Flexibilisierung des modellbasierten Boosting durch Resampling und Regularisierung

Ludwig-Maximilians-Universität München

Fakultät für Mathematik, Informatik und Statistik

Bachelor Statistik

Bachelorarbeit

26.02.2020

Autor
Fabian Obster

Betreuer
Prof. Dr. Christian Heumann

Inhaltsverzeichnis

Kapitel 1

Einleitung

Eine große Schwierigkeit in der Statistik ist es, ein Modell bei sehr vielen Variablen anzupassen. Häufig sind in diesem Fall auch einige Variablen korreliert und die Modellschätzung wird bei vielen Methoden instabil. Es wurden mehrere Schätzalgorithmen eingeführt, um dieser Problematik entgegenzuwirken. $\mathcal{L}^1$ Regularisierung (Lasso Regression) oder $\mathcal{L}^2$ Regularisierung (Ridge Regression) Regularisierung sind häufig verwendete Methoden. Eine andere nützliche Mothode ist das modellbasierte Boosting [10]. Hier ist das Ziel, ein Modell sequenziell zu optimieren, indem immer nur eine Variable aufgenommen wird. Dadurch, dass man den Schätzalgorithmus frühzeitig abbricht, kann ein sparsames Modell angepasst werden, das einen $\mathcal{L}^1$ Regularisierungseffekt aufweist, da nur die für die Verlustfunktion wichtigsten Variablen in das Modell einfließen. Ein der $\mathcal{L}^2$ Regularisierung ähnlicher Effekt wird dadurch erzeugt, dass die geschätzten Parameter nur mit einer fixen Lernrate aufgenommen werden und durch das frühzeitige Stoppen der Effekt kleiner ist, als wenn man das Modell beispielsweise über Fischer Scoring anpasst. Zusätzlich kann bei Boosting Verfahren immer noch ein weiterer Regularisierungsterm in der Verlustfunktion angegeben werden, um eine weitere Regularisierung zu erzielen.

Ziel dieser Arbeit ist es, den Schätzalgorithms des modellbasierten Boosting flexibler zu gestalten, und weitere Regularisierungseffekte zu erzielen. Um diese Effekte zu erzielen, werden Bootstrap Verfahren (Bagging), wie aus dem maschinellen Lernen, beispielsweise dem Random Forest, [2] in jeder Boosting Iteration verwendent um mehrere Teilmodelle zu einem Modell zu aggregieren. Für jedes Teilmodell soll nur eine bestimmte Anzahl zufällig ausgewählter Variablen verwendet werden, um einen ähnlichen Effekt wie bei dem Random Forest zu erzielen.

Das Modell "rmboostïst in R für lineare Modelle implementiert. Charakteristische Eigenschaften des Schätzalgorithmus werden anhand simulierter Daten gezeigt. Die Hauptergebnisse sind, dass rmboost bei den simulierten Daten schneller zu der optimalen Lösung konvergiert. Außerdem deutet vieles darauf hin, dass der Bias, der durch das frühzeitige Stoppen entsteht, leicht geringer ist als bei anderen Regularisierungsmethoden. Insgesamt hat man bei rmbooost die Möglichkeit mehrere Regularisierugnsmethoden gleichzeitig zu verwenden und ist somit flexibler, eine differenziertere Lösung für den Bias Varianz Konflikt zu finden. Das wird dadurch erreicht, dass man zusätzliche Hyperparameter, wie die Anzahl der Bootstrap-Stichproben oder Anzahl der zufällig auszuwählenden Variablen, zur Verfügung hat. Außerdem können die nicht zur Schätzung der Teilmodelle verwendeten Observationen dazu genutzt werden, den Vorhersagefehler des Modells zu bestimmen.

Kapitel 2

Datenbasis

Zum Vergleich von rmboost mit anderen Methoden wurden mehrere Datensätze simuliert. Jede Simulation soll einen anderen in der Realität vorkommende Fall betrachten. Einflussgrößen werden in allen Fällen eine Normalverteilung unterstellt und dementsprechend simuliert. Zur Manipulation der Daten wurde das R Paket "dplyr"verwendet [20]. Für die Simulation multivariat normalverteilter Zufallsgrößen wurde das "MASS"Paket verwendet [18]

2.1 Datensatz 1

Folgende Variablen wurden simuliert mit $n = 100$.

- $(\mathbf{predictor.1}_i)_{i\in\{1,\dots,n\}} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mu = 0, \sigma^2 = 1)$.
- $(\mathbf{predictor.2}_i)_{i\in\{1,\dots,n\}} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mu = 0, \sigma^2 = 16)$.
- $(\mathbf{predictor.3}_i)_{i\in\{1,\dots,n\}} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mu = 0, \sigma^2 = 9)$.
- $(\epsilon_i)_{i\in\{1,\dots,n\}} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mu = 0, \sigma^2 = 0.5)$.
- $(\mathbf{target}_i)_{i\in\{1,\dots,n\}} = \text{predictor.1}_i + \text{predictor.2}_i + \epsilon_i$.

2.2 Datensatz 2

Der zweite Datensatz entspricht Datensatz 1, nur dass der Fehlerterm eine Standardabweichung von 4 hat

$$(\epsilon_i)_{i\in\{1,\dots,n\}} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mu = 0, \sigma^2 = 16).$$

Somit hat die Zielvariable ein deutlich höheres Rauschen, was die Modellschätzung unsicherer macht.

2.3 Datensatz 3

Dieser simulierte Datensatz enthält keinerlei Struktur zwischen Prädiktoren und der Zielvariable. Wir setzen $n = 100$. Für $j \in \{1, 2, 3, 4\}$ simulieren wir

$$\mathbf{predictor.j}_{i\in\{1,\dots,n\}} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mu = 0, \sigma^2 = 1),$$

sowie die Zielgröße,

$$\mathbf{target}_{i\in\{1,\dots,n\}} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mu = 0, \sigma^2 = 1),$$

welche unabhängig von den Prädiktoren ist.

2.4 Datensatz 4

Dieser simulierte Datensatz ist geprägt von einer hohen Korrelation der ünabhängigen"Variablen. Wir setzen $n = 500$. Für $j \in \{1, 2, 3\}$ simulieren wir

$$\mathbf{predictor.j}_{i \in \{1,\dots,n\}} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mu = 0, \sigma^2 = 1),$$

mit der Kovarianz

$$\begin{pmatrix} 1 & 0.99 & 0.99 \\ 0.99 & 1 & 0.99 \\ 0.99 & 0.99 & 1 \end{pmatrix}$$

sowie den davon unabhängigen Fehlerterm,

$$\epsilon_{i \in \{1,\dots,n\}} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mu = 0, \sigma^2 = 1),$$

welche unabhängig von den Prädiktoren ist.
Die abhängige Variable wird wie folgt definiert

$$(\mathbf{target}_i)_{i \in \{1,\dots,n\}} = \text{predictor.1}_i + \text{predictor.2}_i + \text{predictor.2}_i + 1 + \epsilon_i.$$

Kapitel 3

Resampling-Methoden

Generell kann man Resampling-Methoden in zwei Klassen unterscheiden: modellbasierte Verfahren und modellfreie Verfahren. Bei modellbasierten Verfahren existiert bereits Sekundärinformation über die Verteilung, aus der die ursprünglichen Daten stammen.
Aus diesen Verteilungsannahmen können dann weitere Observationen generiert werden. Falls die Verteilung stetig ist, so kann die Wahrscheinlichkeitsdichte verwendet werden, um neue Observationen zu ziehen. Diese Methode ist jedoch nicht zur Quantifizierung des Vorhersagefehlers geeignet, da implizit zur Ziehung der neuen Stichprobe angenommen wurde, dass man das Modell schon kennt. Das würde dann einer Postdiktion entsprechen.

Eine weitere Anwendung von Resampling-Methoden ist das Schätzen der Verteilung von unbekannten Parametern [16]. Mit dieser Verteilung können dann statistische Tests oder Konfidenzintervalle konstruiert werden, ohne eine Annahme über die Verteilung zu treffen. Insbesondere muss in diesem Fall keine Normalverteilung angenommen werden. Häufig wird dieses Verfahren angewendet, um die Unsicherheit von Regressionskoeffizienten anzugeben. Jedoch kann auf die gleiche Art und Weise die Verteilung des Vorhersagefehlers angegeben werden. Somit bekommt man nicht nur Aussagen über den Fehler, sondern auch über den Fehler von dem Fehler.

Die verwendeten Resampling-Methoden finden dann Anwendung, wenn neue Daten aus der gewünschten Verteilung nicht einfach generiert werden können und keine von den Daten unabhängigen Testdaten in großem Umfang vorliegen. Falls eine endliche Population mit Vollerhebung vorliegt, dann sind die hier beschriebenen Methoden nicht zielführend, da der Vorhersagefehler eines Modells keine Generalisierung braucht, denn in diesem Fall siegen schon alle Informationen vor.

Die Gemeinsamkeit aller Resampling-Methoden ist, dass die vorhandenen Daten immer in Trainingsdaten und Testdaten aufgeteilt werden. Einige Modelle benötigen zusätzlich noch eine Anpassung von globalen Parametern. Diese sind zum Beispiel bei neuralen Netzen wichtig, da sich bei einem solchen nicht konvexen Optimierungsproblem die Lernrate auf die lokal optimalen Parameter des Modells auswirken. In solchen Fällen werden Resampling-Methoden häufig verwendet, um eine Überanpassung der Modelle durch globale Parameter zu vermeiden. In dem ersten Teil dieser Arbeit werden diese globalen Parameter nicht berücksichtigt.

Auf die folgende Art und Weise lässt sich allgemein eine Stichprobe in Trainingsdaten und Testdaten unterteilen [13].

Sei $S^{(n)} = (S_1^{(n)}, ..., S_n^{(n)})$ eine binäre Zufallsvariable mit n Realisationen, sowie $D \in \mathbb{R}^{n \times p}$ eine Stichprobe. So kann D in den Trainingsdatensatz, charakterisiert durch

$$\Big\{ i \in \{1, ...n\} | S_i^{(n)} = 0 \Big\},$$

sowie den Testdatensatz, charakterisiert durch

$$\Big\{ i \in \{1, ...n\} | S_i^{(n)} = 1 \Big\},$$

aufgeteilt werden. Je nach Verteilung von $S^{(n)}$ erhält man unterschiedliche Resampling-Methoden. Sei $\tilde{p}$ der Anteil der Observationen in dem Testdatensatz. Je größer $\tilde{p}$, desto aussagekräftiger sind die Testdaten und desto weniger aussagekräftig sind die Trainingsdaten.

3.1 Datentrennung

Bei einem im Voraus festgelegten Anteil $\tilde{p}$ werden die Daten in Trainingsdaten und Testdaten aufgeteilt. Falls $\tilde{p} = \frac{1}{2}$ und $k = 2$, so kann man von einem Spezialfall der Kreuzvalidierung sprechen, wie in 3.2 beschrieben. Häufig ist es aber wünschenswerter den Trainingsdatensatz größer zu wählen, damit das Modell mit größerer Sicherheit geschätzt werden kann.

1. **Vorteile** Ein Vorteil dieser Methode gegenüber allen anderen ist, dass das Modell nur einmal angepasst werden muss. Das macht den Prozess nicht nur einfacher, sonder auch rechnerisch weniger aufwendig. Ein weiterer Vorteil ist, dass die Evaluation genau für das spezifizierte Modell mit allen Parametern durchgeführt wird und nicht nur für eine Klasse von Modellen, wie bei der k fachen Kreuzvalidierung mit $k \geq 2$.

2. **Nachteile** Es können zwei potentielle Arten von Bias auftreten.
   Einerseits kann eine Observation entweder nur zur Anpassung oder Evaluation des Modells beitragen, nicht jedoch beides. Zudem wird sowohl die Wahl des Modells als auch die Variablenselektion ausschließlich durch einen kleineren Datensatz als dem Ursprünglichen bestimmt. Deshalb sollte bestenfalls Variablenselektion und Modellwahl aufgrund inhaltlicher Überlegungen stattfinden und nicht aufgrund der Trainingsdaten. Dadurch dass auch der Testdatensatz kleiner ist als der Ursprüngliche, ist die Schätzung des Vorhersagefehlers mit einer gewissen Unsicherheit behaftet.

3.2 k fache Kreuzvalidierung

Das Prinzip der k-fachen Keruzvalidierung ist, dass die Daten in eine Partition der Mächtigkeit $k \in \mathbb{N}$ zerlegt werden. Die Aufteilung erfolgt zufällig. Optimaler Weise besteht die Partition aus gleich großen Teilen. Jeweils $k - 1$ Teildaten dienen als Trainingsdaten und deren Kompliment als Testdatensatz. Diese Aufteilung wird so oft durchgeführt, bis jeder Teildatensatz einmal als Testdatensatz verwendet worden ist. Ein Modell wird also genau k Mal evaluiert. Jedoch unterscheiden sich die Modelle, da die Parameter des Modells von den Testdaten abhängen. Falls die Anzahl der Observationen ein Vielfaches von k ist, so entspricht $\tilde{p}$ gleich $\frac{1}{k}$. Ansonsten stimmt die Gleichheit nur ungefähr. Je nach Betrachtung erhöht k den Bias und verringert ihn gleichzeitig.

