Kapitel 13

Vorhersagegüte prädiktiver Modelle

Da empirische Daten fehlerbehaftet sind, bezieht die Anpassung eines statistischen Modells immer auch die Messfehler mit ein, die Parameterschätzungen orientieren sich daher zu stark an den zufälligen Besonderheiten der konkreten Stichprobe (*overfitting*). Die Güte der Passung des Modells lässt sich als Funktion $f(\cdot)$ der Abweichungen $E = Y - \hat{Y}$ der Modellvorhersage $\hat{Y}$ zu den tatsächlichen Werten der vorhergesagten Variable Y quantifizieren. Genauer soll $\hat{Y}_{X,Y}(X')$ die folgende Vorhersage bezeichnen: Zunächst wird ein Modell an einer Stichprobe mit Werten für Prädiktoren X und Zielvariable Y (Kriterium) angepasst. In die Vorhersagegleichung mit den Parameterschätzungen dieses Modells werden dann (potentiell andere) Prädiktorwerte X' eingesetzt, um die Vorhersage $\hat{Y}$ zu berechnen, die mit den tatsächlichen Beobachtungen Y' zu vergleichen sind. $f(E)$ ist die *Verlustfunktion*, die alle individuellen absoluten Abweichungen e_i auf einen Gesamtwert für die Vorhersagegenauigkeit abbildet.

Angewendet auf die zur Modellanpassung verwendete *Trainingsstichprobe* selbst ($X' = X$) soll $f(E)$ hier als *Trainingsfehler* (auch Resubstitutionsfehler) bezeichnet werden, angewendet auf andere *Teststichproben* aus derselben Grundgesamtheit ($X' \neq X$) als *Vorhersagefehler*. Die Passung des Modells für die Trainingsstichprobe ist dabei im Mittel besser als für andere Teststichproben aus derselben Grundgesamtheit. Der Trainingsfehler ist dahingehend verzerrt, dass er den Vorhersagefehler systematisch unterschätzt, also zu optimistisch bzgl. der Generalisierbarkeit des angepassten Modells ist. Die Größe des Vorhersagefehlers liefert auch einen Anhaltspunkt für die Auswahl eines bestimmten Modells aus mehreren möglichen (vgl. Abschn. 6.3.3).

Die folgenden Abschnitte stellen Kreuzvalidierung und Bootstrapping für eine möglichst unverzerrte Schätzung des Vorhersagefehlers vor (Hastie, Tibshirani & Friedman, 2009; James, Witten, Hastie & Tibshirani, 2013; Kuhn & Johnson, 2013). Beide Methoden sind insofern Resampling-Verfahren (vgl. Kap. 11), als sie aus Daten einer gegebenen Basisstichprobe mehrfach neue Stichproben erstellen und mit ihnen den Vorhersagefehler schätzen.

13.1 Kreuzvalidierung linearer Regressionsmodelle

Für die Kreuzvalidierung ist eine vorliegende Gesamtstichprobe vom Umfang n in zwei komplementäre Teilmengen zu partitionieren: Die Trainingsstichprobe liefert die Datenbasis für die Parameterschätzung. Diese Modellanpassung wird dann auf die verbleibende Teststichprobe angewendet, indem die Werte der Kovariaten X' aus der Teststichprobe in die ermittelte Vorhersagegleichung eingesetzt werden. Als Verlustfunktion $f(E)$ dient etwa die mittlere Fehlerquadratsumme, im Falle einer linearen Regression der Standardschätzfehler.

13.1.1 k-fache Kreuzvalidierung

Für eine bessere Beurteilung der Vorhersagegenauigkeit sollte die dargestellte Prüfung mehrfach mit wechselnden Trainings- und Teststichproben durchgeführt werden. Häufig wird hierzu die Ausgangsstichprobe in k disjunkte Teilmengen partitioniert. Jede von ihnen dient daraufhin reihum als Teststichprobe, während die jeweils verbleibenden $k-1$ Stichproben gemeinsam die Trainingsstichprobe bilden. Der Mittelwert der Verlustfunktion über die k Wiederholungen dient dazu, die Generalisierbarkeit der Parameterschätzung insgesamt zu beurteilen.[1] Häufig empfohlene Werte für k sind 5 oder 10, für höhere Werte ist der steigende numerische Aufwand zur Kreuzvalidierung sehr umfassender Modelle zu berücksichtigen. Bei $k = 2$ mit zwei gleich großen Teilstichproben handelt es sich um die doppelte Kreuzvalidierung.[2]

