SDS 385 Ex 07: ADMM for LASSO

October 18, 2016

Jennifer Starling

ADMM for LASSO

Algorithm Details

I used the ADMM algorithm as detailed in Section 6.4 of Boyd, "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers", 2011.

The setting for this algorithm as applied to lasso is below.

Objective Function:

minimize $\frac{1}{2}||X\beta - y||_2^2 + \lambda||x||_1$
with $\lambda > 0$ as the l1 regularization penalty parameter.

ADMM Form of Problem:

minimize $f(x) + g(x)$
subject to $x - z = 0$
where:
$f(x) = \frac{1}{2}||X\beta - y||_2^2$
$g(x) = \lambda||x||_1$

ADMM Steps:

$\beta^{k+1} = (X^T X + \rho I)^{-1}(X^T y + \rho(z^k - u_k))$
$z^{k+1} = S_{\lambda/\rho}(\beta^{k+1} + u^k)$
$u^{k+1} = u^k + \beta^{k+1} - z^{k+1}$

where $S_{\lambda\rho}(a) = sign(a)(|a| - \lambda/\rho)_+$
and u^k is the augmented lagrangian at the k^{th} iteration.

Implementation Notes:
• Since the step size is fixed, to increase efficiency, I cached the matrix inverse $(X^T X + \rho I)^{-1}$ before beginning iterations.
• Boyd notes that when not using fixed step size, it is best to still cache this inverse, and only update it when the ρ step size is updated.

My Implementation

My ADMM implementation for LASSO converged in 230 iterations. This was considerably faster than my proximal gradient descent (7687 iterations) and my accelerated proximal gradient descent (1853 iterations). I used $\lambda = .01$ for the l1 lasso penalty and $\rho = .01$ for the step size. I used the convergence criteria of absolute change in the lasso objective function $< tol$, where $tol = 1E - 10$.

Note that the lambda and rho were the same values used for exercise 6, for consistency. In practice, I would select an optimal lambda value via cross-validation.

The plot of the lasso objective function is as follows.

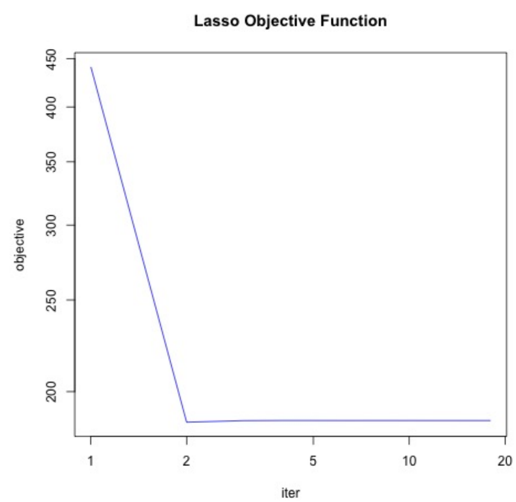

Figure 1: LASSO objective using ADMM($\hat{\alpha}$)

