4 Logistische Regression in R

Die logistische Regression ist wie in Abschn. 3.2 gesehen ein Spezialfall eines verallgemeinerten linearen Modells (generalized linear model, kurz GLM). In R wird daher zum Anpassen eines logistischen Regressionsmodells an Daten die Funktion `glm` verwendet. Bevor wir uns die Details dieser Funktion anschauen, beginnen wir mit einem Datenbeispiel.

Damit alle Beispiele auch selber durchgerechnet werden können, werden die Datensätze jeweils von folgender Webseite heruntergeladen:

```
book.url <- "https://stat.ethz.ch/~meier/teaching/book-logreg"
```

Beispiel: Spende

In einer Umfrage wurden 1000 Personen befragt, ob sie bereit sind, eine Spende für einen bestimmten wohltätigen Zweck zu machen. Zudem wurde das Alter der Personen erhoben. Gibt es einen Zusammenhang zwischen der Spendebereitschaft und dem Alter? Die Daten sind im data frame `spende` zu finden. Er enthält die Spalte `alter` für das erhobene Alter (numerische Variable) und die Spalte `antwort` für die Spendebereitschaft: eine Faktorvariable mit den beiden Levels `"nein"` und `"ja"`. Das Referenzlevel ist `"nein"`. Die Daten sind in Abb. 4.1 dargestellt. ◀

Um einen besseren Eindruck zu erhalten, betrachten wir die Struktur und die ersten paar Zeilen des Datensatzes. Mit der Funktion `levels` werden die Faktorstufen eines Faktors angezeigt. Die zuerst genannte Faktorstufe ist das Referenzlevel (mit der Funktion `relevel` könnte die Reihenfolge der Faktorstufen geändert werden).

M. Kalisch und L. Meier, *Logistische Regression,* essentials,
https://doi.org/10.1007/978-3-658-34225-8_4

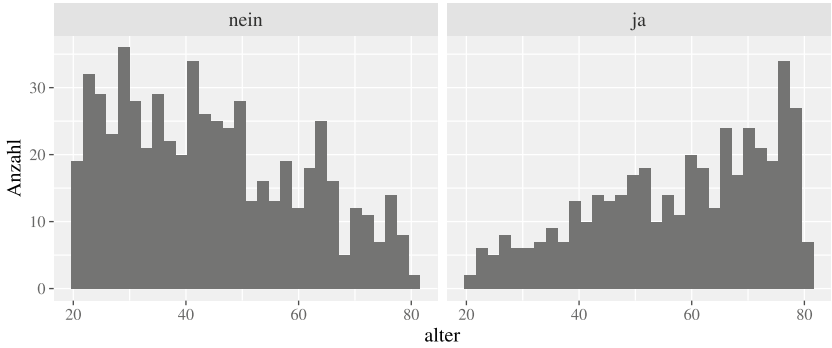

Abb. 4.1 Histogramme der Variable `alter` für Spendebereitschaft „nein“ (links) und Spendebereitschaft „ja“ (rechts)

```
load(url(file.path(book.url, "data/spende.rda")))
str(spende)
## 'data.frame':    1000 obs. of  2 variables:
##  $ alter  : num  35.9 42.3 54.4 74.5 32.1 ...
##  $ antwort: Factor w/ 2 levels "nein","ja": 1 2 1 1 1 ...
head(spende, 4)
##   alter antwort
## 1  35.9    nein
## 2  42.3      ja
## 3  54.4    nein
## 4  74.5    nein
levels(spende$antwort)
## [1] "nein" "ja"
```

4.1 Modell an Daten anpassen

Bei der Funktion `glm` wird das Modell mit einer Formel spezifiziert und die Daten mit dem Argument `data` übergeben. Hinzu kommt das Argument `family`. Mit diesem Argument kann festgelegt werden, welche Verteilung genau verwendet werden soll. Für die logistische Regression müssen wir im Argument `family` die Binomialverteilung angeben (denn: die Bernoulli-Verteilung ist ein Spezialfall einer Binomialverteilung) und als Linkfunktion die Logit-Funktion spezifizieren: `family = binomial(link = "logit")` (da die Logit-Funktion standardmäßig als Linkfunktion ausgewählt wird, würde auch die Kurzform `family = binomial` aus-

reichen). Wichtig ist, dass die Log-Odds für das Level der Zielgröße modelliert werden, das *nicht* das Referenzlevel ist (hier: `antwort = "ja"`, also Spendebereitschaft vorhanden). Mit der Funktion `summary` werden die geschätzten Parameter und weitere Informationen angezeigt:

```
fit.spende <- glm(antwort ~ alter,
                  family = binomial(link = "logit"),
                  data = spende)
summary(fit.spende)
## ...
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.909573   0.236690  -12.29   <2e-16 ***
## alter        0.049963   0.004326   11.55   <2e-16 ***
## ...
```

