An extended example: Poisson regression

Patrick Breheny

November 6, 2024

Introduction

- Today is the final lecture of "part 2" of this course, and we will use it to go through an extended example involving Poisson regression
- This example will be a good case study of the likelihood tools we've developed so far, in particular:
    - We will see how these ideas work in a regression setting
    - We will go into detail regarding the implementation, algorithms, and computational issues surrounding the fitting of these models

Our case study: The British Doctors' Study

- We're actually going to analyze real data today!
- Our motivating example will be the landmark British Doctors' Study, the first large prospective cohort study to examine the health consequences of smoking
- In 1951, British doctors were sent a brief questionnaire about whether they smoked tobacco; since then, information about their deaths has been collected
- Our analysis today will focus on estimating the effect of smoking on death, while adjusting for age and duration of follow-up

Cohort studies and follow-up time

- Certainly, if we follow one group (smokers / nonsmokers) longer, we are likely to observe more deaths in that group
- Assuming deaths are independent between subjects, this can be quantified using total person-years of follow-up for a group (duration of follow-up for each person added up over all subjects in the group)
- Letting t_i denote the person-years of follow-up in group i, because the sum of independent Poisson random variables also follows a Poisson distribution, the number of deaths Y_i in group i can be modeled as

$$Y_i \sim \text{Pois}(\mu_i) \qquad \mu_i = t_i \lambda_i,$$

  where λ_i is the rate of interest (in our case, it will be death rate per 1,000 person-years)

Poisson likelihood

- The Poisson probability function is $p(y|\mu) = \mu^y e^{-\mu}/y!$, so the contribution to the log-likelihood coming from the ith observation (in our case, the ith group, as individual observations have been pooled together into groups) is

$$\ell_i = y_i \log \mu_i - \mu_i$$

- Recall that this is in the exponential family, with natural parameter $\eta_i = \log \mu_i$
- In GLM terminology, this is known as the "random component" of the model

Link function

- What remains is to connect μ_i to the covariates; this is known as the "systematic component" of the model
- For a variety of reasons, both theoretical and practical, it makes sense for our model to operate on the scale of the natural parameter η_i rather than μ_i directly
- In our case, the assumption $\log \lambda_i = \mathbf{x}_i^\top \boldsymbol{\beta}$ means that

  $$\log \mu_i = \eta_i = \log t_i + \mathbf{x}_i^\top \boldsymbol{\beta};$$

  the $\log t_i$ term is known as the *offset* of the model
- Note that the offset is *not* an intercept – there is no coefficient associated with $\log t_i$

Link function (cont'd)

- In previous lectures and homework assignments, we've seen many advantages of working with natural parameters (minimal sufficiency, CRLB, observed information = expected information)
- In the Poisson case, we also have the practical advantage that $\mathbf{x}_i^\top \boldsymbol{\beta}$ can be any real number, but μ_i is restricted to be positive
- If we were to fit the model $\lambda_i = \mathbf{x}_i^\top \boldsymbol{\beta}$, we run the risk of predicting negative mean counts; not only is this nonsensical, it may cause problems with code and the algorithms we're using
- In GLM terminology, modeling the natural parameter is known as the "canonical link"

Score

- Let's begin by determining the score and information of this model
- Taking the gradient of the log-likelihood with respect to $\boldsymbol{\beta}$, we have

$$\mathbf{u}(\boldsymbol{\beta}) = \mathbf{X}^\top(\mathbf{y} - \boldsymbol{\mu})$$

- Solving for the root of this likelihood equation, however, is not something we are going to be able to accomplish analytically, as μ is a nonlinear function of $\boldsymbol{\beta}$: $\mu_i = \exp(\mathbf{x}_i^\top \boldsymbol{\beta})$

Information

- To proceed, we can take a Taylor series expansion of the score to obtain a linear approximation which *can* be solved in closed form
- To do so, we require the information, so let's derive that now:

  $$\mathcal{I}_n = \mathbf{X}^\top \mathbf{W} \mathbf{X},$$

  where $\mathbf{W}$ is a diagonal matrix with elements e^{η_i}
- The element $w_i = e^{\eta_i}$ is often called the weight of observation i, for reasons that you will explore in the next homework assignment

