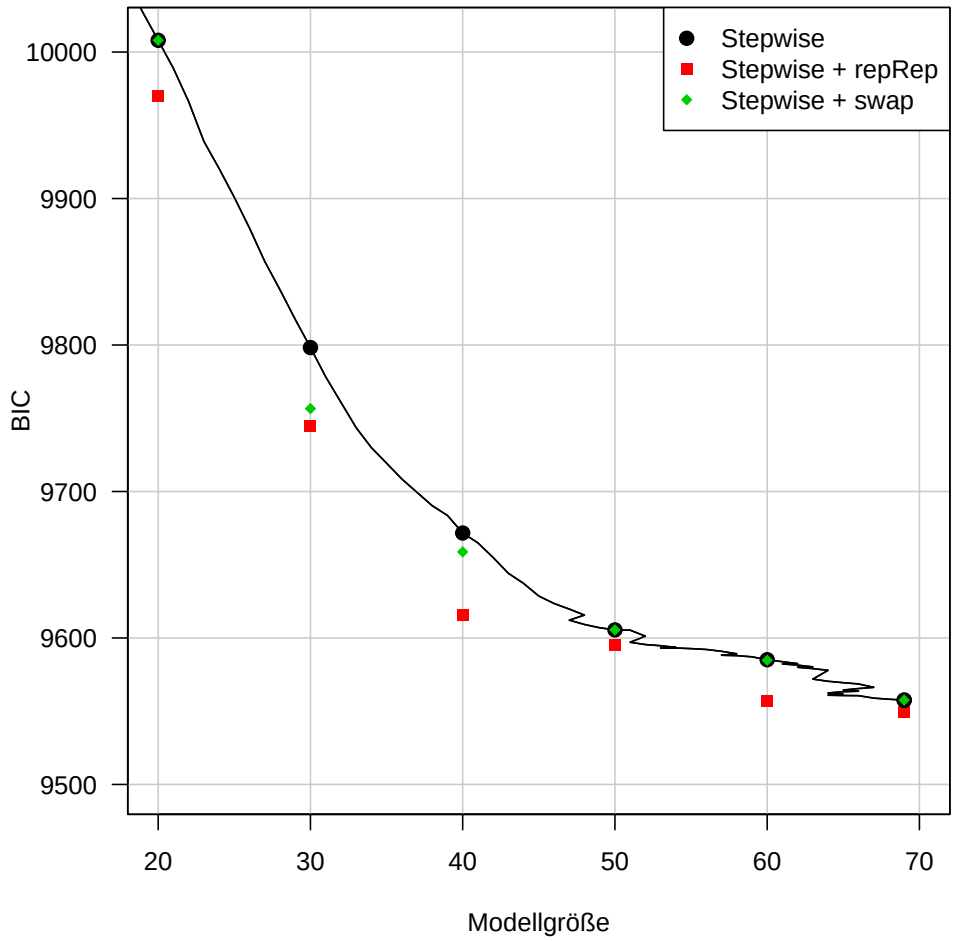

Abbildung 22: Verbesserung der Stepwise-Modelle mit den Methoden `repRep` und `swap` mit den TOX-Daten

Betrachtet man z. B. ein mittels *stepwise* gewonnenes Modell, so stellt sich die Frage, ob und wie dieses weiter verbessert werden kann. Eine Möglichkeit, die die Modellgröße unverändert lässt, ist das *Austauschen* von Variablen. Dies kann z. B. systematisch geschehen, wobei es (für jeden Schritt) genau $\dot{p}\,(p-\dot{p})$ Möglichkeiten gibt.

```
1  rep1 <- function(y, x, m) {
2      m1 <- which(m==1)
3      m0 <- which(m==0)
4      l1 <- length(m1)
5      l2 <- length(m0)
6
7      namesX <- names(x)
8      r2 <- summary(lm(form(m, namesX), data=x))$r.squared
9      mTmp <- m
10     r2res <- NULL
```

```
11      models <- NULL
12
13      for(i in m1) {
14          for(j in m0) {
15              mTmp <- m
16              mTmp[i] <- 0
17              mTmp[j] <- 1
18              tmp <- summary(lm(form(mTmp, namesX), data=x))$r.squared
19              if(tmp > r2) {
20                  models <- rbind(models, mTmp)
21                  r2res <- c(r2res, tmp)
22              }
23          }
24      }
25
26      if(!is.null(r2res)) {
27          wm <- which.max(r2res)
28          return(models[wm,])
29      }
30      return(NULL)
31  }
```

Listing 7: Die Methode `rep1` führt systematisch alle möglichen Vertauschungen *einer* Variable durch.

```
1   repRep <- function(y, x, m) {
2       tmp <- m
3       res <- NULL
4       while(!is.null(tmp)) {
5           res <- rbind(res, tmp)
6           tmp <- rep1(y=y, x=x, m=tmp)
7       }
8       return(res)
9   }
```

Listing 8: Die Methode `repRep` ruft `rep1` wiederholt auf, bis sich keine Verbesserung mehr ergibt.

Die Methode `rep1()` (*replace one*) führt diese systematische Vertauschung durch. Die Eingabeparameter sind y, X und ein Startmodell. Rückgabewert ist das beste durch Vertauschen *einer* Variable gefundene Modell und *NULL*, falls keine Verbesserung gefunden

wurde (Listing 7).

Die Funktion `repRep()` (*Repeated Replacement*) führt `rep1` so lange aus, bis keine Verbesserung mehr gefunden wird (Listing 8). Rückgabewert ist eine Sequenz der nach jedem `rep1`-Aufruf besten Modelle.

```
swap <-function(y, x, k, m) {
    P <- sum(m)
    p <- ncol(x)
    n <- nrow(x)
    if(missing(k)) k <- log(n)
    namesX <- names(x)
    dat    <- data.frame(y, x)
    mT     <- m

    lmTmp  <- lm(y ~ 1, data=dat)
    lmTmp  <- update(lmTmp, form(m, namesX))
    lmFull <- lm(form(rep(1,p), namesX), data=dat)
    lm0    <- lm(form(rep(0,p), namesX), data=dat)

    bicB <- AIC(lmTmp, k=k)

    while(TRUE) {
        mT[which(mT==1)[which.max(summary(lmTmp)$coef[-1,4])]] <- 0
        lmTmp  <- update(lmTmp, form(mT, namesX))
        lmTmp  <- step(object=lmTmp, scope=list(upper=lmFull, lower=lm0)
            , direction="forward", trace=0, steps=1, k=k)
        bicTmp <- AIC(lmTmp, k=k)
        mT     <- as.numeric(is.element(el=namesX, set=names(lmTmp$coef)
           ))

        if(bicTmp >= bicB || sum(mT) < P) {
            break
        } else {
            bicB <- bicTmp
            m    <- mT
        }
    }
    return(m)
}
```

Listing 9: `swap` ersetzt in jedem Schritt die Variable mit dem größten P-Wert vom Test auf Signifikanz der Regressionskoeffizienten.

Während `repRep` in jedem einzelnen Schritt alle $\dot{p}\,(p - \dot{p})$ möglichen Vertauschungen ausprobiert, wird bei `swap` (Listing 9) ein vereinfachter, schnellerer Weg beschritten: Es wird nur *eine* Variable entfernt, nämlich diejenige, die im aktuellen Modell den größten P-Wert beim Test ob dieser Regressionskoeffizient 0 ist, aufweist. Dann wird von den $p - \dot{p}$ in Frage kommenden Möglichkeiten die beste ausgewählt (das entspricht genau einem *step-forward*-Schritt).

Die Abbildungen 21 und 22 zeigen die Verbesserungen, die mit `repRep` und `swap` erzielt wurden. Die Ausgangsmodelle stammen vom Stepwise-Algorithmus. Wie zu erwarten war, ist `repRep` durchwegs besser, was aber mit der $\dot{p}$-fachen Rechenzeit teuer erkauft ist.

Natürlich kann auch eine komplett zufällige Vertauschung durchgeführt werden. In diesem Fall kann man unabhängig von p festlegen, wie viele Versuche in jedem Schritt unternommen werden sollen. Dieses Verfahren wird bei der Methode RASAR (Kapitel 3.4) angewandt.

3.3 STAP kombiniert Stepwise und Swap

Die Funktion `stap` kombiniert die Methode `step` mit der in Kapitel 3.2 beschriebenen `swap`-Methode. Nach jedem Stepwise-Schritt wird versucht, durch Austauschen von Variablen bessere Modelle gleicher Größe zu finden. Die resultierenden Modelle sind ungefähr gleich gut wie die mit Stepwise gefundenen, aber kleiner.

Ein Vergleich von Stepwise und STAP ist in Abbildung 23 und 24 gegeben. Bei gegebener Größe sind die STAP-Modelle etwas besser, wobei STAP natürlich etwas mehr Rechenzeit benötigt.

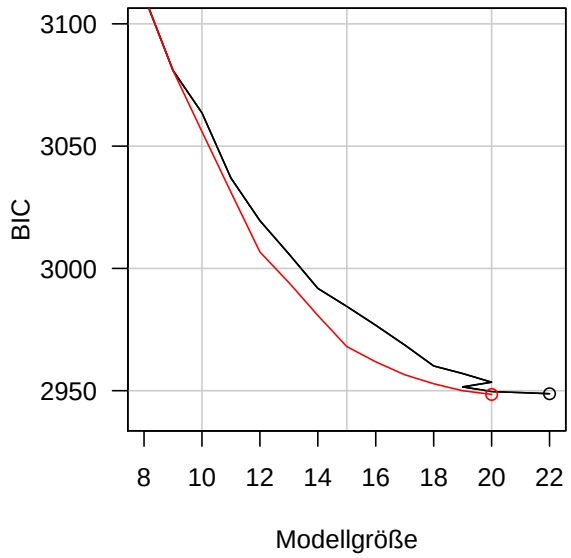

Abbildung 23: Vergleich der `stepwise`- und `stap`-Modelle anhand der ART-Daten

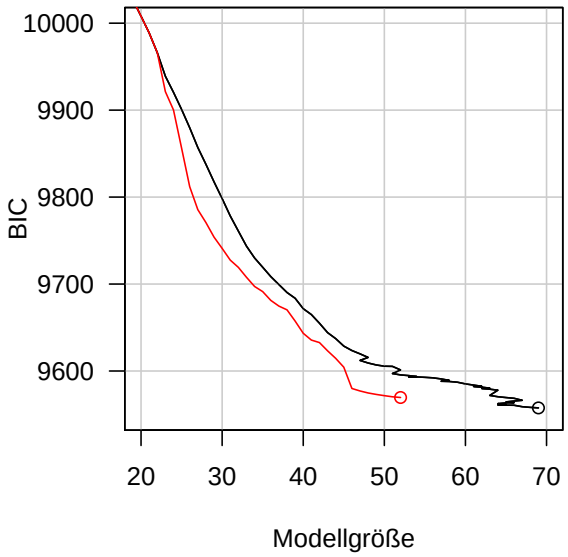

Abbildung 24: Vergleich der `stepwise`- und `stap`-Modelle anhand der TOX-Daten

```
stap <-function(y, x, k, maxTime=3600, resname="stapRes", lcol=2, lty1
   =1) {
    startTime <- as.integer(Sys.time())
    p <- ncol(x)
    n <- nrow(x)
    if(missing(k)) k <- log(n)
    namesX <- names(x)
    dat    <- data.frame(y, x)

    usedTime <- NULL
    bics     <- NULL
    models   <- NULL

    lmTmp  <- lm(y ~ 1, data=dat)
    lmTmp  <- update(lmTmp, form(rep(0,p), namesX))
    lmFull <- lm(form(rep(1,p), namesX), data=dat)
    lm0    <- lm(form(rep(0,p), namesX), data=dat)

    lmTmp    <- step(object=lmTmp, scope=list(upper=lmFull, lower=lm0),
       direction="forward", trace=0, steps=5, k=k)
    models   <- rbind(models, as.numeric(is.element(el=namesX, set=names
       (lmTmp$coef))))
    usedTime <- c(usedTime, as.integer(Sys.time())-startTime)
    bics     <- c(bics, AIC(lmTmp, k=k))

```

```
23      while(as.integer(Sys.time())-startTime < maxTime) {
24          m      <- swap(y=y, x=x, k=k, m=models[nrow(models),])
25          lmTmp  <- update(lmTmp, form(m, namesX))
26          lmTmp  <- step(object=lmTmp, scope=list(upper=lmFull, lower=lm0)
                , direction="both", trace=0, steps=1, k=k)
27          bicTmp <- AIC(lmTmp, k=k)
28
29          if(bicTmp >= bics[length(bics)]) {
30              break
31          } else {
32              bics     <- c(bics, bicTmp)
33              usedTime <- c(usedTime, as.integer(Sys.time())-startTime)
34              models   <- rbind(models, as.numeric(is.element(el=namesX,
                    set=names(lmTmp$coef))))
35          }
36
37          write.table(cbind(usedTime, bics, models), file=paste(resname, "
                .txt", sep=""), col.names=FALSE, row.names=FALSE)
38          lines(usedTime, bics, col=lcol, lty=lty1)
39      }
40      return(list(usedTime=usedTime, bic=bics, models=models))
41  }
```

Listing 10: Die Methode `stap` kombiniert `stepwise` und `swap`.

