Anwendung von neuronalen Netzen, Projection Pursuits und CARTs für die Schätzer-konstruktion unter R

Diplomarbeit

von

Markus-Hermann Koch

29. Oktober 2002

Betreuer: Privat-Dozent Dr. M. Kohler
Mathematisch-Physikalische Fakultät
Universität Stuttgart

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen und Hilfsmittel benutzt habe.

Markus-Hermann Koch

Inhaltsverzeichnis

Abbildungsverzeichnis

0 Vorwort

Unter einem *Regressionsproblem* versteht man die Aufgabe, aus einem gegebenen d-Vektor von Regressoren X auf eine unbekannte, aber als von X abhängig angenommene, Response Y zu schließen.

Im Folgenden wird von einem $\mathbb{R}^d \times \mathbb{R}$-wertigen Zufallsvektor (X, Y) unbekannter Verteilung ausgegangen. Gesucht wird eine messbare Funktion $f : \mathbb{R}^d \to \mathbb{R}$, so dass $f(X)$ einen „guten“ Schätzer für Y liefert. Das heißt, für eine gegebene $\mathbb{R}_0^+$-wertige Verlustfunktion L soll die Zufallsvariable $L(f(X), Y)$ ein möglichst geringes Risiko $\mathbf{E}\{L(f(X), Y)\}$ aufweisen.

Die Wahl von L ist dabei prinzipiell beliebig und mag direkt von Überlegungen beeinflusst sein, die auf dem real entstehenden Schaden im Falle bestimmter Fehlprognosen fußen. In der vorliegenden Arbeit wird im Regelfall das handwerklich recht geschickte L_2-Risiko mit der Verlustfunktion $|f(X) - Y|^2$ herangezogen.

Ein Vorteil des L_2-Risikos ist, dass man die optimal zur Regression geeignete Funktion sofort aufschreiben kann: $m(X) = \mathbf{E}\{Y|X\}$. Dazu folgendes Lemma aus dem Vorlesungsskript [Koh01]:

Lemma 0.1
Sei (X, Y) ein $\mathbb{R}^d \times \mathbb{R}$-wertiger Zufallsvektor und sei $\mathcal{F}$ die Menge aller Abbildungen $\mathbb{R}^d \to \mathbb{R}$. Sei ferner $m(X) := \mathbf{E}\{Y|X\}$. Dann gilt:

$$\mathbf{E}\{|m(X) - Y|^2\} = \min_{f \in \mathcal{F}} \mathbf{E}\{|f(X) - Y|^2\}$$

Beweis: Sei $f \in \mathcal{F}$. Dann ist

$$\begin{aligned} \mathbf{E}\{|f(X) - Y|^2\} &= \mathbf{E}\{|(f(X) - m(X)) + (m(X) - Y)|^2\} \\ &= \mathbf{E}\{(f(X) - m(X))^2\} + \mathbf{E}\{(m(X) - Y)^2\} + 2A \end{aligned} \tag{0.1}$$

mit $A := \mathbf{E}\{(f(X) - m(X))(m(X) - Y)\} = 0$, denn

$$\begin{aligned} A &= \mathbf{E}\left\{(f(X) - m(X))\ (m(X) - Y)\right\} \\ &= \mathbf{E}\left\{\mathbf{E}\{(f(X) - m(X))\ (m(X) - Y)\ |X\}\right\} \\ &= \mathbf{E}\left\{(f(X) - m(X))\ \mathbf{E}\{(m(X) - Y)\ |X\}\right\} \\ &= \mathbf{E}\left\{(f(X) - m(X))\ (m(X) - \mathbf{E}\{Y\ |X\})\right\} \\ &= \mathbf{E}\left\{(f(X) - m(X))\ (m(X) - m(X))\right\} \\ &= 0 \end{aligned}$$

Damit erhält man aus (0.1):

$$\mathbf{E}\{|f(X) - Y|^2\} = \mathbf{E}\{(f(X) - m(X))^2\} + \mathbf{E}\{(m(X) - Y)^2\}$$

Hierbei ist $\mathbf{E}\{(m(X) - Y)^2\}$ von der Wahl von f unabhängig und $\mathbf{E}\{(f(X) - m(X))^2\} \geq 0$ mit Gleichheit falls $f = m$. □

Das Problem mit $m(X) = \mathbf{E}\{Y|X\}$ ist die im Allgemeinen unbekannte Verteilung von (X, Y). Im Regelfall liegt nur eine Stichprobe $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ vor.

Viele Regressionsmethoden setzen neben der Verwendung einer Stichprobe mit Annahmen zur Verteilung von (X, Y) oder der Art der Abhängigkeit $Y(X)$ an. Einige davon führen bereits bei kleinen Stichprobenumfängen auf sehr gute Schätzfunktionen. Vorausgesetzt die getroffenen Annahmen sind richtig.

An dieser Stelle setzt die *nichtparametrische Regressionsschätzung* an. Ihre Zielsetzung liegt im Finden von Regressionsschätzern, deren Konstruktion nur von $\mathcal{D}_n$ abhänget und die ohne Einschränkungen an die Allgemeinheit der Verteilung von (X, Y) auskommt.

Gegenstand dieser Diplomarbeit ist die Vorstellung und Anwendung von drei modernen, rechnerintensiven, iterativen, nichtparametrischen Verfahren. Die Rede ist von *neuronalen Netzen*, *Projection Pursuits* und Regressionsbäumen im *CART*-Stil.

Dabei gab es vier Zielsetzungen:

i. Motivation und Arbeitsweise der einzelnen Verfahren sollten erklärt werden. Dies geschieht in den ersten drei Kapiteln.

ii. Mit der Statistik-Software R sollte mit Hilfe der dort in den Paketen `nnet`, `modreg` und `tree` bereits implementierten Grundfunktionen ein Programmpaket erstellt werden, welches die erwähnten Verfahren auf eine gegebene Stichprobe anwendet. Die dabei entstandene Funktionensammlung `nonpar.r` befindet sich auf der beigelegten CD und wird im Anhang dokumentiert. Das ebenfalls auf der CD befindliche und in Kapitel vier dokumentierte Demoprogramm `demo.r` vermittelt einen Eindruck über die Funktionsweise der Funktionen in `nonpar.r`.

iii. Im Wintersemeseter 2001/2002 und im darauffolgenden Sommersemester wurde an der Universität Stuttgart die Vorlesung *Statistik für Wirtschaftswissenschaftler* gehalten. Beide Semester waren von einem umfangreichen Übungsgruppenbetrieb begleitet und endeten mit einer Prüfung. Die dritte Zielsetzung für diese Arbeit war nun, auf der Grundlage von Prüfungsnoten und in den Übungen erfassten Daten des Wintersemesters 2001/2002 eine Regressionsschätzfunktion zu konstruieren, die nach Ende des Übungsbetriebs im anschließenden Sommersemester zu einer Prognose für die Prüfung verwendet werden konnte. Für dreißig der sechzig schlechtest prognostizierten Prüflinge wurde daraufhin ein Prüfungsvorbereitungskurs angeboten.

iv. Schließlich, nachdem auch die Prüfungsnoten des Sommersemesters feststanden, sollte eine Analyse durchgeführt werden, die sowohl die Qualität des verwendeten Schätzers, als auch den Nutzen des erwähnten Zusatzkurses untersuchen sollte. Dies ist der Gegenstand des fünften Kapitels.

Noch einige Worte zur Notation: Zentrales Hilfsmittel der Schätzerkonstruktion ist die Verfügbarkeit von Stichproben. Darum wird eine praktische Schreibweise benötigt, um Stichproben gut handhaben zu können. In dieser Arbeit wird zu diesem Zweck eine Mengenschreibweise verwendet. Die Stichprobe wird notiert als

$$\mathcal{D}_n = \{(X_1, Y_1), \ldots (X_1, Y_1)\}$$

und ein einzelnes Paar aus der Probe wird des öfteren mit $(X_i, Y_i) \in \mathcal{D}_n$ referenziert. Diese Schreibweise ist nicht ganz korrekt, da $\mathcal{D}_n$ keine Menge im mathematischen Sinne ist und einzelne Realisierungswerte durchaus auch öfters in $\mathcal{D}_n$ enthalten sein können. Dennoch werden Mengenschreibweisen auf die Stichproben angewandt, da diese Notation intuitiv und praktisch ist.

1 Neuronale Netze

1.1 Historische Entwicklung

Über einen Jahrmillionen andauernden Zeitraum hinweg hat die Evolution auf der Erde das Gehirn als leistungsfähiges Kontrollorgan vieler komplexer Organismen hervorgebracht.

Mit dem Aufkommen leistungfsähiger Computer liegt nun der Versuch nahe, bei deren Programmierung dieses bewährte und perfektionierte System zu imitieren. Diesem Grundgedanken entspringt die Entwicklung selbsttrainierender, künstlicher, neuronaler Netze.

Die Grundlagen hierzu stammen aus dem Jahre 1943, als der Neurophysiologe Warren McCulloch und der Mathematiker Walter Pitts in ihrem Aufsatz *A logical calculus of the ideas immanent in nervous activity* das *McCulloch-Pitts-Neuron* entwarfen und zeigten, dass sich auch komplizierte logische und arithmetische Funktionen mit neuronalen Netzen beschreiben lassen.

Im Jahre 1986 entwickelten Rumelhart und McClelland in ihren Arbeiten das Verfahren der *Rückpropagierung*, auf das später in diesem Kapitel noch eingegangen wird.

1.2 Grundsätzlicher Aufbau neuronaler Netze

1.2.1 Biologische neuronale Netze

Der wichtigste Bestandteil des Gehirns sind die beim erwachsenen Menschen ca. 10^{11} Nervenzellen[1] (Neuronen). Sie sind in einem komplizierten Netzwerk über sogenannte Synapsen untereinander verschaltet und tauschen Informationen aus.

Ein Neuron besteht im Wesentlichen aus dem Zellkörper, einer Vielzahl von Dendriten, über welche die Zelle Signale von im Mittel 10.000 anderen Zellen aufnimmt, und dem Axon, über welches die Zelle selbst Impulse an andere Zellen aussendet.

[1] Dieser Abschnitt orientiert sich an [Hir]

Die Zelle empfängt hemmende oder erregende Signale aus den Dendriten. Diese addieren sich im Zellkörper auf und verursachen bei Überschreitung eines bestimmten Schwellwertes das Auslösen (Feuern) des Neurons. In diesem Fall sendet die Zelle einen ca. drei Millisekunden andauernden elektrischen Impuls von etwa einhundert Millivolt.

Der letztendliche Transport der Signale fällt den Synapsen zu. Dort werden bei Eintreffen eines elektrischen Impulses Übertragungsstoffe (Neurotransmitter) ausgeschüttet, die das Signal an mehrere tausend Zellen übertragen. Die Intensität dieser Übertragung kann für jede Zielzelle individuell stark ausgeprägt sein.

In der Biologie wird heute davon ausgegangen, dass sich ein Lernvorgang in der Modifikation von Synapsenausprägungen zwischen den Neuronen ausdrückt. Ein lernendes Gehirn verändert also seine physikalische Struktur.

1.2.2 Künstliche neuronale Netze

Seien im Folgenden X eine $\mathbb{R}^d$-wertige und Y eine in irgendeiner Weise von X abhängige, $\mathbb{R}$-wertige Zufallsvariable.

Gegenstand dieses Kapitels sind sogenannte *Feed Forward Neural Networks*. Von außen betrachtet ist ein solches Netz eine beschränkte Abbildung

$$\begin{aligned} f : \mathbb{R}^d &\to \mathbb{R} \\ \mathbf{x} &\mapsto y. \end{aligned}$$

Intern besteht f aus s getrennten, verdeckten Schichten von Neuronen, wobei die Ausgaben einer Schicht als Eingaben für die nächste fungieren.

Im Folgenden wird zunächst die Funktionsweise eines Neurons innerhalb des Netzes beschrieben. Sein Zellkörper besteht aus einer für das ganze Netz verbindlichen *Schwellwert-*, oder *Sigmoidfunktion*.

Definition 1.1 *Eine Abbildung $\sigma : \mathbb{R} \longrightarrow [0, 1]$ heisst* Sigmoidfunktion*, wenn sie folgenden Anforderungen genügt:*

1. *σ ist monoton nichtfallend,*
2. $\lim_{a \to -\infty} \sigma(a) = 0, \lim_{a \to \infty} \sigma(a) = 1$

In a sollen sich nun die akkumulierten Reize ausdrücken, die das Neuron aus der vorangegangenen Schicht empfängt. Der Parameter für das t-te Neuron in der j-ten Schicht des Netzes soll im Folgenden als $a_t^{(j)}$ bezeichnet werden. Er hat die Form

$$a_t^{(j)} := w_{0,t}^{(j-1)} + \sum_{i=1}^{l_{j-1}} w_{i,t}^{(j-1)} o_i^{(j-1)}. \tag{1.1}$$

Hierbei und im Folgenden sind

- $w_{0,t}^{(j-1)}$ ein Offset, der es erlaubt, die Schwellwerte einzelner Neuronen trotz global verbindlicher Sigmoidfunktion individuell einzustellen. Man kann sich hierbei für jede verdeckte Schicht ein 0-tes Neuron denken, welches stets den Wert $o_0^{(j-1)} = 1$ liefert. In diesem Fall ist $w_{0,t}^{(j-1)}$ ein Maß für die Ausprägung der Synapse, die dieses Offsetneuron mit dem t-ten Neuron der j-ten Schicht verbindet.
- l_{j-1} die Anzahl der Neuronen in der $(j-1)$-ten Schicht.
- $o_i^{(j-1)}$ der Ausgabewert des i-ten Neurons in der $(j-1)$-ten Schicht, also der Reiz, den dieses an die j-te Schicht abgeschickt hat.
- $w_{i,t}^{(j-1)} \in \mathbb{R}$ das Gewicht, welches die Ausgabe des i-ten Neurons der $(j-1)$-ten Schicht für das t-te Neuron der j-ten Schicht haben soll, also ein Maß für die Ausgeprägtheit der beide Zellen verbindenden Synapse.

Damit erhält man für die Ausgabe $o_t^{(j)}$ des t-ten Neurons in der j-ten verdeckten Schicht:

$$o_t^{(j)} = \sigma(a_t^{(j)}) = \sigma\left(w_{0,t}^{(j-1)} + \sum_{i=1}^{l_{j-1}} w_{i,t}^{(j-1)} o_i^{(j-1)} \right) = \sigma(w_{0,t}^{(j-1)} + \mathbf{w}_t^{(j-1)} \mathbf{o}^{(j-1)}). \tag{1.2}$$

Das Neuron ist in Abbildung 1.1 noch einmal verdeutlicht dargestellt.

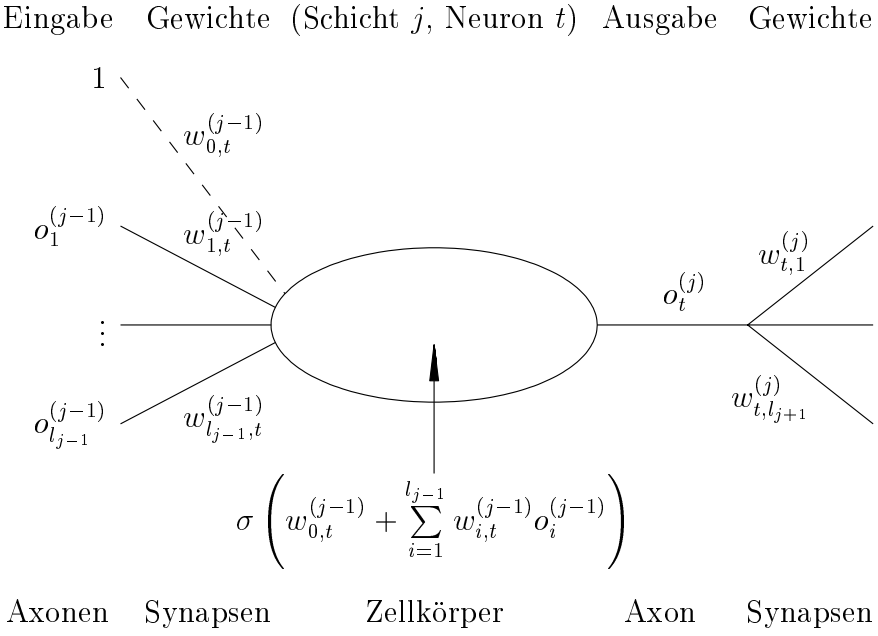

Abbildung 1.1: Ein Neuron und seine Umgebung

Nun muss noch geklärt werden:

1. Wie gelangt die Eingabe $\mathbf{x}$ zu den Neuronen der ersten Schicht?
2. Wie gewinnt man aus den Ausgaben der letzten verdeckten, s-ten Schicht den Regressionsschätzwert y?

Sei ein *Dummy-Neuron* ein Neuron, welches anstelle einer Sigmoidfunktion die Identität hat.

Zur Beantwortung der ersten Frage wird zunächst der Eingabevektor zu $\tilde{\mathbf{x}} = (1, x_1, \ldots, x_{d_X})$ erweitert. Dann werden seine einzelnen Einträge in eine gedachte 0-te Schicht aus $(d+1)$ Dummy-Neuronen eingespeist, die über Synapsengewichte $w_{i,j}^{(0)}$ mit den einzelnen Neuronen der ersten verdeckten Schicht verbunden werden. Die zusätzliche 1 stellt die Ausgabe eines Offsetneurons dar. Es ist also $\mathbf{o^{(0)}} = \tilde{\mathbf{x}}$.

Analog werden die Ausgaben der Neuronen in der s-ten Schicht über Gewichte $w_{i,j}^{(s)}$ an eine gedachte $(s+1)$-te Schicht aus einem Dummy-Neuron geschickt. Es ist dann $y = f(\mathbf{x}) = a^{(n+1)}$.

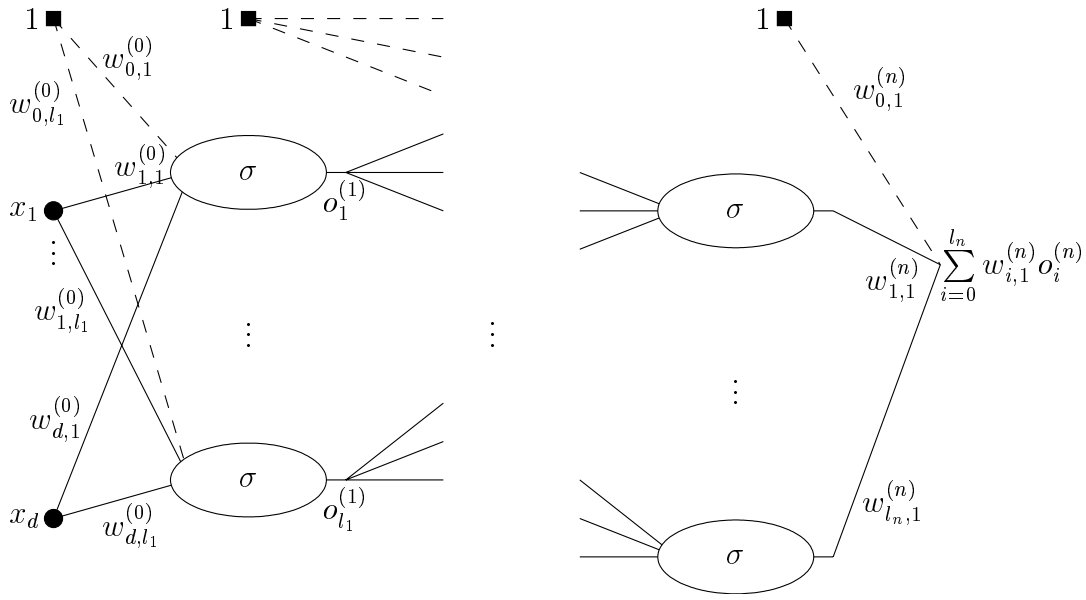

Abbildung 1.2: Schematischer Aufbau eines neuronalen Netzes

Abbildung 1.2 verdeutlicht beide Vorgänge.

Bemerkungen:

1. Um ein wie hier beschriebenes Netz vollständig zu charakterisieren, benötigt man die Sigmoidfunktion der Neuronen, die Anzahl s der verdeckten Schichten, deren einzelne Neuronenanzahlen l_j mit $j \in \{1, \ldots, s\}$, sowie die $\sum_{i=0}^{s}(l_i + 1)l_{i+1}$ (mit $l_0 := d$ und $l_{s+1} := 1$) Gewichte für Offsets und Synapsen.

2. Die explizite Wahl der Sigmoidfunktion ist recht willkürlich. In der Natur feuert ein Neuron, oder es feuert nicht. Um dies zu modellieren, würde man eine Indikatorfunktion heranziehen. Für künstliche neuronale Netze ist aber, mit Blick auf die später beschriebene Rückpropagierung zur Bestimmung der einzelnen Gewichte, eine bestenfalls beliebig oft stetig differenzierbare Sigmoidfunktion von Vorteil. Darum wird beispielsweise in der entsprechenden R-Routine die sogenannte *logistische Sigmoidfunktion* verwendet mit $\sigma(a) = \frac{1}{1+e^{-a}}$.

1.3 Gleichmäßige Approximation auf kompakten Mengen

Das Ziel ist, Funktionen $m : \mathbb{R}^d \to \mathbb{R}$ durch neuronale Netze zu approximieren. Bisher wurde allerdings noch keine Aussage darüber getroffen, wie gut diese Näherung bei allgemein gefaßtem m überhaupt sein kann. Für stetige m wird diese Lücke gefüllt durch den nun folgenden, auch in [KH93] oder [MK01] zu findenden, Satz:

Satz 1.2
Sei σ eine Sigmoidfunktion und K eine kompakte Teilmenge des $\mathbb{R}^d$. Dann existiert für jede stetige Funktion $m : \mathbb{R}^d \to \mathbb{R}$ und für jedes $\varepsilon > 0$ ein neuronales Netz mit einer verdeckten Schicht

$$h(\mathbf{x}) = \sum_{i=1}^{k} c_i \sigma(\mathbf{a}_i^T \mathbf{x} + b_i) + c_0 \text{ mit } \mathbf{a}_i \in \mathbb{R}^d, b_i, c_i \in \mathbb{R},$$

so dass

$$\sup_{\mathbf{x} \in K} |m(\mathbf{x}) - h(\mathbf{x})| < \varepsilon$$

Der Beweis verwendet folgende Definitionen, sowie den hier nicht bewiesenen Satz von Stone-Weierstrass[2].

Definition 1.3 *Sei K eine Menge und $\mathcal{F}$ eine Familie von Funktionen die von K nach $\mathbb{R}$ abbilden.* $\mathcal{F}$ separiert *die Punkte in K, falls für $\mathbf{x}, \mathbf{y} \in K$ mit $\mathbf{x} \neq \mathbf{y}$ ein $f \in \mathcal{F}$ existiert mit $f(\mathbf{x}) \neq f(\mathbf{y})$. Ferner sagt man,* $\mathcal{F}$ verschwinde auf keinem Punkt *von K, falls für jedes $\mathbf{x} \in K$ ein $f \in \mathcal{F}$ gefunden werden kann mit $f(\mathbf{x}) \neq 0$.*

Satz 1.4 Stone-Weierstrass
Sei K eine kompakte Menge und sei $\mathcal{F}$ eine Algebra auf dem Raum $C_K(\mathbb{R})$ der reellwertigen, stetigen Funktionen auf K. $\mathcal{F}$ separiere die Punkte von K und verschwinde in keinem Punkt von K. Dann ist $\mathcal{F}$ gleichmäßig dicht in $C_K(\mathbb{R})$.

Beweis zu 1.2: Der Beweis gliedert sich in mehrere Schritte:

Zunächst wird eine spezielle Familie von Funktionen $\mathcal{F}$ betrachtet und gezeigt, dass sie den Bedingungen des Satzes von Stone-Weierstrass genügt und

[2] siehe auch [Rud64]

somit gleichmäßig dicht in $C_K(\mathbb{R})$ ist. Dann ist $\overline{\mathcal{F}} = C_K(\mathbb{R})$ und

$$\forall\ \varepsilon > 0\ \forall\ m \in C_K(\mathbb{R})\ \exists\ f \in \mathcal{F}:\ \sup_{\mathbf{x}\in K}\{|f(\mathbf{x}) - m(\mathbf{x})| < \varepsilon\}.$$

Im zweiten Schritt wird diese Funktionenklasse durch neuronale Netze mit einer speziellen Sigmoidfunktion $c(\cdot)$ überdeckt.

Dann werden Netze beliebiger Sigmoidfunktionen herangezogen, um wiederum $c(\cdot)$ zu approximieren.

Schließlich wird die Eingabe dieser mit einem eindimensionalen Parameter $u \in \mathbb{R}$ versehenen Näherung auf $x \in \mathbb{R}^d$ erweitert und der Beweis abgeschlossen.

Schritt 1: Betrachtet wird die Funktionenklasse

$$\mathcal{F} = \left\{ \sum_{i=1}^{k} c_i \cos(\mathbf{a}_i^T \mathbf{x} + b_i) \;\middle|\; k \in \mathbb{N}, \mathbf{a}_i \in \mathbb{R}^d, b_i, c_i \in \mathbb{R} \right\}$$

a) $\mathcal{F}$ ist abgeschlossen bezüglich Addition, Multiplikation[3] und Multiplikation mit Skalaren und damit eine Algebra.

b) $\mathcal{F}$ separiert die Punkte in K: Für $\mathbf{x}, \mathbf{y} \in K$ mit $\mathbf{x} \neq \mathbf{y}$ nimmt man ein $\mathbf{a} \in \mathbb{R}^d$ mit $\mathbf{a}^T\mathbf{x}, \mathbf{a}^T\mathbf{y} \in (-\pi, \pi)$ und $\mathbf{a}^T\mathbf{x} \neq \mathbf{a}^T\mathbf{y}$ und setzt $b = 0$. Damit erhält man $f_{\mathbf{a},b}(x) = \cos(a^T x + 0) \neq \cos(a^T y + 0) = f_{\mathbf{a},b}(y)$ für $f_{\mathbf{a},b} \in \mathcal{F}$.

c) $\mathcal{F}$ verschwindet in keinem Punkt von K: Seien $a, b = 0$ und wieder $\mathbf{x} \in K$. Dann ist $f_{a,b}(x) = \cos(0x + 0) = 1$.

$\mathcal{F}$ erfüllt also die Bedingungen des Satzes von Stone-Weierstrass und ist damit gleichmäßig dicht in $C_K(\mathbb{R})$.

Schritt 2: Sei die Sigmoidfunktion

$$c(x) := \frac{1}{2}(1 + \cos(x + \frac{3}{2}\pi))\mathbf{I}_{\{x\in[-\frac{\pi}{2},\frac{\pi}{2}]\}} + \mathbf{I}_{\{x>\frac{\pi}{2}\}}$$

vorgegeben.

