logitModel

Marcelo Rainho Avila

24.04.2018

Abstract

As part of the *Statistical Programming with R* course ministered in the Winter Term of 2017 / 2018, this vignette aims to explore the implementation of an alternative package that conducts a logistic regression estimation. The created package includes a Maximum Likelihood Estimator based on the Newton-Rahpson algorithm, a formula interface for easier user interaction, a `print`, `summary`, `plot` and S3-Methods that mimics the `glm(..., family = binomial)` implementation of the *stats* package as well as a `pairs` method for an overview of the interaction between the explanatory and explained variables.

Contents

1 Introduction

Differently to the linear regression estimation methods, the logistic regression, being one member of the generalized linear models, is used when conducting estimations for a categorical response data. A categorical variable has a measurement scale of a set of countable categories. A student might be trying to decide between studying Economics, Administration or Psychology. A political scientist, for example, might be interested in the political view of the participants in a interview, such as “left-leaning”, “moderate” or “right-leaning”. A physician could be classifying the stage of a patients disease as in “initial”, “middle” or “advanced” stage. A computer scientist might be interested in classifying whether an image contains a hot-dog or it does not. These type of data is of importance in several applications and research areas.

From the above examples, one is able to verify certain distinctions between the types of information. The choices faced by the student are called to have a *nominal* scale, since there is not a natural way of ordering those subjects. On the other hand, the disease stage of interest to the physician has a natural ordering and, thus, can be said to have a *ordinal* scale. The most simple of these type of data are *binary*, that is, it can only take two possible values, such as “success / failure”, “won / lost”, “hot-dog / not hot-dog” and, more generally, “1 / 0”. For this type of data, a logistic regression can be conducted for estimating the probability of an event occurring give the set of explanatory variables that the model is built upon and will be more deeply explored in the next sections.

2 Theoretical Background

In this section, I will explore the theoretical underpinning of the logistic regression model.

Assume there are p explanatory variables and one wants to model their influence over the probability π, such that

$$\pi_i = P(Y_i = 1) = \mathbb{E}(Y_i) \ .$$

Assuming a linear regression model holds, it would follow that

$$\pi_i = P(Y_i = 1) = \mathbb{E}(Y_i) = \boldsymbol{x_i'\beta} \ ,$$

where the probability of success changes linearly with changes of x, with β being the change in expected probability in π respective to a unit change in x. This simple model can be a very good approximation and very useful for a range of data values of x. Nevertheless, due to the linearity of $\boldsymbol{x_i'\beta}$, this model could take values that exceed the range $[0, 1]$, which is nonsensical when dealing with probability values.

To solve this problem, the probability $P(Y_1 = 1)$ is transformed via a Distribution function $F(\eta)$, where η is a linear predictor $\eta_i = \boldsymbol{x_i'\beta}$. There are a few commonly used *reponse* functions with the only restriction that $F()$ is strictly monotonically increasing over the whole Real Numbers with $F(\eta_i) \in [0, 1]$. The exact choice of the *response function* $F()$, and the corresponding *link function* (that is, the inverse of the response function) will define the *kind* of binary response variable modeling to be conducted. For the standard normal distribution, one obtains a *probit model.*

For the **binary logit model**, the Logist function is employed. The model can be, thus, modeled as a logistic regression such that

$$\pi_i = F(\eta_i) = \frac{\exp(\eta_i)}{1 + \exp(\eta_i)} \tag{1}$$

or with the **logit link function**

$$g(\pi_i) = \log\left(\frac{\pi_i}{1 - \pi_i}\right) = logit(\pi_i) = \eta_i = \boldsymbol{x_i'\beta} \ . \tag{2}$$

In order to better visualize the relationship between the explanatory and the response variable (in a partial scenario), refer to Figure 1. Here it is easy to observe the non-linearity between the explanatory and the response variable. For different values of x, the increased probability of *sucess* behave very differently if the starting values of x are relatively low compared to the same variation in x when starting from a higher value.

2.1 Interpretation of the results

Due to the non-linearity of the model, one cannot directly interpret the variable coefficients as changes in the probability of an event occurring. To understand the model, we can refer to the *Odds* of the occurrence of an event. This is defined as the probability of the event occurring against the probability that it does not. Formally,

$$Odds(\pi) \equiv \frac{P(Y = 1)}{P(Y = 0)} = \frac{P(Y = 1)}{1 - P(Y = 1)} = \frac{\pi}{1 - \pi} \ .$$

This terminology is often used in gambling scenarios. That is, if the odds of winning in a hand of poker is 3 to 1 (i.e. 3/1 or 3:1), it is expected that the player wins 3 times out of every 4 games, which equals to 75 percent. And, as expected $\frac{3}{1} = \frac{0.75}{1-0.75}$.

