Logistic Regression

Roman Schulze

2019-04-02

Contents

Abstract

This vignette explains the application of logistic regression given some appropriate dataset using the package "*myLogit*". To facilitate the readers understanding of the most important functionalities of the package some theoretical introduction is given in the first place. The section is followed by an illustration using the titanic dataset. For the plot methods the package *ggplot*2 and *gridExtra* are used throughout the vignette.

```r
# Loading packages ----
library(myLogit)
library(ggplot2)
library(gridExtra)
```

1. Introduction

Logistic regression is an appropriate statistical method when dealing with a binary classification problem. A variable y_i can take two values:

$$y_i = 1$$
$$y_i = 0$$

An example might be:

- Is it going to rain tomorrow or not
- Is student i passing the examen or not

In logisitc regression the endogenous variable y_i is modelled by probability p_i. This means that y_i occurs with some probability p_i. Applying the linear regression will lead to problems, because probabilty values might be larger than one or smaller than zero given some predictor x_i. To get rid of this problem the linear predictor from the linear regressor approch needs to be transformed by a function $G()$. In particular, the transformation needs to guarantee that each probabilty value $p_i \in (0, 1)$ (Verbeek 2012):

$$P(y_i = 1|x_i) = G(x_i, \beta) \tag{1}$$

One function that is appropriate in this scenario is the standard logistic function:

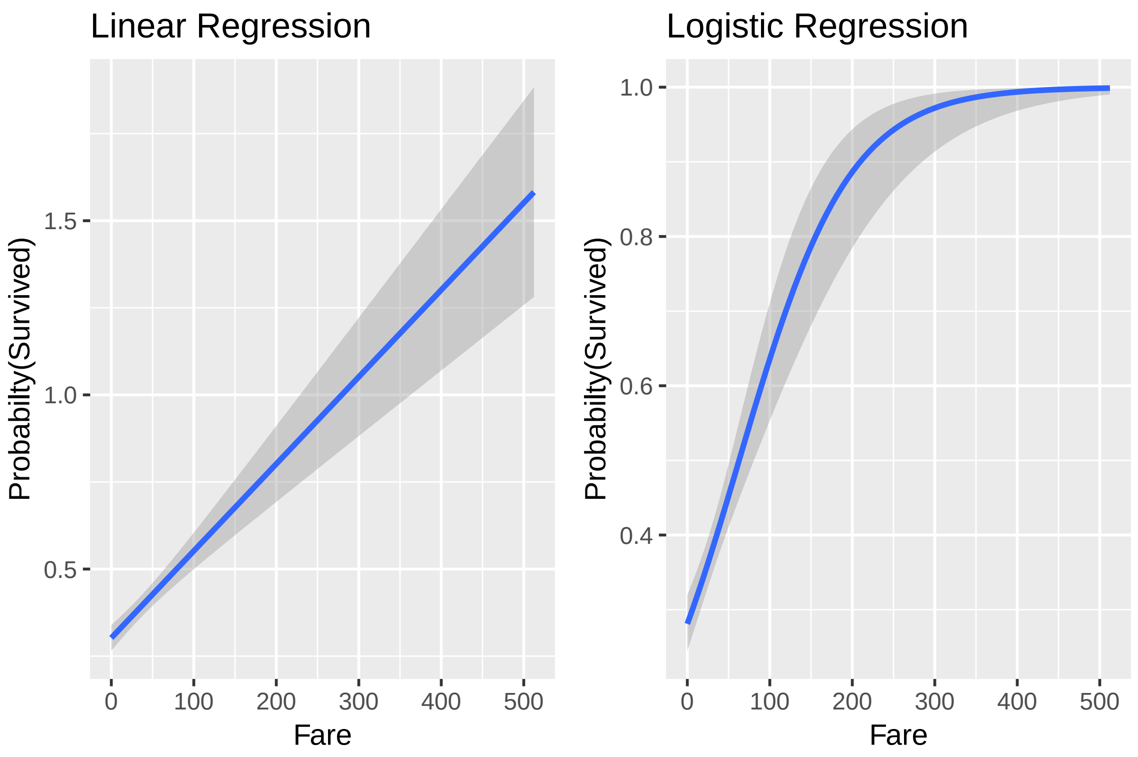
$$P(y_i = 1|x_i) = \frac{exp(x_i'\beta)}{1 + exp(x_i'\beta)} \tag{2}$$

