A tiny introduction to a Poisson Generalized Linear Model (GLM), spelled-out in code

Adrian Scheerer

October 2023

Introduction

This is an introduction to a very simple example of a Poisson GLM in R. The goal is that we understand and manually implement all output of `summary()` for this special case. This includes:

- how the GLM is fitted to the data,
- how to predict from the GLM,
- how categorical vs. numerical variables influence the fit,
- the derivation and interpretation of quality of fit indicators (deviance and AIC),
- one possibility to test for statistical significance of including an additional parameter in the model (likelihood ratio test),
- the meaning and derivation of parameter standard errors and p-values.

The emphasis will be on the expositional code, rather than on mathematical derivations or intuition for the quantities introduced. For this we rather refer to these references:

- The book *Non-Life Insurance Pricing with Generalized Linear Models* by E. Ohlsson and B. Johansson, Chapters 1 - 3. All references to sections are to this book.
- The paper *GLM for Dummies (and Actuaries)* by D. Clark, available here: https://eforum.casact.org/article/83925-glm-for-dummies-and-actuaries . This is a very good introduction to the intuition behind GLM's without relying on any mathematical derivations.

We will work with the following example of a Poisson GLM:

```r
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- rep(1:3, times=3) # 1, 2, 3, 1, ...
treatment <- rep(1:3, each=3) # 1, 1, 1, 2, ....
glm.numeric <- glm(counts ~ outcome + treatment, family = poisson())
```

This model has two explaining variables (`outcome` and `treatment`), one intercept, and the sample size is 9. It is a variant of the first example of `?glm`. In the appendix we explain why we use numeric variables instead of categorical variables as the original example does.

Fitting the GLM

GLM's are generally fitted by maximizing the likelihood of the data under the given model specification. In the Poisson GLM case the unique maximum likelihood estimator is the one that solves the following equation:

$$X' exp(X\beta) = X' y.$$

Here,

- X is the ("design-") matrix of observations (each row has the values of outcome and treatment, but with an additional 1 for the intercept),
- the dash ' is the matrix transpose,
- β is the column vector of coefficients (intercept, outcome and treatment),
- y is the response variable `claims` as column vector,

The dimensions of the matrices involved are (number of rows x number of columns):

- X: number of samples x (number of parameters + 1 for the intercept),
- β: number of samples x 1,
- y: number of samples x 1.

On both sides of the equations is a column vector of dimension (number of parameters + 1 for the intercept) x 1.

Fisher's scoring method

The exact parameter values in this example (up to floating point precision) can be obtained with Fisher's scoring method from Section 3.2.3.

```r
# The response variable counts as a column vector
y <- matrix(counts)

# Design matrix: each row starts with a 1 for the intercept,
# then contains the values for outcome and treatment.
X <- cbind("(Intercept)"=1, outcome, treatment)

# Starting values
beta <- c(log(mean(counts)), 0, 0) # mean(counts) is predicted for each observation

# Loop
n_fisher_scoring_iterations <- 4

for (istep in 1:n_fisher_scoring_iterations) {
  mu <- exp(X %*% beta)
  Iinv <- solve(t(X) %*% diag(c(mu)) %*% X) # Inverse of Fisher information
  beta <- beta + Iinv %*% (t(X) %*% y - t(X) %*% mu) # Equation (3.20)
}
print(t(beta))
```

```
##      (Intercept)    outcome    treatment
## [1,]    3.126198 -0.1606877 3.441801e-18
```

```r
print(glm.numeric$coefficients)
```

```
##   (Intercept)       outcome     treatment 
##  3.126198e+00 -1.606877e-01  2.186029e-14
```

The number of iterations 4 is sufficient to be within machine accuracy of the parameter values derived by `glm()` (which also uses 4 iteration steps).

(We omitted the weight matrix W from equation (3.20). In our case, it is just a diagonal matrix of 1's. W will also not update during the loop, the denominator before equation (2.32) is 1.)

Can we not log-transform the response variable and fit a linear model?

An interesting side observation is that alternatively one could think that we could fit a linear model to the log-transformed response variable `counts`. This is not true. It shows because the fitted parameters are slightly different:

```r
lm.log <- lm(log(counts) ~ outcome + treatment)
lm.log$coefficients
```

```
## (Intercept)     outcome   treatment 
##  3.13688882 -0.15271512 -0.02715058
```

The underlying reason for the difference is that in general maximizing the log-likelihood and minimizing the sum of squared differences is not the same. For the linear model (with Gaussian errors) the two approaches are the same, but for the Poisson GLM they are not.