Nun soll gezeigt werden, dass es für beliebige $f \in \mathcal{F}$ ein neuronales Netz f^N mit der Sigmoidfunktion c gibt, so dass $f^N = f$.

[3] mit $\cos a \cos b = \frac{1}{2}(\cos(a+b) + \cos(a-b))$

f ist nur eine Linearkombination aus Elementarfunktionen $\cos(u) := \cos(\mathbf{a}_i^T\mathbf{x} + b_i)$. Da $\mathbf{a}_i, \mathbf{b}_i = \text{const}$ und $x \in K$ kompakt, gibt es ein $M \in \mathbb{R}$, so dass $u \in [-M, M]$.

Darum, und da Linearkombinationen von neuronalen Netzen derselben Sigmoidfunktion wieder neuronale Netze bilden, genügt es, $\cos(u)$ auf einem endlichen Intervall $[-M, M]$ mit f^N zu beschreiben.

Sei f^N ein Netz der Form

$$f^N(u) = \sum_{i=1}^{l} \gamma_i c(\alpha_i u + \beta_i) \text{ mit } l \in \mathbb{N}, \alpha_i, \beta_i, \gamma_i \in \mathbb{R}.$$

Hierzu zwei Beobachtungen:

i. Die β_i ermöglichen ein Verschieben der Funktion entlang der Abszisse

ii. Durch Einfügen eines zusätzlichen Summanden $\gamma_{l+1} c(\pi/2) = \gamma_{l+1}$ läßt sich der Graph entlang der Ordinate verschieben.

Solcherlei gerüstet reicht es, wenn man ein $f^N = (\cos(u) + 1)\mathbf{I}_{\{x \in [-\pi, \pi]\}}$ findet. Dieses kann dann durch endliche Summation verschobener Kopien seiner selbst auf das ganze Intervall $[-M, M]$ ausgedehnt werden.

Das Netz mit $(l = 2, \alpha_i = 1, \beta_i = \frac{\pi}{2} \cdot (-1)^{(i-1)}, \gamma_i = 2 \cdot (-1)^{(i-1)})$ leistet das Gewünschte:

$$\begin{aligned} f^N(u) &= 2\left[c(u + \frac{\pi}{2}) - c(u - \frac{\pi}{2})\right] \\ &= 2[\frac{1}{2}(1 + \cos(u + 2\pi))\mathbf{I}_{\{u \in [-\pi, 0]\}} + \mathbf{I}_{\{u > 0\}} - \\ &\qquad \frac{1}{2}(1 + \cos(u + \pi))\mathbf{I}_{\{u \in [0, \pi]\}} - \mathbf{I}_{\{u > \pi\}}] \\ &= (1 + \cos(u))\mathbf{I}_{\{u \in [-\pi, \pi]\}} \end{aligned} \tag{1.3}$$

Für den Fall neuronaler Netze mit der Sigmoidfunktion c ist der Beweis damit abgeschlossen.

Schritt 3: Im weiteren Verlauf soll nun gezeigt werden, dass bei beliebiger Sigmoidfunktion σ und für beliebiges $\varepsilon > 0$ ein neuronales Netz

$$h_\varepsilon(u) = c_0 + \sum_{i=1}^{k} c_i \sigma(a_i u + b_i) \tag{1.4}$$

existiert mit

$$\sup_{u\in\mathbb{R}} |h_\varepsilon(u) - c(u)| < \varepsilon \tag{1.5}$$

Tatsächlich wird sogar ein etwas einfacheres Netz

$$h_\varepsilon(u) = \frac{1}{k}\sum_{i=1}^{k-1} \sigma(a_i u + b_i) \tag{1.6}$$

diesem Anspruch gerecht. Doch vor der Abschätzung noch einige Überlegungen:

Seien im Folgenden $\varepsilon \in (0,1)$ und $k > 4$ genügend groß, dass $1/k < \varepsilon/4$.

a. Da σ eine Sigmoidfunktion ist, findet man ein $M > 0$ mit

$$\sigma(-M) < \frac{\varepsilon}{2k} \tag{1.7}$$

und

$$\sigma(M) > 1 - \frac{\varepsilon}{2k} \tag{1.8}$$

Da ferner c streng monoton steigend ist, gibt es für $i = 1, ..., k-1$ eindeutige, paarweise verschiedene r_i mit $c(r_i) = i/k$, außerdem ein $r_k > r_{k-1}$ mit $c(r_k) = 1 - \frac{1}{2k}$.

b. Durch das Gleichungssystem

$$\begin{pmatrix} r_i & 1 \\ r_{i+1} & 1 \end{pmatrix} \begin{pmatrix} a_i \\ b_i \end{pmatrix} = \begin{pmatrix} -M \\ M \end{pmatrix}$$

mit den Lösungen $a_i = \frac{2M}{r_{i+1}-r_i}$ und $b_i = \frac{M(r_i+r_{i+1})}{r_i - r_{i+1}}$ lassen sich Geraden $a_i u + b_i$ durch die Punkte $(r_i, -M), (r_{i+1}, M)$ konstruieren. Hierbei ist $0 < a_i < \infty$.

c. Für den Fall, dass $u \in (r_i, r_{i+1}]$ fällt, liefert Punkt (a), dass $c(u) \in (\frac{i}{k}, \frac{i+1}{k})$.

d. Ferner ist für $j \in \{1, ..., i-1\}$, $u \in (r_i, r_{i+1}]$

$$\sigma(a_j u + b_j) \geq \sigma(a_j r_{j+1} + b_j) = \sigma(M) > 1 - \frac{\varepsilon}{2k}. \tag{1.9}$$

Das „$\geq$“ erhält man aus $u > r_i \geq r_{j+1}$, das „$=$“ aufgrund der Wahl der a_j und b_j wie in (b).

e. Analog ist für $j \in \{i+1, ..., k-1\}$, $u \in (r_i, r_{i+1}]$

$$\sigma(a_j u + b_j) \leq \sigma(a_j r_j + b_j) = \sigma(-M) < \frac{\varepsilon}{2k} \tag{1.10}$$

Die nun folgenden Abschätzungen sollen zeigen, dass (1.5) für $u \in \mathbb{R}$ gilt. Dies geschieht in drei Schritten: Für $u \in (r_1, ..., r_k]$, $u \in (-\infty, r_k]$ und $u \in (r_k, \infty)$.

i. Sei $u \in (r_i, r_{i+1}]$. Mit (d) und (e) macht es Sinn (1.6) umzuschreiben in

$$h_\varepsilon(u) = \frac{1}{k} \sum_{j=1}^{i-1} \sigma(a_j u + b_j) + \sigma(a_i u + b_i) + \sum_{j=i+1}^{k-1} \sigma(a_j u + b_j) \tag{1.11}$$

Dann läßt sich (1.5) zeigen über

$$\begin{aligned} |c(u) - h_\varepsilon(u)| \leq & \left| c(u) - \frac{1}{k} \sum_{j=1}^{i-1} \sigma(a_j u + b_j) \right| \\ & + \frac{1}{k} \sigma(a_i u + b_i) + \frac{1}{k} \sum_{j=i+1}^{k-1} \sigma(a_j u + b_j) \end{aligned}$$

Einfügen von $-\frac{i-1}{k} + \frac{i-1}{k}$ in den Betragsblock und Anwendung der Dreiecksungleichung:

$$\begin{aligned} \leq & \left| c(u) - \frac{i-1}{k} \right| + \left| \frac{i-1}{k} - \frac{1}{k} \sum_{j=1}^{i-1} \sigma(a_j u + b_j) \right| \\ & + \frac{1}{k} \sigma(a_i u + b_i) + \frac{1}{k} \sum_{j=i+1}^{k-1} \sigma(a_j u + b_j) \end{aligned}$$

Das nun folgende „$\leq$“ gilt im

- linken Block, da $u \in (r_i, r_{i+1}]$ und $c(r_{i+1}) = i + 1/k$.
- mittleren Block wegen (d).
- rechten Block durch $\sigma(\cdot) < 1$ und wegen (e).

$$
\begin{aligned}
&\leq \left(\frac{i+1}{k} - \frac{i-1}{k}\right) + \left(\frac{i-1}{k} - \frac{1}{k}(i-1)(1-\frac{\varepsilon}{2k})\right) \\
&\quad + \left(\frac{1}{k} + \frac{1}{k}(k-1-i)\frac{\varepsilon}{2k}\right) \\
&= \frac{2}{k} + \frac{1}{k}(i-1)\frac{\varepsilon}{2k} + \frac{1}{k} + \frac{1}{k}(k-1-i)\frac{\varepsilon}{2k} \\
&\leq \frac{3}{k} + \frac{\varepsilon}{2k}
\end{aligned}
$$

da nach Voraussetzung $1/k < \varepsilon/4$:

$$< \frac{3}{4}\varepsilon + \frac{1}{8}\varepsilon^2$$

und da ferner $\varepsilon < 1$:

$$< \varepsilon$$

ii. Sei $u \in (-\infty, r_1]$. Dann folgt aufgrund von $\sigma(a_i u + b_i) < \sigma(a_i r_1 + b_i) = \sigma(-M) < \frac{\varepsilon}{2k}$, $c(u) < c(r_1) = \frac{1}{k}$ und $h_\varepsilon(u), c(u) \geq 0$, sowie $1/k < \varepsilon/4$ und der Monotonie von c und h_ε

$$|c(u) - h_\varepsilon(u)| \leq \max\{\frac{1}{k}, \frac{k-1}{k}\,\frac{\varepsilon}{2k}\} < \max\{\frac{\varepsilon}{4}, \varepsilon\} = \varepsilon \tag{1.12}$$

iii. Sei $u \in (r_k, \infty)$. Dann ist, da $c(u), h_\varepsilon(u) \in [0,1]$

$$
\begin{aligned}
|c(u) - h_\varepsilon(u)| &= |1 - c(u) - (1 - h_\varepsilon(u))| \\
&\leq \max\{|1 - c(u)|, |1 - h_\varepsilon(u)|\}
\end{aligned}
$$

wegen $c(u) > c(r_k) = 1 - \frac{1}{2k}$, $\sigma(u) > 1 - \frac{\varepsilon}{2k}$ und der Monotonie von c und h_ε:

$$
\begin{aligned}
&\leq \max\{1 - (1 - \frac{1}{2k}), 1 - \frac{k-1}{k}(1 - \frac{\varepsilon}{2k})\} \\
&= \max\{\frac{1}{2k}, \frac{1}{k} + \frac{\varepsilon}{2k}(1 - \frac{1}{k})\} \\
&= \frac{1}{k} + \frac{\varepsilon}{2k}(1 - \frac{1}{k})
\end{aligned}
$$

mit $1/k < \varepsilon/4$ und $\varepsilon \in (0,1)$:

$$
\begin{aligned}
&< \frac{\varepsilon}{4} + \frac{\varepsilon}{2k} \\
&< \varepsilon
\end{aligned}
$$

Schritt 4: In Schritt 2 wurde mit Hilfe von $c(u)$ der Ausdruck $(\cos(u)+1)$ exakt beschrieben. In Schritt 3 wurde ein neuronales Netz gefunden, welches $\sup_{u\in\mathbb{R}} |h_\varepsilon(u) - c(u)| < \varepsilon$ erfüllt. Kombiniert man nun (1.3) mit (1.5), erhält man

$$\begin{aligned}&\sup_{u\in[-\pi,\pi]} [\cos(u)+1] - 2[h_\varepsilon(u+\frac{\pi}{2}) - h_\varepsilon(u-\frac{\pi}{2})] \\ &\quad =: \sup_{u\in[-\pi,\pi]} [\cos(u)+1] - \overline{C}_{[-\pi,\pi],4\varepsilon,c_0=0}(u) < 4\varepsilon.\end{aligned}$$

Approximiert man nun auf einem beliebigen, kompakten Intervall $[-M,M]$ den $\cos(u)$ durch Addition endlich (q) vieler Funktionen $\overline{C}_{[p\pi,(p+2)\pi],4\varepsilon,c_0=0}$ mit $p \in (2\mathbb{Z}-1)$, so folgt

$$\sup_{u\in[-M,M]} [\cos(u)+1] - \sum_p \overline{C}_{[p\pi,(p+2)\pi],4\varepsilon,c_0=0}(u) < 4q\varepsilon$$

Fügt man nun diese Summe von Netzen zu einem großen Netz $\overline{C}_{M,\varepsilon}(u)$ zusammen, verleiht ihm das Absolutglied $c_0 = -1$ und passt ε und k entsprechend an, so erhält man als Zwischenergebnis die Form

$$\overline{C}_{M,\varepsilon}(u) = \overline{c}_0 + \sum_{i=1}^{k} \overline{c}_i \sigma(\overline{a}_i u + \overline{b}_i) \tag{1.13}$$

mit $\overline{a}_i, \overline{b}_i, \overline{c}_i \in \mathbb{R}$ und

$$\sup_{u\in[-M,M]} |\overline{C}_{M,\varepsilon}(u) - \cos(u)| < \varepsilon \tag{1.14}$$

Wieder mit der Argumentation aus Schritt 1 ist der Beweis damit beinahe abgeschlossen. $u \in [-M,M]$ muss lediglich noch auf den allgemeineren Fall $\mathbf{x} \in K \subseteq \mathbb{R}^d$ ausgeweitet werden.

Schritt 5: Sei $g(\mathbf{x}) = \sum_{i=1}^{k} \tilde{c}_i \cos(\tilde{\mathbf{a}}_i^T \mathbf{x} + b_i)$ ein Netz aus $\mathcal{F}$. Es soll gezeigt werden, dass für beliebige Sigmoidfunktion σ, beliebige kompakte Menge $K \subseteq \mathbb{R}^d$ und $\varepsilon > 0$ ein neuronales Netz $s(\mathbf{x}) = \sum_{i=1}^{k} c_i \sigma(\mathbf{a}_i^T \mathbf{x} + b_i)$ existiert, so dass

$$\sup_{\mathbf{x}\in K} |s(\mathbf{x}) - g(\mathbf{x})| < \varepsilon$$

Da K kompakt und die Funktionen $\mathbf{a}_i^T \mathbf{x} + b_i$ stetig sind, gibt es ein $M > 0$, so dass $\forall i \in \{1,...,k\} : \sup_{\mathbf{x}\in K} |\mathbf{a}_i^T \mathbf{x} + b_i| \leq M$. Sei $c = \sum_{i=1}^{k} |\tilde{c}_i|$. Setzt man nun

$u_i = (\tilde{\mathbf{a}}_i^T \mathbf{x} + \tilde{b}_i)$, so erhält man aus (1.14)

$$\sup_{\mathbf{x}\in K} |\sum_{i=1}^{k} \tilde{c}_i \cos(u_i) - \sum_{i=1}^{k} \tilde{c}_i \overline{C}_{M,\varepsilon/c}(u_i)|$$
$$\leq \sum_{i=1}^{k} |\tilde{c}_i| \sup_{u\in[-M,M]} |\cos(u) - \overline{C}_{M,\varepsilon/c}(u)| < c\frac{\varepsilon}{c} = \varepsilon$$

was für jedes Netz in $\mathcal{F}$ funktioniert, womit der Beweis mit Blick auf Schritt 1 und den Satz von STONE-WEIERSTRASS abgeschlossen ist. □

1.4 Finden geeigneter Gewichte – Rückpropagierung

Im Folgenden wird davon ausgegangen, dass die Architektur des zur Regressionsschätzung benutzten Netzes bereits vorliegt, dass also s, die zugehörigen l_j und $\sigma(\cdot)$ gegeben sind.

Die im fünften Kapitel beschriebene konkrete Anwendung neuronaler Netze verwendet die entsprechenden Routinen aus dem R-Paket `nnet`, die eine verdeckte Schicht vorgeben und per Default die logistische Sigmoidfunktion verwenden. Die Neuronenzahl wird mit Hilfe einer Kreuzvalidierung festgelegt werden. Siehe hierzu auch Abschnitt 4.2.1.

Sei (X, Y) ein $\mathbb{R}^d \times \mathbb{R}$-wertiger Zufallsvektor mit $\mathbf{E}\{Y^2\} < \infty$. Sei ferner $\mathcal{D}_n := \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ mit $(X, Y), (X_1, Y_1), \ldots, (X_n, Y_n)$ unabhängig identisch verteilt.

Nicht als festgelegt angenommen ist der Gewichtevektor $\mathbf{w}$. Optimal wäre, ihn so zu wählen, dass das L_2-Risiko $R(f_{\mathbf{w}})$ minimiert wird. Dass also

$$\mathbf{w} = \arg\min_{\widehat{\mathbf{w}}} R(f_{\widehat{\mathbf{w}}}) := \arg\min_{\widehat{\mathbf{w}}} \mathbf{E}\{|f_{\widehat{\mathbf{w}}}(X) - Y|^2\}.$$

Die Schwierigkeit hierbei ist, dass eine exakte Berechnung von $R(f_{\mathbf{w}})$ die Kenntnis der unbekannten Verteilung von (X, Y) voraussetzt.

Statt dessen weicht man auf das aus einer Stichprobe zu gewinnende empirische L_2-Risiko R_n aus und bestimmt

$$\mathbf{w} := \arg\min_{\widehat{\mathbf{w}}} R_n(f_{\widehat{\mathbf{w}}}) := \arg\min_{\widehat{\mathbf{w}}} \frac{1}{n} \sum_{i=1}^{n} |f_{\widehat{\mathbf{w}}}(X_i) - Y_i|^2.$$

Eine Methode dieses $\mathbf{w}$ zu suchen stellt die sogenannte *Rückpropagierung* dar, die Gegenstand dieses Abschnittes ist. Ihr Ansatz ist ein Gradientenabstieg, bei dem ein zufälliges $\mathbf{w}_0$ initialisiert und folgende Rekursion gestartet wird:

$$\mathbf{w}_{t+1} := \mathbf{w}_t - \gamma_t \nabla_{\mathbf{w}} R_n(f_{\mathbf{w}_t}),$$

mit $(\gamma_t) > 0$ als Schrittweitenfolge und $\nabla_{\mathbf{w}} = (\frac{\partial}{\partial w_{0,1}^{(0)}}, \ldots, \frac{\partial}{\partial w_{l_s,1}^{(s)}})$.

Das Problem ist nun die Bestimmung von $\nabla_{\mathbf{w}} R_n(f_{\mathbf{w}})$ für gegebenes $\mathbf{w}$.

Sei $\frac{\partial}{\partial \mathbf{w}_{i_0,j_0}^{(s_0-1)}} R_n(f_{\mathbf{w}})$ ein Eintrag des gesuchten Gradienten. Dann ist

$$\begin{aligned} \frac{\partial}{\partial \mathbf{w}_{i_0,j_0}^{(s_0-1)}} R_n(f_{\mathbf{w}}) &= \frac{\partial}{\partial \mathbf{w}_{i_0,j_0}^{(s_0-1)}} \left(\frac{1}{n} \sum_{k=1}^{n} |f_{\mathbf{w}}(X_k) - Y_k|^2 \right) \\ &= \frac{2}{n} \sum_{k=1}^{n} |f_{\mathbf{w}} - Y_k| \frac{\partial}{\partial \mathbf{w}_{i_0,j_0}^{(s_0-1)}} f_{\mathbf{w}}(X_k). \end{aligned} \tag{1.15}$$

Was nun $\frac{\partial}{\partial \mathbf{w}_{i_0,j_0}^{(s_0-1)}} f_{\mathbf{w}}(X_k)$ betrifft, so bezeichnet $w_{i_0,j_0}^{(s_0-1)}$ ein Gewicht, von welchem ausschließlich die Eingabe $a_{j_0}^{(s_0)}$ für das j_0-te Neuron der s_0-ten verdeckten Schicht direkt abhängig ist. Die Kettenregel erlaubt also

$$\frac{\partial}{\partial \mathbf{w}_{i_0,j_0}^{(s_0-1)}} f_{\mathbf{w}}(X_k) = \frac{\partial f_{\mathbf{w}}(X_k)}{\partial a_{j_0}^{(s_0)}} \cdot \frac{\partial a_{j_0}^{(s_0)}}{\partial \mathbf{w}_{i_0,j_0}^{(s_0-1)}} = \delta_{j_0}^{(s_0)} \cdot \frac{\partial a_{j_0}^{(s_0)}}{\partial \mathbf{w}_{i_0,j_0}^{(s_0-1)}}, \tag{1.16}$$

mit $\delta_{j_0}^{(s_0)} := \frac{\partial f_{\mathbf{w}}(X_k)}{\partial a_{j_0}^{(s_0)}}$.

Der Faktor $\frac{\partial a_{j_0}^{(s_0)}}{\partial \mathbf{w}_{i_0,j_0}^{(s_0-1)}}$ aus Gleichung (1.16) ist rasch bestimmt:

$$\begin{aligned} \frac{\partial a_{j_0}^{(s_0)}}{\partial \mathbf{w}_{i_0,j_0}^{(s_0-1)}} &= \frac{\partial}{\partial \mathbf{w}_{i_0,j_0}^{(s_0-1)}} \left[w_{0,j_0}^{(s_0-1)} + \sum_{i=1}^{l_{s_0-1}} w_{i,j_0}^{(s_0-1)} \sigma(a_i^{(s_0-1)}) \right] \\ &= \begin{cases} \sigma(a_{i_0}^{(s_0-1)}) & \text{für} \quad i_0 \geq 1 \\ 1 & \text{für} \quad i_0 = 0 \end{cases} \\ &=: o_{i_0}^{(s_0-1)} \end{aligned} \tag{1.17}$$

$\delta_{j_0}^{(s_0)}$ gelingt auf dem Umweg einer Rekursion. Sofort lösbar ist

$$\delta_1^{(s+1)} = \frac{\partial f_{\mathbf{w}}(X_k)}{\partial a_1^{(s+1)}} = \frac{\partial a_1^{(s+1)}}{\partial a_1^{(s+1)}} = 1 \tag{1.18}$$

Ansonsten wird $\delta_{j_0}^{(s_0)}$ auf eine Summe mit Größen $\delta_{\cdot}^{(s_0+1)}$ und bekannten Konstanten zurückgeführt.

Hierzu eine Vorüberlegung: Für $\delta_{j_0}^{(s_0)}$ wird $f_{\mathbf{w}}$ als Funktion von $a_{j_0}^{(s_0)}$ aufgefasst. Direkt von $a_{j_0}^{(s_0)}$ abhängig sind nur die Eingänge $a_i^{(s_0+1)}$ in die Sigmoidfunktionen der Neuronen der (s_0+1)-ten verdeckten Schicht. Man kann also schreiben

$$f_{\mathbf{w}}(a_{j_0}^{(s_0)}) = f_{\mathbf{w}}\left(a_1^{(s_0+1)}(a_{j_0}^{(s_0)}), \ldots, a_{l_{s_0+1}}^{(s_0+1)}(a_{j_0}^{(s_0)})\right)$$

und erhält folglich über die Kettenregel

$$\delta_{j_0}^{(s_0)} = \frac{\partial f_{\mathbf{w}}}{\partial a_{j_0}^{(s_0)}} = \sum_{i=1}^{l_{s_0+1}} \frac{\partial f_{\mathbf{w}}}{\partial a_i^{(s_0+1)}} \cdot \frac{\partial a_i^{(s_0+1)}}{\partial a_{j_0}^{(s_0)}} = \sum_{i=1}^{l_{s_0+1}} \delta_i^{(s_0+1)} \cdot \frac{\partial a_i^{(s_0+1)}}{\partial a_{j_0}^{(s_0)}} \tag{1.19}$$

So wurde das $\delta_{j_0}^{(s_0)}$ durch einige $\delta_{\cdot}^{(s_0+1)}$ abgelöst und die Ausdrücke $\frac{\partial a_i^{(s_0+1)}}{\partial a_{j_0}^{(s_0)}}$ lassen sich wie folgt bestimmen:

$$\frac{\partial a_i^{(s_0+1)}}{\partial a_{j_0}^{(s_0)}} = \frac{\partial}{\partial a_{j_0}^{(s_0)}} \left[w_{0,i}^{(s_0)} + \sum_{j=0}^{l_{s_0}} w_{j,i}^{(s_0)} \sigma(a_j^{(s_0)}) \right] = w_{j,i}^{(s_0)} \sigma'(a_j^{(s_0)}) \tag{1.20}$$

Aus den Gleichungen (1.15) bis (1.20) läßt sich nun $\frac{\partial}{\partial \mathbf{w}_{i_0,j_0}^{(s_0-1)}} R_n(f_{\mathbf{w}})$ berechnen:

$$\begin{aligned} \frac{\partial}{\partial \mathbf{w}_{i_0,j_0}^{(s_0-1)}} R_n(f_{\mathbf{w}}) &= \frac{2}{n} \sum_{k=1}^{n} |f_{\mathbf{w}} - Y_k| \frac{\partial}{\partial \mathbf{w}_{i_0,j_0}^{(s_0-1)}} f_{\mathbf{w}}(X_k) \\ &= \frac{2}{n} \sum_{k=1}^{n} |f_{\mathbf{w}} - Y_k| o_{i_0}^{(s_0-1)} \delta_{j_0}^{s_0}, \end{aligned}$$

wobei von $\delta_{j_0}^{s_0}$ rekursiv durch wiederholtes Anwenden der Gleichung (1.19) auf $\delta_1^{(s+1)} = 1$ „zurückgeschritten“ werden muss. Daher auch der Name des Verfahrens: „Rückpropagierung“.

Anmerkungen:

- Ausgehend von einem zufälligen $\mathbf{w}_0$ findet der Gradientenabstieg im Allgemeinen leider nur ein loakles Minimum von $R_n(f_{\mathbf{w}})$. Um diese Situation etwas zu entschärfen wird oft eine Vielzahl von Netzen mit unterschiedlichen Startpunkten trainiert und dann daraus das Bestgeeignete gewählt.

- Eigentlich sollte anstelle von $R_n(f_{\mathbf{w}})$ das L_2-Risiko $R(f_{\mathbf{w}})$ über $\mathbf{w}$ minimiert werden. Dabei ergibt sich das Problem, dass $R_n(f_{\mathbf{w}})$ im Allgemeinen nicht zusammen mit $R(f_{\mathbf{w}})$ minimal wird, sondern eher für Funktionen $f_{\mathbf{w}}$, die einfach die gegebenen Datenpunkte miteinander verbinden. Nach Satz 1.2 existieren beispielsweise für das $R_n(p) = 0$ erreichende Lagrange-Polynom (für $X_i \in \mathcal{D}_n$ paarweise verschieden)

$$p(\mathbf{x}) = \sum_{j=1}^{n} Y_j \prod_{\substack{i=1 \\ i \neq j}}^{n} \frac{|\mathbf{x} - X_i|}{|X_j - X_i|}$$

  neuronale Netze, die dieses beliebig gut gleichmäßig approximieren. Bei n Datenpunkten hat man dann eine Approximation an ein Polynom n-ten Grades, welches sich, davon abgesehen, dass es die Datenpunkte trifft, chaotisch verhalten dürfte und keine gute Schätzung für die Regressionsfunktion darstellt. Dieses Phänomen wird als *Overfitting* bezeichnet.