The following graphic illustrates the issue which arises when applying the linear regression approach: the probabilty for Fare larger than ca 280 leads to probabilty values larger than one while in case of logistic regression the probabilty values remain within the interval of zero and one.[1]

```r
# Plot linear model
plot1 <- ggplot2::ggplot(myLogit::myData, ggplot2::aes(y = Survived, x = Fare)) +
  ggplot2::geom_smooth(method = "lm") +
  ggplot2::ggtitle("Linear Regression") +
  ggplot2::labs(y = "Probabilty(Survived)")

# Plot logistic regression
plot2 <- ggplot2::ggplot(myData, ggplot2::aes(y = Survived, x = Fare)) +
  ggplot2::geom_smooth(method = "glm", method.args = list(family = "binomial")) +
  ggplot2::ggtitle("Logistic Regression") +
  ggplot2::labs(y = "Probabilty(Survived)")

# Plot next to each other using gridExtra package
grid.arrange(plot1, plot2, ncol = 2)
```



[1]The data used here is from the prominent Titanic dataset, which is included in the package.

2. Estimation & Implementation

Logistic regression is a parametric model. The set of parameters to be determined is the vector of regression coefficients β. In comparison to linear regression there is no analytical solution to find the optimal values for β. Instead a numerical approach is applied to identify the parameters of interest. The log likelihood takes the following form (Verbeek 2012)[^2]:

$$logL(\beta) = \sum_{i=1}^{N} y_i logL(x_i'\beta) + \sum_{i=1}^{N} (1 - y_i) log(1 - L(x_i'\beta)) \tag{3}$$

In case of logisitc regression the function $F()$ is given by the standard logistic function. Substituting and taking the derivative w.r.t. β leads to the following optimality condition (Verbeek 2012):

$$\frac{\partial logL(\beta)}{\partial \beta} = \sum_{i=1}^{N} [y_i - \frac{exp(x_i'\beta)}{1 + exp(x_i'\beta)}] x_i = 0 \tag{4}$$

Equation (4) is the *gradient* vector of the log likelihood function, also known as the score vector. The maximum likelihood estimator for β solves the equation above. As mentioned above, there is no closed form solution for beta. Hence, a numerical search process is applied to determine β. This is the *Fisher scoring algorithm* (Gill 2001), an iterative search process. The algorithm is implemented in R by the following function:

2.1 Fisher Scoring algorithm

```r
# Fisher scoring algorithm ----
fScore <- function(X, y, maxIter = 100, rate = 1, accuracy = 0.01, ...) {
  # Initial beta vector
  beta <- rep(0, ncol(X))
  # Starting value for change in beta
  delta <- 1
  #iteration counter
  iter <- 0
  # convergence
  convergence <- FALSE
  # probability
  p <- logistic(X %*% beta)
  # empty vecotr to store log-Likelihood
  l <- c()
  # Optimization using while loop with two conditions
  while (convergence == FALSE && iter <= maxIter) {
    # Update iteration
    iter <- iter + 1
    # Calculate log - Likelihood for each combination
    l[iter] <- logLike(X, y, beta)
    # update beta
    betaNew <- beta + rate * solve(fisher(X, p)) %*% score(X, y, beta)
    pnew <- logistic(X %*% betaNew)
    # Change in beta
    delta <- abs(betaNew - beta)
    beta <- betaNew
    p <- pnew
    # check convergence condition
    if (sum(delta) < accuracy) {
      convergence <- TRUE
      break
    }
  }
  # if no convergence...
  if (convergence == FALSE) {
    stop(paste(iter, "iterations reached without convergence. Increase maxIter?"))
  }
  # return list with results
  return(list(coefficients = beta,
              prob = as.vector(p),
              iterations = iter,
              logLikelihood = l,
              convergence = convergence))
}
```

