GLM notes

Exponential family & GLMs

Linear regression can be summarised in one equation: $EY|\,X \sim N(X\beta, \sigma^2 I)$. It assumes that the conditional mean $EY|\,X$ is IID normal, can be estimated using a linear prediction $X\beta$, and has constant variance σ^2. This course relaxes the normality assumption, allowing us to model a wider variety of processes. The analysis of hierarchical & other dependent data course relaxes the IID assumption.

The normal distribution is a member of the *exponential family*. The p parameter exponential family has a density $f(x)$ of the form

$$\ln f_\theta(x) = \sum_{i=1}^{p} \eta_i T_i(x) - A(\theta) + c(x)$$

GLMs replace the assumption that the conditional mean is normally distributed with the assumption that the conditional mean is an exponential family distribution. For most applications it's better to work with the *exponential dispersion model* instead, which has a simpler form

$$\ln f(x) = \frac{x\theta - A(\theta)}{\phi} + c(x, \phi)$$

where θ, ϕ are scalar parameters and $\phi > 0$. Most of the common distributions are in the exponential family, here's some examples (ignore the g column for now, that's explained in the next few paragraphs:

distribution	density	θ	$A(\theta)$	ϕ	g
Normal	$\frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2}$	μ	$\mu^2/2$	σ^2	identity
Binomial	$\binom{n}{x} p^x (1-p)^{n-x}$	$\ln\left(\frac{p}{1-p}\right)$	$n\ln(1+e^\theta)$	1	logit
Poisson	$e^{-\lambda}\lambda^x / x!$	$\ln\lambda$	e^θ	1	log
Negative binomial	$\binom{x+r-1}{x} p^x (1-p)^r$	$\ln p$	$-x\ln(1-e^\theta)$	1	log

ϕ is the dispersion parameter, and $A(\theta)$ is called the log partition function. These are related to the mean and variance through

$$EX = A'(\theta), \qquad VX = \phi A''(\theta)$$

The binomial and poisson distributions have relationships between their means & variances, which may not hold in the data. For example using a poisson distribution assumes that $VX = EX$, if the variance in a particular

group is very different to the mean then the data is overdispersed or underdispersed. This can be accounted for by allowing ϕ to vary instead of taking the values in the table - these are called quasipoisson & quasibinomial models. ϕ is usually estimated through the method of moments in these 'quasi' models, because it saves on needing to explicitly define the $c(x, \phi)$ term so is more computationally efficient.

The generalised linear model says that the *natural parameter* θ can be modelled by a linear predictor, $\theta = X\beta$. In the table above, the natural parameter is related to the mean of the distribution in some way (this is always true for a member of the exponential family), so GLMs have the form

$$g(EY|X) = X\beta$$

The function g is called the *canonical link function*. The canonical link function should be used when modelling unless you have a good reason to use something else. This is because canonical links are guaranteed to be 'nice' functions, in the sense that their log-likelihoods will always be quadratic and so easy to maximise through the usual techniques. Generally you would want a link function to have a range of $(-\infty, \infty)$, because there is no restriction on the range of values for the linear predictor. Canonical links always have this property. Using a different link - a log link for a binomial response for example - may cause convergence issues due to the range mismatch.

The interpretation of regression coefficients depends on the response distribution used in GLMs. To figure out the interpretation, start from the fact that the model is linear on the link scale and consider a one-unit change in one of the predictors. Imagine you have two people who are identical in every way, except one person has $x_i = \tau + 1$ and the other person has $x_i = \tau$. Then the linear predictors η for the two people are

$$\eta_1 = \beta_0 + \beta_1 x_1 + \cdots + \beta_i \tau + \cdots + \beta_p x_p$$
$$\eta_2 = \beta_0 + \beta_1 x_1 + \cdots + \beta_i (\tau + 1) + \cdots + \beta_p x_p$$

The difference between the two people is

$$\eta_2 - \eta_1 = \beta_i$$

So a one unit change in variable i causes a β_i unit change *on the link scale*. To interpret this properly you need to convert back to the response scale, which depends on the specific link function used. A similar thing holds for calculating combinations of the model parameters - first calculate everything on the link scale where things are linear, then transform back to the response scale for interpretation.