3.4 RASAR: Vorwärts- und Rückwärtsschritte kombiniert mit Variablenaustausch

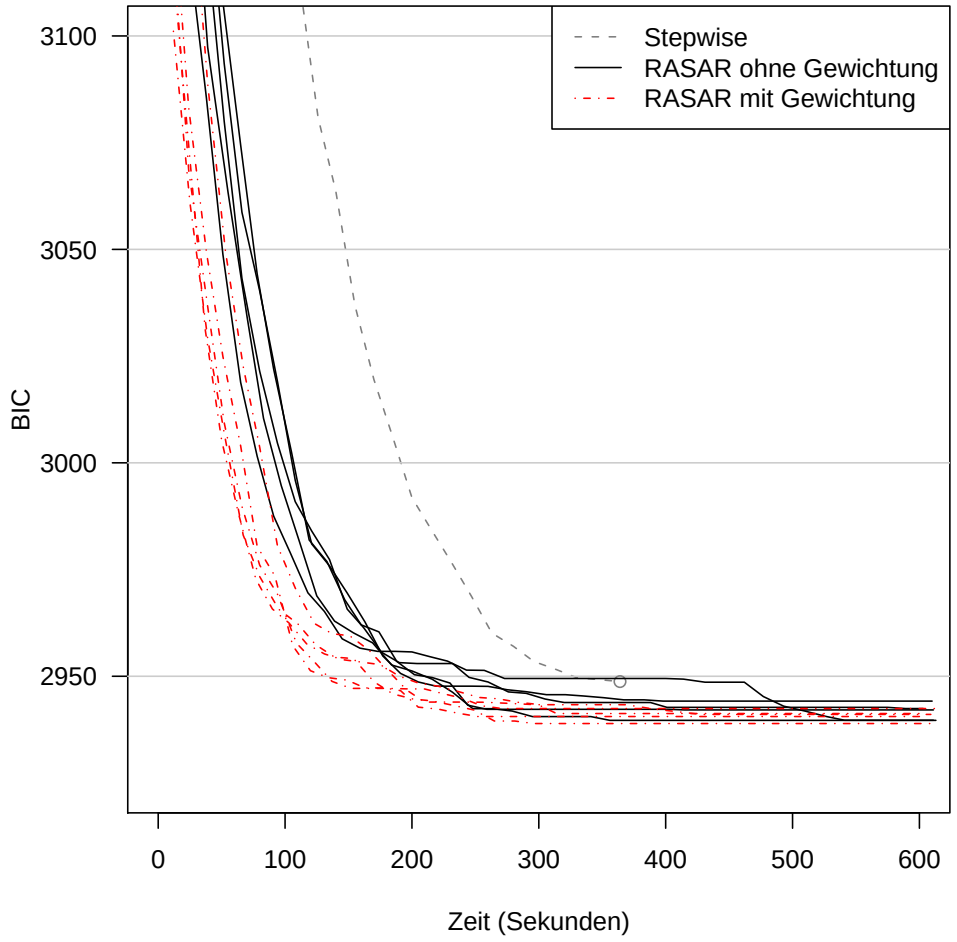

Abbildung 25: Die Methode RASAR mit den ART-Daten

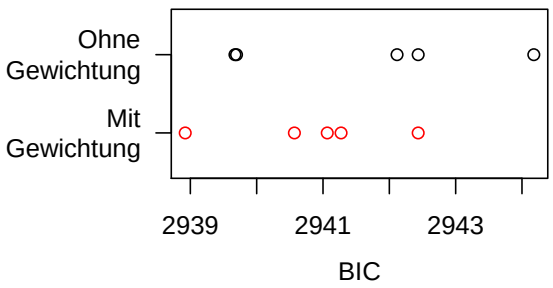

Abbildung 26: Die beiden Gruppen (mit und ohne Gewichtung) unterscheiden sich im Endergebnis (nach 600 Sekunden) nur unwesentlich (ART-Daten).

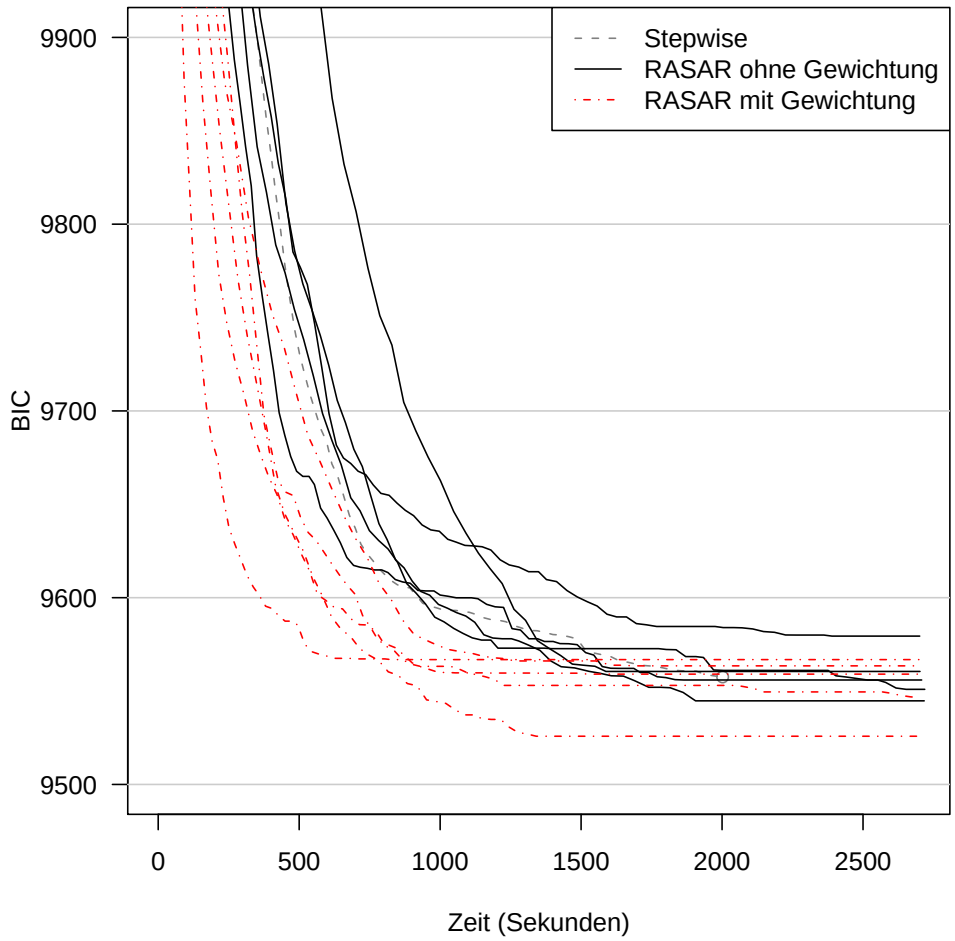

Abbildung 27: Die Methode RASAR mit den TOX-Daten

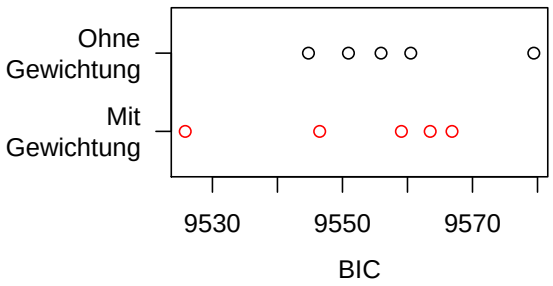

Abbildung 28: Auch bei den TOX-Daten ist der Einfluss der Gewichtung auf die Endergebnisse unwesentlich.

Die Methode RASAR (*RAndom Steps And Replacement*) ist eine Erweiterung und Modifizierung des Stepwise-Algorithmus. Beginnend mit einem zufälligen Startmodell werden in jedem Schritt drei Verbesserungsversuche unternommen:

1. Für alle $\dot{p}$ im Modell befindlichen Variablen wird überprüft, ob sich durch das

Entfernen eine Verbesserung ergibt. Falls das der Fall ist, wird diejenige Variable entfernt, die die größte Verbesserung bringt.

2. Eine vorher festgelegte Anzahl k an Versuchen wird unternommen, durch das *zufällige* Austauschen einer Variable Verbesserungen zu erzielen. Es wird die Vertauschung ausgeführt, die den größten Nutzen bringt.

3. Es werden k ($k \ll p$) *zufällig* ausgewählte Variablen zum Modell genommen, und wieder wird die beste Variante beibehalten.

Eine wesentliche Eigenschaft von RASAR ist, dass die zufälligen Auswahlschritte beeinflusst werden können, indem die Variablen *gewichtet* werden. Diese Gewichtung steuert:

- mit welcher Wahrscheinlichkeit eine Variable im Startmodell enthalten ist
- mit welcher Wahrscheinlichkeit versucht wird, eine Variable durch eine andere zu ersetzen
- wie wahrscheinlich eine Variable zusätzlich ins Modell aufgenommen wird.

Für die Erzeugung der Gewichte eignen sich z. B. die Kriterien, die in Kapitel 3.1.1 zur Reihung der Variablen verwendet wurden (Korrelation, P-Wert, Ridge-Koeffizient). In diesem Beispiel (Abbildung 25 und 27) wird

$$w_i = 1 + \frac{100 \cdot |\beta_i^{Ridge}|}{max(|\beta^{Ridge}|)} \tag{5}$$

zur Gewichtung verwendet. In diesen beiden Beispielen sinkt der BIC-Wert bei Verwendung der Gewichtung schneller ab, die Ergebnisse am Ende der betrachteten Rechenzeit unterscheiden sich allerdings nicht mehr wesentlich von den Ergebnissen ohne Gewichtung.

Während bei Stepwise in jedem Schritt p Modelle gerechnet werden, sind es bei RASAR $\dot{p} + 2k$, wobei bis zu drei Verbesserungen realisiert werden.

Die Methode wird nach einer vorgegebenen Zeit beendet. Als günstig hat sich eine Rechenzeit erwiesen, die etwa gleich oder etwas länger als die Rechenzeit von `stepwise` ist.

Das mit RASAR erzielbare Ergebnis hängt vor allem von folgenden Faktoren ab:

- ob sinnvolle Gewichte gefunden werden können
- von einer günstigen Wahl der Größe des Startmodells. Als günstig dürften Startwerte betrachtet werden, die gleich oder etwas kleiner sind als das tatsächlich beste Modell.
- von der Anzahl der Versuche

```
rasar <- function(y, x, k, startM, maxTime=3600, resname="rasarRes",
    weightS, lcol=2, lty1=1, startSize=30) {
     startTime <- as.integer(Sys.time())
     p <- ncol(x)
     n <- nrow(x)
     if(missing(k)) k <- log(n)

     nM1 <- 100

     namesX <- names(x)
     if(missing(weightS)) weightS <- rep(1, p)
     e <- weightS
     E <- 1/e

     if(missing(startM)) {
         best <- rep(0, p)
         best[sample(1:p, startSize, prob=e)] <- 1
     } else {
         best <- startM
     }
     bestBic <- AIC(lm(form(best, namesX), data=x), k=k)

     usedTime <- NULL
     bics     <- NULL
     models   <- NULL

     inB  <- which(best == 1)
     outB <- which(best == 0)
     P    <- sum(best)
     z    <- 1

     while(as.integer(Sys.time())-startTime < maxTime) {
         if(length(bics) >= 2 && bics[length(bics)] < bics[length(bics)
            -1]) {
             m <- matrix(0, P, p)
             m[,inB] <- 1
             b <- rep(Inf, P)
             for(i in 1:P) {
                 m[i, inB[i]] <- 0
                 b[i] <- AIC(lm(form(m[i,], namesX), data=x), k=k)
             }
             tmp <- which.min(b)
             if(b[tmp] <= bestBic) {
                 bestBic <- b[tmp]
```

```
43                  best    <- m[tmp,]
44                  inB     <- which(best == 1)
45                  outB    <- which(best == 0)
46                  P <- sum(best)
47                  z <- 0
48              }
49          }
50
51          m <- matrix(0, nM1, p)
52          m[,inB] <- 1
53          b <- rep(Inf, nM1)
54          for(i in 1:nM1) {
55              m[i, sample(inB, 1, prob=E[inB])] <- 0
56
57              m[i, sample(outB, 1, prob=e[outB])] <- 1
58              b[i] <- AIC(lm(form(m[i,], namesX), data=x), k=k)
59          }
60          tmp <- which.min(b)
61          if(b[tmp] < bestBic) {
62              bestBic <- b[tmp]
63              best <- m[tmp,]
64              inB  <- which(best == 1)
65              outB <- which(best == 0)
66              P <- sum(best)
67              z <- 0
68          }
69
70          m <- matrix(0, nM1, p)
71          m[,inB] <- 1
72          b <- rep(Inf, nM1)
73          for(i in 1:nM1) {
74              m[i, sample(outB, 1, prob=e[outB])] <- 1
75              b[i] <- AIC(lm(form(m[i,], namesX), data=x), k=k)
76          }
77          tmp <- which.min(b)
78          if(b[tmp] < bestBic) {
79              bestBic <- b[tmp]
80              best <- m[tmp,]
81              inB  <- which(best == 1)
82              outB <- which(best == 0)
83              P <- sum(best)
84              z <- 0
85          }
86
```

```
87          z <- z + 1
88          usedTime <- c(usedTime, as.integer(Sys.time())-startTime)
89          bics     <- c(bics, bestBic)
90          models   <- rbind(models, best)
91          lines(usedTime, bics, col=lcol, lty=lty1)
92          write.table(cbind(usedTime, bics, models), file=paste(resname, "
                .txt", sep=""), col.names=FALSE, row.names=FALSE)
93
94      }
95
96      return(list(usedTime=usedTime, bic=bics, models=models, best=best))
97  }
```

Listing 11: Die Methode `rasar` führt zufällige Stepwise-Schritte und Vertauschungen durch.

3.4.1 Untersuchung der Ähnlichkeit verschiedener Modelle

Die Methode RASAR führt zumindest beim ART-Datensatz zu Modellen, die hinsichtlich des verwendeten Gütekriteriums sehr ähnlich sind (siehe Abbildung 25 und Tabelle 2).

Es stellt sich daher die Frage, wie stark sich diese Modelle bezüglich der Größe und der beteiligten Variablen unterscheiden. Die fünf besten Modelle (A, ..., E) sind in den Tabellen 2 bis 4 verglichen.

Die mittlere Hamming-Distanz ist mit 16 überraschend groß, vor allem wenn man bedenkt, dass RASAR in jedem Schritt nur eine H von 1 bis 2 überwindet (Stepwise überhaupt nur eine H von 1).

	$\dot{p}$	BIC
A	27	2 938,923
B	27	2 939,672
C	25	2 939,696
D	25	2 940,569
E	24	2 941,064

Tabelle 2: Modellgröße und BIC einiger mit RASAR erzeugten Modelle

	A	B	C	D	E
A	27	15	17	16	12
B	15	27	23	20	17
C	17	23	25	20	16
D	16	20	20	25	20
E	12	17	16	20	24

Tabelle 3: Anzahl der gemeinsamen Variablen der RASAR-Modelle. Die Diagonal-Einträge entsprechen der Modellgröße.

