Generalised linear models



## Generalised linear models

Sometimes the data that we are handling are not continuously or normally distributed (and cannot be easily transformed into such a distribution). In these situations we may opt to use a generalised linear model. We will cover logistic regression and poisson regression and then also survival analysis.

## Logistic regression

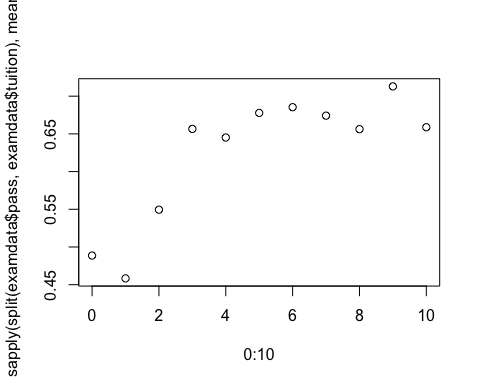
This is characterised by an outcome which can take on two values (and two values only). Examples are: success/fail, dead/alive, yes/no, present/absent. These types of random variables are governed by the binomial distribution.

We can explore the examdata file which contains test scores (expressed as a percentage) on 1,000 students.

examdatpath <- "//Users//Mark//Documents//Cardiff//Teaching//R//learnR//Handbook//Datasets//examdat.csv"  
examdata <- read.csv(file=examdatpath)

The file contains information on the number of hours of tuition they received (between 0 and 10 hours) and the students star signs. We might hypothesise that hours of tuition will be positively associated with test scores, while star signs would have no bearing at all. In addition, we are strictly interested in the pass/fail rate, where a pass is 60% or greater. We could explore the situation graphically as follows.

examdata$pass <- I(examdata$testscore>=60)  
plot(0:10, sapply(split(examdata$pass,examdata$tuition),mean))

 This certainly indicates that more tuition results in higher chances of passing the exam. We could fit this formally as

exammodel <- glm(examdata$pass~tuition,  
data=examdata,  
family="binomial")  
summary(exammodel)

##   
## Call:  
## glm(formula = examdata$pass ~ tuition, family = "binomial", data = examdata)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -1.6017 -1.2932 0.8359 0.9969 1.1370   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 0.09575 0.12325 0.777 0.437   
## tuition 0.08623 0.02122 4.064 4.82e-05 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 1319.0 on 999 degrees of freedom  
## Residual deviance: 1302.2 on 998 degrees of freedom  
## AIC: 1306.2  
##   
## Number of Fisher Scoring iterations: 4

Due to the nature of the link function used in a logistic regression we need to transform the coefficients returned into odds ratios. We do this by taking the exponetial of them as follows

library(magrittr)  
#This is one way to get the exponential of the coefficients  
exp(coef(exammodel))

## (Intercept) tuition   
## 1.100485 1.090054

#This is a better way  
exammodel %>% #our logisitic regression model object  
 coef() %>% #get the coefficients  
 exp() #exponentiate them

## (Intercept) tuition   
## 1.100485 1.090054

We may want to round these to three decimal places we could type

round(exp(coef(exammodel)),3)

## (Intercept) tuition   
## 1.10 1.09

Try doing the above using pipes.

We could get confidence intervals in the same way as before, but we need to again transform them into odds ratios.

exammodel %>% #This is our model object  
 confint() %>% #This calculates the confidence intervals  
 exp() #This exponentiates

## Waiting for profiling to be done...

## 2.5 % 97.5 %  
## (Intercept) 0.8645884 1.402132  
## tuition 1.0458476 1.136604

We can include information on the star sign now.

exammodelplus <- glm(pass~tuition + starsign,  
data=examdata,  
family="binomial")  
  
summary(exammodelplus)

##   
## Call:  
## glm(formula = pass ~ tuition + starsign, family = "binomial",   
## data = examdata)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -1.6900 -1.3041 0.8487 0.9722 1.2342   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -0.05836 0.21965 -0.266 0.790   
## tuition 0.08608 0.02141 4.020 5.82e-05 \*\*\*  
## starsignCancer -0.07418 0.27637 -0.268 0.788   
## starsignCapricorn 0.08642 0.28416 0.304 0.761   
## starsignGemini 0.35134 0.27422 1.281 0.200   
## starsignLeo 0.26509 0.27983 0.947 0.343   
## starsignPisces 0.24070 0.26939 0.894 0.372   
## starsignSagittarius 0.20540 0.27182 0.756 0.450   
## starsignTaurus 0.04569 0.27351 0.167 0.867   
## starsignVirgo 0.25474 0.27002 0.943 0.345   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 1319 on 999 degrees of freedom  
## Residual deviance: 1298 on 990 degrees of freedom  
## AIC: 1318  
##   
## Number of Fisher Scoring iterations: 4

exammodelplus %>%  
 coef() %>%  
 exp()

