GlmNet for Exponential Response Variables via RcppEigen

Xin Chen, Doug Martin, Aleksandr Aravkin, Dan Hanson

August 28, 2017

Abstract

This package implements the Generalized Linear Model with Elastic Net regularization for Exponential Response Variables (GLM-EN-EXP). This package is intended to fill the current gap in the R software ecosystem where an implementation that 1) supports exponential distribution, 2) supports Elastic Net model selection, 3) is easy to parallelize on multicore machine, does not exist, to the best knowledge of the authors. Significant speed improvement has been shown compared with both native R and H2O implementations in simple benchmark tets.

1 Theoretical Background

1.1 Generalized Linear Models For Exponential Distributions

The theory and methodology of generalized linear models (GLM's) is well established for independent observations $Y_1, Y_2, \cdots, Y_N$ whose mean values are $\mu_1, \mu_2, \cdots, \mu_N$ and whose distributions are members of the *exponential families*

$$f(y_k; \theta_k) = \exp\{(y_k\theta_k - b(\theta_k))/a(\phi_k) + c(y_k, \phi_k)\} \quad k = 1, 2, \cdots, N \tag{1.1}$$

along with a link function g from the $\mu_k$to a linear model form $\boldsymbol{x}_k'\boldsymbol{\beta}$:

$$g(\mu_k) = \boldsymbol{x}_k'\boldsymbol{\beta} = \eta_k. \tag{1.2}$$

The form stated above with y_k in the first term in the exponent rather than some function $a(y_k)$ is called the *canonical* form, and θ_k is called the *natural parameter*. See for example McCullagh and Nelder (1989) and Fox (2016)[1], where one finds simple derivations of the following expressions for the mean and variance of the observations:

$$\mu_k = E(Y_k) = b'(\theta_k) \tag{1.3}$$

$$var(Y_k) = b''(\theta_k)a(\phi_k). \tag{1.4}$$

1.2 Maximum Likelihood Model Estimation for GLM

The likelihood function for observed values $y_1, y_2, \cdots, y_N$ of the independt set $Y_1, Y_2, \cdots, Y_N$ is

$$L(\boldsymbol{y}; \boldsymbol{\theta}) = \prod_{k=1}^{N} f(y_k; \theta_k)$$

and the log-likelihood function is

$$l(\boldsymbol{y}; \boldsymbol{\theta}) = \log \prod_{k=1}^{N} f(y_k; \theta_k) = \sum_{k=1}^{N} \log f(y_k; \theta_k) = \sum_{k=1}^{N} l(y_k; \theta_k).$$

The classic GLM model is usually fitted using a method called iterative reweighted least squares (IRWLS) method.

[1] Other authors use slightly different but equivalent mathemtatical forms of an exponential family, e.g., see Dobson(2002)

1.3 GLM with Elastic Net Regularization

One problem with the standard GLM formulation is that it does not have regularization penalty for overfitting. More Regularization usually comes in the form of augmenting the original cost function (such as the log-likelihood function) with a penalty term that expresses the modeler's belief about the properties of the model. For example, in ridge regulariaztion, the sume of least square errors is augmented by the L2 norm of the coefficients

$$\hat{\beta}_{Ridge} = \underset{\boldsymbol{\beta},\lambda}{\operatorname{argmin}} LL(\boldsymbol{y};\boldsymbol{\theta}) + \lambda\|\boldsymbol{\beta}\|_2$$

This implies that the modeler believes there is unlikely to be particularly large coeffcients in the model. In deed, large coefficients will be penalized heavily by the L2 regularization, resulting in smaller values for the coefficients.

A different kind of regularization, which became very popular in recent years, is called least absolute shrinkage and selection operator (LASSO). The idea is similar to that of ridge regression, except that instead of the believing the coefficients should not be too large, the modeler believes the model should be sparse with respect to the variables. This is achieved by augmenting the MSE with L1 norm of the coefficients.

$$\hat{\beta}_{LASSO} = \underset{\boldsymbol{\beta},\lambda}{\operatorname{argmin}} LL(\boldsymbol{y};\boldsymbol{\theta}) + \lambda\|\boldsymbol{\beta}\|_1$$

The sparsity of $\hat{\beta}_{LASSO}$ is controlled by adjusting the value of λ. One thing to note is that even thought $\hat{\beta}_{LASSO}$ tends to be sparse, it could very well have large coefficients, which may or may not be desirable.

The latest development in regularization comes in the form of Elastic Net (EN). It combines Ridge Regression and LASSO regression by using weighted sum of L1 nrom and L2 norm of the coeffcients as the penalty function.

$$\hat{\beta}_{EN} = \underset{\boldsymbol{\beta},\lambda}{\operatorname{argmin}} LL(\boldsymbol{y};\boldsymbol{\theta}) + \lambda(\alpha\|\boldsymbol{\beta}\|_1 + (1-\alpha)\|\boldsymbol{\beta}\|_2)$$

The the parameter $\alpha \in [0,1]$ expresses our belief about the relative importance of sparsity and coefficient value.