	A	B	C	D	E
A	0	24	18	20	27
B	24	0	6	12	17
C	18	6	0	10	17
D	20	12	10	0	9
E	27	17	17	9	0

Tabelle 4: Hamming-Distanz zwischen den RASAR-Modellen

	A	B	C	D	E
A	1,00	0,56	0,68	0,64	0,50
B	0,56	1,00	0,92	0,80	0,71
C	0,68	0,92	1,00	0,80	0,67
D	0,64	0,80	0,80	1,00	0,83
E	0,50	0,71	0,67	0,83	1,00

Tabelle 5: Overlap der RASAR-Modelle (Das Overlap-Maß wird auf Seite 55 beschrieben.)

3.4.2 RASAR in Kombination mit anderen Methoden

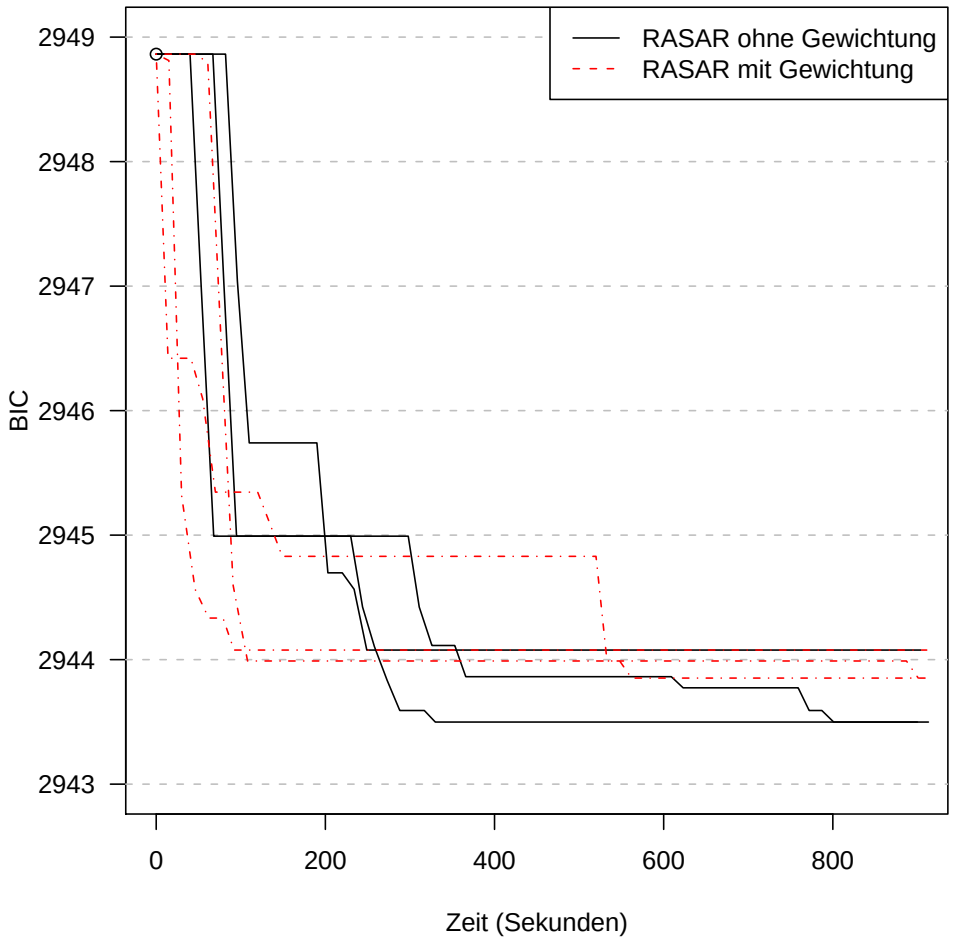

Abbildung 29: Die Methode RASAR mit dem Ergebnis der ResCor-Methode als Startmodell (ART-Daten): Die gefundenen Verbesserungen sind gering.

Die Methode RASAR kann mit anderen (vor allem schnelleren) Methoden kombiniert werden, indem das Resultat der Letzteren als Startmodell verwendet wird. Stepwise kommt dafür nicht in Frage, die Methoden sind einander zu ähnlich: Vorwärts- und Rückwärtsschritte führen beim Stepwise-Output per Definition zu keiner Verbesserung und das Austauschen von Variablen kann mit anderen Methoden (siehe Kapitel 3.2) effizienter ausgeführt werden. ResCor scheint sich für die Kombination mit RASAR dagegen anzubieten: Es ist sehr schnell und arbeitet ganz anders.

Die Performance dieser Kombination ist beim TOX-Datensatz durchaus zufrieden stellend (Abb. 30). Bei den ART-Daten sind die Verbesserungen (gegenüber vom Startmodell) aber nur sehr bescheiden (Abb. 29). Noch dazu sind sie insgesamt schlechter als die RASAR-Ergebnisse ohne Startmodell (Abb. 31)!

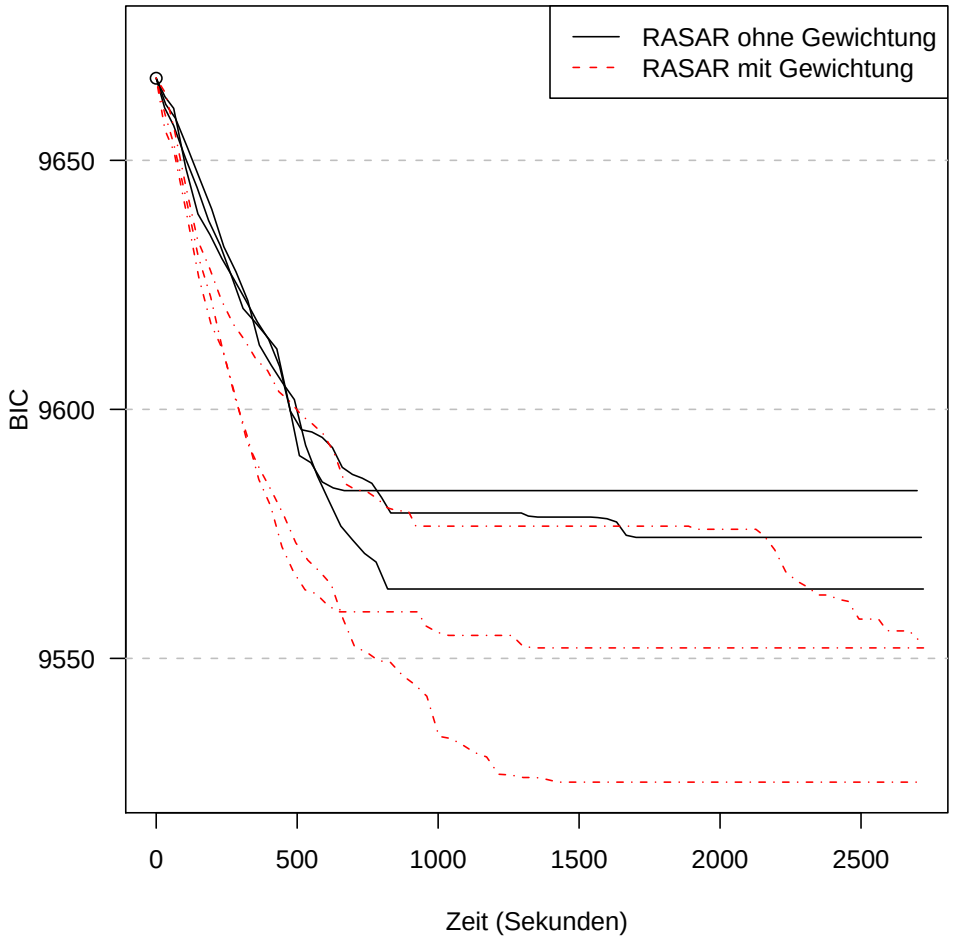

Abbildung 30: Die Methode RASAR mit dem Ergebnis der ResCor-Methode als Startmodell (TOX-Daten)

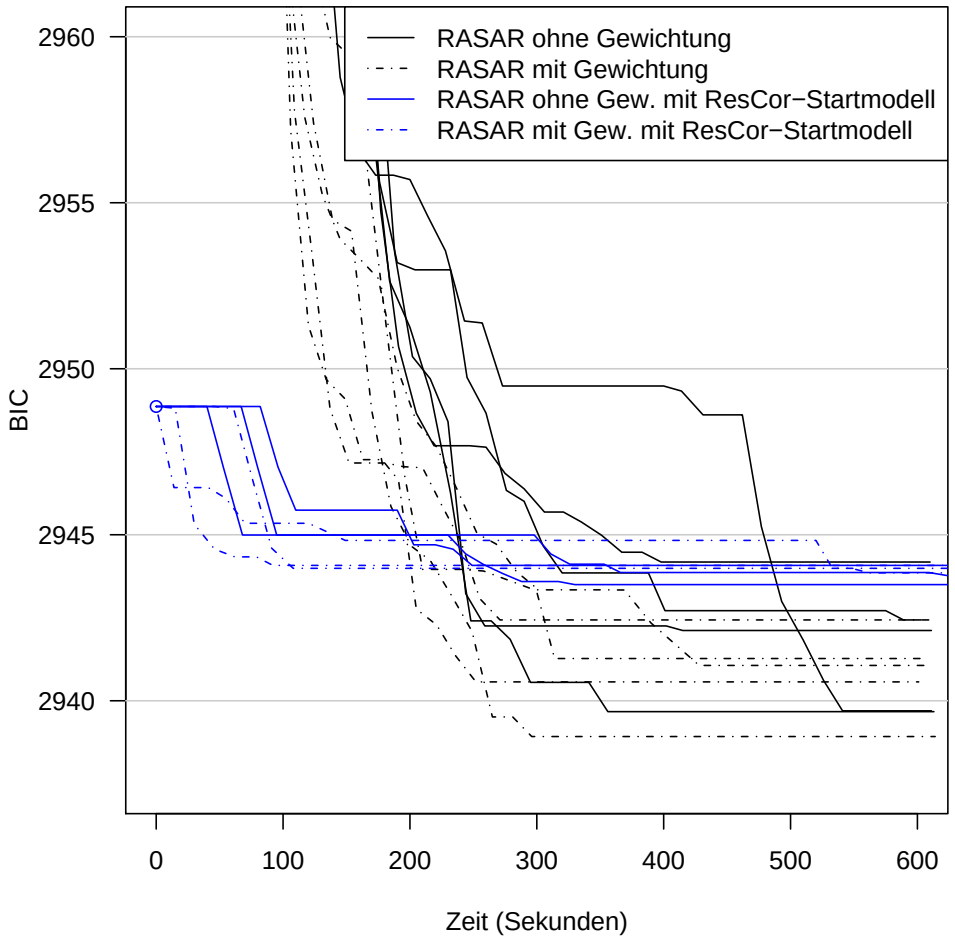

Abbildung 31: Vergleich der Methode RASAR ohne und mit ResCor-Startmodell: Das Ergebnis ist bei den ART-Daten ohne Startmodell besser.

3.5 FASS

FASS, *F*orward selection combined with *A*ll *S*ub*S*ets Regression, ist eine in [Liebminger, 2006] vorgeschlagene Methode.

FASS funktioniert durch das iterative Ausführen der folgenden Prozedur: Sei V die Menge aller Variablen und M die Menge aller Variablen, die sich im aktuellen Modell befinden. Jene Variablen, die nicht bereits im Modell sind, also die Menge $V \backslash M$, werden in I gleich große Blöcke mit g Variablen unterteilt (der letzte Block kann auch weniger als g Variablen haben). Seien $\Theta = \theta_1, \ldots, \theta_I$ die Indexmengen der Variablenblöcke. Für alle Mengen $m \subseteq M \cup \theta_i \quad i \in I$ wird ein neues Modell geschätzt und die Gütefunktion berechnet.

Mit anderen Worten: Die Variablen werden in gleich große Blöcke aufgeteilt und der Reihe nach zum Modell genommen, wobei auch immer alle Untermodelle berechnet werden.

Das beste Modell wird behalten und die Prozedur beginnt wieder von vorne, so lange bis in einem Schritt keine Verbesserung mehr gefunden wird oder die Maximalzeit erreicht wird.