  Einen Ausweg aus diesem Dilemma bietet beispielsweise die Unterteilung der Stichprobe $\mathcal{D}_n$ in Trainings- und Testdatensatz $\mathcal{D}_{n_1}^{\text{Train}}$ und $\mathcal{D}_{n_2}^{\text{Test}}$. In diesem Fall kann man auf $\mathcal{D}_{n_1}^{\text{Train}}$ eine Rückpropagierung durchführen, gleichzeitig aber bei jedem Iterationsschritt überprüfen, ob sich auch das empirische L_2-Risiko auf den Testdaten merklich verkleinert hat. Falls nicht, wird die Iteration vorzeitig abgebrochen, um Overfitting zu vermeiden.

  Eine alternative oder ergänzende Vorgehensweise wäre die künstliche Einführung einer $\mathbb{R}_0^+$-wertigen Straf-Funktion $g(K, \mathbf{w})$, die für große K oder zu große Einträge von $\mathbf{w}$ hohe Werte annimmt. Beim Training eines neuronalen Netzes sucht man dann

$$f_{\mathbf{w}^*} = \arg\min_{f_{\mathbf{w}}} R_n(f_{\mathbf{w}}) + g(f_{\mathbf{w}})$$

2 Projection Pursuits

2.1 Grundidee und Motivation

Im Jahre 1981 veröffentlichten J. FRIEDMAN und W. STUETZLE in ihrer Arbeit [JHF81] die Beschreibung eines neuen Verfahrens nichtparametrischer Regression, den Projection Pursuit.

Ein Projection Pursuit Schätzer m_n ist eine Summe aus M univariaten Einzelschätzern g_ϑ mit Parameter ϑ, die sich für ihren Beitrag an die Vorhersage nach jeweils nur einer Projektionsrichtung $\mathbf{a}$ des Prädiktors $\mathbf{x}$ richten. Welcher Art diese g_ϑ sind ist prinzipiell beliebig, es wird sich aber noch herausstellen, dass im Zweifelsfall differenzierbare Funktionen zu bevorzugen sind. FRIEDMAN, STUETZLE und im übrigen auch R greifen auf *Super Smoother Splines* zurück.

m_n hat somit die Parameter $M \in \mathbb{N}, g_{\vartheta_1}, \ldots, g_{\vartheta_M} : \mathbb{R} \to \mathbb{R}, \mathbf{a}_1, \ldots, \mathbf{a}_M \in \mathbb{R}^d$ und die den im vorigen Kapitel beschriebenen neuronalen Netzen nicht unähnliche Gestalt

$$m_n(\mathbf{x}) = \sum_{j=1}^{M} g_{\vartheta_j}(\mathbf{a}_j^T \mathbf{x}), \tag{2.1}$$

welche gegenüber den klassischen, auf lokaler Mittelung basierenden, nichtparametrischen Verfahren zwei sofort einsehbare Vorteile aufweist:

1. Da sie sich auf sorgfältig ausgewählte Projektionsrichtungen im Prädiktorraum bezieht, ist die fertige Schätzfunktion leichter interpretierbar als viele andere Schätzverfahren, die eine komplizierter strukturierte Abbildung des $\mathbb{R}^d$ auf $\mathbb{R}$ heranziehen.

2. Auch wenn M unter Umständen sehr gross wird, hat der Schätzer die Gestalt einer Summe univariater Funktionen, auf denen der *Fluch der hohen Dimension* nicht lastet. Unter diesem „Fluch“ versteht man das Phänomen der schlechten Abdeckbarkeit hochdimensionaler Prädiktorräume durch Stichproben von zumeist eher beschränktem Umfang.

Diese Vorteile haben ihren Preis: Mit der in Gleichung (2.1) verwendeten Bauart des Schätzers geht die Projection Pursuit Methode davon aus, dass die

unbekannte wahre Regressionsfunktion m eine Darstellung

$$m(\mathbf{x}) = \sum_{j=1}^{M} f_j^*(\mathbf{a}_j^{*T}\mathbf{x}) \tag{2.2}$$

aus M univariaten Funktionen $f_j^* : \mathbb{R} \to \mathbb{R}$ und Projektionsrichtungen $\mathbf{a}_j^*$ besitzt.

Diese Einschränkung an die Allgemeinheit von m wird durch den nun folgenden Satz hinnehmbar. Siehe hierzu auch [Koh00].

Satz 2.1
Sei K eine kompakte Teilmenge des $\mathbb{R}^d$ und sei $m : K \to \mathbb{R}$ stetig. Dann gibt es für alle $\varepsilon > 0$ geeignete $M \in \mathbb{N}$, $f_1, \ldots, f_M \in \mathbb{R} \to \mathbb{R}$, $\mathbf{a}_1, \ldots, \mathbf{a}_M \in \mathbb{R}^d$, so dass

$$\sup_{\mathbf{x}\in K} |m(\mathbf{x}) - \sum_{j=1}^{M} f_j(\mathbf{a}_j^T\mathbf{x})| < \varepsilon$$

Beweis: Der Satz folgt mit Hilfsmitteln des bereits bewiesenen Satzes 1.2:

Sei $\sum_{j=1}^{M} f_j(\mathbf{a}_j^T\mathbf{x})$ noch weiter eingeschränkt auf die Funktionenklasse

$$\mathcal{F} = \left\{ \sum_{j=1}^{M} c_j \cos(\mathbf{a}_j^T\mathbf{x} + b_j) | M \in \mathbb{N}, \mathbf{a}_1, \ldots, \mathbf{a}_M \in \mathbb{R}^d, b_1, c_1, \ldots, b_M, c_M \in \mathbb{R} \right\}.$$

Mit Blick auf *Schritt 1* des Beweises zu Satz 1.2, sowie den Satz von Stone-Weierstrass folgt, dass $\mathcal{F}$ dicht liegt in $C_K(\mathbb{R})$. Damit ist der Beweis abgeschlossen, denn es gilt:

$$\forall\ \varepsilon > 0\ \forall\ m \in C_K(\mathbb{R})\ \exists\ f \in \mathcal{F} : \sup_{\mathbf{x}\in K} |m(\mathbf{x}) - f(\mathbf{x})| < \varepsilon.$$

□

2.2 Finden geeigneter Projection Pursuit Schätzer - Backfitting Algorithmus

Seien im Folgenden $(X, Y), (X_1, Y_1), \ldots, (X_n, Y_n)$ unabhängig identisch verteilte $\mathbb{R}^d \times \mathbb{R}$-wertige Zufallsvektoren und ferner $\mathcal{D}_n := \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$.

In diesem Abschnitt wird der sogenannte Backfitting-Algorithmus zur Findung eines geeigneten Schätzers m_n bei gegebener Stichprobe $\mathcal{D}_n$ und Ridge-Funktionenzahl M vorgestellt. M lässt sich beispielsweise mit Hilfe einer Kreuzvalidierung ermitteln, auf die an späterer Stelle noch eingegangen wird.

Ziel des Algorithmus ist, die Paare $(\mathbf{a}_1, g_{\vartheta_1}), \ldots, (\mathbf{a}_M, g_{\vartheta_M})$ so zu wählen, dass dabei ein Schätzer mit möglichst geringem L_2-Risiko herauskommt. Da nicht alle Parameter simultan eingestellt werden können, geschieht dies sukzessiv.

Zunächst der Algorithmus in der Übersicht. Auf die komplizierteren Einzelschritte wird im Anschluss eingegangen.

1. Festlegung der gewünschten Anzahl M der Ridge-Funktionen.
2. Auswahl eines univariaten Schätzverfahrens g_ϑ mit Parameter ϑ.
3. Zufällige Partitionierung der Stichprobe $\mathcal{D}_n$ in einen Trainingsdatensatz $\mathcal{D}_{n_1}^{\text{Train}}$ und einen Testdatensatz $\mathcal{D}_{n_2}^{\text{Test}}$.
4. Initialisierung der $g_{\vartheta_1} := 0, \ldots, g_{\vartheta_M} := 0$, sowie der $\mathbf{a_1}, \ldots, \mathbf{a_M}$ zufällig in $\mathbb{R}^d \backslash \{\mathbf{0}\}$.
5. Sei $k := 0$ initialisiert als Zähler für die Ridge-Funktionen.
6. Sei $k := k + 1$.
7. Neueinstellung von g_{ϑ_k} in Abhängigkeit der aktuellen
$$(\mathcal{D}_{n_1}^{\text{Train}}, \mathbf{a}_1, \ldots, \mathbf{a}_M, g_{\vartheta_1}, \ldots, g_{\vartheta_{k-1}}, g_{\vartheta_{k+1}}, \ldots, g_{\vartheta_M}).$$
8. Neueinstellung von $\mathbf{a}_k$ in Abhängigkeit der aktuellen
$$(\mathcal{D}_{n_1}^{\text{Train}}, \mathbf{a}_1, \ldots, \mathbf{a}_{k-1}, \mathbf{a}_{k+1}, \ldots, \mathbf{a}_M, g_{\vartheta_1}, \ldots, g_{\vartheta_M}).$$
9. Falls $k < M$ zurück zu Schritt 6.
10. Test, ob ein von $\mathcal{D}_{n_2}^{\text{Test}}$ abhängiges Abbruchkriterium eingetreten ist. Falls nicht, zurück zu Schritt 5.
11. Zurückliefern des fertigen Schätzers.
$$m_n := \sum_{j=1}^{M} g_{\vartheta_j}(\mathbf{a}_j^T \mathbf{x}).$$

Erläuterungen:

Schritt 1: Die Anzahl M der Ridge-Funktionen wird an dieser Stelle als gegeben angenommen. Sie kann beispielsweise, wie im Abschnitt 4.2.1 beschrieben, anhand einer Kreuzvalidierung festgelegt werden.

Schritt 2: g_ϑ sollte einer Klasse von differenzierbaren, konsistenten, univariaten Schätzern mit Parameter ϑ entstammen die geeignet sind, die f_j in Gleichung (2.2) zu approximieren.

Schritt 3: Wie bei den neuronalen Netzen kann auch hier nur $R_n(m_n, \mathcal{D}_{n_1}^{\text{Train}})$ anstelle des eigentlich gesuchten L_2-Risikos $R(m_n)$ bestimmt werden. Dem Overfitting-Problem wird begegnet, indem die Stichprobe in Trainingsdatensatz $\mathcal{D}_{n_1}^{\text{Train}}$ und Testdatensatz $\mathcal{D}_{n_2}^{\text{Test}}$ unterteilt und m_n auf dem Trainingsdatensatz nur so lange verfeinert wird, wie auch eine ausreichende Verkleinerung des Fehlers bei der Vorhersage der Testdaten vorliegt.

Schritt 7:[1] g_{ϑ_k} soll verfeinert werden. Dazu betrachte man

$$Y = m(X) + \varepsilon \text{ mit } \varepsilon := Y - m(X). \tag{2.3}$$

Dann ist

$$\mathbf{E}\{\varepsilon \,|X\} = \mathbf{E}\{Y - m(X) \,|X\} = \mathbf{E}\{Y|X\} - m(X) = 0.$$

Es wird angenommen, dass der zufällige Fehler ε und X voneinander unabhängige Zufallsvariablen sind und gilt, dass

$$\mathbf{E}\{\varepsilon \,|X\} = \mathbf{E}\{\varepsilon\} = 0.$$

Weiterhin wird wie angekündigt davon ausgegangen, dass m wie in Gleichung (2.2) darstellbar ist.

Mit diesen Annahmen und Gleichung (2.3) gelangt man zu

$$\begin{aligned} Y &= m(X) + \varepsilon \\ &= \sum_{j=1}^{M} f_j^*(\mathbf{a}_j^{*T} X) + \varepsilon \\ \Rightarrow Y - \sum_{\substack{j=1 \\ j \neq k}}^{M} f_j^*(\mathbf{a}_j^{*T} X) &= f_k^*(\mathbf{a}_k^{*T} X) + \varepsilon \end{aligned}$$

[1] Vergleiche auch [Koh00].

Seien nun $\overline{X} := \mathbf{a}_k^{*T} X$ und $\overline{Y} := Y - \sum_{j=1|j\neq k}^{M} f_j^*(\mathbf{a}_j^{*T} X)$, wobei $\overline{X}, \overline{Y} \in \mathbb{R}$.

$$\begin{aligned} \Rightarrow \overline{Y} &= f_k^*(\overline{X}) + \varepsilon \\ \Rightarrow \mathbf{E}\{\overline{Y}|\overline{X}\} &= \mathbf{E}\{f_k^*(\overline{X})|\overline{X}\} + \mathbf{E}\{\varepsilon|\overline{X}\} \\ &= f_k^*(\overline{X}) + \mathbf{E}\{\varepsilon\} \\ &= f_k^*(\overline{X}). \end{aligned}$$

f_k^* ist also die Regressionsfunktion von $(\overline{X}, \overline{Y})$, für deren Approximation mit dem univariaten Schätzverfahren g_ϑ die Stichprobe $\overline{\mathcal{D}}_n = \{(\overline{X}_1, \overline{Y}_1), \ldots, (\overline{X}_n, \overline{Y}_n)\}$, respektive deren Partitionierung in $\overline{\mathcal{D}}_{n_1}^{\text{Train}}$ und $\overline{\mathcal{D}}_{n_2}^{\text{Test}}$ vorliegt.

Schritt 8: Nun soll $\mathbf{a}_k$ angepasst werden. Dies geschieht mit dem *Gauß-Newton-Verfahren*[2]:

Sei $\widehat{\mathbf{a}}_k$ der gesuchte, verbesserte Wert. Dann ist für nicht zu großes $(\widehat{\mathbf{a}}_k - \mathbf{a}_k)$

$$\begin{aligned} m_n = m_n(\widehat{\mathbf{a}}_k) &= g_{\vartheta_k}(\widehat{\mathbf{a}}_k^T \mathbf{x}) + \sum_{\substack{j=1 \\ j\neq k}}^{M} g_{\vartheta_j}(\mathbf{a}_j^T \mathbf{x}) \\ &\approx g_{\vartheta_k}(\mathbf{a}_k^T \mathbf{x}) + (\widehat{\mathbf{a}}_k^T \mathbf{x} - \mathbf{a}_k^T \mathbf{x}) \frac{\partial}{\partial(\mathbf{a}_k^T \mathbf{x})} g_{\vartheta_k}(\mathbf{a}_k^T \mathbf{x}) + \sum_{\substack{j=1 \\ j\neq k}}^{M} g_{\vartheta_j}(\mathbf{a}_j^T \mathbf{x}) \\ &=: g_{\vartheta_k}(\mathbf{a}_k^T \mathbf{x}) + (\widehat{\mathbf{a}}_k^T \mathbf{x} - \mathbf{a}_k^T \mathbf{x}) g'_{\vartheta_k}(\mathbf{a}_k^T \mathbf{x}) + \alpha, \end{aligned}$$

mit $\alpha = \text{const}$ über $\widehat{\mathbf{a}}_k$. Damit ergibt sich für das empirische L_2-Risiko auf den Trainingsdaten $\mathcal{D}_{n_1}^{\text{Train}}$

$$\begin{aligned} R_n(m_n, \mathcal{D}_{n_1}^{\text{Train}}) &= \frac{1}{n_1} \sum_{i=1}^{n_1} (Y_i - m_n(\widehat{\mathbf{a}}_k^T, X_i))^2 \\ &\approx \frac{1}{n_1} \sum_{i=1}^{n_1} (Y_i - g_{\vartheta_k}(\mathbf{a}_k^T X_i) - (\widehat{\mathbf{a}}_k^T X_i - \mathbf{a}_k^T X_i) g'_{\vartheta_k}(\mathbf{a}_k^T X_i) - \alpha)^2 \\ &= \frac{1}{n_1} \sum_{i=1}^{n_1} (g'_{\vartheta_k}(\mathbf{a}_k^T X_i))^2 \cdot \left[\left(\frac{Y_i - g_{\vartheta_k}(\mathbf{a}_k^T X_i) - \alpha}{g'_{\vartheta_k}(\mathbf{a}_k^T X_i)} + \mathbf{a}_k^T X_i \right) - \widehat{\mathbf{a}}_k^T X_i \right]^2 \\ &=: \frac{1}{n_1} \sum_{i=1}^{n_1} \beta_i^2 [\gamma_i - \widehat{\mathbf{a}}_k^T X_i]^2 =: \text{RSS}(\widehat{\mathbf{a}}_k). \end{aligned}$$

[2] Quelle: [TH01] unter Projection Pursuit Regression

Nun hat man ein Kleinste Quadrate Schätzproblem zu den γ_i mit Gewichten β_i, wobei die β_i, γ_i konstant sind über $\widehat{\mathbf{a}}_k$.

Ziel ist nun diese *Residual Sum of Squares* über $\widehat{\mathbf{a}}_k$ zu minimieren.

Seien hierzu

$$\mathbf{v} := \begin{pmatrix} \beta_1\gamma_1 \\ \vdots \\ \beta_{n_1}\gamma_{n_1} \end{pmatrix} \text{ und } \mathbf{X}^* := \begin{pmatrix} \beta_1 X_1^{(1)} & \cdots & \beta_1 X_1^{(d)} \\ \vdots & & \vdots \\ \beta_{n_1} X_{n_1}^{(1)} & \cdots & \beta_{n_1} X_{n_1}^{(d)} \end{pmatrix}$$

Dann lässt sich obige RSS umschreiben zu

$$\mathrm{RSS}(\widehat{\mathbf{a}}_k) = \frac{1}{n_1}(\mathbf{v} - \mathbf{X}^*\widehat{\mathbf{a}})^T(\mathbf{v} - \mathbf{X}^*\widehat{\mathbf{a}}).$$

Nullsetzen von $\frac{\partial}{\partial \widehat{\mathbf{a}}_k}\mathrm{RSS}(\widehat{\mathbf{a}}_k)$ ergibt das d-zeilige Normalengleichungssystem

$$\begin{aligned} 0 &= \mathbf{X}^{*T}(\mathbf{v} - \mathbf{X}^*\widehat{\mathbf{a}}_k) \\ \Rightarrow \mathbf{X}^{*T}\mathbf{X}^*\widehat{\mathbf{a}}_k &= \mathbf{X}^{*T}\mathbf{v}, \end{aligned}$$

welches sich für $(\mathbf{X}^{*T}\mathbf{X}^*)$ invertierbar sogar eindeutig, ansonsten mit einem beliebigen Lösungsverfahren für lineare Gleichungssysteme lösen lässt. Die die RSS minimierende Lösung wird als neuer Wert für die k-te Projektionsrichtung des Pursuits verwendet.

Schritt 10: Wenn sich der Fehler auf den Testdaten gegenüber der vorherigen Iteration noch wesentlich verbessert hat, zurück zu Schritt 5. Sonst Abbruch und eventuell Verwerfen des letzten Durchlaufs.

3 Classification and Regression Trees

3.1 Historische Entwicklung

Ein weiteres Verfahren der nichtparametrischen Regressionsschätzung, welches ohne die Verfügbarkeit moderner Computer undenkbar wäre, ist die Anwendung sogenannter Regressionsbäume.

Bäume sind eine Klasse von stückweise konstanten Schätzfunktionen, die den Prädiktorraum nach einem rekursiven Schema in T zu den Koordinatenachsen parallele Quader $Q_1, \ldots, Q_T$ partitionieren, für jedes dieser Q_j mit Hilfe der Teilstichprobe $\mathcal{D}_{n_j} := \{(X, Y) \in \mathcal{D}_n | X \in Q_j\}$ einen konstanten Schätzer c_j für $\mathbf{E}\{Y|X \in Q_j\}$ bestimmen und schließlich $m_n(\mathbf{x})$ nach der Vorschrift

$$m_n(\mathbf{x}) = \sum_{j=1}^{T} \mathbf{I}_{\{\mathbf{x} \in Q_j\}} c_j(\mathcal{D}_{n_j})$$

ermitteln.

Die ersten Schätzer dieser Art wurden in den frühen sechziger Jahren von Morgan und Sonquist am Institut für Sozialwissenschaften der University of Michigan entwickelt.

Zehn Jahre danach schrieben Messenger und Morgan ein Programm, welches Bäume auf Klassifikationsprobleme anwandte.

Beide Grundkonzepte wurden später von den U.S.-amerikanischen Wissenschaftlern Breiman, Friedman, Olshen und Stone aufgegriffen, weiterentwickelt und schließlich in das Programmpaket *Classification and Regression Trees*, kurz *CART*, umgesetzt. R stellt die darin verwendeten Methoden in dem Programmpaket `tree` zur Verfügung.

Gegenstand dieses Kapitels ist die Funktionsweise von Regressionsbäumen und deren Training.

3.2 Aufbau von Regressionsbäumen

Definition 3.1 *Sei* $\sim$ *eine Relation auf* $\mathbb{R}$. *Eine Relation* $\sim^*$ *heißt* zu $\sim$ komplementäre Relation *falls*

$$\forall\ a, b \in \mathbb{R}: \quad (a \sim b) \Leftrightarrow \neg(a \sim^* b)$$

Anmerkung: Die zu einer Relation $\sim$ komplementäre Relation wird auch im Folgenden stets mit $\sim^*$ bezeichnet.

Definition 3.2 *Sei $\mathcal{A}$ eine Partitionierung des $\mathbb{R}^d$ und $B \subseteq \mathbb{R}^d$, $j \in \{1, \ldots, d\}$, $s \in \mathbb{R}$. Sei $\sim$ eine der Relationen $<, \leq, >, \geq$. $\tau^{\sim}_{s,j}(B)$ heißt* Trennung *in B an der Stelle s in der j-ten Koordinate und nach Relation $\sim$, falls*

$$\tau^{\sim}_{j,s}(B) = \{\mathbf{x} \in B | x_j \sim s\}.$$

$\tau^{\sim}_{s,j}(B)$ heißt gültige Trennung *in Bezug auf $\mathcal{A}$ und wird auch $\tau^{\sim}_{s,j,\mathcal{A}}(B)$ geschrieben, falls*

$$\forall\ A \in \mathcal{A}:\ \ (A \subseteq \tau^{\sim}_{j,s,\mathcal{A}}(B)) \vee (A \cap \tau^{\sim}_{j,s,\mathcal{A}}(B) = \emptyset)$$

Definition 3.3 *Eine Partition $\mathcal{Q}$ des $\mathbb{R}^d$ heißt* binär *oder* rekursiv erzeugt, *falls eine der folgenden Bedingungen erfüllt ist:*

i. $\mathcal{Q} = \{\mathbb{R}^d\}$.

ii. Es exisitiert eine binäre Partition $\mathcal{Q}^*$ des $\mathbb{R}^d$ mit $Q \in \mathcal{Q}^*$, $j \in \{1, \ldots, d\}$, $s \in \mathbb{R}$, $\sim\ \in \{<, \leq\}$, so dass sich $\mathcal{Q}$ schreiben läßt als

$$\mathcal{Q} := (\mathcal{Q}^* \backslash \{Q\})\ \dot{\cup}\ \{\tau^{\sim}_{j,s}(Q), \tau^{\sim^*}_{j,s}(Q)\}$$

Anschaulich betrachtet wird, mit Ausnahme von $\{\mathbb{R}^d\}$, jede binäre Partition des $\mathbb{R}^d$ ausgehend von $\mathcal{Q}_0 = \{\mathbb{R}^d\}$ erzeugt, indem man $\mathbb{R}^d$ mit einer geeigneten, zu einer Koordinatenrichtung orthogonalen Hyperebene in zwei Hälften spaltet und diese Hälften dann intern gegebenenfalls weiter auftrennt, bis die gewünschte Partition erreicht ist.

Sei nun $\mathcal{Q}_T$ eine binäre Partition des $\mathbb{R}^d$ mit Mächtigkeit T und $c_1, \ldots, c_T \in \mathbb{R}$. Sei schließlich die Abbildung f definiert als

$$\begin{aligned} f:\ \mathbb{R}^d &\to \mathbb{R} \\ \mathbf{x} &\mapsto \sum_{i=1}^{T} \mathbf{I}_{\{\mathbf{x} \in Q_i\}} c_i. \end{aligned} \tag{3.1}$$

Ein im Folgenden wichtig werdender Vorteil solcher f ist ihre Darstellbarkeit mit Hilfe eines Baumdiagramms. Die rechte Hälfte von Abbildung 3.1 zeigt ein Beispiel für die Darstellung einer solchen Funktion f mit $d = 2$ und einer sechselementigen binären Partition.

