Fitting linear models in R (2)

Yue Jiang

Duke University

Estimating bike crashes in NC counties

Newton-Raphson in higher dimensions

Score vector and Hessian for $\log \mathcal{L}(\boldsymbol{\theta}|\mathbf{X})$ with $\boldsymbol{\theta} = (\theta_1, \cdots, \theta_p)^T$:

$$\nabla \log \mathcal{L} = \begin{pmatrix} \frac{\partial \log \mathcal{L}}{\partial \boldsymbol{\theta}_1} \\ \vdots \\ \frac{\partial \log \mathcal{L}}{\partial \boldsymbol{\theta}_p} \end{pmatrix}$$

$$\nabla^2 \log \mathcal{L} = \begin{pmatrix} \frac{\partial^2 \log \mathcal{L}}{\partial \theta_1^2} & \frac{\partial^2 \log \mathcal{L}}{\partial \theta_1 \theta_2} & \cdots & \frac{\partial^2 \log \mathcal{L}}{\partial \theta_1 \theta_p} \\ \frac{\partial^2 \log \mathcal{L}}{\partial \theta_2 \theta_1} & \frac{\partial^2 \log \mathcal{L}}{\partial \theta_2^2} & \cdots & \frac{\partial^2 \log \mathcal{L}}{\partial \theta_2 \theta_p} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 \log \mathcal{L}}{\partial \theta_p \theta_1} & \frac{\partial^2 \log \mathcal{L}}{\partial \theta_p \theta_2} & \cdots & \frac{\partial^2 \log \mathcal{L}}{\partial \theta_p^2} \end{pmatrix}$$

Newton-Raphson in higher dimensions

We can modify the Newton-Raphson algorithm for higher dimensions:

- Start with initial guess $\boldsymbol{\theta}^{(0)}$
- Iterate $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \left(\nabla^2 \log \mathcal{L}(\boldsymbol{\theta}^{(t)}|\mathbf{X})\right)^{-1} \left(\nabla \log \mathcal{L}(\boldsymbol{\theta}^{(t)}|\mathbf{X})\right)$
- Stop when convergence criterion is satisfied

Under certain conditions, a global maximum exists; this again is guaranteed for many common applications.

Computing the Hessian can be computationally demanding (and annoying), but there are ways around it in practice.

Poisson regression

$$\log \mathcal{L} = \sum_{i=1}^{n} y_i \mathbf{X}_i \boldsymbol{\beta} - e^{\mathbf{X}_i \boldsymbol{\beta}} - \log y_i!$$

$$\nabla \log \mathcal{L} = \sum_{i=1}^{n} \left(y_i - e^{\mathbf{X}_i \boldsymbol{\beta}} \right) \mathbf{X}_i^T$$

$$\nabla^2 \log \mathcal{L} = - \sum_{i=1}^{n} e^{\mathbf{X}_i \boldsymbol{\beta}} \mathbf{X}_i \mathbf{X}_i^T$$

Newton-Raphson update steps for Poisson regression:

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} - \left(- \sum_{i=1}^{n} e^{\mathbf{X}_i \boldsymbol{\beta}} \mathbf{X}_i \mathbf{X}_i^T \right)^{-1} \left(\sum_{i=1}^{n} \left(y_i - e^{\mathbf{X}_i \boldsymbol{\beta}} \right) \mathbf{X}_i^T \right)$$

Back to bike crashes

```
## # A tibble: 100 x 6
##    county        pop med_hh_income traffic_vol pct_rural crashes
##    <chr>       <dbl>         <dbl>       <dbl>     <dbl>   <dbl>
##  1 Alamance   166436          50.5         182        29      77
##  2 Alexander   37353          49.1          13        73       1
##  3 Alleghany   11161          39.7          28       100       1
##  4 Anson       24877          38            79        79       7
##  5 Ashe        27109          41.9          18        85       4
##  6 Avery       17505          41.7          35        89       5
##  7 Beaufort    47079          46.4          53        66      37
##  8 Bertie      19026          35.4          24        83      10
##  9 Bladen      33190          37            19        91       9
## 10 Brunswick 136744          60.2          43        43      88
## # ... with 90 more rows
```