Binary data - logistic regression part 1

The canonical link function for binomial data is the logit $\ln[p/(1-p)]$. Models which use a logit as the response variable are called *logistic regression* models. So for binary data, the canonical GLM model is

$$\ln\left(\frac{p}{1-p}\right) = X\beta$$

Following on from the end of the previous section, a one unit change in the x_i variable is associated with a β_i unit change on the link scale, that is

$$\begin{aligned}\beta_i &= \ln\left(\frac{p_2}{1-p_2}\right) - \ln\left(\frac{p_1}{1-p_1}\right) \\ &= \ln\left(\frac{p_2}{1-p_2} \Big/ \frac{p_1}{1-p_1}\right)\end{aligned}$$

The right hand side is the log odds ratio. So e^{β_i} represents the odds ratio for the outcome in someone with xi one unit higher than someone else, with all other variables held constant.

GLMs are fit in R using the `glm` command. As an example, here's how to fit a logistic model to the `insect.dta` data. The insect data contains data on 8 *groups* of insects. The number of insects `n` in each group is recorded, along with the `dose` of insecticide given to the group and the number `r` of insects who died after receiving the dose. There's two ways to model this data, the first is to work with the grouped data directly, and the other is to convert the data into individual format. Here's both of them:

```
insect = haven::read_dta('practicals/data/insect.dta')

# Model on grouped data
m1 = glm(cbind(r, n - r) ~ dose, data = insect, family = binomial(link = 'logit'))

# Convert to individual format and fit second model
died = c()
for (i in 1:nrow(insect)) died = c(died,
                                   rep(1, insect$r[i]),
                                   rep(0, insect$n[i] - insect$r[i]))
insect_indiv = data.frame(dose = rep(insect$dose, insect$n),
                          died = died)

# Fit model
m2 = glm(died ~ dose, data = insect_indiv, family = binomial(link = 'logit'))
```

The syntax for `m1` looks a bit weird - if the data is grouped then `glm` function needs the left hand side of the formula to be a 2-column table, the first column is the number of successes and the second is the number of failures. This is needed because `glm` assumes you're working with individual data. Stata makes you do something similar, the syntax for the same model in stata is `glm r dose, family(binomial n) link(logit)`. Stata needs you to tell it what the group size variable is, and R needs you to make a table. No big deal.
The syntax for `m2` is much more similar to the usual `lm` type models. The only new bit with `glm` is the `family` argument which lets you specify the distribution of the response variable and, inside the `family` function, you specify the link function to use. All the family functions will use the canonical link by default, so the link argument only needs to be specified if you're working with non-canonical links. As was mentioned in the previous section, this can be a bad idea - changing the link to log instead of logit will cause the model to not converge.

Both of the models have the exact same parameter estimates, makes sense seeing as they're the same models fit to the same data:

```
# Use the broom package to collect the output into the usual format
broom::tidy(m1)
```

```
# A tibble: 2 x 5
```

```
  term        estimate std.error statistic  p.value
  <chr>          <dbl>     <dbl>     <dbl>    <dbl>
1 (Intercept)  -14.1      1.23       -11.5 1.92e-30
2 dose           0.237    0.0203      11.7 2.21e-31
```

```
broom::tidy(m2)
```

```
# A tibble: 2 x 5
  term        estimate std.error statistic  p.value
  <chr>          <dbl>     <dbl>     <dbl>    <dbl>
1 (Intercept)  -14.1      1.23       -11.5 1.92e-30
2 dose           0.237    0.0203      11.7 2.21e-31
```

Accounting for dispersion is easy, just change the family argument. This code fits a quasibinomial model:

```
glm(died ~ dose, data = insect_indiv, family = quasibinomial())
```

Quasibinomials allow for under or over dispersion compared to a binomial model, so the standard error and confidence intervals from this model will be slightly different to the previous two models. The parameter estimates will be the same. It's not possible to do inference about the dispersion parameter ϕ since it requires the $c(x, \phi)$ term in the exponential family equation to be known which isn't possible in general. This means that there isn't a formal test available to determine if dispersion is present in the data, it's more of a vibe.

How are these models fit?

When you run `glm` R starts doing a bunch of matrix algebra. It's nice to walk through the steps R does to fit GLMs, just so it isn't a black box and you can understand what is happening. All the steps are just the usual maximum likelihood estimation process, though the maths is a little bit more involved.