Je größer k, desto mehr Schätzungen für den Fehler gibt es und desto geringer die Varianz. Jedoch ist

zu beachten, dass die Schätzungen der Fehler nicht unabhängig von einander sind, da sich die Testdaten zu $\frac{k-2}{k}$ überschneiden. Berechnet man also die Varianz des Schätzers für den Vorhersagefehler, ohne die Varianzstruktur zu berücksichtigen, erhält man eine verzerrte Schätzung. [1]

Der Rechenaufwand steigt in der Größenordnung $\mathcal{O}(k)$, da das Modell k Mal angepasst und evaluiert werden muss. Die Rechenzeit sollte vor allem bei komplexeren Modellen nicht vernachlässigt werden. Jedoch kann die Rechenzeit aufgrund der Linearität gut abgeschätzt werden. Außerdem ist es möglich, die Modelle parallel anzupassen. Das liegt daran, dass die Teilschritte nicht aufeinander aufbauen. Die Parallelisierung kann die Rechenzeit erheblich reduzieren.

3.3 Leave one out Kreuzvalidierung (LOOCV)

LOOCV kann als Spezialfall der k fachen Kreuzvalidierung aufgefasst werden mit $k = n$ und $\tilde{p} = \frac{1}{k}$. Wobei n der Anzahl der Observationen entspricht.

Bei der Variablenselektion eignet sich LOOCV nur bedingt, da jede Anpassung einem anderen Modell entspricht und die Evaluation dann nur auf einem einzigen Datenpunkt basiert, was nicht sonderlich aussagekräftig ist. Modelle werden dann aufgrund von Ausreißern diskriminiert [4].
Diese Evaluationsmethode ist computationell sehr aufwendig, da das Modell n Mal angepasst werden muss. Wie bei der Kreuzvalidierung steigt der Rechenaufwand linear in $k = n$. Falls viele Observationen zur Verfügung stehen, ist LOOCV möglicherweise nicht die geeignetste Methode; vor allem bei komplexeren Modellen.

3.4 Monte Carlo Kreuzvalidierung (MCCV)

Bei der MCCV werden die Daten N Mal in Trainingsdaten und Testdaten partitioniert. Dabei kann der Anteil $\tilde{p}$ auch wieder festgelegt werden, je nachdem ob der Fokus eher auf der Anpassung oder der Evaluierung liegt. Für jede der N Partitionen wird dann der Vorhersagefehler berechnet und dann über alle Partitionen gemittelt. Prinzipiell ist es möglich, einen geringeren Anteil der Daten in den Testdaten zu haben. Die Unsicherheit in der Evaluation wird schließlich dadurch ausgeglichen, dass die Evaluation mehrfach durchgeführt und anschließend gemittelt wird.

MCCV hat einen ähnlichen Bias wie die einfache Datenaufteilung, jedoch eine geringere Varianz. Dieser Vorteil geht jedoch damit einher, dass mit steigendem N der Rechenaufwand linear steigt. Jedoch ist auch hier eine Parallelisierung möglich.
Für $n \to \infty$, ist die Wahrscheinlichkeit, dass eine Observation unendlich oft in dem Trainingsdatensatz und unendlich oft in dem Testdatensatz ist, gleich ∞, bei einer endlichen Stichprobe und $\tilde{p} \in]0, 1[$. Diese Eigenschaft folgt aus dem Borel-Cantelli Lemma und der Tatsache, dass die Aufteilungen voneinander unabhängig sind. Deswegen konvergiert der gemittelte Vorhersagefehler ab einem bestimmten N. Das bedeutet, dass man nach hinreichend vielen Iterationen abbrechen kann. Ein heuristisches Argument hierfür ist, dass keine Informationen mehr in den Daten stecken, denn jede Konstellation von Trennung schon vorgekommen ist.

3.5 Wiederholte Kreuzvalidierung

Das Problem bei der k-fachen Kreuzvalidierung ist, dass der Vorhersagefehler von der Partitionierung abhängt. Das Problem kann man dadurch beheben, dass man die Kreuzvalidierung mehrmals mit unterschiedlichen Partitionierungen durchführt. Die Ergebnisse werden dann über alle Partitionierungen gemittelt. Sei $\overline{k}$ die Anzahl der Partitionierungen, dann muss das Modell $k \cdot \overline{k}$ mal angepasst und evaluiert werden. Diese Methode kann man also als Kombination aus der Kreuzvalidierung und MCCV auffassen. Das macht diese Methode sehr rechenaufwendig.

3.6 Bootstrap

Die bisherigen Resampling-Methoden basieren auf einer Partitionierung der Daten, die durch Ziehen ohne Zurücklegen erreicht wird. Bei Bootstrap Verfahren wird jedoch mit Zurücklegen aus den Observationen N Mal gezogen. Dadurch, dass mit Zurücklegen gezogen wird, gelangen nicht alle Observationen in die Bootstrap-Stichprobe. Diese Observationen dienen dann als Testdatensatz, mit dem der Vorhersagefehler berechnet werden kann. Der Vorgang kann wieder mehrfach wiederholt werden. Seien $B_1, ...B_N$ die N Bootstrap Stichproben und $\widehat{\theta^{BS}}_1, ...\widehat{\theta^{BS}}_N$ die dazugehörigen Schätzer für den Vorhersagefehler.
Im Mittel entspricht die Anzahl an Observationen, die nicht in der Bootstrap-Stichprobe sind, $0.368 \cdot n$. Dementsprechend ist die Anzahl an nicht doppelten Observationen in der Bootstrap-Stichprobe im Mittel $0.632 \cdot n$. Theoretisch ist es möglich, dass bei der Ziehung der gleiche Datensatz gezogen wird und keine Observation in den Testdatensatz gelangt. Jedoch ist das bei einer hinreichend großen Stichprobe sehr unwahrscheinlich.
Falls der Testdatensatz nicht leer ist, so gibt es mindestens eine doppelte Observation in dem Trainingsdatensatz. Das führt zu einem Bias, der korrigiert werden muss. Der .632 Bootstrap und der .632+ Bootstrap sind zwei Möglichkeiten diesen Bias zu korrigieren.

$$\widehat{\theta}_i = \omega\widehat{\theta^{BS}}_i + (1-\omega)\widehat{\theta^{RS}}_i,$$
$$\text{für} \quad i \in \{1, ..., N\}$$

Hier entspricht $\widehat{\theta^{RS}}_n$ dem Resubstitutionsfehler. Da der eine Schätzer den Fehler überschätzt und der andere unterschätzt, gleicht die Konvexkombination beider bei der geeigneten Wahl der Gewichte ω den Bias aus. Falls $\omega = 0.632$, so entspricht $\widehat{\theta}_n$ dem .632 Bootstrap Schätzer. Jedoch kann ω auch von der tatsächlichen Anzahl der Observationen im Testdatensatz abhängen. Dann entspricht der Schätzer dem Bootstrap .632+ Bootstrap Schätzer [6].

Kapitel 4

Modellbasiertes Resampling

4.1 Motivation durch etablierte Machine Learning Methoden

4.1.1 Bootstrapping für Entscheidungsbäume

Bootstrapping kann nicht nur zur Schätzung des Vorhersagefehlers verwendet werden, sondern auch zur Verbesserung der Vorhersage beziehungsweise der Modellgüte selber. Diese Methode wurde beispielsweise für Entscheidungsbäume verwendet, um diese zu aggregieren. Ein sehr bekanntes Modellensemble ist der Random Forest [2]. Die Idee ist, dass Entscheidungsbäume einen geringen Bias aber dafür deine hohe Varianz haben und durch die Aggregation mehrerer unabhängiger Bäume ein Schätzer mit geringerer Varianz entsteht. Die Bäume werden unabhängig gemacht, indem mehrere Bootstrap-Stichproben betrachtet werden und immer nur eine Teilmenge aller Variablen bei jedem Split des Entscheidungsbaumes betrachtet wird. Hierdurch kann implizit eine Variablenselektion erfolgen, da Variablen, die häufiger aufgenommen wurden, mehr in die Vorhersage einfließen. [15]. Die Hyperparameter m_{var} und B können beispielsweise mit einer Gittersuche mittels Kreuzvalidierung bestimmt werden.

Random Forest

1. Berechne $B \in \mathbb{N}$ Bootstrap-Stichproben wie in 3.6.
2. Für jede Bootstrap-Stichprobe D_b mit $b \in \{1, ..., B\}$ passe einen Entscheidungsbaum an mit den folgenden Anpassungen:
   - Anstatt bei jedem Split aus allen Variablen die Beste auszuwählen, werden m_{var} zufällige Variablen ausgewählt, von denen die Beste für den Split verwendet wird. Die beste Variable ist hier als diejenige, die Verlustfunktion am meisten minimiert, zu verstehen, z.B. Gini Verlustfunktion oder Missklassifikationrate.
3. Als Klassifikator für eine Observation wird dann das Mehrheitsvotum der einzelnen Bäume verwendet. Bei Regressionsbäumen wird der Mittelwert der Bäume verwendet.

4.1.2 Boosting zur Fehlerminimierung

Ein weiteres Konzept, welches häufig verwendet wird, um die Vorhersage zu verbessern ist Boosting. Insbesondere das komponentenweise modellbasierte Boosting [5], welches in dem Paket "mboostïn R [10] implementiert ist, soll als theoretisches Grundkonzept für modellbasiertes Resampling dienen. Die generelle Idee ist, dass man ein Grundmodell, häufig auch base learner genannt, verwendet. Dieses Modell wird dann schrittweise verbessert, indem Observationen, die schlecht vorhergesagt wurden, stärker

gewichtet werden. Im Falle einer Regression werden schrittweise die Residuen verbessert. Für jeden Schritt kann immer nur die Variable verwendet werden, die Verlustfunktion am stärksten minimiert. Man kann zeigen, dass die Boostingmethoden äquivalent zu funktionalen Gradientenverfahren sind, bei dem die Verlustfunktion durch den Funktionalgradienten approximiert wird [12]. Im Folgenden werden zwei Boosting Algorithmen beschrieben, um die Ähnlichkeit zum modellbasierten Resampling zu sehen.

Adaboost

Sei hierfür n die Anzahl an Observationen, p die Anzahl an Variablen und $m \in \{1, ..., M\}$ der m- te Schritt und M die Anzahl der Schritte.

1. Wähle ein Grundmodell mit Struktur $g : \mathbb{R}^p \mapsto \mathbb{R}$.
2. Setzte ursprüngliche Gewichte für die Observationen $(w_i^{(0)})_{i\in\{1,...,n\}} = \frac{1}{n}$. Setzte $m = 0$.
3. Passe g an die mit $w_i^{(m-1)}$ gewichteten Observationen an. Dadurch erhalten wir $\widehat{g}^{(m)}$.
4. Berechne die gewichtete Fehlerrate von $\widehat{g}^{(m)}$:
   - $\epsilon^{(m)} = \frac{\sum_{i=1}^n w_i^{(m-1)} \mathcal{I}\left(Y_i \neq \widehat{g}^{(m)}(X_i)\right)}{\sum_{i=1}^n w_i^{(m-1)}}$,
     wobei $\mathcal{I}$ die Indikatorfuntion darstellen soll mit
     $$\mathcal{I}(x) = \begin{cases} 1, & x \quad \text{wahr} \\ 0, & x \quad \text{falsch} \end{cases}.$$
   - $\alpha^{(m)} = \log \frac{1-\epsilon^{(m)}}{\epsilon^{(m)}}$.
   - Aktualisiere die Gewichte:
     $$\tilde{w}_i = w_i^{(m-1)} \cdot \exp \alpha^{(m)} \mathcal{I}\big(Y_i \neq \widehat{g}^{(m)}(X_i)\big)$$
     $$w_i^{(m)} = \frac{\tilde{w}_i}{\sum_{i=1}^n w_i^{(m)}}.$$
   - Iteriere die Schritte 3 und 4 bis $m = M$. Erhalte den aggregierten Schätzer $\widehat{f}_{\text{AdaBoost}}(x)$ als
     $$\widehat{f}_{\text{AdaBoost}}(x) = \arg\max_{y\in\{0,1\}} \sum_{m=1}^{M} \alpha^{(m)} \mathcal{I}\big(y = \widehat{g}^{(m)}(x)\big).$$

Komponentenweises L_2 Boosting

1. Wähle ein Grundmodell mit Struktur $g : \mathbb{R}^p \mapsto \mathbb{R}$.
2. Setze $f^{(0)} = \overline{Y}$ als konstante Funktion und $m = 0$.
3. Erhöhe m um 1 und berechne die Residuen $U_1^{(m)}, ...U_n^{(m)}$ als $U_i^{(m)} = Y_i - \widehat{f}^{(m-1)}(X_i)$.
4. Für alle $j \in \{1, ..., p\}$ passe $\widehat{g}_j^{(m-1)}$ mit Zielgröße $(U_1^{(m)}, ..., U_n^{(m)})$ an die Daten an und wähle das j^* aus, für das die Summe der quadratischen Residuen von $\widehat{g}_j^{(m-1)}$ minimal ist. Also
   $$j^* = \text{argmin}_j \sum_{i=1}^{n} \left[U_i^{(m)} - \widehat{g}_j^{(m-1)}(X_i)\right]^2.$$

5. Aktualisiere
$$\widehat{f}^{(m)} = \widehat{f}^{(m-1)} + \eta \cdot \widehat{g}_{j^*}^{(m-1)},$$
wobei η als Lernrate interpretiert werden kann mit $\eta \in]0, 1[$.

6. Wiederhole Schritte 3,4 und 5 bis $m = M$.

Je größer η gewählt wird, desto größere Sprünge macht das Modell. Prinzipiell sollte η klein gehalten werden, solange die Rechenzeit überschaubar ist. Die Lernrate klein zu wählen hat auf jeden Fall keinen Nachteil im Vergleich zu einer großen Lernrate, da eine Variable auch mehrfach hintereinander ausgewählt werden kann. Das bedeutet, dass wenn der große Schritt optimal wäre, dann würde bei den kleinen Schritten immer die gleiche Variable ausgewählt, bis die Summe der kleinen Schritte gleich dem großen Schritt entspricht.

