Central University of Finance and Economics

中央财经大学

科研项目

glm lasso for R

马景义

吴宇翀

统计与数学学院

2020 年 9 月 25 日

目录

1 Introduction

This document contains definitions of all the functions as well as a test case for the algorithm in the end of the article.

Here is the source file for the algorithm on github.

2 resp

```
resp = function(x, beta, family)
{
  if(family == "logit")
  {
    eta = x %*% beta
    expeta = exp(eta)
    mu = expeta / (1 + expeta)
    return(rbinom(nrow(mu) * ncol(mu), size = 1, prob = mu))
  }
  if(family == "probit")
  {
    mu = x %*% beta
    return(rnorm(nrow(mu) * ncol(mu), mean = mu))
  }
  if(family == "poisson")
  {
    eta = x %*% beta
    mu = exp(eta)
    return(rpois(nrow(mu) * ncol(mu), lambda = mu))
  }
}
```

3 sigma_ma

递归容易超过限制（p = 1000）

```
sigma_ma = function(p, rho)
{
  if(p == 1)
  {
    return(matrix(1, 1, 1))
  } else
  {
    mat_left = matrix(rho ^ seq(p - 1, 1, -1),
                      nrow = p - 1, ncol = 1)
    mat_below = matrix(rho ^ seq(p - 1, 0, -1),
                       nrow = 1, ncol = p)
    mat_above = cbind(sigma_ma(p - 1, rho), mat_left)
    return(rbind(mat_above, mat_below))
  }
}
```

改成循环拼接

```
sigma_ma = function(p, rho)
{
  accum = matrix(1, 1, 1)
  for(i in 2:p)
  {
    mat_left = matrix(rho ^ seq(i - 1, 1, -1),
                      nrow = i - 1, ncol = 1)
    mat_below = matrix(rho ^ seq(i - 1, 0, -1),
                       nrow = 1, ncol = i)
    mat_above = cbind(accum, mat_left)
    accum = rbind(mat_above, mat_below)
  }
  return(accum)
}
```

4 sim_data

```
sim_data = function(beta, rho, n, family)
{
  p = length(beta)
  cov = sigma_ma(p - 1, rho) # matrix
  uper_mat = chol(cov)
  x = matrix(rnorm(n*(p - 1)), n, p - 1)
  x = x %*% uper_mat
  x_1 = cbind(rep(1, n), x)
  y = resp(x_1, beta, family)
  return(list(x, y))
}
```

5 downdating

```
gives_tran = function(mx, lmx)
{
  mc = mx[1] / lmx
  ms = mx[2] / lmx
  tran_mat = matrix(c(mc, -ms, ms, mc), nrow = 2, ncol = 2, byrow = TRUE)
```

```r
  return(tran_mat)
}

downdating = function(left_mat, k)
{
  p = dim(left_mat)[1]
  p = p - 1
  if((k - 1) > p)
  {
    return("Wrong input of k!")
  }
  left_mat_k = left_mat[-k,]
  mk = k
  while((mk - 1) < p)
  {
    mx = left_mat_k[mk, mk:(mk + 1)]
    lmx = sqrt(sum(mx^2))
    left_mat_k[mk, mk] = lmx
    left_mat_k[mk, mk + 1] = 0
    if((mk - 1) < (p - 1))
    {
      tmp_mat = left_mat_k[(mk + 1):p, mk:(mk + 1)]
      tmp_mat = tmp_mat %*% gives_tran(mx, lmx)
      left_mat_k[(mk + 1):p, mk:(mk + 1)] = tmp_mat
    }
    mk = mk + 1
  }
  return(left_mat_k[,-(p+1)])
}
```

6 updating

```r
updating = function(left_mat, xxk, xkxk)
{
  k = dim(left_mat)[1]
  lk = backsolve(left_mat, xxk)
  lkk = sqrt(xkxk - sum(lk * lk))
  left_mat_up = cbind(left_mat, rep(0,k))
  left_mat_down = c(lk, lkk) # not sure cbind or rbind
  return(rbind(left_mat_up, matrix(left_mat_down, nrow = 1, ncol = (k+1))))
}
```

7 lars_iter

7.1 lars_init

```
lars_init = function(w, xt, cc_t, is_active_t, active_set_t)
{
  cc_t_abs = abs(cc_t)
  cci_t = which(cc_t_abs == max(cc_t_abs))[1]
  cc_t_max = cc_t_abs[cci_t]
  lamb_t = cc_t_max
  is_active_t[cci_t] = TRUE
  active_set_t = c(active_set_t, cci_t)
  xt_a = xt[is_active_t,]
  xtx_a = (xt_a * w) %*% t(xt_a)
  return(list(lamb_t, t(chol(xtx_a)), is_active_t, active_set_t))
}
```