- `pop`: county population
- `med_hh_income`: median household income in thousands
- `traffic_vol`: mean traffic volume per meter of major roadways
- `pct_rural`: percentage of county population living in rural area

Back to bike crashes

Let's fit a model with `traffic_vol` and `pct_rural` for now:

```r
m1 <- glm(crashes ~ traffic_vol + pct_rural,
          data = bike, family = "poisson")

round(summary(m1)$coef, 6)
```

```
##              Estimate Std. Error    z value Pr(>|z|)
## (Intercept)  5.982181   0.053749 111.298625        0
## traffic_vol  0.001541   0.000166   9.262671        0
## pct_rural   -0.044558   0.000875 -50.919036        0
```

What might we conclude / interpret from this model?

Newton-Raphson (rough) implementation

Newton-Raphson update steps for Poisson regression:

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} - \left(- \sum_{i=1}^{n} e^{\mathbf{X}_i \boldsymbol{\beta}} \mathbf{X}_i \mathbf{X}_i^T \right)^{-1} \left(\sum_{i=1}^{n} \left(y_i - e^{\mathbf{X}_i \boldsymbol{\beta}} \right) \mathbf{X}_i^T \right)$$

Function for score vector, given vector `beta`, matrix `X`, and vector `y`:

```
d1func <- function(beta, X, y){
  d1 <- rep(0, length(beta))
  for(i in 1:length(y)){
    d1 <- d1 + (y[i] - exp(X[i,] %*% beta)) %*% X[i,]
  }
  return(colSums(d1))
}
```

Newton-Raphson (rough) implementation

Newton-Raphson update steps for Poisson regression:

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} - \left(-\sum_{i=1}^{n} e^{\mathbf{X}_i \boldsymbol{\beta}} \mathbf{X}_i \mathbf{X}_i^T \right)^{-1} \left(\sum_{i=1}^{n} \left(y_i - e^{\mathbf{X}_i \boldsymbol{\beta}} \right) \mathbf{X}_i^T \right)$$

Function for Hessian, given vector `beta`, matrix `X`, and vector `y`:

```
d2func <- function(beta, X, y){
  d2 <- matrix(0, nrow = length(beta), ncol = length(beta))
  for(i in 1:length(y)){
    d2 <- d2 - t((exp(X[i,] %*% beta)) %*% X[i,]) %*% (X[i,])
  }
  return(d2)
}
```

Newton-Raphson (rough) implementation

Some bookkeeping:

```
beta <- c(mean(log(bike$crashes)), 0, 0)
X <- cbind(1, bike$traffic_vol, bike$pct_rural)
y <- bike$crashes
iter <- 1
delta <- 1

temp <- matrix(0, nrow = 500, ncol = 3)
```

Newton-Raphson (rough) implementation

Actual Newton-Raphson implementation:

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} - \left(-\sum_{i=1}^{n} e^{\mathbf{X}_i \boldsymbol{\beta}} \mathbf{X}_i \mathbf{X}_i^T \right)^{-1} \left(\sum_{i=1}^{n} \left(y_i - e^{\mathbf{X}_i \boldsymbol{\beta}} \right) \mathbf{X}_i^T \right)$$

```
while(delta > 0.000001 & iter < 500){
  old <- beta
  beta <- old - solve(d2func(beta = beta, X = X, y = y)) %*%
              d1func(beta = beta, X = X, y = y)
  temp[iter,] <- beta

  delta <- sqrt(sum((beta - old)^2))
  iter <- iter + 1
}
```

Newton-Raphson (rough) implementation

```
iter
```

```
## [1] 22
```

```
delta
```

```
## [1] 3.911961e-07
```

