Logistische Regression

Xuan Son Le (4669361), Freie Universität Berlin

02/04/2018

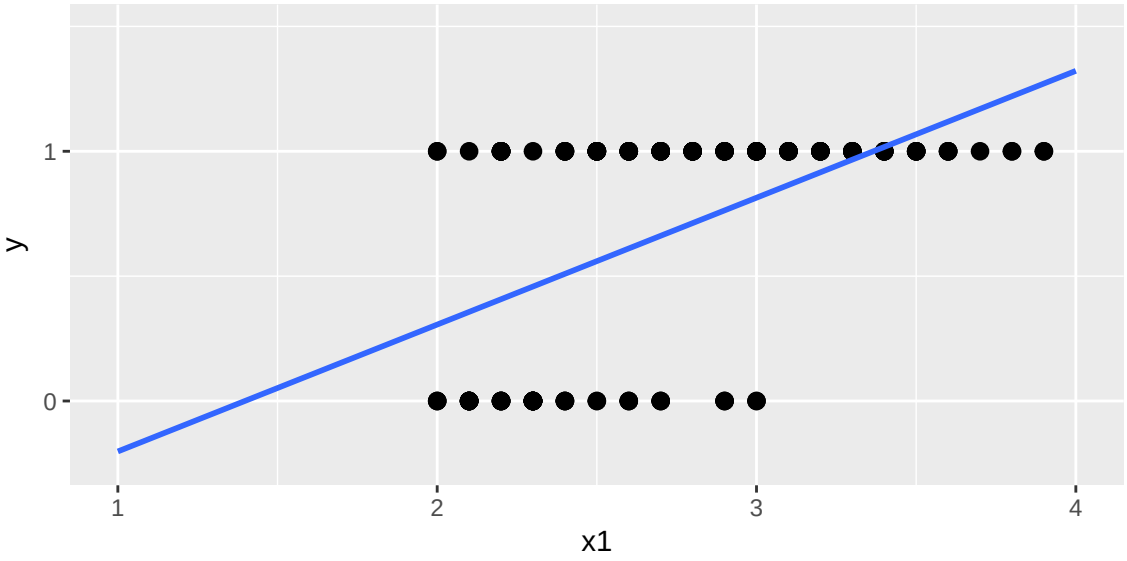
Abstract: Im Rahmen der Abschlussarbeit des Moduls Programmieren mit R im Wintersemester 2017/2018 an der Freie Universität Berlin wird für diese Arbeit die statistische Methode namens binäres Logit-Modell ausgewählt. Diese Arbeit besteht aus zwei großen Hauptteilen: der Theorieteil, wobei die ausgewählte Methode theoretisch vorgestellt wird und der Implementierungsteil, welcher die Erklärung der Funktionalität vom selbst entwickelten Paket beinhaltet. Im Theorieteil wird zunächst ein Überblick über die grundliegende Funktionsweise vom (binären) Logit-Modell widergegeben. Die Grundidee von Generalisierten linearen Modellen wird anschließend kurz eingeführt, bevor der Aufbau vom binären Logit-Modell durch das Maximum Likelihood Verfahren vorgenommen wird. 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, Paket, R*

---

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. Die folgende Grafik stellt den Ansatz graphisch dar, binäre Variable durch lineare Regression zu modellieren:



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, welches ebenfalls als binäre logistische Regression oder binäres logistisches Regressionsmodell bezeichnet werden kann. 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 . 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. Die Grundidee 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, R. (2013) können Modelle zu GLMs zugeordnet werden, wenn sie: 1. 2.

2.1 Modellspezifikation

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 $\mathbf{X}_j = (x_{1j}, x_{2j}, ..., x_{nj})$ - 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 $\eta_i = \beta_0 + \beta_1.x_{i2} + \beta_2.x_{i3} + ... + \beta_p.x_{ip} = \mathbf{x'}_i.\beta$ (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:

$$\pi_i = \frac{\exp(\eta_i)}{1 + \exp(\eta_i)} = \frac{\exp(\beta_0 + \beta_1.x_{i2} + \beta_2.x_{i3} + ... + \beta_p.x_{ip})}{1 + \exp(\beta_0 + \beta_1.x_{i2} + \beta_2.x_{i3} + ... + \beta_p.x_{ip})} = \frac{\exp(\mathbf{x'}_i.\beta)}{1 + \exp(\mathbf{x'}_i.\beta)} \quad (3)$$