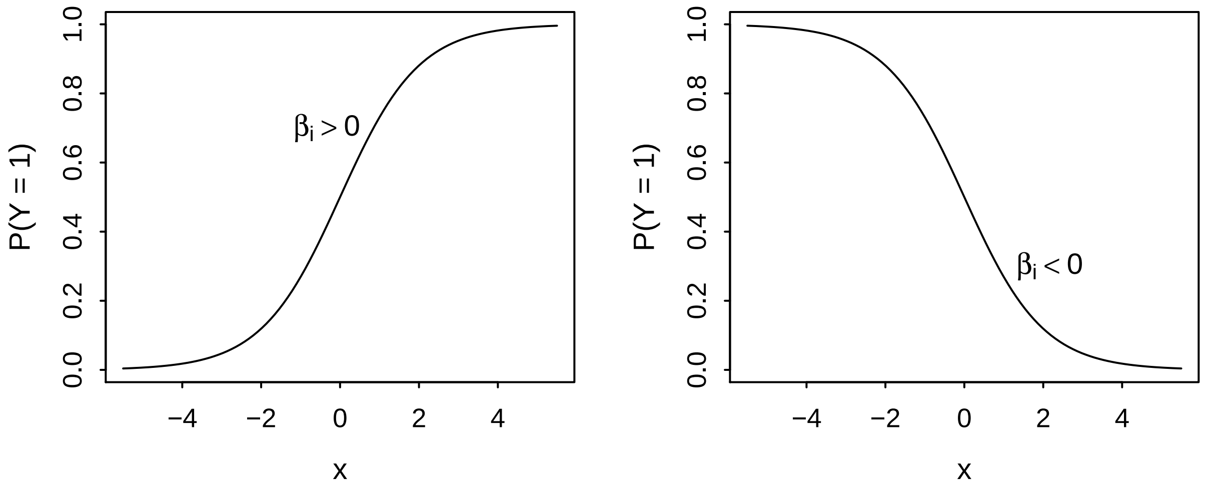


Figure 1: Shape of logistic function for positive and negative values of β and $\alpha = 0$.

From Equations 1 and 2 one can show that

$$Odds(\pi_i) = \frac{\pi_i}{1-\pi_i} = \frac{\frac{\exp(\eta_i)}{1+\exp(\eta_i)}}{\frac{1}{1+\exp(\eta_i)}} = \exp(\eta_i)\ . \tag{3}$$

Moreover, taking the log on both sides of equation (3), one gets

$$\log\left(\frac{\pi_i}{1-\pi_i}\right) = \eta_i = \boldsymbol{x_i'\beta} = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_k x_{ik}\ .$$

Furthermore, due to the product rule of exponentials, one can show that

$$Odds_1(\pi \mid x_{i1}) = \frac{P(Y=1 \mid x_{i1})}{P(Y=0 \mid x_{i1})} = exp(\beta_0) \cdot exp(\beta_1 x_{i1}) \cdot\ ...\ \cdot exp(\beta_k x_{ik}). \tag{4}$$

And, if one increases x_{i1} by one unit, while keeping other variables constant, one gets

$$Odds_2(\pi \mid x_{i1}+1) = \frac{P(Y=1 \mid x_{i1}+1)}{P(Y=0 \mid x_{i1}+1)} = exp(\beta_0) \cdot exp(\beta_1(x_{i1}+1)) \cdot\ ...\ \cdot exp(\beta_k x_{ik}). \tag{5}$$

Finally, if we divide $Odds_2$ by $Odds_1$, what is defined as the *Odds Ratio* between the two scenarios, we see that all terms cancel themselves out, leaving only the term with the added unit, as in

$$Odds\ Ratio_{(2,1)} = \frac{Odds_2(\pi, x_{i1}+1)}{Odds_1(\pi, x_{i1})} = exp(\beta_1). \tag{6}$$

For instance, if $\beta_1 = 0.3$ and x_1, $exp(\beta_1) \approx 1.35$, which means that - for a continuous x - the **odds** (not the probability) of the event occurring increases by about 35 percent, that means, it gets multiplied by 1.35, as x increases by one unit. One can observe that while the Odds Ratio remains constant for any x, the probability itself depends on the base value one starts with. In case x_1 was a dummy variable (such as $x_1 = 0$, for male and $x_1 = 1$ for female) and keeping the same coefficient as illustrated above, than the odds of the event occurring is 35 percent higher for a female, when comparing to a male with, otherwise, same characteristics.