Anstatt der L_2-Norm kann natürlich auch eine andere Verlustfunktion verwendet werden. Im Falle einer linearen Regression können die Parameter für eine gegebene Variable aufgrund der linearen Strukturannahme aufaddiert werden.

Allgemeines funktionales Gradientenboosting

Generell soll das folgende Problem minimiert werden:

$$f^*(\cdot) = \text{argmin}_{f(\cdot)} = \mathbb{E}[\rho(Y, f(X)]$$

Hierbei ist $\rho(\cdot, \cdot)$ die Verlustfunktion, von der angenommen wird, dass sie konvex und differenzierbar ist. Theoretisch ist es jedoch möglich, dass f nicht konvex ist: In diesem Fall ist allerdings nicht garantiert, dass die richtige Lösung, oder überhaupt eine Lösung gefunden wird. Diese hängt dann auch von der Initialisierung der Parameter ab. Dies ist insbesondere bei der Anpassung neuronaler Netzwerke der Fall. Wird die Verlustfunktion als quadratischen Fehler gewählt, dann erhalten wir den Spezialfall des L^2 Boosting wie in 4.1.2. Zur Schätzung der Verlustfunktion kann die empirische Verlustfunktion

$$\rho = \frac{1}{n} \sum_{i=1}^{n} \rho(Y_i, f(X_i))$$

verwendet werden.

1. Wähle ein Grundmodell mit Struktur $g : \mathbb{R}^p \mapsto \mathbb{R}$.

2. Setze $\widehat{f}^{(0)} \equiv 0$ und $m = 0$, oder
$$\rho \equiv \text{argmin}_{c=0} \quad \frac{1}{n} \sum_{i=1}^{n} \rho(Y_i, c)$$

3. Erhöhe m um 1 und berechne die negativen Gradienten $-\frac{\partial}{\partial f}\rho(Y, f)$ und setze diesen in $\widehat{f}^{(m-1)}$ ein. Dabei erhält man die Pseudoresiduen $U_1, ..., U_n$ mit
$$U_i^{(m)} = -\frac{\partial}{\partial f}\rho(Y, f)|_{f=\widehat{f}^{(m-1)}},$$
für alle $i = 1, ..., n$.

4. Passe den Basislerner$g$ mit Zielgröße $(U_1^{(m)}, ..., U_n^{(m)})$ an die Daten an. Dabei erhält man das Modell $\widehat{g}^{(m-1)}$, welche als als Approximation des negativen Gradienten aufgefasst werden kann.

5. Aktualisiere
$$\widehat{f}^{(m)} = \widehat{f}^{(m-1)} + \eta \cdot \widehat{g}^{(m-1)},$$
wobei η als Lernrate interpretiert werden kann mit $\eta \in]0, 1[$.

6. Wiederhole Schritte 3,4 und 5 bis $m = M$.

Man beachte, dass bei beliebigen Verlustfunktionen ein deutlich komplexeres Modell angepasst werden kann, als das herangezogene Grundmodell. Jedoch kann jedes einzelne Teilmodell durch die additive Struktur interpretiert werden. Vor allem bei der negativen log-likelihood als Verlustfunktion erhält man dieselbe Interpretation wie für generalisierte Regressionsmodelle, falls ein solches als Grundmodell angenommen wird. in den folgenden Abschnitten werden Algorithmen für unterschiedliche Arten des modellbasierten Resampling angegeben.

4.2 Resampling im komponentenweisen modellbasierten Boosting Modell: Rmboost

4.2.1 Motivation

Breimann führte das adaptive Bagging ein, welches als boosting-bagging [3] Hybridmodell angesehen werden kann. Es bietet die Möglichkeit, sowohl die Varianz, als auch den Bias des Modells zu verringern. Der Bias wird dadurch verringert, dass statt der Residuen des Modells, die Out-of-bag-Residuen benutzt werden. Anstatt die out-of-bag-Residuen zur Reduktion des Bias zu verwenden, wollen wir diese nutzen, um eine realistische Schätzung des Modellfehlers, korrigiert auf diesen Bias, zu berechnen.

Das Problem bei dem normalen modellbasierten Boosting Modell ist, dass wenn das Modell nicht frühzeitig gestoppt wird, die Residuen so lange verbessert werden, bis das Modell zu nah an die Daten angepasst ist. Dieses Problem tritt vor allem bei einer großen Anzahl an Variablen im Verhältnis zu der Anzahl an Observationen. Im schlimmsten Fall erhält man ein Modell, das auf dem Testdatensatz keinen Fehler macht. Insbesondere bei nicht parametrischer Regression muss man hier vorsichtig sein. Optimal wäre es also, wenn das Modell die optimale Stoppzeit selber berechnen würde, um den Bias Varianz Tradeoff selber in den Griff zu bekommen. Dann wäre keine Hyperparameteranpassung notwendig und die Arbeit bei der Anpassung eines optimalen Modells wäre deutlich geringer. Insbesondere muss man sich keine Gedanken um die Art des Resamplings bei der Hyperparameteranpasung machen. Dadurch, dass das Modell Variablenselektion durchführen soll, kann man auf den gleichen Daten keine Fehlerevaluation durchführen, da das einer Postdiktion gleichkommen würde.

4.2.2 Der Algorithmus

1. Wähle ein Grundmodell mit Struktur $g : \mathbb{R}^p \mapsto \mathbb{R}$.

2. Setze $f^{(0)} = \overline{Y}$ als konstante Funktion und $m = 0$.

3. Setzte $\epsilon_0 = \frac{1}{n} \sum_{i=1}^{n} |\overline{Y} - Y_i|$ als mittlere absolute Residualsumme des Mittelwerts, und $(U_i^{(0)})_{i \leq n}$ die Residuen des Mittelwerts.

4. Erhöhe m um 1 und berechne die Residuen $U_1^{(m-1)}, ..., U_n^{(m-1)}$ als $U_i^{(m-1)} = Y_i - \widehat{f}^{(m-1)}(X_i)$,

sowie die mittlere absolute Residualsumme

$$\overline{U}^{(m-1)} = \frac{1}{n}\sum_{i=1}^{n} |U_i^{(m-1)}|.$$

5. Ziehe B Bootsrap Stichproben, wie in 3.6 aus den Daten in Kombination mit den Residuen $U^{(m)}$.

6. Für alle Bootstrap-Stichproben D_b, wähle n_{var} Variablen zufällig aus und für alle $j \in \{1, ..., n_{\text{var}}\}$ passe $(\widehat{g}_{jb}^{(m)})$ mit Zielgröße $(U_{1b}^{(m)}, ..., U_{nb}^{(m)})$ an die Daten an. Wähle das j^* aus, für das die Summe der quadratischen Residuen von $\widehat{g}_{jb}^{(m)}$ minimal ist. Also

$$j_b^* = \text{argmin}_j \sum_{i=1}^{n} \left[U_{ib}^{(m)} - (\widehat{g}_{jb}^{(m)})(X_i)\right]^2.$$

7. Berechne die Out-of-bag-Residuen $\tilde{U}_i^b$ für alle Bootstrap Stichproben.

8. Berechne

$$\widehat{g}_{j^*}^{(m)} = \frac{1}{B}\sum_{b=1}^{B} \widehat{g}_{jb}^{(m)},$$

sowie die gemittelten mittleren absolute Out-of-bag-Residualsummen

$$\tilde{U}^{(m)} = \frac{1}{B}\sum_{b=1}^{B} \frac{1}{n_b}\sum_{i=1}^{n_b} |\tilde{U}_{ib}^{(m)}|.$$

9. Update

$$\widehat{f}^{(m)} = \widehat{f}^{(m-1)} + \eta \cdot \widehat{g}_{j^*}^{(m)},$$

wobei η als Lernrate interpretiert werden kann mit $\eta \in]0, 1[$.

10. Aktualisiere die out-of-sample absoluten Residualsummen des gesamten Modells:

$$\epsilon_m = \epsilon_{m-1} + \tilde{U}^{(m)} - \overline{U}^{(m-1)}.$$

11. Wiederhole Schritte 3,4 und 5 bis $m = M$ oder bis die Out-of-bag-Residualsumme kleiner ist als die Residualsumme des vorherigen Modells.

$$U^{(m-1)} \leq \tilde{U}^{(m)}.$$

Das finale Modell ist dann $\widehat{f}^{(M)}$, beziehungsweise $\widehat{f}^{(m^*)}$. Und die out-of-sample mittleren absoluten Residualsummen erhält man durch ϵ_M beziehungsweise ϵ_{m^*}

4.1 Lemma. Für $B \to \infty$ konvergiert Rmboost fast sicher gegen das zu Grunde liegende Modell, falls dieses additiv ist, alls zusätzlich $M \to \infty$.

Proof. Wir müssen zeigen, dass in jedem Schritt der Grenzwert $\lim_{B\to\infty} \widehat{g}_{j^*}^{(m)} = \frac{1}{B}\sum_{b=1}^{B} \widehat{g}_{jb}^{(m)}$ existiert und gegen das Grundmodell $\widehat{g}$ konvergiert. Da wir mit Zurücklegen ziehen, ist die Wahrscheinlichkeit, dass eine Variable zufällig ausgewählt wird: $\frac{n_{\text{var}}}{p} \in \mathbb{R}$. Somit ist

$$\sum_{b=1}^{\infty} \frac{n_{\text{var}}}{p} = \infty.$$

Nach dem Borel-Cantelli Lemma folgt, dass jede Variable fast sicher unendlich oft zur Auswahl gestellt wird. Da die Bootstrap-Stichproben unabhängig von einander sind, und die n, p endlich sind, existiert der Erwartungswert von $\widehat{g}_{jb}^{(m)}$ und das starke Gesetzt der großen Zahlen greift. Somit liegt fast siche komponentenweise Konvergenz vor. □

Das folgende Lemma zeigt, dass für die absolute Residualsumme eine Obergrenze geschätzt werden kann. Wir wissen, dass für jedes Teilmodell der Fehler erwartungstreu geschätzt wird, da wir unabhängige Daten zur Schätzung des Fehlers verwenden.

4.2 Lemma (Absolute Residualsummen). ϵ_M ist ein erwartungstreuer Schätzer für die mittleren absoluten Residualsummen, mit

$$\epsilon_0 = \sum_{i=1}^{n} |\overline{Y} - Y_i|,$$
$$\epsilon_m = \epsilon_{m-1} + \tilde{U}^{(m)} - U^{(m-1)},$$

mit

$$\tilde{U}^{(m)} = \frac{1}{B} \sum_{b=1}^{B} \frac{1}{n_b} \sum_{i=1}^{n_b} |\tilde{U}_{ib}^{(m)}|,$$

ist eine Obergrenze für den Residualfehler des Modells für alle $m \in \mathbb{N}$.

Proof. Induktion über m:

- $m = 0$ ist klar, da der Mittelwert erwartungstreu ist
- $m - 1 \to m$: Gelte die Aussage für $m - 1$.
  Wir kennen die Residualsummen $U^{(m-1)}$, sowie deren out-of-Bag-Schätzung $\tilde{U}(m-1)$ von $\widehat{g}_{j^*}^{(m-1)}$.
  Das neue Modell wird an die Residuen angepasst. Die Differenz entspricht dem Bias der Residualsumme.
  $$\text{Bias}(U^{(m-1)}) = \tilde{U}^{(m-1)} - U^{(m-1)}.$$
  Die verzerrten Residuen $U^{(m-1)}$ haben nun eine erwartreue Schätzung:
  $$\mathbb{E}[U^{(m-1)}] = \mathbb{E}[\widehat{g}_{j^*}^{(m)}] + \mathbb{E}[\tilde{U}^{(m)}].$$
  Somit Verringert sich die absulute Residualsumme um die Differenz der mittlernen absoluten Residualsumme zum Zeitpunkt $m - 1$ und dem Zeitpunkt m: $\tilde{U}^{(m-1)} - U^{(m-1)}$. Addieren wir noch den Bias hinzu, so erhalten wir:
  $$\epsilon_m = \epsilon_{m-1} - (\tilde{U}^{(m-1)} - \tilde{U}^{(m)}) + \tilde{U}^{(m-1)} - U^{(m-1)} \Leftrightarrow$$
  $$\epsilon_m = \epsilon_{m-1} + \tilde{U}^{(m)} - U^{(m-1)}.$$
  Nach den Induktionsprinzip folgt die Aussage für alle $m \in \mathbb{N}$

. □

4.3 Theorem (Optimalität von Rmboost). Das Modell, wie oben beschrieben ist optimal, in dem Sinne, dass das weitere Hinzufügen von Variablen die tatsächliche Residualsumme nicht mehr verbessert.

Proof. Aus Lemma 4.2 folgt, dass sich die absuluten Residualsummen um $\tilde{U}^{(m-1)} - U^{(m-1)}$ verbessern. Da der Algorithmus abgebrochen wird, sobald

$$\tilde{U}^{(m)} \geq U^{(m-1)},$$

kann eine weitere Anpassung der Residuen das Modell nicht mehr verbessern. □

Wie man sieht, enthält der Algorithmus sowohl Elemente des Random Forest, als auch Elemente des komponentenweisen modellbasierten Boosting. Würde man das Resampling weglassen, so erhielte man das normale Boosting Modell. Jedoch besteht durch das Resampling der große Unterschied, im Verglich zu dem komponentenweisen Boosting Modell darin, dass in jedem Schritt nicht nur eine Variable, sondern mehrere Variablen einfließen können. Die Ähnlichkeit zum Random Forest besteht darin, dass möglichst unabhängige Modelle angepasst werden, über die gemittelt wird. Das führt zu einer Art Gruppenintelligenz. Bei beiden Modellen werden die Einzelmodelle dadurch dekorreliert, dass nicht alle Variablen gleichzeitig betrachtet werden, sondern immer nur eine zufällige Teilmenge.