1.4 GLM-EN via Proximal Gradient Descent

The proximal gradident descent algorithm is designed to solve problems of the form

$$\min_x \quad f(x) + g(x)$$

where $f(x)$ and $g(x)$ are closed proper convex and $f(x)$ is differentiable. The proximal gradient method is

$$x^{k+1} = \text{prox}_{t^k g}(x^k - \nabla f(x))$$

where

$$\text{prox}_{tg}(v) = \min_x \quad f(x) + \frac{1}{2t}||x-v||_2^2$$

For our problem, $f(x)$ would be the sum negative log-likelihood of the exponential response variables and the L2 regularization and $g(x)$ would be the L1 regularization. Therefore,

$$\begin{aligned}\text{prox}_{tg}(v) = \text{prox}_{t||x||_1}(v) &= \text{argmin}_x \left(||x||_1 + \frac{1}{2t}||x-v||_2^2\right) \\ &= \begin{cases} v_i - t & v_i \geq t \\ v_i + t & v_i \leq -t \\ 0 & |v_i| \leq t \end{cases}\end{aligned}$$

2 Sample Code

2.1 Installation

To install the package, run the following code in R.

```
Sys.setenv( "PKG_CXXFLAGS"="-std=c++14" )
libary(devtools)
install_github("chenx26/glmnetRcpp")
```

2.2 Structure of the Sample Code for Timings and Results Comparison between glmnetRcpp and H2O

To compare our implementation with H2O, we simulate test data and then use both algorithms to try to find the true coefficients used to genereate the simulated data. We assume that the data follows the model

$$b_i \sim exp(\lambda_i)$$

where

$$\log \lambda_i = -A_i * x$$

The test data set is generated by the following steps:

1. Randomly genereate the independent variable matrix A of size $n \times p$, where n is the number of observations, p is the number of independent variables.
2. Randomly generate the true coefficient of the model x.true, which is a vector of length p.
3. Compute the vector of parameters for the exponential distributions, $\lambda = \exp(-A * \text{x.true})$
4. For each λ_i, generate a sample from the exponential distribution. This is the response variable b_i.

Once the test data is generated, we apply glmnet_exp() and h2o.glm() to compute the estimated coeffcients x, and compare it with the true coefficients x.true.

2.3 Sample Code for Timings and Results Comparison between glmnetRcpp and H2O

```
rm(list = ls())

######### load package
library(glmnetRcpp)
library(h2o)

######### helper function
compute_mean_vector = function(data_list){
  tmp_mat = matrix(unlist(data_list), nrow = length(data_list[[1]]))
  apply(tmp_mat, 1, mean)
}

######### set parameters
set.seed(20170827)
nobs = 50
nvars = 7
ntests = 100 # takes a while to run h2o
alpha = 0.5
lasso.lambda = 0.1
x.true = rnorm(nvars)
x.true[sample(2:nvars, floor(nvars / 2) - 1)] = 0
```

```
######## generate test data

params_list = list()
for(j in 1:ntests){
  ## random normal matrix A
  A = matrix(rnorm(nvars * nobs), ncol = nvars)
  exp.lambdas = exp(-A %*% x.true)
  b = sapply(exp.lambdas, function(x) rexp(1, x))
params_list[[j]] = list(
  b = b,
  # The response variables
  A = A,
  # The matrix
  alpha = alpha,
  # 1 means lasso
  lambda = lasso.lambda
) # the regularization coefficient
}

#### try doing things parallel using foreach
#### note that h2o.glm does not work with foreach
library(doParallel)
library(foreach)

h2o.init()
h2o_time_single = system.time({h2o_single_res_list = lapply(params_list,
                                                            function(dat){

                                                              b = dat[["b"]]
                                                              A = dat[["A"]]
                                                              alpha = dat[["alpha"]]
                                                              x.h2o.df = as.h2o(data.frame(b, A))
                                                              predictors = colnames(x.h2o.df)[-1]
                                                              response = colnames(x.h2o.df)[1]
                                                              my.glm.lasso = h2o.glm(
                                                                x = predictors,
                                                                y = response,
                                                                family = 'gamma',
                                                                intercept = FALSE,
                                                                training_frame = x.h2o.df,
                                                                ignore_const_cols = TRUE,
                                                                link = "log",
                                                                #         lambda = enet_lambda,
                                                                lambda_search = TRUE,
                                                                alpha = alpha,
                                                                standardize = FALSE
                                                              )
                                                              return(my.glm.lasso@model$coefficients[-1])
                                                            }
)
}
)
h2o.shutdown(FALSE)

cl <- makeCluster(3)
registerDoParallel(cl)

time_single = system.time({cpp_single_res_list = foreach(i = 1:length(params_list),
                                                         .packages = 'glmnetRcpp') %do% {
                                                           x = params_list[[i]]
                                                           glmnet_exp(x[["A"]], x[["b"]], x[["alpha"]])
```

```
                                            }
})

time_multi = system.time({cpp_multi_res_list = foreach(i = 1:length(params_list),
                                            .packages = 'glmnetRcpp') %dopar% {
                                              x = params_list[[i]]
                                              glmnet_exp(x[["A"]], x[["b"]], x[["alpha"]])
                                            }
})
stopCluster(cl)
res_cpp_single = compute_mean_vector(cpp_single_res_list)
res_cpp_multi = compute_mean_vector(cpp_multi_res_list)
res_h2o_single = compute_mean_vector(h2o_single_res_list)
```

```
### timings
data.frame(cpp_time_single = time_single[3],
           cpp_time_multi = time_multi[3],
           h2o_time_single = h2o_time_single[3])

##         cpp_time_single cpp_time_multi h2o_time_single
## elapsed           0.496          0.313          55.206
```

```
### fitted coefficients
data.frame(x.true = x.true,
           x.single = res_cpp_single,
           x.multi = res_cpp_multi,
           x.h2o = res_h2o_single)

##        x.true     x.single      x.multi        x.h2o
## 1  1.23913750  1.193198732  1.188035011  0.931126065
## 2  0.00000000 -0.001271256 -0.003624581 -0.001349756
## 3  0.44916967  0.473292044  0.470835520  0.298115188
## 4  0.02784615  0.022762432  0.024281721  0.018058563
## 5 -1.58000686 -1.542547236 -1.543901928 -1.233928142
## 6  0.28012296  0.297994504  0.293722912  0.150870989
## 7  0.00000000 -0.001405834  0.004031774 -0.005693584
```