In general, if:

- $\beta_i > 0 \Rightarrow exp(\beta_i) > 1 \Rightarrow Odds\ increases$
- $\beta_i < 0 \Rightarrow 0 < exp(\beta_i) < 1 \Rightarrow Odds\ decreases$
- $\beta_i = 0 \Rightarrow exp(\beta_i) = 1 \Rightarrow Odds\ remains\ constant$

2.2 Maximum Likelihood Estimation Method

The Maximum Likelihood Estimation (MLE) is an method that estimates the parameters of a model by maximizing the *Likelihood* of obtaining the observed data. For a binary response variable stands that each $Y_i \sim B(1, \pi_i)$, where $\pi_i = \mathbb{E}(Y_i)$, that is, each observation is follows Binomial Distribution with one observation, which follows, per definition, a Bernoulli Distribution

$$f(x \mid \pi_i) = {\pi_i}^{y_i}(1-\pi_i)^{1-y_i} \ .$$

Assuming independent and identically distributed (iid) observed variables, the **Likelihood** function is defined as

$$\mathscr{L}(\boldsymbol{\beta}) = \prod_{i=1}^{n} f(y_i \mid \pi_i) = \prod_{i=1}^{n} {\pi_i}^{y_i}(1-\pi_i)^{1-y_i} \ ,$$

and for easier handling of the product over multiple terms, one can take the logarithm and arrive at the **log-likelihood**,

$$\begin{aligned}\ell(\boldsymbol{\beta}) \equiv log(\mathscr{L}(\boldsymbol{\beta})) &= \sum_{i=n}^{n} \Big(y_i \log(\pi_i) - y_i \log(1-\pi_i) + log(1-\pi_i) \Big) \\ &= \sum_{i=n}^{n} \Big(y_i \log(\frac{\pi_i}{1-\pi_i}) + \log(1-\pi_i) \Big)\end{aligned}$$

In the case of the logit models, from the above equation, it follows that

$$\ell(\boldsymbol{\beta}) = \sum_{i=n}^{n} \big(y_i(\boldsymbol{x_i'}\boldsymbol{\beta}) - \log(1+\exp(\boldsymbol{x_i'}\boldsymbol{\beta})) \big) \ .$$

2.2.1 Score Vector

For the calculation of the parameters that maximizes the Likelihood-function one can take the first derivative of the log-likelihood function over the parameters $\boldsymbol{\beta}$, which is defined as the *score vector*

$$\begin{aligned}s(\boldsymbol{\beta}) = \frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} &= \frac{\partial}{\partial \boldsymbol{\beta}} \sum_{i=1}^{n} \Big(y_i\eta_i - \log(1+\exp(\eta_i)) \Big) \\ &= \sum_{i=1}^{n} \Big(y_i - \frac{1}{1+\exp(\eta_i)} \exp(\eta_i) \Big) \boldsymbol{x_i'} \qquad (7) \\ &= \sum_{i=1}^{n} (y_i - \pi_i) \boldsymbol{x_i'}\end{aligned}$$

Extreme values for the estimates of $\boldsymbol{\beta}$ can be found by setting Eq. 7 equal to zero and solving for each β coefficient. Due to the monotonic transformation property of the logarithmic function, it can be shown that the vector of parameters that maximizes the log-likelihood is the same as maximizing the actual likelihood function (A Czepiel 2002, 5).

The solutions, if existing, define the maximum point if the matrix of the second partial derivatives, denominated as Hessian Matrix, is negative definite, that is, if every diagonal element is less than zero (see A Czepiel 2002, 6). Since the Hessian Matrix computes the second derivative, that is, the curvature of the log-likelihood function, one can better grasp that the Hessian Matrix forms the variance-covariance matrix of the parameters estimates. That is, a matrix with a strong curvature contains *abundant* information over the observed data and, therefore, less variation on the parameters is expected. On the other hand, a weakly curved matrix contains little information on the observed data and the estimated parameters can vary greatly.