Eigenschaften von FASS:

- FASS führt tendenziell zu kleinen Modellen.
- Das Ergebnis ist nicht eindeutig, also vom Zufall abhängig.
- In jedem Schritt werden auch Modelle berechnet, die schon in den vorigen Schritten gerechnet wurden.
- Je größer das Modell wird, desto länger dauert jeder einzelne Schritt.

```
1  fass <- function(y, x, startM, maxTime=3600, resname="fass00",
       maxPredictors = 50, addSize = 10) {
2      startTime <- as.integer(Sys.time())
3      k <- log(nrow(x)) # BIC
4      p <- ncol(x)
5      if(missing(startM)) {
6          startM <- rep(0, p)
7          startM[sample(1:p, size=20)] <- 1
8      }
9      require(leaps)
10     namesX <- names(x)
11     m      <- startM
12     m0     <- which(m == 0)
13
14     bestBic  <- AIC(lm(form(m, namesX), data=x), k=k)
15     bics     <- bestBic
```

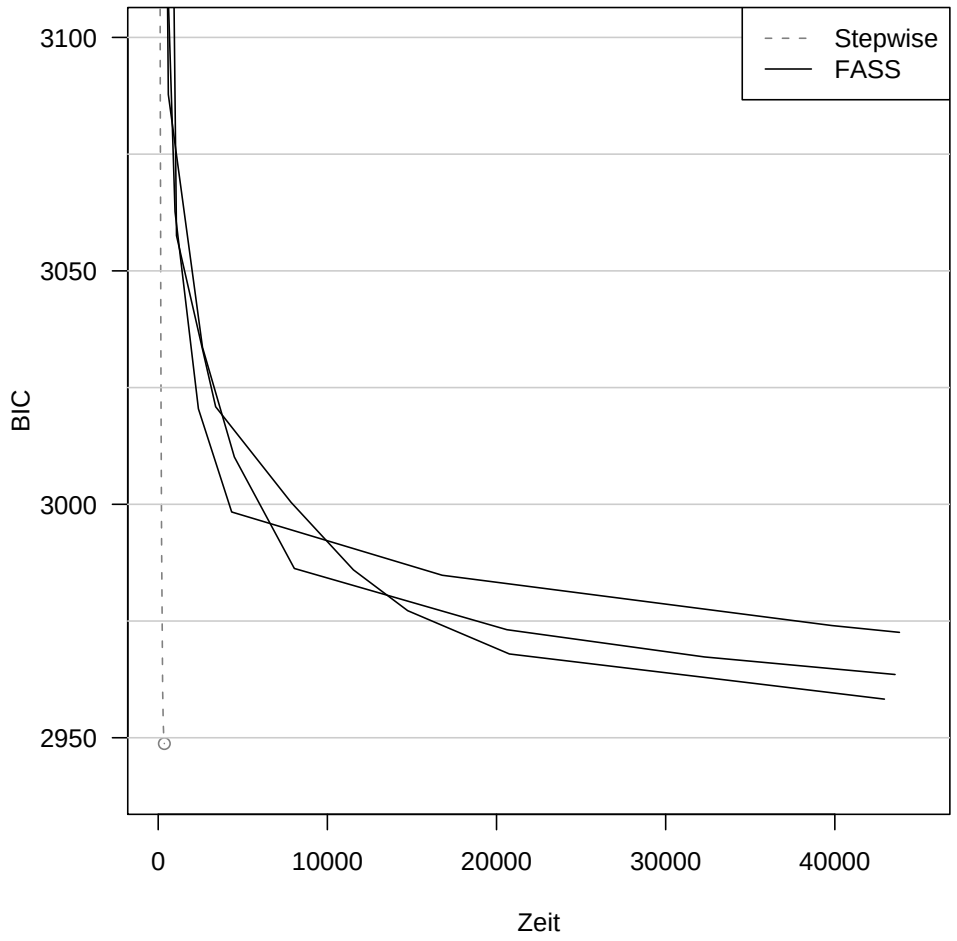

Abbildung 32: Die Methode FASS mit den ART-Daten

```
16      usedTime <- as.integer(Sys.time())-startTime
17      models   <- m
18
19      # outer loop
20      while(as.integer(Sys.time())-startTime < maxTime) {
21                  print(bestBic)
22          tmpModels <- list()
23          tmpBics   <- NULL
24
25          # inner loop
26          for(i in 1:ceiling(length(m0)/addSize)) {
27              tmpM <- m
28              tmpM[m0[((i-1)*addSize):min(length(m0),i*addSize)]] <- 1
29              res <- summary(regsubsets(x=x[,as.logical(tmpM)], y=y, nbest
                    =1, nvmax=maxPredictors, method="exhaustive", really.big=
```

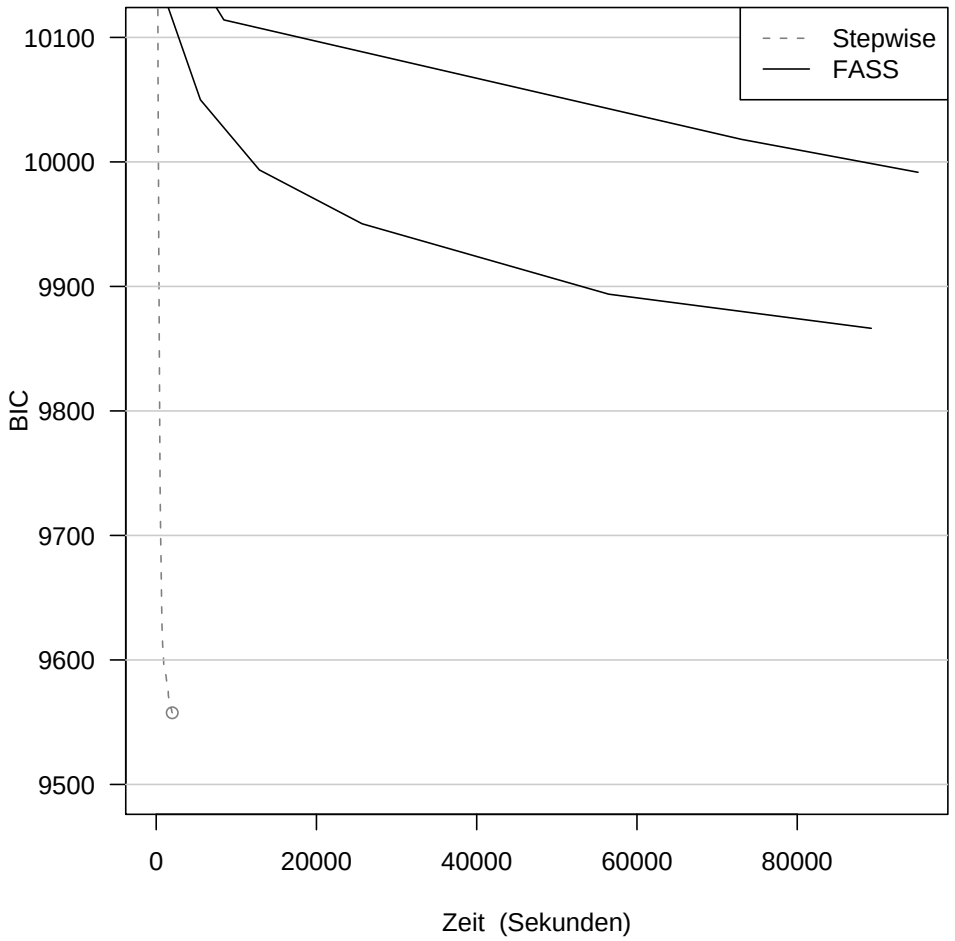

Abbildung 33: Die Methode FASS mit den TOX-Daten

```
          TRUE))
30
31        tmp <- which.min(res$bic)
32        pos <- length(tmpModels)+1
33        mN  <- rep(0,p)
34        mN[which(tmpM==1)[res$which[tmp,-1]]] <- 1
35        tmpModels[[pos]] <- mN
36        tmpBics[pos] <- AIC(lm(form(mN, namesX), data=x), k=k)
37
38        if(as.integer(Sys.time())-startTime > maxTime) break
39    }
40    # end inner loop
41
42    pos2 <- which.min(tmpBics)
43
```

```
44          if(all(tmpModels[[pos2]] == m) || tmpBics[pos2] >= bestBic) {
45              break
46          } else {
47              bestBic  <- tmpBics[pos2]
48              bics     <- c(bics, bestBic)
49              usedTime <- c(usedTime, as.integer(Sys.time())-startTime)
50              m <- tmpModels[[pos2]]
51              try(lines(usedTime, bics))
52
53              models <- rbind(models, m)
54              try(write.table(cbind(usedTime, bics, models), file=paste(
                    resname, ".txt", sep=""), col.names=FALSE, row.names=
                    FALSE))
55          }
56
57          m0 <- which(m==0)
58      }
59      #end outer loop
60      try(write.table(cbind(usedTime, bics, models), file=paste(resname, "
            .txt", sep=""), col.names=FALSE, row.names=FALSE))
61      return(list(usedTime=usedTime, bic=bics, models=models))
62  }
```

Listing 12: FASS

4 Vergleich der Ergebnisse

4.1 Stabilität der Variablenauswahl

In diesem Abschnitt werden ein paar ausgewählte Methoden dahingehend untersucht, ob sie bei variierenden Trainings-Daten stabile Ergebnisse liefern. Die folgenden Fragen sollen beantwortet werden:

1. Gibt es Methoden, die ähnliche Modelle finden, wenn verschiedene Teilmengen der Daten verwendet werden (Stabilität)?
2. Welche Modellgrößen ergeben sich bei den verschiedenen Heuristiken mit den verschiedenen Daten?
3. Sind die Modelle stärker von der Heuristik oder von den Daten abhängig?
4. Wie verhält sich der Prognosefehler unter den gegebenen Umständen?

Dabei wird folgendermaßen vorgegangen (siehe auch Abb. 34):

- Die Daten werden in drei gleich große Teile A, B und C geteilt. (Beim ART-Datensatz sind das 400, beim TOX-Datensatz 270 Beobachtungen pro Teil.)
- Die Daten $A \cup B$, $A \cup C$ und $B \cup C$ bilden jeweils ein Trainings-Set. C, B und A sind die dazugehörenden Test-Sets. Statt $A \cup B$ usw. wird im Folgenden einfach nur AB geschrieben.
- Mit jedem Trainings-Set wird die Variablenselektion durchgeführt (bei den ART-Daten mit den Methoden Stepwise, ResCor, RASAR und FASS, bei den TOX-Daten mit den Methoden Stepwise, ResCor und RASAR).
- Mit jedem Trainings-Set wird die Schätzung der Parameter für die gefundenen Modelle durchgeführt.
- Die Prognose für die Testdaten wird berechnet.
- Der quadrierte Prognosefehler wird festgestellt. (In diesem Fall wird die gesamte Verteilung betrachtet, nicht nur der Mittelwert.)

Als Maß für die Ähnlichkeit zweier Modelle wird das *Overlap-Maß* (O) verwendet:

$$O_{(a,b)} = \frac{\sum_{i=1}^{p} a_i b_i}{min(\sum_{i=1}^{p} a_i, \sum_{i=1}^{p} b_i)} \tag{6}$$

Dabei sind a und b 0-1-Vektoren, die jeweils ein Modell repräsentieren. *Overlap* ist also die Summe der in beiden Modellen befindlichen Variablen dividiert durch die Größe des kleineren Modells. Falls ein Modell eine Teilmenge des anderen ist, ist Overlap 1. Falls keine gemeinsame Variable existiert, ist Overlap 0.

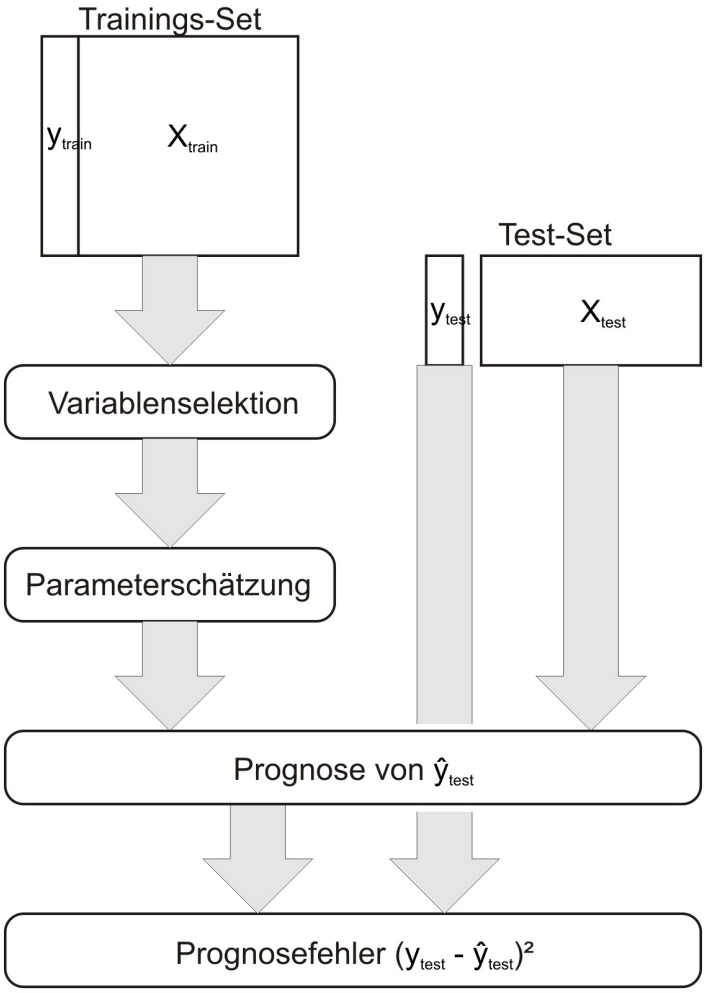

Abbildung 34: Vorgehensweise zur Evaluation der *Stabilität* der Heuristiken: Wie bei CV wird der Datensatz in ein Trainings- und ein Test-Set geteilt. Das Trainings-Set wird für die Variablenauswahl und für die Parameterschätzung verwendet. Diese Parameter werden dann auf das Test-Set angewendet mit dem der Prognosefehler berechnet wird.

4.1.1 Interpretation der Stabilität

Die *Modellgröße* $\dot{p}$ kann beim ART-Datensatz (Tabelle 6) als stabil bezeichnet werden: Innerhalb einer Methode gibt es kaum Unterschiede, auch zwischen den verschiedenen Methoden bewegt sich $\dot{p}$ immer zwischen 22 und 25. Beim TOX-Datensatz (Tabelle 10) führt ResCor zu größeren Modellen als Stepwise und RASAR. Auch innerhalb derselben Methode führen die verschiedenen Teil-Datensätze zu unterschiedlichen Modellgrößen, wobei diese aber generell viel größer sind als bei den ART-Daten.

Der *Overlap*, also die *Ähnlichkeit* zwischen den Modellen, kann bei den ART-Daten insgesamt als *gering* betrachtet werden. Durchschnittlich sind nur 34% der Variablen vom

Methode	Trainings-Datensatz	$\dot{p}$	$\bar{\dot{p}}$	s
Stepwise	BC	22	22,0	0,0
	AC	22		
	AB	22		
ResCor	BC	25	25,0	0,0
	AC	25		
	AB	25		
RASAR	BC	25	24,0	1,0
	AC	24		
	AB	23		
FASS	BC	23	23,3	0,6
	AC	24		
	AB	23		

Tabelle 6: Vergleich der Modellgröße bei verschiedenen Heuristiken und den Teil-Datensätzen *BC*, *AC* und *AB* (ART-Daten).