7.2 lars_step

```
lars_step = function(xt, w, p, b_t, cc_t, active_set_t, is_active_t, left_mat_t, lamb_t, df_t)
{
  s_t = sign(cc_t)
  sa_t = s_t[active_set_t]
  sa_t[1] = 0
  d_t = forwardsolve(left_mat_t, forwardsolve(left_mat_t, sa_t), transp=TRUE)
  u_t = as.vector(d_t %*% xt[active_set_t,])
  a_t = as.vector(xt[!is_active_t,] %*% (u_t * w))
  gam = rep(1, p) * lamb_t
  if(df_t > 1)
  {
    ww = - b_t[active_set_t] / d_t
    gam[active_set_t] = ifelse((ww > 0) & (ww < lamb_t), ww, lamb_t)
    gam[1] = lamb_t
  }
  if(df_t < (p - 1))
  {
    gam[!is_active_t] = ifelse((a_t * lamb_t) <= cc_t[!is_active_t],
                               (lamb_t - cc_t[!is_active_t]) / (1 - a_t),
                               (lamb_t + cc_t[!is_active_t]) / (1 + a_t))
  }
  return(list(gam, d_t, sa_t, a_t))
}
```

7.3 lars_iter

```
lars_iter = function(y, xt, b_, is_active_, lamb, pmax)
{
  #  -----------------------------输入参数--------------------------------------------  #
  #  y: 因变量，一维数组，shape = (n, ); xt: 自变量，二维数组，shape = (p, n)
  #  lamb: 调节参数
  #  b_ : 初始值
  #  is_active_: b_ 中非 0 元素位置
  #  pmax: 模型中最大非 0 变量个数
  #  --------------------------------------------------------------------------------  #
  b = NULL
  is_active = NULL
  df = NULL
  lamb_next = NULL
  b_next = NULL
  is_active_next = NULL
  p = dim(xt)[1]
  n = dim(xt)[2]
  count1 = 0
  while(count1 < 100)
  {
    count1 = count1 + 1
    eta = as.vector(b_[is_active_] %*% xt[is_active_,])
    exp_eta = exp(eta)
    mu = exp_eta / (1 + exp_eta)
    w = mu * (1 - mu)
    z = y - mu
```

```
is_active_t = c(TRUE, rep(FALSE, p - 1))
active_set_t = c(1)
b_t = rep(0, p)
b_t[1] = sum(eta * w + z) / sum(w)
cc_t = as.vector((xt %*% ((eta * w) - (b_t[1] * w))) + as.vector(xt %*% z))
list = lars_init(w, xt, cc_t, is_active_t, active_set_t)
lamb_t = list[[1]]
left_mat_t = list[[2]]
is_active_t = list[[3]]
active_set_t = list[[4]]
rm(list)
count2 = 0
df_t = 1
gam_min = NULL
gam_min_t = NULL
d_t = NULL
while(count2 < (2 * p))
{
  count2 = count2 + 1
  list = lars_step(xt, w, p, b_t, cc_t, active_set_t, is_active_t, left_mat_t, lamb_t, df_t)
  gam = list[[1]]
  d_t = list[[2]]
  sa_t = list[[3]]
  a_t = list[[4]]
  rm(list)
  j = which(gam == min(gam))[1]
  gam_min = gam[j]
  if((lamb_t - gam_min) < lamb)
  {
    gam_min_t = lamb_t - lamb
  } else
  {
    gam_min_t = gam_min
  }
  b_t[active_set_t] = b_t[active_set_t] + gam_min_t * d_t
  cc_t[active_set_t] = cc_t[active_set_t] - gam_min_t * sa_t
  cc_t[!is_active_t] = cc_t[!is_active_t] - gam_min_t * a_t
  lamb_t = lamb_t - gam_min_t
```

```
      if(lamb_t > lamb)
      {
        if(is_active_t[j])
        {
          k = which(active_set_t == j)[1]
          another = active_set_t[-k]
          left_mat_t = downdating(left_mat_t, k)
          df_t = df_t - 1
        } else
        {
          xt_w = xt[j,] * w
          xtx_j = apply(xt[active_set_t,] * xt_w, 2, sum)
          xtx_jj = sum(xt[j,] * xt_w)
          left_mat_t = updating(left_mat_t, xtx_j, xtx_jj)
          active_set_t = c(active_set_t, j)
          df_t = df_t + 1
        }
        is_active_t[j] = !is_active_t[j]
      } else
      {
        break
      }
    }
    eps = max(abs(b_t - b_))
    b_ = b_t
    # active_set_ = active_set_t
    is_active_ = is_active_t
    # is_active_next = is_active
    if(eps < 1e-8)
    {
      b = b_
      is_active = is_active_
      df = df_t
      is_active_next = is_active
      if(df < pmax)
      {
        b_next = b_t
        if(gam_min_t != gam_min)
        {
          lamb_next = lamb - (gam_min - gam_min_t)
          b_next[active_set_t] = b_next[active_set_t] + (gam_min - gam_min_t) * d_t
        } else
        {
          list = lars_step(xt, w, p, b_t, cc_t, active_set_t, is_active_t, left_mat_t, lamb_t, df_
          gam = list[[1]]
          d_t = list[[2]]
          j = which(gam == min(gam))[1]
          gam_min = gam[j]
          lamb_next = lamb - gam_min
          b_next[active_set_t] = b_next[active_set_t] + gam_min * d_t
          if(lamb - lamb_next < 0.01)
          {
            lamb_next = lamb * 0.95
          }
        }
      }
      break
    }
  }
  return(list(b, is_active, df, lamb_next, b_next, is_active_next))
}
```

