Logistische Regression

Xuan Son Le (4669361), Freie Universität Berlin

02/04/2018

Abstract: Im Rahmen der Abschlussarbeit des Moduls Programmieren mit R im Wintersemester 2017/2018 an der Freie Universität Berlin wird für diese Arbeit die statistische Methode namens binäres Logit-Modell ausgewählt. Diese Arbeit besteht aus zwei großen Hauptteilen: der Theorieteil, wobei die ausgewählte Methode theoretisch vorgestellt wird und der Implementierungsteil, welcher die Erklärung der Funktionalität vom selbst entwickelten Paket beinhaltet. Im Theorieteil wird zunächst die Grundidee von Generalisierten Linearen Modellen (GLMs) widergegeben. Anschließend werden die grundliegende Funktionsweise vom (binären) Logit-Modell sowie dessen Aufbau durch das Maximum Likelihood Verfahren vorgenommen. Demzufolge folgt die Interpretation der Koeffizienten vom binären Logit-Modell. Schließlich werden im Implementierungsteil alle Funktionen vom R-Paket schritterweise vorgestellt.

Keywords: *Logit-Modell, logistische Regression, R-Paket, Maximum Likelihood*

---

1 Motivation

Die Anwendung von der klassischen linearen Regression ist für binäre (binomiale oder dichotome) Zielvariable (Response- oder zu erklärende Variable), welche lediglich zwei Werte (ja/nein, männlich/weiblich, erfolgreich/nicht erfolgreich, etc.) annehmen kann, nicht mehr geeignet, da die Zielvariable von der linearen Regression metrisch skaliert ist. Oft wird binäre Variable als 0/1-Variable kodiert, das heißt sie nimmt nur den Wert 0 oder 1 an. Abbildung 1 stellt den Ansatz graphisch dar, binäre Variable durch lineare Regression zu modellieren:

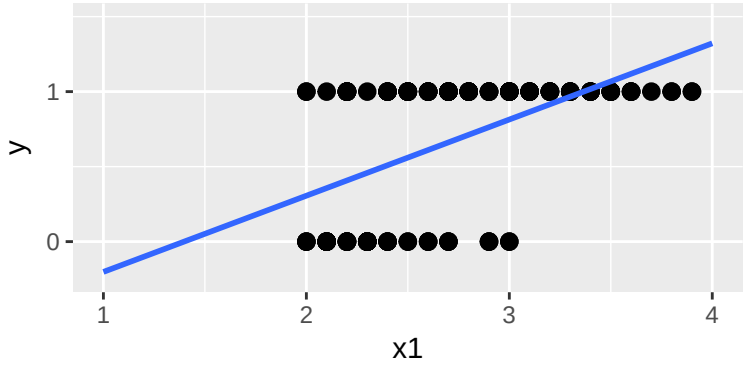

Abbildung 1: Beispielshafte lineare Regression für binäre Zielvariable

Graphisch lässt sich festlegen, dass die lineare Regression den Wertebereich [0,1] von binären Responsevariablen sehr schnell verlässt. Aus diesem Grund wird ein ganz anderer Ansatz benötigt, um binäre Zielvariable zu modellieren, nämlich das binäre Logit-Modell (auch binäre logistische Regression oder binäres logistisches Regressionsmodell). In der Statistik lassen sich Logit-Modelle noch in multinomiale und kumulative Logit-Modelle aufteilen, je nachdem ob die abhängige Variable multinominal- oder ordinalskaliert sind (vgl. Schlittgen (2013), S.225 ff.). Diese Arbeit beschäftigt sich mit dem binären Logit-Modell, welches den Zusammenhang zwischen einer binären abhängigen Variable und einer/mehreren unabhängigen Variablen untersucht. Bei allen Arten von Logit-Modellen können die unabhängigen Variablen (erklärende oder Kovariablen) beliebig skaliert sein.

Im Unterschied zu der klassischen linearen Regression, welche den wahren Wert einer Zielvariable vorhersagt, interessiert sich das binäre Logit-Modell eher für die Wahrscheinlichkeit, dass die Zielvariable den Wert 1 annimmt. Das Hauptziel vom binären Logit-Modell ist es, die Wahrscheinlichkeit für den Eintritt der Zielvariable vorherzusagen. Dadurch soll die folgende theoretische Fragestellung beantwortet werden: *Wie stark ist der Einfluss von den unabhängigen (erklärenden) Variablen auf die Wahrscheinlichkeit, dass die abhängige (zu erklärende / Response) Variable eintritt beziehungsweise den Wert 1 annimmt?* In der Praxis kann diese Fragestellung beispielsweise so formuliert werden: “Haben Alter, Geschlecht, Berufe oder andere Merkmale der Kunden Einfluss auf die Wahrscheinlichkeit, dass sie ein Kredit rechtzeitig zurückzahlen?” oder “Lässt sich die Wahrscheinlichkeit, dass es regnet, durch die Temparatur, die Windstärke oder Sonnenstrahlungsintensität vorhersagen?”.

2 Das binäre Logit-Modell

Das Logit-Modell ist eine Methode aus der Algorithmenklasse namens *Generalisierte Lineare Modelle* (engl. generalized linear model, kurz GLM), welche eine Verallgemeinerung des klassischen linearen Regressionsmodells anstrebt. Dazu gehören noch die klassische lineare Regression, Probitmodell und Poisson-Regression. Der grundliegende Ansatz von GLM ist die Transformation der linearen Regressionsgleichung, so dass der Wertebereich der vorhergesagten Zielvariable dem gewünschten entspricht. Die Theorie von GLMs wurde von Nelder und Wedderburn entwickelt. In Anlehnung an Schlittgen (2013), S.238 können Modelle zu GLMs zugeordnet werden, wenn:

1. der bedingte Erwartungswert von der abhängigen Variable y_i, $\mu_i = E(y_i|x_i)$, kann über eine Linkfunktion $\mathbf{g}(\mu_i)$ in eine lineare Kombination der unabhängigen Variablen transformiert werden:
$$\mathbf{g}(\mu_i) = \eta_i = x_i.\beta$$
Wird die Umkehrfunktion gebildet, ergibt sich die Responsefunktion, welche die Abhängigkeit der Erwartungswerte von einer Funktion des linearen Prädikators darstellt:
$$\mu_i = \mathbf{g}^{-1}(\eta_i) = \mathbf{h}(\eta_i) = \mathbf{h}(\mathbf{g}(\mu_i))$$
2. Die abhängige Variable gehört zu einer Exponentialfamilie mit der Dichtefunktion:
$$f(y_i;\theta_i) = \exp\left(\frac{y_i.\theta_i + b(\theta_i)}{a_i(\phi)} + c(y,\phi)\right)$$

Die mathematische Erklärung von GLMs wird hierbei vernachlässigt, denn diese Arbeit konzentriert sich lediglich auf dem binären Logit-Modell, welches in Folgenden vorgestellt werden.

2.1 Modellspezifikation

Die folgende Modellspezifikation vom binären Logit-Modell basiert sich auf Kapitel 9.1 aus dem Buch *Regressionsanalysen mit R* (Schlittgen, 2013, S.215-225) und Kapitel 4.1 der Vorlesungsfolien vom Modul *Statistische Modellierung* im Wintersemester 2016/2017 an der Freie Universität Berlin.

Gegeben seien n unabhängige Beobachtungen $y_1, y_2, ..., y_n$ der binären Zielvariable $\mathbf{Y}$. Ein Verteilungsmodell für $\mathbf{Y}$ ist die Binomialverteilung: $\mathbf{Y}_i \sim B(1, \pi_i)$ mit $\pi_i = P(Y_i = 1)$. Für diese Arbeit wird $\pi_i = (\pi_1, \pi_2, ..., \pi_n)$ als die Eintrittwahrscheinlichkeit von der einzelnen $\mathbf{Y}_i$ benannt. Weiterhin seien p erklärende Variablen $\mathbf{X}_0, \mathbf{X}_1, .., \mathbf{X}_p$ gegeben mit jeweils n unabhängigen Beobachtungen $\mathbf{X}_j = (x_{1j}, x_{2j}, ..., x_{nj})$ mit j $\in$ {0,1,2,..,p} - gegeben. Daraus ergeben sich p Koeffizienten $\beta = (\beta_0, \beta_1, \beta_2, ..., \beta_p)$, welche die Stärke den Zusammenhang zwischen die einzelne erklärende Variable mit der Zielvariable widerspiegeln. Dabei ist es sinnvoll, diese in einer Designmatrix $\mathbf{X}$ zu speichern. Da der Interzept (β_0) ebenfalls geschätzt

werden soll, sind alle Werte der ersten Spalte von X gleich Eins, also $x_{10} = x_{20} = ... = x_{n0} = 1$. Zusammengefasst lässt sich die Designmatrix wie folgt darstellen:

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1p} \\ 1 & x_{21} & x_{22} & \cdots & x_{2p} \\ 1 & x_{31} & x_{32} & \cdots & x_{3p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix}$$

Die dazugehörige lineare Regressionsgleichung lautet: $\mathbf{Y} = \mathbf{X}.\beta + \epsilon$ mit $\epsilon = (\epsilon_1, \epsilon_2, \epsilon_3, ..., \epsilon_n)$ als die Abweichung der einzelnen Schätzungen gegenüber dem wahren Wert, wobei $\mathbf{Y}$ ein (nx1)-Vektor, $\mathbf{X}$ ein (nxp) und β_i sowie ϵ_i ein (px1)-Vektor ist.

Die einzelne Beobachtung lässt sich wie folgt darstellen:

$$y_i = \beta_0 + \beta_1.x_{i1} + \beta_2.x_{i2} + ... + \beta_p.x_{ip} + \epsilon_i = \mathbf{x'}_i.\beta + \epsilon_i \qquad \forall_i = 1, 2, 3, ..., n$$

Wobei $\mathbf{x}_i$ der i-ten Zeile der Designmatrix $\mathbf{X}$ entspricht. Da bei der Multiplikation A.B die Regel gilt, dass die Spaltenanzahl von A der Zeilenanzahl von B entsprechen muss. Da β ein (px1)-Vektor ist, muss $\mathbf{x}_i$ (px1-Vektor) in $\mathbf{x'}_i$ (1xp-Vektor) transponiert werden, damit die Multiplikation durchführbar ist.

Um die Werte im Bereich der reellen Zahlen von der linearen Regression auf dem Wertebereich von Wahrscheinlichkeiten zwischen 0 und 1 zu beschränken, sollte die rechte Seite der Gleichung transformiert werden. Das Ziel ist es, eine sinnvolle Verteilungsfunktion (Responsefunktion) zu finden, deren Wertebereich in [0,1] liegt: $\pi_i = P(\mathbf{Y_i} = 1) = F(\beta_0 + \beta_1.x_{i2} + \beta_2.x_{i3} + ... + \beta_p.x_{ip}) = F(\eta_i)$. Der lineare Prädikator

$$\eta_i = \beta_0 + \beta_1.x_{i2} + \beta_2.x_{i3} + ... + \beta_p.x_{ip} = \mathbf{x'}_i.\beta \quad (1)$$

wird ebenfalls als Linkfunktion genannt, weil dadurch eine Verbindung (Link) zwischen der Eintrittwahrscheinlichkeit und den unabhängigen Variablen erfolgt wird. Für das binäre Logit-Modell wird anstelle der Responsefunktion die standardisierte logistische Verteilung verwendet:

$$F(\eta_i) = Logist(\eta_i) = \frac{\exp(\eta_i)}{1 + \exp(\eta_i)} \quad (2)$$

Da durch die Responsefunktion die Eintrittwahrscheinlichkeit π_i modelliert werden soll, ergibt sich die Gleichung für das binäre Logit-Modell wie folgt:

$$\pi_i = h(\eta_i) = \frac{\exp(\eta_i)}{1 + \exp(\eta_i)} = \frac{\exp(\mathbf{x'}_i.\beta)}{1 + \exp(\mathbf{x'}_i.\beta)} \quad (3)$$

Dabei kann π_i maximal den Wert 1 nehmen, wenn $\exp(\eta_i)$ sehr groß ist und minimal den Wert 0, wenn $\exp(\eta_i)$ sehr nah rechts von 0 liegt. $\exp(\eta_i)$ kann nicht negativ sein. Diese Gleichung erfüllt somit die Anforderung bezüglich dem Wertebereich von Wahrscheinlichkeiten.