2.2.2 Fisher Information Matrix

The Fisher Information Matrix is a measure of the information that a random variable X carries about a unknown parameter upon which the probability of X depends. It is derived from the Hessian Matrix and computed as

$$\mathcal{I}(\boldsymbol{\beta}) = -\mathbb{E}\Big(H(\boldsymbol{\beta})\Big) = -\mathbb{E}\left(\frac{\partial s(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}\right)$$

It can also be shown that the asymptotic variance of the ML Estimator reaches the Cramer-Rao bound, which fixes the lower bound of the variance of a unbiased estimator

$$\text{Var}(\hat{\boldsymbol{\beta}}_{ML}) = \frac{1}{\mathcal{I}(\boldsymbol{\beta})} \leq \text{Var}(\theta) \ .$$

2.3 The Newton-Raphson Method

Solving a system of non-linear equations cannot be done algebraically, as it is the case with the ordinary least squares estimator. One popular algorithm for solving such problems is the Newton-Raphson method, which estimates the unknown parameters via a iterative process, that is explored in this section, which is based on the explanation from Czepiel (2002, pp. 7-8).

Beginning with an initial guess for the solution, the Newton-Raphson method guess the first two terms of a Taylor polynomial approximation evaluated at the initial guess to obtain a better estimate, that is, one closer to the solution. The iterative algorithm can be basically described by

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

in order to solve a non-linear equation $f(x) = 0$ (see Ypma 1995, 531). For a illustration of the method, refer to Fig. 2, based on (Polyak 2007, 1087).

In the logistic regression, the (system of) equations to be solved are as defined in Eq. (7) for each β, where

$$\boldsymbol{\beta}^{i+1} = \boldsymbol{\beta}^{i} - \frac{s(\boldsymbol{\beta})}{H(\boldsymbol{\beta})} \ .$$

In matrix notation, one can show that

$$s(\boldsymbol{\beta}) = \boldsymbol{X}^T(y - \pi)$$

and

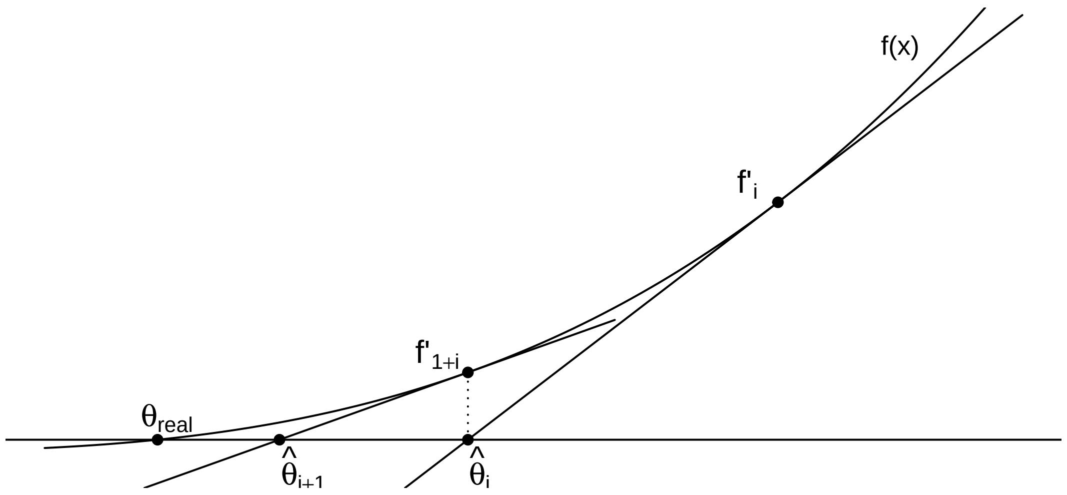


Figure 2: Illustration of the Newton-Rahpson Method.

$$H(\boldsymbol{\beta}) = -\boldsymbol{X^T W X} \ ,$$

where $s()$ and $H()$ are the score and hessian matrix, respectively, and $\boldsymbol{W}$ is a square matrix with $\pi_i(1-\pi_i)$ on its diagonal.

Thus, in matrix notation, we have

$$\boldsymbol{\beta}^{i+1} = \boldsymbol{\beta}^i + (\boldsymbol{X^T W X})^{-1} X^T(\boldsymbol{y} - \boldsymbol{\pi}) \tag{8}$$

The algorithm essentially apply the above formulation until a certain tolerance criterion is reached, such that the difference $\boldsymbol{\beta}^{i+1} - \boldsymbol{\beta}^i$ is smaller than a chosen tolerance level. The details about the necessary regularity conditions for convergence is out of the scope of this paper, nonetheless, one should always be aware that the algorithm might never converge or get stuck in a local maximum among other problematic scenarios.