2.4 Testing glmnetRcpp for estimating the standard errors of sample mean

The following code chunk tests using glmnetRcpp to estimate the standard error of sample for IID normal data.

```
rm(list=ls())
library(glmnetRcpp)
library(EstimatorStandardError)

#' Influence Function of Mean
#'
#' @param data Vector of the data
#' @param ... other parameters
#'
#' @return IF of Mean
#' @export
#' @author Xin Chen, \email{chenx26@uw.edu}
#'
#' @examples
#' mu.IF(rnorm(10))
mu.IF=function(data,...){
```

```
        mu.hat=mean(data)
        return(data-mu.hat)
}

#' Compute the standard error of the mean of the data using glmnet_exp
#'
#' @param data vector of data
#' @param d degree of polynomial
#' @param alpha weight for Elastic Net regularizer
#' @param keep what portion of sample spectral density to use for model fitting
#'
#' @return variance of the mean of the data
#' @export
#'

SE.glmnet_exp=function(data, d=7, alpha=0.5, keep=1){

        N=length(data)
        # Step 1: compute the periodograms
        my.periodogram=myperiodogram(data)
        my.freq=my.periodogram$freq
        my.periodogram=my.periodogram$spec

        # remove values of frequency 0 as it does not contain information about the variance
        my.freq=my.freq[-1]
        my.periodogram=my.periodogram[-1]

        # implement cut-off
        nfreq=length(my.freq)
        my.freq=my.freq[1:floor(nfreq*keep)]
        my.periodogram=my.periodogram[1:floor(nfreq*keep)]

        # Step 2: use GLM with BFGS optimization

        # create 1, x, x^2, ..., x^d
        x.mat=rep(1,length(my.freq))
        for(col.iter in 1:d){
                x.mat=cbind(x.mat,my.freq^col.iter)
        }

        # b0 = rnorm(d + 1)

        # fit the glmnet_exp model
        res = glmnet_exp(x.mat, my.periodogram, alpha = alpha)

        # Step 3: return the estimated variance
        return(res[1]/N)
}

# set random seed
set.seed(20180828)

# number of observations for each simlumation run
nobs = 50

# number of simulation runs
n.mcsim = 200

# mean of the data
mu = 0
```

```r
# sd of the data
StdDev = 2

# generate data matrix
test.mat = matrix(rnorm(nobs * n.mcsim, mean = mu, sd = StdDev), nrow = nobs)

# compute se using glmnet_exp
res.glmnet_exp = apply(test.mat, 2,
                       function(x) {
                               data.IF = mu.IF(x)
                               SE.glmnet_exp(data.IF)
                       }
)
```

```r
# comparison of results
data.frame(theoretical.se =StdDev/sqrt(nobs), # theoretical SE of mean is StdDev / sqrt(nobs)
           mc.se = sd(apply(test.mat, 2, mean)), # SE from Monte Carlo assuming true params are known, which is
           glmnet_exp.se = mean(res.glmnet_exp) # SE estimated by glmnet_exp
)

##   theoretical.se     mc.se glmnet_exp.se
## 1      0.2828427 0.2675957    0.02616021
```