Newton's method

- We can now proceed to find $\widehat{\beta}$ by repeatedly solving the approximate score equations, then re-approximating, re-solving, and so on
- Each iteration of this process looks like this:

$$\widehat{\boldsymbol{\beta}}^{(m+1)} = \widehat{\boldsymbol{\beta}}^{(m)} + (\mathbf{X}^\top \mathbf{W}^{(m)} \mathbf{X})^{-1} \mathbf{X}^\top (\mathbf{y} - \boldsymbol{\mu}^{(m)}),$$

- This algorithm is known as *Newton's method*, or the *Newton-Raphson algorithm*

Fisher scoring

- Specifically, Newton's method refers to updating the parameters according to $\widehat{\boldsymbol{\beta}}^{(m+1)} = \widehat{\boldsymbol{\beta}}^{(m)} + \boldsymbol{\mathcal{I}}_n^{-1}\mathbf{u}$
- A related method, known as *Fisher scoring*, uses the expected information instead: $\widehat{\boldsymbol{\beta}}^{(m+1)} = \widehat{\boldsymbol{\beta}}^{(m)} + \boldsymbol{\mathscr{I}}_n^{-1}\mathbf{u}$
- Both approaches are valid and usually perform similarly
- In our case, of course, the two approaches are the same, but if we had used a non-canonical link function this would not be the case

Implementing these ideas

- Let's now implement these derivations with code
- As with other lectures, this code is available online, but unlike other lectures, we're going to discuss coding details in lecture as well
- Furthermore, I'm going to do this in an iterative manner, with an initial, bare-bones version of the algorithm that we then improve upon

Initial version

```r
# MLE: Simple
beta <- rep(0, ncol(X))
for (i in 1:30) {
  eta <- drop(log(Time) + X %*% beta)
  mu <- exp(eta)
  W <- diag(mu)
  Info <- crossprod(X, W) %*% X
  beta <- beta + solve(Info) %*% crossprod(X, y-mu)
}
```

Let's compare with the output of `fit <- glm(...)`:

```
# beta
# -1.0116 1.4840 2.6275 3.3505 3.7001 0.3545
# coef(fit):
# -1.0116 1.4840 2.6275 3.3505 3.7001 0.3545
```

Areas we could improve

Our algorithm works, although could be improved in several ways:

- Monitor convergence instead of assuming 30 iterations is appropriate
- Use a smarter choice of initial value
- Put our code into a function to make it more reproducible and easier to use
- Take steps to ensure robust convergence

Monitoring convergence

Adding a simple convergence monitor:

```
# Improvement 1: Monitor convergence
converged <- function(old, new, eps=1e-6) {
  all(abs(old-new) < eps)}
beta <- rep(0, ncol(X)); iter <- 0
repeat {
  old <- beta
  iter <- iter + 1
  eta <- drop(log(Time) + X %*% beta)
  mu <- exp(eta)
  W <- diag(mu)
  Info <- crossprod(X, W) %*% X
  beta <- old + solve(Info) %*% crossprod(X, y-mu)
  if (converged(old, beta) | iter > 50) break
}
```

Comments

- The code on the previous slide required 22 iterations to reach convergence
- It is always a good idea to have a maximum number of iterations (here, we chose 50) so that your program doesn't run forever in case something goes wrong
- Here, we based our assessment of convergence on the maximum absolute difference in coefficients, $\|\widehat{\boldsymbol{\beta}}^{(m+1)} - \widehat{\boldsymbol{\beta}}^{(m)}\|_\infty$, which is simple and reasonable, although there are a variety of other options

Relative vs. absolute convergence

- The main drawback of monitoring absolute convergence is that it is not invariant to scale
- In other words, whether a model has converged or not depends on whether we measure distance in feet or inches; this is a bit absurd
- To avoid this, relative criteria are sometimes used instead:

$$\frac{|\widehat{\beta}_j^{(m+1)} - \widehat{\beta}_j^{(m)}|}{|\widehat{\beta}_j^{(m)}|} < \epsilon$$

- One drawback of a relative criterion is that it can be unstable for $\widehat{\beta}_j \approx 0$, so a hybrid approach can also be applied:

$$\frac{|\widehat{\beta}_j^{(m+1)} - \widehat{\beta}_j^{(m)}|}{|\widehat{\beta}_j^{(m)}| + \epsilon} < \epsilon$$

Which scale?