3 R Implementation

In this section I will elaborate on the R implementation of the logist regression method as detailed on the previous section.

For this example, I will use the *Donner–Reed Party*[1] data set that is included in `package:logitModel`. The data is about the tragic event that occurred during the winter of 1846–47 when a group of American pioneers set out to California in a wagon train and got stuck midway in Sierra Nevada. Only about half of the party survived to reach California.

The logitModel function formula interface

Via the formula interface, the user can easily pass the arguments whether continuous or categorical to the function as it is implemented in other common `R` functions.

```r
library(logitModel)
fit <- logitModel(survived ~ age + sex, data = DonnerData)
```

While the logtiModel function is rather long, the process is rather straightforward and can be divided into three sections. Initialization, estimation computation and return of results.

The initialization part first checks if the user passed a `data.frame` as an argument and uses the variables passed in the formula from that `data.frame`. In case the user desires not to use a data frame, as, for example, it might be desired to use variables defined in the global environment, then the function will search for the variable the parent environment.

```r
logitModel <- function(formula, data, precision = 1e-10, iterMax = 25) {

  # generate model.frame. If "data" missing, search variables in Parent Env
  if(missing(data)) {
    modelFrame <- model.frame(formula, data = parent.frame())
  } else {
    modelFrame <- model.frame(formula, as.data.frame(data))
  }
  ...
```

Over the next few steps, the design matrix and response variable are retrieved via the `model.matrix()` and `model.frame()` functions, respectively, and a few checks are conducted in order to make sure the variables are binary and coded as zeros and ones for the following computations.

```r
  ...
  # Extract design matrix (X) and response var (y)
  X <- model.matrix(formula, modelFrame)
  y <- model.response(modelFrame)

  # make sure response variable is coded with 0s and 1s
  if (!(0 %in% y && 1 %in% y)) {
    y <- factor(y, labels = c(0, 1))
  }
  y <- as.numeric(as.character(y))

  # sanity check
```

[1]A detailed account of the tragic event can be found on https://en.wikipedia.org/wiki/Donner_Party

```
  if (length(unique(y)) != 2) {
    stop("Response variable is expected to be binary")
  }
  ...
```

In the computation part, first a *Null model* estimation is conducted, where only the intercept is passed to the `newtonRaphson()` function and a few statistics are computed for this model. Then, the estimation for the *full model* is conducted and some more statistics are computed.

```
  ...
  # Restricted Model
  XNull <- as.matrix(rep(1, length(y)))
  restricModel <- newtonRaphson(y = y,
                                X = XNull,
                                precision = precision,
                                iterMax = iterMax)

  bNull <- restricModel$beta
  logLikelihoodNull <- (sum((y * XNull %*% bNull) -
                             (log(1 + exp(XNull %*% bNull)))))

  devianceNull <- -2 * logLikelihoodNull
  dfNull <- nrow(X) - 1       # DF Null Model


  # Estimation Model
  MLEstimation <- newtonRaphson(y, X, precision = precision, iterMax = iterMax)
  ...
```

The S3 print() Method

The S3 `print()` method mimics the `stats:::print.glm()` output, albeit with changes for a more compact view of the basic results.

```
fit
# Call:  logitModel(formula = survived ~ age + sex, data = DonnerData)
#
# Coefficients:
# (Intercept)          age      sexMale
#     1.36141     -0.02937     -1.05989
#
# Obs.     :  90.00 DF  :    87.00
# Deviance : 114.02 AIC :   120.02
```