## (Intercept) tuition starsignCancer   
## 0.9433146 1.0898962 0.9285058   
## starsignCapricorn starsignGemini starsignLeo   
## 1.0902665 1.4209699 1.3035421   
## starsignPisces starsignSagittarius starsignTaurus   
## 1.2721450 1.2280129 1.0467488   
## starsignVirgo   
## 1.2901324

exammodelplus %>%  
 confint() %>%  
 exp()

## Waiting for profiling to be done...

## 2.5 % 97.5 %  
## (Intercept) 0.6140337 1.455044  
## tuition 1.0452889 1.136885  
## starsignCancer 0.5396725 1.597244  
## starsignCapricorn 0.6252403 1.908585  
## starsignGemini 0.8313964 2.440405  
## starsignLeo 0.7545186 2.264429  
## starsignPisces 0.7507883 2.162221  
## starsignSagittarius 0.7213162 2.097298  
## starsignTaurus 0.6122921 1.792024  
## starsignVirgo 0.7605903 2.195974

We can create our own function here if we want to

logregressout <- function(logregressmodel){  
OR <- logregressmodel %>%  
 coef() %>%  
 exp() %>%  
 round(2)  
   
OR.CI <- logregressmodel %>%  
 confint() %>%  
 exp() %>%  
 round(2)  
   
return(data.frame(OR=OR,OR.CI=OR.CI))  
}  
logregressout(exammodelplus)

## Waiting for profiling to be done...

## OR OR.CI.2.5.. OR.CI.97.5..  
## (Intercept) 0.94 0.61 1.46  
## tuition 1.09 1.05 1.14  
## starsignCancer 0.93 0.54 1.60  
## starsignCapricorn 1.09 0.63 1.91  
## starsignGemini 1.42 0.83 2.44  
## starsignLeo 1.30 0.75 2.26  
## starsignPisces 1.27 0.75 2.16  
## starsignSagittarius 1.23 0.72 2.10  
## starsignTaurus 1.05 0.61 1.79  
## starsignVirgo 1.29 0.76 2.20

We can check model performance simply by exploring predicted values versus observed. We can take a predicted probability of an event of >0.5 as meaning that we predict that case will have an event.

examdata$pred.pass <- fitted(exammodelplus)>0.5  
table(examdata$pred.pass,examdata$pass)

##   
## FALSE TRUE  
## FALSE 23 20  
## TRUE 348 609

## Poisson regression

Poisson regression is characterised by outcome variables which are non-negative, discrete and integer values. This is taken from here <https://stats.idre.ucla.edu/r/dae/poisson-regression/>

## Examples of Poisson regression

Example 1. The number of persons killed by mule or horse kicks in the Prussian army per year. Ladislaus Bortkiewicz collected data from 20 volumes of Preussischen Statistik. These data were collected on 10 corps of the Prussian army in the late 1800s over the course of 20 years.

Example 2. The number of people in line in front of you at the grocery store. Predictors may include the number of items currently offered at a special discounted price and whether a special event (e.g., a holiday, a big sporting event) is three or fewer days away.

Example 3. The number of awards earned by students at one high school. Predictors of the number of awards earned include the type of program in which the student was enrolled (e.g., vocational, general or academic) and the score on their final exam in math.

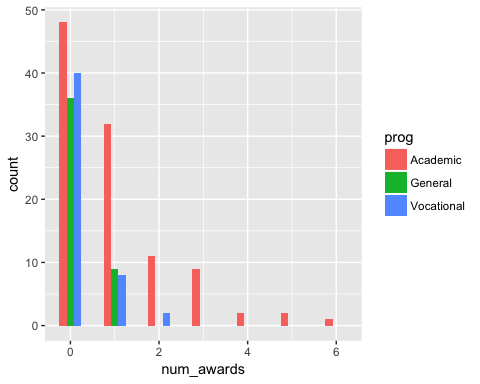
For the purpose of illustration, we have simulated a data set for Example 3 above. In this example, num\_awards is the outcome variable and indicates the number of awards earned by students at a high school in a year, math is a continuous predictor variable and represents students’ scores on their math final exam, and prog is a categorical predictor variable with three levels indicating the type of program in which the students were enrolled. It is coded as 1 = “General”, 2 = “Academic” and 3 = “Vocational”. Let’s start with loading the data and looking at some descriptive statistics.

poissonpath <- "/Users/Mark/Documents/GitHub/learnR/data/poisson.csv"  
  
p <- read.csv(file=poissonpath)

Each variable has 200 valid observations and their distributions seem quite reasonable. The unconditional mean and variance of our outcome variable are not extremely different. Our model assumes that these values, conditioned on the predictor variables, will be equal (or at least roughly so).