Das Paket `boot` (vgl. Abschn. 11.1) stellt mit `cv.glm()` eine Funktion für die k-fache Kreuzvalidierung verallgemeinerter linearer Modelle (vgl. Kap. 8) bereit.

```
> cv.glm(data=⟨Datensatz⟩, glmfit=⟨glm-Modell⟩, K=⟨Anzahl⟩,
+        cost=⟨Verlustfunktion⟩)
```

Für `data` ist der Datensatz zu übergeben, aus dem die Variablen eines verallgemeinerten linearen Modells stammen, das unter `glmfit` zu nennen ist. Da die lineare Regression ein Spezialfall eines solchen Modells ist, kann `cv.glm()` auch für sie zum Einsatz kommen. Die Regression ist hierfür lediglich mit `glm()` zu formulieren (vgl. Abschn. 8.1.6). Mit `K` lässt sich die gewünschte Anzahl zufälliger Partitionen für die Aufteilung in Trainings- und Teststichproben wählen. In der Voreinstellung ist die Verlustfunktion die mittlere Fehlerquadratsumme, kann aber über `cost` auf eine selbst definierte Funktion geändert werden (vgl. Abschn. 13.2).

Das Ergebnis von `cv.glm()` ist eine Liste, die in der Komponente `delta` den Kreuzvalidierungsfehler (CVE, cross validation error) beinhaltet, in der Voreinstellung also den Mittelwert der k mittleren Fehlerquadratsummen in der Teststichprobe. Das zweite Element von `delta` berücksichtigt dabei die Korrektur einer aus der Wahl von k herrührenden Verzerrung.

```
# passe lineare Regression mit glm() an
> regrDf <- data.frame(weight, height, age, sport)      # Datensatz
> glmFit <- glm(weight ~ height + age + sport, data=regrDf,
+               family=gaussian(link="identity"))

> library(boot)                                        # für cv.glm()
> k    <- 3                                            # Anzahl Partitionen
> kfCV <- cv.glm(regrDf, glmfit=glmFit, K=k)           # k-fache Kreuzvalid.
> kfCV$delta                        # Kreuzvalidierungsfehler und Korrektur
10.52608  10.38042
```

[1] Mit Stichproben der Größe n_k können die Einzelwerte der Verlustfunktion auch gewichtet mit dem Faktor $\frac{n_k}{n}$ in den Mittelwert eingehen. Die k-fache Kreuzvalidierung eines linearen Regressionsmodells ist für $k = n\left(1 - \frac{1}{\ln(n)-1}\right)$ asymptotisch äquivalent zum Informationskriterium BIC (vgl. Abschn. 6.3.3).

[2] Die *stratifizierte* k-fache Kreuzvalidierung sorgt bei der Einteilung in Teilmengen dafür, dass die Verteilung von Y in den Partitionen ähnlich ist. Für kontinuierliche Y führt dies zu einem annähernd konstanten Mittelwert von Y in den Partitionen. Für kategoriale Y bleibt so der Anteil der Kategorien weitgehend gleich. Für eine Umsetzung vgl. die Pakete `caret` (Kuhn, 2014) oder `ipred` (Peters & Hothorn, 2013).

Für die manuelle Berechnung ist zunächst eine eigene Funktion zu erstellen, die auf Basis des logischen Indexvektors einer Teststichprobe eine einzelne Kreuzvalidierung des gegebenen Regressionsmodells durchführt (vgl. Abschn. 16.2).

```
# einzelne Kreuzvalidierung gegeben logische Indizes der Teststichprobe
> doCV <- function(idxTst) {
+     # passe Regression in Trainingsstichprobe an
+     fitTrn <- lm(weight ~ height + age + sport, regrDf, subset=!idxTst)
+
+     # berechne für Teststichprobe Vorhersage aus angepasstem Modell
+     predTst <- predict(fitTrn, regrDf[idxTst, ])
+
+     # mittlere Fehlerquadratsumme für Teststichprobe
+     mean((predTst - weight[idxTst])^2)
+ }
```

Daraufhin muss die gesamte Stichprobe vom Umfang n in k zufällige Gruppen partitioniert werden, die dann der Reihe nach als Teststichprobe dienen.

```
> tstGrp <- (sample(1:N, N, replace=FALSE) %% k) + 1      # Gruppen
> (tst1  <- doCV(tstGrp == 1))    # Kreuzvalidierung 1. Teststichprobe
[1] 11.96789

> (tst2 <- doCV(tstGrp == 2))     # Kreuzvalidierung 2. Teststichprobe
[1] 10.45271

> (tst3 <- doCV(tstGrp == 3))     # Kreuzvalidierung 3. Teststichprobe
[1] 11.85106

>  mean(c(tst1, tst2, tst3))      # Kreuzvalidierungsfehler
[1] 11.42389
```