Die geschätzten Parameter werden im Output unter `Coefficients` angezeigt. Die Zeile `(Intercept)` bezieht sich auf den Parameter β_0 (Achsenabschnitt auf der Skala der Log-Odds) und die Zeile `alter` bezieht sich auf die Steigung bezüglich der Variable `alter` (auf der Skala der Log-Odds), also β_1. Die Spalten enthalten Informationen zum geschätzten Parameter (`Estimate`), zur Genauigkeit der Parameterschätzung, dem sogenannten Standardfehler (`Std. Error`), zum entsprechenden Verhältnis (`z value`) und zum p-Wert für die Nullhypothese (mit zweiseitiger Alternativhypothese), dass der entsprechende Parameter gleich Null ist (`Pr(>|z|)`).

Beispiel: Spende (Fortsetzung) – Geschätztes Modell interpretieren

Das Referenzlevel der Zielgröße `antwort` ist `"nein"` (keine Spendebereitschaft). Also werden die Log-Odds für das *andere* Level, d. h. `"ja"` (Spendebereitschaft) modelliert. Gemäß Output der Funktion `summary` wurde also folgendes zweistufige Modell geschätzt: Die Zielgröße Y ist 1, falls die Person zu einer Spende bereit ist (`antwort="ja"`) und sonst 0 (`antwort="nein"`), d. h. $Y \sim \text{Bernoulli}(p(\text{alter}))$, wobei

$$\log\left(\frac{p(\text{alter})}{1-p(\text{alter})}\right) \approx -2.910 + 0.050 \cdot \text{alter}.$$

Die Wahrscheinlichkeit für Spendebereitschaft in einem gewissen Alter ist also $p(\text{alter})$. Die geschätzten Parameter sind: $\widehat{\beta}_0 \approx -2.910$ und $\widehat{\beta}_1 \approx 0.050$. Gemäß dem geschätzten Modell sind also die Log-Odds für `antwort="ja"` (d. h., Person ist bereit zu Spende) z. B. bei einer Person mit alter $= 50$ ca. $-2.910 + 0.050 \cdot$

$50 = -0.41$, was in etwa einer Wahrscheinlichkeit von 40 % entspricht. Die p-Werte für beide Hypothesentests $H_0 : \beta_0 = 0$ und $H_0 : \beta_1 = 0$ sind sehr klein: `<2e-16` bedeutet, dass die p-Werte kleiner als $2 \cdot 10^{-16}$ sind, also im Bereich der numerischen Genauigkeit des Computers. Die entsprechenden Nullhypothesen können also deutlich verworfen werden. ◄

4.2 Interpretation der Effektstärke

In Abschn. 3.4 haben wir die Faustregel kennengelernt: „Die Interpretation der Modellparameter der logistischen Regression auf der Skala der Log-Odds ist genau gleich wie bei einer linearen Regression". Entsprechend einfach ist die Interpretation der geschätzten Parameter auf der Skala der Log-Odds.

Beispiel: Spende (Forts.) – Effektstärke auf Skala Log-Odds

Wenn das Alter um ein Jahr erhöht wird, erhöhen sich die Log-Odds für `antwort="ja"` um ca. 0.050. Weil sich Log-Odds und Wahrscheinlichkeiten in die gleiche Richtung ändern, heißt das: Die Spendebereitschaft nimmt mit dem Alter zu. ◄

Vertrauensintervalle für die geschätzten Parameter werden mit der Funktion `confint` berechnet. Mit dem Argument `level` wird die Überdeckungswahrscheinlichkeit des Vertrauensintervalls festgelegt.