The function `fScore()` starts at a chosen point β_{start} on the log likelihood function. From here one moves with step size δ uphill, because the goal is to find the maximum of the log likelihood function.
In this application the starting values for the parameter vector β is chosen to be zero, while δ is equal to one. The idea of the algorithm works as follows:

$$\beta_1 = \beta_{Null} + \delta P_1 \gamma_1 \tag{5}$$

$$\beta_2 = \beta_1 + \delta P_2 \gamma_2 \tag{6}$$

$$...$$

$$...$$

$$\beta_{t+1} = \beta_t + \delta P_t \gamma_t \tag{7}$$

P_t is the Fisher Information Matrix (negative of the Hessian Matrix). γ_t is the gradient of the log likelihood with respect to β. The algorithm keeps running until γ_t is close to zero. This implies that

$$\beta_{t+1} - \beta_t < \theta = 0.01 \tag{8}$$

which is defined as the accuarcy in the ‘grad()’ function. If the condition is reached the algorithm converged and the search process is completed. θ has a default value of 0.01 but can be adjusted manually.
As the search process is iterative it is implemented using a **while loop**. The loop keeps running until either convergence turns from FALSE to TRUE or until the maximum number of iterations is reached. In the latter case the algorithm will ask you to increase the maximum number of iterations which is set to 100 by default. If convergence turns TRUE a list is generated, including five elements:

1. the estimated parameter vector β
2. the vector of probablites
3. the number of iterations

4. The log likelihood values for each iteration

5. if convergence is TRUE

The probabilites are calculated using the following equation (Verbeek 2012):

$$\hat{p_i} = \frac{exp(x_i'\beta)}{1 + exp(x_i'\beta)} \tag{9}$$

```
# Example ----
set.seed(1)
X <- replicate(2, rnorm(100))
# Add a vector of ones as first column to estimate Intercept
X <- cbind(1, X)
z <- 2 * X[, 2] + 3 * X[, 3]
# The function logstic calculates the estimated probabilities according to equation (9)
y <- rbinom(100, 1, logistic(z))
fScore(X, y)
#> $coefficients
#>           [,1]
#> [1,] 0.1527442
#> [2,] 1.6290416
#> [3,] 2.8845411
#>
#> $prob
#>   [1] 0.065543669 0.639545444 0.021121409 0.961110441 0.231715426
#>   [6] 0.980429737 0.953209680 0.981674042 0.900151984 0.989094901
#>  [11] 0.686044394 0.367299840 0.963462497 0.004810872 0.800129706
#>  [16] 0.258548758 0.310739794 0.707894264 0.948634373 0.647648104
#>  [21] 0.547442231 0.995037780 0.414656243 0.026443881 0.705465524
#>  [26] 0.892546783 0.422320991 0.086928365 0.069625276 0.474576317
#>  [31] 0.926873414 0.152724327 0.910307882 0.013193151 0.230362692
#>  [36] 0.006997003 0.204606469 0.187288671 0.515928896 0.774145381
#>  [41] 0.003549757 0.958270043 0.028900928 0.431066607 0.014947000
#>  [46] 0.040482410 0.998850389 0.810750591 0.023188004 0.041316912
#>  [51] 0.890889418 0.289500986 0.447929747 0.012520128 0.141432726
#>  [56] 0.568923827 0.919764329 0.034215438 0.051534453 0.995154175
#>  [61] 0.994990011 0.354442821 0.986998402 0.940211045 0.054976147
#>  [66] 0.998913697 0.028658043 0.172384785 0.496468482 0.986489318
#>  [71] 0.999492156 0.332123408 0.921714612 0.169174295 0.054530785
#>  [76] 0.628865670 0.845871668 0.997851156 0.962217882 0.935634758
#>  [81] 0.013054991 0.941067369 0.937396143 0.001411694 0.932341032
#>  [86] 0.559023529 0.997782689 0.072253414 0.380967615 0.110706811
#>  [91] 0.224104193 0.963737353 0.483089132 0.975604697 0.321474803
#>  [96] 0.123422442 0.902928972 0.023860463 0.342124108 0.152170927
#>
#> $iterations
#> [1] 6
#>
#> $logLikelihood
#> [1] -69.31472 -39.20062 -34.20027 -33.26950 -33.21167 -33.21133
#>
#> $convergence
#> [1] TRUE
```