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:

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

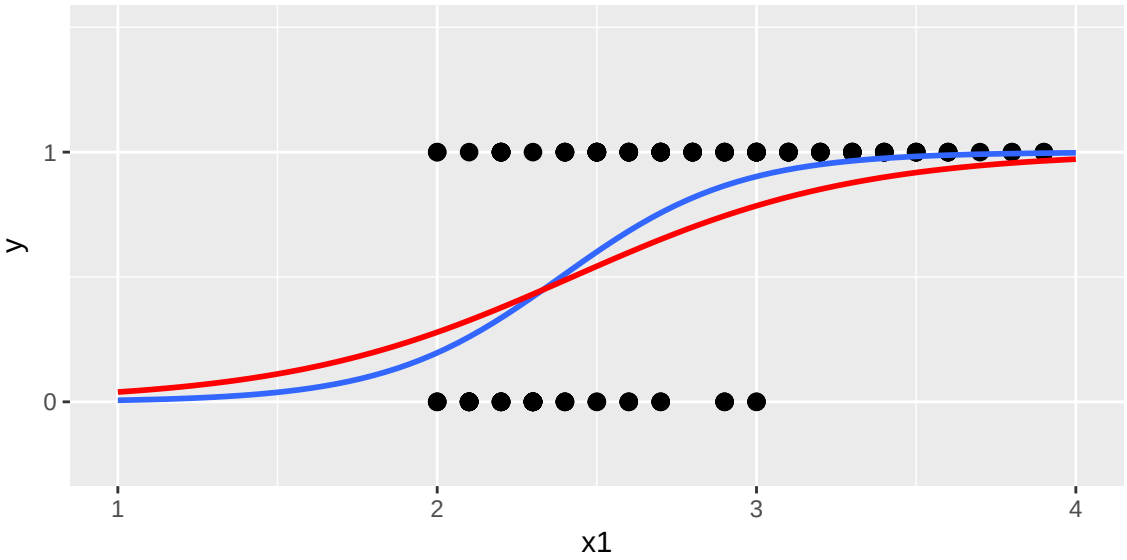
Es gilt nämlich:

$$\beta_0 + \beta_1.x_{i2} + \beta_2.x_{i3} + ... + \beta_p.x_{ip} = \ln(\frac{\pi_i}{1 - \pi_i}) \quad (5)$$

2.2 Maximum Likelihood Schätzung

Ähnlich wie bei der linearen Regression muss 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 Abweichung zwischen dem wahren Wert und dem geschätzten Wert 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 .. zeigt ein Beispiel mit zwei mögliche binäre logistiche Regressionskurven, die durch das Maximum Likelihood optimiert werden sollen.



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 Likelihood-Funktion wie folgt logarithmisieren:

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

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 ausgewählt. Dafür muss die zweite Ableitung der Funktion noch gebildet werden, welche als Hesse-Matrix (H) bezeichnet wird:

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

Wegen

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

gilt es:

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

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

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

2.2.5 Newton-Raphson-Methode

Die Newton-Raphson-Methode setzt sich zum Ziel, eine nichtlinear lösbare Funktion zu optimieren. Diese Methode wurde ... (QUELLE).

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:

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

In Anlehnung an ... wird demnächst die Newton-Raphson-Methode anhand einem Pseudocode näher erläutert:

```
1 function(X, y):
2   initialisiere: $$\beta_0$$, maxIteration, i, tolerance, diff, M
3   while (diff > tolerance)
4       betaChange = $(\mathbf{X}'\mathbf{M}\mathbf{X})^{-1}.\mathbf{X}'.(y-\pi)$
5       diff =
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

Die Bruchrechnung $\frac{\pi_i}{1-\pi_i}$ (siehe Gleichung 3) spielt bei dem binären Logit-Modell eine besondere Rolle, weil sie der Verbindung zwischen der Eintrittswahrscheinlichkeit und der Gegenwahrscheinlichkeit direkt widerspiegelt.

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 anhand dem manuell berechneten Maximum Likelihood ein Objekt von der Klasse *logitMod* erstellt und anschließend drei S3 Methoden für diese Klasse (print, summary und plot) definiert, und einer Vignette, welche den R-Code anhand einem konkreten Beispiel ausführt.

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.

Zunächst wird der Datensatz importiert. Dabei wird die Zielvariable aus dem Datensatz entnommen und in einem Vektor gespeichert. Da diese schon als 0/1-Variable vorgegeben wird, besteht es in diesem Fall keine Notwendigkeit, die Zielvariable zu faktorisieren. Der Code funktioniert allerdings ebenfalls mit Zielvariable, welche zum Beispiel als weiblich/männlich oder Erfolg/kein Erfolg kodiert wird und transformiert diese in eine 0/1-Variable.