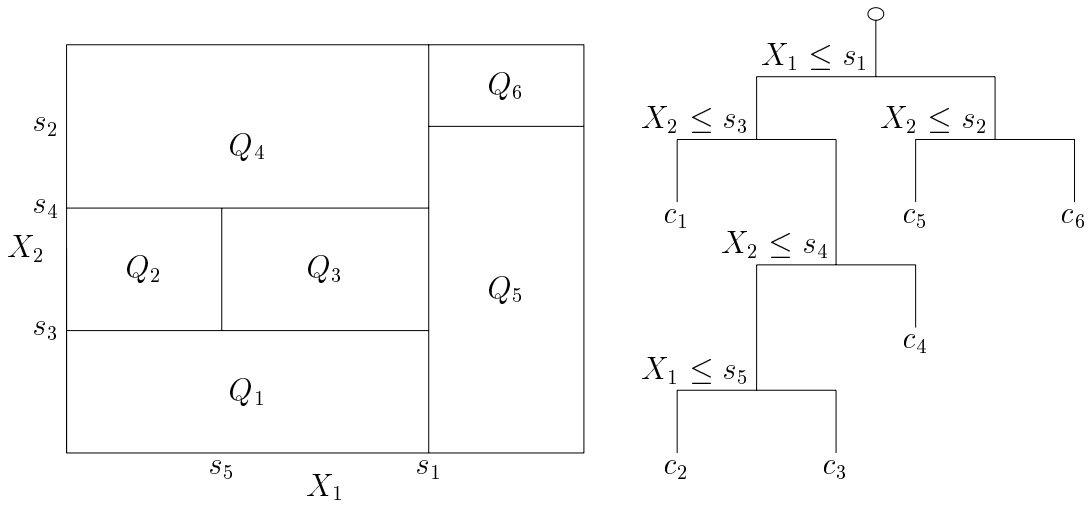

Abbildung 3.1: Partitionierung eines zweidimensionalen Prädiktorraumes

Sie bedarf allerdings noch einiger Erläuterungen:

- Jeder Knoten des Baumdiagramms ist mit einer Ungleichung verknüpft. Falls diese gilt, wird per Konvention der linke Pfad gewählt. Sonst der rechte.
- Links im Bild ist die zum Baum gehörende binäre Partition des Prädiktorraumes zu sehen. Stellt man die einzelnen Q_i durch sukzessive Anwendung gültiger Trennungen des Prädiktorraumes dar, werden die Pfade erkennbar, die im Baumdiagramm vom Startpunkt zu den einzelnen c_i führen:

$$\begin{aligned}
Q_1 &= \tau^{\leq}_{j=2,s_3,\mathcal{Q}} \circ \tau^{\leq}_{j=1,s_1,\mathcal{Q}}(\mathbb{R}^2) \\
Q_2 &= \tau^{\leq}_{j=1,s_5,\mathcal{Q}} \circ \tau^{\leq}_{j=2,s_4,\mathcal{Q}} \circ \tau^{>}_{j=2,s_3,\mathcal{Q}} \circ \tau^{\leq}_{j=1,s_1,\mathcal{Q}}(\mathbb{R}^2) \\
Q_3 &= \tau^{>}_{j=1,s_5,\mathcal{Q}} \circ \tau^{\leq}_{j=2,s_4,\mathcal{Q}} \circ \tau^{>}_{j=2,s_3,\mathcal{Q}} \circ \tau^{\leq}_{j=1,s_1,\mathcal{Q}}(\mathbb{R}^2) \\
Q_4 &= \tau^{>}_{j=2,s_4,\mathcal{Q}} \circ \tau^{>}_{j=2,s_3,\mathcal{Q}} \circ \tau^{\leq}_{j=1,s_1,\mathcal{Q}}(\mathbb{R}^2) \\
Q_5 &= \tau^{\leq}_{j=2,s_2,\mathcal{Q}} \circ \tau^{>}_{j=1,s_1,\mathcal{Q}}(\mathbb{R}^2) \\
Q_6 &= \tau^{>}_{j=2,s_2,\mathcal{Q}} \circ \tau^{>}_{j=1,s_1,\mathcal{Q}}(\mathbb{R}^2)
\end{aligned}$$

- Einige Bezeichnungen:
  Die mit Ungleichungen verknüpften Knotenpunkte werden im Folgenden als *Entscheidungsknoten*, die Endpunkte als *Endknoten* bezeichnet. Der oberste, in Abbildung 3.1 mit einem Kreis markierte, Knoten des Baumes heiße auch *Wurzel*.

Sei K ein beliebiger Knoten und Q_K die Menge, auf die der Prädiktorraum durch den die Wurzel mit K verbindenden Pfad eingeschränkt wird. Dann heißt Q_K *Bereich* des Knoten K.

Weitere Bemerkungen:

- Die Darstellung eines Baumes mit Hilfe eines Diagramms wie in Abbildung 3.1 rechts ist nicht in jedem Fall eindeutig. Abbildung 3.2 zeigt ein entsprechendes Beispiel.
- Nicht jede Partitionierung des $\mathbb{R}^d$ in Quader ist binär und durch Bäume darstellbar. Abbildung 3.3 etwa zeigt ein Beispiel einer Quaderpartitionierung, für die es keine Darstellung der einzelnen Mengen durch Ketten gültiger Trennungen gibt.

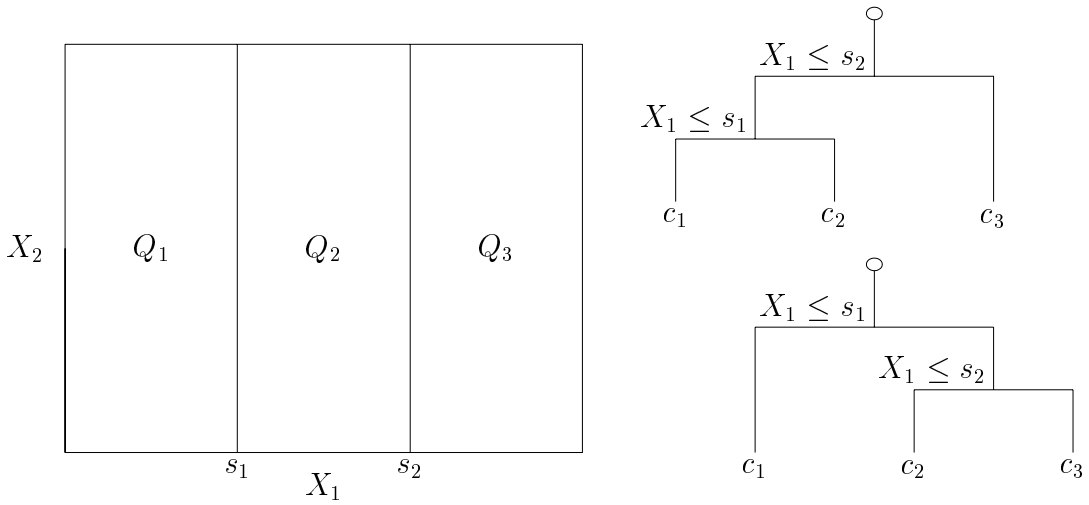

Abbildung 3.2: Eine Partition, zwei Diagramme

Definition 3.4 *Sei $\mathcal{R} \subseteq \mathbb{R}^d$ Sei $\mathcal{Q}_T = \{Q_1, \ldots, Q_T\}$ eine binäre Partition von $\mathcal{R}$ mit Mächtigkeit T. Sei f eine stückweise konstante Funktion*

$$\begin{aligned} f: \ \mathcal{R} &\to \mathbb{R} \\ \mathbf{x} &\mapsto \sum_{i=1}^{T} \mathbf{I}_{\{\mathbf{x} \in Q_i\}} c_i \end{aligned} \tag{3.2}$$

mit $c_1, \ldots, c_T \in \mathbb{R}$ und D ein Baumdiagramm, welches f beschreibt. Das Tripel $(\mathcal{Q}_T, f, D)$ heißt Baum *zu $\mathcal{Q}_T$, f und D auf $\mathcal{R}$.*

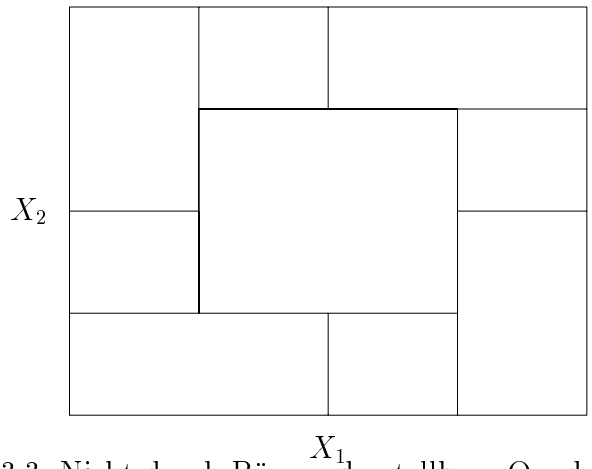

Abbildung 3.3: Nicht durch Bäume darstellbare Quaderpartitionierung

Bemerkung zu Definition 3.4: Da D sowohl f als auch $\mathcal{Q}_T$ festlegt, beinhaltet das Baumtripel eine gewisse Redundanz. Diese wird hier in Kauf genommen, um eine bessere Handhabe der Bäume zu bekommen.

3.3 Zuschnitt von Bäumen

Im weiteren Verlauf des Kapitels werden Bäume zu Regressionszwecken trainiert. Ein Vorgang, bei dem es zum Teil notwendig sein wird, komplizierte Bäume zu vereinfachen. Die dafür noch benötigten Begriffe folgen nun.

Definition 3.5 *Sei $\mathcal{T}_0 := (\mathcal{Q}_0, f_0, D_0)$ ein beliebiger Baum und K ein Knoten in D_0 mit Bereich Q_K. Sei $\mathcal{Q}_1 := \{Q \in \mathcal{Q}_0 |\ Q \subseteq Q_K\}$, f_1 die Einschränkung von f_0 auf Q_K und D_1 das Teildiagramm von D_0, welches mit K wurzelt. Dann heißt der Baum $\mathcal{T}_1 := (\mathcal{Q}_1, f_1, D_1)$ ein* Teilbaum *von $\mathcal{T}_0$. Im Folgenden wird auch die Schreibweise $\mathcal{T}_1 \subseteq \mathcal{T}_0$ verwendet.*

Definition 3.6 *Sei $\mathcal{T}_0 := (\mathcal{Q}_0, f_0, D_0)$ ein Baum mit $f_0 : \mathbb{R}^d \to \mathbb{R}$ und K ein Entscheidungsknoten in $\mathcal{T}_0$ mit Bereich Q_K.*
Sei $\mathcal{Q}_1 := \{Q \in \mathcal{Q}_0 |\ Q \cap Q_K = \emptyset\} \dot\cup \{Q_K\}$
und

$$f_1(x) := \begin{cases} f_0(x) & \text{für } x \in \mathbb{R}^d \backslash Q_K \\ c_K & \text{für } x \in Q_K \end{cases}$$

mit $c_K \in \mathbb{R}$. Sei ferner D_1 das zu f_1 und $\mathcal{Q}_1$ passende Baumdiagramm, welches aus D_0 entsteht, wenn der an K wurzelnde Teilbaum fallengelassen und an

seine Stelle ein Endknoten mit Wert c_K gesetzt wird. Der Übergang von $\mathcal{T}_0$ auf $\mathcal{T}_1$ wird im Folgenden als Ausschnitt *des Baumes $\mathcal{T}_0$ am Entscheidungsknoten K zugunsten des Wertes c_K bezeichnet.*

Definition 3.7 *Sei $\mathcal{T}_0$ ein Baum. Ein Baum $\mathcal{T}$, der durch eine Anzahl von Ausschnitten an geeigneten Entscheidungsknoten in $\mathcal{T}_0$ erzeugbar ist, heißt* Unterbaum *von $\mathcal{T}_0$. Im Folgenden wird auch die Schreibweise $\mathcal{T} \leq \mathcal{T}_0$ verwendet.*

Definition 3.8 *Seien K_1 und K_2 zwei Knoten in einem Baum mit Bereichen Q_{K_1} und Q_{K_2} so, dass $Q_{K_2} \subset Q_{K_1}$. In diesem Fall wird K_1 als* Oberknoten *von K_2 und K_2 als* Unterknoten *von K_1 bezeichnet.*

Definition 3.9 *Sei K ein Knoten in einem Baum $\mathcal{T}$. Unter der* Hierarchie *$H(K)$ von K sei die Anzahl der Oberknoten von K in $\mathcal{T}$ verstanden.*

Definition 3.10 *Sei $\mathcal{T}$ ein Baum. Unter der* Komplexität *von $\mathcal{T}$ sei die Anzahl der Endknoten in $\mathcal{T}$ verstanden und dafür die Schreibweise $|\mathcal{T}|$ eingeführt.*

Definition 3.11 *Ein Baum, der nur einen Endknoten enthält, also eine konstante Funktion beschreibt, wird als* trivialer Baum *bezeichnet.*

Zum Schluss dieses Abschnittes noch eine Schreibweisenkonvention: Wenn im Folgenden von einem Knoten K die Rede sein wird, stehen automatisch

- Q_K für den Bereich von K,
- $\mathcal{T}_K = (\mathcal{Q}_K, f_K, D_K)$ für den in K wurzelnden Teilbaum.

3.4 Training von Regressionsbäumen

Seien wieder $(X, Y), (X_1, Y_1), \ldots, (X_n, Y_n)$ unabhängig identisch $\mathbb{R}^d \times \mathbb{R}$-wertige Zufallsvektoren und $\mathcal{D}_n := \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$. Wieder ist ein Schätzer m_n für die Regressionsfunktion $m(\mathbf{x}) := \mathbf{E}\{Y|X = \mathbf{x}\}$ gesucht.

Zunächst einige weitere Schreibweisen, um die Darstellung zu vereinfachen.

Für $A \subseteq \mathbb{R}^d$ sei

$$\mathcal{D}(A) := \{(X, Y) \in \mathcal{D}_n | X \in A\} =: \{(X_1^{(A)}, Y_1^{(A)}), \ldots, (X_{n_A}^{(A)}, Y_{n_A}^{(A)})\}$$

mit $n_A = |\mathcal{D}(A)|$ und daran anknüpfend $\mathcal{D}_X(A) := \{X_1^{(A)}, \ldots, X_{n_A}^{(A)}\}$ und $\mathcal{D}_Y(A) := \{Y_1^{(A)}, \ldots, Y_{n_A}^{(A)}\}$.

Sei ferner

$$\overline{Y}(A) := \frac{1}{n_A} \sum_{i=1}^{n_A} Y_i^{(A)}.$$

Der nun vorgestellte Algorithmus lehnt sich an [TH01] an.

Um einen zur Regressionsschätzung taugenden Baum zu erhalten, müssen zunächst eine geeignete binäre Partition $\mathcal{Q}_T$ und eine passende stückweise konstante Funktion f, beides wie in Definition 3.4 beschrieben, gefunden werden.

Ein entsprechender Algorithmus muss also in der Lage sein, selbständig die Trennparameter $s_1, \ldots, s_{T-1} \in \mathbb{R}; j_1, \ldots, j_{T-1} \in \{1, \ldots, d\}; \sim_1, \ldots, \sim_{T-1}$, wie in Definition 3.2, und die Schätzwerte $c_1, \ldots, c_T$ festzulegen.

Dabei soll Overfitting ebenso vermieden werden, wie das Heranziehen zu einfach strukturierter Bäume.

Von dem naheliegenden Ansatz, die einzelnen Äste so lange immer feiner aufzutrennen bis keine hinreichend große Verbesserung des entstehenden Schätzers mehr auftritt, wird in [LB84][1] mit dem Argument abgeraten, dass sich eine zunächst schlechte Trennung bei weiterer Verfeinerung als sehr gut entpuppen kann.

Stattdessen wird dem Problem von der anderen Seite her begegnet, indem zunächst ein sehr verzweigter, overfittender Baum $\mathcal{T}_0$ erzeugt wird. Dieser Baum wird dann in geeigneter Weise „zugeschnitten“. Dabei wird auf die in [LB84] und [TH01] vorgestellte Methode des *Cost Complexity Prunings* zurückgegriffen. Dazu mehr im Abschnitt 3.4.2.

3.4.1 Teil I des Trainingsalgorithmus: Wachstum

Es folgt ein Algorithmus, dessen Ziel das Erzeugen eines Baumes $\mathcal{T}_0$ ist, der die Daten überanpasst. Zunächst ein Überblick. Die komplizierteren Schritte werden im Anschluss erläutert.

[1] Im Abschnitt „Right sized Trees and honest Estimates“

1. Wahl einer Methode zur Bestimmung eines empirischen Risikos auf $\mathcal{D}_n$. Hier wird zu diesem Zweck wieder auf das empirische L_2-Risiko zurückgegriffen:

$$R_n(f, \mathcal{D}_n) := \sum_{i=1}^{n} (Y_i - f(X_i))^2$$

2. Jetzt wird der Baum $\mathcal{T}_0 = (\mathcal{Q}_0, f_0, D_0)$ wie folgt initialisiert:

   - $\mathcal{Q}_0 := \{A\}$ mit $A := \mathbb{R}^d$.
   - $f_0(x)$ wird zu diesem Zeitpunkt noch nicht benötigt.
   - Dementsprechend besteht das Diagramm D_0 zunächst nur aus der Wurzel, die zugleich Endknoten mit unbestimmtem Wert c_A ist.

   Damit kann in die den Baum erzeugende Rekursion eingestiegen werden:

3. Sei l die Anzahl der Trennungen, die den Pfad von der Wurzel bis zu A beschreiben. Letzterer ist aus D_0 ablesbar und beschreibt A in der Form

$$A = \tau_{j_l, s_l}^{\sim_l} \circ \cdots \circ \tau_{j_1, s_1}^{\sim_1} (\mathbb{R}^d)$$

   Jetzt wird eine geeignete Verzweigung mit $j_{l+1}, s_{l+1}, \sim_{l+1}$ gesucht, um A aufzuspalten in

$$\begin{aligned} B &:= \tau_{j_{l+1}, s_{l+1}}^{\sim_{l+1}} \circ \tau_{j_l, s_l}^{\sim_l} \circ \cdots \circ \tau_{j_1, s_1}^{\sim_1} (\mathbb{R}^d) \text{ und} \\ B^* &:= \tau_{j_{l+1}, s_{l+1}}^{\sim^*_{l+1}} \circ \tau_{j_l, s_l}^{\sim_l} \circ \cdots \circ \tau_{j_1, s_1}^{\sim_1} (\mathbb{R}^d). \end{aligned}$$

   Außerdem werden Schätzwerte c_B und c_{B^*} für eine Aktualisierung von f_0 ermittelt. Dies geschieht hier mit Hilfe des arithmetischen Mittels $c_B := \overline{Y}(B)$

4. Als nächstes werden $\mathcal{Q}_0$ zu $\mathcal{Q}_0 := (\mathcal{Q}_0 \backslash \{A\}) \dot{\cup} \{B, B^*\}$ und f_0 zu

$$f_0(x) := \begin{cases} c_B & \text{für } x \in B \\ c_{B^*} & \text{für } x \in B^* \\ f_0(x) & \text{sonst} \end{cases}$$

   redefiniert. Außerdem wird D_0 um die entsprechende Verzweigung erweitert.

5. Soweit noch kein entsprechendes Abbruchkriterium eingetreten ist, werden B und/oder B^* ihrerseits ab Punkt 3 anstelle von A weiter aufgespalten, wodurch der Baum an Komplexität weiter zunimmt.

   Liegen in $\mathcal{Q}_0$ keine noch aufzuspaltenden Mengen mehr vor, gilt $\mathcal{T}_0$ als ausgewachsen und die zweite Phase des Algorithmus, das Zuschneiden, beginnt.

Erläuterungen:

Schritt 3: Da die Schnittpunkte $s_1, \ldots, s_{T-1}$ reellwertig gewählt werden, kann ohne Einschränkung der Allgemeinheit $\sim_{l+1}$ als $\leq$ und $\sim^*_{l+1}$ als $>$ gesetzt werden.

Sei für $\sim \in \{\leq, >\}$

$$\mathbf{Y}^{\sim} = \mathbf{Y}^{\sim}(A, j_{l+1}, s_{l+1}) := \mathcal{D}_X(\tau^{\sim}_{j_{l+1}, s_{l+1}}(A)).$$

Gesucht sind geeignete Schätzwerte $\mathbf{c}_B, \mathbf{c}_{B^*}$, sowie Parameter j_{l+1}, s_{l+1} für Trennungen

$$B := \tau^{\leq}_{j_{l+1}, s_{l+1}}(A) \text{ und } B^* := \tau^{>}_{j_{l+1}, s_{l+1}}(A),$$

die das Minimierungsproblem

$$\min_{j_{l+1}, s_{l+1}} \left[\min_{c_B} \sum_{Y_i \in \mathbf{Y}^{\leq}} (Y_i - c_B)^2 + \min_{c_{B^*}} \sum_{Y_i \in \mathbf{Y}^{>}} (Y_i - c_{B^*})^2 \right] \tag{3.3}$$

lösen.

In Bezug auf die beiden inneren Minimierungen wird unabhängig von der Wahl der anderen Parameter stets $c_B := \overline{Y}(B, \mathcal{D}^{\text{Train}}_{n_1})$ und $c_{B^*} := \overline{Y}(B^*, \mathcal{D}^{\text{Train}}_{n_1})$ gesetzt.

Diese Wahl hat Vor- und Nachteile.

Der Nachteil ist, dass das arithmetische Mittel mit $\varepsilon = \frac{1}{n_1}$ von Natur aus einen schlechten Breakdown Point[2] hat. Eine Alternative bietet hier die Medianbildung.

[2] $\varepsilon = \frac{1}{n_1}$ bedeutet, dass der betreffende Mittelwert gegen unendlich strebt, falls dies für den n_1-ten Teil der Daten der Fall ist. Im vorliegenden Fall genügt hierfür also bereits ein einzelner Datenpunkt.

Doch das arithmetische Mittel hat auch einen Vorteil: Auf $\mathcal{D}_n$ gilt für alle $A \subseteq \mathbb{R}^d$, dass

$$\overline{Y}(A) = \arg\min_{a \in \mathbb{R}} \sum_{i=1}^{n_1} (Y_i - a)^2,$$

denn

$$\begin{aligned} \tfrac{1}{n}\sum_{i=1}^{n}(Y_i - a)^2 &= \tfrac{1}{n}\sum_{i=1}^{n}[(Y_i - \overline{Y}) + (\overline{Y} - a)]^2 \\ &= \tfrac{1}{n}\sum_{i=1}^{n}(Y_i - \overline{Y})^2 + (\overline{Y} - a)^2 + 2(\overline{Y} - a)\tfrac{1}{n}\sum_{i=1}^{n}(Y_i - \overline{Y}) \\ &=: \alpha + \beta + \gamma \end{aligned}$$

wird minimal für $a = \overline{Y}$, denn α hängt nicht von a ab, $\beta = 0$ für $a = \overline{Y}$ und $\beta \geq 0$ sonst und schließlich $\gamma = 0$, da für die Summe gilt, dass

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \overline{Y}) = \left(\frac{1}{n}\sum_{i=1}^{n} Y_i\right) - \overline{Y} = \overline{Y} - \overline{Y} = 0.$$

So reduziert sich (3.3) auf

$$\min_{j_{l+1}, s_{l+1}} \left[\sum_{Y_i \in \mathbf{Y}^{\leq}} (Y_i - \overline{Y}(\tau^{\leq}_{j_{l+1}, s_{l+1}}(A)))^2 + \sum_{Y_i \in \mathbf{Y}^{>}} (Y_i - \overline{Y}(\tau^{>}_{j_{l+1}, s_{l+1}}(A)))^2 \right].$$

Das nun vorliegende Problem

$$s_{l+1} = \arg\min_{\substack{\widehat{s} \in (\min(X_j^{(A)}), \\ \max(X_j^{(A)}))}} \left[\sum_{Y_i \in \mathbf{Y}^{\leq}} (Y_i - \overline{Y}(\tau^{\leq}_{j_{l+1}, \widehat{s}}(A)))^2 + \sum_{Y_i \in \mathbf{Y}^{>}} (Y_i - \overline{Y}(\tau^{>}_{j_{l+1}, \widehat{s}}(A)))^2 \right]$$

ist für jedes $j_{l+1} \in \{1, \ldots, d\}$ mit tragbarem Rechenaufwand numerisch lösbar. Das in Bezug auf (3.3) bestbefundene Paar (j_{l+1}, s_{l+1}) wird dann zusammen mit den zugehörigen Mittelwerten c_B und c_{B^*} verwendet, um $\mathcal{T}_0$ im vierten Schritt zu redefinieren.

Schritt 5: Im Augenblick besteht das Ziel darin, einen Baum zu erzeugen, der die Daten überanpasst und nur noch durch Ausschneiden weiter verbessert werden kann. Diesem Gedanken sollte auch das Entscheidungskriterium für oder wider eine weitere Aufspaltung von B und/oder B^* Rechnung tragen.

Ein mögliches Kriterium wäre beispielsweise ein Abbruch der Aufspaltung von B, falls $|\mathcal{D}(B, \mathcal{D}_{n_1}^{\text{Train}})| < r$ wird, wobei r „klein “, im Extremfall gleich zwei, gewählt wird. Analog für B^*.

Hat $\mathcal{T}_0$ eine ausreichende Komplexität erreicht, beginnt die zweite Phase des Algorithmus: Das Zuschneiden.

3.4.2 Teil II des Trainingsalgorithmus: Cost Complexity Pruning

Der folgende Abschnitt orientiert sich an [Koh00], verwendet aber für die empirischen Kosten eines Teilbaumes $\mathcal{T}_K$ nicht die Anzahl der in $\mathcal{T}_K$ enthaltenen Entscheidungsknoten, sondern die Anzahl der Endknoten.

Im zweiten Teil des Algorithmus wird $\mathcal{T}_0$ geeignet ausgeschnitten, um eine möglichst gute Schätzfunktion zu erhalten.

Zu diesem Zweck wird für einen gegebenen Unterbaum $\widehat{\mathcal{T}} \leq \mathcal{T}_0$ mit $\widehat{\mathcal{T}} = (\widehat{\mathcal{Q}}, \widehat{f}, \widehat{D})$ ein Bewertungskriterium benötigt, welches sowohl eine gute Anpassung an die Daten honoriert, als auch eine übermäßig komplizierte Baumstruktur negativ auslegt.

Das beim Cost Complexity Pruning verwendete Kriterium hat die Gestalt einer zweigliedrigen Summe. Dabei beschreibt der erste, mit einem gewichtenden Faktor $\alpha \geq 0$ versehene, Ausdruck die Komplexität des zu bewertenden Baumes anhand der Anzahl seiner Endknoten. Der zweite Ausdruck schätzt den durch Anpassungsfehler zu erwartenden Verlust mit Hilfe einer dem empirischen L_2-Risiko ähnlichen Summe ab.

$$C_\alpha(\widehat{\mathcal{T}}) := \alpha|\widehat{\mathcal{T}}| + \sum_{i=1}^{n}(Y_i - \widehat{f}(X_i))^2 \tag{3.4}$$

Auf die konkrete Wahl eines geeigneten α wird im Abschnitt 3.5 noch eingegangen. Für den Augenblick sei α gegeben.