Soll die Gleichung nach dem linearen Prädikator η_i gelöst werden, ergibt sich schließlich die Logit-Linkfunktion:

$$\begin{aligned} \pi_i.(1+\exp(\eta_i)) &= \exp(\eta_i) \\ \Leftrightarrow \pi_i + \pi_i.\exp(\eta_i) &= \exp(\eta_i) \\ \Leftrightarrow \pi_i &= \exp(\eta_i) - \pi_i.\exp(\eta_i) \\ \Leftrightarrow \pi_i &= \exp(\eta_i).(1-\pi_i) \\ \Leftrightarrow \exp(\eta_i) &= \frac{\pi_i}{1-\pi_i} \\ \Leftrightarrow \eta_i &= \ln(\frac{\pi_i}{1-\pi_i}) \quad (4) \end{aligned}$$

Schlittgen (2013), S.236-237 zeigt, dass die Wahrscheinlichkeitsfunktion vom binären Logit-Modell sich mit der Exponentialfunktion schreiben lässt. Somit erfüllt das binäre Logit-Modell alle Anforderungen für ein GLM.

2.2 Maximum Likelihood Schätzung

Ähnlich wie bei der linearen Regression müssen bei dem binären Logit-Modell die unbekannten Parameter β_i (i = 0,1,2,...,k) ebenfalls geschätzt werden. Bei der klassischen linearen Regression wird die Methode der Kleinsten Quadrate (engl. *method of least squares*, kurz KQ-Methode) genutzt, um eine Regressionslinie zu bestimmen, welche die Summe der quadratischen Abweichungen von den beobachteten Punkten minimiert. Da bei dem binären Logit-Modell nicht der wahre Wert der Zielvariable sondern die Eintrittswahrscheinlichkeit geschätzt wird, ist die direkte Abweichung zwischen dem wahren Wert (0/1-Variable) und dem geschätzten Wert (Wahrscheinlichkeit) nicht mehr aussagekräftig wie bei der linearen Regression. Die Koeffizienten müssen anders geschätzt werden. Dementsprechend wird bei dem binären Logit-Modell die sogenannte Maximum Likelihood Schätzung (kurz ML-Schätzung) eingesetzt. Abbildung 2 zeigt ein Beispiel mit zwei mögliche binäre logistiche Regressionskurven, die durch das Maximum Likelihood optimiert werden sollen. Es gibt unendlich viele solche Kurven. Das Ziel ist es, die Kurve mit dem höchsten Maximum Likelihood herauszufinden.

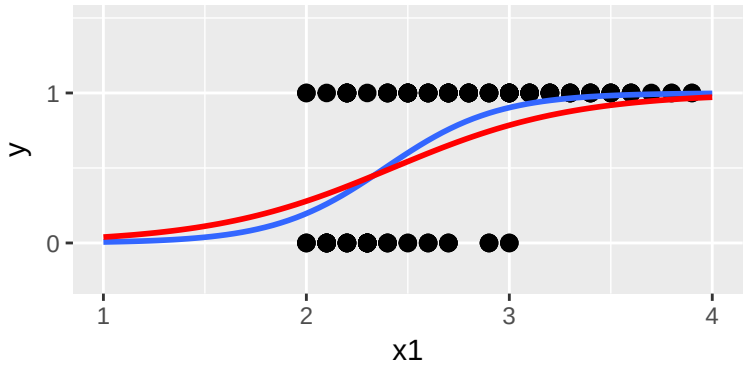

Abbildung 2: Beispielshafte logistische Regressionskurve

Das Ziel der ML-Schätzung besteht darin, die Eintrittswahrscheinlichkeit für die empirischen Beobachtungswerte zu maximieren. Dafür kommt die (Log-)Likelihood-Funktion zum Einsatz. Im Folgenden wird die Vorgehensweise zum Lösen der ML-Schätzung anhand der Log-Likelihood-Funktion wiedergegeben:

1. Maximum Likelihood Funktion
2. Log-Likelihood-Funktion
3. Score-Funktion (erste Ableitung)
4. Hesse Matrix (zweite Ableitung)
5. Newton-Raphson-Methode

2.2.1 Maximum Likelihood Funktion

Gegeben sei $y_i = 1$ mit der Eintrittswahrscheinlichkeit π_i, und $y_i = 0$ mit der Gegenwahrscheinlichkeit $(1 - \pi_i)$. Die Likelihood-Funktion lässt sich wie folgt definieren:

$$\mathcal{L}(\beta) = \prod_{i=1}^{n} \pi_i^{y_i} . (1 - \pi_i)^{1-y_i} \quad (6)$$

Wenn y_i gleich 1 ist, ergibt sich für die betroffene Beobachtung die Eintrittswahrscheinlichkeit π_i und umgekehrt. Das Likelihood ist gleich die Multiplikation der Wahrscheinlichkeiten von allen Beobachtungen. Dieses soll maximiert werden.