Predicting from the model

The following are all equivalent methods to get the model values for the observations used to fit the model:

The explicit version of exp(linear term):

```r
exp(glm.numeric$coefficients["(Intercept)"] +
      outcome * glm.numeric$coefficients["outcome"] +
      treatment * glm.numeric$coefficients["treatment"])
```

```
## [1] 19.40460 16.52414 14.07126 19.40460 16.52414 14.07126 19.40460 16.52414
## [9] 14.07126
```

There are different ways to get the in-sample fitted values:

```r
glm.numeric$fitted.values
```

```
##        1        2        3        4        5        6        7        8 
## 19.40460 16.52414 14.07126 19.40460 16.52414 14.07126 19.40460 16.52414 
##        9 
## 14.07126
```

```r
predict(glm.numeric, type = "response")
```

```
##        1        2        3        4        5        6        7        8 
## 19.40460 16.52414 14.07126 19.40460 16.52414 14.07126 19.40460 16.52414 
##        9 
## 14.07126
```

A general expression to predict from the model is:

```r
predict(glm.numeric, newdata = data.frame(outcome, treatment), type="response")
```

```
##        1        2        3        4        5        6        7        8 
## 19.40460 16.52414 14.07126 19.40460 16.52414 14.07126 19.40460 16.52414 
##        9 
## 14.07126
```

The argument `type="response"` is necessary to include the link function. Otherwise, only the linear terms would be calculated:

```r
predict(glm.numeric) # missing argument: type="response"
```

```
##        1        2        3        4        5        6        7        8 
## 2.965510 2.804822 2.644135 2.965510 2.804822 2.644135 2.965510 2.804822 
##        9 
## 2.644135
```

```r
glm.numeric$coefficients["(Intercept)"] +
  outcome * glm.numeric$coefficients["outcome"] +
  treatment * glm.numeric$coefficients["treatment"]
```

```
## [1] 2.965510 2.804822 2.644135 2.965510 2.804822 2.644135 2.965510 2.804822
## [9] 2.644135
```

Deviance and model comparison

We would like to better understand whether variables in a GLM are really necessary, or whether they can be excluded. This can be done by comparing the deviances of two models, that we will introduce here. First, we give an explicit calculation of the log-likelihood of the fitted model.

Log-likelihood

We work with the Poisson distribution which has probability mass function

$$\lambda^k \frac{e^{-\lambda}}{k!}.$$

- The parameter λ is equal to the mean of the distribution, which is the output of the GLM for each observation. That is, λ is given by `fitted(glm.numeric)`. Note that the lambda's are specific to each observation.

- The number k is given by `counts`.

The likelihood is then the product of the individual probabilities. We take the log, and the log of the product becomes a sum of log's, where each term can be written as

$$\log(\lambda^k \frac{e^{-\lambda}}{k!}) = k\log(\lambda) - \lambda - \sum_{l=1}^{k} \log(l).$$

Working with the log-likelihood instead of the likelihood is easier for numerical reasons (it is easier to work with log of k! rather than with k! directly).

Replacing λ by `fitted(glm.numeric)`, k by `counts`, and introducing a short-hand for the sum of logs up to k, we can obtain the log-likelihood like this:

```
lambda <- fitted(glm.numeric)
k <- counts

# A vector of sums of logs of positive integers up to k, for each value k in counts
sum_logs <- sapply(counts, function(k) sum(log(1:k)))

# The log-likelihood is the sum of the log-likelihoods for each observation
ll <- sum(k * log(lambda) - lambda - sum_logs)
ll
```

```
## [1] -24.82407
```

This is the same as

```
logLik(glm.numeric)
```

```
## 'log Lik.' -24.82407 (df=3)
```

Deviance

The Log-likelihood of the perfect (so-called "saturated") model in which all observations are perfectly predicted is obtained by setting the λ's to be equal to the observations:

```
# Perfect fit in the saturated model
lambda_saturated <- counts

llsaturated <- sum(k * log(lambda_saturated) - lambda_saturated - sum_logs)
llsaturated
```

```
## [1] -20.81609
```

The the deviance (or residual deviance) is defined as

```
2*(llsaturated - ll)
```

```
## [1] 8.015958
```

The (residual) deviance can also be obtained by calling `deviance(glm.numeric)` or `glm.numeric$deviance`.