Mit Hilfe von C_α lässt sich das Ausschneideproblem konkretisieren: Gesucht ist $\mathcal{T}^*$ mit

$$\mathcal{T}^* = \underset{\widehat{\mathcal{T}} \leq \mathcal{T}_0}{\arg\min}\ C_\alpha(\widehat{\mathcal{T}}). \tag{3.5}$$

Leider ist die Untersuchung jedes einzelnen Baumes $\widehat{\mathcal{T}} \leq \mathcal{T}_0$ bei vielen praktischen Problemstellungen nicht in vernünftiger Zeit durchführbar. Stattdessen

wird das Ziel verfolgt, $\mathcal{T}^*$ durch sukzessives Ausschneiden einzelner Knoten zu finden. Dabei soll ein Knoten K ausgeschnitten werden, falls die dabei aufkommenden empirischen Kosten geringer ausfallen als die empirischen Kosten jedes Unterbaums $\widehat{\mathcal{T}}_K \leq \mathcal{T}_K$. Damit dies gelingt, muss zunächst C_α zu einem Kriterium ausgebaut werden, mit dem man die Kosten von in einzelnen Knoten K wurzelnden Teilbäumen $\mathcal{T}_0$ bemessen kann:

Sei K ein Knoten in $\mathcal{T}_0$. Die *empirischen Erhaltungskosten* $EK_\alpha(\mathcal{T}_K)$ des in K wurzelnden Teilbaumes $\mathcal{T}_K$ werden definiert als folgende Einschränkung von $C_\alpha(\mathcal{T}_0)$ auf Q_K:

$$EK_\alpha(\mathcal{T}_K) := \alpha|\mathcal{T}_K| + \sum_{i=1}^{n_{Q_K}} (Y_i^{(Q_K)} - f_K(X_i^{(Q_K)}))^2. \tag{3.6}$$

Diese Einschränkung ist mit C_α kompatibel. Das wird von folgendem Lemma begründet:

Lemma 3.12
Sei $\alpha \geq 0$, $\mathcal{R} \subseteq \mathbb{R}^d$, $\mathcal{D}_n$ eine wie üblich definierte Stichprobe mit X-Werten in $\mathcal{R}$ und $\mathcal{T}$ ein Baum auf $\mathcal{R}$. Seien $K_1, \ldots, K_r$ Knoten in $\mathcal{T}$ mit paarweise disjunkten Bereichen $Q_{K_1}, \ldots, Q_{K_r}$, welche die Eigenschaft haben, dass $Q_{K_1} \dot\cup \ldots \dot\cup Q_{K_r} = \mathcal{R}$. Dann gilt

$$C_\alpha(\mathcal{T}) = \sum_{j=1}^{r} EK_\alpha(\mathcal{T}_{K_j}).$$

Beweis: Da die Knotenbereiche $Q_{K_1}, \ldots, Q_{K_r}$ paarweise disjunkt sind und darüberhinaus ganz $\mathcal{R}$ abdecken, fällt jedes Stichprobenelement in genau einen Bereich $Q \in \{Q_{K_1}, \ldots, Q_{K_r}\}$. Ferner gibt es für den Bereich Q_K eines beliebigen Endknotens in $\mathcal{T}$ genau ein $Q \in \{Q_{K_1}, \ldots, Q_{K_r}\}$ mit $Q_K \subseteq Q$, weswegen

$$|\mathcal{T}| = \sum_{j=1}^{r} |\mathcal{T}_{K_j}|.$$

Das ermöglicht folgende Umformung:

$$\begin{aligned} C_\alpha(\mathcal{T}) &= \alpha|\mathcal{T}| + \sum_{i=1}^{n_\mathcal{R}} (Y_i^{(\mathcal{R})} - f(X_i^{(\mathcal{R})}))^2 \\ &= \alpha \left[\sum_{j=1}^{r} |\mathcal{T}_{K_j}| \right] + \sum_{j=1}^{r} \sum_{i=1}^{n_\mathcal{R}} \mathbf{I}_{\{(X_i,Y_i) \,\in\, \mathcal{D}(Q_{K_j})\}} (Y_i^{(\mathcal{R})} - f(X_i^{(\mathcal{R})}))^2 \\ &= \sum_{j=1}^{r} \left[\alpha|\mathcal{T}_{K_j}| + \sum_{i=1}^{n_{Q_{K_j}}} (Y_i^{(Q_{K_j})} - f_{K_j}(X_i^{(Q_{K_j})}))^2 \right] \\ &= \sum_{j=1}^{r} EK_\alpha(\mathcal{T}_{K_j}) \end{aligned}$$

□

Damit lassen sich die benötigten Kostenbegriffe für und wider das Ausschneiden einzelner Knoten einführen: Sei wieder K ein Knoten in $\mathcal{T}_0$.

- Die Kosten, die entstehen, wenn K zugunsten einer Konstanten c_K ausgeschnitten wird, werden im Folgenden als die *empirischen Ausschneidekosten* $AK_\alpha(K)$ bezeichnet. Sie berechnen sich direkt mit $EK_\alpha(\widehat{\mathcal{T}}_K)$, wobei angenommen wird, dass $\widehat{\mathcal{T}}_K \leq \mathcal{T}_K$ der triviale Baum $(\widehat{\mathcal{Q}}_K = \{Q_K\}, \widehat{f}_K = c_K, \widehat{D}_K)$ ist:

$$AK_\alpha(K) := EK_\alpha(\widehat{\mathcal{T}}_K) = \alpha \cdot 1 + \sum_{i=1}^{n_{Q_K}} (Y_i^{(Q_K)} - c_K)^2 \tag{3.7}$$

- Für die Kosten eines Verzichts auf das Ausschneiden von K muss jeder denkbare, nicht triviale Unterbaum $\widehat{\mathcal{T}}_K \leq \mathcal{T}_K$ in Betracht gezogen werden. Dennoch lassen sich auch diese *empirischen Knotenkosten* $KK_\alpha(K)$ mit EK_α ausdrücken:

$$KK_\alpha(K) := \min_{\substack{\widehat{\mathcal{T}}_K \leq \mathcal{T}_K \\ \widehat{\mathcal{T}}_K \text{ nicht trivial}}} EK_\alpha(\mathcal{T}_K) \tag{3.8}$$

Das in (3.8) aufgeworfene Minimierungsproblem lässt sich bei geschickter Wahl der Knotenreihenfolge umgehen. Doch dazu mehr in der Beschreibung des eigentlichen Ausschneidealgorithmus, dessen Vorbereitung an dieser Stelle abgeschlossen ist.

Zunächst wieder ein kurzer Überblick:

1. Festlegung eines Komplexitätsstraffaktors α zum Beispiel via Kreuzvalidierung. Siehe hierzu auch unter Abschnitt 3.5.
2. Vermerken der Hierarchie $H(K)$ jedes Knotens in $\mathcal{T}_0$.
3. Initialisierung des Ebenenzählers
$$l := 1 + \max_{K \in \{\text{Knoten in } \mathcal{T}_0\}} H(K).$$
4. $l := l - 1$
5. Für jeden Endknoten mit $H(K) = l$:
Vermerken der empirischen Erhaltungskosten $EK_\alpha(\mathcal{T}_K)$.
6. Für jeden Entscheidungsknoten mit $H(K) = l$:
   - Bestimmung von $AK_\alpha(K)$ und $KK_\alpha(K)$.
   - Falls $KK_\alpha(K) \geq AK_\alpha(K)$: Ausschneiden von K und Vermerken von $AK_\alpha(K)$ als empirische Erhaltungskosten $EK_\alpha(\mathcal{T}_K)$ des trivial gewordenen Teilbaumes $\mathcal{T}_K$.
   - Falls $KK_\alpha(K) < AK_\alpha(K)$: Verzicht auf das Ausschneiden von K und Vermerken von $KK_\alpha(K)$ als empirische Erhaltungskosten $EK_\alpha(\mathcal{T}_K)$
7. Falls $l > 0$ zurück zu Punkt 4.
8. Zurückliefern des ausgeschnittenen Baumes als Regressionsschätzer.

Erläuterungen:

Schritt 3: Der Algorithmus setzt an in den Knoten, die im Diagramm am weitesten von der Wurzel entfernt sind. Im ersten Schleifendurchlauf erhält man auf diese Weise nur Endknoten.

Schritt 6, Punkt 1: Die Kosten $AK_\alpha(K)$ für das Ausschneiden von K zugunsten einer Konstante c_K können, falls c_K gegeben ist, mit (3.7) sofort ausgerechnet werden. Mit derselben Argumentation wie in Schritt 3 des Wachstumsalgorithmus wird dabei $c_K = \overline{Y}(Q_K)$ gesetzt.

Zu den Knotenkosten: Seien K_1 und K_2 die beiden direkt an K anschließenden Unterknoten. K_1 und K_2 liegen im Augenblick in der $(l+1)$-ten Schicht, was bedeutet, dass die Erhaltungskosten ihrer Teilbäume $\mathcal{T}_{K_1}$ und $\mathcal{T}_{K_2}$ bereits bekannt sind.

Da $\mathcal{T}_{K_1}$ und $\mathcal{T}_{K_2}$ außerdem bereits auf die geringstmöglichen empirischen Erhaltungskosten zugeschnitten worden sind gilt, wieder mit der Argumentation aus Lemma 3.12 und $Q_K = Q_{K_1} \dot\cup Q_{K_2}$, dass

$$\begin{aligned} KK_\alpha(K) &= \min_{\substack{\widehat{\mathcal{T}}_K \leq \mathcal{T}_K \\ \widehat{\mathcal{T}}_K \text{ nicht trivial}}} EK_\alpha(\widehat{\mathcal{T}}_K) \\ &= EK_\alpha(\mathcal{T}_K) \\ &= \alpha|\mathcal{T}_K| + \sum_{i=1}^{n_{Q_K}} (Y_i^{(Q_K)} - f_K(X_i^{(Q_K)}))^2 \\ &= \sum_{j=1}^{2} \left[\alpha|\mathcal{T}_{K_j}| + \sum_{i=1}^{n_{Q_{K_j}}} (Y_i^{(Q_{K_j})} - f_K(X_i^{(Q_{K_j})}))^2 \right] \\ &= EK_\alpha(\mathcal{T}_{K_1}) + EK_\alpha(\mathcal{T}_{K_2}) \end{aligned}$$

Damit können $AK_\alpha(K)$ und $KK_\alpha(K)$ direkt verglichen werden.

3.5 Zur Wahl von α

Der Komplexitätsstraffaktor α wägt den Einfluss der Komplexität des Baumes gegen seine Fähigkeit, die Daten zu beschreiben ab.

$\alpha = 0$ ist das eine Extrem. Dies ergibt ein Kriterium, welches, unabhängig vom in Kauf genommenen Overfitting, den Baum bevorzugt, der die Daten am besten anpasst. Dies wäre der unbeschnittene Baum $\mathcal{T}_0$.

Das andere Extrem ist erreicht, sobald α das $(n-1)$-fache der empirischen Varianz der Daten erreicht oder übersteigt mit

$$\alpha \geq \sum_{i=1}^{n} (Y_i - \overline{Y}(\mathbb{R}^d))^2.$$

In diesem Fall gibt C_α stets dem trivialen Baum $\mathcal{T}_{\text{Trivial}} = \overline{Y}(\mathbb{R}^d)$ den Vorzug, da dann für beliebige Bäume $\mathcal{T}$ mit $|\mathcal{T}| \geq 2$ gilt, dass

$$C_\alpha(\mathcal{T}_{\text{Trivial}}) = \alpha + \sum_{i=1}^{n}(Y_i - \overline{Y}(\mathbb{R}^d))^2 \leq 2\alpha \leq \alpha|\mathcal{T}| + \sum_{i=1}^{n}(Y_i - \mathcal{T}(X_i))^2 = C_\alpha(\mathcal{T}).$$

Offensichtlich steht und fällt das Kriterium also mit der Wahl von α.

Dennoch war seine Festlegung nicht Gegenstand des Algorithmus. Vielmehr wird α als gegeben angenommen und die konkrete Wahl dieses Parameters beispielsweise einer Kreuzvalidierung überlassen, wie sie in Abschnitt 4.2.1 vorgestellt wird.

4 Nichtparametrische Regression in R

4.1 Motivation

Im Wintersemester 2001/2002 und im darauffolgenden Sommersemester wurde an der Universität Stuttgart die zweisemestrige Vorlesung *Statistik I/II für Wirtschaftswissenschaftler* gelesen.

Wie üblich wurden dazu Übungsgruppen angeboten, bei denen Aufgaben zum Teil auch schriftlich abzugeben waren. Ferner legten die meisten Studenten gegen Ende jeden Semesters zunächst eine Scheinklausur und später eine Prüfung ab.

Dabei kam die Frage auf, ob es möglich sei, schon während des laufenden Semesters zielsicher Studenten zu erkennen, die bei der Prüfung eher schlecht abschneiden werden, um diese schon im Vorfeld intensiver betreuen zu können. Eine statistische Problemstellung, für die sich eine Anwendung der bislang beschriebenen nichtparametrischen Regressionsmethoden anbietet. Sie gab den Anstoß zu dieser Arbeit.

Am Ende des Wintersemesters war zu den meisten teilnehmenden Studenten neben der Prüfungsnote eine Vielzahl von Informationen, wie deren Anwesenheitsdisziplin in den Übungsgruppen, mittlere erreichte Punktzahl bei schriftlicher Abgabe von Übungsaufgaben, Note in der Scheinklausur und noch einige andere Merkmale erfasst worden.

Die Zielsetzung war nun, dem Computer eines der bereits vogestellten Regressionsverfahren vorzugeben, ihn dann selbständig eine Auswahl aus den verfügbaren Prädiktormerkmalen treffen und schliesslich einen möglichst guten Schätzer trainieren zu lassen.

Die Wahl der Implementierungsplattform fiel dabei auf die speziell für statistische Fragestellungen entwickelte Interpretersprache R, die bereits grundlegende Funktionen zu neuronalen Netzen, Projection Pursuits und CARTS zur Verfügung stellt[1].

Auf dieser Grundlage entstand das in diesem Kapitel vorgestellte Programmpaket

[1] Für weitere Informationen zu R sei an dieser Stelle empfehlend auf die R Homepage verwiesen:
http://pbil.univ-lyon1.fr/Rdoc/

`nonpar.r`, welches auf der beigelegten CD zu finden ist und mit dem Befehl `source("nonpar.r")` in einer R-Umgebung geladen werden kann.

Eine Dokumentation der einzelnen Programme befindet sich im Anhang A, die zugehörigen Quellcodes im Anhang B.

Gegen Ende des Sommersemesters lag der zweite Datenblock bis auf die noch ausstehende Prüfung vollständig vor. Damit fanden die Programme bereits eine praktische Anwendung: Anhand der Daten des ersten Semesters konnte ein neuronales Netz trainiert werden, welches auf die Daten des zweiten Semesters angewandt wurde, um die Prüfungsnoten vorherzusagen. Daraufhin wurden aus den sechzig am schlechtesten bewerteten Studenten dreißig zufällig ausgewählt, die dann zu einem speziellen Prüfungsvorbereitungskurs eingeladen wurden. Doch dazu an späterer Stelle mehr.

4.2 Zwei weitere Parameter

Bevor ein fertiger Regressionsschätzer zu einem gegebenen Problem zur Verfügung steht, ist in der Regel eine Vielzahl von Parametern festzulegen. Diese müssen hier, um dem Anspruch der Nichtparametrisierung gerecht zu werden, automatisch bestimmt werden.

Die von den R-Paketen `nnet`, `modreg` und `tree` zur Verfügung gestellten Funktionen leisten dies für die drei bereits vorgestellten Verfahren, soweit diese behandelt wurden. Doch auf zwei wichtige Parameter wurde noch nicht eingegangen:

1. die Auswahl des *Hauptparameters*. Darunter wird im Folgenden verstanden:

   - Für neuronale Netze: Die Neuronenzahl.
   - Für Projection Pursuits: Die Anzahl der Ridge-Funktionen.
   - Für Regressionsbäume: Der für das in Gleichung (3.4) eingeführte Cost Complexity Kriterium benötigte Komplexitätsstraffaktor α.

   An dieser Stelle sei auch erwähnt, dass diese Auswahl nicht optimal geglückt ist. Die Programme in `nonpar.r` benötigen eine endliche Vorauswahl von möglichen Werten, aus denen sie dann mit Hilfe einer *K-fachen Kreuzvalidierung* einen geeigneten auswählen. Dazu mehr im Abschnitt 4.2.1.

2. Die Auswahl der Prädiktoren. Für das Problem der Notenvorhersage bei den Wirtschaftswissenschaftlern standen ursprünglich zehn mögliche Merkmale zur Verfügung. Welche davon der Konstruktion guter Vorhersagefunktionen zuträglich sind und welche lediglich die Varianz des Schätzers steigern und das Problem unnötig hochdimensional gestalten, war zunächst unklar.

4.2.1 Wahl des Hauptparameters - Kreuzvalidierung

Seien wie üblich $(X, Y), (X_1, Y_1), \ldots, (X_n, Y_n)$ unabhängig identisch verteilt mit $\mathcal{D}_n := \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ und sei $m_\vartheta(X)$ ein Schätzer für $m(X) = \mathbf{E}\{Y|X\}$ mit aus einer endlichen Menge $\Theta := \{\vartheta_1, \ldots, \vartheta_r\}$ zu wählendem Parameter $\vartheta \in \Theta$. Sei schließlich $\mathbf{E}\{L(Y, m_\vartheta(X))\}$ das Risiko, welches die Qualität des Schätzers anhand einer Kostenfunktion L bestimmt.

Da eine Berechnung des in Bezug auf L optimalen Parameters

$$\vartheta^* := \arg\min_{\vartheta \in \Theta} \mathbf{E}\{L(Y, m_\vartheta(X))\} \tag{4.1}$$

die Kenntnis der Verteilung von (X, Y) vorraussetzt, kann für $\mathbf{E}\{L(Y, m_\vartheta(X))\}$ in der Regel nur eine Schätzung angegeben werden.

Der naheliegende Ansatz, ϑ^* zu schätzen mit

$$\widehat{\vartheta} = \arg\min_{\vartheta \in \Theta} \sum_{j=1}^{n} L(Y_j, m_\vartheta(X_j)) \tag{4.2}$$

favorisiert dabei oftmals Funktionen, welche die Datenpunkte überanpassen.

Ein zweiter Ansatz, der beim Training eines Schätzers derartiges *Overfitting* vermeidet, liegt in der zufälligen Unterteilung des Datensatzes in einen Trainingsdatensatz $\mathcal{D}_{n_1}^{\text{Train}}$ und einen Testdatensatz $\mathcal{D}_{n_2}^{\text{Test}}$. Ist dies geschehen, kann ein anhand von $\mathcal{D}_{n_1}^{\text{Train}}$ trainierter Schätzer m_ϑ unter Verwendung „neuer“ Daten $\mathcal{D}_{n_2}^{\text{Test}}$ bewertet werden. Gesucht ist dann

$$\widehat{\vartheta} = \arg\min_{\vartheta \in \Theta} \sum_{j=1}^{n_2} L(Y_j^{(Test)}, m_\vartheta(X_j^{(Test)})). \tag{4.3}$$

Doch auch diese Vorgehensweise hat ihren Preis: Insbesondere wenn nur wenige Daten zur Verfügung stehen bedeutet die Unterteilung, dass für das Training der Schätzer nur ein Teil der Stichprobe zur Verfügung steht, der unter

Umständen bereits zu klein ist, um alle wesentlichen Eigenschaften der Regressionsfunktion auszudrücken. Damit wird die anhand von $\mathcal{D}_{n_2}^{\text{Test}}$ getroffene Wahl nur den Besten aus einer Menge von wiederum schlechten Schätzern liefern.

Die sogenannte *K-fache Kreuzvalidierung* mit $K \in \{1, \ldots, n\}$, wie sie unter anderem in [TH01] vorgestellt wird, stellt eine dritte Methode dar, die ebenfalls Unterteilungen in Trainings- und Testblöcke vornimmt, aber dennoch jeden Datenpunkt zum Training verwendet. Die Kreuzvalidierung findet in `nonpar.r` im Programm `getparameterCV` Verwendung.

Dabei wird zunächst $\mathcal{D}_n$ in $K \leq n$ ungefähr gleichgroße Blöcke

$$\left(\mathcal{D}^{(1)} = \{X_1^{(1)}, \ldots, X_{n_1}^{(1)}\}\right), \ldots, \left(\mathcal{D}^{(K)} = \{X_1^{(K)}, \ldots, X_{n_K}^{(K)}\}\right)$$

unterteilt. K ist ein Parameter der Kreuzvalidierung. Für die im vorliegenden Fall etwas mehr als 250 dem Training zur Verfügung stehenden Datenpunkte wurde $K = 10$ gewählt.

Sei nun für $j \in \{1, \ldots, K\}$ die Schreibweise $m_\vartheta^{(-j)}$ eingeführt für einen Schätzer, der mit Parameter ϑ auf den Daten

$$\mathcal{D}^{(1)}, \ldots, \mathcal{D}^{(j-1)}, \mathcal{D}^{(j+1)}, \ldots, \mathcal{D}^{(K)}$$

trainiert wurde.

Als nächstes werden für $\vartheta \in \Theta$ Schätzer $m_\vartheta^{(-1)}, \ldots, m_\vartheta^{(-K)}$ trainiert und der CV-Fehler (englisch: Cross Validation) wie folgt definiert:

$$CV(\vartheta) := \frac{1}{n} \sum_{j=1}^{K} \sum_{i=1}^{n_j} L(Y_i, m_\vartheta^{(-j)}(X_i)) \tag{4.4}$$

Die Schätzung für ϑ^* hat dann die Form

$$\widehat{\vartheta} := \operatorname*{arg\,min}_{\vartheta \in \Theta} CV(\vartheta).$$

4.2.2 Wahl geeigneter Prädiktoren

Eine weitere wichtige Entscheidung ist die Auswahl der Prädiktoren. Diese Problemstellung tritt bei der bereits erwähnten Prüfungsnotenvorhersage auf: Während des Wintersemesters wurden nicht weniger als zehn verschiedene

Merkmale erfasst, über deren Aussagekraft in Bezug auf die Response zunächst nur Vermutungen angestellt werden konnten.

Der naheliegende Ansatz, zunächst für jedes einzelne Merkmal Schätzer zu trainieren und auf dieser Grundlage eine geeignete Auswahl zu treffen, muss verworfen werden. Es ist nicht auszuschließen, dass zwei Merkmale, die einzeln betrachtet nicht für eine zuverlässige Vorhersage der Response taugen, zusammengenommen durchaus brauchbar sind. Ein einfaches Beispiel dazu:

Beispiel 4.1
Seien $X_1, X_2 \sim b(n = 1, p = 0.5)$-verteilte Zufallsvariablen und sei $Y \in \{0, 1\}$ wie folgt direkt von X_1 und X_2 abhängig:

$$Y = \mathbf{I}_{\{X_1 = X_2\}} \tag{4.5}$$

Offensichtlich ist auch $m(X_1, X_2) = \mathbf{E}\{Y|X_1, X_2\} = \mathbf{I}_{\{X_1 = X_2\}}$. Gesucht sei eine in Bezug auf das L_2-Risiko möglichst gute Approximation m^*. Dies soll für verschiedene Prädiktorenkombinationen untersucht werden.

Fall 1: Es soll eine möglichst gute Schätzfunktion m^* für $m(X_1, X_2)$ gefunden werden, die zunächst weder von X_1 noch von X_2 abhängen soll. In diesem Fall ist $m^* = \text{const}$ und man erhält

$$\begin{aligned} m^* &:= \arg\min_{\widehat{m} \in \mathbb{R}} \mathbf{E}\{|\widehat{m} - Y|^2\} \\ &= \arg\min_{\widehat{m} \in \mathbb{R}} \frac{1}{2}(\widehat{m} - 1)^2 + \frac{1}{2}(\widehat{m} - 0)^2 \\ &= \frac{1}{2} \end{aligned}$$

Fall 2: Nun sei ein einzelner Prädiktor, etwa X_1, gegeben. Dann hat m^* die Form

$$m^*(X_1) = \mathbf{I}_{\{X_1 = 1\}} a + \mathbf{I}_{\{X_1 = 0\}} b, \tag{4.6}$$

mit $a, b \in \mathbb{R}$.

Da wegen $Y \in \{0, 1\}$ gilt, dass $\mathbf{E}\{Y^2|X_1\} = \mathbf{E}\{Y|X_1\}$ und des weiteren

$\mathbf{E}\{Y|X_1\} = \mathbf{P}[X_1 = X_2] \cdot 1 + \mathbf{P}[X_1 \neq X_2] \cdot 0 = \frac{1}{2}$, folgt:

$$\begin{aligned} m^* :&= \arg\min_{\widehat{m}(X_1)} \mathbf{E}\{|\widehat{m}(X_1) - Y|^2 | X_1\} \\ &= \arg\min_{\widehat{m}(X_1)} \mathbf{E}\{\widehat{m}(X_1)^2 | X_1\} - \mathbf{E}\{2\widehat{m}(X_1)Y | X_1\} + \mathbf{E}\{Y^2 | X_1\} \\ &= \arg\min_{\widehat{m}(X_1)} \widehat{m}(X_1)^2 - \widehat{m}(X_1) + \frac{1}{2}. \end{aligned}$$

Hierbei ist mit (4.6)

$$\widehat{m}(X_1)^2 - \widehat{m}(X_1) + \frac{1}{2} = \mathbf{I}_{\{X_1=1\}} a(a-1) + \mathbf{I}_{\{X_1=0\}} b(b-1) + \frac{1}{2}. \tag{4.7}$$

Die rechte Seite von (4.7) wird Minimal für $(a, b) = (\frac{1}{2}, \frac{1}{2})$. Setzt man dies in (4.6) ein, so ergibt sich abermals $m^* = \frac{1}{2} = \text{const}$. Die Hinzunahme von X_1 allein hat also keinen Vorteil gegenüber dem Fall 1 erbracht. Analoges gilt für die alleinige Verwendung von X_2.

Fall 3: Sind sowohl X_1 als auch X_2 gegeben, kann sofort $m^* = \mathbf{I}_{\{X_1=X_2\}} = m$ gesetzt werden, obwohl X_1 beziehungsweise X_2 alleine betrachtet für eine Vorhersage von Y ungeeignet sind.

Die Aufgabe einer geeigneten Auswahl von Prädiktoren fällt in `nonpar.r` dem Programm `praediktorwahl()` zu. Dort wird dem in Beispiel 4.1 aufgezeigten Dilemma auf rechenintensive Weise begegnet, indem für jede denkbare, nichtleere Auswahl von Prädiktormerkmalen folgende Prozedur erfolgt:

- Aus einer vorgegebenen, endlichen Menge möglicher Hauptparameter wird ein geeigneter Wert via 10-facher Kreuzvalidierung ausgewählt.
- Zu diesem und der aktuellen Merkmalkombination wird ein konkreter Schätzer trainiert, der dann mit seinem empirischen L_2-Risiko auf einem zuvor abgespaltenen Testdatensatz bewertet wird.

`praediktorwahl()` liefert die Prädiktoren des Schätzers mit dem geringsten empirischen L_2-Risiko zurück.

4.3 Training

Sind die zu verwendenden Prädiktoren und der Hauptparameter ausgewählt, können die von R zur Verfügung gestellten Funktionen benutzt werden, um auf einem gegebenen Datensatz einen Regressionsschätzer zu trainieren.

Da zu diesem Zeitpunkt keine extern festzulegenden Parameter mehr übrig sind, kann der gesamte Datensatz den entsprechenden R-Funktionen übergeben werden.

4.4 Konstruktion und Anwendung eines konkreten Schätzers

In diesem Abschnitt wird ein Skript vorgestellt, welches die Funktionen in `nonpar.r` benutzt, um auf dem Datensatz `Rws.tab` des Wintersemesters ein neuronales Netz zur Prüfungsnotenvorhersage zu trainieren. Der dabei gewonnene Schätzer wird daraufhin auf die Daten `Rss.tab` des Sommersemesters angewandt und die Vorhersagen den tatsächlichen Leistungen der Studenten gegenübergestellt.