Da π_i von dem linearen Prädikator η_i abhängt, ist die Likelihood-Funktion von β abhängig. Wird $\pi_i = \frac{\exp(\eta_i)}{1+\exp(\eta_i)}$ in die Likelihood-Funktion eingesetzt, ergibt sich:

$$\mathcal{L}(\beta) = \prod_{i=1}^{n} \left[\left(\frac{\exp(\eta_i)}{1 + \exp(\eta_i)} \right)^{y_i} . \left(1 - \frac{\exp(\eta_i)}{1 + \exp(\eta_i)} \right)^{1-y_i} \right] \quad (7)$$

2.2.2 Log-Likelihood-Funktion

Der Versuch, Gleichung (7) zu differenzieren und nach β zu lösen, um die Extremwerte zu finden, ist extrem aufwendig, weil sie eine Serie von Multiplikationen enthält. Wegen den exponentialen Komponenten kann die logistische Funktion aus der Mathematik zur Vereinfachung der Likelihood-Funktion Einsatz finden. Da die logistische Funktion eine monotone Funktion ist, entspricht jedes Maximum von der Likelihood-Funktion dem Maximum von der Log-Likelihood-Funktion und umgekehrt. Es gelgen für den Logarithmus folgende Regelungen (seien alle Vorzeichenvoraussetzungen für den Logarithmus erfüllt):

$$(8) \quad \ln(\prod_{i=1}^{n} x_i) = \ln(x_1 . x_2 ... x_n) = \ln(x_1) + \ln(x_2) + \ ... \ + \ln(x_n) = \sum_{i=1}^{n} \ln(x_i)$$

$$(9) \quad \ln(x^{\alpha}) = \alpha . \ln(x)$$

$$(10) \quad \ln(\frac{x}{y}) = \ln(x) - \ln(y)$$

Dementsprechend lässt die Log-Likelihood-Funktion wie folgt logarithmisieren:

$$\begin{aligned}
\ell(\beta) = \ln(\mathcal{L}(\beta)) = &\quad \ln\left(\prod_{i=1}^{n} \pi_i^{y_i}.(1-\pi_i)^{1-y_i}\right) \\
\overset{(8)}{=} &\quad \sum_{i=1}^{n} \ln\left(\pi_i^{y_i}.(1-\pi_i)^{1-y_i}\right) \\
\overset{(9)}{=} &\quad \sum_{i=1}^{n} \Big(y_i.\ln(\pi_i) + (1-y_i).\ln(1-\pi_i)\Big) \\
= &\quad \sum_{i=1}^{n} \Big(y_i.\ln(\pi_i) - y_i.\ln(1-\pi_i) + \ln(1-\pi_i)\Big) \\
\overset{(10)}{=} &\quad \sum_{i=1}^{n} \left(y_i.\ln\left(\frac{\pi_i}{1-\pi_i}\right) + \ln(1-\pi_i)\right) \\
\overset{(4),(3)}{=} &\quad \sum_{i=1}^{n} \left(y_i.\eta_i + \ln\left(1 - \frac{\exp(\eta_i)}{1+\exp(\eta_i)}\right)\right) \\
= &\quad \sum_{i=1}^{n} \left(y_i.\eta_i + \ln\left(\frac{1}{1+\exp(\eta_i)}\right)\right) \\
= &\quad \sum_{i=1}^{n} \Big(y_i.\eta_i - \ln(1+exp(\eta_i))\Big) \quad (11)
\end{aligned}$$

2.2.3 Score-Funktion

Zum Herausfinden des ML-Schätzers, welcher die log-Likelihood-Funktion optimiert, wird Gleichung (10) nach β differenziert. Die erste Ableitung von der log-Likelihood-Funktion wird als Score-Funktion benannt:

$$\begin{aligned}
s(\beta) = \frac{\partial}{\partial\beta}\ell(\beta) &= \frac{\partial}{\partial\beta}\sum_{i=1}^{n}\Big(y_i.\eta_i - \ln(1+\exp(\eta_i))\Big) \\
&\overset{(1)}{=} \frac{\partial}{\partial\beta}\sum_{i=1}^{n}\Big(y_i.\mathbf{x'}_i.\beta - \ln(1+\exp(\mathbf{x'}_i.\beta))\Big)
\end{aligned}$$

Seien alle Vorzeichnenanforderungen erfüllt, gelten folgende Regelungen bezüglich der Differenzierungsrechnung:

$$(12)\frac{\partial}{\partial t}a.f(t) = (a.f(t))' = a.f'(t)$$
$$(13)\frac{\partial}{\partial t}\ln(f(t)) = [\ln(f(t))]' = \frac{f'(t)}{f(t)}$$
$$(14)\frac{\partial}{\partial t}\exp(f(t)) = [\exp(f(t))]' = f'(t).\exp(f(t))$$

Eingesetzt in die Score-Funktion:

$$\begin{aligned} s(\beta) &= \sum_{i=1}^{n} \left(y_i.\mathbf{x'}_i - \frac{\mathbf{x'}_i.\exp(\mathbf{x'}_i.\beta)}{1+\exp(\mathbf{x'}_i.\beta)} \right) \\ &\overset{(1)}{=} \sum_{i=1}^{n} \left[\mathbf{x'}_i \left(y_i - \frac{\exp(\eta_i)}{1+\exp(\eta_i)} \right) \right] \\ &\overset{(2)}{=} \sum_{i=1}^{n} \mathbf{x'}_i(y_i - \pi_i) \\ &= \mathbf{X'}.(y-\pi) \quad (15) \end{aligned}$$

2.2.4 Hesse Matrix

Da $s(\beta)$ wegen exponentiellen Komponenten nicht linear von β abhängt, wird zur Maximierung der Funktion ein iteratives Verfahren verwendet. Für diese Arbeit wird die Newton-Raphson-Methode (vgl. ...) ausgewählt. Dafür muss die zweite Ableitung der Funktion noch gebildet werden, welche als Hesse-Matrix (H) bezeichnet wird:

$$\begin{aligned} \frac{\partial^2}{\partial\beta^2}\ell(\beta) = \frac{\partial}{\partial\beta}s(\beta) = H(\beta) &\overset{(15)}{=} \frac{\partial}{\partial\beta}\sum_{i=1}^{n} (\mathbf{x'}_i.y_i - \mathbf{x'}_i.\pi_i) \\ &= -\sum_{i=1}^{n} \left(\mathbf{x'}_i \,.\left(\frac{\partial}{\partial\beta}\pi_i \right) \right) \\ &\overset{(3)}{=} -\sum_{i=1}^{n} \left(\mathbf{x'}_i \,.\left(\frac{\partial}{\partial\beta}\frac{\exp(\mathbf{x'}_i.\beta)}{1+\exp(\mathbf{x'}_i.\beta)} \right) \right) \end{aligned}$$