2.2 Goodness-of-fit

To evaluate the quality of the estimated model there exists a variety of goodness of fit measures.
Mc Faddens R^2 is implemented by the following formula, where l_1 is the log likelihood of the model of interest and l_0 the log likelihood value, if only the intercept is included (Verbeek 2012):

$$McFaddenR^2 = 1 - \frac{log(l_1)}{log(l_0)} \tag{10}$$

Another popular measure is the AIC (Sakamoto 1986),

$$AIC = -2l(\beta) - 2k \tag{11}$$

where $l(\beta)$ is the log likelihood value and k corresponds to the number of estimated parameters.
Furthermore the predictive quality of the model is given by the *accuracy* which compares the predicted outcome to the actual realization.

Table 1: *Cross tabulation of actual and predicted outcomes*

		$\hat{y}_i$		
		0	1	$Total$
y_i	0	n_{00}	n_{01}	N_0
	1	n_{10}	n_{11}	N_1
	$Total$	n_0	n_1	N

The accuracy is calculated by summing up the correct predictions divided by the total number of observations:

$$Accuracy = \frac{n_{00} + n_{11}}{N} \tag{12}$$

3. Package & Usage

The package *myLogit* offers a variety of functions, all linked to the logsistic regression model. Each function obtains an own help page for further information and examples.
The functions `print()`, `summary()`, `predict()` and `plot()` are implemented using the S3 class system and work for objects of class *logit*.

3.1 Dataset

To illustrate the most important features of the package the prominent Titanic dataset is included within the package.[2]

```
# dataset
help(myData)
head(myData)
#>   Survived Pclass    Sex      Age SibSp Parch    Fare Embarked
#> 1        0      3   male 22.00000     1     0  7.2500        S
#> 2        1      1 female 38.00000     1     0 71.2833        C
#> 3        1      3 female 26.00000     0     0  7.9250        S
#> 4        1      1 female 35.00000     1     0 53.1000        S
#> 5        0      3   male 35.00000     0     0  8.0500        S
#> 6        0      3   male 29.69912     0     0  8.4583        Q
```

3.2 Model Estimation & results

To estimate the model, use the function `logit()`. In this example the dataset from the package is used. The dependent binary variable is Survived. The estimated model includes three predictors: Pclass, Age and the gender of observation i.
Executing the command line creates a list consisting of 19 elements being of class *logit*. The `summary()` function returns a table including model specific information. The goodness of fit measures mentioned in the previous section are also part of the summary table:

```
# Estimate model
mod <- logit(Survived ~ Pclass + Age + Sex , data = myData)
# Check class
class(mod)
#> [1] "logit"
# Print
print(mod)
#> Call:
#> logit.formula(formula = Survived ~ Pclass + Age + Sex, data = myData)
#>
#> Coefficients:
#> (Intercept)      Pclass         Age     Sexmale
#>     4.72513    -1.16518    -0.03366    -2.60470
#>
#> Degrees of freedom:  888 Total (i.e. Null);  885 Residual
#> Null Deviance: 1183
#> Resiudal Deviance: 804.7
#> AIC: 812.7
#> Accuracy: 0.8
```

[2]The dataset is available at: https://cran.r-project.org/web/packages/titanic/index.html

The `print()` and the `summary()` methods are inspired by the ones from the `stats::glm()` function. While the `print()` function imitates the print method from the `glm()` function, the `summary()` of the *myLogit* package function produces two additional measures of fit: *Mc Fadden* R^2 and the *Accuracy* measure.

```
# Summary of model
summary(mod)
#> Call:
#> logit.formula(formula = Survived ~ Pclass + Age + Sex, data = myData)
#>
#> Deviance Residuals:
#>     Min      1Q  Median      3Q     Max
#> -2.6543 -0.6575 -0.4218  0.6387  2.4265
#>
#> Coefficients:
#>               Estimate   Std.Err z.value  P(>|z|)
#> (Intercept)  4.725127  0.449779  10.505  < 2e-16 ***
#> Pclass      -1.165181  0.118964  -9.794  < 2e-16 ***
#> Age         -0.033661  0.007364  -4.571 4.86e-06 ***
#> Sexmale     -2.604696  0.186758 -13.947  < 2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Null deviance: 1182.82 on 888 degrees of freedom
#> Residual deviance: 804.72 on 885 degrees of freedom
#>
#> Number of Fisher of Scoring Iterations: 5
#> AIC: 812.72
#> Accuracy: 0.8
#> McFadden R-squared: 0.32
```