We can use the tapply function to display the summary statistics by program type. The table below shows the average numbers of awards by program type and seems to suggest that program type is a good candidate for predicting the number of awards, our outcome variable, because the mean value of the outcome appears to vary by prog. Additionally, the means and variances within each level of prog–the conditional means and variances–are similar. A conditional histogram separated out by program type is plotted to show the distribution.

library(ggplot2)  
  
ggplot(p, aes(num\_awards, fill = prog)) +  
 geom\_histogram(binwidth=.5, position="dodge")

 We are ready to perform our Poisson model analysis using the glm function. We fit the model and store it in the object m1 and get a summary of the model at the same time

summary(m1 <- glm(num\_awards ~ prog + math, family="poisson", data=p))

##   
## Call:  
## glm(formula = num\_awards ~ prog + math, family = "poisson", data = p)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.2043 -0.8436 -0.5106 0.2558 2.6796   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -4.16327 0.66288 -6.281 3.37e-10 \*\*\*  
## progGeneral -1.08386 0.35825 -3.025 0.00248 \*\*   
## progVocational -0.71405 0.32001 -2.231 0.02566 \*   
## math 0.07015 0.01060 6.619 3.63e-11 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for poisson family taken to be 1)  
##   
## Null deviance: 287.67 on 199 degrees of freedom  
## Residual deviance: 189.45 on 196 degrees of freedom  
## AIC: 373.5  
##   
## Number of Fisher Scoring iterations: 6

Cameron and Trivedi (2009) recommended using robust standard errors for the parameter estimates to control for mild violation of the distribution assumption that the variance equals the mean. We use R package sandwich below to obtain the robust standard errors and calculated the p-values accordingly. Together with the p-values, we have also calculated the 95% confidence interval using the parameter estimates and their robust standard errors.

library(sandwich)  
cov.m1 <- vcovHC(m1, type="HC0")  
std.err <- sqrt(diag(cov.m1))  
r.est <- cbind(Estimate= coef(m1), "Robust SE" = std.err,  
"Pr(>|z|)" = 2 \* pnorm(abs(coef(m1)/std.err), lower.tail=FALSE),  
LL = coef(m1) - 1.96 \* std.err,  
UL = coef(m1) + 1.96 \* std.err)  
  
r.est

## Estimate Robust SE Pr(>|z|) LL UL  
## (Intercept) -4.1632653 0.64809429 1.328638e-10 -5.43353007 -2.89300044  
## progGeneral -1.0838591 0.32104816 7.354745e-04 -1.71311353 -0.45460476  
## progVocational -0.7140499 0.29864225 1.680312e-02 -1.29938873 -0.12871111  
## math 0.0701524 0.01043516 1.783975e-11 0.04969947 0.09060532

The output begins with echoing the function call. The information on deviance residuals is displayed next. Deviance residuals are approximately normally distributed if the model is specified correctly.In our example, it shows a little bit of skeweness since median is not quite zero. Next come the Poisson regression coefficients for each of the variables along with the standard errors, z-scores, p-values and 95% confidence intervals for the coefficients. The coefficient for math is .07. This means that the expected log count for a one-unit increase in math is .07. The indicator variable progAcademic compares between prog = “Academic” and prog = “General”, the expected log count for prog = “Academic” increases by about 1.1. The indicator variable prog.Vocational is the expected difference in log count ((approx .37)) between prog = “Vocational” and the reference group (prog = “General”). The information on deviance is also provided. We can use the residual deviance to perform a goodness of fit test for the overall model. The residual deviance is the difference between the deviance of the current model and the maximum deviance of the ideal model where the predicted values are identical to the observed. Therefore, if the residual difference is small enough, the goodness of fit test will not be significant, indicating that the model fits the data. We conclude that the model fits reasonably well because the goodness-of-fit chi-squared test is not statistically significant. If the test had been statistically significant, it would indicate that the data do not fit the model well. In that situation, we may try to determine if there are omitted predictor variables, if our linearity assumption holds and/or if there is an issue of over-dispersion.