Wegen

$$\frac{\partial}{\partial t}\frac{f(t)}{g(t)} = \frac{f'(t).g(t) - g'(t).f(t)}{(g(t))^2}$$

gilt:

$$\begin{aligned}
\frac{\partial}{\partial\beta}s(\beta) = H(\beta) &= -\sum_{i=1}^{n}\left[\mathbf{x'}_i \cdot\left(\frac{\mathbf{x'}_i.\exp(\mathbf{x'}_i.\beta).(1+\exp(\mathbf{x'}_i.\beta)) - \mathbf{x'}_i.\exp(\mathbf{x'}_i.\beta).\exp(\mathbf{x'}_i.\beta)}{(1+\exp(\mathbf{x'}_i.\beta))^2}\right)\right] \\
&= -\sum_{i=1}^{n}\left[\mathbf{x'}_i \cdot\frac{\mathbf{x'}_i.\exp(\mathbf{x'}_i.\beta)}{(1+\exp(\mathbf{x'}_i.\beta))^2}\right] \\
&= -\sum_{i=1}^{n}\left(\mathbf{x'}_i \cdot\frac{\mathbf{x'}_i}{1+\exp(\mathbf{x'}_i.\beta)}.\frac{\exp(\mathbf{x'}_i.\beta)}{1+\exp(\mathbf{x'}_i.\beta)}\right) \\
&\overset{(3)}{=} -\sum_{i=1}^{n}\left(\mathbf{x'}_i.\mathbf{x'}_i.(1-\pi_i).\pi_i\right) \\
&= -\sum_{i=1}^{n}\left((\mathbf{x'}_i)^2.(1-\pi_i).\pi_i\right) \\
&= -\sum_{i=1}^{n}\left(\mathbf{x'}_i.\mathbf{x}_i.(1-\pi_i).\pi_i\right)
\end{aligned}$$

Sei M eine nxn-Matrix mit dem i-ten diagonalen Element $\mathbf{W}_{ii} = \pi_i.(1-\pi_i)$ mit $i \in (1, 2, ..., n)$, ergibt sich:

$$s'(\beta) = H(\beta) = -\sum_{i=1}^{n}\mathbf{x'}_i.\mathbf{x}_i.\mathbf{W}_{ii} = -\mathbf{X'WX} \quad (16)$$

2.2.5 Newton-Raphson-Methode

Die Bezeichungen von der Newton-Raphson-Methode stammt aus den Namen von Isaac Newton und Joseph Raphson. Diese Methode setzt sich zum Ziel, eine nichtlinear lösbare Funktion zu optimieren. Die Grundidee sowie historische Entwicklung von Newton-Raphson-Methode wird durch Ypma (1995) ausführlich vorgestellt.

Sei $\mathbf{f}$ eine Funktion von x, $\mathbf{f'}(x)$ die erste Ableitung und x_0 die Initiallösung. Das Ziel ist es, die Gleichung $\mathbf{f}(x) = 0$ zu lösen. Seien alle Anforderungen der Differentialrechnung erfüllt, kann eine Verbesserung von x_0 wie folgt berechnet werden:

$$x_1 = x_0 - \frac{\mathbf{f}(x_0)}{\mathbf{f'}(x_0)}$$

Wobei eine Verbesserung bedeutet, dass $\mathbf{f}(x_1)$ näher an 0 liegt als $\mathbf{f}(x_0)$. Dieser Prozess wird so oft wiederholt, bis eine akzeptable Lösung anhand der vordefinierten Abbruchskriterien bestimmt wird:

$$x_{n+1} = x_n - \frac{\mathbf{f}(x_n)}{\mathbf{f'}(x_n)} \quad (17)$$

$\mathbf{f}(x)$ repräsentiert für diese Arbeit die Score-Funktion $s(\beta)$, da die Gleichung $s(\beta) = 0$ gelöst

werden soll. Eingesetzt in Gleichung 17 ergibt sich:

$$\begin{aligned} \beta_{n+1} &= \beta_n - \frac{s(\beta)}{s'(\beta)} \\ &\overset{(15),(16)}{=} \beta_n - \frac{\mathbf{X}'.(y-\pi)}{-\mathbf{X}'\mathbf{W}\mathbf{X}} \\ &= \beta_n + (\mathbf{X}'\mathbf{W}\mathbf{X})^{-1}.\mathbf{X}'.(y-\pi) \quad (17) \end{aligned}$$

In Anlehnung an (Quinn, 2001) wird demnächst die Newton-Raphson-Methode anhand einem Pseudocode näher erläutert:

```
1  function(X, y) :
2      initialisiere  β_0, maxIteration, i, tolerance, diff, M
3      while (diff > tolerance)
4          betaChange = (X'WX)^-1.X'.(y − π)
5          diff = |betaChange|
6          i = i + 1
7          if (i > maxIteration) :
8                  break
9  end
```

Es wird eine zulässige Startlösung β_0 initialisiert. Je näher $s(\beta_0)$ an 0 liegt, umso schneller sollte die Laufzeit der Schleife sein. Dieses wird bei der Implementierung berücksichtigt, um herauszufinden, ob es sich lohnt, eine gute Startlösung festzustellen. *tolerance* ist in der Regel sehr klein positiv und dient als Indikator für die Konvergenz des Algorithmus. Grundsätzlich sollte die Differenz $diff = |\beta_{i+1} - \beta_i|$ nach jeder Iteration immer kleiner werden. Wenn der Algorithmus konvergiert, wird nach einer bestimmten Anzahl an Iterationen diese Differenz kleiner als der vorgegebene Konvergenzwert. Wenn die Differenz immer größer als der Konvergenzwert bleibt, kann festgestellt werden, dass der Algorithmus nicht konvergiert. Wenn es der Fall ist, terminiert der Algorithmus bei Erreichung der maximalen Anzahl an Iterationen (*maxIteration*).