Man beachte, dass falls man die Anzahl der Bootstrap-Stichproben auf eine beschränkt, also $B = 1$ und $k = p$ setzt, der Algorithmus dem stochastischen, funktionalen Gradientenverfahren gleicht, wie in [7] beschrieben.

4.2.3 Vor-und Nachteile von Rmboost

Vorteile

- **Interpretierbarkeit und Inferenz**
  Wie wir gesehen haben, konvergiert Rmboost gegen das zu Grunde liegende Regressionsmodell. Somit erhalten wir ein interpretierbares Modell, da sich die Struktur des Modells nicht verändert hat. Lediglich die Schätzweise ist eine andere. Für Inferenz sollten jedoch unabhängige Daten genutzt werden, da durch die Variablenselektion ein Bias entsteht, wenn die gleichen Daten verwendet werden.

- **Variablenselektion**
  Wir können analog zum komponentenweisen Boosting Modell eine Variablenselektion durchführen. Variablen, die durch das Modell nicht ausgesucht werden, fallen raus. Da aber immer nur eine gewisse Anzahl an Variablen zufällig ausgewählt wird, ist die Wahrscheinlichkeit groß, dass jede Variable zumindest einmal von dem Modell ausgewählt wird. Jedoch werden diese Variablen nicht häufig ausgewählt und daran kann man sich eine Schwelle überlegen, anhand derer Variablen aufgenommen werden sollen oder nicht.

- **Fehlerbestimmung**
  Der Fehler kann wegen des Resampling out-of-bag geschätzt werden.

- **Hyperparameter**
  Der wichtigste Hyperparameter beim Boosting, nämlich die Anzahl an Schritten, muss nicht extra bestimmt werden, da ein alternatives Kriterium vorliegt. Das Modell verbessert seine Residuen nur, wenn sich auch die Out-of-sample-Residuen verbessern. Somit ist das Modell leichter anpassbar, da man sich nicht um Hyperparameter kümmern muss.

- **Overfitting**
  Dadurch, dass der Algorithmus vor Konvergenz der Residuen abgebrochen wird, wird eine zu

starke Anpassung an die Daten verhindert und der Schätzer hat eine geringere Varianz als das Grundmodell. Somit ist das Risiko für Overfitting geringer, als bei einer Anpassung des Modells über Fischer Scoring, oder ähnlichen Methoden.

- **Multikollinearität**
  Selbst bei hoch korrelierten Variablen kann das Modell stabil geschätzt werden. Das liegt daran, dass die Parameter einzeln und bedingt unabhängig voneinander geschätzt werden.

- **Geschwindigkeit**
  Sowohl bei klassischen statistischen Methoden, als auch neuronalen Netzen konvergieren stochastische Gradientenverfahren in der Regel schneller als reine Gradientenverfahren [21]. Ähnlich könnte das auch bei Rmboost der Fall sein, denn Boosting kann als funktionales Gradientenverfahren aufgefasst werden.

Nachteile

- **Rechenzeit**
  Bei gleicher Lernrate ist die Anpassung von Rmboost um $\frac{B \cdot p}{n_{\text{var}}}$ höher als beim Boosting. p ist hier die Anzahl der Variablen, B die Anzahl an Bootstrap-Stichproben und n_{var} die Anzahl der zufällig ausgewählten Variablen bei jeder Booostrap Stichprobe. Man sollte B auf jeden Fall deutlich größer als $\frac{p}{n_{\text{var}}}$ wählen, um sicher zu stellen, dass nicht durch Zufall eine Variable in einem Schritt gar nicht in die Stichprobe kommt. Jedoch muss man beachten, dass man bei dem Boosting Algorithmus noch die Hyperparameter anpassen sollte, wofür man wieder Resampling-Methoden benutzten muss, was die Rechenzeit wieder erhöht.

- **Weniger Felxibilität**
  Dadurch, dass das Hyperparametertuning in der Schätzung schon enthalten und auch die Variablenselektion ebthalten, kann man keine Parameter, abgesehen von der Lernrate und den in Frage kommenden Variablen, selber anpassen.

- **Inferenz** Wie beim Boosting, ist die Inferenz des Modells verzerrt und deshalb muss diese auf unabhängigen Daten betrieben werden.

- **Effektstärke**
  Dadurch, dass der Algorithmus vorzeitig gestoppt wird, wird die Effektstärke ähnlich wie beim Boosting tendenziell unterschätzt. Dieser Effekt ist jedoch nicht so ausgeprägt, wie beim Boosting, da das Abbruchkriterium durch den Out-of-sample-Fehler bestimmt wird und eine weiter Verbesserung der Residuen das Modell nicht verbessern würde.

- **Modellverletzungen**
  Bei Multikollinearität besteht die Möglichkeit, dass eine Variable nicht aufgenommen wird, obwohl sie einen Einfluss hat, und stattdessen eine andere korrelierte Variable aufgenommen wird. Dieses Problem wird jedoch durch das Betrachten einer zufälligen Teilmenge von der Menge aller Variablen minimiert, ähnlich wie bei dem Random Forest.

Betrachte ein Grundmodell (base learner) f. Sei p die Anzahl an Variablen und n die Anzahl an Observationen.

Kapitel 5

Implementierung von Rmboost in R

Der Algorithmus ist wie in 4.2.2 beschrieben in R implementiert [14]. Die Funktion rmboost() gibt ein Objekt vom Typ "rmboostßurück und besitzt die Methoden "rmboost.plot", "rmboost.print", und "rmboost.summary". Der Befehl enthält mehrere Parameter.

5.1 Parameter von rmboost()

- **formula:**
  Symbolische Darstellung des Modells. Diese kann wie die formula eines "*mboost*Öbjekts angegeben werden . Bisher kann mit "*bols()*ëin linearer Effekt spezifiziert werden.
- **data:**
  Data Frame, das die Variablen für das Modell enthält.
- **mvar = 5:**
  Die Anzahl an Variablen, die für jede Bootstrap-Stichprobe zufällig ausgewählt werden sollen.
- **B = 100:**
  Anzahl der Bootstrap-Stichproben, die bei jeder Iteration gezogen werden sollen.
- **alpha = 1:**
  Der Faktor, um den der Out-of-Bag-Fehler einer Bootstrap-Stichprobe eines Modell größer sein muss, als der des Nullmodells.
- **nu = 0.1:**
  Der Faktor mit, dem in jedem Schritt das neue Modell zu dem bisherigen Modell, aktualisiert wird. Dieser entspricht der Lernrate. Der Wert sollte zwischen in $]0, 1[$ liegen.
- **bstop =.6:**
  Nur falls *stop_intern == TRUE*. Dieser Term entspricht dem Anteil der Modelle, bei denen der Out-of-bag-Fehler größer ist, als der des Nullmodells.
- **nstop = 100:**
  Anzahl der Boosting Schritte.
- **cov_weights = F:** Gibt an, ob Variablen nach ihrer Kovarianz gewichtet werden sollen.
- **iter = F:**
  Gibt an, ob Iterationsschritte in der Konsole ausgegeben werden sollen.

- **stop_intern = F:**
  Gibt an, ob das Modell in einzelnen Teilschritten bei zu schlechtem Out-of-bag-Fehler nicht aktualisiert werden soll.

Die in 2 simulierten Datensätze werden nun genutzt, um Eigenschaften von rmboost zu untersuchen und einen Vergleich mit anderen Modellen vorzunehmen.

5.2 Methoden von Rmboost

Im Vergleich zu der in mboost implementierten Funktion glmboost, gibt "plot(object)ëin Object vom Typ ggplot zurück und nicht ein Objekt vom in base R enthaltenen Plot [19]. In mboost ist der Befehl die Überschrift des Koeffizienten Pfad Plots. In rmboost wurde auf eine Überschrift verzichtet, da der Modellbefehl in der Regel zu lang ist, siehe Abbildung 6.1.
Mit "predict(object, newdata)"kann basierend auf den Daten "newdata"von Typ data frame durch das Model öbject"vom Typ rmboost die erwartete Zeilgröße geschätzt werden.

Kapitel 6

Eigenschaften von Rmboost und Vergleich mit anderen Methoden

6.1 Großes n, kleines p, kleiner Fehler

6.1.1 Konvergenz

Bei den Koeffizientenpfaden fällt im Allgemeinen auf, dass die Parameter des Modells bei rmboost schneller konvergieren, als die von mboost. Das ist bei allen hier getesteten Parametereinstellungen der Fall. Die durchgezogene (gestrichelte) Linie in den Grafiken Abbildung 6.1, 6.2 und 6.3 markiert den Iterationsschritt, bei dem alle von null verschiedenen Koeffizienten höchstens um 10 Prozent (5 Prozent) von dem tatsächlichen Parameter abweichen. Bei mboost wird der Intercept auf 0.5, statt auf 0 gesetzt. Eine mögliche Begründung, weshalb rmboost schneller konvergiert, ist, dass es möglich ist, dass in einem Schritt mehrere Variablen in das Modell aufgenommen werden können.

Bei einem Verglich von Abbildung 6.2 und Abbildung 6.3, erkennt man, dass für bei mvar $= 3$ predictor 3 bei einer niedrigeren Iterationszahl eher aufgenommen wird als bei mvar $= 2$. Das ist sowohl bei $B = 10$, als auch bei $B = 100$ der Fall.

Bei 100 Bootsrap-Stichproben ist die Stabilität der Koeffizientenschätzung höher als bei 10. Auch die Konvergenzgeschwindigkeit ist hier leicht höher. Ähnlich wie bei dem random forest ist es daher naheliegend, dass es sinnvoll sein kann die Anzahl der Bootstrap-Stichproben so hoch zu setzen, wie die Rechenzeit erlaubt. Diese steigt nämlich mit der Sequenzialität des Boosting ohne Parallelisierung mit $\mathcal{O}(B \cdot \text{nstop})$.

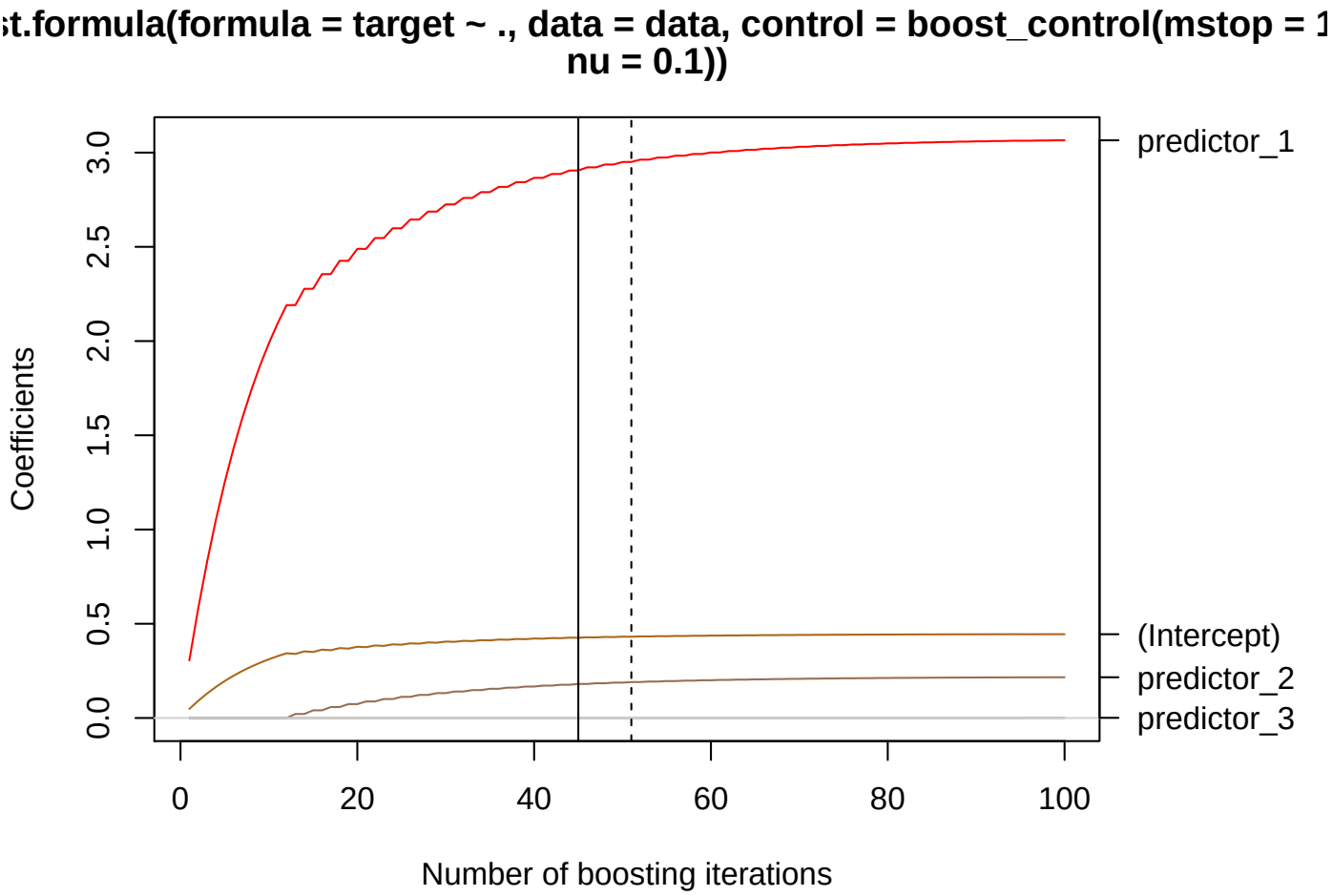

Abbildung 6.1: Koeffizienten Pfade von glmboost für ein lineares Modell, angepasst an simulierte Daten 1