In comparison the results from `stats::glm()` look as follows:

```
# Estimate same model using glm() function
test <- stats::glm(Survived ~ Pclass + Age + Sex , data = myData, family = binomial)
summary(test)
#>
#> Call:
#> stats::glm(formula = Survived ~ Pclass + Age + Sex, family = binomial,
#>     data = myData)
#>
#> Deviance Residuals:
#>     Min       1Q   Median       3Q      Max
#> -2.6543  -0.6575  -0.4218   0.6387   2.4265
#>
#> Coefficients:
#>               Estimate Std. Error z value Pr(>|z|)
#> (Intercept)  4.725127   0.449779  10.505  < 2e-16 ***
#> Pclass      -1.165181   0.118964  -9.794  < 2e-16 ***
#> Age         -0.033661   0.007364  -4.571 4.86e-06 ***
#> Sexmale     -2.604696   0.186758 -13.947  < 2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
```

```
#>     Null deviance: 1182.82  on 888  degrees of freedom
#> Residual deviance:  804.72  on 885  degrees of freedom
#> AIC: 812.72
#>
#> Number of Fisher Scoring iterations: 5
```

The summary tables show identical results with respect to all included features.

3.3 Prediction

The predict function offers the user to make predictions for objects of class *logit*. The function call `predict()` for objects of class *logit* returns a matrix, which has n (number of observations in dataset) rows and three columns:

1. Linear Predictor

2. Probability value

3. Predicited class

In Logistic Regression the predicted class $\hat{y_i}$ depends on the probabilty value p_i. Observations having a probability values larger than $t = 0.5$ will be assigned to class 1. Consequently values equal or smaller than t will take the value 0. The function allows to determine the threshold value t manually.

```
# Predict model
head(predict(mod, t = 0.5))
#>   linear predictor probability predicted class
#> 1       -2.1156563  0.10758439               0
#> 2        2.2808254  0.90727651               1
#> 3        0.3543953  0.58768303               1
#> 4        2.3818086  0.91542956               1
#> 5       -2.5532502  0.07220844               0
#> 6       -2.3748168  0.08511331               0

# Compare predictions results
all(round(predict(mod)[, 1], 7) %in% round(stats::predict(test), 7))
#> [1] TRUE
```

Comparing the results from the `predict()` functions show similar yet not identical results. Up to the seventh decimal place the results coincide. The fact that they do not match completely might arise from a different stopping criteria in the Fisher Scoring algorithm.