Beispiel: Spende (Forts.) – Vertrauensintervalle auf Skala Log-Odds

Das jeweilige 95 %-Vertrauensintervall für β_0 und β_1 ist durch die erste bzw. die zweite Zeile des folgenden Outputs gegeben:

```
confint(fit.spende, level = 0.95)
##                    2.5 %      97.5 %
## (Intercept) -3.38249016 -2.45398788
## alter        0.04161265  0.05858367
```

Wir haben gesehen, dass sich die Log-Odds für `antwort="ja"` um ca. 0.050 erhöhen, wenn das Alter um ein Jahr erhöht wird. Ein 95%-Vertrauensintervall für diesen Schätzwert ist (gerundet) [0.042, 0.059]. ◄

Etwas schwieriger wird es, wenn wir die Ergebnisse auf der Skala der Odds oder der Wahrscheinlichkeit interpretieren wollen (siehe die Abschn. 3.4.2 und 3.4.3). Eine wichtige Größe ist dabei das Odds-Ratio: Wenn die erklärende Variable um eine Einheit erhöht wird, dann erhöhen sich die Odds (für Erfolg) um den *Faktor* $\exp(\beta_1)$. Dies entspricht gerade dem Odds-Ratio. Ein Vertrauensintervall für das Odds-Ratio erhalten wir, indem wir ganz einfach die Exponentialfunktion auf die Grenzen des Vertrauensintervalls von β_1 anwenden.

Beispiel: Spende (Fortsetzung) – Odds Ratio

Gemäß Output ist $\widehat{\beta}_1 \approx 0.050$. Das (geschätzte) Odds-Ratio bezüglich Alter ist also

$$\exp(\widehat{\beta}_1) \approx \exp(0.050) \approx 1.05.$$

Dies bedeutet: Wenn das Alter um ein Jahr zunimmt, erhöhen sich die Odds für Spendebereitschaft um den *Faktor* 1.05. Ein 95%-Vertrauensintervall für β_1 ist [0.042, 0.059]. Daher ist ein 95%-Vertrauensintervall für das Odds-Ratio gegeben durch

$$[\exp(0.042), \exp(0.059)] \approx [1.043, 1.061].$$

Einfacher geht es, wenn diese Werte in R direkt berechnet werden:

```
exp(confint(fit.spende, level = 0.95))
##                  2.5 %     97.5 %
## (Intercept) 0.03396278 0.08595014
## alter       1.04249059 1.06033370
```

Die Zeile `alter` entspricht (diesmal mit weniger Rundungsfehlern) den manuell berechneten Grenzen des 95%-Vertrauensintervalls für das Odds-Ratio. ◀

In Abschn. 3.4.3 haben wir gesehen, dass sich die Effektstärke auf der Skala der Wahrscheinlichkeit nicht universell quantifizieren lässt.

4.3 Vorhersagen

Während die Effektstärke nur auf der Skala der Log-Odds oder der Odds einfach quantifiziert werden kann, ist eine Vorhersage für einen gegebenen Wert der erklärenden Variable auf jeder Skala möglich.

Beispiel: Spende (Fortsetzung) – Vorhersagen

Wie groß ist gemäß unserem Modell die Spendebereitschaft einer 60-jährigen Person?

- *Skala Log-Odds:* In die Modellgleichung können wir alter $= 60$ einsetzen und erhalten die Log-Odds:

$$\widehat{\beta}_0 + \widehat{\beta}_1 \cdot 60 \approx -2.910 + 0.050 \cdot 60 \approx 0.09.$$

  Die Log-Odds für Spendebereitschaft für eine 60-jährige Person sind gemäß unserem Modell also etwa 0.09.
- *Skala Odds:* Aus den geschätzten Log-Odds berechnen wir die geschätzten Odds für Spendebereitschaft:

$$\exp(0.09) \approx 1.094$$

  Die Odds für Spendebereitschaft sind für eine 60-jährige Person gemäß Modell also etwa 1.09. D. h., die Wahrscheinlichkeit, dass eine 60-jährige Person Spendebereitschaft hat, ist also (gemäß Modell) um den Faktor 1.09 größer als die Wahrscheinlichkeit, dass diese Person keine Spendebereitschaft hat.
- *Skala Wahrscheinlichkeit:* Durch Umformen der Odds berechnen wir die Wahrscheinlichkeit für Spendebereitschaft:

$$\frac{1.094}{1 + 1.094} \approx 0.522$$

  Die Wahrscheinlichkeit, dass eine 60-jährige Person eine Spendebereitschaft hat, ist also etwa 52 %.