13.1.2 Leave-One-Out Kreuzvalidierung

Ein Spezialfall der k-fachen Kreuzvalidierung ist die *Leave-One-Out*-Kreuzvalidierung (LOOCV) für $k = n$.[3] Hier dient nacheinander jede Einzelbeobachtung separat als Teststichprobe für ein Modell, das an der Trainingsstichprobe aus jeweils allen übrigen Beobachtungen angepasst wurde. Im Fall der linearen Regression lässt sich der Kreuzvalidierungsfehler hier numerisch effizient als Mittelwert der *PRESS*-Residuen $\frac{1}{n}\sum_i \left(\frac{e_i}{1-h_i}\right)^2$ (*predicted residual error sum of squares*) direkt berechnen. Dabei ist e_i das i-te Residuum $y_i - \hat{y}_i$ und h_i der Hebelwert der Beobachtung i (vgl. Abschn. 6.5.1).

```
# LOOCV-Kreuzvalidierungsfehler inkl. korrigierter Variante
> LOOCV <- cv.glm(data=regrDf, glmfit=glmFit, K=N)
> LOOCV$delta
```

[3] Sie ist asymptotisch äquivalent zum Informationskriterium AIC des Regressionsmodells (vgl. Abschn. 6.3.3).

```
       1         1
10.59404  10.58995

# Kreuzvalidierungsfehler hier = Mittelwert PRESS-Residuen
> lmFit <- lm(weight ~ height + age + sport)
> PRESS <- (residuals(lmFit) / (1 - hatvalues(lmFit)))^2
> mean(PRESS)
[1] 10.59404
```

Bei der Kontrolle wird die zuvor selbst erstellte Funktion `doCV()` mittels `sapply()` für jeden Index der Stichprobe aufgerufen.

```
> idx <- seq(length.out=N)        # Indizes aller Beobachtungen
> res <- sapply(idx, function(x) { doCV(idx == x) } )     # LOOCV
> mean(res)                       # Kreuzvalidierungsfehler
[1] 10.59404
```

Für penalisierte Regressionsmodelle und verallgemeinerte additive Modelle (vgl. Abschn. 6.6) ist die Berechnung der Hat-Matrix $\boldsymbol{H}$ – und damit der Diagonaleinträge h_i – aufwendig. In diesen Fällen lässt sich der Kreuzvalidierungsfehler über die verallgemeinerte Kreuzvalidierung (GCV) mit $\frac{1}{n}\sum_i \left(\frac{e_i}{1-\text{Spur}(H)/n}\right)^2$ approximieren. Hier ist Spur(H) die Anzahl zu schätzender Parameter.

13.2 Kreuzvalidierung verallgemeinerter linearer Modelle

Bei der Kreuzvalidierung verallgemeinerter linearer Modelle (vgl. Kap. 8) ist zu beachten, dass die Wahl der Verlustfunktion als Gütemaß der Vorhersagegenauigkeit für diskrete Variablen Besonderheiten mit sich bringt. Im Fall der logistischen Regression (vgl. Abschn. 8.1) wäre es etwa naheliegend, die Vorhersagegüte als Anteil der korrekt vorhergesagten Treffer umzusetzen. Tatsächlich ist dieses Maß weniger gut geeignet, da mit ihm nicht das wahre statistische Modell den kleinsten Vorhersagefehler besitzt. Verlustfunktionen, die durch das wahre Modell minimiert werden, heißen *proper score*.

Das Argument `cost` von `cv.glm()` für selbst definierte Verlustfunktionen erwartet eine Funktion, die auf Basis je eines Vektors der beobachteten Werte Y und der vorhergesagten Werte $\hat{Y}$ ein Abweichungsmaß zurückgibt. `cv.glm()` verwendet beim Aufruf von `cost()` für Y den Vektor ⟨`glm-Modell`⟩`$y` und für $\hat{Y}$ `predict(`⟨`glm-Modell`⟩`, type="response")`.