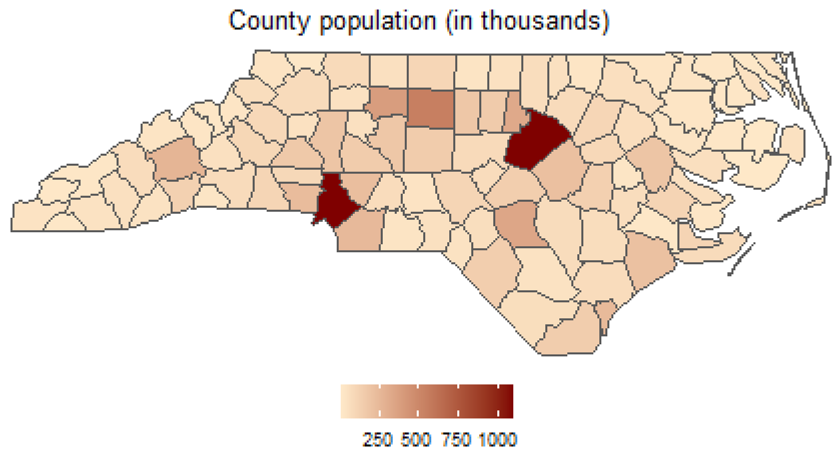
```
beta
```

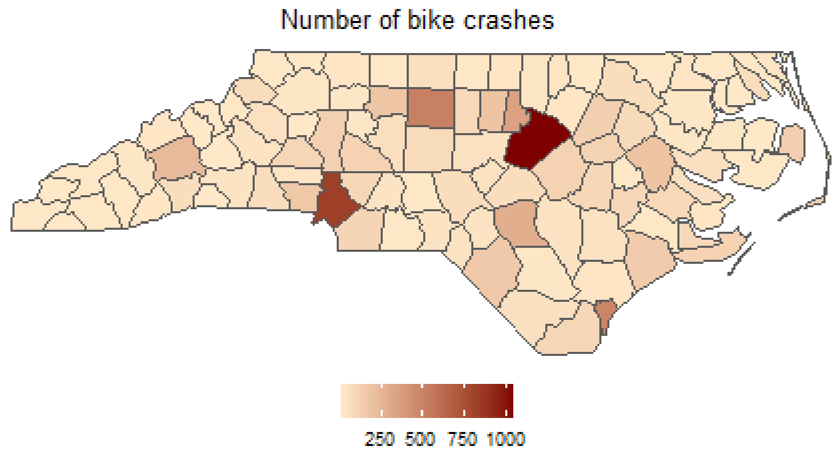
```
##              [,1]
## [1,]  5.98218054
## [2,]  0.00154064
## [3,] -0.04455809
```

```
m1$coefficients
```

```
## (Intercept) traffic_vol   pct_rural
##  5.98218054  0.00154064 -0.04455809
```

Back to bike crashes





Back to bike crashes

$$\log(E(Y|\mathbf{X})) = \beta_0 + \beta_1(pop) + \beta_2(traffic) + \beta_3(rural)$$

```
m2 <- glm(crashes ~ traffic_vol + pct_rural + pop,
          data = bike, family = "poisson")
```

$$\log\left(\frac{E(Y|\mathbf{X})}{pop}\right) = \beta_0 + \beta_1(traffic) + \beta_2(rural)$$

```
m3 <- glm(crashes ~ traffic_vol + pct_rural, offset = log(pop),
          data = bike, family = "poisson")
```

What are the differences in the two models above?

Back to bike crashes

$$\log\left(\frac{E(Y|\mathbf{X})}{pop}\right) = \beta_0 + \beta_1(traffic) + \beta_2(rural)$$
$$\log(E(Y|\mathbf{X})) = \beta_0 + \beta_1(traffic) + \beta_2(rural) + \log(pop)$$

Here, we are using `pop` as an **offset**. This means that our dependent variable is actually a *rate*, even though we are providing counts, and we can look at covariate effects directly on this rate.

If we use `pop` as a covariate, then the response is no longer a rate of bike crashes per unit population. However, we would be able to assess association between population and number of bike crashes (conditionally on traffic volume and urbanicity).