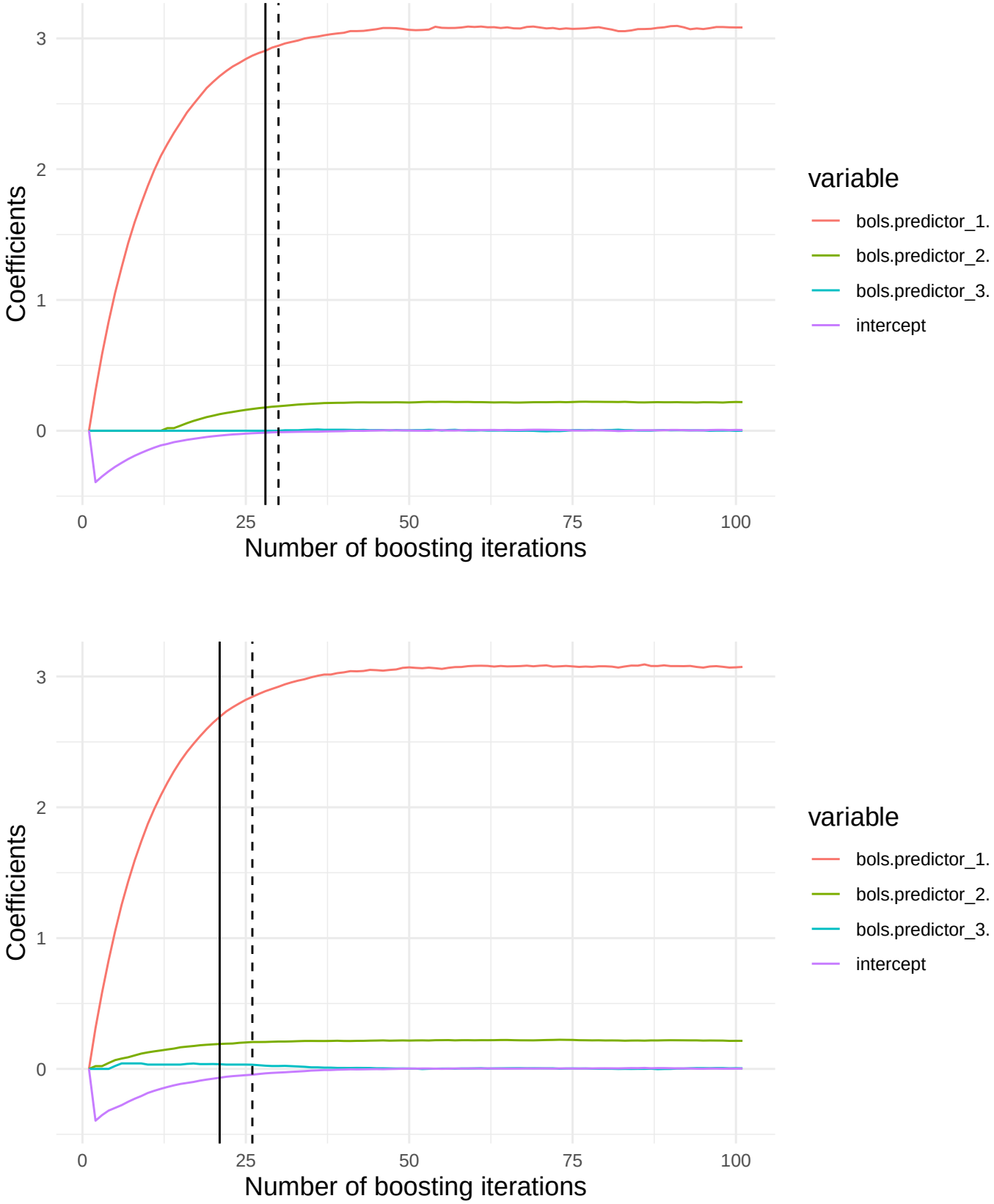

Abbildung 6.2: Koeffizienten Pfade von rmboost (mvar = 3 oben, mvar = 2 unten, B = 10) für ein lineares Modell, angepasst an simulierte Daten 1

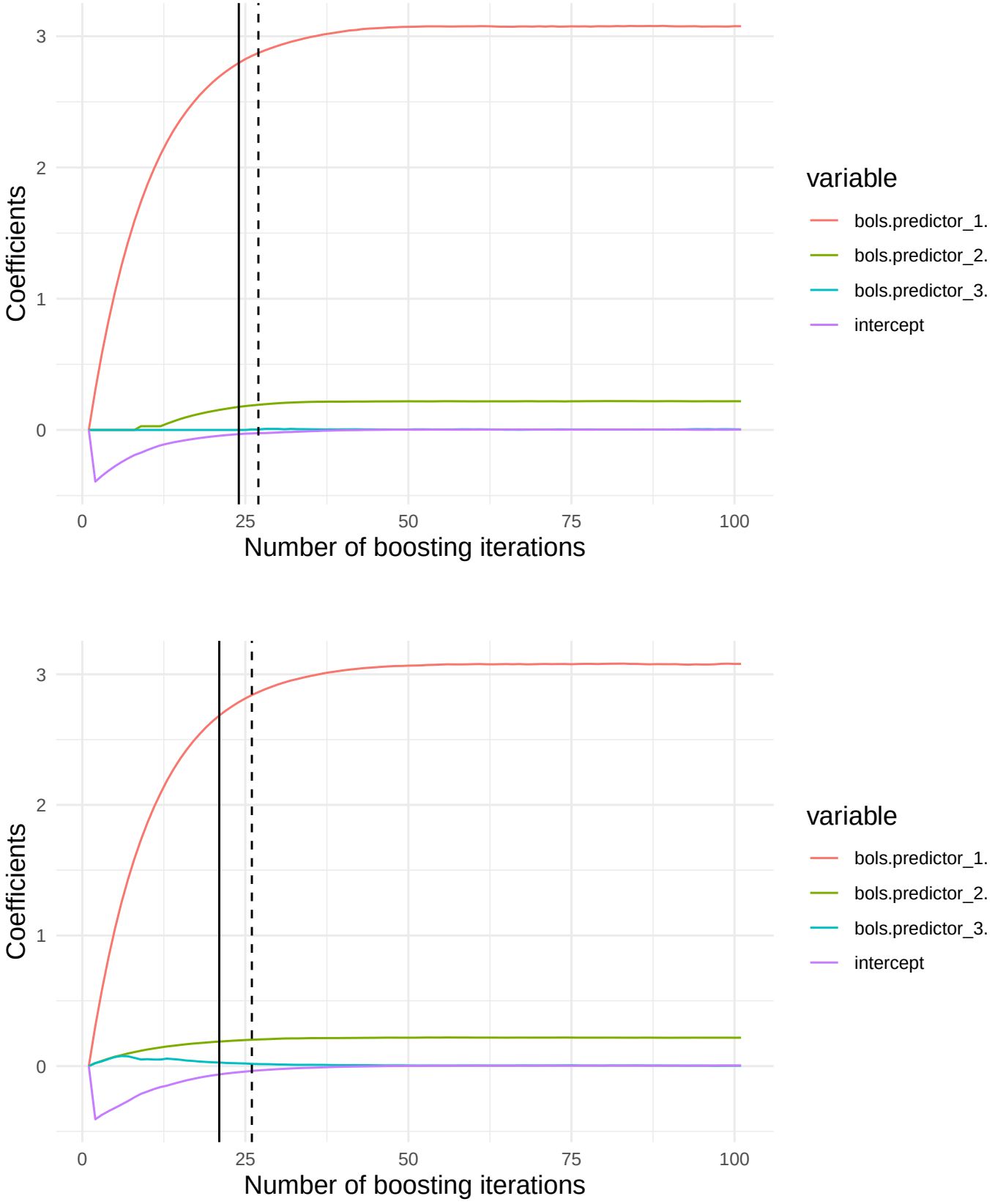

Abbildung 6.3: Koeffizienten Pfade von rmboost (mvar = 3 oben, mvar = 2 unten, B = 100) für ein lineares Modell, angepasst an simulierte Daten 1

Bei hundertfacher Durchführung der Simulation und Anpassung der Modelle , kommt man zu dem Ergebnis, dass rmboost nicht nur schneller bei der gegebenen Datenbasis konvergiert, sondern auch konsistenter konvergiert. Denn die Standardabweichung der Stoppzeit ist bei rmboost in beiden Modellen deutlich niedriger als bei mboost, wie Tabelle 6.1 zeigt.

Modell	rmboost(mvar = 3)	rmboost(mvar = 2)	mboost
Mittelwert	23.38	24.0300	33.130
Standardabweichung	1.890	1.732	3.311

Tabelle 6.1: Konvergenzgeschwindigkeit Daten 1, B = 100, mstop = 100, simulierte Daten, 100 Simulationen

6.1.2 Vergleich mit anderen Modellen

Abgesehen vom Boosting gibt es auch andere Regularisierungsmethoden. Prominent sind $\mathcal{L}^1$ und $\mathcal{L}^2$ Regularisierung. Bei $\mathcal{L}^1$ Regularisierung, auch Lasso Regression genannt, wird eine mit der $\mathcal{L}^1$ Norm modifizierte Likelihood optimiert [17].

$$\min_\beta \sum_{i=1}^{n} l_i(\beta, y_i) \quad \text{u.d.N} \quad \sum_{j=1}^{p} |\beta_p| \leq t.$$

Mit dem Lagrange-Multiplikator lässt sich das Problem folgendermaßen darstellen:

$$\min_\beta \sum_{i=1}^{n} l_i(\beta, y_i) + \lambda \sum_{j=1}^{p} |\beta_p|,$$

mit $\lambda, t \geq 0$.
Durch diese Regularisierung werden Parameter des Modells eher auf null gesetzt und es kommt zu einer sparsamen Lösung. Je größer λ, desto stärker der Effekt. Bei $\mathcal{L}^2$ Regularisierung, auch Ridge Regression genannt, wird eine mit der $\mathcal{L}^2$ Norm modifizierte Likelihood optimiert [11].

$$\min_\beta \sum_{i=1}^{n} l_i(\beta, y_i) \quad \text{u.d.N} \quad \sum_{j=1}^{p} (\beta_p)^2 \leq t.$$

Mit dem Lagrange-Multiplikator lässt sich auch hier das Problem folgendermaßen darstellen:

$$\min_\beta \sum_{i=1}^{n} l_i(\beta, y_i) + \lambda \sum_{j=1}^{p} (\beta_p)^2,$$

mit $\lambda, t \geq 0$.
Bei der Ridge Regression werden Parameter mit steigendem Regularisierungsparameter λ in Richtung null geschrumpft.

Man kann die Vorteile beider Modelle kombinieren und damit ein Modell sowohl mit Variablenselektion als auch mit geschrumpften Parametern anpassen. Diese Kombination wird als elastisches Netz bezeichnet [8].

$$\min_\beta \sum_{i=1}^{n} l_i(\beta, y_i) + \lambda_1 \sum_{j=1}^{p} |\beta_p| + \lambda_2 \sum_{j=1}^{p} (\beta_p)^2$$

Tabelle 6.2 zeigt die Mittelwerte der Koeffizienten für die Modelle mboost, Lasso, Ridge, elastisches Netz und rmboost. Tabelle 6.3 zeigt die Standardabweichung der Koeffizienten. Der Datensatz 1 aus dem zweiten Kapitel wurde hierfür 300 Mal simuliert.
In Tabelle 6.2 erkennt man, dass alle Regularisierungsmethoden im Mittel die Parameter leicht unterschätzen. Zur Bestimmung des Hyperparameters λ wurde Kreuzvalidierung verwendet. Für mboost und rmboost wurde die interne Stoppfunktion verwendet.

Tendenziell fällt der Bias gegen null bei rmboost durchschnittlich leicht geringer aus als bei den anderen Methoden. Die Standardabweichung ist über alle Modelle hinweg vergleichbar.

Koeffizienten	Ridge	Lasso	Elastisches Netz	mboost	rmboost.2	rmboost.3	echt
Prediktor.1	2.990	2.980	2.990	2.99	2.996	2.998	3
Prediktor.2	0.199	0.196	0.198	0.197	0.200	0.199	0.2
Prediktor.3	0.000	0.000	0.001	0.000	0.000	0.000	0
Intercept	1.000	1.000	1.001	.	1.000	1.000	1

Tabelle 6.2: Vergleich von Koeffizienten verschiedener Modelle, angepasst an simulierte Daten 1

Koeffizienten	Ridge	Lasso	Elastisches Netz	mboost	rmboost.2	rmboost.3
Prediktor.1	0.028	0.029	0.029	0.029	0.029	0.028
Prediktor.2	0.007	0.007	0.007	0.007	0.007	0.007
Prediktor.3	0.010	0.010	0.010	0.008	0.010	0.010
Intercept	0.028	0.028	0.028	.	0.028	0.028

Tabelle 6.3: Vergleich der Standardabweichung der Koeffizienten verschiedener Modelle, angepasst an simulierte Daten 1

6.2 Großes n, kleines p, großer Fehler

Bei dem zweiten simulierten Datensatz weist das Modell einen deutlich höheren Fehler auf, als bei dem ersten simulierten Datensatz.