8 logit_lasso

8.1 logit_lasso_init

```
logit_lasso_init = function(x, xt, y, n, p)
{
  y_mean = mean(y)
  mu = y_mean * rep(1, n)
  w = mu * (1 - mu)
  b = rep(0, p)
  b[1] = log(y_mean / (1 - y_mean))
  cc = as.vector(xt %*% (y - mu))
  s = sign(cc)
  s[1] = 0
  cc_abs = cc * s
  cc_abs[0] = 0
  tmp = cc * s
  j = which(tmp == max(tmp))[1]
  rm(tmp)
  lamb = cc_abs[j]
  is_active = c(TRUE, rep(FALSE, p - 1))
  is_active_ = is_active
  is_active_[j] = TRUE
  xt_a = xt[is_active_,]
  x_a = x[, is_active_]
  xtx_a = (xt_a * w) %*% x_a
  left_mat = t(chol(xtx_a))
  sa = s[is_active_]
  d = forwardsolve(left_mat, forwardsolve(left_mat, sa), transp=TRUE)
  u = as.vector(d %*% xt[is_active_,])
  a = as.vector(xt[!is_active_,] %*% (u * w))
  gam = rep(1, p) * lamb
  judge = a * lamb <= cc[!is_active_]
  temp = gam[!is_active_]
  temp[ judge] = ((lamb - cc[!is_active_]) / (1 - a))[ judge]
  temp[!judge] = ((lamb + cc[!is_active_]) / (1 + a))[!judge]
  gam[!is_active_] = temp
  rm(temp)
  rm(judge)
  j = which(gam == min(gam))[1]
  gam_min = gam[j]
  b_ = b
  b_[is_active_] = b_[is_active_] + gam_min * d
  lamb_ = lamb - gam_min
  return(list(lamb, b, is_active, lamb_, b_, is_active_))
}
```

8.2 logit_lasso

```
logit_lasso = function(x, y, pmax)
{
  n = dim(x)[1]
  p = dim(x)[2]
  p = p + 1
  x = cbind(rep(1, n), x)
  xt = t(x)
  outcome = logit_lasso_init(x, xt, y, n, p)
  lamb = outcome[[1]]
  b = outcome[[2]]
  is_active = outcome[[3]]
  lamb_ = outcome[[4]]
  b_ = outcome[[5]]
  is_active_ = outcome[[6]]
  rm(outcome)
  df = 0
  while(df < pmax)
  {
    # 输入 lamb_，初始值为 b_：
    # （1）计算数值解 b
    # （2）如果活跃集元素个数小于 pmax，计算下一个 lamb_ 和相应的初始值 b_
    outcome = lars_iter(y, xt, b_, is_active_, lamb_, pmax)
    b = outcome[[1]]
    is_active = outcome[[2]]
    df = outcome[[3]]
    lamb_ = outcome[[4]]
    b_ = outcome[[5]]
    is_active_ = outcome[[6]]
    rm(outcome)
    cat("df=", df, "|")
    print(b[is_active])
  }
  return(list(b, is_active, lamb, df))
}
```