```
# 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))
```

Es muss immer vorab überprüft werden, in welcher Art die Zielvariable eingegeben wird, denn das Maximum Likelihood braucht als Input numerische Vektoren für weitere Berechnungen. Dieser Schritt wird extra gemacht, damit sich das manuelle Modell im Hinblick auf den Input gleich verhält wie das Standardmodell.

Für das Paket wird eine Extra-Vignette in R aufgebaut, um die Ergebnisse anhand dem Beispieldatensatz zu verdeutlichen. Zum Aufruf der Extra-Vignette muss der folgende Code bei dem Errichten des Paketes ausgeführt werden:

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

3.2 Implementierung der Maximum-Likelihood-Schätzung

Bevor das eigentliche Logit-Modell erstellt wird, wird in diesem Abschnitt die Implementierung der Maximum Likelihood Schätzung auseinandergesetzt. Der Code dazu ist auf Basis von dem betroffenen theoretischen Teil (siehe Abschnitt ...) aufgebaut. Die Funktion *maxLikeEst* dient dazu, anhand der Newton-Raphson-Methode das maximale Likelihood zu berechnen. Als Input werden ein Vektor (Zielvariable) und eine Matrix (Designmatrix) benötigt. Als Output wird ein Objekt erwartet, welches die Maximum-Likelihood-Schätzer enthält. Daneben werden ebenfalls die ersten notwendigen Kennwerte für den Aufbau von S3-Methoden in dem Objekt gespeichert.

Der Aufbau der Funktion maxLikeEst 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:

```
# initialisiere beta
beta <- rep(0, times = ncol(X))

# initialisiere ...
M <- diag(nrow = nrow(X))

# stelle Abbruchskriterien fest
tolerance <- exp(-6)
diff <- 10 * abs(tolerance)
maxIteration <- 100
i <- 0
```

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 folgender iterativer Vorgang:

```
while (diff > tolerance & i < maxIteration) {
```

```
    # berechne die Wahrscheinlichkeit
    eta <- X %*% beta
    exp_eta <- exp(eta)
    p <- as.vector(exp_eta / (1 + exp_eta))

    # aktualisiere ...
    M <- diag(p * (1 - p))

    # berechne die Änderung von Beta
    betaChange <- solve(t(X) %*% M %*% X) %*% t(X) %*% (y - p)

    # aktualisiere Beta
    beta <- beta + betaChange

    # berechne die totale Änderung von Beta
    diff <- sum(abs(betaChange))

    # nächste Iteration
    i <- i + 1

    if (i > maxIteration) {
        stop("Die Funktion konvergiert nicht!")
    }
}
```

Die Änderung von β führt dazu, dass nach jeder Iteration ebenfalls die Eintrittswahrscheinlichkeit sowie ... Matrix neu definiert werden müssen.

Als Output der Funktion maxLikeEst werden folgende Parameter in ein Objekt gespeichert, um mit den Werten aus dem R-Standard-Modell zu vergleichen.

```
# Freiheitsgrad = Anzahl an Beobachtungen - Anzahl an Parameter
dfRes <- nrow(X) - ncol(X)  # ResiduensFreiheitsgrad
dfNull <- nrow(X) - 1       # Nullmodell Freiheitsgrad

# Devianz Residual
s <- y
s[s == 0] = -1
devianceResiduals = as.numeric(s * sqrt(-2*((y * eta) - (log(1 + exp(eta))))))

# Kovarianzmatrix
vcov <- solve(t(X) %*% M %*% X)

# Maximumwert der Log Likelihood Funktion
maxLogLikeValue <- (sum((y * X %*% beta) - (log(1 + exp(X %*% beta)))))
```

Für die Print-Methode werden die Freiheitsgrade, das Devianz-Residual und das Maximal-Likelihood bestimmt. Dieses wird demnächst in der Print-Methode näher erläutert. Zum Vergleich der Ergebnisse der Funktion *maxLikeEst* mit dem Standardmodell werden die ML-Schätzer und die Kovarianzmatrix zurückgegeben.

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" bezeichnet und wird wie folgt definiert:

```
logitMod <- function(formula, data) {

    # initialisiere y (Zielvariable) und X (Designmatrix)
    modelFrame <- model.frame(formula, data)
    X <- model.matrix(formula, modelFrame)
    y <- model.response(modelFrame)

    # # falls y nicht als 0-1/Variable eingegeben wird
    # if (!(0 %in% y && 1 %in% y)) {
    #     y <- factor(y, labels = c(0,1))
    # }
    # y <- as.numeric(as.character(y))

    # erstelle eine Objekt aus der maxLikeEst-Funktion
    result <- maxLikeEst(y, X)

    # erstelle das Null Modell und speichere es in die Ergebnisliste
    nullModell <- maxLikeEst(X = matrix(rep(1, times = nrow(X)), ncol = 1),
                             y = y)
    result$nullModell <- nullModell

    # speichere notwendige Parameter in die Ergebnisliste
    result$formula <- formula
    result$call <- match.call()
    result$X <- X
    result$y <- y

    # ordne die Ergebnisliste der Klasse "logitMod" zu
    class(result) <- "logitMod"

    return(result)
}
```

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 werden die Zielvariable sowie die Designmatrix entnommen und anschließend in die Funktion maxLikeEst eingesetzt. Darüber hinaus wird das Null-Modell erstellt und ebenfalls in das "Hauptobjekt" für leichteren späteten Aufruf gespeichert.

3.4 Definiere S3-Methoden

In diesem Abschnitt wird die Implementierung von drei S3-Methoden anhand bisherigen Ergebnissen vorgestellt. Der Code wird ebenfalls anhand des Beispielsdatensatzes in der Extra-Vignette umgesetzt.

3.4.1 Print-Methode

```
print.logitMod <- function(x, ...){

    cat("Call: ", paste0(deparse(x$call)), fill = TRUE)

    cat("\n\nCoefficients:\n")

    print.default(format(coef(x)[,1], digits = 4L),
                  print.gap = 1L, quote = FALSE, right = TRUE)

    cat("\nDegrees of Freedom: ", x$dfNull,
        " Total (i.e. Null); ", x$dfRes, " Residual")

    # Berechnung von null deviance, residual deviance & aic
    nullDeviance <- -2 * x$nullModell$maxLogLikeValue
    x$nullDeviance <- nullDeviance
    residualDeviance <- -2 * x$maxLogLikeValue
    x$residualDeviance <- residualDeviance
    x_AIC <- (-2*x$maxLogLikeValue + 2*ncol(x$X))
    x$AIC <- x_AIC

    cat("\nNull Deviance:\t", round(nullDeviance,1))
    cat("\nResidual Deviance:", round(residualDeviance,1), "\t",
        "AIC: ", round(x_AIC,1), "\n")

    cat("\nNumber of Fisher Scoring iterations: ", x$anzahlIteration, "\n")

    # invisibly return linMod object
    invisible(x)
```

```
}
```

3.4.2 Summary Methode

```
summary.logitMod <- function(object, ...) {

    # Koeffizienten Standardfehler
    object$betaStandardError  <- as.matrix(sqrt(diag(object$vcov)))

    # z-Statistik
    zStat <- object$coefficients / object$betaStandardError
    object$zStat <- zStat

    # p-Werte
    pValue <- 2 * pnorm(-abs(zStat))
    object$pValue <- pValue

    # Zusammenfassung der Werte für die Koeffizienten
    object$coefficients <- cbind("Estimate" = object$coefficients[,],
                            "Std. error" = object$betaStandardError[,],
                            "z value" = object$zStat[,],
                            "Pr(>|z|)" = object$pValue[,])

    # Berechnung von nullDeviance, residualDeviance & aic
    nullDeviance <- -2 * object$nullModell$maxLogLikeValue
    object$nullDeviance <- nullDeviance
    residualDeviance <- -2 * object$maxLogLikeValue
    object$residualDeviance <- residualDeviance
    x_AIC <- (-2*object$maxLogLikeValue + 2*ncol(object$X))
    object$AIC <- x_AIC

    class(object) <- "summary.logitMod"

    return(object)

}
```