◀

Viel einfacher ist es, wenn wir solche Berechnungen mit der Funktion `predict` erledigen. Vorhersagen mit dieser Funktion sind sowohl auf der Skala des linearen Prädiktors, also der Log-Odds (Argument `type = "link"`) sowie auch auf der Skala der Wahrscheinlichkeit (Argument `type = "response"`) möglich. Wir müssen dabei zunächst festlegen, für welche Werte der erklärenden Variablen Vorhersagen gemacht werden sollen. Dazu erstellen wir einen data frame, der als Spalten

alle erklärenden Variablen unseres Modells enthält. Jede Zeile dieses data frames füllen wir anschliessend mit der Kombination der erklärenden Variablen, für die wir eine Vorhersage wünschen. Mehr Details zu dieser Funktion findet man in der Hilfe unter `?predict.glm`.

Beispiel: Spende (Fortsetzung) – Vorhersage mit `predict`

Unser Modell hat nur eine erklärende Variable (`alter`) und wir wünschen eine Vorhersage für nur einen Wert dieser Variable (`alter = 60`). Mit diesen Informationen können wir einen data frame erzeugen:

```
spende.new <- data.frame(alter = 60)
```

Diesen data frame übergeben wir der Funktion `predict` nun im Argument `newdata` und berechnen die vorhergesagten Werte auf der Skala der Log-Odds (`type = "link"`) oder der Wahrscheinlichkeit (`type = "response"`).

```
## Vorhergesagte Log-Odds
(lo.pred <- predict(fit.spende, newdata = spende.new,
                    type = "link"))
##          1
## 0.08819857
## Vorhergesagte Wahrscheinlichkeit
(p.pred <- predict(fit.spende, newdata = spende.new,
                   type = "response"))
##         1
## 0.5220354
```

Abgesehen von Rundungseffekten sind diese Werte und die manuell berechneten Werte identisch. Übrigens ist die Antwortfunktion h schon im Objekt `fit.spende` enthalten, sodass es eine weitere Möglichkeit gibt, die Wahrscheinlichkeit aus den Log-Odds zu berechnen:

```
fit.spende$family$linkinv(lo.pred)
##         1
## 0.5220354
```

◀

Den vorhergesagten Wert auf der Skala der Odds können wir entweder aus der vorhergesagten Wahrscheinlichkeit oder aus den vorhergesagten Log-Odds berechnen.

Beispiel: Spende (Fortsetzung) – Vorhersage der Odds

```
## Vorhergesagte Odds: Variante 1
p.pred / (1 - p.pred)
##        1
## 1.092205
## Vorhergesagte Odds: Variante 2
exp(lo.pred)
##        1
## 1.092205
```

◀

Auf allen drei Skalen können wir auch entsprechende Vertrauensintervalle angeben. Dies funktioniert am einfachsten mit der Funktion `predict` und dem Argument `se.fit = TRUE`. Die „klassischen" Vertrauensintervalle der Form „Schätzwert ± Quantil × Standardfehler" machen aber nur auf der Skala der Log-Odds Sinn (weil es dort keine Restriktion bzgl. der Skala gibt). Für die anderen Skalen (Odds, Wahrscheinlichkeit) werden die Vertrauensintervalle mit den entsprechenden Funktionen mittransformiert:

```
pred.link <- predict(fit.spende, newdata = spende.new,
                     type = "link", se.fit = TRUE)
quant <- 1.96 ## oder: qnorm(0.975) für 95%-Vertrauensintervall
## Skala Log-Odds
(CI.link <- pred.link$fit + c(-1, 1) * quant * pred.link$se.fit)
## [1] -0.06310872  0.23950585
## Skala Odds
exp(CI.link)
## [1] 0.9388414 1.2706211
## Skala Wahrscheinlichkeit
fit.spende$family$linkinv(CI.link)
## [1] 0.4842281 0.5595919
```

Dies bedeutet, dass gemäß Modell die Wahrscheinlichkeit, dass eine 60-jährige Person mit „Ja" antwortet, im Intervall [0.48,0.56] liegt (95%-Vertrauensintervall). Wenn wir dies für verschiedene Alter ausrechnen und einzeichnen, erhalten wir die in Abb. 4.2 dargestellten Vertrauensbänder.