		Stepwise			ResCor			RASAR			FASS		
		BC	AC	AB	BC	AC	AB	BC	AC	AB	BC	AC	AB
Stepwise	BC	1,00	0,23	0,23	0,68	0,23	0,18	0,73	0,18	0,23	0,68	0,23	0,23
	AC	0,23	1,00	0,18	0,18	0,68	0,23	0,36	0,73	0,18	0,32	0,82	0,18
	AB	0,23	0,18	1,00	0,09	0,27	0,55	0,23	0,27	0,86	0,32	0,27	0,41
ResCor	BC	0,68	0,18	0,09	1,00	0,24	0,08	0,68	0,17	0,09	0,57	0,21	0,09
	AC	0,23	0,68	0,27	0,24	1,00	0,24	0,32	0,83	0,22	0,35	0,71	0,22
	AB	0,18	0,23	0,55	0,08	0,24	1,00	0,16	0,21	0,61	0,22	0,25	0,43
RASAR	BC	0,73	0,36	0,23	0,68	0,32	0,16	1,00	0,33	0,26	0,83	0,42	0,13
	AC	0,18	0,73	0,27	0,17	0,83	0,21	0,33	1,00	0,22	0,35	0,75	0,17
	AB	0,23	0,18	0,86	0,09	0,22	0,61	0,26	0,22	1,00	0,35	0,30	0,48
FASS	BC	0,68	0,32	0,32	0,57	0,35	0,22	0,83	0,35	0,35	1,00	0,35	0,17
	AC	0,23	0,82	0,27	0,21	0,71	0,25	0,42	0,75	0,30	0,35	1,00	0,26
	AB	0,23	0,18	0,41	0,09	0,22	0,43	0,13	0,17	0,48	0,17	0,26	1,00

Tabelle 7: Overlap-Maß der Modelle bei allen verglichenen Heuristiken und Trainings-Sets (ART-Daten). Die Werte innerhalb einer bestimmten Methode sind grau hinterlegt, in der Tabelle 8 sind die Mittelwerte angegeben. Die O-Werte für einen bestimmten Trainings-Datensatz sind blau hervorgehoben, die Durchschnittswerte sind in der Tabelle 9 angegeben.

kleineren Modell im größeren enthalten (Tabelle 7). Interessant ist aber, dass der Overlap bei gegebener Heuristik und verschiedenen Trainings-Sets (Tabelle 8) geringer ist, als bei verschiedenen Heuristiken und den gleichen Daten (Tabelle 9). Dies ist vor allem vor dem Hintergrund bemerkenswert, dass sich die verschiedenen Trainings-Daten sowieso jeweils

Methode	$\bar{O}$
Stepwise	0,21
ResCor	0,19
RASAR	0,27
FASS	0,26

Tabelle 8: Vergleich des mittleren Overlap bei verschiedenen Methoden (ART-Daten)

Teil-Datensatz	$\bar{O}$
BC	0,69
AC	0,75
AB	0,56

Tabelle 9: Vergleich des mittleren Overlap bei verschiedenen Trainings-Sets (ART-Daten)

zur Hälfte überschneiden und die Modellvoraussetzungen bei diesem Datensatz praktisch perfekt erfüllt sind (siehe Kap. 1.4).

Bei den TOX-Daten (Tabelle 11) ist der durchschnittliche Overlap mit 46% ein bisschen höher, wobei der O innerhalb der Methoden (Tabelle 8) größer ist als bei den ART-Daten. Der O innerhalb der Teil-Datensätze (Tabelle 9) ist etwas geringer als bei den ART-Daten. Nachdem der TOX-Datensatz über weniger Beobachtungen verfügt (TOX-Daten: 270 Beobachtungen pro Block, ART-Daten: 400 Beobachtungen pro Block), wäre eher ein größerer Einfluss der Daten zu erwarten gewesen.

Beim *Prognosefehler* ist kein signifikanter Unterschied, sowohl hinsichtlich der Methoden als auch der Trainings-Daten, festzustellen (siehe Abb. 35 und 36).

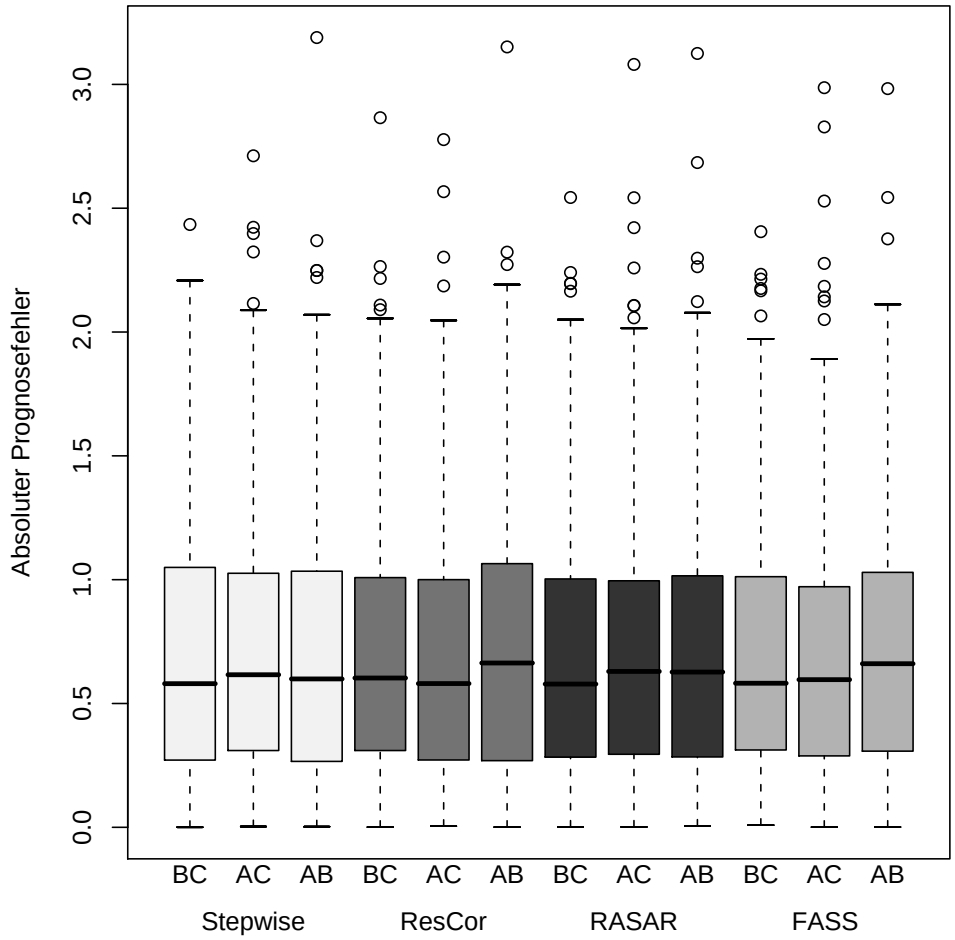

Abbildung 35: Prognosefehler bei verschiedenen Heuristiken und Teil-Datensätzen (ART-Daten). Zur besseren Interpretierbarkeit wird hier der absolute und nicht der quadrierte Prognosefehler dargestellt.

Methode	Trainings-Datensatz	$\dot{p}$	$\bar{\dot{p}}$	s
Stepwise	BC	49	55,3	6,5
	AC	55		
	AB	62		
ResCor	BC	72	76,0	4,6
	AC	75		
	AB	81		
RASAR	BC	59	61,0	3,5
	AC	59		
	AB	65		

Tabelle 10: Vergleich der Modellgrößen bei verschiedenen Heuristiken und Teil-Datensätzen (TOX-Daten)

		Stepwise			ResCor			RASAR		
		BC	AC	AB	BC	AC	AB	BC	AC	AB
Stepwise	BC	1,00	0,53	0,41	0,59	0,41	0,51	0,67	0,59	0,55
	AC	0,53	1,00	0,42	0,45	0,49	0,49	0,53	0,47	0,36
	AB	0,41	0,42	1,00	0,37	0,44	0,45	0,42	0,37	0,53
ResCor	BC	0,59	0,45	0,37	1,00	0,50	0,54	0,54	0,42	0,38
	AC	0,41	0,49	0,44	0,50	1,00	0,56	0,32	0,41	0,34
	AB	0,51	0,49	0,45	0,54	0,56	1,00	0,42	0,44	0,31
RASAR	BC	0,67	0,53	0,42	0,54	0,32	0,42	1,00	0,49	0,47
	AC	0,59	0,47	0,37	0,42	0,41	0,44	0,49	1,00	0,39
	AB	0,55	0,36	0,53	0,38	0,34	0,31	0,47	0,39	1,00

Tabelle 11: Overlap-Maß der Modelle bei verschiedenen Heuristiken und Teil-Datensätzen (TOX-Daten)

Methode	$\bar{O}$
Stepwise	0,45
ResCor	0,53
RASAR	0,45

Tabelle 12: Vergleich des mittleren Overlap bei verschiedenen Methoden (TOX-Daten)

Teil-Datensatz	$\bar{O}$
BC	0,60
AC	0,46
AB	0,43

Tabelle 13: Vergleich des mittleren Overlap bei verschiedenen Trainings-Sets (TOX-Daten)

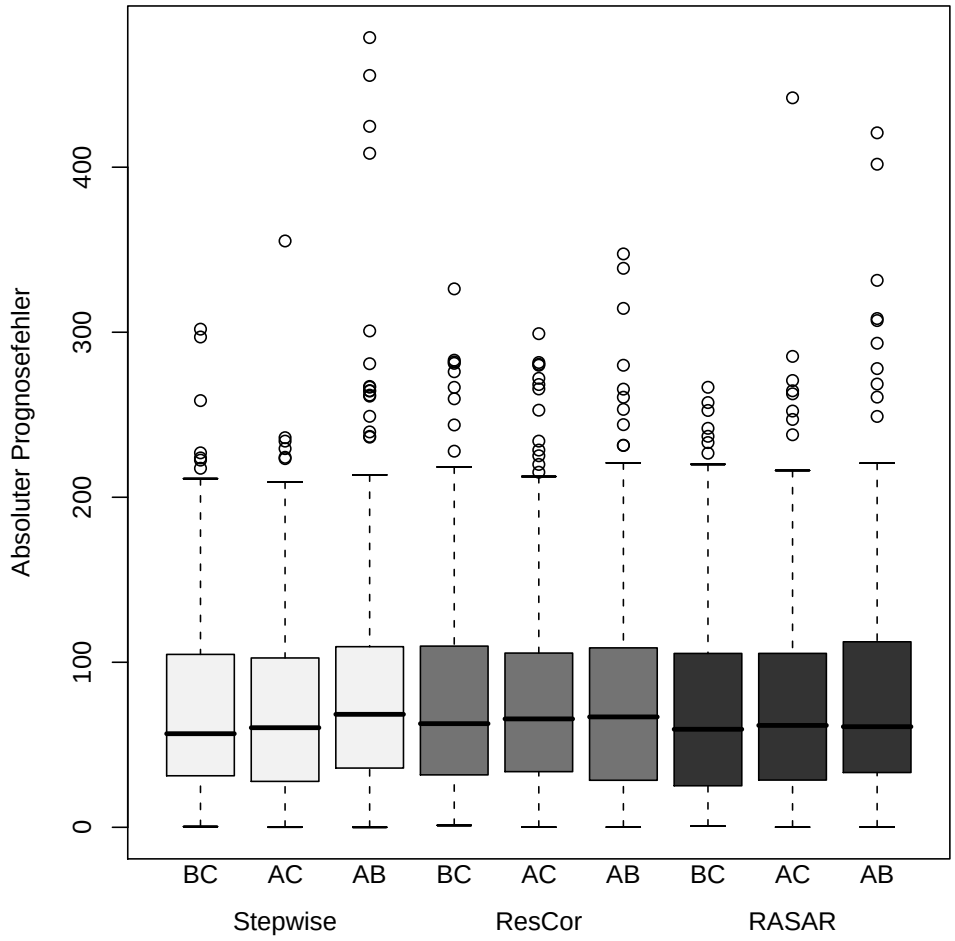

Abbildung 36: Absoluter Prognosefehler bei verschiedenen Heuristiken und Teil-Datensätzen (TOX-Daten)

4.2 BIC und Prognosefehler

In diesem Kapitel werden 8 Modelle, die mit 8 verschiedenen Methoden gefunden wurden, verglichen. Im Folgenden ist eine kurze Erläuterung zu den Modellen gegeben:

Stepwise0	ist das Resultat der Stepwise-Methode (siehe Kap. 1.5), wobei mit dem leeren Modell begonnen wurde.
Stepwise1	ist das Ergebnis der Stepwise-Methode mit dem vollen Modell als Startmodell.
ResCor	wurde mit der Methode ResCor (Kap. 3.1.2) gefunden.
Stepwise-RepRep	wurde duch die Methode RepRep (siehe Kap. 3.2) erzeugt, wobei das Stepwise0-Modell als Startmodell diente.
STAP	ist das Ergebnis der Methode STAP (Kap. 3.3).
RASAR	ist das beste Resultat von mehreren RASAR-Durchläufen (Kap. 3.4).
ResCor-Rasar	wurde ebenfalls aus mehreren RASAR-Prozessen ermittelt, wobei aber das ResCor-Modell als Start verwendet wurde (Kap. 3.4.2).
FASS	ist das beste von mehreren mit FASS erzeugten Modellen (Kap. 3.5).

Bei allen Methoden wurde das BIC-Kriterium optimiert. In den Tabellen 14 und 15 sind die Modellgröße $\dot{p}$, der BIC-Wert, der Prognosefehler und die Rechenzeit angegeben. Der Prognosefehler wurde durch 200-fach wiederholte Kreuzvalidierung mit der Methode `rcv` (Kapitel 5.5) mit Test-Set $= \frac{n}{10}$ ermittelt. Die Rechenzeit bezieht sich bei jenen Methoden, die kein eindeutiges Ergebnis liefern (RASAR und FASS), auf einen einzelnen Durchlauf.

In den Abbildungen 37 und 38 sind jeweils BIC und Prognosefehler bei den ART- und TOX-Daten gegenübergestellt. In Abb. 39 sind alle 4 Dimensionen, beide Kriterien mit beiden Datensätzen, verglichen.