The log-likelihood of the null model, in which all observations are modeled by just one parameter (the mean of all observations) is

```r
# The output of the null model is the mean over all observations
lambda_null <- mean(counts)

llnull <- sum(k * log(lambda_null) - lambda_null - sum_logs)
llnull
```

```
## [1] -26.10681
```

The null deviance is then defined as

```r
2*(llsaturated - llnull)
```

```
## [1] 10.58145
```

This value is the same as `glm.numeric$null.deviance`. Both the null and the residual deviances are displayed in the `summary()` of a GLM.

Likelihood ratio test to compare nested models

The difference in deviances of two nested models is approximately χ^2 distributed with degrees of freedom equal to the difference in the number of parameters in the two nested models. For example,

```r
# The null model:
glm.null <- glm(counts ~ 1, family = poisson())

# Models with one term only (including intercept):
glm.numeric.dropTreatment <- glm(counts ~ outcome,   family = poisson())
glm.numeric.dropOutcome   <- glm(counts ~ treatment, family = poisson())
```

The deviances are

```r
dev.null          <- glm.null$deviance # same as glm.numeric$null.deviance
dev.dropTreatment <- glm.numeric.dropTreatment$deviance
dev.dropOutcome   <- glm.numeric.dropOutcome$deviance
dev.full          <- glm.numeric$deviance
```

- 10.5814459 for the null model,
- 8.015958 for the outcome-only model,
- 10.5814459 for the treatment-only model,
- 8.015958 for the full model.

The likelihood ratio test acts on the differences between these deviances, whenever two models are nested. Whenever there is one additional free parameter, the degrees of freedom for the χ^2 distribution is 1:

```
p.null2outcome   <- pchisq(dev.null - dev.dropTreatment, df=1, lower.tail = FALSE)
p.null2treatment <- pchisq(dev.null - dev.dropOutcome   , df=1, lower.tail = FALSE)
p.outcome2full   <- pchisq(dev.dropTreatment - dev.full, df=1, lower.tail = FALSE)
p.treatment2full <- pchisq(dev.dropOutcome - dev.full   , df=1, lower.tail = FALSE)
```

The p-values of the likelihood ratio test for the nested models are:

- 0.1092189 going from the Null model to term outcome,
- 1 going from the Null model to term treatment,
- 1 going from the outcome-only to full model,
- 0.1092189 going from the treatment-only to full model,

Going from the Null model to the full model, there are 2 additional parameters and the degrees of freedom is 2:

```
p.null2full <- pchisq(dev.null - dev.full, df=2, lower.tail = FALSE)
```

The p-value is 0.2772754.

Parameter standard errors

Asymptotically (see Sections 3.2.2 and 3.2.5), the estimated parameters are normally distributed, with mean equal to the true parameters, and covariance matrix equal to the inverse of the Fisher information matrix.

```
# Fisher information matrix I
D <- diag(fitted(glm.numeric))
I <- t(X) %*% D %*% X

# Inverse of Fisher information
Iinv <- solve(I)

# Standard errors
se <- sqrt(diag(Iinv)) # variances on the diagonal
se
```

```
## (Intercept)     outcome   treatment
##   0.2880594   0.1006457   0.1000000
```

```
# z-values
z.values <- glm.numeric$coefficients / se
z.values
```

```
##   (Intercept)       outcome     treatment
##  1.085261e+01 -1.596567e+00  2.186029e-13
```

```
# p-values
2 * pnorm(abs(z.values), lower.tail = FALSE)
```

```
##  (Intercept)      outcome    treatment
## 1.938031e-27 1.103623e-01 1.000000e+00
```

These values are those as produced by the `summary()` function:

```r
summary(glm.numeric)$coefficients
```

```
##                 Estimate Std. Error       z value     Pr(>|z|)
## (Intercept)  3.126198e+00 0.28805942  1.085261e+01 1.938022e-27
## outcome     -1.606877e-01 0.10064574 -1.596567e+00 1.103623e-01
## treatment    2.186029e-14 0.09999999  2.186029e-13 1.000000e+00
```

Additional output of `summary()`

The degrees of freedoms are 8 for the null deviance (9 observations minus one parameter), and 6 for the proposed model (9 observations minus 3 parameters).