The S3 summary() Method

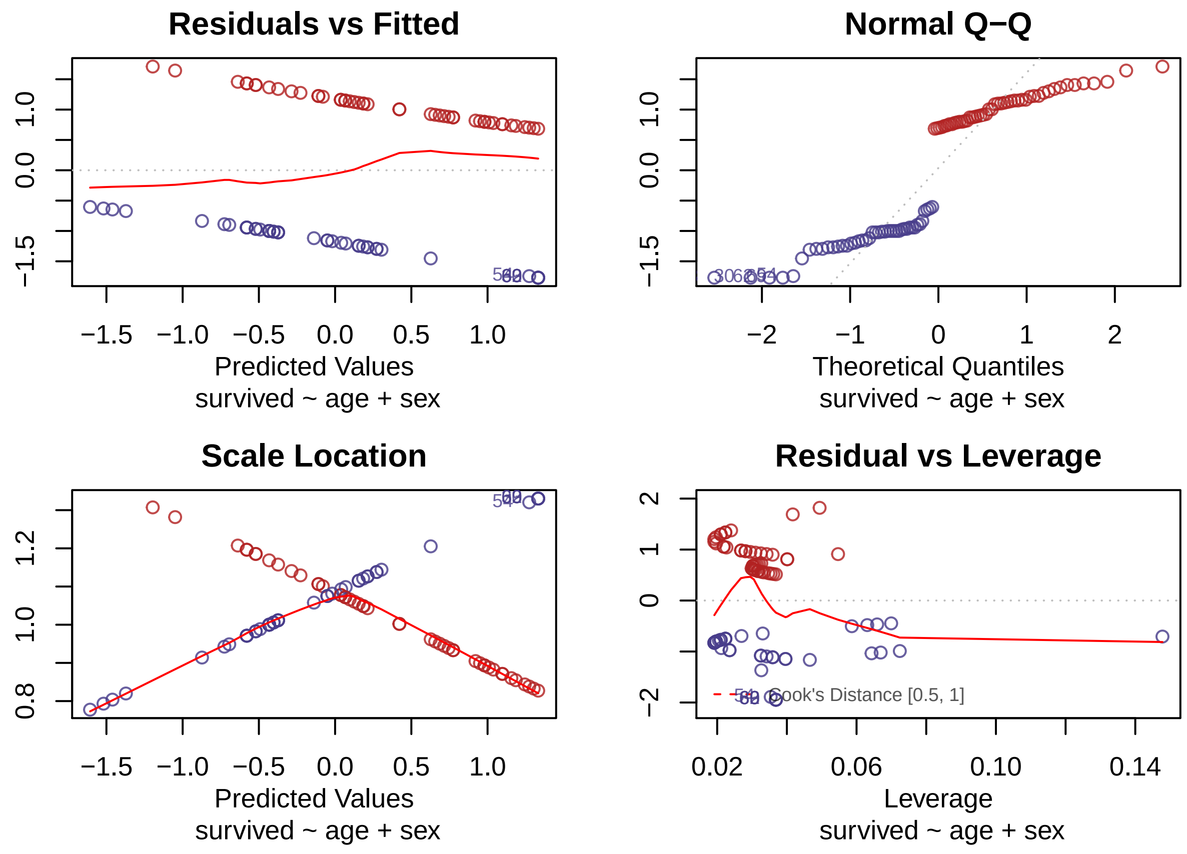
Also mimicking the `summary()` output of an glm object, the summary method includes a few other statistics and information on the model results. The choice was again to portray the results in a bit more compact form, which also makes it possible to include a few other important statistics on the model, such as Bayesian Information Criterion (BIC) and the Log-Likelihood value of the full model.

```
summary(fit)
# Call:  logitModel(formula = survived ~ age + sex, data = DonnerData)
#
# Deviance Residuals:
#      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
# -1.769900 -1.020384  0.697998  0.011996  1.096457  1.708992
#
# Coefficients:
#                Estimate  Std. error  z value  Pr(>|z|)
# (Intercept)    1.361407    0.481375   2.8282  0.004682  **
# age           -0.029366    0.014686  -1.9995  0.045550  *
# sexMale       -1.059886    0.466965  -2.2697  0.023224  *
#
# Obs.             :     90.00   DF          :    87.00
# Resid. Deviance :    114.02   AIC         :   120.02
# Null Devinance  :    124.59   BIC         :   127.52
# Log-Likelihood  :    -57.01   NR Iterat. :     5.00
```

The S3 plot() Method

The plot method, as shown below, tries to mimic the behavior of the `plot.lm` method of the `stats` package. Apart from the addition of colors to help identify the response variables.

```
op <- par(mfrow = c(2,2))
plot(fit)
```



This implementation includes a helper function that is called when plotting interactively. From the second plot onward, if the user intends to view more than one plot, the user is given the option to quit the plotting procedure before cycling through all plots. As shown in the next chunk, this is done with help from the `readline()` function, that waits for an input from the user. If the use inputs the letter q, the helper function returns `"no"` to the parent environment (that is, the `plot.logitModel` function), which in turns breaks the function.