- Alternatively, convergence can be monitored on the scale of the linear predictors, which are invariant to changes of scale among the features
- Finally, we could also choose to assess convergence by looking at changes in the likelihood (this is what `glm()` does)
- These last two approaches also avoid issues due to lack of convergence due to multicollinearity, although whether this is desirable or not depends on the situation
- In general, all of these approaches are reasonable, but none of them is perfect

Initial value

- One of the reasons that our algorithm requires 22 iterations is that it starts at $\boldsymbol{\beta} = \mathbf{0}$
- Since the MLE of the intercept-only model is trivial to calculate, we could at least start there:

$$\widehat{\beta}_0 = \log \frac{\sum_i y_i}{\sum_i t_i}$$

- In code:

```
beta[1] <- log(sum(y) / sum(Time))
```

- Now our algorithm requires just 8 iterations

Function

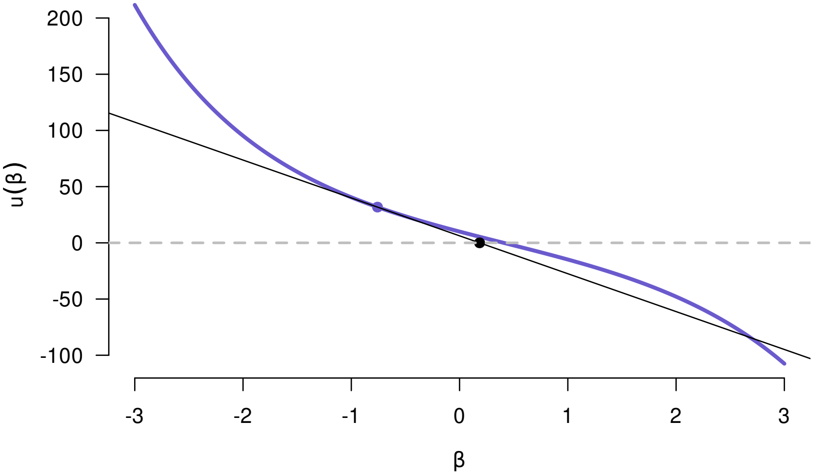
- Our block of code is starting to be useful, so we should *definitely* turn it into a function rather than copy and paste it every time we want to use it
- Copying and pasting blocks of code is the number one source of errors, so the moment you find yourself doing it, **immediately** stop and turn it into a function
- Our function looks like this (see R code for full definition):

```r
pois_fit <- function(X, y, Time, eps=1e-6) {
  ...
  list(
    loglik=loglik, beta=drop(beta), iter=iter,
    r=y-exp(eta), eta=eta, Info=Info)
}
```

Is convergence guaranteed?

- Finally, let's take a moment to discuss the possibility of non-convergence
- Is Newton's method guaranteed to converge?
- No
- Why not? And is it possible to fix Newton's method so that it does converge?
- First, let's take a look at a (1-d) animation that will help us understand how Newton's method works

Newton's method: animation



Convergence theory

- This isn't a course on numerical analysis, so we're not going to go into the full details of convergence theory, but let's give a rough picture of the requirements for Newton's method to converge
- These requirements turn out to be rather similar to the requirements for the MLE to be consistent, in particular that the information matrix is positive definite (in 1d, this would mean that the score is monotone decreasing)
- If this is true locally, we can only hope to say that we converge to a given solution if our initial value is reasonably close to the solution
- If the information is p.d. everywhere (i.e., the likelihood is log-concave), then likelihood equations have (at most) one solution, and Newton's method will probably find it

Monotone likelihood

- Why "probably"? What can go wrong? Two things
- First of all, there may not be any solutions to the score equations
- This happens when the likelihood is a monotone function of θ, so $\hat{\theta}^{(m)}$ would increase without bound
- This happens a lot in logistic regression, for example: the MLE of the probability could be 1, which corresponds to $\widehat{\beta} = \infty$ on the log-odds scale

Unstable updates

- The other thing that could go wrong is that the Newton step could overshoot the solution
- Sometimes this is fine, but it could also spiral out of control, increasingly overshooting the solution in opposite directions with each iteration
- This situation, however, is fixable
- The problem is that the Newton step is too large, so we can simply take smaller steps