6.2.1 Konvergenz

Die Koeffizientenpfade von mboost, suggerieren eine große Sicherheit bei der Koeffizientenschätzung, denn die Koeffizienten konvergieren bei konvexen Optimierungsproblemen immer. Bei rmboost ist das nicht mehr der Fall, denn die Daten, auf denen das Grundmodell angepasst wird, entsprechen im Mittel nur 63,2 Prozent der gesamten Daten. Das hat zur Folge, dass das Modell über die Iterationsschritte hinweg irgendwann anfängt zu alternieren, wie man in Abbildung 6.5 sehen kann. Wenn man aber die interne Stoppmethode verwendet, wird das Modell nicht mehr in allen Schritten aktualisiert. Setzt man zusätzlich die Schwelle des Anteils der Modelle, die den Out-of-bag-Fehler verringern müssen, mit der Anzahl der Boosting Schritte herab, so konvergiert das Modell irgendwann, wie man in Abbildung 6.6 sehen kann.

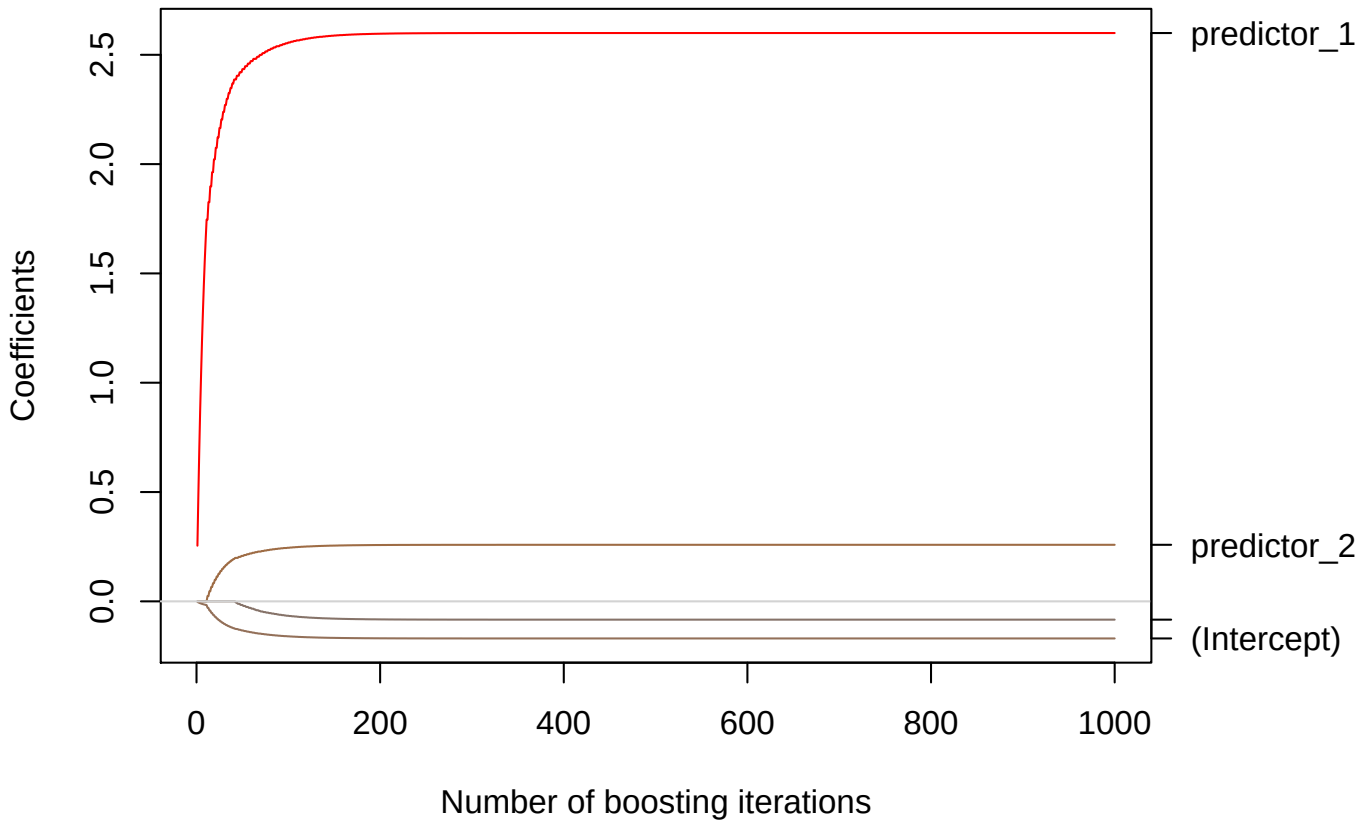

Abbildung 6.4: Koeffizienten Pfade von glmboost (mstop = 1000) für ein lineares Modell, angepasst an simulierte Daten 2

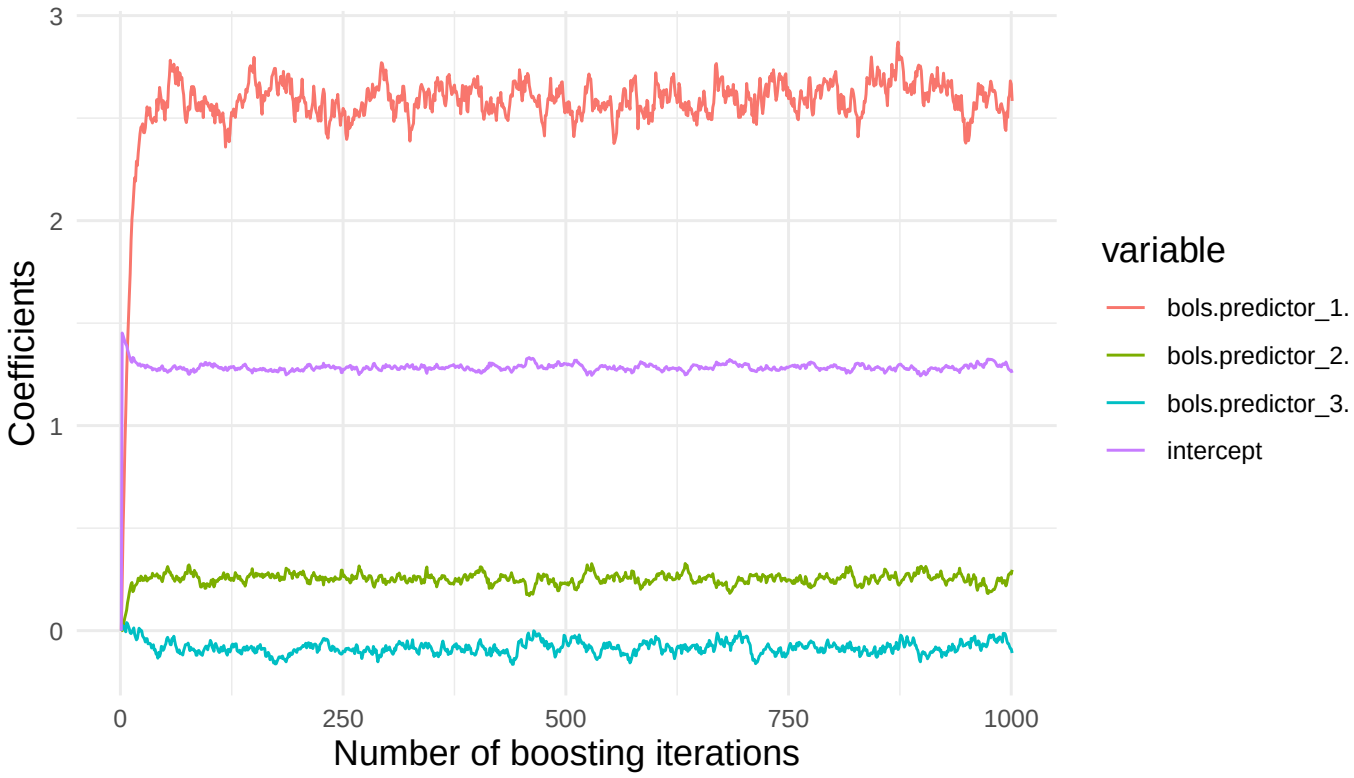

Abbildung 6.5: Koeffizienten Pfade von rmmboost (B = 10, nstop = 1000, mvar = 2) für ein lineares Modell, angepasst an simulierte Daten 2

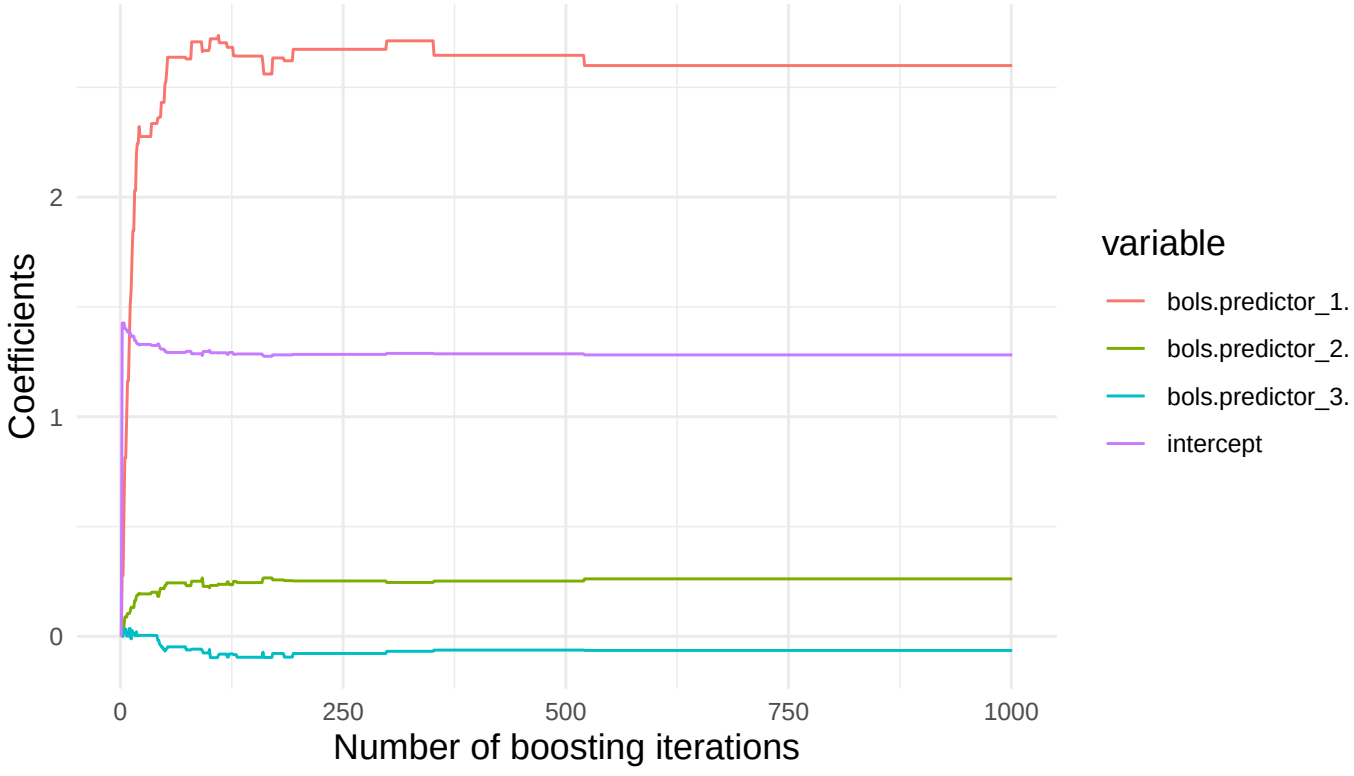

Abbildung 6.6: Koeffizienten Pfade von rmmboost (B = 10, nstop = 1000, stopintern = T, mvar = 2) für ein lineares Modell, angepasst an simulierte Daten 2

Auch bei diesem Datenbeispiel konvergiert rmboost schneller und konsistenter als mboost, wie Tabelle 6.4 zeigt.

Modell	rmboost(mvar = 3)	rmboost(mvar = 2)	mboost
Mittelwert	24.477	25.165	45.750
Standardabweichung	7.375	7.713	18.424

Tabelle 6.4: Konvergenzgeschwindigkeit Daten 1, B = 10, maxmstop = 100, simulierte Daten 2, 100 Simulationen

6.2.2 Vergleich mit anderen Modellen

Die Ergebnisse aus der vorherigen Simulation bestätigen sich auch für das Modell mit stärkeren Rauschen. Wie in Tabelle 6.5 zu erkennen ist, weist rmboost auch hier einen geringeren Bias auf, der durch die Regularisierung entsteht. Zusätzlich wird für Prediktor.3 der Koeffizient im Mittel niedriger geschätzt als bei den anderen Methoden. Die Regularisierung zeigt hier also bei rmboost einen minimal bessere Wirkung bei geringerem Bias. Über die Standardabweichung lässt sich keine klare Aussage treffen, da sich hier ein gemischtes Bild abzeichnet, siehe Tabelle 6.6.