	$\dot{p}$	BIC	Prognosefehler	Rechenzeit (sek)
Stepwise0	22	2 948,76	0,9792	364
Stepwise1	26	2 939,15	0,9774	115 321
ResCor	25	2 948,86	0,9762	4
Stepwise-RepRep	22	2 942,88	0,9783	4 883
STAP	20	2 948,48	0,9778	654
RASAR	27	2 939,16	0,9816	611
ResCor-Rasar	24	2 943,85	0,9772	905
FASS	23	2 958,27	0,9778	42 930

Tabelle 14: Übersicht über die Ergebnisse (Modellgröße, BIC, Prognosefehler und benötigte Rechenzeit) beim ART-Datensatz

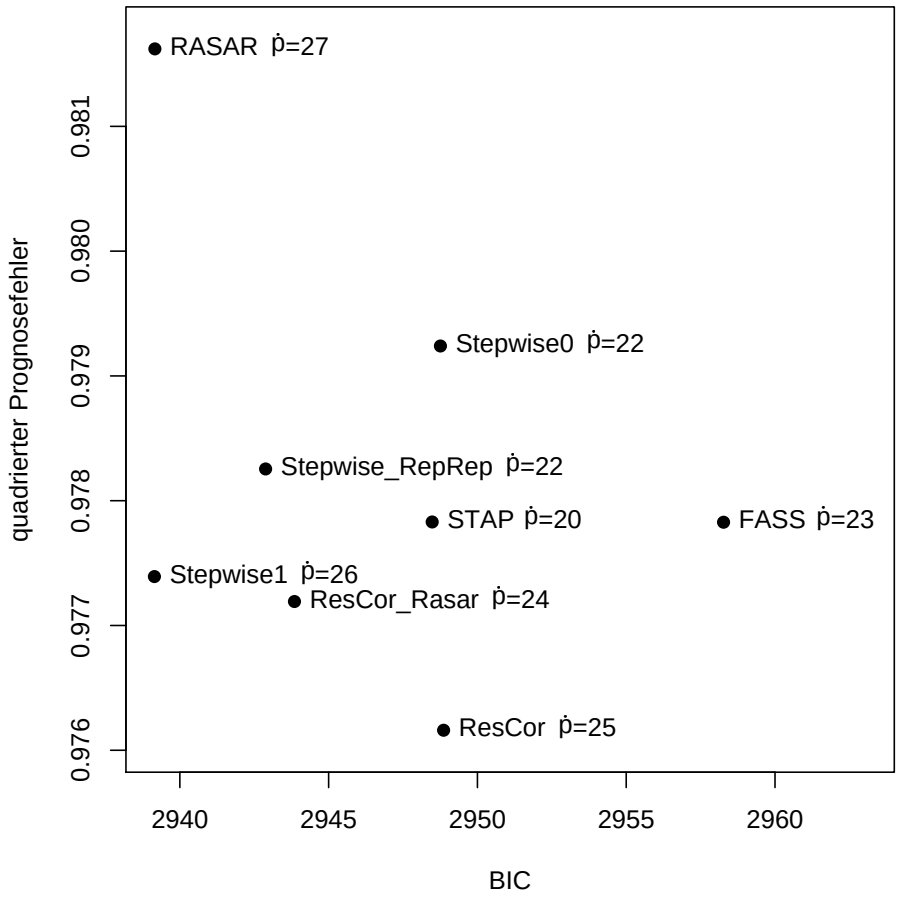

Abbildung 37: Ergebnisse ART-Daten

	$\dot{p}$	BIC	Prognosefehler	Rechenzeit (sek)
Stepwise0	69	9 557,57	12 534,59	2 002
Stepwise1	98	9 581,35	12 563,36	74 648
ResCor	74	9 666,49	12 512,21	11
Stepwise-RepRep	69	9 549,20	12 527,26	24 310
STAP	52	9 569,54	12 553,88	1 924
RASAR	62	9 525,82	12 440,02	1 341
ResCor-Rasar	77	9 525,15	12 593,83	1 436
FASS	36	9 866,35	12 499,61	89 238

Tabelle 15: Übersicht über die Ergebnisse beim TOX-Datensatz

4.2.1 Interpretation BIC und Prognosefehler

- Die untersuchten Methoden führen sowohl bei BIC als auch beim Prognosefehler zu sehr ähnlichen Ergebnissen.

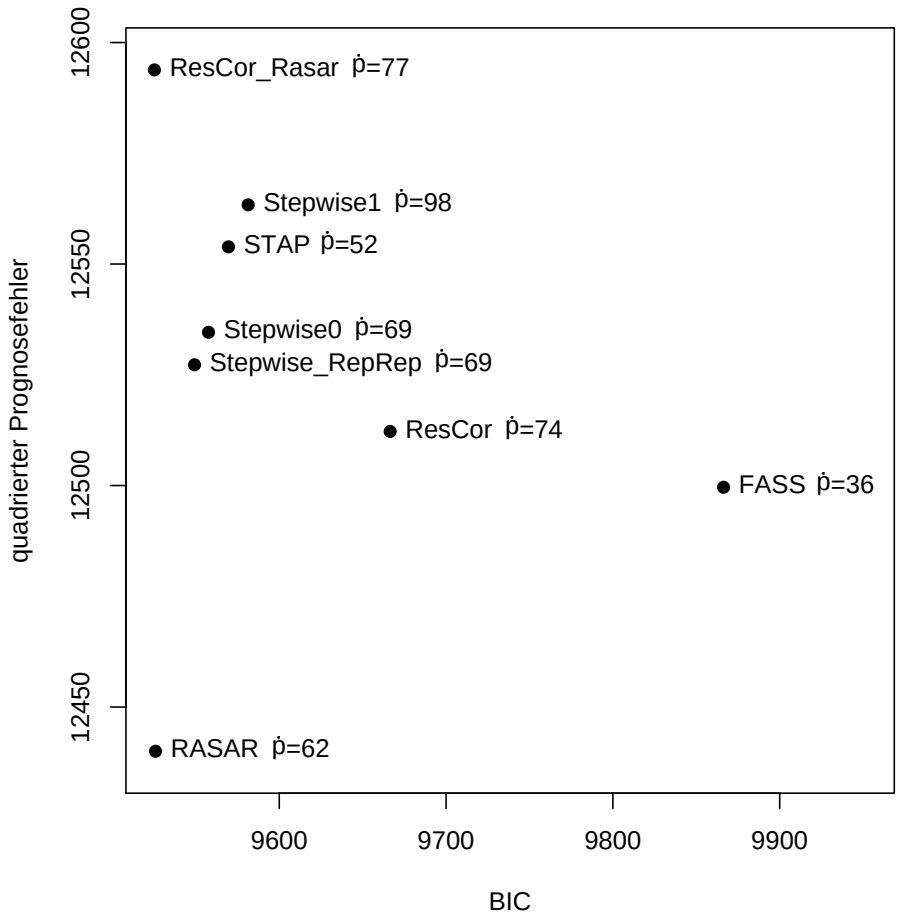

Abbildung 38: Ergebnisse TOX-Daten

- Zwischen BIC und Prognosefehler (Abb. 37 und 38) ist in diesem Fall kein Zusammenhang feststellbar. Dies liegt vermutlich daran, dass sich die Ergebnisse hinsichtlich BIC kaum unterscheiden. (Vergleicht man unterschiedlichere Modelle, z. B. die Modellsequenz von Kap. 1.3.5, so ist der Zusammenhang selbstverständlich vorhanden.)
- Die lange Rechenzeit mancher Methoden scheint sich, zumindest bei den untersuchten Datensätzen, nicht zu lohnen.
- Interessant für weitere Untersuchungen scheint mir die Kombination von verschiedenen, sehr schnellen Heuristiken zu sein.

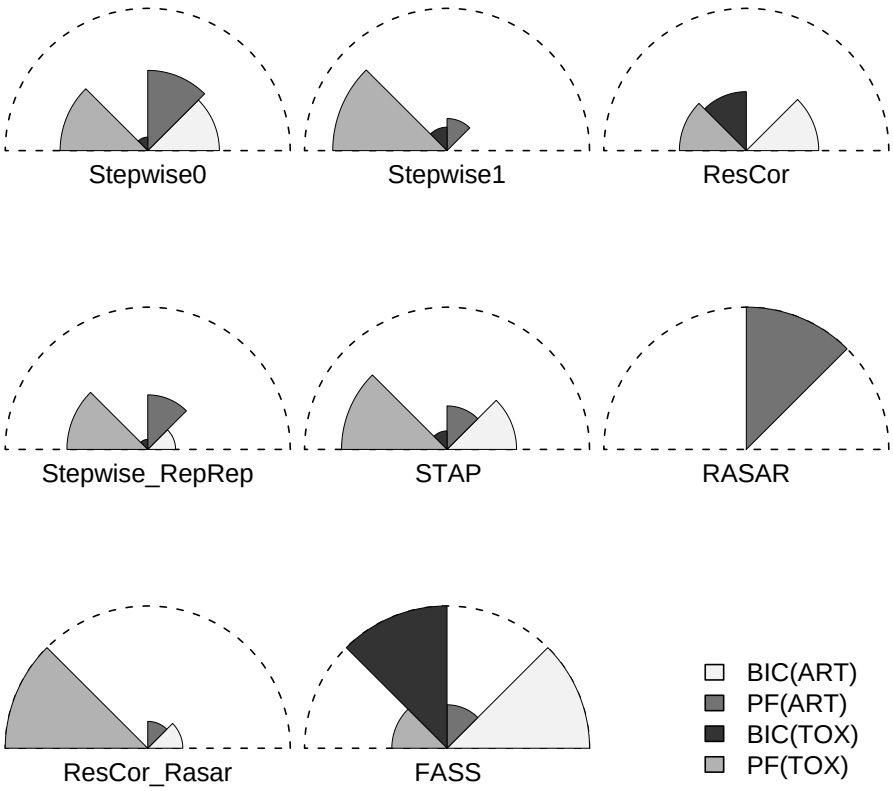

Abbildung 39: BIC und Prognosefehler für beide Datensätze: Keine Methode dominiert in allen Dimensionen. (Beide Kriterien, BIC und Prognosefehler sind zu minimieren. *Kleine* Werte, dargestellt als kleine Sektoren, sind also gut.)

5 Hilfsmethoden

In diesem Kapitel werden die in den verschiedenen Heuristiken verwendeten Hilfsmethoden erklärt und deren R-Code wiedergegeben.

5.1 Generieren der Formelnotation mit `form`

Eine anschauliche und einfach zu speichernde Form, ein bestimmtes Modell zu repräsentieren, ist die eines 0-1-Vektors (ein Vektor, der nur aus Nullen und Einsen besteht). Dabei bedeutet 1 an der Stelle i, dass die i-te Variable im Modell enthalten ist, 0 bedeutet, dass diese Variable nicht im Modell enthalten ist.

Die Methode `form()` wandelt einen 0-1-Vektor (oder einen logischen Vektor) in ein R-Objekt der Klasse `formula` um. Als zweites Argument wird ein Vektor mit den Variablennamen von X übergeben.

Ist `m` z. B. ein Vektor: `(1 0 1 1 0)` und die Variablennamen von X sind `x1` ... `x5`, so ergibt der Aufruf von `form(m, names(X))`:

$$\texttt{y} \sim \texttt{x1 + x3 + x4}.$$

```
1 form <- function(a, namesX) {
2     if(sum(a)==0) return(as.formula("y~1"))
3     as.formula(paste("y~", paste(namesX[as.logical(a)], collapse="+"),
          sep=""))
4 }
```

Listing 13: `form` erzeugt `formula`-Objekte

5.2 Schrittweise Variablenauswahl mit `stepwise`

Die im Folgenden beschriebene Methode `stepwise()` führt im Wesentlichen einen wiederholten Aufruf der R-Methode `step` durch, wobei aber nicht nur die maximale Schrittanzahl, sondern auch die maximale Rechenzeit vorgegeben werden kann. Außerdem werden für jeden Schritt die benötigte Zeit, der Wert des Gütekriteriums und das Modell gespeichert. Die Rückgabe ist also nicht ein einzelnes Modell, sondern eine ganze Sequenz. Die Eingabeparameter sind:

`y`	y
`x`	X
`k`	$k = \begin{cases} 2 & \text{für AIC} \\ log(n) & \text{für BIC} \end{cases}$
`startM`	Vektor der Länge p, der das Startmodell repräsentiert
`maxTime`	die maximal erlaubte Rechenzeit in Sekunden
`direction`	`"both"`: Es kann pro Schritt eine Variable entfernt oder ergänzt werden. `"backward"`: Variablen können nur entfernt werden. `"forward"`: Variablen können nur hinzugenommen werden.
`resname`	Das Ergebnis wird (zusätzlich zur Rückgabe der Funktion) in ein Textfile mit dem angegebenen Namen geschrieben.
`maxsteps`	maximale Anzahl der Schritte

Der Rückgabewert ist eine Liste mit folgenden Elementen:

`usedTime`	Der i-te Eintrag in diesem Vektor gibt an, wie viele Sekunden seit dem Aufruf der Funktion bis zur fertigen Berechnung des i-ten Modells vergangen sind.
`bic`	die Werte für das Gütekriterium (BIC oder AIC) für alle Modelle
`models`	Matrix mit allen gefundenen Modellen, wobei jede Zeile ein Modell darstellt.

```
1  stepwise <- function(y, x, k, startM, maxTime=3600, direction="both",
       resname="stepres00", maxsteps=500) {
2      startTime <- as.integer(Sys.time())
3      if(missing(k)) k <- log(nrow(x)) # BIC
4      dat    <- data.frame(y, x)
5      p      <- ncol(x)
6      namesX <- names(x)
7
8      bic      <- rep(Inf, maxsteps)
9      usedTime <- rep(NA, maxsteps)
10     models   <- matrix(NA, maxsteps, p)
11
12     if(missing(startM)) startM <- rep(0, p)
13     lmTmp  <- lm(y ~ 1, data=dat)
14     lmTmp  <- update(lmTmp, form(startM, namesX))
15     lmFull <- lm(form(rep(1,p), namesX), data=dat)
16     lm0    <- lm(form(rep(0,p), namesX), data=dat)
17
18     for(i in 1:maxsteps) {
19         lmTmp <- step(object=lmTmp, scope=list(upper=lmFull, lower=lm0)
               , direction=direction, trace=0, steps=1, k=k)
20
21         usedTime[i] <- as.integer(Sys.time())-startTime
22         bic[i]      <- AIC(lmTmp, k=k)
```

```
23          models[i,]  <- as.numeric(is.element(el=namesX, set=names(lmTmp$
                coef)))
24          if(i >= 2) {
25              if(bic[i] >= bic[i-1]) break
26          }
27          if(as.integer(Sys.time())-startTime >= maxTime) break
28          try(write.table(cbind(usedTime, bic, models), file=paste(resname
                , ".txt", sep=""), col.names=FALSE, row.names=FALSE))
29      }
30     if(any(is.na(models[,1]))) {
31          tmp      <- which(is.na(models[,1]))[1]-1
32          bic      <- bic[1:tmp]
33          usedTime <- usedTime[1:tmp]
34          models   <- models[1:tmp,]
35      }
36      if(length(bic) > 1) {
37          if(bic[length(bic)] >= bic[length(bic)-1]) {
38              tmp      <- length(bic)-1
39              bic      <- bic[1:tmp]
40              usedTime <- usedTime[1:tmp]
41              models   <- models[1:tmp,]
42          }
43      }
44      write.table(cbind(usedTime, bic, models), file=paste(resname, ".txt"
            , sep=""), col.names=FALSE, row.names=FALSE)
45      return(list(usedTime=usedTime, bic=bic, models=models))
46  }
```

Listing 14: `stepwise` ruft `step` wiederholt auf und speichert die Rechenzeit und die Zwischenergebnisse

5.3 Kreuzvalidierung mit `cv`

Die Methode `cv()` ist eine Umsetzung der Kreuzvalidierung (*Cross Validation*) zur Ermittlung des mittleren Prognosefehlers. Die Argumente sind außer y und X noch ein 0-1-Vektor, der ein Modell repräsentiert (siehe Kapitel 5.1), und der Anteil der Daten, der zur Prognose verwendet werden soll (*train*).

$$train \in [(\dot{p}+2)/n, (n-1)/n]$$

Die Daten werden in einen Trainings- und einen Testdatensatz aufgeteilt. Dabei werden zwei Fälle unterschieden:

$train \geq 0.5$ In diesem Fall werden mehrere möglichst gleich große Blöcke gebildet, so dass ein Block den Umfang $n(1 - train)$ hat. Jeder dieser Blöcke ist einmal der Testdatensatz, die restlichen Daten sind jeweils der Trainingsdatensatz. Mit dem Trainingsdatensatz wird das Modell geschätzt, und es werden die $\hat{y}$-Werte für den Testdatensatz berechnet. In Abbildung 40 ist eine Illustration für den Fall $train = 2/3$ gegeben.

$train < 0.5$ Hier werden die Daten in Blöcke der Größe $n \cdot train$ geteilt. Jeder Block bildet einmal den Trainingsdatensatz, wobei die Schätzung von y nicht für die gesamten restlichen Daten, sondern nur für einen weiteren Block durchgeführt wird. In Abbildung 41 wird $train = 1/3$ dargestellt. Jeder Block ist einmal Test- und einmal Trainingsdatensatz.

Nachdem alle $\hat{y}$ berechnet wurden, wird

$$\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

als Ergebnis zurückgegeben (mittlerer quadrierter Prognosefehler).

Im Allgemeinen kann davon ausgegangen werden, dass der Prognosefehler kleiner wird, wenn der Anteil der Trainingsdaten größer wird.

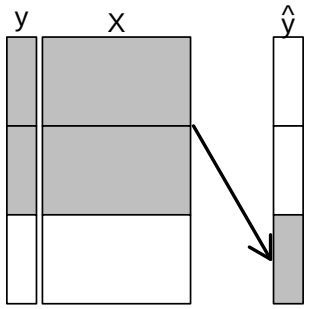

Abbildung 40: Trainings- und Testblöcke mit $train = 2/3$

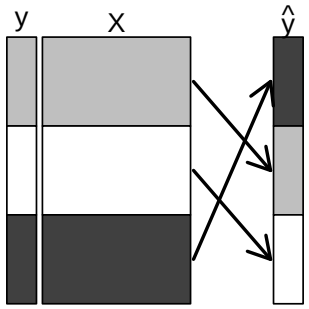

Abbildung 41: Trainings- und Testblöcke mit $train = 1/3$

```
cv <- function(y, x, model, train) {
    tmp    <- sample(length(y))
    y      <- y[tmp]
    namesx <- names(x)[as.logical(model)]
    x      <- as.data.frame(x[tmp, as.logical(model)])
    names(x) <- namesx
    n <- nrow(x)
    if(missing(train)) train <- 0.75
    if(train > (n-1)/n || train*n < sum(model)+2) {
        stop("falsches train-argument")
    }
    res <- rep(NA, n)
    gr  <- sort(1:round(n/min(n*train, n*(1-train))) + rep(0,n))

    for(i in 1:max(gr)) {
        if(train >= 0.5) {
            tes <- gr==i
            tra <- gr!=i
        } else {
            tra <- gr==i
            tes <- ifese(i==max(gr), gr==1, gr==(i+1))
        }

        xTrain <- as.data.frame(x[tra,])
        names(xTrain) <- namesx
        yTrain <- y[tra]
        xTest  <- as.data.frame(x[tes,])
        names(xTest) <- namesx
        yTest  <- y[tes]

        lmt      <- lm(yTrain ~ ., data=xTrain)
        pred     <- predict(object=lmt, type="response", newdata=xTest)
        res[tes] <- (yTest - pred)^2
    }

    return(sum(res)/n)
}
```

Listing 15: cv für die Kreuzvalidierung

5.4 Wiederholte Kreuzvalidierung mit `rep.cv`

Diese Methode gibt den Mittelwert von mehreren Aufrufen von `cv()` (siehe Kapitel 5.3) zurück. Die Argumente sind gleich wie bei `cv()`, zusätzlich kann mit dem Argument `times` die Anzahl der gewünschten Wiederholungen eingestellt werden.

Dass das Mitteln mehrerer `cv()`-Aufrufe besonders bei kleinem Trainings-Set günstig ist, wird in Abbildung 42 veranschaulicht. Mit einem Datensatz, $n = 200$, $p = 96$ und einem Modell mit $\dot{p} = 27$ wurden jeweils 30 Aufrufe von `cv()` mit den `train`-Argumenten 1/3, 1/2, 2/3, 3/4 und 31/32 durchgeführt. Die Ergebnisse sind als Boxplot dargestellt.

```
1 rep.cv <- function(y, x, model, train, times=5) {
2     tmp <- rep(NA, times)
3     for(i in 1:times) {
4         tmp[i] <- cv(y=y, x=x, model=model, train=train)
5     }
6     return(mean(tmp))
7 }
```

Listing 16: `rep.cv` führt wiederholte Kreuzvalidierung durch.

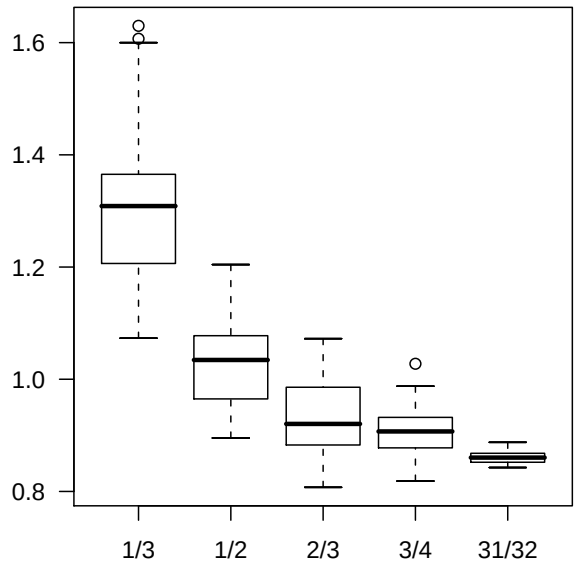

Abbildung 42: Die Varianz der `cv`-Resultate nimmt bei kleinen `train`-Werten zu.

5.5 Prognosefehler mit oder ohne Kreuzvalidierung mit `rcv`

`rcv` ist eine weitere Methode zum (experimentierenden) Berechnen des Prognosefehlers. Anders als bei den in Kapitel 5.3 und 5.4 beschriebenen Methoden wird hier bereits der auf die benötigten Variablen reduzierte X-Datensatz übergeben, das Argument `model`

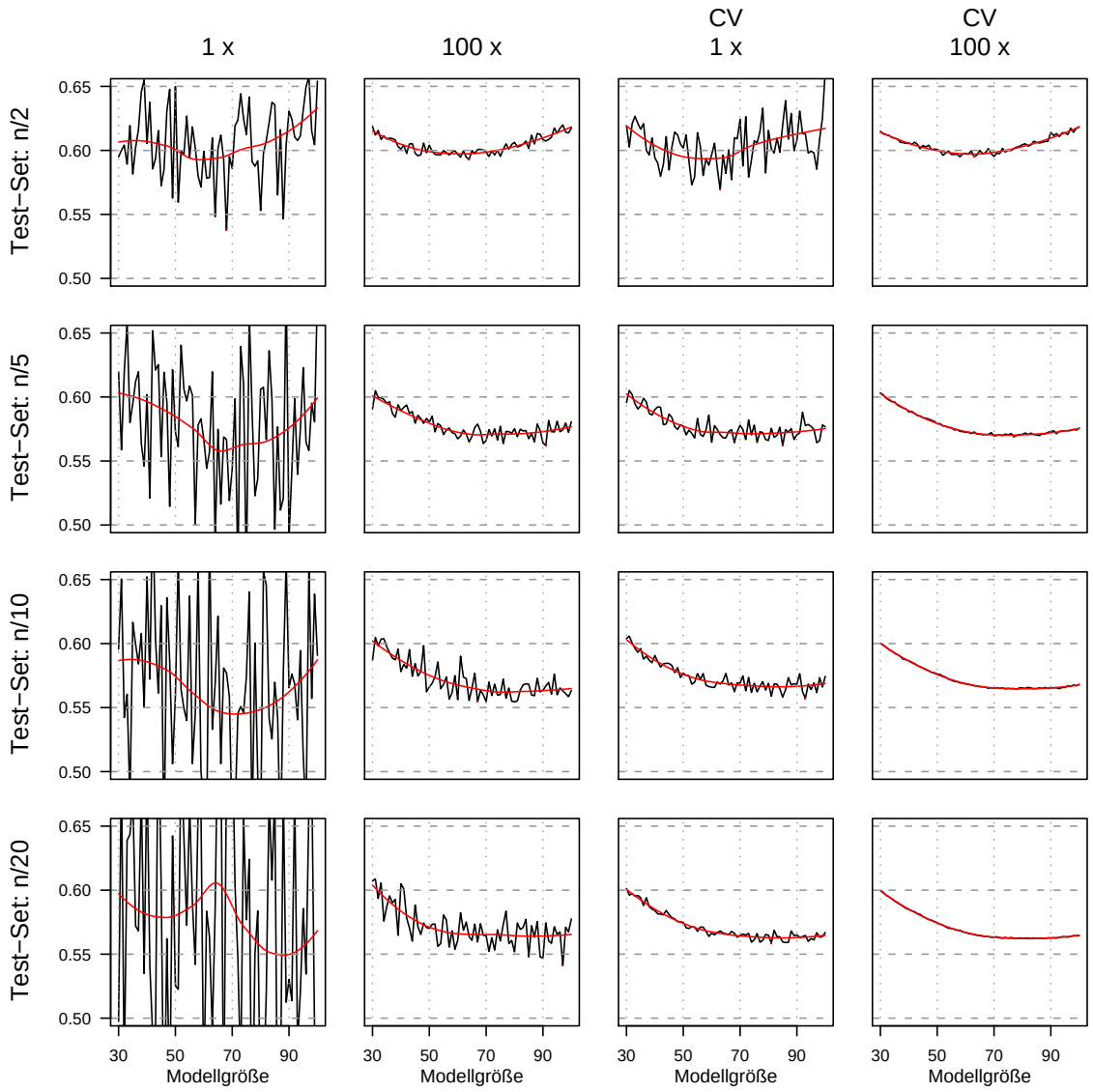

Abbildung 43: Kreuzvalidierung führt zu stabileren Ergebnissen als die wiederholte Berechnung ohne CV.

kann daher entfallen. Mit dem Argument `blocks` wird festgelegt, in wie viele Blöcke der Datensatz zu teilen ist. (Die Blöcke sind abwechselnd das *Test-Set*, wobei jeweils die restlichen Blöcke als *Trainings-Set* verwendet werden.) Um den Code einfach zu halten, wird davon ausgegangen, dass `blocks` ein Teiler von n ist. Mit `cvi` wird festgelegt, ob Kreuzvalidierung durchgeführt werden soll (default) oder nicht. Falls `cvi==FALSE`, wird der Prognosefehler nur für *einen Block* berechnet. Unabhängig davon, ob CV ausgeführt wird oder nicht, kann mit `repe` die Anzahl der Wiederholungen eingestellt werden.

Beispiel: Mit `blocks=10, cvi==TRUE, repe=1` werden insgesamt genau n Werte für den Prognosefehler berechnet, der Mittelwert wird als Ergebnis zurückgegeben. Mit `blocks=10, cvi==FALSE, repe=10` werden ebenso n Werte berechnet, wobei aber Be-

obachtungen öfters im Test-Set sein können!

In Abbildung 43 ist die Methode `rcv` mit verschiedenen Einstellungen vorgeführt. Die evaluierten Modelle sind eine Sequenz von Step-Forward-Modellen, die Modellgröße ist auf der x-Achse aufgetragen. Da die Modelle sehr ähnlich sind (jedes kleinere Modell ist eine Teilmenge des größeren Modells), sollte die Kurve der Prognosefehler möglichst glatt sein.

