Algorithm documentation

Generalized Linear Model

Federated linear regression

M. Cellamare, F. Martin, A. van Gestel

IKNL

January 5, 2022

Contents

1 Introduction

The term generalized linear model (GLM) refers to a larger class of models popularized by McCullagh and Nelder (1982, 2nd edition 1989). In these models, the response variable y_i is assumed to follow an exponential family distribution with mean μ_i, which is assumed to be some (often nonlinear) function of $x_i^T\beta$.

2 Mathematics

2.1 Central

There are three components to any GLM:

- **Random Component** - refers to the probability distribution of the response variable y; e.g. normally distributed in the linear regression, or binomially distributed in the binary logistic regression. More generally, we consider all distribution that can be expressed in the form:

  $$f(y;\theta) = exp\left\{\frac{y\theta - b(\theta)}{a(\phi)} + c(y,\phi)\right\},$$

  where θ is the canonical parameter, such that $\mathbb{E}(y) = \mu = b'(\theta)$ and $Var(y) = a(\phi)b''(\theta)$. This is also called exponential family. Can be easily showed that, for instance, the canonical parameter for $y \sim N(\mu, \sigma^2)$ is $\theta = \mu$, and the canonical parameter for $y \sim Bin(n, \pi)$ is $\theta = logit(\pi) = log\left(\frac{\pi}{1-\pi}\right)$.

- **Systematic Component** - specifies the explanatory variables $x = (x_1, x_2, \ldots, x_k)$ in the model, more specifically their linear combination define the so called linear predictor

  $$\eta = x^T\beta,$$

  where β must be estimated.
- **Link Function** $g(\cdot)$ - specifies the link between random and systematic components. It says how the expected value of the response relates to the linear predictor of explanatory variables

  $$g(\mu) = \eta$$

  The most commonly used link function for a normal model is $\eta = \mu$, and the most commonly used link function for the binomial model is $\eta = logit(\pi)$. When $\eta = \theta$ we say that the model has a canonical link.

Estimation procedure

In the GLM estimation procedure, the maximum likelihood estimation for β can be carried out via Fisher scoring. The generic $(j+1)$-th step can be calculate by

$$\beta^{(j+1)} = \beta^{(j)} + \left[-\mathbb{E}l''\left(\beta^{(j)}\right)\right]^{-1} l'(\beta^{(j)}) \tag{1}$$

where l is the log-likelihood of the entire sample. Ignoring constants, the log-likelihood is

$$l(\theta; y) = \frac{y\theta - b(\theta)}{a(\phi)}$$

After some mathematical operations and using the canonical link $\eta = \theta$, the first derivative and expected second derivative of the log-likelihood are

$$\frac{\delta l}{\delta \beta_j} = \frac{y-\mu}{Var(y)}\left(\frac{\delta \mu}{\delta \eta}\right)x_{ij}$$
$$-\mathbb{E}\left(\frac{\delta^2 l}{\delta \beta_j \delta \beta_k}\right) = \frac{1}{Var(y)}\left(\frac{\delta \mu}{\delta \eta}\right)^2 x_{ij}x_{ik}$$

where x_{ij} (or x_{ik}) is the j-th element of the covariate vector $x_i = x$ for the i-th observation.

It follows that the score vector for the entire data set $y_1, \ldots, y_N$ can be written as

$$\frac{\delta l}{\delta \beta} = X^T A(y-\mu) \tag{2}$$

where $X = (x_1, \ldots, x_N)^T$, and $A = diag\left[Var(y_i)\left(\frac{\delta \eta_i}{\delta \mu_i}\right)\right]^{-1}$ and the expected Hessian matrix becomes

$$-\mathbb{E}\left(\frac{\delta^2 l}{\delta \beta_j \delta \beta_k}\right) = X^T W X$$

where $W = diag\left[Var(y_i)\left(\frac{\delta \eta_i}{\delta \mu_i}\right)^2\right]^{-1}$.

Therefore the Fisher scoring iteration in 2.1 can be expressed as

$$\beta^{(j+1)} = \beta^{(j)} + \left(X^T W X\right)^{-1} X^T A(y-\mu) \tag{3}$$

We can arrange the step of Fisher scoring to make it resemble weighted least squares.

Noting that $X\beta = \eta$ and $A = W\frac{\delta \eta}{\delta \mu}$, we can rewrite 2.1 as

$$\beta^{(j+1)} = \left(X^T W X\right)^{-1} X^T W z \tag{4}$$

where $z = \eta + \frac{\delta \eta}{\delta \mu}(y-\mu)$. Therefore, Fisher scoring can be regarded as Iteratively Reweighted Least Squares (IRWLS) carried out on a transformed version of the response variable.

The IRWLS algorithm can be describe as

Algorithm 1 GLM Fisher Scoring algorithm

procedure

initialize $\beta^{(0)}$

$\eta = X\beta^{(0)}$

$dev^{(0)}$

loop

compute $\mu = g'(\eta)$

$z = \eta + \frac{y-\mu}{\Delta g'}$

$W = w\frac{\Delta g'^2}{Var(\mu)}$

update $\beta^{(j)} = \left(X^TWX\right)^{-1}X^TWz$

$\eta = X\beta^{(j)}$

compute $dev^{(j)}$

if $|dev^{(j)} - dev^{(j-1)}| < \epsilon$ **then**

return $\beta^{(j)}$

end loop

else

$j = j + 1$

end if

end loop

end procedure

where $g(\cdot)$ is the link function, $\Delta g' = \frac{\delta\mu}{\delta\eta}$ is the derivative of the inverse-link function $g'(\cdot)$ with respect to the linear predictor and $w = w_1, \ldots, w_n$ are arbitrary weights assign to the units (by default equal to 1).