`nonpar.r`, `Rws.tab`, `Rss.tab`, sowie das Programm `demo.r` befinden sich auf der beiliegenden CD. Letzteres kann, nachdem die Dateien in ein entsprechendes Arbeitsverzeichnis kopiert wurden, in einer R-Umgebung mit dem Befehl `source("demo.r")` ausgeführt werden. `demo.r` ist eine zusätzlich mit Pausen und Kommentaren versehene Version des weiter unten aufgeführten Skriptes[2] und kann unter R bequem mit dem Befehl `source("demo.r")` ausgeführt werden. Die nachfolgende Tabelle enthält Kommentare zum Beispielprogramm. Eine detaillierte Dokumentation der Programme in `nonpar.r` befindet sich im Anhang A.

Zeile(n)	Kommentar
2-3	Benutzerabfrage, ob die Prädiktorspalten der einzulesenden Tabellen empirisch standardisiert werden sollen. Dazu mehr im Abschnitt 5.2.
5-9	Benutzerabfrage, welches der drei in nonpar.r verwendeten Regressionsverfahren benutzt werden soll. 1 steht hierbei für die neuronalen Netze, 2 für die Projection Pursuits und 3 für die Regressionsbäume.

[2] Das unmodifizierte Skript befindet sich ebenfalls auf der CD unter demoskript.r.

	Parallel wird die Auswahl der bei den Kreuzvalidierungen in Betracht zu ziehenden möglichen Hauptparameter getroffen. Um Rechenzeit zu sparen, werden nur wenige Werte in `ParRange` aufgenommen.
12	Liste der aus den Tabellen zu ladenden Merkmale. Wieder werden einige Merkmale von vornherein weggelassen, um die Rechenzeit dieses Beispielskriptes nicht ausufern zu lassen. Eine Legende der Spaltennamen ist im Anhang C zu finden.
13-14	Laden der Datenpunkte des Wintersemesters mit `klausur!=-1`. Datenpunkte mit `klausur==-1` gehören zu Studenten, die an der (freiwilligen) Klausur nicht teilgenommen haben. Diese Kürzung muss in Kauf genommen werden, da die Programme in `nonpar.r` nur mit dichotonen und ordinalen Daten zurecht kommen. Betroffen davon sind im Wintersemester 28 von 283 und im Sommersemester 25 von 220 Datenpunkten.
15-16	Ermittlung einer geeigneten Auswahl von Merkmalen als Prädiktoren für die Prüfungsnotenvorhersage. Für weitere Details zu `praediktorwahl()` sei auf die Dokumentation im Anhang A verwiesen.
17-18	Zu der bestbefundenen Prädiktorauswahl wird eine 10-fache Kreuzvalidierung über alle Datenpunkte vorgenommen, um einen geeigneten Hauptparameter zu finden. Dafür werden alle Werte in `ParRange` in Betracht gezogen.
19-21	Nun wird unter Verwendung der in `A$OptSpalten` aufgeführten Merkmale und des in `B$ParOpt` empfohlenen Hauptparameters eine Schätzfunktion trainiert.
22	Vermerken der zum soeben trainierten Schätzer gehörigen Formula.
25-26	Laden der Daten des Sommersemesters.
27-28	Sortieren der Daten nach den tatsächlich erreichten Prüfungsnoten.
29-32	Vorhersage der Prüfungsnoten.
33	Für den Sonderfall der Regressionsbäume wird der fertige Baum graphisch dargestellt.

34	Berechnung des empirischen L_2-Risikos der Vorhersagen mit Hilfe der tatsächlichen Prüfungsnoten.
35-37	Auftragen der geschätzten Werte gegen die tatsächlichen Prüfungsnoten.
38	Zug eines Smoothing Splines durch die Punktwolke.

```
#> Teil 1: Training eines Schaetzers mit den Daten Rws.tab-------------
Eingabe <- readline(" Praediktoren standardisieren (j/[n])? ");
if (Eingabe=="j") StdColumns<-2 else StdColumns<-FALSE;

Eingabe <-as.numeric(readline(" Bitte Verfahren waehlen ([1]/2/3): "));
Verfahren <- 1; ParRange <- c(1:3,5);
switch(Eingabe,{Verfahren <- 1; ParRange <- c(1:3,5);},
               {Verfahren <- 2; ParRange <- c(1:3,5);},
               {Verfahren <- 3; ParRange <- (0:6)/2;});

source("nonpar.r");
Col <- c("vmnm","stdgng","anwsnht","punkte","klausur","prfng")
SWS1 <- vorbereitung(fname="Rws.tab",Columns=Col,DelColumns="klausur",
                     DelSymbol=-1,StdColumns=StdColumns);
A <- praediktorwahl(Daten=SWS1,Response="prfng",Verfahren=Verfahren,
       Partitionen=10,ParRange=ParRange,Durchlaeufe=10,TrTsRatio=2/3);
B <- getparameterCV(Daten=SWS1[,A$OptSpalten],Response="prfng",
       Verfahren=Verfahren,Partitionen=10,ParRange=ParRange);
C <- training(Daten=SWS1[,A$OptSpalten],Response="prfng",
       Verfahren=Verfahren,Hauptparameter=B$ParOpt,
       Durchlaeufe=500,maxit=300);
Form <- C$Form;
#< ---------------------------------------------------------------------
#> Teil 2: Vorhersage der Pruefungsnoten auf den Daten Rss.tab---------
SWS2 <- vorbereitung(fname="Rss.tab",Columns=Col,DelColumns="klausur",
                     DelSymbol=-1,StdColumns=StdColumns);
SortIndex <- sort(SWS2$prfng,index.return=TRUE)$ix;
SWS2 <- SWS2[SortIndex,];
switch(Verfahren,
       {Predict <- cbind(SWS2$prfng,predict.nnet(C$fOpt,SWS2));},
       {Predict <- cbind(SWS2$prfng,predict.ppr(C$fOpt,SWS2)); },
       {Predict <- cbind(SWS2$prfng,predict.tree(C$fOpt,SWS2));
        par(mfrow=c(1,2)); plot(C$fOpt); text(C$fOpt);});
L2 <- round(sum((Predict[,1]-Predict[,2])^2)/(dim(Predict)[1]),digits=2);
main <- paste("Emp. L2-Risiko: ",as.character(L2),sep="");
plot(Predict,xlim=c(1,5),ylim=c(1,5),main=main,
     xlab="Wahre Pruefungsnoten",ylab="Vorhergesagte Pruefungsnoten");
lines(smooth.spline(Predict));
#< ---------------------------------------------------------------------
```

Am Ende des Demos steht ein Spline, der durch die Punktwolke aus Vorhersagen und tatsächlich erreichten Prüfungsnoten gelegt wird. Dieser sollte monoton steigend sein.

Ausserdem dürfte er auffallend flach verlaufen. Das liegt daran, dass die Prüfungen insgesamt sehr gut ausgefallen sind, und dass für das Training nur wenige Datenpunkte eher schlechter abschneidender Studenten vorliegen. Dies führt dazu, dass die fertigen Schätzer zwar gute Leistungen recht zielsicher vorhersagen, aber bei den eher schlechten Prüfungsnoten Probleme haben und oft zu gute Prognosen treffen.

Dem kann entgegengewirkt werden, indem die Datenpunkte schwacher Studenten stärker gewichtet werden. Eine Methode, dies zu bewerkstelligen, ist eine Vervielfältigung der betreffenden Datenpunkte. Dieser Trick wird im folgenden Kapitel Verwendung finden.

Abbildung 4.1 zeigt das Resultat eines Aufrufs des Demoprogramms für neuronale Netze mit nichtstandardisierten Prädiktoren. Das im Endeffekt trainierte Netz verfügte über zwei Neuronen und betrachtete die Merkmale `punkte` und `klausur`.

Die der Abbildung vorangestellte Tabelle enthält die ersten und die letzten fünf Zeilen des dabei von `praediktorwahl()` zurückgelieferten Evaluierungsframes. Die Tabelle ist nach der zweiten, die empirischen L_2-Risiken der zu den einzelnen Merkmalkombinationen trainierten Netze auf den Testdaten enthaltenden, Spalte sortiert. Dabei fällt auf, dass sich die Werte der ersten Zeilen nur geringfügig voneinander unterscheiden. Da letztere bei wiederholtem Aufruf von `praediktorwahl()` aufgrund der zufälligen Wahl des Testdatensatzes und der Startvektoren für die Gradientenabstiege außerdem schwanken, ergeben sich Probleme beim Versuch, eine „eindeutig beste“ Kombination von Merkmalen auszuwählen. Im überwiegenden Teil der Fälle wird jedoch das Merkmal `klausur` ausgewählt.

Abbildung 4.2 zeigt ein Ergebnis des Demoprogramms für Regressionsbäume[3] mit nichtstandardisierten Prädiktoren.

[3] Das Baumdiagramm wurde in MATLAB rekonstruiert.

Neuronen	L_2	CV	vmnm	stdgng	anwsnht	punkte	klausur
1	0.4678	0.6936	0	0	0	1	1
1	0.4685	0.6839	0	1	0	1	1
1	0.4697	0.6954	1	1	1	0	1
1	0.4698	0.7025	1	1	0	1	1
1	0.4708	0.7040	1	0	0	1	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2	0.6573	0.9235	0	1	1	1	0
2	0.6710	0.9782	1	1	0	1	0
1	0.6921	0.8517	1	1	1	1	0
1	0.7096	0.9376	1	0	0	1	0
3	0.9275	0.8731	1	0	1	1	0

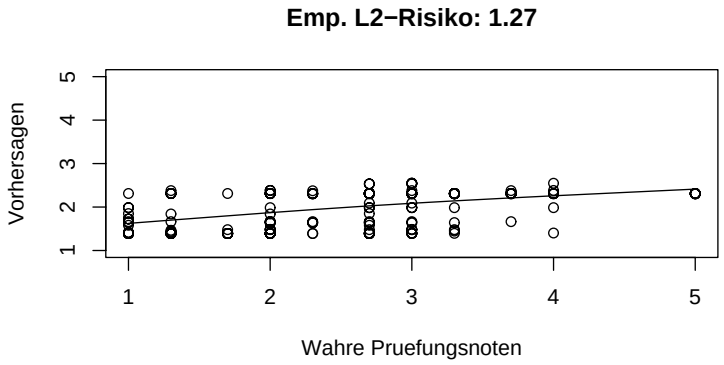

Abbildung 4.1: Demoresultat für ein neuronales Netz

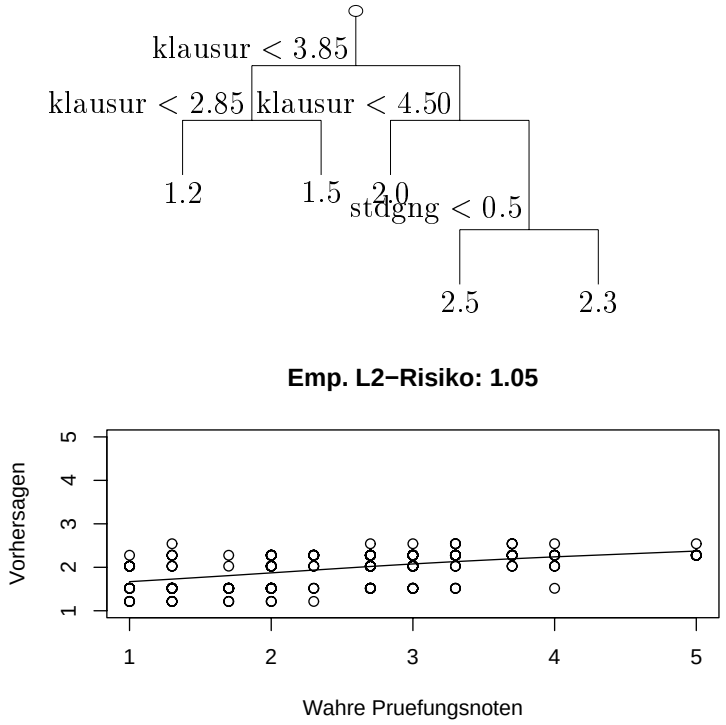

Abbildung 4.2: Demoresultat für einen Regressionsbaum

5 Anwendung

5.1 Bericht

Im Juli 2002 fanden die Programme in `nonpar.r` ihren Einsatz: Auf der Grundlage der vollständigen Daten des Wintersemesters sollte ein konkreter Schätzer für eine Regressionsfunktion gefunden werden. Letzterer sollte auf die während des Sommersemesters erfassten Daten angewandt werden, um Notenprognosen für die zugehörige Prüfung zu erhalten.

Nachdem der Schätzer trainiert war und Vorhersagen für die Prüfungsnoten der Wirtschaftswissenschaftler vorlagen, wurden die sechzig am schlechtesten prognostizierten Studenten herausgegriffen. Von diesen wurde die Hälfte zufällig als Fallgruppe ausgewählt und zu einem zur zusätzlichen Prüfungsvorbereitung organisierten, über drei Termine verteilten, Zusatzkurs eingeladen. Die restlichen dreißig Studenten wurden als Kontrollgruppe vermerkt.

Künftig werden diejenigen Studenten, die weder der Fallgruppe noch der Kontrollgruppe zugewiesen wurden, als „unauffällig“ bezeichnet.

Die beiden Fragen, um die es dabei im Wesentlichen ging, waren:

- Lassen sich schwächere Studenten annehmbar zuverlässig erkennen?
- Bringt das Anbieten eines Zusatzkurses einen sichtbaren Erfolg bei der Prüfung?

5.2 Training und Prognose

Die Wahl des grundsätzlichen Verfahrens fiel aus technischen Gründen auf die neuronalen Netze: Auf der Rechnerplattform, auf der `nonpar.r` entstand, verursachten die Programme `ppr()` und `tree()` leider zu häufig einen R zum Absturz bringenden *Segmentation Error*, als dass `praediktorwahl()` anwendbar gewesen wäre.

Für die zuverlässig laufenden Netze war eine Anwendung von `praediktorwahl()` möglich.

Vor dem Training des endgültigen Schätzers wurden noch einige Vorbereitungen getroffen:

1. Da die Programme in `nonpar.r` nur für dichotone und ordinale Daten ausgelegt sind, mussten alle Daten von Studenten, die nicht an der Scheinklausur teilgenommen hatten, fallengelassen werden. Davon waren im Wintersemester 28 von 283 und im Sommersemester 25 von 220 Datenpunkten betroffen.

2. Ferner wurden einige Merkmale von vornherein weggelassen:

   - `gruppe`[1] und `tutor`. Beides nicht ordinale, polychotone Merkmale.
   - `fachsem`. Konnte für die im Wintersemester an der Klausur teilnehmenden Studenten in 36 Fällen nicht erfasst werden. Außerdem wurden nur fünf Studenten registriert die sich nicht im ersten Semester befanden.
   - `whg`. Im Wintersemester gab es nur drei vollständige Datenpunkte zu Studenten, für welche die Prüfung eine Wiederholungsprüfung darstellte.

   Die `praediktorwahl()` zur Verfügung gestellten Merkmale waren somit `vmnm`, `zweit`, `stdgng`, `anwsnht`, `punkte`, `klausur`, `schein` und die Response `prfng`.

3. Alle Prädiktoren wurden empirisch standardisiert. Damit sollte verhindert werden, dass nach Training eines Schätzers auf den Daten des Wintersemesters die Vorhersage von Prüfungsnoten des Sommersemesters durch globale Schwankungen von Mittelwert und Standardabweichung der Merkmale beeinflusst wird. Weiterführende Veränderungen der Merkmalsverteilungen in Bezug auf die Semester lassen sich auf diese Weise nicht abfangen.
   Ein Nachteil dieser Vorgehensweise ist die für menschliche Betrachter schlechtere Interpretierbarkeit der fertigen Schätzer.

4. Die für die Kreuzvalidierung in Betracht gezogenen Neuronenzahlen wurden auf die Menge $\{1, \ldots, 10\}$ eingeschränkt. Diese Entscheidung fußt auf der Erfahrung aus Testläufen, die nie mehr als fünf Neuronen als Hauptparameter zurücklieferten.

5. Ein Anspruch an den Schätzer war, dass er eher schwache Studenten möglichst zuverlässig als solche erkennt. Demzufolge sollten Datenpunkte von Studenten, die in der ersten Prüfung schlecht abgeschnitten hatten,

[1] Eine Legende der Merkmale befindet sich im Anhang C.

beim Training ein höheres Gewicht erhalten. Dies wurde auf einfache Weise bewerkstelligt, indem vor dem Training alle Datenpunkte mit einer schlechteren Prüfungsnote als $2,0$ verdoppelt wurden. Betroffen waren 92 der 255 Datenpunkte des Wintersemesters.

Unter diesen Voraussetzungen wurde gegen Ende des Sommersemesters eine Reihe von `praediktorwahl()`-Aufrufen getätigt, die, wie schon im vorigen Kapitel anhand des Demobeispiels erläutert, teilweise unterschiedliche Merkmalkombinationen auswiesen. Bei fast jedem Aufruf wurde das Merkmal `klausur` als geeignet erkannt. Gute Schätzer ergaben sich vor allem für Kombinationen aus den Merkmalen `stdgng`, `anwsnht`, `punkte` und `klausur`. Die endgültige Wahl fiel auf die Prädiktoren `anwsnht`, `punkte` und `klausur`.

`getparameterCV()` schwankte bei der Neuronenzahlempfehlung zwischen den Werten eins und drei. Für das schließlich trainierte Netz wurden drei Neuronen verwendet. Dabei wurde davon ausgegangen, dass in diesem Fall für drei Merkmale und 255 Datenpunkte noch kein Overfitting vorliegt.

Auf der beigelegten CD befindet sich die Datei `netz.sav`, die in einer R-Umgebung mit dem Befehl `load("netz.sav")` geladen werden kann. Sie enthält die Objekte

Objekt	Beschreibung
`f`	Das zur Prüfungsnotenvorhersage herangezogene neuronale Netz.
`Form`	Die zugehörige Formula für den Befehl `predict.nnet()`.
`L2`	Das empirische L_2-Risiko von `f` auf den Daten des Wintersemesters.
`Predict`	Frame mit den Werten `anwsnht`, `punkte`, `klausur` für die Studenten des Wintersemesters zusammen mit den zugehörigen, von `f` erzeugten, Prüfungsnotenprognosen. Die sechzig schlechtest vorhergesagten wurden in Fall- und Kontrollgruppe für den bereits erwähnten Kurs eingeteilt.

5.3 Auswertung

5.3.1 Heuristische Betrachtungen

Anfang Oktober lagen auch die Prüfungsergebnisse des Sommersemesters vor. In der in einer R-Umgebung mit dem Befehl `load()` lesbaren Datei `noten.sav`

befinden sich die Vektoren `Kontroll`, `Fall` und `Unauff`. Sie enthalten die Prüfungsergebnisse der Studenten der einzelnen Gruppen. Ferner ist in der Datei das um die tatsächlichen Prüfungsergebnisse und eine Statusspalte erweiterte Frame `Predict` enthalten. In der Statusspalte steht `K` für Kontrollgruppe, `F` für Fallgruppe und `U` wieder für Unauffällige.

Die nachfolgende Tabelle enthält einige Werte zu den einzelnen Datensätzen. Dabei bezeichnet n die Anzahl der Datenpunkte, $\overline{y}$ das arithmetische Mittel der Prüfungsnoten, s^2 die empirische Varianz, *# best.* die Anzahl der Studenten, die bestanden haben, *% best.* deren Prozentsatz innerhalb der Gruppe und *# n.best.* die Anzahl der Studenten, die nicht bestanden haben.

	n	$\overline{y}$	s^2	# best.	# n.best.	% n.best.
Kontrollgruppe	29	3.08	1.22	24	5	17.2
Fallgruppe	29	2.82	0.55	29	0	0.0
Unauffällige Gruppe	131	2.37	0.89	128	3	2.3
Gesamt	189	2.55	0.95	181	8	4.2

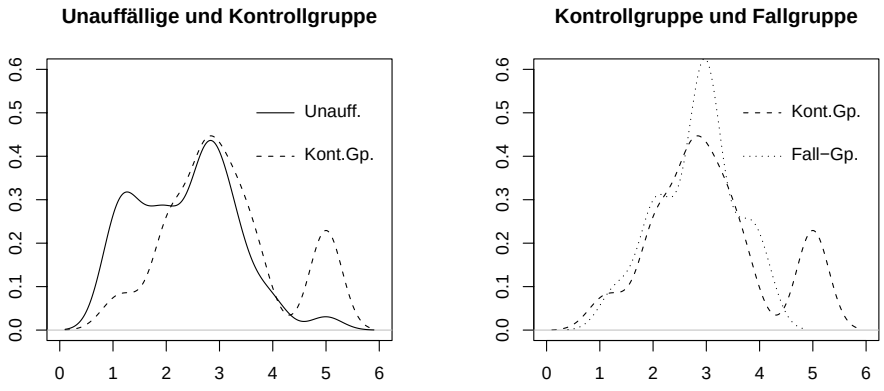

Abbildung 5.1: Geglättete Histogramme zu den einzelnen Gruppen

Abbildung 5.1 zeigt übereinandergelegte Plots von mit Hilfe eines Gauß-Kerns mit Bandbreite 0.3 geglätteten Histogrammen zu den oben beschriebenen Datenvektoren `Kontroll`, `Fall` und `Unauff`.

Tabelle und Abbildung sollen einen ersten Eindruck von den Notenverteilungen vermitteln. Tatsächlich scheinen die unauffällige Gruppe und die Fallgrup-

pe besser abgeschnitten zu haben als die Kontrollgruppe. Diese heuristischen Beobachtungen werden in den Abschnitten 5.3.2, 5.3.3 und 5.3.4 überprüft.

Zuvor sei jedoch noch angemerkt, dass die Gruppen nicht ganz unverfälscht sind: Obwohl versucht wurde, die Existenz des Kurses vor den nicht zur Fallgruppe gehörenden Studenten geheimzuhalten, konnte nicht verhindert werden, dass auch einige Studenten der unauffälligen Gruppe, oder gar der Kontrollgruppe den Kurs besuchten. Wer erschien durfte trotz des für die Untersuchung entstehenden Schadens teilnehmen. Andererseits besuchte auch nicht jeder Student der Fallgruppe den Kurs.

Dazu kam, dass nicht jeder Student in der Fallgruppe eine Einladung bekam: Jene, die einerseits in der Prüfung im Wintersemester mit 1.3 oder besser abgeschnitten hatten und deren beide Klausurnoten sich nicht um mehr als 0.5 unterschieden, wurden nicht eingeladen. Vielmehr wurde davon ausgegangen, dass sie trotz schlechter Prognose in der Prüfung gut abschneiden würden.

Um das Einfließen konfundierter Faktoren in die einzelnen Gruppen zu vermeiden, wurde jedoch an deren Zusammensetzung nichts geändert und obige Verfälschungen in Kauf genommen.

Die nachfolgende Tabelle beschreibt die Zahlen der Studenten, die am Kurs teilnahmen. Hierbei wird ein Student als *teilnehmend* gezählt, wenn er an mindestens einem der drei Termine anwesend war.

	# Teilnehmend	# Fernbleibend
Kontrollgruppe	3	26
Fallgruppe	19	10
Unauffällige Gruppe	4	127
Gesamt	26	163

5.3.2 Zweistichproben-t-Test

Zu einem vorgegebenen Signifikanzniveau von $\alpha = 5\%$ soll getestet werden, ob der Prüfungsnotendurchschnitt in der Kontrollgruppe schlechter ist, als in der unauffälligen oder der Fallgruppe.

Dies war mit einem einseitigen Zweistichproben-t-Test für unverbundene Stichproben mit im Allgemeinen unterschiedlicher Varianz zu zeigen.

Die zugehörige Teststatistik für zwei Stichproben $\mathcal{D}_1, \mathcal{D}_2$ zu Zufallsvariablen Y_1, Y_2, mit Stichprobenumfängen n_1, n_2, arithmetischen Mittelwerten $\overline{y}_1, \overline{y}_2$, empirischen Varianzen s_1^2, s_2^2 und für die Hypothesen $H_0 : \mathbf{E}\{Y_1\} \leq \mathbf{E}\{Y_2\}$, $H_1 : \neg H_0$ lautet (siehe zum Beispiel [Bro65])

$$T(\mathcal{D}_1, \mathcal{D}_2) = \frac{\overline{y}_1 - \overline{y}_2}{\sqrt{s_1^2/n_1 + s_2^2/n_2}}. \tag{5.1}$$

Die Anzahl der Freiheitsgrade für die t-Verteilung errechnet sich aus

$$f(\mathcal{D}_1, \mathcal{D}_2) = \frac{(s_1^2/n_1 + s_2^2/n_2)}{\frac{(s_1^2/n_1)^2}{n_1 - 1} + \frac{(s_2^2/n_2)^2}{n_2 - 1}}. \tag{5.2}$$

Steht nun $t_f(\alpha)$ für dasjenige Quantil der t-Verteilung mit f Freiheitsgraden, welches mit genau $[(1-\alpha)\cdot 100]$-prozentiger Wahrscheinlichkeit unterschritten wird, so ist H_0 zum Niveau α ablehnbar, falls

$$T > t_f(\alpha),$$

respektive falls für eine Zufallsvariable X, welche mit f Freiheitsgraden t-verteilt sei, der p-Wert $\mathbf{P}[X > T]$ kleiner als α bleibt.

Die nachfolgende Tabelle enthält die Werte für die einzelnen Vergleiche.

Vergleichsgruppen	$T(\mathcal{D}_\cdot, \mathcal{D}_\cdot)$	$f(\mathcal{D}_\cdot, \mathcal{D}_\cdot)$	p-Wert	H_0 ablehnen?
$(\mathcal{D}^{\text{Kontroll}}, \mathcal{D}^{\text{Unauff}})$	3.21	37.6	0.0014	Ja
$(\mathcal{D}^{\text{Kontroll}}, \mathcal{D}^{\text{Fall}})$	1.08	49.1	0.1437	Nein

Damit lässt sich zum Niveau α aussagen, dass die Gruppe der für das Netz unauffälligen Studenten in der Prüfung tatsächlich signifikant stärker war, als die Studenten der beiden als gefährdet ausgewiesenen Gruppen.

Eine entsprechende Aussage zwischen der Kontrollgruppe und der Fallgruppe ließ sich nicht treffen, was zum Teil auch an den oben genannten Verfälschungen durch Fallgruppenstudenten, die dem Zusatzkurs fernblieben und durch Kontrollgruppenstudenten, die uneingeladen kamen liegen mag.

5.3.3 Test für ein dichotones Merkmal

Interessanterweise erhält man für den Vergleich Kontrollgruppe/Fallgruppe eine signifikante Aussage, wenn man nicht die Prüfungsnoten der Studenten, sondern schlicht deren Bestehverhalten untersucht. Dazu folgende Überlegung:

Die Kontrollgruppe besteht aus $n_1 = 29$ Studenten. Ebenso die Fallgruppe mit $n_2 = 29$ Prüflingen. In diesen Gruppen haben insgesamt $D = 5$ Kandidaten die Prüfung nicht bestanden. Davon waren alle, $k = 5$, in der Kontrollgruppe.