Vergleicht man z. B. die Grafiken in der dritten Zeile, so ist die Varianz in der zweiten Spalte (Prognose ohne CV mit 100 Wiederholungen: $100 \times 120 = 12000$ Werte) größer als in der dritten Spalte (Prognose mit CV: $1 \times 1200 = 1200$ Werte).

```
1  rcv <- function(x, y, blocks=10, cvi=TRUE, repe=1) {
2      if(length(y) %% blocks != 0) stop("Das Argument blocks muss ein
           Teiler von n sein")
3      g <- sort(rep(1:blocks, length(y)/blocks))
4      tmp <- NULL
5      for(v in 1:repe) {
6          ind <- sample(nrow(x))
7          xi <- x[ind,]
8          yi <- y[ind]
9          for(u in 1:ifelse(cvi,blocks,1)) {
10             xtrain  <- xi[g!=u,]
11             xtest   <- xi[g==u,]
12             ytrain  <- yi[g!=u]
13             ytest   <- yi[g==u]
14             lm1     <- lm(ytrain ~ ., data=xtrain)
15             pred1   <- predict(lm1, newdata=xtest)
16             err1    <- (pred1-ytest)^2
17             tmp     <- c(tmp, err1)
18         }
19     }
20     return(mean(tmp))
21 }
```

Listing 17: `rcv` berechnet den Prognosefehler mit oder ohne Kreuzvalidierung mit beliebig vielen Wiederholungen.

5.6 Leave-One-Out-Kreuzvalidierung mit `loo`

Leave-One-Out-Kreuzvalidierung ist ein Sonderfall der im vorigen Kapitel beschriebenen Methode. Dabei stellt jede Beobachtung einmal das Test-Set dar. Das Ergebnis ist eindeutig, Wiederholungen erübrigen sich daher.

Leave-One-Out-Kreuzvalidierung führt im Allgemeinen zu einer zu optimistischen Einschätzung des Prognosefehlers (siehe Abb. 44).

Die Methode `loo` führt zum gleichen Ergebnis wie die Methode `rcv` mit den Parametern `blocks=n`, `cvi=TRUE` und `repe=1`, ist aber schneller.

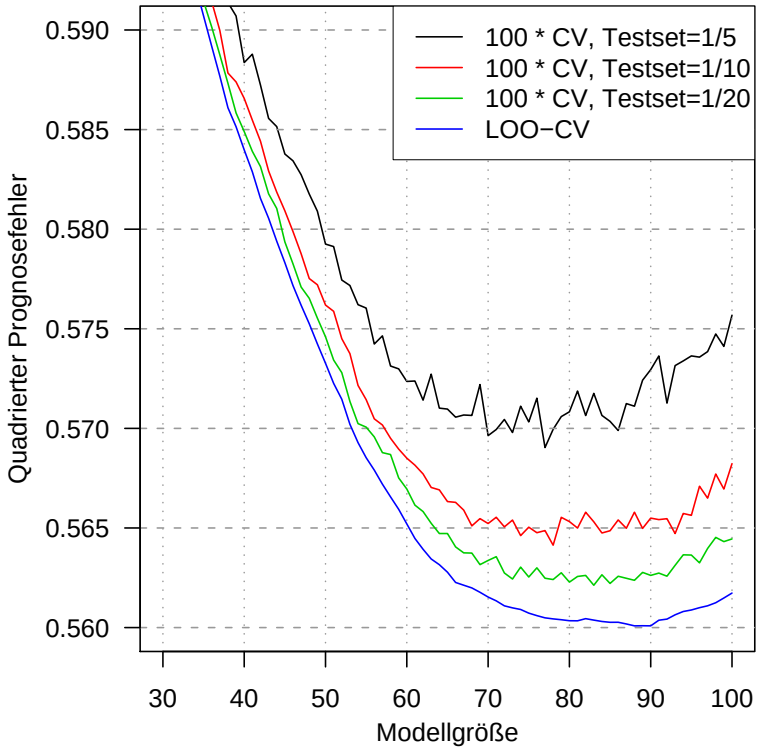

Abbildung 44: Vergleich Leave-One-Out-Kreuzvalidierung mit k-facher Kreuzvalidierung: Je kleiner das Test-Set, umso geringer der Prognosefehler und die Varianz. Bei LOO-CV sind keine Wiederholungen nötig, weil das Ergebnis eindeutig ist.

```
1 loo <- function(x, y) {
2     tmp <- NULL
3     for(i in 1:nrow(x)) {
4         lm1     <- lm(y[-i] ~ ., data=x[-i,])
5         pred1   <- predict(lm1, newdata=x[i,])
6         err1    <- (pred1-y[i])^2
7         tmp     <- c(tmp, err1)
8     }
```

```
9       return(mean(tmp))
10  }
```

Listing 18: `loo` führt Leave-One-Out-Kreuzvalidierung durch.

5.7 Entfernen von Variablen mit `rmAlmostConst`

Bei manchen Datensätzen kommt es vor, dass einige Variablen für fast alle Beobachtungen denselben Wert (z. B. 0 oder *NA*) aufweisen. Es kann sein, dass man solche Variablen entfernen möchte, nämlich aus folgenden Gründen:

- Es ist eine im Sinne der Rechenzeit billige Möglichkeit, den Datensatz zu reduzieren.
- Es könnte bei *Cross Validation* Probleme geben: Wenn eine Variable im Trainings-Set nur einen konstanten Wert hat, kann der Koeffizient nicht geschätzt werden.

Die Methode `rmAlmostConst()` nimmt als Argument X und k und retourniert X um jene Variablen reduziert, die höchstens k von „Standardwert“ verschiedene Werte besitzen.

Es sei erwähnt, dass es theoretisch durchaus möglich ist, dass diese Variablen auch einen sinnvollen Beitrag zu einem Modell leisten können. Die Entscheidung, ob und bis zu welchem k man Variablen entfernt, hängt vom Datensatz und von der konkreten Aufgabenstellung ab.

```
1   rmAlmostConst <- function(x, k = 3) {
2       x <- as.data.frame(x)
3       n <- nrow(x)
4       if(n <= k*2) stop("n muss groesser als 2*k sein!")
5       j <- rep(TRUE, ncol(x))
6       for(i in 1:ncol(x)) {
7           temp <- sort(x[,i], na.last=TRUE)
8           if(is.na(temp[floor(n/2)+1])) {
9               if(sum(is.na(temp)) >= n-k) j[i] <- FALSE
10          } else {
11              if(sum(temp == temp[floor(sum(!is.na(temp))/2)+1], na.rm=
                    TRUE) + sum(is.na(temp)) >= n-k) {
12                  j[i] <- FALSE
13              }
14          }
15      }
16      return(as.data.frame(x[,j]))
17  }
```

Listing 19: `rmAlmostConst` reduziert den Datensatz

5.8 Entfernen linear abhängiger Variablen mit `rmLinDep`

Falls X linear abhängige Variablen enthält, kann dies bei manchen Algorithmen zu Problemen führen, nämlich falls diese gleichzeitig in ein Modell aufgenommen wurden: Die Koeffizienten können dann nicht mehr geschätzt werden. Mit der Methode `rmLinDep` werden die „überflüssigen“ Variablen entfernt. Argument ist X (als Matrix oder Data-Frame), Rückgabewert ist das eventuell reduzierte X (als Data-Frame).

```
1  rmLinDep <- function(x, y=NULL) {
2      x  <- as.data.frame(x)
3      if(is.null(y)) y <- rep(1,nrow(x))
4      j  <- rep(TRUE, dim(x)[2])
5      al <- alias(y ~ ., data=x)
6      n1 <- dimnames(al$Complete)[[1]]
7      for(i in 1:dim(x)[2]) {
8          if(is.element(dimnames(x)[[2]][i], n1)) j[i] <- FALSE
9          if(is.element(paste("`",dimnames(x)[[2]][i],"`",sep=""), n1)) j[
               i] <- FALSE
10     }
11     x <- (x[,j])
12     return(as.data.frame(x))
13 }
```

Listing 20: `rmLinDep` entfernt Linearkombinationen.

Besser ist es, nach sachlichen Überlegungen zu entscheiden, *welche* der abhängigen Variablen entfernt werden. In diesem Fall hilft der Befehl `alias`, die Abhängigkeiten aufzudecken.

Abbildungsverzeichnis

1 Ablaufdiagramm für schrittweise Verfahren 9
2 Überblick über einige Kriterien 10
3 CV-Prognosefehler einer Stepwise-Modellsequenz (ART-Daten) 14
4 CV-Prognosefehler einer Stepwise-Modellsequenz (Der hier verwendete TOX-Datensatz wird in Kapitel 1.4 beschrieben) 15
5 s^2 der ersten 10 Hauptkomponenten der ART-Daten 17
6 χ^2-Plot der ART-Daten 17
7 Verteilung der VIFs der ART-Daten 18
8 s^2 der ersten 10 Hauptkomponenten der TOX-Daten 18
9 χ^2-Plot der TOX-Daten 19
10 Verteilung der VIFs der TOX-Daten 19
11 Vergleich leeres und volles Startmodell 20
12 Lokales Optimum 23
13 Graph der möglichen Modelle 23
14 Die ersten 10 Hauptkomponenten von X 27
15 R^2 verschiedener Stepwise-Modelle. 28
16 Der Prognosefehler, abhängig von der Modellgröße. Für die Ermittlung des Prognosefehlers wurde die Methode `rep.cv` von Kapitel 5.4 verwendet. 28
17 Vergleich einfacher Methoden mit den ART-Daten 30
18 Vergleich einfacher Methoden mit den TOX-Daten 31
19 Entwicklung des BIC-Kriteriums bei der Methode *ResCor*, ART-Daten . 33
20 Entwicklung des BIC-Kriteriums bei der Methode *ResCor*, TOX-Daten . 33
21 Die Methoden `repRep` und `swap` mit den ART-Daten 34
22 Die Methoden `repRep` und `swap` mit den TOX-Daten 35
23 Vergleich der `stepwise`- und `stap`-Modelle anhand der ART-Daten . . . 38
24 Vergleich der `stepwise`- und `stap`-Modelle anhand der TOX-Daten . . . 39
25 Die Methode RASAR mit den ART-Daten 41
26 Endergebnisse der `rasar`-Methode mit den ART-Daten 41
27 Die Methode RASAR mit den TOX-Daten 42
28 Endergebnisse der `rasar`-Methode mit den TOX-Daten 42
29 RASAR mit ResCor-Startmodell, ART-Daten 48
30 RASAR mit ResCor-Startmodell, TOX-Daten 49
31 RASAR mit und ohne Startmodell 50
32 FASS mit den ART-Daten 52
33 FASS mit den TOX-Daten 53
34 Vorgehensweise zur Evaluation 56
35 Verteilung des Prognosefehlers (ART-Daten) 59
36 Verteilung des Prognosefehlers (TOX-Daten) 61
37 Ergebnisse ART-Daten 63
38 Ergebnisse TOX-Daten 64
39 Ergebnisse beider Datensätze 65
40 Trainings- und Testblöcke mit $train = 2/3$ 69

41 Trainings- und Testblöcke mit $train = 1/3$ 69
42 Varianz der `cv`-Resultate 71
43 CV führt zu stabileren Ergebnissen 72
44 Vergleich LOO-CV und CV 74

Tabellenverzeichnis

1 Vergleich der schrittweisen Vorwärtsselektion und Rückwärtselimination . 21
2 Größe und BIC einiger RASAR-Modelle 46
3 Anzahl gemeinsamer Variablen der RASAR-Modelle 47
4 Hamming-Distanz der RASAR-Modelle 47
5 Overlap der RASAR-Modelle 47
6 Vergleich der Modellgrößen bei verschiedenen Heuristiken (ART-Daten) . 57
7 Overlap-Maß der Modelle (ART-Daten) 57
8 Mittlerer O nach Methoden (ART-Daten) 58
9 Mittlerer O nach Trainings-Set (ART-Daten) 58
10 Vergleich der Modellgrößen bei verschiedenen Heuristiken (TOX-Daten) . 59
11 Overlap-Maß der Modelle (TOX-Daten) 60
12 Mittlerer O nach Methoden (TOX-Daten) 60
13 Mittlerer O nach Trainings-Set (TOX-Daten) 60
14 Ergebnis-Übersicht (ART-Daten) 62
15 Ergebnis-Übersicht (TOX-Daten) 63

Listingsverzeichnis

1 AIC 11
2 BIC 12
3 Stepwise: lokales Optimum 21
4 Die Methode `artdata` für die Erzeugung künstlicher Datensätze. 25
5 Beispiel-Anwendung von `artdata` 26
6 Die Methode `resCor` 33
7 Die Methode `rep1` für systematisches Vertauschen 35
8 Die Methode `repRep` ruft `rep1` wiederholt auf 36
9 Die Methode `swap` 37
10 Die Methode `stap` kombiniert `stepwise` und `swap`. 39
11 Die Methode `rasar` 44
12 FASS 51
13 `form` erzeugt `formula`-Objekte 66
14 `stepwise` für den wiederholten Aufruf von `step` 67
15 `cv` für die Kreuzvalidierung 70
16 `rep.cv` führt wiederholte Kreuzvalidierung durch. 71
17 `rcv` berechnet den Prognosefehler mit oder ohne Kreuzvalidierung 73

18 `loo` führt Leave-One-Out-Kreuzvalidierung durch. 74
19 `rmAlmostConst` reduziert den Datensatz 75
20 `rmLinDep` entfernt Linearkombinationen. 76

Literaturverzeichnis

John Fox. *car: Companion to Applied Regression*, 2007. URL `http://www.r-project.org`, `http://socserv.socsci.mcmaster.ca/jfox/`. R package version 1.2-2.

Andreas Liebminger. *Stable isotope dynamics in precipitation and groundwater: determination and assessment of changes over the last 40 years*. PhD thesis, Vienna University of Technology, Austria, 2006.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2007. URL `http://www.R-project.org`. ISBN 3-900051-07-0.

Thomas Lumley und Alan Miller. *leaps: regression subset selection*, 2007. URL `http://www.r-project.org`, `http://cran.r-project.org/web/packages/leaps/`. R package version 2.7.