Suppose you've got individual data (X, y) and you want to fit a logistic model to y. The log likelihood is

$$\begin{aligned}\ln L(y) &= \sum_{i=1}^{n} y_i \ln p_i + (1 - y_i) \ln(1 - p_i) \\ &= \ln(1 - p_i) + y_i \ln\left(\frac{p_i}{1 - p_i}\right)\end{aligned}$$

Where p_i is the probability of success for person i ($y_i = 1$). Using the canonical link you would fit a linear model to the log odds, since it is the natural parameter for the problem:

$$\ln\left(\frac{p}{1-p}\right) = X\beta, \qquad \ln\left(\frac{p_i}{1-p_i}\right) = x_i^T\beta \text{ for individual } i$$

This lets us write the log likelihood in terms of the design matrix X and the parameter vector β, a bit of rearranging gives

$$p_i = \frac{e^{x_i^T \beta}}{1 + e^{x_i^T \beta}} = \frac{1}{1 + e^{-x_i^T \beta}}$$
$$1 - p_i = \frac{1}{1 + e^{x_i^T \beta}}$$

Giving a log likelihood of

$$\ln L(\beta) = \sum_{i=1}^{n} -\ln(1 + e^{x_i^T \beta}) + y_i x_i^T \beta$$

Differentiating with respect to the j^{th} component of β, β_j gives

$$\partial_{\beta_j} \ln L(\beta) = \sum_{i=1}^{n} y_i x_{i,j}^T - p_i x_{i,j}^T$$

Where $x_{i,j}^T$ is the j^{th} element of x_i^T. This can be written in matrix form as

$$\nabla_\beta \ln L(\beta) = X^T (y - p)$$

This is the *score equation*, and setting it equal to zero and solving for β gives the MLE. This is difficult / impossible to do in general, so instead R uses numerical methods to get the solution. R uses Newton Raphson to find β, which finds the roots of a function $f(x)$ using the update rule of

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

(To derive this imagine x_{n+1} is the root, calculate the slope at x_n and solve for x_{n+1}). Applying this to the score equation gives an update rule

$$\beta_{k+1} = \beta_k - H^{-1}(\beta_k) \nabla_\beta \ln L(\beta_k)$$

Where H is the matrix of derivative of the score function (or the second derivatives of the log likelihood), called the *Hessian*. Differentiating the equation for the j^{th} component of the derivative of the log likelihood with respect to β_k gives the k, j element of H as

$$\begin{aligned}
\partial_{\beta_k} (\nabla_\beta \ln L)_{(j)} &= \partial_{\beta_k} \sum_{i=1}^{n} y_i x_{i,j}^T - \frac{x_{i,j}^T}{1 + e^{-x_i^T \beta}} \\
&= -\sum_{i=1}^{n} x_{i,j}^T x_{i,k}^T \frac{e^{-x_i^T \beta}}{(1 + e^{-x_i^T \beta})^2} \\
&= -\sum_{i=1}^{n} x_{i,j}^T x_{i,k}^T p_i (1 - p_i)
\end{aligned}$$

Or, in matrix form

$$H = -X^T D X, \qquad D = diag(p_i (1 - p_i))$$

All what R does is use the update rule (along with the matrices we just derived) to keep updating the estimate of β. Eventually the estimate doesn't change much between steps - the estimate converges - and the algorithm stops.

When a model is fit, R calculates the design matrix X and it can be accessed using `model.matrix(my_model)`. Here's the algorithm done 'by hand' on the insect data:

```
logistic_byhand = function(X, y, max_iter = 25, tol = 1e-10){
  # X = design matrix, y = response vector

  beta = rep(0, ncol(X)) # Initial guess for beta, to be updated using NR
  for (i in 1:max_iter){
    beta_old = beta
    p = 1 / (1 + exp(-X %*% beta))
    D = diag(as.numeric(p * (1 - p)))
    H = -crossprod(X, D %*% X)
    score = t(X) %*% (y - p)

    beta = beta_old - solve(H, score) # Update step
    if (sqrt(crossprod(beta - beta_old)) < tol) break
  }
  return(beta)
}

X = model.matrix(m2)
y = insect_indiv$died

logistic_byhand(X, y)
```

```
                  [,1]
(Intercept) -14.0864027
dose          0.2365929
```

Which is exactly the same coefficients in `m2`.

Count data

Count data is usually modelled using either the Poisson or negative binomial distributions.

Model fit, checks, and performance

Observational data, marginal effects, collapsability

Cohort, case control, and matched studies - logisitic regression part 2

Multinomial & ordinal models - logistic regression part 3