Back to bike crashes

```r
round(summary(m1)$coef, 6)
```

```
##              Estimate Std. Error    z value Pr(>|z|)
## (Intercept)  5.982181   0.053749 111.298625        0
## traffic_vol  0.001541   0.000166   9.262671        0
## pct_rural   -0.044558   0.000875 -50.919036        0
```

```r
round(summary(m2)$coef, 6)
```

```
##              Estimate Std. Error    z value Pr(>|z|)
## (Intercept)  5.655725   0.054837 103.136325 0.000000
## traffic_vol -0.000093   0.000179  -0.518756 0.603931
## pct_rural   -0.037761   0.000878 -43.015409 0.000000
## pop          0.000001   0.000000  30.215337 0.000000
```

```r
round(summary(m3)$coef, 6)
```

```
##              Estimate Std. Error     z value Pr(>|z|)
## (Intercept) -6.916803   0.054480 -126.961100 0.000000
## traffic_vol -0.000047   0.000171   -0.272118 0.785531
## pct_rural   -0.010936   0.000857  -12.766690 0.000000
```

Poisson regression with offset

Can we simply use `bike$crashes/bike$pop` as our outcome variable in the code we've already written?

```
beta <- c(mean(log(bike$crashes)), 0, 0)
X <- cbind(1, bike$traffic_vol, bike$pct_rural)
y <- bike$crashes / bike$pop
iter <- 1
delta <- 1

temp <- matrix(0, nrow = 500, ncol = 3)

while(delta > 0.000001 & iter < 500){
  old <- beta
  beta <- old - solve(d2func(beta = beta, X = X, y = y)) %*%
               d1func(beta = beta, X = X, y = y)
  temp[iter,] <- beta

  delta <- sqrt(sum((beta - old)^2))
  iter <- iter + 1
}
```

Poisson regression with offset

```
round(beta, 6)
```

```
##             [,1]
## [1,] -6.810266
## [2,]  0.000314
## [3,] -0.011783
```

```
round(m3$coefficients, 6)
```

```
## (Intercept) traffic_vol   pct_rural
##   -6.916803   -0.000047   -0.010936
```

They're close, but not quite right. Did something go wrong?

Poisson regression with offset

```r
m3_wrong <- m2 <- glm(crashes/pop ~ traffic_vol + pct_rural,
          data = bike, family = "poisson")

round(m3_wrong$coefficients, 6)
```

```
## (Intercept) traffic_vol   pct_rural
##   -6.810266    0.000314   -0.011783
```

```r
round(beta, 6)
```

```
##           [,1]
## [1,] -6.810266
## [2,]  0.000314
## [3,] -0.011783
```

What's happening? (keep in mind, all output on this page is **wrong**)

Poisson regression with offset

Let's denote an offset term by ω. If we directly use `crashes/pop` in our Poisson regression likelihood, we would have a log-likelihood along the lines of

$$\log \mathcal{L} \propto \sum_{i=1}^{n} \frac{y_i}{\omega_i} \mathbf{X}_i \boldsymbol{\beta} - e^{\mathbf{X}_i \boldsymbol{\beta}}$$

This is incorrect. We cannot assume `crashes/pop` has a Poisson distribution.

Poisson regression with offset

If we write the log-likelihood for a Poisson regression with offset correctly, we have:

$$\log(E(Y|\mathbf{X})) = \beta_0 + \mathbf{X}^T\boldsymbol{\beta} - \log \boldsymbol{\omega}$$

$$\log \mathcal{L} \propto \sum_{i=1}^{n} y_i \mathbf{X}_i \boldsymbol{\beta} - \omega_i e^{\mathbf{X}_i \boldsymbol{\beta}}$$

Thus, we must use this *correct* log-likelihood to determine the score vector and Hessian for our Newton-Raphson implementation:

$$\nabla \log \mathcal{L} = \sum_{i=1}^{n} \left(y_i - \omega_i e^{\mathbf{X}_i \boldsymbol{\beta}} \right) \mathbf{X}_i^T$$

$$\nabla^2 \log \mathcal{L} = -\sum_{i=1}^{n} \omega_i e^{\mathbf{X}_i \boldsymbol{\beta}} \mathbf{X}_i \mathbf{X}_i^T$$