Für die logistische Regression ist der Brier-Score B eine geeignete Verlustfunktion. Für jede Beobachtung i berücksichtigt b_i die vorhergesagte Wahrscheinlichkeit $\hat{p}_{ij} = \hat{p}(y_i = j)$ jeder Kategorie j beim Vergleich mit der tatsächlich vorliegenden Kategorie y_i. Hier soll $\hat{p}_i$ kurz für die vorhergesagte Trefferwahrscheinlichkeit $p(y_i = 1)$ stehen, und es gilt $1 - p_i = p(y_i = 0)$.

$$b_i = \begin{cases} (1-\hat{p}_i)^2 + (0-(1-\hat{p}_i))^2 & \text{falls } y_i = 1 \\ (1-(1-\hat{p}_i))^2 + (0-\hat{p}_i)^2 & \text{falls } y_i = 0 \end{cases}$$

Der Brier-Score für eine Stichprobe vom Umfang n ist dann $B = \frac{1}{n}\sum_i b_i$. Mit $k = 2$ Kategorien und dem Kronecker δ als Indikatorfunktion lässt sich b_i auch verkürzt schreiben, wobei $\delta_{ij} = 1$ ist, wenn $y_i = j$ gilt und $\delta_{ij} = 0$ sonst.

$$b_i = \sum_{j=1}^{k} (\delta_{ij} - \hat{p}_{ij})^2$$

Als Beispiel seien die Daten aus Abschnitt 8.1 zur logistischen Regression betrachtet.

```
> glmLR <- glm(postFac ~ DVpre, family=binomial(link="logit"),
+              data=dfAncova)

# Verlustfunktion für Brier-Score
> brierA <- function(y, pHat) {
+     mean(((y == 1) * pHat)^2 + ((y == 0) * (1-pHat))^2)
+ }

> library(boot)                            # für cv.glm()
> B1 <- cv.glm(data=regDf, glmfit=glmFit, cost=brierA, K=10)
> B1$delta
[1] 0.5528585 0.5523104
```

Die genannte Definition des Brier-Score lässt sich unmittelbar auf Situationen mit $k > 2$ Kategorien verallgemeinern, etwa auf die multinomiale Regression oder die Diskriminanzanalyse, wo er ebenfalls ein proper score ist. Speziell für die logistische Regression führt auch die folgende vereinfachte Variante zu einem proper score, sie berücksichtigt nur die tatsächlich beobachtete Kategorie. Diese Verlustfunktion $b_i = (y_i - \hat{p}_i)^2$ entspricht der Voreinstellung von `cost` in `cv.glm()`.

```
# Verlustfunktion für Brier-Score - vereinfachte Variante
> brierB <- function(y, pHat) {
+     mean((y-pHat)^2)
+ }

> B2 <- cv.glm(data=regDf, glmfit=glmFit, cost=brierB, K=10)
> B2$delta
[1] 0.1787110 0.1773381
```

Eine Alternative zum Brier-Score basiert auf der logarithmierten geschätzten Wahrscheinlichkeit für die tatsächlich beobachtete Kategorie j von Y. Die Verlustfunktion $-2 \cdot \sum_i \ln \hat{p}_{ij}$ ist äquivalent zu Devianz-Residuen und führt ebenfalls zu einem proper score, der sich auch auf andere verallgemeinerte lineare Modelle anwenden lässt.

13.3 Bootstrap-Vorhersagefehler

Bootstrapping (vgl. Abschn. 11.1) liefert weitere Möglichkeiten zur unverzerrten Schätzung des Vorhersagefehlers prädiktiver Modelle, von denen hier die einfache Optimismus-Korrektur

vorgestellt wird. Für weitere Ansätze wie den .632 bzw. .632+ bootstrap vgl. Hastie et al. (2009) und für eine Implementierung die Pakete `caret` oder `ipred`.

Für die Optimismus-Berechnung wird die (negative) systematische Verzerrung des Trainingsfehlers über viele Replikationen der Basisstichprobe hinweg geschätzt und danach als Bias-Korrektur vom ursprünglichen Trainingsfehler subtrahiert. Zur Erläuterung der notwendigen Schritte sei folgende Notation vereinbart:

- $\mathbb{E}[Y - \hat{Y}_{X_0,Y_0}(X)]$ ist der wahre Vorhersagefehler eines an den Beobachtungen X_0, Y_0 der Basisstichprobe angepassten Modells für eine neue Stichprobe mit Beobachtungen X, Y von Personen aus derselben Grundgesamtheit.
- $\overline{E}[Y_0 - \hat{Y}_{X_0,Y_0}(X_0)]$ ist der mittlere Trainingsfehler eines an den Beobachtungen X_0, Y_0 der Basisstichprobe angepassten Modells. $\overline{E}[Y_0 - \hat{Y}_{X_0,Y_0}(X_0)]$ unterschätzt $\mathbb{E}[Y - \hat{Y}_{X_0,Y_0}(X)]$ systematisch, die Differenz wird als *Optimismus* des Trainingsfehlers bezeichnet.
- $\overline{E}[Y_0 - \hat{Y}_{X^\star,Y^\star}(X_0)]$ ist der mittlere Vorhersagefehler der Bootstrap-Replikationen, deren jeweilige Vorhersagen für die Basisstichprobe berechnet werden. $\overline{E}[Y_0 - \hat{Y}_{X^\star,Y^\star}(X_0)]$ ist plug-in-Schätzer für $\mathbb{E}[Y - \hat{Y}_{X_0,Y_0}(X)]$.
- $\overline{E}[Y^\star - \hat{Y}_{X^\star,Y^\star}(X^\star)]$ ist der mittlere Trainingsfehler der Bootstrap-Replikationen und plug-in-Schätzer für $\overline{E}[Y_0 - \hat{Y}_{X_0,Y_0}(X_0)]$.

Da bootstrapping die Beziehung zwischen Basisstichprobe und Population auf die Beziehung zwischen Replikation und Basisstichprobe überträgt, erfolgt die Bootstrap-Schätzung des wahren Trainingsfehler-Optimismus durch die Differenz der jeweiligen plug-in Schätzer. Die bias-korrigierte Schätzung des Vorhersagefehlers $\hat{\mathbb{E}}$ ist dann die Differenz vom Trainingsfehler der Basisstichprobe und dem (negativen) geschätzten Optimismus.

$$\hat{\mathbb{E}}[Y - \hat{Y}_{X_0,Y_0}(X)] = \overline{E}[Y_0 - \hat{Y}_{X_0,Y_0}(X_0)] - (\overline{E}[Y^\star - \hat{Y}_{X^\star,Y^\star}(X^\star)] - \overline{E}[Y_0 - \hat{Y}_{X^\star,Y^\star}(X_0)])$$

Der Stichprobenumfang im hier verwendeten Beispiel einer logistischen Regression (vgl. Abschn. 8.1) ist gering. In den Replikationen können deshalb auch solche Zusammensetzungen der Beobachtungen auftreten, die zu einer vollständigen Separierbarkeit führen und ungültige Parameterschätzungen liefern (vgl. Abschn. 8.1.7). Mit den in Abschn. 16.2.3 und 16.3.2 vorgestellten Methoden kann diese Situation innerhalb der von `boot()` für jede Replikation aufgerufenen Funktion identifiziert werden, um dann einen fehlenden Wert anstatt der ungültigen Optimismums-Schätzung zurückzugeben. Als Verlustfunktion soll der vereinfachte Brier-Score für die logistische Regression (s. o.) dienen.

```
# Funktion, die Optimismus in einer Replikation schätzt
> getBSB <- function(dat, idx) {
+     op <- options(warn=2)  # mache Warnungen zu Fehlern
+     on.exit(par(op))       # bei Verlassen: Option zurücksetzen
+
+     # logistische Regression, bei der Fehler abgefangen werden
+     bsFit <- try(glm(postFac ~ DVpre, family=binomial(link="logit"),
+                      subset=idx, data=dat))
+
+     fail <- inherits(bsFit, "try-error")   # Fehler in glm()?
```

```
+     if(fail || !bsFit$converged) { # Fehler | keine Konvergenz
+         return(NA)
+     } else {                                # alles ok
+         # Trainingsfehler Replikation
+         BbsTrn <- brierB(bsFit$y, predict(bsFit, type="response"))
+
+         # Vorhersagefehler Replikation bzgl. Basisstichprobe
+         BbsTst <- brierB(as.numeric(dat$postFac)-1,
+                          predict(bsFit, newdata=dat, type="response"))

+         return(BbsTrn - BbsTst)             # Optimismus Replikation
+     }
+ }
```

Im Anschluss an das bootstrapping sollte geprüft werden, wie oft Konvergenz- oder Separierbarkeitsprobleme aufgetreten sind.

```
> library(boot)                             # für boot()
> nR    <- 999                              # Anzahl Replikationen
> bsRes <- boot(dfAncova, statistic=getBSB, R=nR)
> sum(is.na(bsRes$t))             # Anzahl problematischer Anpassungen
[1] 5

# Traingsfehler Basisstichprobe
> (Btrain <- brierB(glmLR$y, predict(glmLR, type="response")))
[1] 0.1494605

> (optimism <- mean(bsRes$t, na.rm=TRUE))     # mittlerer Optimismus
[1] -0.02267715

# Bootstrap-Schätzung Optimismus-korrigierter Vorhersagefehler
> (predErr <- Btrain - optimism)
[1] 0.1721376
```