2.3 Intepretation der Koeffizienten

Im Unterschied zu der linearen Regression lassen sich bei dem binären Logit-Modell die Koeffizienten nicht direkt interpretieren, da die Eintrittswahrscheinlichkeit von β durch eine komplexe Funktion abhängig ist (siehe Gleichung 3). Die Bruchrechnung $\frac{\pi_i}{1-\pi_i}$ (Gleichung 4) spielt bei dem binären Logit-Modell eine besondere Rolle, weil sie der Verbindung zwischen der Eintrittswahrscheinlichkeit und der Gegenwahrscheinlichkeit direkt widerspiegelt. Dieses wird als **Odds** bezeichnet. Der Begriff **Chance** ist eine andere Möglichkeit, die **Odds** darzustellen. Beispielsweise wird beim Münzwurf von einer 1:1-Chance für Kopf bespochen,

da die Wahrscheinlichkeiten von Kopf und Zahl gleich 0.5 sind und der Odd somit 1 ist. **Odds** lassen sich wie folgt zerlegen:

$$\begin{aligned} Odd(\pi_i) = \frac{P(Y=1)}{P(Y=0)} = \frac{\pi_i}{1-\pi_i} &\overset{(4)}{=} \exp(\eta_i) \\ &\overset{(1)}{=} \exp(\beta_0 + \beta_1.x_{i1} + \beta_2.x_{i2} + ... + \beta_p.x_{ip}) \\ &= \exp(\beta_0).\exp(\beta_1.x_{i1}).\exp(\beta_2.x_{i2})...\exp(\beta_p.x_{ip}) \end{aligned}$$

Wenn sich irgendein x_{ij} um 1 erhöht, folgt $\exp(\beta_j.(x_{ij}+1)) = exp(\beta_j.x_{ij} + \beta_j) = \exp(\beta_j.x_{ij}).\exp(\beta_j)$. Das Verhältnis von zwei Odds (**Odds Ratio**) beträgt somit:

$$\textbf{OR} = \frac{Odd(\pi_i|x_{ij}+1)}{Odd(\pi_i|x_{ij})} = \exp(\beta_j)$$

Die Chance, $\mathbf{Y}=1$ zu erhalten, verändert sich um $\exp(\beta_j)$-mal, wenn $\mathbf{X}_j$ um 1 steigt. Ist $\beta_j > 0$ und somit $\exp(\beta_j) > 1$, steigt die Chance der Eintrittswahrscheinlichkeit. Ist $\beta_j = 0$ und somit $\exp(\beta_j) = 1$, bleibt die Chance der Eintrittswahrscheinlichkeit gleich. Ist $\beta_j < 0$ und somit $\exp(\beta_j) < 1$, sinkt die Chance der Eintrittswahrscheinlichkeit.

3 Implementierung in R

Im Folgenden wird die Funktionalität von dem Paket **logitModell** erklärt, welches zum Ziel setzt, die Grundidee hinter dem binären Logit-Modell programmiert darzustellen. Das Paket besteht aus dem R-Code, welcher folgende Funktionen beinhaltet:

- `maxLikeEst(y,X)`: berechnent das Maximum Likelihood
- `logitMod(formula, data)`: erstellt Objekt mit der Klasse "logitMod", um mit S3-Methoden zu arbeiten
- `print.logitMod(x,..)`: Print-Methode
- `summary.logitMod(object,..)`: Summary-Methode
- `print.summary.logitMod(object,..)`: Bessere Einordnung der Ergebnisse aus der Summary-Methode
- `plot.logitMod(x,..)`: Plot-Methode

Alle diese Methoden werden in einer Extra-Vignette anhand einem konkreten Beispiel ausgeführt. Zum Aufruf der Extra-Vignette muss der folgende Code bei dem Errichten des Paketes ausgeführt werden. Die Extra-Vignette wird demnächst in der Help-Seite angezeigt:

```
devtools::install("~/Desktop/Uni/Master/WS1718/ProgR/Abschlussarbeit/logisticRegression/
vignette(topic = "vignetteLogitModell")
```

3.1 Beispieldatensatz

Der Beispieldatensatz wird im Folgenden verwendet, um die Richtigkeit und Vollständigkeit der Ergebnisse der implementierten Methode im Vergleich zu der R-Standardmethode für Logit-Modell *glm(..., family = "binomial")* zu testen. Die binäre Responsevariable heißt *admit*, welche besagt ob ein Kandidat eine Zulassung bekommt. Zudem enthält der Datensatz drei unabhängige Variablen: *gre*, *gpa* (metrisch) und *rank* (kategorial). Der Datensatz soll ein Modell unterstützen, welche die Abhängigkeit von der Wahrscheinlichkeit einer Zulassung von der Abschlussnote, GRE-Note sowie dem Ruf von der angestrebten Institution.

3.2 Implementierung der Maximum-Likelihood-Schätzung

Bevor das eigentliche Logit-Modell erstellt wird, wird die Implementierung der Maximum Likelihood Schätzung auseinandergesetzt. Der Code dazu ist auf Basis von dem dazugehörigen theoretischen Teil aufgebaut. Die Funktion *maxLikeEst* dient dazu, anhand der Newton-Raphson-Methode das maximale Likelihood zu berechnen. Als Input werden ein Vektor y (Zielvariable) und eine Matrix X (Designmatrix) benötigt. Als Output wird ein Objekt erwartet, welches die Maximum-Likelihood-Schätzer enthält.