H_0: „Die Gruppenzugehörigkeit der Studenten hat keinen Einfluss auf deren Bestehverhalten bei der Prüfung.“.

Unter H_0 ist die Unterteilung der Stichprobe in Kontrollgruppe und Fallgruppe eine rein zufällige Angelegenheit und für eine Zufallsvariable Z, die die Anzahl der nicht bestehenden Studenten in der Kontrollgruppe beschreibt erhält man mit elementarer Kombinatorik

$$\mathbf{P}[Z \geq k] = \sum_{i=k}^{D} \frac{\binom{D}{i}\binom{n_1+n_2-D}{n_1-i}}{\binom{n_1+n_2}{n_1}} = \frac{\binom{5}{5}\binom{53}{24}}{\binom{58}{29}} \approx 0.026$$

und kann H_0 ablehnen[2].

Man kann also zum Niveau von 5% aussagen, dass die zum Zusatzkurs eingeladene Fallgruppe eine geringere Durchfallquote aufweist als die Kontrollgruppe.

Woran liegt das? Ein Erklärungsversuch: Die Studenten hatten insgesamt nur sehr wenig Zeit sich auf die Prüfung vorzubereiten. Auf diese Weise ist anzunehmen, dass sich viele der Prüflinge jenseits des Kurses nicht mehr all zu intensiv mit der Materie befassten. Der Kurs jedoch war nicht perfekt auf die kommende Prüfung abgestimmt und weder wurde dort alles behandelt, was gefragt wurde, noch wurde alles abgeprüft, was der Kurs vermittelte. So mag das alleinige Verfolgen des Vorbereitungskurses zwar ausgereicht haben, um die Prüfung zu bestehen, aber eben nicht, um eine überdurchschnittlich gute Note zu erhalten. Tatsächlich liefert die Tabelle am Anfang von Abschnitt 5.3.1 für die Fallgruppe eine wesentlich geringere empirische Standardabweichung, als für die Kontrollgruppe.

[2] Analog erhält man für den Vergleich der Kontrollgruppe mit der Gruppe unauffälliger Studenten mit $n_1 = 29, n_2 = 131, D = 8$ und $k = 5$ den Wert $\mathbf{P}[Z \geq 5] \approx 0.0054$.

Aus diesem Anlass wird darum zum Abschluss dieser Untersuchung noch ein Test auf Gleichheit der Varianzen in beiden Stichproben durchgeführt.

5.3.4 F-Test zum Vergleich der Varianzen

Sei wieder das Niveau $\alpha = 5\%$ vorgegeben.

Seien ferner zwei Stichproben $\mathcal{D}_1, \mathcal{D}_2$ zu Zufallsvariablen Y_1, Y_2 mit Stichprobenumfängen n_1, n_2 und empirischen Varianzen s_1^2, s_2^2 gegeben.

Zu prüfen ist die Hypothese $H_0 : \mathbf{V}\{Y_1\} \leq \mathbf{V}\{Y_2\}$ gegen $H_1 : \neg H_0$.

Zu diesem Zweck wird ein F-Test zur Gleichheit der Standardabweichungen, zweier unabhängiger Stichproben, wie er beispielsweise in [Bro65] vorgestellt wird, durchgeführt. Die zugehörige Statistik lautet:

$$F = \frac{s_1^2}{s_2^2}.$$

Dabei wird H_0 zum Niveau α abgelehnt, falls

$$F > F_{1-\alpha,n_1-1,n_2-1},$$

wobei $F_{1-\alpha,n_1-1,n_2-1}$ das $(1-\alpha)$-Quantil der F-Verteilung und $(n_1 - 1)$ die Anzahl der Freiheitsgrade für s_1^2 und $(n_2 - 1)$ die Anzahl der Freiheitsgrade für s_2^2 beschreibt.

Im vorliegenden Fall mit $s_1^2 = 1.22$ für die Kontrollgruppe und $s_2^2 = 0.55$ für die Fallgruppe ergibt sich

$$F = 2.22 > 1.88 \approx F_{0.95,28,28},$$

respektive, für eine F(28,28)-verteilte Zufallsvariable X, der p-Wert $\mathbf{P}[X > 2.22] \approx 0.019$.

Man kann also zum Niveau 5% davon ausgehen, dass die Varianz der Prüfungsnoten in der Fallgruppe wirklich größer ist, als in der Kontrollgruppe.

Bemerkung: Dennoch wird ein gefährlicher Pfad beschritten, wenn der Erklärungsversuch am Ende vom Abschnitt 5.3.3 nun als statistisch gesichert betrachtet wird: Bereits die Aussage der höheren Durchfallquote bei den Angehörigen der Kontrollgruppe basierte auf einem Niveau von 5%. Wird diese zusammen mit der soeben, ebenfalls auf 5%-Niveau getroffenen, Aussage verwendet, wird das Risiko eines Irrtums erster Art bereits zwei mal eingegangen, was die Fehlerwahrscheinlichkeit erhöht.

Anhang

A Dokumentation der Programme

Im Folgenden werden Bedienung und Funktionsweise der einzelnen Programme erklärt. Die kommentierten Quellcodes können im Anhang B eingesehen werden. Für die Dokumentation wurde, soweit möglich, der Stil der R-Dokumentationen imitiert.

Bemerkungen:

- Die Programme in `nonpar.r` sind nur auf ordinale und dichotone Daten ausgelegt.
- Zugunsten übersichtlicherer Quellcodes wurde in allen Funktionen auf umfangreiche Fehlerabfragen verzichtet.

A.1 Anwenderfunktionen

vorbereitung

Beschreibung:
Liest eine mit Header versehene Tabelle mit Dateinamen `fname` als Data-Frame ein. Voraussetzung ist, dass die betreffende Datei mit dem R-Befehl `read.table()` lesbar ist. Ferner werden die Pakete `nnet`, `modreg` und `tree` eingebunden.

Syntax:

```
Daten <- vorbereitung(fname,Columns=TRUE,DelColumns=NULL,
                      DelSymbol=-1,StdColumns=NULL)
```

Parameter:

`fname`:	String, welcher den Dateinamen der zu lesenden Tabelle enthält.
`Columns`:	Liste mit Spaltennamen enthaltenden Strings. Es werden nur Spalten zurückgeliefert, die in `Columns` aufgeführt sind. Ausnahme: Wird `Columns=TRUE` gesetzt, wird die Tabelle komplett zurückgeliefert.
`DelColumns`:	Liste mit Spaltennamen enthaltenden Strings. Alle Zeilen der Tabelle, die in einer der betreffenden Spalten den Wert `DelSymbol` beinhalten, werden gelöscht. Soll keine derartige Nachbereitung der Tabelle stattfinden, kann `DelColumns=NULL` gesetzt werden.
`DelSymbol`:	Wert, der bei Auftreten in den in `DelColumns` beschriebenen Spalten das Löschen der betroffenen Zeilen nach sich zieht.
`StdColumns`:	Liste mit Spaltennamen enthaltenden Strings. Alle betreffenden Spalten der Tabelle werden empirisch standardisiert. Dies ist als Option für die Prädiktorspalten gedacht. Alternativ setzbar sind • `StdColumns=TRUE`: Alle Spalten werden standardisiert. • `StdColumns=2`: Alle Spalten außer der letzten werden standardisiert. • `StdColumns=FALSE`: Keine Spalte wird standardisiert. • `StdColumns=NULL`: Keine Spalte wird standardisiert.

Ausgabe:

Zurückgeliefert wird ein Datenframe.

Beispiel:

```
Daten <- vorbereitung(fname="studread.tab",
 Columns=c("anwsnht","punkte","klausur","prfng"),StdColumns=TRUE
 DelColumns=c("klausur"),DelSymbol=-1)
```

getparameterCV

Beschreibung:

Findet einen geeigneten Hauptparameter für einen Regressionsschätzer.

Syntax:

```
A <- getparameterCV(Daten,Response,Verfahren,Partitionen=10,
                    ParRange,CvHistory=FALSE,maxit=1000)
```

Parameter:

`Daten`: Datenframe, zu dem ein Regressionsschätzer gefunden werden soll.

`Response`: String, welcher den Namen der als Response zu verwendenten Spalte von `Daten` enthält.

`Verfahren`: Kann nur einen der Werte $1, 2, 3$ enthalten. Legt fest, welches Schätzverfahren verwendet wird.

- 1: Neuronales Netz.
- 2: Projection Pursuit.
- 3: Regression Tree.

`Partitionen`: Enthält die Mächtigkeit der für die Kreuzvalidierung zu verwendenden Partitionierung der Daten.

`ParRange`: Vektor mit den für den Schätzer in Frage kommenden Werten für den Hauptparameter. Dieser ist bei den

- neuronalen Netzen die Neuronenzahl.
- Projection Pursuits die Anzahl der Ridge-Funktionen.
- CARTs der Faktor α des Komplexitätsstrafterms.

`CvHistory`:	Boolsch. Wird `CVhistory=TRUE` gesetzt, so wird für CvOpt eine Matrix zurückgeliefert, deren erste Spalte die getesteten Hauptparameter und deren zweite Spalte die dabei gefundenen CV-Fehler enthält.
`maxit`:	Wird ignoriert, falls als Verfahren nicht die neuronalen Netze gewählt wurden. `maxit` enthält die maximale Anzahl von Iterationen, die beim Gradientenabstieg eines Aufrufs von `nnet.formula()` durchlaufen werden sollen.

Details:

Auf `Daten` wird zum gewählten Verfahren eine `Partitionen`-fache Kreuzvalidierung über alle Einträge des Vektors `ParRange` ausgeführt und dasjenige Element herausgegriffen, welches dabei als Hauptparameter den geringsten CV-Fehler liefert.

Ausgabe:

Zurückgeliefert wird der bestbefundene Hauptparameter `ParOpt`, sowie der zu diesem gehörige CV-Fehler, respektive die Matrix `CvHistory`.

Beispiel:

```
Daten <- vorbereitung(fname="studread.tab",
 Columns=c("anwsnht","punkte","klausur","prfng"),StdColumns=TRUE
 DelColumns=c("klausur"),DelSymbol=-1)
A <- getparameterCV(Daten,Response="prfng",Verfahren=1,
 Partitionen=10,ParRange=1:20,CvHistory=TRUE,maxit=1000)
```

training

Beschreibung:

Trainiert einen Regressionsschätzer.

Syntax:

```
A <- training(Daten,Response,Verfahren,HauptParameter,
              Durchlaeufe=1000,maxit=1000,
              Save=FALSE,fname="temp.dat")
Form <- A$Form
```

Parameter:

`Daten`: Datenframe, zu dem ein Regressionsschätzer gefunden werden soll.

`Response`: String, welcher den Namen der als Response zu verwendenten Spalte in `Daten` enthält.

`Verfahren`: Darf nur einen der Werte $1, 2, 3$ enthalten. Legt fest, welches Schätzverfahren verwendet wird.

- 1: Neuronales Netz.
- 2: Projection Pursuit.
- 3: Regression Tree.

`Hauptparameter`: Zu dem gewünschten Verfahren gehöriger Hauptparameter. Dieser ist bei den

- neuronalen Netzen die Neuronenzahl.
- Projection Pursuits die Anzahl der Ridge-Funktionen.
- CARTs der Faktor des Komplexitätsstrafterms.

`maxit`: Wird ignoriert, falls als Verfahren nicht die neuronalen Netze gewählt wurden. `maxit` enthält die maximale Anzahl von Iterationen, die beim Gradientenabstieg eines Aufrufs von `nnet.formula()` durchlaufen werden sollen.

`Durchlaeufe`: Anzahl der Trainingsdurchläufe. `Durchlaeufe` > 1 ist nur bei den `ppr()` und `nnet()`-Schätzern sinnvoll. Siehe unter Details. Wurden als Verfahren die Regressionsbäume gewählt wird automatisch `Durchlaeufe=1` gesetzt.

`Save`: Boolsch. Falls `Save==TRUE` werden `fOpt`, `L2Opt` und `Form` in eine Datei `fname` gespeichert.

`fname`: String. Enthält den Namen für die Datei, in die gegebenenfalls die Ergebnisse von `training()` abgespeichert werden.

Details:
Zu dem Regressionsproblem

(Response in `Daten`) $\sim$ (Prädiktoren in `Daten`)

mit gegebenem `Hauptparameter` wird ein Regressionsschätzer ermittelt, der wahlweise die Form eines neuronalen Netzes, eines Projection Pursuits, oder eines Regressionsbaumes annimmt. Dazu werden die von R zur Verfügung gestellten Funktionen `nnet()`, `ppr()` und `tree()` herangezogen.

Nachdem alle den letzteren Funktionen vorzugebenden Parameter bereits festgelegt wurden, wird keine weitere Unterteilung der Daten in Traings- und Testdatensatz mehr vorgenommen.

Da die neuronalen Netze und die Projection Pursuits Parameter enthalten, die zufällig initialisiert werden und darum zwei identische Aufrufe von `nnet()` oder `ppr()` zu verschiedenen Schätzern führen können, besteht die Möglichkeit `Durchlaeufe` viele Schätzfunktionen zu trainieren und dann diejenige auszuwählen, welche das geringste empirische L2-Risiko auf den Daten erreicht. Diese Vorgehensweise macht für die rein deterministisch ermittelten Regressionsbäume keinen Sinn.

Wichtig: Die zurückgelieferte Formula `Form` sollte in eine Variable `Form` übertragen werden. `predict()` kann sonst Probleme bei der Auswertung des fertigen Schätzers bekommen.

Ausgabe:
Zurückgeliefert werden

- `fOpt`, der bestbefundene Schätzer,
- `L2Opt`, das empirisch ermittelte L2-Risiko von `fOpt`.
- `Form`, die zu `fOpt` gehörende Formula.

Beispiel:

```
Daten <- vorbereitung(fname="studread.tab",
 Columns=c("anwsnht","punkte","klausur","prfng"),StdColumns=TRUE,
 DelColumns=c("klausur"),DelSymbol=-1)
```

```
P <- getparameterCV(Daten,Response="prfng",Verfahren=1,
  Partitionen=10,ParRange=1:10,CvHistory=TRUE,maxit=1000)
S <- training(Daten,Response="prfng",Verfahren=1,
  Hauptparameter=P$ParOpt,Durchlaeufe=1000,maxit=1000)
Form <- S$Form
```

praediktorwahl

Beschreibung:

Liefert einen Vorschlag für die bei einer Schätzung zu verwendenden Spalten des Daten-Frames.

Syntax:

```
A <- praediktorwahl(Daten,Response,Verfahren,Partitionen=10
                    ParRange,Durchlaeufe,TrTsRatio=2/3)
```

Parameter:

`Daten`:	Datenframe, zu dem ein Regressionsschätzer gefunden werden soll.
`Response`:	String, welcher den Namen der als Response zu verwendenten Spalte in `Daten` enthält.
`Verfahren`:	Darf nur einen der Werte $1, 2, 3$ enthalten. Legt fest, welches Schätzverfahren verwendet wird.

- 1: Neuronales Netz.
- 2: Projection Pursuit.
- 3: Regression Tree.

`Partitionen`: Wird direkt an `getparameterCV()` weitergereicht. Siehe unter Details.

`ParRange`: Vektor mit den für den Schätzer in Frage kommenden Werten für den Hauptparameter. Dieser ist bei den

- neuronalen Netzen die Neuronenzahl.
- Projection Pursuits die Anzahl der Ridge-Funktionen
- CARTs der Faktor des Komplexitätsstrafterms.

`Durchlaeufe`: Anzahl der Trainingsdurchläufe pro Spaltenkombination. `Durchlaeufe` > 1 ist nur bei den `ppr()` und `nnet()`-Schätzern sinnvoll. Siehe auch unter Details der Beschreibung von `training()`.
Wurden als Verfahren die Regressionsbäume gewählt wird automatisch `Durchlaeufe=1` gesetzt.

`TrTsRatio`: `praediktorwahl()` verwendet eine Unterteilung der Stichprobe in Trainings- und Testdatensatz. TrTsRatio ist in $(0, 1)$ zu wählen und gibt den Anteil der Daten an, der als Trainingsdatensatz verwendet wird.

Details:

Die Hauptaufgabe von `praediktorwahl()` besteht darin, für einen gegebenen Datensatz mit mehreren potenziellen Prädiktormerkmalen und einer Response eine Empfehlung auszugeben, welche Merkmale zur Konstruktion eines konkreten Regressionsschätzers des in `Verfahren` anzugebenden Typs gut geeignet sind.

Das Programm geht dabei wie folgt vor:

1. Unterteilung der Daten in Trainings- und Testdatensatz. Der Anteil der Trainingsdaten kann durch den Parameter `TrTsRatio` festgelegt werden. Default ist 2/3. Auf eine Kreuzvalidierung wird an dieser Stelle zugunsten einer Rechenzeiteinsparung verzichtet.
2. Für jede nichtleere Auswahl von Prädiktormerkmalen:
   - Auf den Trainingsdaten: Bestimmung eines geeigneten Hauptparameters. Die Parameter `ParRange`, `Partitionen`, `Verfahren` und `Response`, sowie die zu untersuchenden Merkmale samt Response

des Trainingsdatensatzes werden dabei direkt an `getparameterCV()` weitergeleitet. Für weitere Details siehe dort in der Dokumentation.

- Training eines konkreten Schätzers. Die Parameter `Durchlaeufe`, `Verfahren` und `Response`, der soeben ermittelte Hauptparameter und die zu untersuchenden Merkmale samt Response des Trainingsdatensatzes werden direkt an `training()` weitergeleitet. Für weitere Details siehe dort in der Dokumentation.
- Bestimmung des empirischen L_2-Risikos des fertigen Schätzers auf dem Testdatensatz.

Die Resultate dieses Vorgangs werden im Frame `EvaluierungsFrame` festgehalten.

Ausgabe:
Zurückgeliefert werden

- die bestbefundene Merkmalkombination `OptSpalten`.
- das Frame `EvaluierungsFrame`.
- als Nebenprodukt der bestbefundene Schätzer `fOpt` samt Formula `Form`.

`EvaluierungsFrame` ist ein Logbuch des Vorgangs. Jede Zeile entspricht dabei einer getesteten Auswahl von Prädiktorenmerkmalen. Zu den einzelnen Spalten:

- 1. Spalte: von `getparameterCV()` bestbefundener Hauptparameter.
- 2. Spalte: Die empirischen L2-Fehler der einzelnen Schätzfunktionen auf den Testdaten.
- 3. Spalte: Von `getparameterCV()` zurückgelieferter CV-Fehler zu den Hauptparametern der einzelnen Schätzfunktionen.
- Restliche Spalten: Als Prädiktoren verwendete Merkmalspalten.

Beispiel:

```
Daten <- vorbereitung(fname="studread.tab",
 Columns=c("anwsnht","punkte","klausur","prfng"),StdColumns=TRUE
```

```
 DelColumns=c("klausur"),DelSymbol=-1)
A <- praediktorwahl(Daten,Response="prfng",Verfahren=1,
 ParRange=1:10,Durchlaeufe=100,TrTsRatio=2/3)
```

A.2 Interne Funktionen

ausgabe

Beschreibung:

Gibt die eingegebenen Parameter auf dem Monitor aus.

Syntax:

`ausgabe(L2Opt=NULL,CvOpt=NULL,ParOpt=NULL)`

Parameter:

`L2Opt,CvOpt,ParOpt`: Für eine Anwendung von `as.character()` geeignete Ausdrücke. Können alternativ auch `NULL` sein. In diesem Fall wird der betreffende Parameter nicht ausgegeben.

binaer

Beschreibung:

Hilfsfunktion für `praediktorwahl()`. Rechnet eine natürliche Zahl in einen binären Vektor um.

Syntax:

`b <- binaer(n,Ziffern)`

Parameter:

`n`:	Umzuwandelnde natürliche Zahl.
`Ziffern`:	Gewünschte Ziffernanzahl mit $2^{\texttt{Ziffern}} > \texttt{n}$.

Ausgabe:

Zurückgeliefert wird ein boolscher Vektor mit `Ziffern` vielen Einträgen. Wird

`TRUE` als 1 und `FALSE` als 0 aufgefasst und werden die Einträge in einer Reihe aufgeschrieben, so erhält man die Binaerdarstellung von `n`.

Beispiel:

```
binaer(5,8)
binaer(254,8)
```

getdataCV

Beschreibung:

Hilfsfunktion für die Kreuzvalidierung. Liest die k-te Spalte einer `IndexMatrix` aus und verwendet wahlweise diese oder den Rest der Matrix, um die entsprechenden Datenpunkte eines Datenobjekts `Daten` auszulesen und als Test- beziehungsweise Trainingsdatensatz zurückzuliefern.

Syntax:

```
Index <- getdataCV(Daten,IndexMatrix,TestZeile,Ausgabe)
```

Parameter:

`Daten`:	Frame oder Tabelle, deren Zeilen als Datenpunkte aufgefasst werden.
`IndexMatrix`:	Eine Indexmatrix, wie sie von `partitionierung()` zurückgeliefert wird.
`TestZeile`:	Numerischer Wert. Enthält die Zeilennummer der als Testindex zu verwendenden Zeile der IndexMatrix.
`Ausgabe`:	String mit einem der folgenden Inhalte: • `"test"`: Dann wird ein Testdatensatz zurückgeliefert. • `"train"`: Dann wird ein Trainingsdatensatz zurückgeliefert.

Ausgabe:

Je nach Wahl in `Ausgabe` ein Test- oder Trainingsdatensatz aus `Daten`.

Beispiel:

```
Daten <- vorbereitung(fname="studread.tab",
```

```
 Columns=c("anwsnht","punkte","klausur","prfng"),StdColumns=TRUE
 DelColumns=c("klausur"),DelSymbol=-1)
IndexMatrix <- partitionierung(Daten,10)
TrainDat<-getdataCV(Daten,IndexMatrix,TestZeile=3,Ausgabe="train")
TestDat <-getdataCV(Daten,IndexMatrix,TestZeile=3,Ausgabe="test")
```

getformula

Beschreibung:

Liefert für das Datenframe `Daten` und die Spalte `Response` die für die einzelnen Schätzverfahren benötigte Formula zurück.

Syntax:

```
Form <- getformula(Daten,Response)
```

Parameter:

`Daten`:	Datenframe.
`Response`:	String, welcher den Namen der als Response zu verwendenden Spalte von `Daten` enthält.

Ausgabe:

Zurückgeliefert wird ein Formulaobjekt mit der gewünschten Response.

partitionierung

Beschreibung:

Hilfsfunktion für die Kreuzvalidierung. Liefert eine IndexMatrix zurück, deren Zeilen als Indexvektoren für eine zufällige Partitionierung des Frames `Daten` verwendet werden kann.

Syntax:

```
IndexMatrix <- partitionierung(Daten,Partitionen)
```

Parameter:

`Daten`:	Frame oder Tabelle, deren Zeilen als Datenpunkte aufgefasst werden.

`Partitionen` : Gewünschte Partitionenanzahl.

Ausgabe:

Matrix mit der in `Partitionen` angegebenen Anzahl von Zeilen. Jede Zeile ist als Indexvektor einer Partition zu verstehen. Nicht verwendete Matrixzellen sind auf 0 gesetzt.