The estimated beta coefficients obtained using admm are compared with the coefficients from glmnet below. Most appear to be of reasonably similar sign and magnitude.

```
              glmnet        admm
age      0.0084994686  0.03007395
sex     -0.1274718132 -0.16390462
bmi      0.3053578438  0.28268400
map      0.1928804642  0.21180397
tc      -0.0387865853 -0.10040001
ldl      .             .
hdl     -0.1635236545 -0.10958174
tch      .             0.05421951
ltg      0.3230766076  0.42695740
glu      0.0337240035  0.03968565
age.2    0.0351045119  0.04230375
bmi.2    0.0238787313  0.02913876
map.2    .            -0.00496873
tc.2     .             3.32992803
```

```
ldl.2     .             1.78879470
hdl.2     .             0.86057254
tch.2     .             0.45016330
ltg.2     .             0.71081295
glu.2     0.0636440798  0.07115055
age.sex   0.0961729059  0.09425725
age.bmi   .            -0.00974232
age.map   0.0145699540  0.01097956
age.tc    .            -0.10307316
age.ldl  -0.0389574124 -0.03850246
age.hdl   0.0258647293  0.13295890
age.tch   .             0.11473838
age.ltg   0.0407916229  0.07678043
age.glu   0.0184222925  0.03852093
sex.bmi   0.0270461479  0.04070618
sex.map   0.0387600107  0.05455798
sex.tc    .             0.25201571
sex.ldl  -0.0181954985 -0.20580754
sex.hdl   0.0429324533 -0.06940488
sex.tch   .            -0.07673176
sex.ltg   .            -0.07203117
sex.glu   .             0.02857881
bmi.map   0.0837040826  0.09607870
bmi.tc   -0.0002777938 -0.22702825
bmi.ldl   .             0.18330034
bmi.hdl   .             0.08852731
bmi.tch   .            -0.01969956
bmi.ltg   .             0.07900017
bmi.glu   .             0.01519782
map.tc    0.0083855329  0.28434387
map.ldl   .            -0.19175559
map.hdl   0.0233788843 -0.11053647
map.tch   .            -0.03567046
map.ltg   .            -0.09184804
map.glu  -0.0362793960 -0.08424349
tc.ldl    .            -4.61510706
tc.hdl    0.0072487602 -1.93720146
tc.tch   -0.0477242186 -1.10984540
tc.ltg   -0.0098056312 -2.26554521
tc.glu    .            -0.08709231
ldl.hdl   .             1.26150669
ldl.tch   .             0.52479375
ldl.ltg   0.0742412609  1.70305929
ldl.glu   0.0067025050  0.03256134
hdl.tch  -0.0522309970  0.62815286
hdl.ltg   .             0.84045023
hdl.glu   .             0.12679103
tch.ltg  -0.0607125224  0.15900559
tch.glu   0.0214432611  0.14614305
```

```
ltg.glu   .            0.04353096
```