Es muss immer zunächst überprüft werden, in welcher Art die Zielvariable eingegeben wird, denn das Maximum Likelihood braucht als Input numerische Vektoren für weitere Berechnungen. Die folgenden Codezeilen transformiert beispielsweise weiblich/männlich- oder Erfolg/kein Erfolg-Zielvariablen in 0/1-Variable. Dieser Schritt wird extra gemacht, damit sich das manuelle Modell im Hinblick auf den Input gleich verhält wie das Standardmodell.

```
# sei y die eingegebene Zielvariable
if (!(0 %in% y && 1 %in% y)) {
    y <- factor(y, labels = c(0,1))
}
y <- as.numeric(as.character(y))
```

Der Aufbau der Funktion maxLikeEst(y,X) sieht so aus:

```
maxLikeEst <- function(y, X) {
    #1. initialisiere Variablen
    #2. berechne das Maximum Likelihood anhand Newton-Raphson-Schleife
    #3. berechne nötige Parameter und speichere diese in einem Objekt
}
```

Zunächst werden die Parameter und Variablen für die Newton-Raphson-Methode initialisiert. β wird als ein Nullvektor mit der gleichen Länge wie die Anzahl der Spalten der Designmatrix initialisiert, um anschließend in der Newton-Raphson-Schleife nach jeder Iteration aktualisiert zu werden. Tolerance ist der Wert, bei dem die Schleife terminiert, wenn die absolute Veränderung der Lösungsgüte kleiner als dieser Wert ist. Das nächste Abbruchskriterium ist die maximale Anzahl der Iterationen. Solange die beiden Abbruchskriterien nicht erreicht

sind, erfolgt in der Newton-Raphson-Schleife ein iterativer Vorgang, welcher sich auf dem erklärten Pseudocode (siehe Abschnitt) basiert.

Nachdem der ML-Schätzer bestimmt wird, werden folgende Parameter berechnet. Diese Werte werden hierbei berechnet, um die Unabhängigkeit der S3-Methoden voneinander zu garantieren. Das bedeutet, dass beispielsweise `summary()` ohne `print()` aufgerufen werden kann.

- **Maximum Likelihood**: Um den Maximumwert der Log-Likelihood-Funktion zu erhalten, muss lediglich der erhaltene ML-Schätzer aus der Newton-Raphson-Methode in Gleichung 11 eingesetzt werden. Bei der Berechnung der Devianzen sowie des AICs findet dieser Wert Einsatz. Aus der Newton-Raphson-Methode wird ebenfalls die Anzahl der Iterationen zurückgegeben, welche der Anzahl der Fisher-Scoring-Iterationen entspricht.
- **Deviance Residuals**: Die Devianzresiduen lassen sich anhand der folgenden Formel berechnet (vgl. Breheny (2011)):

  $$d_i = s_i.\sqrt{-2[y_i.\ln(\pi_i) + (1-y_i).\ln(1-\pi_i)]}$$

  mit $s_i = 1$ wenn $y_i = 1$ und $s_i = -1$ wenn $y_i = 0$. Dieses gehört zu der Methode `summary()`
- **Degree of freedom**: Die Freiheitsgrade ist die Differenz zwischen der Anzahl an Beobachtungen (entspricht Zeilenanzahl von der Designmatrix) und der Anzahl an Parameter (entspricht Spaltenanzahl von der Designmatrix). Da das Null-Modell nur den Interzept enthält, ist hat die Designmatrix nur eine Spalte. Die Freiheitsgrade taucht bei `print()` und `summary()` auf.
- **Angepasste Wahrscheinlichkeit**: entspricht die Eintrittswahrscheinlichkeit der einzelnen Beobachtungen, welche durch das Modell angepasst wird (siehe Gleichung ... für die Berechnung)
- **Kovarianzmatrix**: $Cov(\beta) = (\mathbf{X'MX})^{-1}$. Dabei wird `solve()` verwendet, um die Inversematrix von $\mathbf{X'MX}$ zu berechnen.

Alle der eben berechneten Werte werden in einem Objekt gespeichert, welches zu der Klasse "logitMod" zugeordnet wird.

3.3 Definiere Klasse "logitMod"

Um anschließend mit S3-Methoden zu arbeiten, muss eine Klasse definiert werden, welche die Objekte aus der Funktion *maxLikeEst* umfasst. Diese Klasse wird als "logitMod". Da sich die Objekte von der Klasse "logitMod" ähnlich wie Objekte von *glm(...,family = "binomial")* verhalten sollen, werden dementsprechend als Input ein Formula und ein Datensatz erwartet. Daraus erfolgt Folgendes:

- Ein Modellframe wird definiert, welcher alle gegebenen Daten anhand einem Dataframe widergibt.

- Die Zielvariable y wird als `model.response()` aus diesem Modellframe entnommen.
- Die Designmatrix **X** wird als `model.matrix()` aus diesem Modellframe entnommen. Die Eins-Spalte für den Interzept wird automatisch erzeugt. Ebenfalls werden alle n-kategorialen Variablen als (n-1) Dummies-Variablen transformiert.
- y und **X** werden in die Funktion maxLikeEst eingesetzt. Das Objekt mit allen definierten Werten wird erzeugt.
- Zusätlich werden folgende Werte in das Objekt gespeichert:
- Der eingegebene Formula & der Call
- Das Null-Modell, welches lediglich aus der Zielvariable und dem Interzept β_0 besteht, das heißt es existiert dabei keine Abhängigkeit von den Kovariaten. Dieses wird bei der Berechnung der Null-Devianz gebraucht.
- **Null Deviance**: da das Null-Modell durch die Funktion maxLikeEst bestimmt wird und somit auch den maximalen Wert des Log-Likelihood enthält, gilt: Null-Devianz von dem angepassten Logit-Modell = Residuen-Devianz vom Null-Modell = -2 * Log-Likelihood(Null-Modell) .
- **Residual Deviance**: entspricht analog dazu -2 * Log-Likelihood(angepasstes Modell). Eine geringe Devianz spricht für eine gute Anpassung des Modells. Anhand Null- und Residuendevianz und dem Chi-Quadrat-Test kann weiterhin sichergestellt werden, ob eine Reduktion der Devianz statistisch signifikant ist (vgl. Zumel et al. (2014), S.169-171)
- **AIC**: $\mathbf{AIC} = (-2.\ell + 2.p)$ wobei ℓ dem maximalen Log-Likelihood und p der Koeffizientenanzahl entspricht.