Poisson regression with offset

Newton-Raphson update steps for Poisson regression with offset:

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} - \left(-\sum_{i=1}^{n} \omega_i e^{\mathbf{X}_i \boldsymbol{\beta}} \mathbf{X}_i \mathbf{X}_i^T \right)^{-1} \left(\sum_{i=1}^{n} \left(y_i - \omega_i e^{\mathbf{X}_i \boldsymbol{\beta}} \right) \mathbf{X}_i^T \right)$$

In writing code, we must now specify the offset ω in addition to the observed data $\mathbf{X}$, y, and candidate $\boldsymbol{\beta}$s.

Poisson regression with offset

Functions for score vector and Hessian (including offset term):

```
d1ofs <- function(beta, X, y, offset){
  d1 <- rep(0, length(beta))
  for(i in 1:length(y)){
    d1 <- d1 + (y[i] -  offset[i] *exp(X[i,] %*% beta)) %*% X[i,]
  }
  return(colSums(d1))
}

d2ofs <- function(beta, X, y, offset){
  d2 <- matrix(0, nrow = length(beta), ncol = length(beta))
  for(i in 1:length(y)){
    d2 <- d2 - offset[i] * t((exp(X[i,] %*% beta)) %*% X[i,]) %*%
  }
  return(d2)
}
```

Poisson regression with offset

Implementing Newton-Raphson:

```
beta <- c(mean(log(bike$crashes)), 0, 0)
X <- cbind(1, bike$traffic_vol, bike$pct_rural)
y <- bike$crashes
offset <- bike$pop
iter <- 1
delta <- 1

temp <- matrix(0, nrow = 500, ncol = 3)

while(delta > 0.000001 & iter < 500){
  old <- beta
  beta <- old - solve(d2ofs(beta = beta, X = X, y = y, offset = of
                 d1ofs(beta = beta, X = X, y = y, offset = offset)
  temp[iter,] <- beta

  delta <- sqrt(sum((beta - old)^2))
  iter <- iter + 1
}
```

Poisson regression with offset

```r
round(beta, 6)
```

```
##             [,1]
## [1,] -6.916803
## [2,] -0.000047
## [3,] -0.010936
```

```r
round(m3$coefficients, 6)
```

```
## (Intercept) traffic_vol   pct_rural
##   -6.916803   -0.000047   -0.010936
```

Our manual Newton-Raphson code lines up, as expected.

Fisher Scoring

```
> summary(m3)

Call:
glm(formula = crashes ~ traffic_vol + pct_rural, family = "poisson",
    data = bike, offset = log(pop))

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-10.7121   -2.1484   -0.6808    1.4805   14.7958

Coefficients:
              Estimate Std. Error  z value Pr(>|z|)
(Intercept) -6.917e+00  5.448e-02 -126.961   <2e-16
traffic_vol -4.661e-05  1.713e-04   -0.272    0.786
pct_rural   -1.094e-02  8.566e-04  -12.767   <2e-16

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 2070.6  on 99  degrees of freedom
Residual deviance: 1527.2  on 97  degrees of freedom
AIC: 2036.8

Number of Fisher Scoring iterations: 5
```

Fisher Scoring

Fisher Scoring replaces $\nabla^2 \log \mathcal{L}$ with the expected Fisher information:

$$E\left(\left(\nabla \log \mathcal{L}\right)\left(\nabla \log \mathcal{L}\right)^T\right),$$

which is asymptotically equivalent to the Hessian. It's often easier to implement because we don't need to take the second derivative.

Fisher Scoring is usually a bit more stable than vanilla Newton-Raphson and the initial steps usually "get to where we want" faster. Newton-Raphson and Fisher Scoring updating steps are identical for GLMs under canonical link.