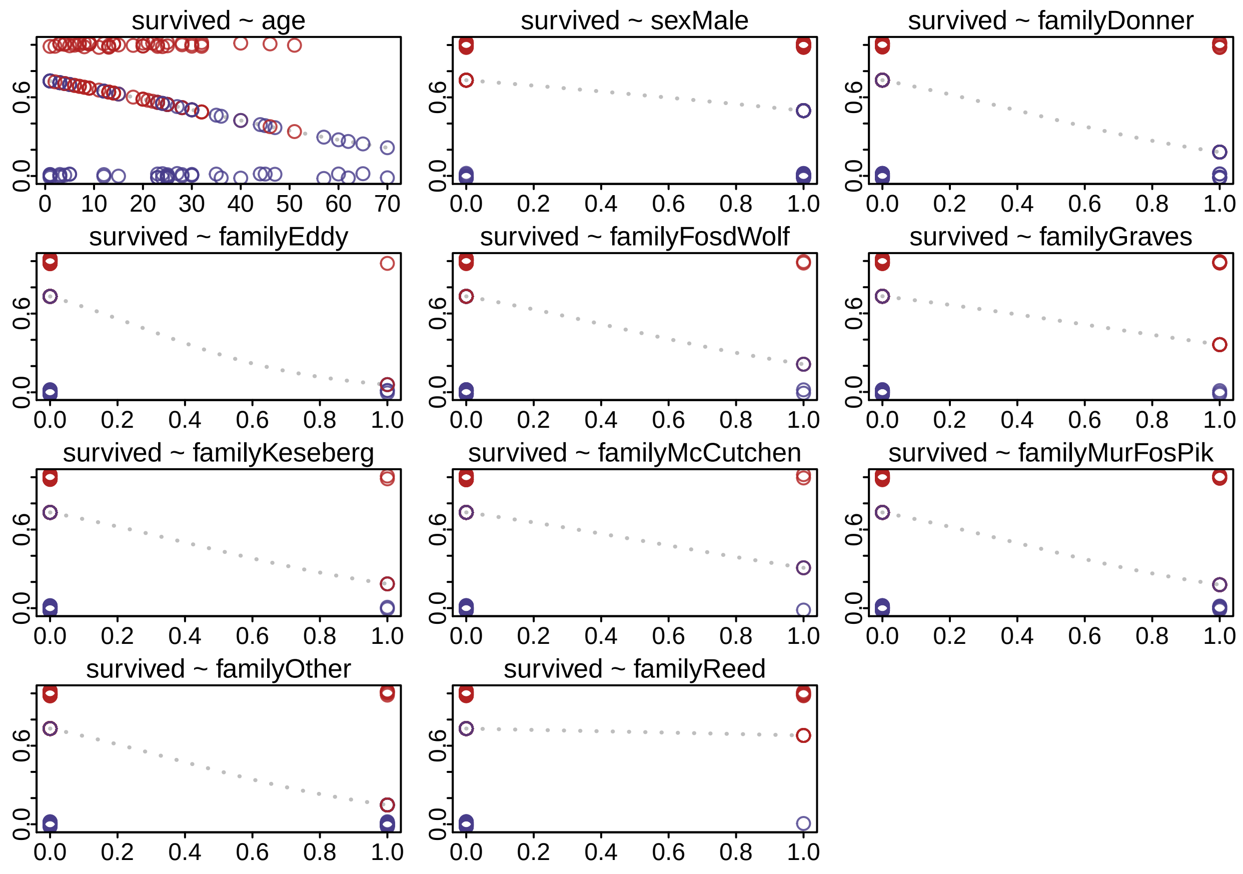
```r
plot.logitModel <- function(model, which = 1:4, col, ask,
                            cookLevels = c(.5, 1), ...) {
  ...
  # Plot 02
  if (2 %in% which) {
    if (length(which) > 1 && ask && nextPlot() == "no") {
      return(invisible())
    }
  ...
}

nextPlot <- function() {
  prompt <- "Enter anything to continue or [q] to quit: "
  UInput <- readline(prompt=prompt)
  if (UInput %in% c("q", "Q")) return(invisible("no"))
  return("yes")
}
```

The S3 pairs() Method

The functionality of the S3 `pairs()` method for the logitModel class object can be better demonstrated when modeling a bigger model, that is, with more covariates. In this case, consider adding the dummy variable *family* into the current model.

```
fit2 <- logitModel(survived ~ age + sex + family, data = DonnerData)
pairs(fit2)
```



Similar to the default `pairs()` method from the `graphics` package that plots all against all variables in order to give the user a quick overview of the data and how each variable interacts with one another. In this implementation, however, all explanatory variables are plotted against the response variable so that one can visually check the resulting coefficients of the model. This function could be greatly improved, however, by adding a confidence interval to indicate the degree of uncertainty of each estimated coefficient.