Koeffizienten	Ridge	Lasso	Elastisches Netz	mboost	rmboost.2	rmboost.3	echt
Prediktor.1	2.960	2.931	2.944	2.97	2.993	2.995	3
Prediktor.2	0.194	0.183	0.188	0.191	0.197	0.197	0.2
Prediktor.3	0.003	0.004	0.004	0.003	0.003	0.003	0
Intercept	1.016	1.016	1.016	.	1.016	1.016	1

Tabelle 6.5: Vergleich von Koeffizienten verschiedener Modelle, angepasst an simulierte Daten 2

Koeffizienten	Ridge	Lasso	Elastisches Netz	mboost	rmboost.2	rmboost.3	echt
Prediktor.1	0.230	0.237	0.230	0.230	0.233	0.234	3
Prediktor.2	0.061	0.062	0.060	0.061	0.061	0.062	0.2
Prediktor.3	0.078	0.095	0.081	0.073	0.078	0.078	0
Intercept	0.219	0.219	0.238	.	0.218	0.218	1

Tabelle 6.6: Vergleich der Standardabweichung von Koeffizienten verschiedener Modelle, angepasst an simulierte Daten 2

6.3 Großes p, kleines n

Es werden die simulierten Daten 3 verwendet. Diese besitzen 4 Prädiktoren, die keinerlei strukturellen Zusammenhang mit der Zielgröße haben. Die Effekte werden in die Modelle als Polynom ersten bis fünften Grades aufgenommen. Somit ist $p = 20$ und $n = 100$.

Mboost findet selbst nach 10.000 Boosting Schritten und interner Stoppfunktion keine eindeutigen Parameter, wie man in Abbildung 6.7 erkennt. Ähnlich ist es bei rmboost, falls die interne Stoppfunktion ausgestellt ist. Abbildung 6.8 erinnert ein wenig an Random Walks. Bei Out-of-bag-Regularisierung wird das Modell irgendwann nicht mehr aktualisiert und das Modell findet eindeutige Werte. Elastische Netze finden jedoch bei Kreuzvalidierung eine deutlich sparsamere Lösung mit kleineren Koeffizienten. Jedoch ist es bei rmboost möglich, die Schwelle für den maximal erlaubten Out-of-bag-Fehler für neue Teilmodelle niedriger zu setzten und das Modell somit stärker zu Regulariseren. Dadurch kann eine noch sparsamere Lösung als die in Abbildung 6.7 gefunden werden.

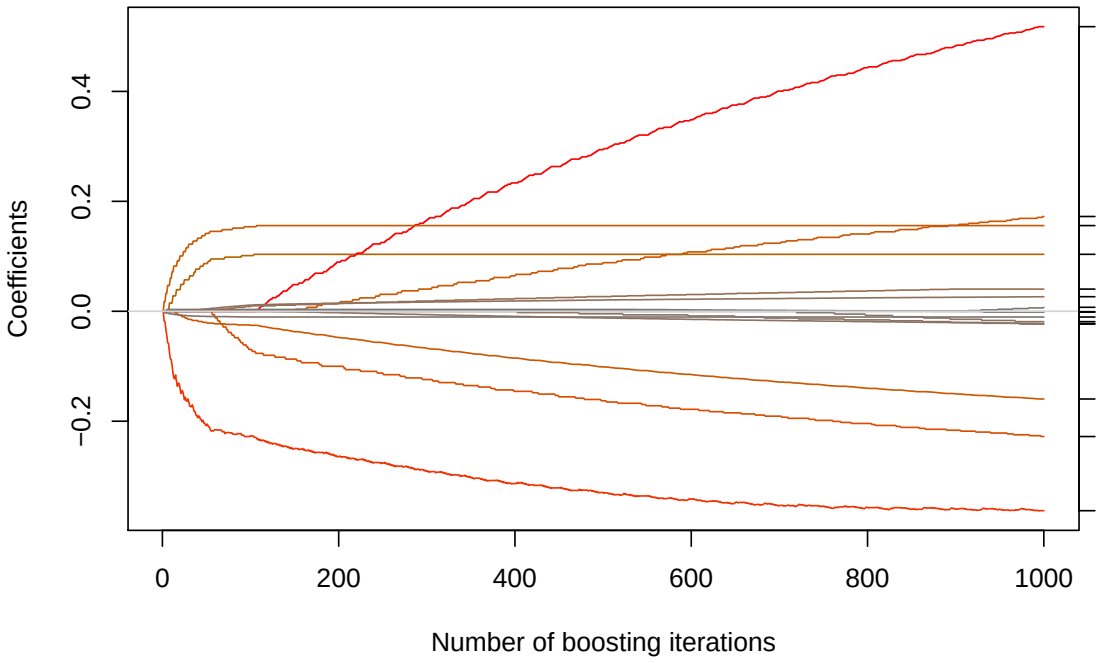

Abbildung 6.7: Koeffizienten Pfade von rmmboost (B = 10, nstop = 1000, stopintern = T, mvar = 2) für ein lineares Modell, angepasst an simulierte Daten 3

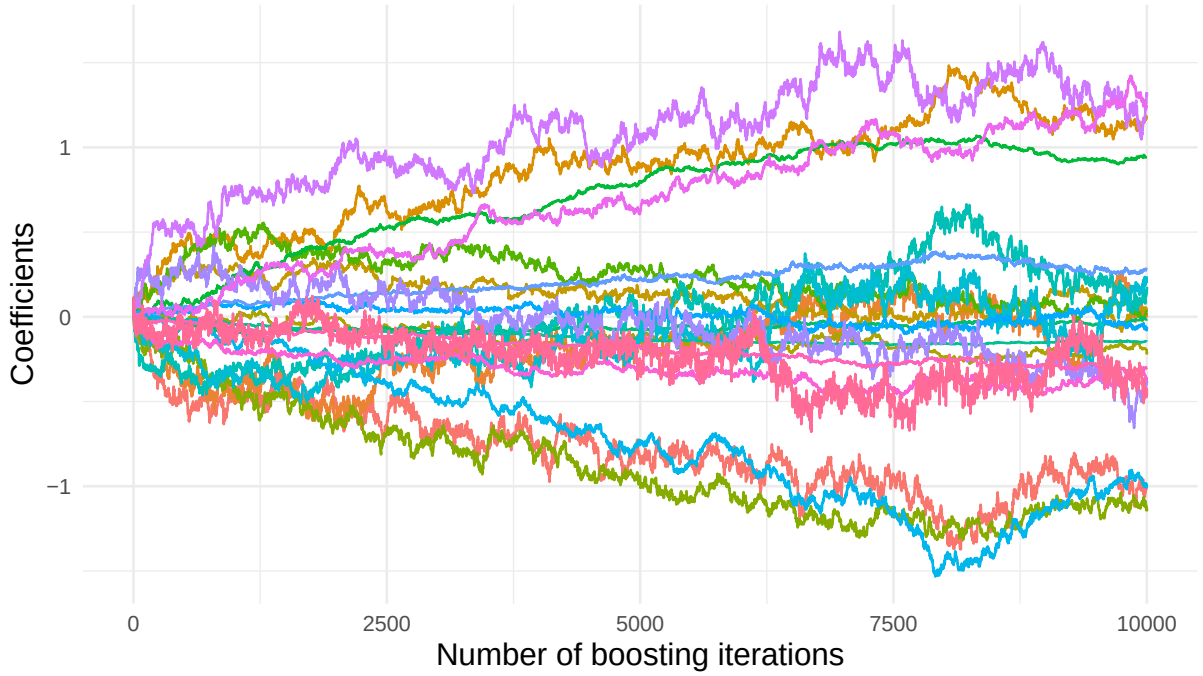

Abbildung 6.8: Koeffizienten Pfade von rmmboost (B = 10, nstop = 1000, stopintern = F, mvar = 3) für ein lineares Modell, angepasst an simulierte Daten 3

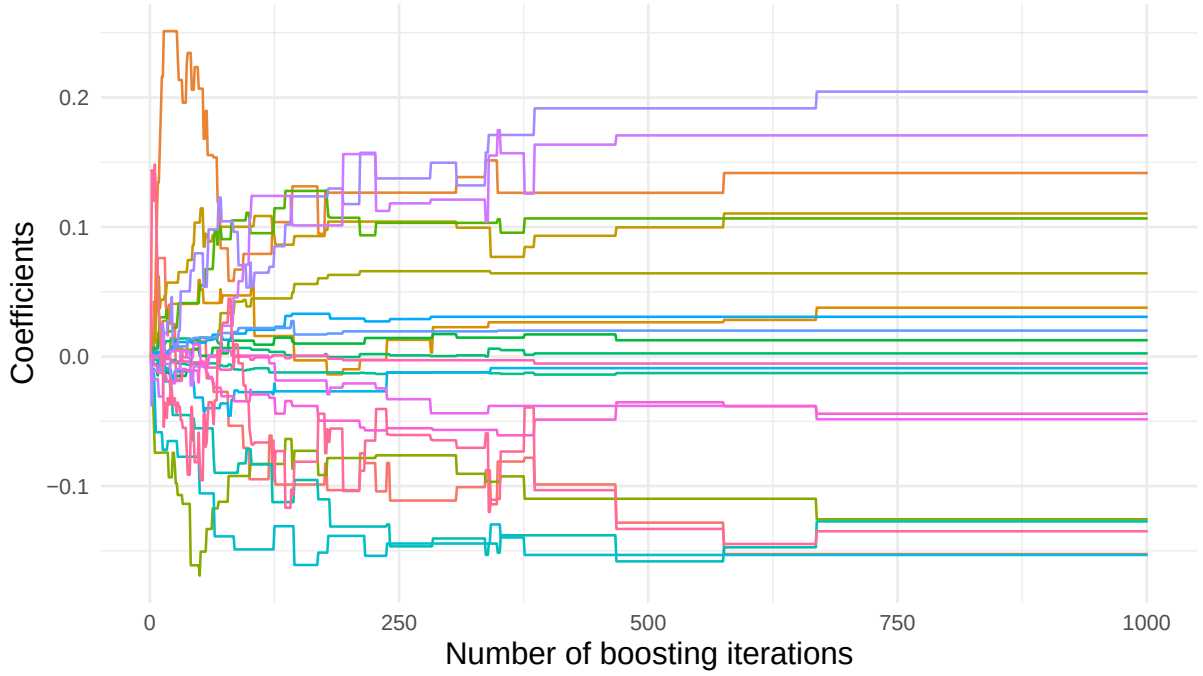

Abbildung 6.9: Koeffizienten Pfade von rmmboost (B = 10, nstop = 1000, stopintern = T, mvar = 3) für ein lineares Modell, angepasst an simulierte Daten 3

6.4 Multikollinearität

Ein häufiges Problem bei Inferenz ist Multikollinearität. Falls diese stark ausgeprägt ist, so ist auch die Schätzung der Koeffizienten instabil. Bei perfekter Multikollinearität existiert der KQ-Schätzer nicht. Bei einer Anpassung mit Boosting, kann in diesem Fall das Modell trotzdem angepasst werden. Jedoch entsprechen die Parameter in der Regel trotzdem nicht den Tatsächlichen. Denn häufig wird ein Parameter stärker gewichtet als er sollte und einer schwächer, siehe Abbildung 6.10. Bei rmboost jedoch ist dies nicht so ausgeprägt. Bei 100 facher Durchführung der Schätzung auf den simulierten Daten 4, wird bei mboost nach 100 Iterationen im Mittel der größte Parameter auf 1.99 (sd: 0.53) geschätzt, obwohl alle Parameter auf 1 geschätzt werden sollten. Bei rmboost hingegen wird der durchschnittliche größte Parameter auf 1.2 (sd: 0.13) gesetzt.
Daraus lässt sich ableiten, dass mboost bei hoher Multikollinearität eher eine Variable bevorzugt und rmboost den Effekt eher auf die korrelierten Variablen aufteilt. Das führt dann zu einer deutlich stabileren Schätzung der Parameter.