We can drop the prog variable if we want

## update m1 model dropping prog  
m2 <- update(m1, . ~ . - prog)  
## test model differences with chi square test  
anova(m2, m1, test="Chisq")

## Analysis of Deviance Table  
##   
## Model 1: num\_awards ~ math  
## Model 2: num\_awards ~ prog + math  
## Resid. Df Resid. Dev Df Deviance Pr(>Chi)   
## 1 198 204.02   
## 2 196 189.45 2 14.572 0.0006852 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

## calculate and store predicted values

p$phat <- predict(m1, type="response")

We can also graph the predicted number of events with the commands below. The graph indicates that the most awards are predicted for those in the academic program (prog = 2), especially if the student has a high math score. The lowest number of predicted awards is for those students in the general program (prog = 1). The graph overlays the lines of expected values onto the actual points, although a small amount of random noise was added vertically to lessen overplotting.

## create the plot

ggplot(p, aes(x = math, y = phat, colour = prog)) + geom\_point(aes(y = num\_awards), alpha=.5, position=position\_jitter(h=.2)) + geom\_line(size = 1) + labs(x = "Math Score", y = "Expected number of awards") ```

## Survival analysis

This is taken from here <https://www.openintro.org/download.php?file=survival_analysis_in_R&referrer=/stat/surv.php>

library(survival)  
library(KMsurv)

# Survival objects

Many functions in the survival package apply methods to Surv objects, which are survival-type objects created using the Surv() function. Here we discuss the construction of right-censored Surv objects and left-truncated right-censored Surv objects. See reference 6 for descriptions of survival data types. For right-censored data, only two arguments are needed in the Surv() function: a vector of times and a vector indicating which times are observed and censored.

data(tongue)  
attach(tongue)  
# create a subset for just the first group by using [type==1]  
my.surv <- Surv(time[type==1], delta[type==1])  
my.surv

## [1] 1 3 3 4 10 13 13 16 16 24 26 27 28 30   
## [15] 30 32 41 51 65 67 70 72 73 77 91 93 96 100   
## [29] 104 157 167 61+ 74+ 79+ 80+ 81+ 87+ 87+ 88+ 89+ 93+ 97+  
## [43] 101+ 104+ 108+ 109+ 120+ 131+ 150+ 231+ 240+ 400+

The plus-signs identify those observations that are right-censored. The first argument in Surv() should be input as a vector of observed and right-censored times. An indicator vector is used in the second argument to signify whether the event was observed (1) or not (0). Boolean arguments may be used in place of 1 and 0 in the indicator vector. We also consider left-truncated right-censored data. The left-truncation times are entered as the first argument, a vector of the event and censored times is input into the second argument, and an indicator vector for right-censoring is input as the third argument.

survfit(my.surv ~ 1)

## Call: survfit(formula = my.surv ~ 1)  
##   
## n events median 0.95LCL 0.95UCL   
## 52 31 93 67 NA

Survfit() also has a number of optional arguments. For example, the confidence level may be changed using the second argument, conf.int (e.g. conf.int=0.90 for 90% confidence bounds). The conf.type argument describes the type of confidence interval. More specifically, it describes the transformation for constructing the confidence interval. The default is "log", which equates to the transformation function g(t) = log(t). The "log-log" option uses g(t) = log(− log(t)). A linear confidence interval is created using the argument conf.type="plain". In the current version of the survival package (version 2.36-10), the arcsine-squareroot transformation must be computed manually using components of the object returned by survfit(). Like many functions in R, the survfit() function returns hidden information that can be accessed with the proper commands. Below we consider several elements of this hidden information, which is stored in a list. For a complete summary of the object, apply the str function to my.fit and to summary(my.fit).

my.fit <- survfit(my.surv ~ 1)  
summary(my.fit)$surv # returns the Kaplan-Meier estimate at each t\_i