2.2 Federated

The main idea behind the federated GLM algorithm is that components of equation 2.1 can be partially computed in each data sources k and merged together afterwords without pulling together the data.

Let us consider $K \geq 2$ data sources (i.e. cancer registries, schools, banks etc..) and let's denote by n_k the number of observations in the k-th data source such that the total sample size of the study is $n = n_1 + \cdots + n_K$. Furthermore, let us denote by $y_{(k)}$ the n_k-vector of response variable and by $X_{(k)}$ the $(n_k \times p)$-matrix of p covariates for the data source $k = 1, \ldots, K$. It is easy to prove that

$$\begin{aligned} X^TWX &= \left[X^T_{(1)}W_{(1)}X_{(1)}\right] + \cdots + \left[X^T_{(K)}W_{(K)}X_{(K)}\right] \\ X^TWz &= \left[X^T_{(1)}W_{(1)}z_{(1)}\right] + \cdots + \left[X^T_{(K)}W_{(K)}z_{(K)}\right] \end{aligned}$$

where $z_{(K)} = \eta_{(k)} + \frac{y_{(k)} - \mu_{(k)}}{\Delta g'_{(k)}}$ and $W_{(k)} = diag\left[Var\left(y_{(k)}\right)\Delta g'^2_{(K)}\right]^{-1}$.

Therefore, following the structure of algorithm 1, a federated procedure can be described as follow:

Algorithm 2 My algorithm

Initialization Server

initialize $\beta^{(0)}$

Initialization Node k

initialize $\eta_{(k)} = X_{(k)}\beta^{(0)}$

initialize $\mu_{(k)} = g'(\eta_{(k)})$

initialize $dev^{(0)}_{(k)} = f(y_{(k)}\mu_{(k)}, w_{(k)})$

loop

Node k

compute $z_{(k)} = \eta_{(k)} + \frac{y_{(k)} - \mu_{(k)}}{\Delta g'_{(k)}}$

compute $W_{(k)} = w_{(k)} \frac{\Delta g'^2_{(k)}}{Var(\mu_{(k)})}$

compute $\mathcal{C}^1_{(k)} = X^T_{(k)} W_{(k)} X_{(k)}$

$\mathcal{C}^2_{(k)} = X^T_{(k)} W_{(k)} z_{(k)}$

return to Server $\mathcal{C}^1_{(k)}$ and $\mathcal{C}^2_{(k)}$

Server

calculate $X^T W X = \sum_{k=1}^{K} C^1_{(k)}$

calculate $X^T W z = \sum_{k=1}^{K} C^2_{(k)}$

update $\beta^{(j+1)} = \left(X^T W X\right)^{-1} X^T W z$

return to Nodes $\beta^{(j+1)}$

Node k

compute $\eta_{(k)} = X_{(k)}\beta^{(j+1)}$

compute $\mu_{(k)} = g'(\eta_{(k)})$

calculate $dev^{(j+1)}_{(k)} = f(y_{(k)}\mu_{(k)}, w_{(k)})$

return to Server $dev^{(j+1)}_{(k)}$

Server

compute $dev^{(j+1)} = \sum_{k=1}^{K} dev^{(j+1)}_{(k)}$

if $|dev^{(j+1)} - dev^{(j)}| < \epsilon$ **then**

return $\beta^{(j+1)}$

break loop

else

$j = j + 1$

end if

end loop

3 Implementation

3.1 Parameters

Input Parameters			
Parameter	Type	Example	Description
formula	string	a b̃ + c	string that can be cast to R formula object, see here
dstar	string	d_star	Column name of dstar sensor (expected value), only applicable for poison family
types	float	1.1	...
family	float	1.1	Family type
tol	float	1.1	Tolerance level
maxit	int	25	Max. number of iterations

3.2 Algorithm

Algorithm 3 master

Require: $n \geq 0$
Ensure: $y = x^n$
$y \leftarrow 1$
$X \leftarrow x$
$N \leftarrow n$
while $N \neq 0$ **do**
  if N is even **then**
    $X \leftarrow X \times X$
    $N \leftarrow \frac{N}{2}$ ▷ This is a comment
  else if N is odd **then**
    $y \leftarrow y \times X$
    $N \leftarrow N - 1$
  end if
end while

3.3 Output

[table of algorithm output(s)]

4 Risks

1. Issue 1
2. issue 2

5 Validation

```python
import do_stuff

from vantage6.client import Client

# create a client and autenticate
client = Client(...)
client.authenaticate(...)

# create task for algorithm
client.task.create(...)

# poll for results
ready = False
while not ready:
    do_stuff()
```

6 Examples

[Preferable multiple examples of how to run it from R, python and a plain API call]

```r
setup.client <- function() {
  # Define parameters
  username <- 'username@example.com'
  password <- 'password'
  host <- 'https://address-to-vantage6-server.domain'
  api_path <- ''

  # Create the client
  client <- vtg::Client$new(host, api_path=api_path)
  client$authenticate(username, password)

  return(client)
}

# Create a client
client <- setup.client()

# Get a list of available collaborations
print( client$getCollaborations() )

# Should output something like this:
#   id     name
# 1  1 ZEPPELIN
# 2  2 PIPELINE

# Select a collaboration
```

```r
client$setCollaborationId(1)

# vtg.dglm contains the function `dglm`.
model <- vtg.glm::dglm(client, formula = num_awards ~ prog + math, family='poisson', tol= 1e-08, maxit=25)
```

```python
import do_stuff

from vantage6.client import Client

# create a client and autenticate
client = Client(...)
client.authenaticate(...)

# create task for algorithm
client.task.create(...)

# poll for results
ready = False
while not ready:
    do_stuff()
```

References