The Akaike information criterion (AIC) is defined as 2 times the number of estimated parameters minus 2 times the log-likelihood. In our case, this is

```r
2*3 - 2*ll
```

```
## [1] 55.64814
```

which is the same as

```r
AIC(glm.numeric)
```

```
## [1] 55.64814
```

The dispersion parameter is an additional parameter of the larger class of the so-called exponential dispersion models. For the Poisson distribution this parameter is equal to 1. For the Poisson GLM, the parameter does not influence the maximum likelihood estimation and is fixed it 1. In our derivation of the log-likelihood we have ignored the dispersion parameter.

`summary()`

We have now seen how all the outputs of `summary()` for a Poisson GLM are produced. All of the following output should now be familiar:

```r
summary(glm.numeric)
```

```
##
## Call:
## glm(formula = counts ~ outcome + treatment, family = poisson())
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.126e+00  2.881e-01  10.853   <2e-16 ***
## outcome     -1.607e-01  1.006e-01  -1.597     0.11
## treatment    2.186e-14  1.000e-01   0.000     1.00
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 10.581  on 8  degrees of freedom
## Residual deviance:  8.016  on 6  degrees of freedom
## AIC: 55.648
##
## Number of Fisher Scoring iterations: 4
```

Appendix: The role of categorical variables (factors)

In these notes we used a variant of the first example from `?glm`. The original version is:

```
counts <- c(18,17,15,20,10,20,25,13,12)
outcome.fac <- gl(3,1,9) # factor 1, 2, 3, 1, ..., with levels 1, 2, 3
treatment.fac <- gl(3,3) # factor 1, 1, 1, 2, ..., with levels 1, 2, 3
glm.D93 <- glm(counts ~ outcome.fac + treatment.fac, family = poisson())
```

Note that the explaining variables `outcome` and `treatment` are categorical (we noted this with suffix `.fac`).

Instead of working with factors, we found it simpler to use a numerical variant of this example. Below, we show that the GLM with 2 categorical variables where both have 3 levels is similar to a GLM with 4 numeric variables.

We one-hot encode the variables `outcome` and `treatment` by creating matrices of zeros that have as many columns as the two variables have levels, and by assigning for each observation a 1 in the column corresponding to the level in that observation.

```
outcome_onehot <- matrix(0, nrow=length(outcome.fac), ncol=nlevels(outcome.fac))
outcome_onehot[cbind(1:9, outcome.fac)] <- 1

treatment_onehot <- matrix(0, nrow=length(treatment.fac), ncol=nlevels(treatment.fac))
treatment_onehot[cbind(1:9, treatment.fac)] <- 1
```

For example, `outcome.fac` is 1, 2, 3, 1, 2, 3, 1, 2, 3, and its one-hot encoding is

```
outcome_onehot
```

```
##       [,1] [,2] [,3]
##  [1,]    1    0    0
##  [2,]    0    1    0
##  [3,]    0    0    1
##  [4,]    1    0    0
##  [5,]    0    1    0
##  [6,]    0    0    1
##  [7,]    1    0    0
##  [8,]    0    1    0
##  [9,]    0    0    1
```

In the corresponding GLM, the intercept is interpreted as the base level. All parameters indicate the difference of the explaining variables to this base level. For example, `outcome.fac` has 3 levels. The first level will be treated as base level, and two parameters will be estimated for the effect of changing from the base level to each of the two other levels. To manually reproduce this, we need to remove the first column of the one-hot encoded matrices:

```r
glm.onehot <- glm(counts ~ outcome_onehot[, -1] + treatment_onehot[, -1],
                  family = poisson())
```

We now see that the GLM with one-hot encoded categorical variables and removed base case is equivalent to the GLM with categorical variables:

```r
glm.D93$coefficients
```

```
##    (Intercept)   outcome.fac2   outcome.fac3 treatment.fac2 treatment.fac3
##   3.044522e+00  -4.542553e-01  -2.929871e-01   1.217511e-15   8.437695e-16
```

```r
glm.onehot$coefficients
```

```
##             (Intercept)   outcome_onehot[, -1]1   outcome_onehot[, -1]2
##            3.044522e+00           -4.542553e-01           -2.929871e-01
## treatment_onehot[, -1]1 treatment_onehot[, -1]2
##            1.217511e-15            8.437695e-16
```