Appendix: R Code

```
### SDS 385 - Exercises 07 - ADMM for LASSO.
### ADMM = Alternating Direction Method of Multipliers

#Jennifer Starling
#18 October 2016

rm(list=ls())   #Clean workspace.

library(glmnet)
library(Matrix)

#Read in Diabetes.csv data.
X <- read.csv(file='/Users/jennstarling/UTAustin/2016_Fall_SDS 385_Big_Data/
    Exercise 05 R Code/DiabetesX.csv',header=T)
y <- read.csv(file='/Users/jennstarling/UTAustin/2016_Fall_SDS 385_Big_Data/
    Exercise 05 R Code/DiabetesY.csv',header=F)

#Scale X and y.
X = scale(X)
y = scale(y)

#-----------------------------------------------------
#LASSO objective function:
#Inputs:
#   X = X matrix (scaled)
#   y = response data (scaled)
#   lambda = a chosen lambda value
#   beta = a vector of beta coefficients.
#Output:
#   Value of the LASSO objective function at specified inputs.
fx <- function(X,y,lambda,beta){
    obj = (1/2) * sum((y - X %*% beta) ^ 2) + lambda * sum(abs(beta))
    return(obj)
}

#-----------------------------------------------------
#Proximal L1 Operator function: (soft thresholding operator)
prox_l1 <- function(x, lambda){

  # Computes the soft thresholding estimator
  # -------------------------------------------
  # Args:
  #   - x: vector of the observations
  #   - lambda: penalization parameter (threshold)
  # Returns:
  #   - theta: the soft thresholding estimator
  # ---------------------------------------------
  theta <- sign(x) * pmax(rep(0, length(x)), abs(x) - lambda)
  return (theta)
}

#-----------------------------------------------------
#ADMM for Lasso:
#Inputs:
#   X = design matrix (A)
#   y = response vector (b)
```

```
#   rho = step size
#   maxiter = maximum iterations
#   eps_abs = primal tolerance for convergence (epsilon_abs)
#   eps_rel = dual tolerance for convergence (epsilon_rel)
#   lambda = l1 norm penalty constant.
#Output:
#   List including estimated beta (x) values and objective function.
#Note: In optimization notation, A=X, b=Y, x=beta (minimizing x).

admmLasso = function(X,Y,rho=1,lambda=.1,maxiter=1000,e_abs=1E-3,e_rel=1E-6){

    #Define dimensions n and p.
    n = nrow(X)
    p = ncol(X)

    #Rescale lambda to match glmnet results.
    lambda = lambda * n

    #Define function Euclidian (l2) norm of a vector.
    l2norm <- function(x) sqrt(sum(x^2))

    i=0                 #Initialize iterator.
    converged <- 0      #Indicator for whether convergence met.

    #Initialize data structures.
    betas <- matrix(0,nrow=maxiter,ncol=p)  #holds beta vector for each iteration.
    obj <- rep(0,maxiter)   #Initialize vector to hold loglikelihood fctn.
    z = matrix(0,nrow=maxiter,ncol=p)   #Initialize z vector to all zeros.

    #Initialize values.
    obj[1] <- fx(X,y,lambda,betas[1,])  #Initialize objective.
    betas[1,] <- rep(0,p)   #Initialize beta vector to 0 to start.
    u = rep(0,p)    #Initialize the lagrangian to all zeros.

    #Pre-cache matrix inverse and Xty, since using fixed step size for each iter.
    Xty = crossprod(X,y)
    inv = solve(crossprod(X) + diag(rep(rho,p)))

    #Initialize residual vectors.
    s = 0   #dual residual
    r = 0   #primal residual

    #ADMM looping.
    for (i in 2:maxiter){

        #Update betas.
        betas[i,] = inv %*% (Xty + rho * (z[i,]-u) )

        #Update z.
        z[i,] = prox_l1(betas[i,] + u,lambda/rho)

        #Update u (lagrangian).
        u = u + betas[i,] - z[i,]

        #Update objective function.
        obj[i] = fx(X,y,lambda=lambda,beta=betas[i,])

        #-------------------------------
        #Convergence check:
```

```
        #Calculate residuals for iteration i.
        r = betas[i,] - z[i,]
        s = -rho * (z[i,] - z[i-1,])

        r.norm = l2norm(r)
        s.norm = l2norm(s)

        e.primal = sqrt(p)*e_abs + e_rel * max(l2norm(betas[i,]), l2norm(z[i,]))
        e.dual =  sqrt(p)*e_abs + e_rel * l2norm(u)

        if (r.norm <= e.primal && s.norm <= e.dual){
            converged=1
            break
        }
        #--------------------------
    }

    #Return function values.
    return(list(obj=obj, betas=betas, beta_hat=betas[i,], converged=converged,
        iter=i))
}
#----------------------------------------------

#Run admm for lasso.
output <- admmLasso(X,y,rho=5,lambda=.01,maxiter=1000,e_abs=1E-6,e_rel=1E-2)

#Iterations to convergence:
print(output$iter)
print(output$converged)

#Plot objective function.
#jpeg(file='/Users/jennstarling/UTAustin/2016_Fall_SDS 385_Big_Data/Exercise 07
    LaTeX Files/admm_objective.jpg')
plot(1:output$iter,output$obj[1:output$iter],type='l',col='blue',log='xy',
    main='Lasso Objective Function',xlab='iter',ylab='objective')
#dev.off()

#Compare results to glmnet:
myLasso <- glmnet(X,y,family='gaussian',alpha=1,lambda=.01,intercept=F,standardize
    =F)   #Fit lasso glmnet model.
beta_glmnet <- myLasso$beta                                #Save glmnet betas.
cbind(glmnet=beta_glmnet,admm=round(output$beta_hat,8))    #Output comparison
```