For more information on the dummy variable family, please refer to the help file via `?DonnerData`.

Implementing the Newton-Rahpson Algorithm

The implemented algorithm can be divided into three sections. First the variables are initiated, then the actual computation is done until convergence or until maximum iteration is reached and then a list of the results is returned.

```
newtonRaphson <- function(y, X, precision = 1e-7, iterMax = 25) {
  #01 initiate variables
```

```r
beta <- rep(0, times = ncol(X))
i <- 0
convergence <- FALSE
...
```

In the initiation part, the beta vector as well as a iteration counter and a convergence flag is initiated. A better implementation could optimize the algorithm by choosing are better initiating values for beta based on the data at hand.

```r
...
#02 compute coefficients until convergence or maximum iteration reached
while (i <= iterMax & convergence == FALSE) {
  # update i
  i <- i + 1

  # calculate probabilities (pi)
  eta <- X %*% beta
  pi <- as.numeric(logist(X %*% beta))

  # init / update Matrix W
  W <- diag(pi * (1 - pi))

  # calculate and update beta (eq. 23 from Czepiel, 2002)
  deltaBeta <- solve(t(X) %*% W %*% X) %*% t(X) %*% (y - pi)
  beta <- beta + deltaBeta

  # check convergence condition
  if (sum(abs(deltaBeta)) < precision) {
    convergence <- TRUE
    break
  }
}

if (convergence == FALSE) {
  stop(paste(i, "iterations reached without convergence. Increase iterMax?"))
}
```

In the above excerpt follows very closely the previously elaborated explanation over the Newton-Raphson method in Section 2, with special attention to Eq. 8, that defines each step of the procedure. Otherwise, the iteration counter `i` counts how many steps is taken, while at each step the absolute change in `deltaBeta` is checked against a precision (or tolerance) level. When the algorithm reaches the tolerance level, the loop is broken and the function carries on to the next section.

In case there is no convergence, the loop runs until the maximum iteration is reached (`iterMax`) and then the function stops throwing an error, that states that the algorithm did not converge (given the desired level of precision). The user could then run the function again with a higher maximum number of iteration, but it is also likely that there is some other problem with the data, because it is very rare that the method would not converge before the 25 (default) number of iterations.

```r
...
#03 return list of results
return(list(beta = beta,
            pi = pi,
            W = W,
              eta = eta,
              iterCount = i))
}
```

Finally, when convergence is achieved, a list of results is returned for further computation.

4 Discussion and Acknowledgements

This package implementation had the intention not only implementing a procedure using the statistical programming language `R`, but also help to familiarize with the language itself. Package creation is at the heart of the language and it is great part of the explanation why `R` is such a popular and rapidly developing language. With the intent of delving a little deeper in all areas of a package creation, I also implemented a automated test suite, which is found under the folder `/tests/testthat/` that just checks if a few results of the alternative implementation matches the `glm()` function from the `stats` package. Also in this regard the "DonnerData" data set is exported with the package as well.

I would also like to thank those responsible for this course. It was a difficult task, but very rewarding and surely great appreciated.

Appendix

Due to naive solution of the Newton-Raphson method, the algorithm doesn't converge when including *family* in the model, because all members of the Breen family survived. This seems to be the case of *perfect discrimination*, when there is no overlap in the explanatory variables relative to response variables being either 0 or 1 (see Agresti 1990, 195). That is, for all $y_i = 0$ and $y_i = 1$ observations, there is a $\bar{x}$ value that divides both set of observations.

To quickly circumvent this issue, I simply changed the survival outcome of one member of the Breen family, so that my implementation could achieve convergence. This is definitely not a very sophisticated solution, but the only intent was to show the functionality of the `pairs()` method.

Apart from, more information on the package is provided in the help file of each function.

References

A Czepiel, Scott. 2002. "Maximum Likelihood Estimation of Logistic Regression Models: Theory and Implementation."

Agresti, Alan. 1990. *Categorical Data Analysis.* A Wiley-Interscience Publication. New York [u.a.]: Wiley.

Polyak, Boris. 2007. "Newton's Method and Its Use in Optimization" 181 (September): 1086–96.

Ypma, Tjalling J. 1995. "Historical Development of the Newton-Raphson Method." *SIAM Rev.* 37 (4). Philadelphia, PA, USA: Society for Industrial; Applied Mathematics: 531–51. doi:10.1137/1037125.