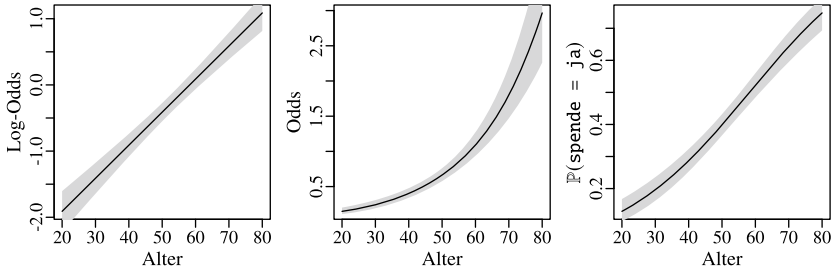

Abb. 4.2 Geschätzte Log-Odds, Odds, und Wahrscheinlichkeiten, inkl. jeweilige punktweise 95%-Vertrauensintervalle (grau)

4.4 Gruppierte Daten

Die bisher verwendeten Daten hatten pro Zeile immer eine Beobachtung mit erklärenden Variablen und der binären Information zur Zielgröße gespeichert (z. B. pro Zeile eine Person mit Alter und Status zu der Spendebereitschaft). Häufig liegen die Daten in einer anderen Form vor: Vor allem bei kontrollierten Experimenten gibt es oft *Gruppen* von (unabhängigen) Versuchseinheiten, die alle die gleichen erklärenden Variablen haben. Statt eine *einzelne* Beobachtung pro Zeile zu speichern, wird dann pro Zeile eine ganze Gruppe mit den erklärenden Variablen der Gruppe und der Anzahl der Erfolge bzw. der Misserfolge pro Gruppe gespeichert.

Beispiel: Daten pro Gruppe

Es werden 11 Gruppen mit je 30 (unabhängigen) kranken Tieren mit einem neuen Medikament behandelt. Innerhalb einer Gruppe erhält jedes Tier die gleiche Dosis, allerdings ist die Dosis für jede Gruppe anders. Wir könnten die Daten wie bisher speichern: Pro Zeile ein Tier mit der zugehörigen Dosis und dem Ausgang des Experiments („krank“ oder „gesund“). Dieser Datensatz hätte 330 Zeilen. Alternativ können wir pro Zeile nur die Informationen einer *Gruppe* (und nicht einer Einzelbeobachtung) speichern. Die Spalte `gesund` des data frames `medikament` enthält die Anzahl der Tiere, die pro Gruppe gesund geworden sind (entsprechend die Spalte `krank`). Die Spalte `Dosis` enthält die Dosis, die jedem der 30 Tiere in einer Gruppe verabreicht wurde. Diese Art der Darstellung hilft uns, die Daten kompakter darzustellen: Wir brauchen nur 11 Zeilen und nicht 330.

```
load(url(file.path(book.url, "data/medikament.rda")))
head(medikament)
##   dosis krank gesund
## 1     0    29      1
## 2     1    29      1
## 3     2    27      3
## 4     3    27      3
## 5     4    21      9
## 6     5    24      6
```

Zum Beispiel wurden von den 30 Tieren in der Gruppe mit `dosis = 5` genau 6 Tiere gesund. ◀

Um die logistische Regression wie gewohnt auf gruppierte Daten anwenden zu können, könnten wir den gruppierten Datensatz mit etwas Programmieraufwand in einen Datensatz mit einer Einzelbeobachtung pro Zeile umwandeln.

Es geht aber auch einfacher: Der gruppierte Datensatz kann direkt in `glm` verwendet werden. Dabei muss lediglich das erste Argument (`formula`) angepasst werden. Anstelle der Zielgröße übergeben wir eine Matrix mit zwei Spalten. Die erste Spalte enthält pro Gruppe die Anzahl der Erfolge (z. B. geheilte Tiere). Die zweite Spalte enthält pro Zeile die Anzahl der Misserfolge (z. B. Tiere, die krank geblieben sind). Hilfreich ist dabei der Befehl `cbind`, mit dem Vektoren spaltenweise in eine Matrix zusammengefasst werden.

Beispiel: Daten pro Gruppe (Fortsetzung)

Zusammengefasst in einer Matrix können wir nun sofort `glm` aufrufen und erhalten den gewohnten Output mit der Funktion `summary`:

```
fit.medi <- glm(cbind(gesund, krank) ~ dosis,
                family = binomial(link = "logit"),
                data = medikament)
summary(fit.medi)
## ...
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -4.29356    0.46067  -9.320   <2e-16 ***
## dosis        0.74103    0.07601   9.749   <2e-16 ***
## ...
```

◀