GlmNet for Exponential Response Variables via RcppEigen

Xin Chen, Doug Martin, Aleksandr Aravkin, Dan Hanson

August 27, 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 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 = 10
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           0.065          0.136          14.052
```

```r
### 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  1.26173485  1.26286278  0.96495101
## 2  0.00000000 -0.03963640 -0.04045372 -0.03053851
## 3  0.44916967  0.55341752  0.55428483  0.38900770
## 4  0.02784615  0.18367382  0.18460500  0.08783021
## 5 -1.58000686 -1.61850237 -1.61885592 -1.34719797
## 6  0.28012296  0.29655882  0.29756093  0.12119234
## 7  0.00000000 -0.02435039 -0.02632243 -0.01700219
```