3.4 Definiere S3-Methoden

Alle in Folgenden definierten S3-Methoden sollen dafür sorgen, dass das manuelle Logit-Modell die exakten Ergebnissen zurückgibt wie beim Standardmodell in R, wenn die Methoden *print, summary und plot* aufgerufen werden. Es werden hierbei kein Code für bessere Übersichtlichkeit beigefügt sondern nur die restlichen notwendigen Berechnungen zusammengefasst. Die Formel der Berechnugnen stammen aus Kapitel 4.1 der Vorlesungsfolien vom Modul *Statistische Modellierung* im Wintersemester 2016/2017 an der Freie Universität Berlin.

3.4.1 Print-Methode

Alle hierbei nötigen Werte werden bereit in das Objekt der Klasse "logitMod" gespeichert. Es geht bei der Print-Methode lediglich um die Einordnung der Werte in passendem Format. `cat()` verknüpft der betroffenen berechneten Werte mit den entsprechenden Begriffen, welche in der Konsole angezeigt werden sollen. `print.default()` sorgt dafür, dass die Werte der Koeffizienten exakt wie bei dem Standardmodell dargestellt werden. `invisible()` wird

schließlich ausgeführt, um einen verdoppelten Output zu vermeiden, weil ansonsten extra `print()` ausgeführt werden muss.

3.4.2 Summary-Methode

In der Summary-Methode werden zunächst die bisher noch fehlende Werte berechnet:

- **Standard Error** der Koeffizienten: entspricht der Wurzel von dem Diagonalen aus der Kovarianzmatrix. Der Standardfehler gibt an, inwieweit der geschätzte Koeffizient sich durchschnittlich schwankt, wenn die Stichprobe unendlich mal gezogen wird.
- **z-Stat**: entspricht die Bruchrechnung der einzelnen Koeffizienten durch der betroffenen Standardabweichung, welche sich annährend über eine Normalverteilung verteilen sollte, wenn die Stichprobenumfang groß genug ist.

- **p-Value**: anhand dem p-Wert kann festgestellt werden, ob sich der geschätzte Koeffizient mit einem bestimmten Konfidenzniveau von Null unterscheidet. Die Formel für den p-Wert bei einem zweiseitigen Test (mit $H_0 : \beta_i = 0$ und $H_1 : \beta_i \neq 0$) lautet: $2 * P(x < -|zStat|)$. Ein kleiner p-Wert erfolgt die Ablehnung von H_0 und besagt, dass sich β_i signifikant von Null unterscheidet.

Bisher gibt die Summary-Methode eine Liste von allen dazugehörigen Werten in einer Liste zurück, was extrem unübersichtlich wäre. Demzufolge sorgt die Funktion `print.summary()` für die richtige Einordnung der jeweiligen Werten. Die meisten Werte müssen lediglich mit dem passenden Begriffen anhand `cat()` verknüpft werden. Für die Devianz-Residuen und Koeffiziententabelle werden zusätzlich folgende Standardfunktionen verwendet:

- **print.summaryDefault**: fasst die Verteilung der Devianz-Residuen in einem Standardformat mit Minimum, Quantilen und Maximum zusammen.
- **printCoefmat**: ordnet alle relevanten Werte in Bezug auf die Koeffizienten in einem Tabellenformat zu. Hierbei wird für `signif.legend = TRUE` entschieden, um ebenfalls die Signifikanzniveaus anzuzeigen.

Schließlich wird ebenfalls `invisible()` aufgerufen, um eine Verdoppelung des Outputs zu vermeiden.

3.4.3 Plot-Methode

Es werden hierbei insgesamt 4 Grafiken erwartet:

1. **Residuals vs Fitted**: stellt graphisch den Zusammenhang zwischen Devianzresiduen und den vorhergesagten wahren Wert dar, welcher der Multiplikation $\mathbf{X}.\beta$ entspricht.
2. **Normal Q-Q**: dient zum Testen, ob die Residuenstandardfehler eine Normalverteilung folgen. Wenn es der Fall ist, soll die Punkte aus `qqnorm()` sehr gut mit der Gerade `qqline()` übereinstimmen.
3. **Scale Location**: zeigt, ob Devianzresiduen sich gleichmäßig über die Spannweite der geschätzen Werten verteilen.

4. **Residual vs Leverage**: identifiziert einflussreiche Beobachtungen. Hierbei werden zwei neue Kennwerte ermittelt.[1]
   - Standardisierte Pearson-Residuen: $r_i = \frac{y_i - \hat{\pi}_i}{\sqrt{\hat{\pi}_i.(1-\hat{\pi}_i)}}$
   - Leverage: Diagonale von $\mathbf{H} = \mathbf{W}^{1/2}\mathbf{X}(\mathbf{X}'\mathbf{M}\mathbf{X})^{-1}\mathbf{X}'\mathbf{W}^{1/2}$

Literaturverzeichnis

Breheny, P., 2011. Logistic regression: Diagnostics. https://web.as.uky.edu/statistics/users/pbreheny/760/S11/notes/4-12.pdf Letzter Zugriff: 26.03.2018

Quinn, K., 2001. The newton raphson algorithm for function optimization. University of Washington Seattle.

Schlittgen, R., 2013. Regressionsanalysen mit r. Walter de Gruyter.

Ypma, T.J., 1995. Historical development of the newton–Raphson method. SIAM review 37, 531–551.

Zumel, N., Mount, J., Porzak, J., 2014. Practical data science with r. Manning.

[1] vgl. https://web.as.uky.edu/statistics/users/pbreheny/760/S11/notes/4-12.pdf