3.4 Plot

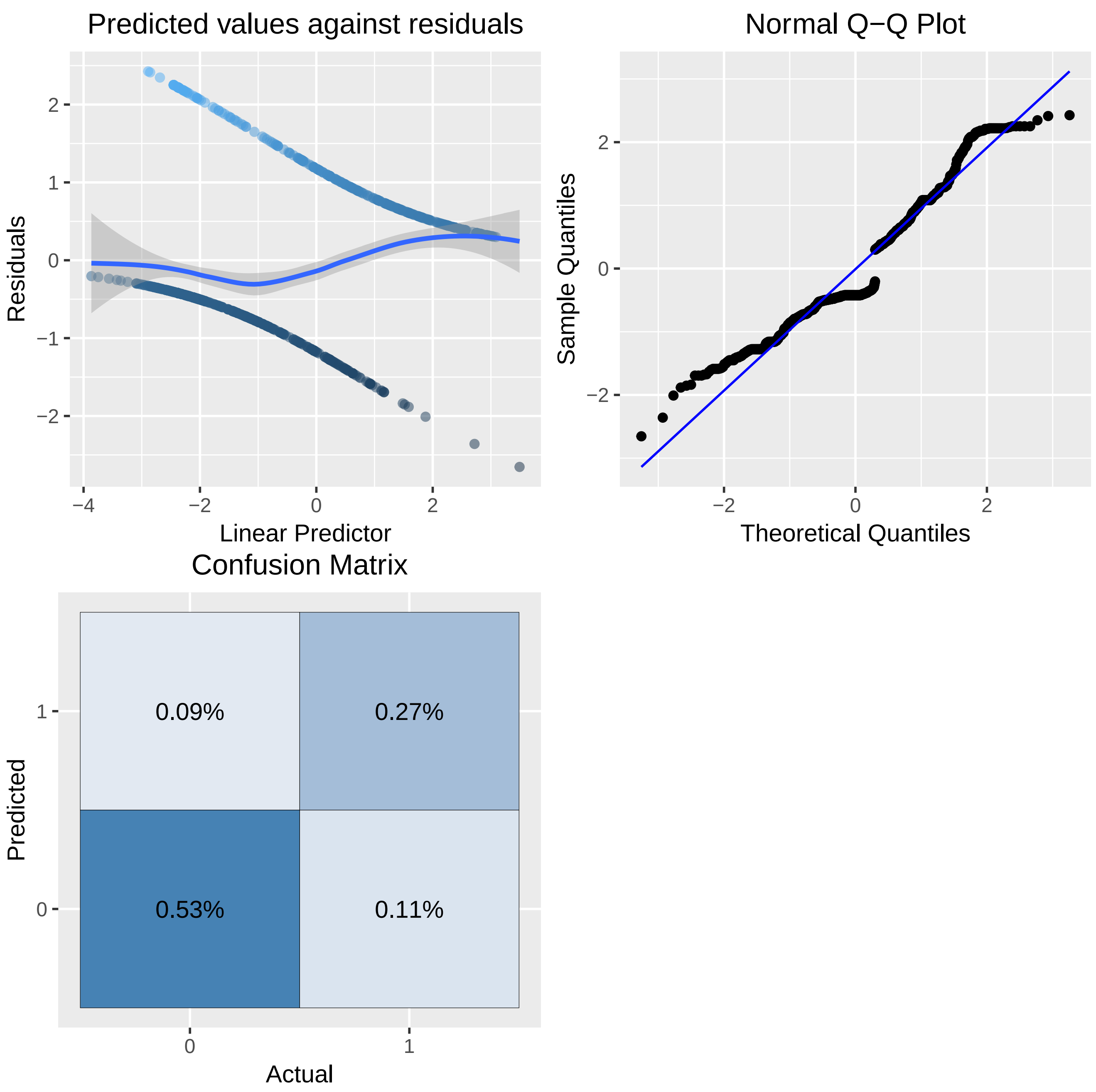
The `plot()` function for objects of class *logit* returns three plots. Each plot is constructed using the package *ggplot2* by Wickham (2016):

1. The first plot links the deviance residuals and the linear predictor. The smoothing line is added using the method "loess".

2. The second plot is a Normal Q-Q plot, to check whether the residuals follow a normal distribution

3. The third plot returns a confusion matrix

```
# Plot
plot(mod)
```



The Confusion matrix (James et al. 2014) or cross tabulation table was introduced in section 2.2 and allows to evaluate the predictive performance of logistic regression. Summing up the (0,0) and (1,1) squares deliveres the accuracy from the summary table.

Furthermore the package provides the option to plot a ROC curve. A ROC curve plots the true positive rate against the false positive rate for different threshold values. Therefore the function `ggRoc()` creates a list of class *roc* using the the variables y and *prob* from an estimated model. The list includes a dataframe having three columns:

1. True positive Rate

2. False positive Rate

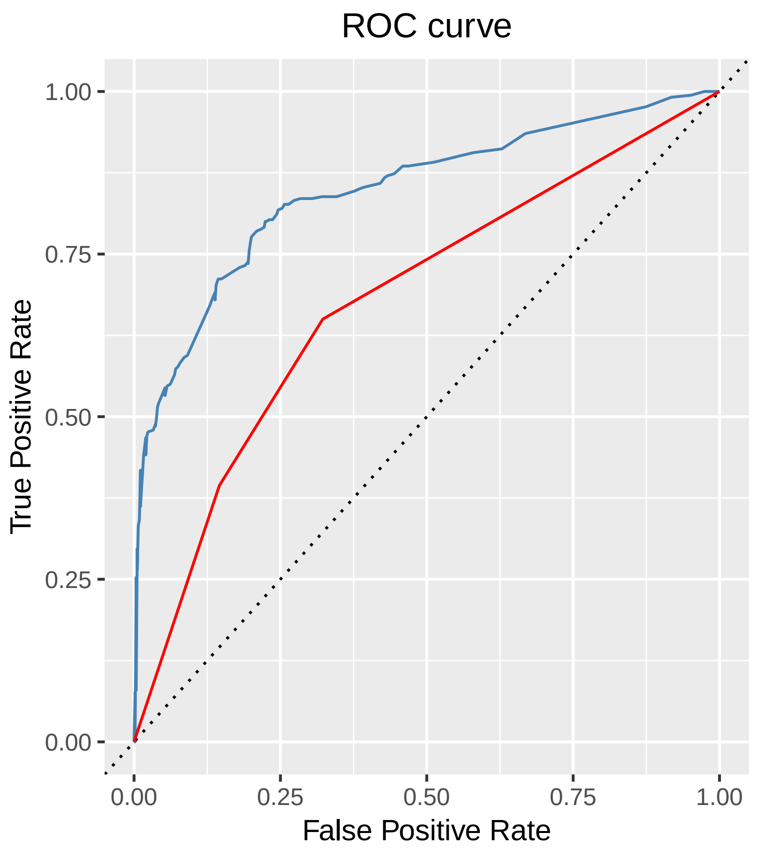
3. threshold values

To plot the ROC curve a `plot()` function for objects of class *roc* is included in the package. In the following example the plot for the estimated model from this section is saved under *a*. Additionaly the ROC curve for a different model specificaion is added to illustrate the difference in predicitive quality.

```
# ROC curve for model mod
obj <- ggRoc(mod$y, mod$prob)
a <- plot(obj)

# Add ROC curve for model which only includes Pclass as a predictor
mod1 <- logit(Survived ~ Pclass, data = myData)
obj1 <- ggRoc(mod1$y, mod1$prob)

# Overlay existing plot a
a <- a + ggplot2::geom_line(data = obj1[[1]],
                            ggplot2::aes(x = obj1[[1]]$falsePositive,
                                         y = obj1[[1]]$truePositive),
                            colour = "red")
# Plot a
a
```



The true positive rate is the percentage of known true instances which the model does identify as true . The false positive rate is the percentage of known false instances which the model identifies as true. Ideally, the true positive rate is 1 and the false positive rate is zero. The the predictive performance of a model is given by the area under the curve (AUC). A higher AUC value reflects a better result (James et al. 2014).

Hence, the original model performs better than the second specification, which includes only one predictor.

4. Conclusion

The package *myLogit* offers a comprehensive set of tools/ methods to the user to estimate a logistic regression model, given an appropiate dataset. The estimation results are very similiar to those of the `stats::glm()` function. While the first two plots are identical, the package *myLogit* returns a confusion matrix as the third plot and provides the option to plot a ROC curve to further evaluate the estimated models. The package could be extended by giving the user the option to estimate a model with a multiclass dependent variable.

5. References

Gill, Jeff. 2001. *Generalized Linear Models*. Quantitative Applications in the Social Sciences. Thousand Oaks: SAGE Publications, Inc.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2014. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated.

Sakamoto, Yosiyuki. 1986. *Akaike Information Criterion Statistics / Y. Sakamoto, M. Ishiguro, and G. Kitagawa.* Tokyo: KTK Scientific Publ. u.a. https://primo.fu-berlin.de/FUB:FUB_ALMA_DS21924435620002883.

Verbeek, M. 2012. *A Guide to Modern Econometrics 4th Edition*. Wiley. https://books.google.de/books?id=4UTeOZDyoeUC.

Wickham, Hadley. 2016. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. http://ggplot2.org.