## [1] 0.9807692 0.9423077 0.9230769 0.9038462 0.8653846 0.8269231 0.8076923  
## [8] 0.7884615 0.7692308 0.7500000 0.7115385 0.6923077 0.6730769 0.6538462  
## [15] 0.6340326 0.6142191 0.5944056 0.5745921 0.5547786 0.5342312 0.5061138  
## [22] 0.4779963 0.4481216 0.4161129 0.3814368 0.3051494 0.2288621

summary(my.fit)$time # {t\_i}

## [1] 1 3 4 10 13 16 24 26 27 28 30 32 41 51 65 67 70  
## [18] 72 73 77 91 93 96 100 104 157 167

summary(my.fit)$n.risk # {Y\_i}

## [1] 52 51 49 48 47 45 43 42 41 40 39 37 36 35 33 32 31 30 29 27 19 18 16  
## [24] 14 12 5 4

summary(my.fit)$n.event # {d\_i}

## [1] 1 2 1 1 2 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

summary(my.fit)$std.err # standard error of the K-M estimate at {t\_i}

## [1] 0.01904496 0.03233357 0.03695265 0.04088170 0.04733150 0.05246268  
## [7] 0.05465371 0.05663478 0.05842727 0.06004806 0.06282630 0.06400387  
## [13] 0.06505092 0.06597368 0.06688358 0.06766462 0.06832122 0.06885695  
## [19] 0.06927461 0.06968954 0.07146916 0.07281989 0.07414397 0.07544148  
## [25] 0.07671114 0.09177120 0.09540529

summary(my.fit)$lower # lower pointwise estimates (alternatively, $upper)

## [1] 0.9441432 0.8810191 0.8534195 0.8271685 0.7774159 0.7302341 0.7073724  
## [8] 0.6849189 0.6628317 0.6410776 0.5984672 0.5775712 0.5569274 0.5365233  
## [15] 0.5156073 0.4949392 0.4745101 0.4543127 0.4343412 0.4137057 0.3837502  
## [22] 0.3546085 0.3240114 0.2916674 0.2571797 0.1692469 0.1010963

str(my.fit) # full summary of the my.fit object

## List of 13  
## $ n : int 52  
## $ time : num [1:45] 1 3 4 10 13 16 24 26 27 28 ...  
## $ n.risk : num [1:45] 52 51 49 48 47 45 43 42 41 40 ...  
## $ n.event : num [1:45] 1 2 1 1 2 2 1 1 1 1 ...  
## $ n.censor : num [1:45] 0 0 0 0 0 0 0 0 0 0 ...  
## $ surv : num [1:45] 0.981 0.942 0.923 0.904 0.865 ...  
## $ type : chr "right"  
## $ std.err : num [1:45] 0.0194 0.0343 0.04 0.0452 0.0547 ...  
## $ upper : num [1:45] 1 1 0.998 0.988 0.963 ...  
## $ lower : num [1:45] 0.944 0.881 0.853 0.827 0.777 ...  
## $ conf.type: chr "log"  
## $ conf.int : num 0.95  
## $ call : language survfit(formula = my.surv ~ 1)  
## - attr(\*, "class")= chr "survfit"

str(summary(my.fit)) # full summary of the my.fit object

## List of 15  
## $ n : int 52  
## $ time : num [1:27] 1 3 4 10 13 16 24 26 27 28 ...  
## $ n.risk : num [1:27] 52 51 49 48 47 45 43 42 41 40 ...  
## $ n.event : num [1:27] 1 2 1 1 2 2 1 1 1 1 ...  
## $ n.censor : num [1:27] 0 0 0 0 0 0 0 0 0 0 ...  
## $ surv : num [1:27] 0.981 0.942 0.923 0.904 0.865 ...  
## $ type : chr "right"  
## $ std.err : num [1:27] 0.019 0.0323 0.037 0.0409 0.0473 ...  
## $ upper : num [1:27] 1 1 0.998 0.988 0.963 ...  
## $ lower : num [1:27] 0.944 0.881 0.853 0.827 0.777 ...  
## $ conf.type : chr "log"  
## $ conf.int : num 0.95  
## $ call : language survfit(formula = my.surv ~ 1)  
## $ table : Named num [1:9] 52 52 52 31 147 ...  
## ..- attr(\*, "names")= chr [1:9] "records" "n.max" "n.start" "events" ...  
## $ rmean.endtime: num 400  
## - attr(\*, "class")= chr "summary.survfit"

The object returned by summary(my.fit) is a list. The str function is useful for seeing more details about what is contained in the list, and as shown above, we can access each item in the list using the $ operator. The Kaplan-Meier estimate may be plotted using plot(my.fit). Standard arguments in the plot function may be used to improve the graphical aesthetics:

plot(my.fit, main="Kaplan-Meier estimate with 95% confidence bounds",  
 xlab="time", ylab="survival function")