Damped updates

- In particular, we could modify our iterative procedure:

$$\widehat{\boldsymbol{\beta}}^{(m+1)} = \widehat{\boldsymbol{\beta}}^{(m)} + \alpha \boldsymbol{\mathcal{I}}^{-1}\mathbf{u}$$

- The "pure" Newton's method uses $\alpha = 1$, but the "damped" or "guarded" Newton's method uses some $\alpha \leq 1$
- This can be implemented in two ways:
  - A 1-d line search over α to find the best α
  - Setting $\alpha = 1/2$ if the update would decrease the log-likelihood (then $\alpha = 1/4$ if it still decreases the log-likelihood, and so on); this is known as step-halving
- With either of these modifications, Newton's method is guaranteed to converge to the MLE (if it exists) of a log-concave likelihood

Wald

- I won't spent a great deal of time on inference, as the story here will be similar to what we just saw with the gamma distribution
- Starting with the Wald approach (these are the same as what you get from `summary(fit)` with `glm()`:

```r
# Wald tests
z <- res$beta / sqrt(diag(solve(res$Info)))
p <- 2*pnorm(-abs(z))
```

- And confidence intervals are easily obtained:

```r
# Wald intervals
SE <- sqrt(diag(solve(res$Info)))
lwr <- res$beta + qnorm(0.025) * SE
upr <- res$beta + qnorm(0.975) * SE
```

LR test

- For the likelihood ratio test:

```
# LRT
X0 <- model.matrix( ~ Age, britdoc)
X1 <- model.matrix( ~ Age + Smoking, britdoc)
ll0 <- pois_fit(X0, y, Time)$loglik
ll1 <- pois_fit(X1, y, Time)$loglik
x <- 2*(ll1-ll0)
pchisq(x, 1, lower.tail=FALSE)
```

- Note that these results are the same as what you get from

```
null <- glm(Deaths ~ offset(log(PersonYears/1000))
            + Age, britdoc, family=poisson)
anova(null, fit, test='Chisq')
```

Profiling

- Confidence intervals are not as easily obtained, as we need to solve for the restricted MLE
- However, it's not actually as bad as you might think, as we can abuse our offset to find this restricted MLE without writing much new code
- In particular,

$$\eta_i = \log t_i + x_{ij}\beta_j + \mathbf{x}_{i,-j}^\top \boldsymbol{\beta}_{-j}$$

- Since our code earlier passed `Time` as opposed to the offset directly, we'll be fitting the restricted MLEs via:
  where here `X` is equal to $\mathbf{X}_{-j}$ in the formula

LR confidence interval

So, for example, we might implement LR intervals like this:

```r
# LR confidence interval
lr_chisq <- function(b, X, x, y, Time, ll1) {
  out <- double(length(b))
  for (j in length(b)) {
    res <- pois_fit(X, y, exp(log(Time) + x*b[j]))
    out[j] <- 2*(ll1-res$loglik)
  }
  out
}
f <- function(b) qchisq(0.95, 1) -
  lr_chisq(b, X[,1:5], X[,6], y, Time, ll1)
lwr <- uniroot(f, interval=c(-5, res$beta[6]))$root
```

Note that this is the same as the output of `confint(fit)`

Score

- Finally, the score test:

```
# Score test
x <- X[, 6]
X0 <- X[, -6]
res0 <- pois_fit(X0, y, Time)
W <- diag(exp(res0$eta))
v <- solve(crossprod(X, W) %*% X)[6,6]
z <- drop(crossprod(x, res0$r)) * sqrt(v)
2*pnorm(-abs(z))
```

- Note that `v` could also be calculated as

```
v <- 1/(crossprod(x, W) %*% x -
        crossprod(x, W) %*% X0 %*%
        solve(res0$Info) %*%
        crossprod(X0, W) %*% x)
```

Results

- In this example, the log-likelihood is well-approximated by a quadratic and all three approaches give very similar results
- The results for the effect of smoking on mortality rate per 1,000 person-years, given as rate ratios (e^{β}):

	95% interval		
	Lower	Upper	p
Wald	1.1550	1.7594	0.00096
LR	1.1609	1.7692	0.00057
Score	1.1554	1.7587	0.00090

- As a historical note, this was arguably the first important study to show the health risks of smoking