Beispiel:

```
Daten <- vorbereitung(fname="studread.tab",
  Columns=c("anwsnht","punkte","klausur","prfng"),StdColumns=TRUE
  DelColumns=c("klausur"),DelSymbol=-1)
IndexMatrix <- partitionierung(Daten,10)
```

B Quellcodes und Kommentare

B.1 Anwenderfunktionen

vorbereitung

Zeile	Kommentar
4-6	Einbinden der Pakete für die neuronalen Netze, Projection Pursuits und Regressionsbäume.
11	Initialisiere boolschen Vektor `ColumnIndex` für die zu verwendenten Spalten in `RAW`.
12-19	Gegebenenfalls Auswertung von Strings in `Columns` für `ColumnIndex`.
28	In `Daten` werden nur die Zeilen erhalten, die in der Spalte `DelColumns[j]` nicht den Eintrag `DelSymbol` enthalten.
33	Das Löschen einzelner Zeilen zerstört die fortlaufende Nummerierung der Zeilen des Datenframes. Dies wird hier korrigiert.
42-53	Ermittlung des boolschen Vektors `StdColumns`, der die in `Daten` zu standardisierenden Spalten markiert.
54-55	Umsetzung von `StdColumns` in einen Indexvektor `Loop`.
56-60	Standardisierung der betroffenen Spalten.

```
 1 vorbereitung <- function(fname,Columns=TRUE,DelColumns=NULL,
 2                          DelSymbol=-1,StdColumns=NULL)
 3 {
 4 library(nnet);
 5 library(modreg);
 6 library(tree);
 7
 8 #> Einlesen der Daten in ein Frame.-----------------------------------
 9 RAW <- read.table(fname,header=TRUE);
10 ColNames <- attributes(RAW)$names;
11 ColumnIndex <- rep(TRUE,dim(RAW)[2]);
12 if (Columns[1]!=1){
13   ColumnIndex <- rep(FALSE,dim(RAW)[2]);
14   for (j in 1:length(Columns)){
15     if (!any(ColNames==Columns[j]))
16       cat(paste("Warnung: Spalte",Columns[j],"nicht gefunden.\n"));
17     ColumnIndex <- ColumnIndex|(ColNames==Columns[j]);
18   }
```

```
19 }
20 Daten <- RAW[,ColumnIndex];
21 #< ---------------------------------------------------------------------
22
23 #> Loeschen von Zeilen mit DelSymbol-Eintraegen.-----------------------
24 ColNames <- attributes(Daten)$names;
25 if (!is.null(DelColumns)){
26   for (j in 1:length(DelColumns)){
27     if (any(ColNames==DelColumns[j])){
28  Daten<-Daten[as.vector(Daten[,ColNames==DelColumns[j]])!=DelSymbol,];
29     } else {
30  cat(paste("Warnung: Del-Spalte",DelColumns[j],"nicht gefunden!\n"));
31     }
32   }
33   row.names(Daten) <- 1:dim(Daten)[1];
34 }
35 #< ---------------------------------------------------------------------
36
37 #> Ggf. Standardisierung der Praediktoren.-----------------------------
38 ColNameErrors <- 0;
39 ColNames <- attributes(Daten)$names;
40 if (!is.null(StdColumns)) if (StdColumns==0) StdColumns <- NULL;
41 if (!is.null(StdColumns)){
42   if (StdColumns==1) StdIndicator <- c(rep(T,dim(Daten)[2]));
43   if (StdColumns==2) StdIndicator <- c(rep(T,(dim(Daten)[2]-1)),F);
44   if (is.character(StdColumns)) {
45     StdIndicator <- rep(FALSE,dim(Daten)[2]);
46     for (j in 1:length(StdColumns)){
47       if (any(ColNames==StdColumns[j]))
48         StdIndicator <- (StdIndicator)|(ColNames==StdColumns[j])
49       else
50         cat(paste("Warnung: Zu standardisierende Spalte",StdColumns[j],
51                   "nicht gefunden!\n"));
52     }
53   }
54   Loop <- StdIndicator*(1:length(StdIndicator));
55   Loop <- Loop[Loop!=0];
56   for (j in Loop){
57     mu <- mean(Daten[,j]);
58     sd <- sd(Daten[,j])
59     Daten[,j] <- (Daten[,j]-mu)/sd;
60   }
61 }
62 #< ---------------------------------------------------------------------
63 return(Daten);
64 }
```

getparameterCV

Zeile	Kommentar
17-35	Für einen gegebenen Parameter `j`: Kreuzvalidierung. Dabei wird ein vorläufiger CV-Fehler ermittelt, der das `m`-fache des in Gleichung 4.4 vorgestellten CV-Fehlers darstellt. Da `m=const` entsteht kein Unterschied, wenn derartige vorläufige Werte verglichen werden. Die noch vorzunehmende Division wird vor dem Zurückliefern der Ergebnisse in den Zeilen 39 beziehungsweise 41 nachgeholt.
37-38	Vergleich des für Parameter `j` ermittelten Wertes `CvNew` mit dem bisherigen Bestwert `CvOpt`. Gegebenenfalls Aktualisierung von `CvOpt` und der Matrix `CvHistory`.

```
 1 getparameterCV <- function(Daten,Response,Verfahren,Partitionen=10,
 2                            ParRange,CvHistory=FALSE,maxit=1000)
 3 {
 4 #> Vorbereitung.-----------------------------------------------------
 5 m <- dim(Daten)[1];
 6 n <- dim(Daten)[2];
 7 FirstLoop <- TRUE;
 8 History <- matrix(c(0,0),1,2);
 9 RespSpalte <- ((attributes(Daten)$names)==Response)*(1:n);
10 RespSpalte <- RespSpalte[RespSpalte!=0];
11 IndexMatrix <- partitionierung(Daten,Partitionen);
12 Form <- getformula(Daten,Response);
13 #< ---------------------------------------------------------------
14
15 #> Finden eines geeigneten Hauptparameters via Kreuzvalidierung.-------
16 for (j in ParRange){
17   cat(paste("Aktueller Parameter: ",as.character(j),".\n"));
18   CvNew <- 0;
19   for (k in 1:Partitionen){
20     TrainDat <- getdataCV(Daten,IndexMatrix,k,"train");
21     TestDat  <- getdataCV(Daten,IndexMatrix,k,"test");
22
23     switch(Verfahren,
24       { f <- nnet.formula(formula=Form,data=TrainDat,
25                size=j,maxit=maxit,linout=TRUE);
26         CvNew <- CvNew+
27           sum((predict.nnet(f,TestDat)-TestDat[,RespSpalte])^2); },
28       { f <- ppr.formula(formula=Form,data=TrainDat,nterms=j);
29         CvNew <- CvNew+
30           sum((predict.ppr(f,TestDat)-TestDat[,RespSpalte])^2); },
31       { fBig  <- tree(formula=Form,data=TrainDat,mindev=0,minsize=2);
```

```
32         f     <- prune.tree(fBig,j);
33         CvNew <- CvNew+
34           sum((predict.tree(f,TestDat)-TestDat[,RespSpalte])^2);});
35   } # Next k
36   if (FirstLoop){ CvOpt <- CvNew+1; FirstLoop <- FALSE; }
37   if ( CvOpt > CvNew ){ ParOpt <- j; CvOpt <- CvNew; }
38   if (CvHistory) { History <- rbind(History,c(j,CvNew/m)); }
39 } # Next j
40 CvOpt <- CvOpt/m;
41 #< ------------------------------------------------------------------
42 ausgabe(L2Opt=NULL,CvOpt=CvOpt,ParOpt=ParOpt);
43 if (CvHistory) CvOpt <- History[2:dim(History)[1],];
44 return(CvOpt,ParOpt);
45 }
```

training

Zeile	Kommentar
7	Für das dritte Verfahren macht es keinen Sinn, mehrere Trainingsdurchläufe zu starten, da bei den Regressionsbäumen keine Parameter zufällig initialisiert werden.
18-32	Training `Durchlaeufe` vieler Schätzer `f`. Auswahl desjenigen `f` als `fOpt`, welches den geringsten empirischen L_2-Fehler auf den Daten liefert.

```
 1 training <- function(Daten,Response,Verfahren,Hauptparameter,
 2                      Durchlaeufe=1000,maxit=1000,
 3                      Save=FALSE,fname="temp.dat")
 4 {
 5 #> Vorbereitung.-----------------------------------------------------
 6 m <- dim(Daten)[1];
 7 if (Verfahren==3) Durchlaeufe <- 1;
 8 ColNames <- attributes(Daten)$names;
 9 FirstLoop <- TRUE;
10 if (!any(ColNames==Response))
11   stop(paste("(training) Response",Response,"nicht gefunden!\n"));
12 RespSpalte <- (ColNames==Response)*(1:dim(Daten)[2]);
13 RespSpalte <- RespSpalte[RespSpalte!=0];
14 Form <- getformula(Daten,Response);
15 #< ------------------------------------------------------------------
16
17 #> Training des Schaetzers, Ausgabe wichtiger Ergebnisse.--------------
18 for (j in 1:Durchlaeufe){
```

```
19   cat(paste(as.character(j),". Durchlauf\n",sep=""));
20   switch(Verfahren,
21     { f <- nnet.formula(formula=Form,data=Daten,
22              size=Hauptparameter,maxit=maxit,linout=TRUE);
23       L2New <- sum((predict.nnet(f,Daten)-Daten[,RespSpalte])^2); },
24     { f <- ppr.formula(formula=Form,data=Daten,nterms=Hauptparameter);
25       L2New <- sum((predict.ppr(f,Daten)-Daten[,RespSpalte])^2); },
26     { fBig  <- tree(formula=Form,data=Daten,mindev=0,minsize=2);
27       f     <- prune.tree(fBig,Hauptparameter);
28       L2New <- sum((predict.tree(f,Daten)-Daten[,RespSpalte])^2);}
29   );
30   if (FirstLoop){ L2Opt <- L2New+1; FirstLoop <- FALSE; }
31   if ( L2Opt > L2New ){ fOpt <- f; L2Opt <- L2New; }
32 }
33 L2Opt <- L2Opt/m;
34 ausgabe(L2Opt=L2Opt,CvOpt=NULL,ParOpt=Hauptparameter);
35 if (Save) save(fOpt,L2Opt,Form,file=fname);
36 return(fOpt,L2Opt,Form);
37 #< ------------------------------------------------------------------------
38 }
```

praediktorwahl

Zeile	Kommentar
12-15	Verschiebung der Response in die letzte Spalte von `Daten`.
17-20	Unterteilung der Stichprobe in Trainingsdatensatz `TrainDat` und Testdatensatz `TestDat`. Um Rechenzeit zu sparen, wurde hier auf eine Kreuzvalidierung verzichtet.
24	Für die Auswahl aus $(n-1)$ Prädiktorspalten werden später boolsche $(n-1)$-Vektoren herangezogen.
26-30	Initialisierung des `Evaluierungsframes`.
35	Jedes $j \in \{1, \ldots, \text{Kombinationen}\}$ entspricht binär und mit `Bin"arziffern` vielen Ziffern dargestellt einer zu testenden Kombination von Prädiktoren.
37	Darstellung der `j`-ten Spaltenkombination als boolscher Vektor. Der zusätzliche Wert `TRUE` am Ende steht für die Response, die bei jedem Durchlauf dabei sein soll.
41-47	Training eines für die aktuelle Spaltenkombination geeigneten Schätzers `S` mit geeignetem Parameter `P`.

```
 1 praediktorwahl <- function(Daten,Response,Verfahren,Partitionen=10,
 2                            ParRange,Durchlaeufe,TrTsRatio=2/3)
 3 {
 4 #> Vorbereitung.-----------------------------------------------------
 5 m <- dim(Daten)[1];
 6 n <- dim(Daten)[2];
 7
 8 ColNames <- attributes(Daten)$names;
 9 if (!any(ColNames==Response))
10  stop(paste("(praediktorwahl) Response",Response,"nicht gefunden!\n"));
11
12 RespSpalte <- (ColNames==Response)*(1:dim(Daten)[2]);
13 RespSpalte <- RespSpalte[RespSpalte!=0];
14 Daten <- Daten[,c((1:n)[-RespSpalte],RespSpalte)];
15 RespSpalte <- n;
16
17 TrainIndex <- sort(sample(1:m,size=TrTsRatio*m));
18 TestIndex  <- sort((1:m)[-TrainIndex]);
19 TrainDat   <- Daten[TrainIndex,];
20 TestDat    <- Daten[TestIndex,];
21
22 SpaltenNamen <- attributes(Daten)$names;
23
24 Binaerziffern <- n-1;
25 Kombinationen <- (2^Binaerziffern)-1;
26 EvaluierungsFrame <- matrix(rep(0,(3+Binaerziffern)*Kombinationen),
27   nrow=Kombinationen,ncol=(3+Binaerziffern));
28 EvaluierungsFrame <- as.data.frame(EvaluierungsFrame);
29 attributes(EvaluierungsFrame)$names<-c("Hauptpar.","L2","CV",
30              SpaltenNamen[-length(SpaltenNamen)]);
31 #< ---------------------------------------------------------------
32
33 #> Test aller Praediktorkombinationen.---------------------------------
34 FirstLoop <- TRUE;
35 for(j in 1:Kombinationen){
36 #>> Konstruktion eines Schaetzers zur Kombination j.-------------------
37   Spalten <- c(binaer(j,Binaerziffern),TRUE);
38   cat("\nNeue Spaltenkombination\n**********************\n  ");
39   cat(SpaltenNamen[Spalten[-length(Spalten)]]);cat("\n");
40
41   P <- getparameterCV(TrainDat[,Spalten],Response=Response,
42          Verfahren=Verfahren,Partitionen=Partitionen,ParRange=ParRange,
43          CvHistory=FALSE,maxit=1000);
44   S <- training(TrainDat[,Spalten],Response=Response,
45          Verfahren=Verfahren,Hauptparameter=P$ParOpt,
46          Durchlaeufe=Durchlaeufe,maxit=1000);
47   Form <- S$Form;
48 #<< ---------------------------------------------------------------
```

```
49
50 #>> Bewerte aktuelles Resultat an TestDat.----------------------------
51   fNew <- S$fOpt;
52   switch(Verfahren,
53     L2New<-sum((predict.nnet(fNew,TestDat)-TestDat[,RespSpalte])^2)/
54            dim(TestDat)[1],
55     L2New<-sum((predict.ppr(fNew,TestDat)-TestDat[,RespSpalte])^2)/
56            dim(TestDat)[1],
57     L2New<-sum((predict.tree(fNew,TestDat)-TestDat[,RespSpalte])^2)/
58            dim(TestDat)[1]
59   );
60
61  EvalZeile <- c(P$ParOpt,round(L2New,digits=4),round(P$CvOpt,digits=4),
62    Spalten[-length(Spalten)]);
63  EvaluierungsFrame[j,] <- EvalZeile;
64
65   if (FirstLoop){ FirstLoop <- FALSE; L2Opt <- L2New + 1; }
66   if (L2Opt > L2New){
67     L2Opt      <- L2New;
68     OptSpalten <- Spalten;
69     fOpt       <- fNew;
70     CvOpt      <- P$CvOpt;
71     ParOpt     <- P$ParOpt;
72     FormOpt    <- S$Form;
73   }
74   string <- paste("\nBisherige Top-Anwendung (praediktorwahl):\n",
75     "Emp. L2-Fehler :",as.character(L2Opt),"\n",
76     "Int. CV-Fehler :",as.character(CvOpt),"\n",
77     "Hauptparameter :",as.character(ParOpt),"\n",
78     "Dieses Bestergebnis trat ein fuer die Spaltenwahl","\n",sep="");
79   cat(string); cat(SpaltenNamen[c(OptSpalten[-length(OptSpalten)],F)]);
80   cat("\n\n");
81 #<< ------------------------------------------------------------------
82 } # Next j
83 #< -------------------------------------------------------------------
84
85 #> Sortiere EvaluierungsFrame nach L2-Fehlern.-----------------------
86 SortIndex <- sort(EvaluierungsFrame[,2],index.return=T)$ix;
87 EvaluierungsFrame <- EvaluierungsFrame[SortIndex,];
88 #< -------------------------------------------------------------------
89
90 #> Ausgabe wichtiger Ergebnisse.--------------------------------------
91 cat("Bestergebnis bei Spaltenwahl\n");
92 cat(SpaltenNamen[c(OptSpalten[-length(OptSpalten)],F)]); cat("\n");
93 ausgabe(L2Opt=L2Opt,CvOpt=CvOpt,ParOpt=ParOpt);
94 #< -------------------------------------------------------------------
95 OptSpalten <- attributes(Daten)$names[OptSpalten];
96 Form <- FormOpt;
```

```
97 return(OptSpalten,EvaluierungsFrame,fOpt,Form);
98 }
```

B.2 Interne Funktionen

ausgabe

```
 1 ausgabe <- function(L2Opt=NULL,CvOpt=NULL,ParOpt=NULL)
 2 {
 3 s <- "\nDie bestbefundene Anwendung lieferte:\n";
 4
 5 if (!is.null(L2Opt))
 6   s <- paste(s,"Emp. L2-Fehler :",as.character(L2Opt),"\n",sep="");
 7 if (!is.null(CvOpt))
 8   s <- paste(s,"CV-Fehler      :",as.character(CvOpt),"\n",sep="");
 9 if (!is.null(ParOpt))
10   s <- paste(s,"Hauptparameter :",as.character(ParOpt),"\n",sep="");
11
12 s <- paste(s,"\n",sep="");
13 cat(s);
14 }
```

binaer

```
 1 binaer <- function(n,Ziffern)
 2 {
 3 nOld <- n;
 4 b <- rep(FALSE,Ziffern);
 5 for (j in ((Ziffern-1):0)){
 6   if (n >= (2^j)){
 7     n <- n-(2^j);
 8     b[Ziffern-j] <- TRUE;
 9   }
10 }
11 if (n>0) stop(paste(as.character(Ziffern),
12            " Ziffern reichen nicht aus,\n  um n=",as.character(nOld),
13            " binaer darzustellen",sep=""));
14 return(b);
15 }
```

getdataCV

```
getdataCV <- function(Daten,IndexMatrix,TestZeile,Ausgabe)
{
m <- dim(IndexMatrix)[1];
TrainZeilen <- (1:m)[!((1:m)==TestZeile)];

TestIndex   <- as.vector(IndexMatrix[TestZeile,]);
TestIndex   <- sort(TestIndex[!(TestIndex==0)]);

TrainIndex  <- as.vector(IndexMatrix[TrainZeilen,]);
TrainIndex  <- sort(TrainIndex[!(TrainIndex==0)]);

TestDat     <- Daten[TestIndex,];
TrainDat    <- Daten[TrainIndex,];

if (Ausgabe=="test")  return(TestDat);
if (Ausgabe=="train") return(TrainDat);
}
```

getformula

```
getformula <- function(Daten,Response)
{
if (!any(attributes(Daten)$names==Response))
 stop(paste("Fehler: Response ",Response," in Daten nicht gefunden."));

n <- dim(Daten)[2];
SpaltenNamen <- attributes(Daten)$names;
RespPosition <- (SpaltenNamen==Response)*(1:n)
RespPosition <- RespPosition[!(RespPosition==0)];
FormIndex    <- c(RespPosition,(1:n)[!((1:n)==RespPosition)]);
Form         <- as.formula(Daten[,FormIndex]);
return(Form);
}
```

partitionierung

```
partitionierung <- function(Daten,Partitionen)
{
n            <- dim(Daten)[1];
LengthPart   <- ceiling(n/Partitionen);
MatrixValues <- c(sample(1:n,n),rep(0,Partitionen*LengthPart-n));
```

```
6 IndexMatrix  <- matrix(MatrixValues,nrow=Partitionen,ncol=LengthPart);
7 return(IndexMatrix);
8 }
```

C Daten

Auf CD beigelegt sind die Dateien `ws.tab` und `ss.tab`. Sie enthalten, in Ascii-Form, die im Verlauf der beiden Vorlesungen *Statistik I/II für Wirtschaftswissenschaftler* erfassten Daten. Die Tabellen sind vollständig, bis auf die Spalten mit den Matrikelnummern und den verantwortlichen Tutoren, die aus Datenschutzgründen weggelassen wurden. Die Dateien enthalten auch unvollständig erfasste Datenpunkte mit Ausnahme derer, bei denen die Prüfungsnote nicht erfasst werden konnte. Letztere wurde als unbrauchbar gelöscht.

Die ebenfalls beigelegten Dateien `Rws.tab` und `Rss.tab` enthalten nur die Datenpunkte von Studenten, bei denen die Merkmale `gruppe`, `vmnm`, `zweit`, `stdgng`, `anwsnht`, `punkte`, `klausur`, `schein` und `prfng` vollständig erfasst werden konnten.

`Rws.tab` und `Rss.tab` können in einer R-Umgebung zum Beispiel mit dem Befehl
`vorbereitung()` aus `nonpar.r` geladen werden.

Jede Zeile der Tabellen entspricht einem Studenten. Die Daten sind, abgesehen von der Halbordnung nach Übungsgruppen, unsortiert.

Legende:

Besondere Einträge:

$+a$	Deutet an, dass die betreffende Information nicht erfasst werden konnte. a ist das Zehnfache der Nummer der betroffenen Spalte. Dies funktioniert, da kein Merkmal in den Tabellen jemals einen Wert von 10 oder Größer annimmt.
-1	Kommt nur in der Spalte `klausur` vor. Deutet an, dass bekannt ist, dass der betreffende Student an der (freiwilligen) Scheinklausur nicht teilgenommen hat.

Spalten:

`gruppe`	Nummer der Übungsgruppe.
`vmnm`	„Vormittag/Nachmittag“. Enthält den Wert 0, falls der betreffende Student seine Übungsgruppe Vormittags besucht hat, andernfalls 1.
`zweit`	„Zweitwoche“. Die Hälfte der Übungsgruppen wurde in den geraden Kalenderwochen abgehalten, die andere in den ungeraden. So kam es, dass Studenten, die in einer Übungsgruppe der jeweiligen „Zweitwoche“ eingeteilt waren mehr Vorbereitungszeit für ein gegebenes Übungsblatt hatten als ihre Kommilitonen in der „Erstwoche“. Ein Student, der eine Übungsgruppe der „Zweitwoche“ besuchte erhielt in der zugehörigen Tabellenspalte den Wert 1, sonst 0.
`fachsem`	Aktuelle Fachsemesteranzahl des betreffenden Studenten.
`stdgng`	„Studiengang“. Es wird unterschieden zwischen Diplom und Magister. Studenten, die auf den Abschluss Diplom studieren erhielten in dieser Spalte den Wert 1, sonst 0.
`anwsnht`	„Anwesenheit“. Anteil der wahrgenommenen Übungstermine. Nimmt einen entsprechenden Wert in $[0, 1]$ an.
`punkte`	Im Mittel pro Übungstermin (wahrgenommene und verpasste) erreichte Anzahl von Punkten durch schriftliche Abgabe von Übungsaufgaben. Maximum pro Termin war im gesamten Semester 3 Punkte.
`klausur`	Erreichte Note in der Scheinklausur. -1 bedeutet, dass der betreffende Student an der Klausur nicht teilgenommen hat.
`schein`	1 falls der betreffende Student am Ende des Semesters den Übungsschein erhalten hat, 0 sonst.
`whg`	„Wiederholungen“ Anzahl der Versuche, die der betreffende Student schon zuvor unternommen hat, um die Prüfung zu bestehen.
`prfng`	„Prüfung“. Prüfungsnote des Studenten.

D Liste häufig verwendeter Symbole

$\sim$	Relationszeichen
$\dot{\cup}$	Mengenverknüpfung: Disjunkte Vereinigung
$\overline{\mathcal{F}}$	Kapitel 1: Abschluss von $\mathcal{F}$ in $C_K(\mathbb{R})$
$\overline{Y}$	Kapitel 3: Arithmetisches Mittel der Komponenten von Y
$\lvert\mathcal{T}\rvert$	für einen Regressionsbaum $\mathcal{T}$: Anzahl der Endknoten in $\mathcal{T}$
$a_j^{(n+1)}$	Input für das j-te Neuron in der $(n+1)$-ten Schicht: $a_j^{(n+1)} := w_{0,j}^{(n)} + \sum_{i=1}^{l_n} w_{i,j}^{(n)} o_i^{(n)}$
$\mathbf{a}_i$	Kapitel 2: Vektor der Vorzugsrichtung der i-ten Ridge-Funktion eines Projection Pursuit Schätzers
AK	empirische Ausschneidekosten eines Knoten in einem Regressionsbaum. Siehe Gleichung (3.7)
C_α	Cost Complexity Kriterium. Siehe Gleichung (3.4)
$C_K(\mathbb{R})$	Menge der stetigen, reellwertigen Funktionen auf einer kompakten Menge K
d	$d := \dim(X)$
$\delta_j^{(s)}$	$\delta_j^{(s)} := \frac{\partial f_{\mathbf{w}}}{\partial a_j^{(s)}}$
$\mathcal{D}_n$	$\mathcal{D}_n := \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ Stichprobe unabhängig identisch verteilter Zufallsvektoren
$\mathcal{D}(A)$	Einschränkung von $\mathcal{D}_n$ auf Stichprobenelemente mit X-Werten in $A \subseteq \mathbb{R}^d$
$\mathcal{D}_X(A)$	X-Werte aus $\mathcal{D}(A)$
$\mathcal{D}_Y(A)$	Y-Werte aus $\mathcal{D}(A)$
$\mathbf{E}\{X\}$	Erwartungswert einer Zufallsvariablen X
EK	empirische Erhaltungskosten eines Regressionsbaumes. Siehe Gleichung (3.6)

$f_{\mathbf{w}}$	neuronales Netz, welches bis auf die Gewichte $\mathbf{w}$ bereits festgelegt ist
g_ϑ	univariate Ridge-Funktion eines Projection Pursuit Schätzers mit Parameter ϑ
$\mathbf{I}_{\{A\}}$	Indikatorfunktion. $\mathbf{I}_{\{A\}} = 1$, falls die Bedingung A erfüllt ist und $\mathbf{I}_{\{A\}} = 0$ sonst
K	Kapitel 1: Neuronenanzahl eines neuronalen Netzes, oder kompakte Teilmenge von $\mathbb{R}^d$.
K	Kapitel 3: Ein Knoten innerhalb eines Baumdiagrammes
KK	empirische Knotenkosten. Siehe Gleichung (3.8)
l_j	Anzahl der Neuronen in der j-ten Schicht
m	Regressionsfunktion $m(X) := \mathbf{E}\{X\|Y\}$
m_n	empirisch bestimmte Schätzfunktion für m
M	Kapitel 2: Anzahl der Ridge-Funktionen eines Projection Pursuit Schätzers
n	Stichprobenumfang
$o_j^{(n)}$	Output des j-ten Neurons in der n-ten Schicht
$\mathbf{P}[A]$	Wahrscheinlichkeit eines Ereignisses A
Q	Quader in $\mathbb{R}^d$
$\mathcal{Q}$	binäre Partition des $\mathbb{R}^d$
$R(m_n)$	L_2-Risiko eines Schätzers m_n: $R(m_n) := \mathbf{E}\{\|m_n(X) - Y\|^2\}$
R_n	empirisches L_2-Risiko eines Schätzers m_n $R_n(m_n, \mathcal{D}_n) := \frac{1}{n}\sum_{i=1}^{n} \|m_n(X_i) - Y_i\|^2$
s	Anzahl der verdeckten Schichten eines neuronalen Netzes
σ	Sigmoidfunktion eines neuronalen Netzes
$\tau_{j,s}^{\sim}, \tau_{j,s,\mathcal{A}}^{\sim}$	Trennoperator. Siehe Definition 3.2

$\cdot^T$	Die Transponierte von $\cdot$
$\mathcal{T}$	Regressionsbaum. Siehe Definition 3.4
$U(A)$	mit $A \subseteq \mathbb{R}^d$. $X \sim U(A)$ bedeutet: X ist uniform- bzw. gleichverteilt auf A
$\mathbf{V}\{X\}$	Varianz einer Zufallsvariablen X
$\mathbf{w}$	Gewichtevektor eines neuronalen Netzes
$\widehat{\mathbf{w}}$	Durch Rückpropagierung bestimmter Vektor $\widehat{\mathbf{w}} := \arg\min_{\mathbf{w}} R_n(f_{\mathbf{w}})$
$w_{a,b}^{(c)}$	reellwertiger Eintrag in $\mathbf{w}$, welcher die Ausgeprägtheit der Verbindung zwischen Output des a-ten Neurons in der c-ten Schicht mit dem b-ten Neuron in der $(c+1)$-ten Schicht ausdrückt
X	d-dimensionaler Zufallsvektor. Prädiktor für Y
Y	Zufallsvariable. Response zu X

Literatur

[Bro65] Brownlee, K. A.: *Statistical Theory and Methodology In Science and Engineering.* Wiley Publications in Statistics, 1965.

[Hir] *Aufbau des Gehirns.* http://pylwww.unibe.ch/sec/. Physiologisches Insitut, Universität Bern.

[JHF81] Jerome H. Friedman, Werner Stuetzle: *Projection Pursuit Regression.* Journal of the American Statistical Association, 76:817ff, 1981.

[KH93] K. Hornik, M. Stinchcombe, H. White: *Some new results on neural network approximation*, 1993. Neural Networks, 6:1069-1072.

[Koh00] Kohler, Michael: *Vorlesungsskript WS 2000/01: Data Mining*, 2000. Universität Stuttgart, Mathematisches Institut A.

[Koh01] Kohler, Michael: *Vorlesungsskript WS 2001/02: Nichtparametrische Regressionsschätzung*, 2001. Universität Stuttgart, Mathematisches Institut A.

[LB84] Leo Breiman, Jerome H. Friedman, et al.: *Classification and Regression Trees.* Wadsworth Statistics/Probability Series, 1984.

[MK01] Michael Kohler, Adam Krzyżak: *A simple proof of approximation properties of neural networks*, 2001. Universität Stuttgart, Mathematisches Institut A.

[Rud64] Rudin, W.: *Principles of Mathematical Analysis.* 1964. McGraw-Hill, New York.

[TH01] Trevor Hastie, Robert Tibshirani, Jerome Friedman: *The Elements of Statistical Learning.* Springer Verlag, 2001.