9 main

9.1 Data Simulation

```
family = "logit"
rho = 0.5
```

```
n = 400
p = 1000
s = 6
pmax = 10
beta_0 = c(0.05, 3, -2.5, 3.5, -1.5, -3)
beta_1 = rep(0, p - s)
beta = c(beta_0, beta_1)
sim = sim_data(beta, rho, n, family)
x = sim[[1]]
y = sim[[2]]
```

9.2 A Test Case

```
model = logit_lasso(x, y, pmax=10)
```

```
## df= 1 |[1]  0.1533724 -0.2019749
## df= 1 |[1]  0.1533363 -0.2022288
## df= 1 |[1]  0.1533362 -0.2022298
## df= 1 |[1]  0.1533362 -0.2022298
## df= 2 |[1]  1.533362e-01  1.013934e-12 -2.022298e-01
## df= 2 |[1]  0.1228312  0.2406614 -0.3935732
## df= 2 |[1]  0.1219295  0.2475593 -0.3988674
## df= 2 |[1]  0.1219064  0.2477361 -0.3990030
## df= 2 |[1]  0.1219058  0.2477406 -0.3990064
## df= 2 |[1]  0.1219058  0.2477407 -0.3990065
## df= 2 |[1]  0.1219058  0.2477407 -0.3990065
## df= 3 |[1]  1.219058e-01  2.477407e-01  1.818221e-11 -3.990065e-01
## df= 5 |[1]  0.099216405  0.432572299  0.168420337 -0.542655989 -0.016019603
## [6] -0.005707534
## df= 6 |[1]  0.0960484198  0.4561645322  0.1898854753 -0.0006511551 -0.5585431972
## [6] -0.0392818838 -0.0288154403
## df= 7 |[1]  0.0850530693  0.5388389890  0.2684195863 -0.0928941526 -0.6041745851
## [6] -0.1217272691 -0.1085428689  0.0006296344
## df= 8 |[1]  0.0840953077  0.5461292926  0.2755693556 -0.1011754997 -0.6085165829
## [6] -0.1284619862 -0.0002205599 -0.1151664391  0.0080125328
## df= 8 |[1]  0.083155909  0.554242967  0.283438435 -0.110477021 -0.612900694
## [6] -0.136034463 -0.008679326 -0.122536639  0.016176891
## df= 8 |[1]  0.083111099  0.554633261  0.283817885 -0.110924904 -0.613111307
## [6] -0.136396708 -0.009084931 -0.122889359  0.016568712
## df= 8 |[1]  0.083109018  0.554651398  0.283835520 -0.110945717 -0.613121094
## [6] -0.136413537 -0.009103776 -0.122905746  0.016586917
## df= 8 |[1]  0.083108921  0.554652239  0.283836338 -0.110946683 -0.613121548
## [6] -0.136414318 -0.009104651 -0.122906506  0.016587762
## df= 8 |[1]  0.083108916  0.554652278  0.283836376 -0.110946727 -0.613121569
## [6] -0.136414354 -0.009104691 -0.122906541  0.016587801
## df= 8 |[1]  0.083108916  0.554652279  0.283836377 -0.110946730 -0.613121569
## [6] -0.136414356 -0.009104693 -0.122906543  0.016587803
## df= 9 | [1]  8.310892e-02  5.546523e-01 -4.435958e-11  2.838364e-01 -1.109467e-01
##  [6] -6.131216e-01 -1.364144e-01 -9.104693e-03 -1.229065e-01  1.658780e-02
## df= 10 | [1]  0.0826349497  0.5594590514 -0.0048793695  0.2886677874 -0.1165602851
##  [6] -0.6152841414 -0.1410872037 -0.0003065598 -0.0139480979 -0.1272196269
## [11]  0.0213287005
```

```
b = model[[1]]
is_active = model[[2]]
lamb = model[[3]]
df = model[[4]]

print(b[1:10])
```

```
##  [1]  0.0826349497  0.5594590514 -0.0048793695  0.2886677874 -0.1165602851
##  [6] -0.6152841414 -0.1410872037  0.0000000000  0.0000000000 -0.0003065598
```

```
print(b[is_active])
```

```
##  [1]  0.0826349497  0.5594590514 -0.0048793695  0.2886677874 -0.1165602851
##  [6] -0.6152841414 -0.1410872037 -0.0003065598 -0.0139480979 -0.1272196269
## [11]  0.0213287005
```

```
print(lamb)
```

```
## [1] 92.94698
```