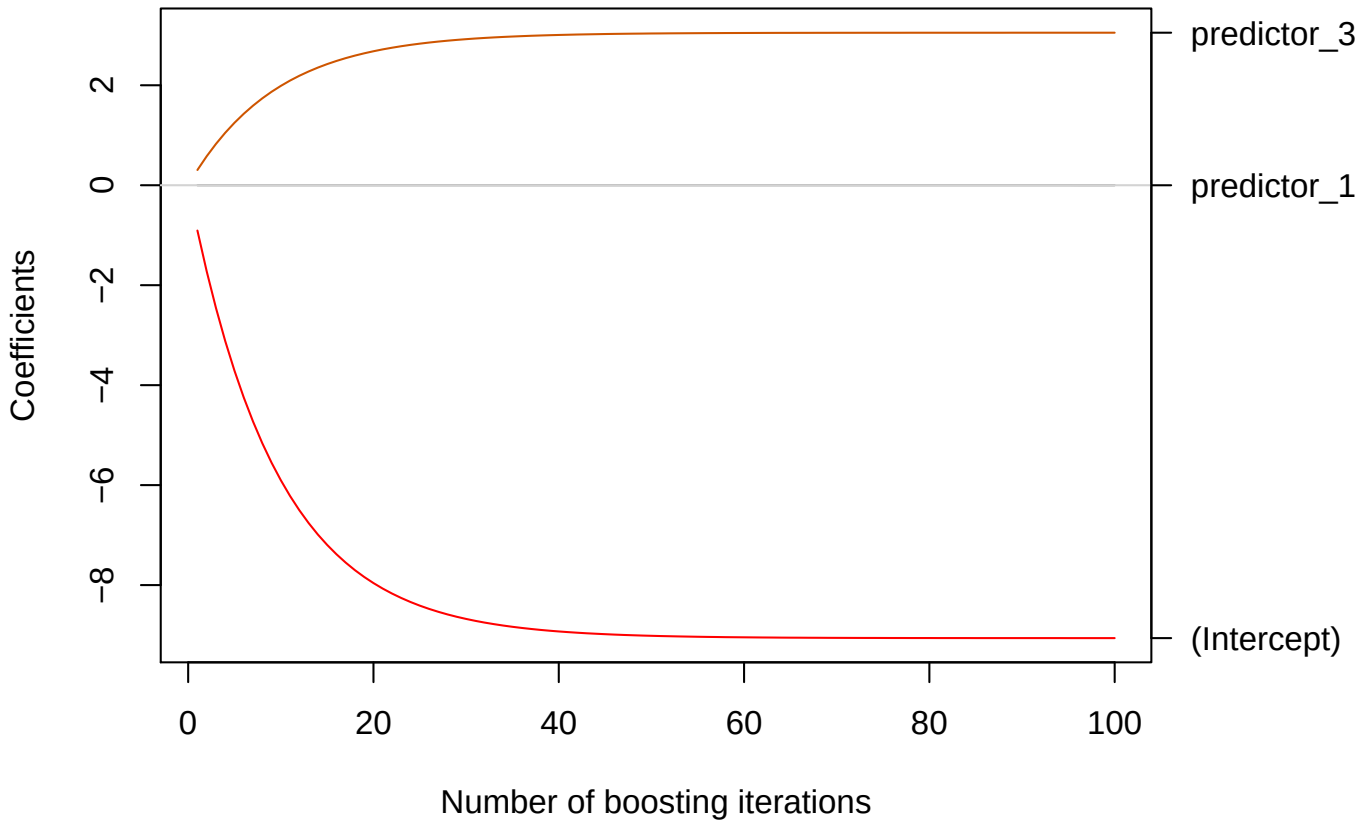

Abbildung 6.10: Koeffizienten Pfade von glmboost für ein lineares Modell, angepasst an simulierte Daten 4

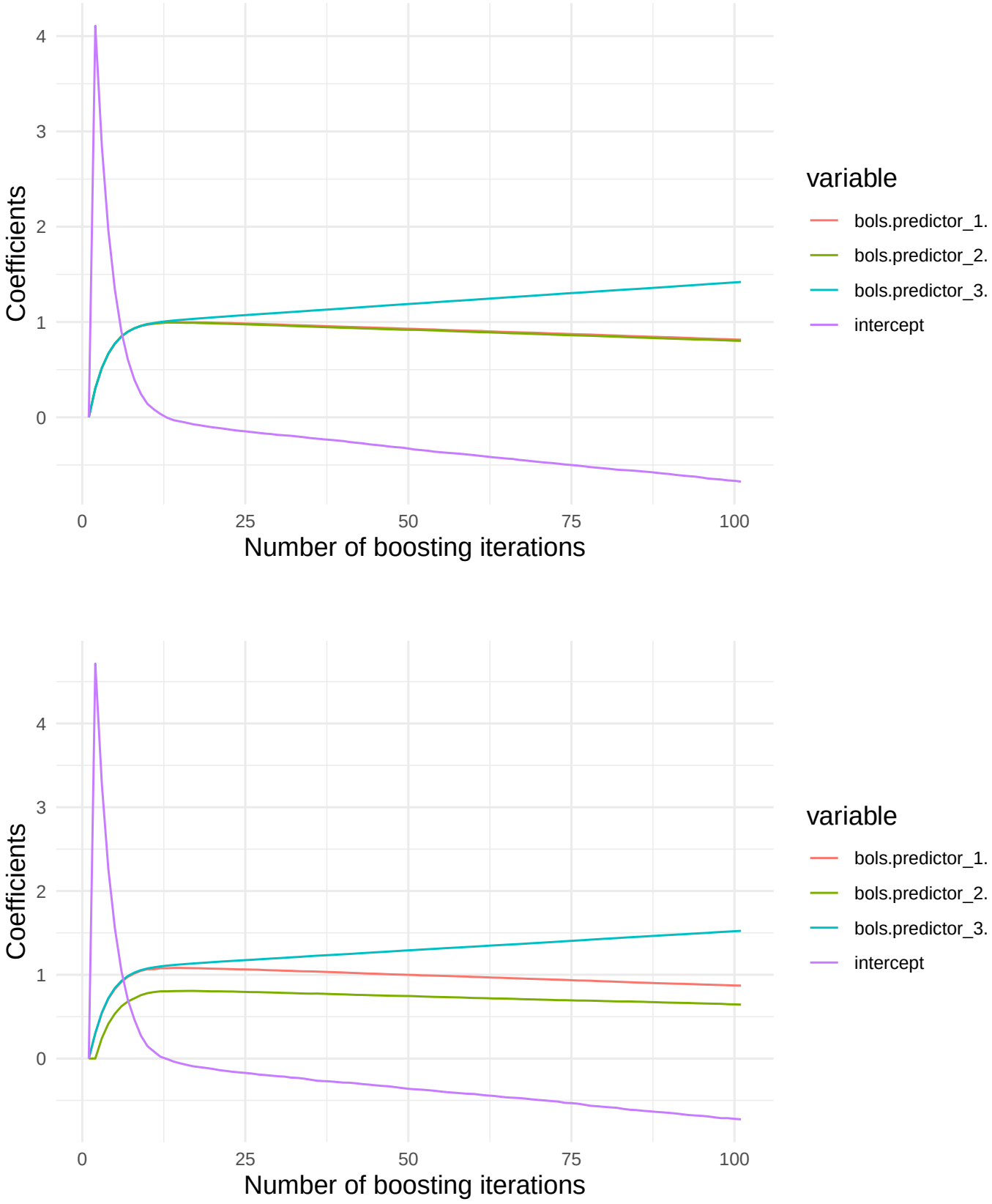

Abbildung 6.11: Koeffizienten Pfade von rmboost (mvar = 20 oben, mvar = 3 unten, B = 100) für ein lineares Modell, angepasst an simulierte Daten 4

Kapitel 7

Zusammenfassung und Ausblick

Resampling-Methoden innerhalb der einzelnen Schritte im modellbasierten Boosting können gewisse Vorteile gegenüber einer Anpassung ohne Resampling-Methoden haben. Einerseits lässt sich ableiten, dass das Modell mit Resampling in vielen Fällen in weniger Iterationsschritten als ohne konvergiert. Andererseits hat die Stoppzeit bei den in der Arbeit simulierten Daten eine geringere Standardabweichung. Die schnellere Konvergenz kann in Zusammenhang mit stochastischen Gradientenverfahren gestellt werden. Zusätzlich kommt der Effekt hinzu, dass in jedem Schritt bei durch das Resampling mehr als nur eine Variable aufgenommen werden kann.

Die schnelle Konvergenz bezüglich der Anzahl der Schritte hat den Preis, dass $B \cdot N$ mal so viele Modelle angepasst werden müssen, wie ohne Resampling. Hier entspricht B der Anzahl der Bootstrap Stichproben und N der Anzahl der Boosting-Iterationen. Jedoch kann die Anpassung der Modelle parallel erfolgen, beispielsweise mit dem "Map-Reduce"Verfahren [9]. Außerdem können die Out-of-sample-Residuen dazu verwendet werden, die Stoppzeit intern zu berechnen und eine Abschätzung des Vorhersagefehlers zu bekommen. Dadurch spart man sich die Anpassung der optimalen Stoppzeit, welche für "mboost"beispielsweise durch Kreuzvalidierung erfolgen kann. Diese hat eine vergleichbare Rechenzeit wie das randomisierte modellbasierte Boosting.

Bei sehr stark kollinearen Variablen kann eine Lösung gefunden werden, in die alle Variablen einfließen, die einen Einfluss haben. Mboost bevorzugt in diesen Fällen eine Variable gegenüber den anderen. Dafür führt mboost tendenziell eher zu einer sparsameren Lösung als rmboost.

Bisher wurde in dieser Arbeit das ransomisierte, modellbasierte Boosting in R für lineare Modelle implementiert. Jedoch lässt sich der Algorithmus auf die gleiche Art für allgemeines Gradientenboosting erweitern. Insbesondere die Erweiterung für genaralisierte lineare Modelle, additive Modelle und gemischte Modelle könnten zu sehr interessanten Ergebinssen führen. Für diese Modelle kann die Rechenzeit relevanter sein, als für klassische lineare Modelle, da inverse Matrizen häufiger berechnet werden müssen. Invertierung ist nicht nur rechenaufwendig, sondern auch schwerer zu parallelisieren als das Bagging.

Interessant wäre auch eine Analyse des Verhaltens des Modells bei zusätzlicher Integration von $\mathcal{L}^2$ Penalisierung. Die optimalen Gewichte der Penalisierung könnten nämlich über eine Gittersuche durch einen Teil der Bootstrap Stichproben geschätzt werden. Dadurch kann man Regressionsmodelle auf

eine sehr flexible Art und Weise bei komplexen Zusammenhängen anpassen und den Fluch der Dimensionalität ein wenig besser in den Griff bekommen.

Abbildungsverzeichnis

Literatur

[1] Yoshua Bengio und Yves Grandvalet. "No Unbiased Estimator of the Variance of K-Fold Cross-Validation". In: *J. Mach. Learn. Res.* 5 (2004), S. 1089–1105. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=1005332.1044695.

[2] Leo Breiman. "Random Forests". In: *Machine Learning* 45.1 (2001), S. 5–32. ISSN: 1573-0565. DOI: 10.1023/A:1010933404324. URL: https://doi.org/10.1023/A:1010933404324.

[3] Leo Breiman. "Using Adaptive Bagging to Debias Regressions". In: (1999).

[4] Leo Breiman und Philip Spector. "Submodel Selection and Evaluation in Regression. The X-Random Case". In: *International Statistical Review / Revue Internationale de Statistique* 60.3 (1992), S. 291–319. ISSN: 03067734, 17515823. URL: http://www.jstor.org/stable/1403680.

[5] Peter Bühlmann und Torsten Hothorn. "Boosting Algorithms: Regularization, Prediction and Model Fitting". In: *Statist. Sci.* 22.4 (2007), S. 477–505. DOI: 10.1214/07-STS242. URL: https://doi.org/10.1214/07-STS242.

[6] Bradley Efron und Robert J Tibshirani. *An Introduction to the Bootstrap*. CRC press, 1994.

[7] Jerome Friedman. "Stochastic Gradient Boosting". In: *Computational Statistics & Data Analysis* 38 (2002), S. 367–378. DOI: 10.1016/S0167-9473(01)00065-2.

[8] Jerome Friedman, Trevor Hastie und Robert Tibshirani. "Regularization Paths for Generalized Linear Models via Coordinate Descent". In: *Journal of Statistical Software* 33.1 (2010), S. 1–22. URL: http://www.jstatsoft.org/v33/i01/.

[9] A Ghoting u. a. "SystemML: Declarative machine learning on MapReduce". In: *2011 IEEE 27th International Conference on Data Engineering*. 2011, S. 231–242. DOI: 10.1109/ICDE.2011.5767930.

[10] Torsten Hothorn u. a. *{mboost}: Model-Based Boosting*. 2018. URL: https://cran.r-project.org/package=mboost.

[11] Donald W Marquardt und Ronald D Snee. "Ridge Regression in Practice". In: *The American Statistician* 29.1 (1975), S. 3–20. ISSN: 00031305. URL: http://www.jstor.org/stable/2683673.

[12] Llew Mason u. a. "Boosting Algorithms As Gradient Descent". In: *Proceedings of the 12th International Conference on Neural Information Processing Systems*. NIPS'99. Cambridge, MA, USA: MIT Press, 1999, S. 512–518. URL: http://dl.acm.org/citation.cfm?id=3009657.3009730.

[13] Annette M Molinaro, Richard Simon und Ruth M Pfeiffer. "Prediction Error Estimation : A Comparison of Resampling Methods". In: (2005).

[14] R Development Core Team. *R A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2008. URL: httpwww.R-project.org.

[15] Marco Sandri und Paola Zuccolotto. "Variable Selection Using Random Forests". In: *Data Analysis, Classification and the Forward Search*. 2006, S. 263–270. DOI: 10.1007/3-540-35978-8_30.

[16] Kesar Singh und Minge Xie. *Bootstrap: A Statistical Method*.

[17] Robert Tibshirani. "Regression Shrinkage and Selection via the Lasso". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 58.1 (1996), S. 267–288. ISSN: 00359246. URL: http://www.jstor.org/stable/2346178.

[18] W N Venables und B D Ripley. *Modern Applied Statistics with S*. Fourth. New York: Springer, 2002. URL: http://www.stats.ox.ac.uk/pub/MASS4.

[19] Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN: 978-3-319-24277-4. URL: http://ggplot2.org.

[20] Hadley Wickham u. a. *dplyr: A Grammar of Data Manipulation*. 2018. URL: https://cran.r-project.org/package=dplyr.

[21] Tong Zhang. "Solving Large Scale Linear Prediction Problems Using Stochastic Gradient Descent Algorithms". In: *Proceedings of the Twenty-first International Conference on Machine Learning*. ICML '04. New York, NY, USA: ACM, 2004, S. 116–. ISBN: 1-58113-838-5. DOI: 10.1145/1015330.1015332. URL: http://doi.acm.org/10.1145/1015330.1015332.

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig angefertigt, alle Zitate als solche kenntlich gemacht sowie alle benutzten Quellen und Hilfsmittel angegeben habe.

München, der 26.02.2020

.....

Fabian Obster