proximal gradient method

Su Chen

October 19, 2016

function to calculate negative likelihood with lasso penalty

```
nll_lasso = function(beta, x, y, lambda)
{
  xbeta = x %*% beta
  res = y - xbeta
  nll = crossprod(res)/2 + lambda * sum( abs(beta) )

  return (nll)
}
```

function to calculate negative likelihood

```
nll = function(beta, x, y)
{
  xbeta = x %*% beta
  res = y - xbeta
  nll = crossprod(res)/2

  return (nll)
}
```

function to calculate gradient

```
grad = function(beta, x, y)
{
  xbeta = x %*% beta
  res = y - xbeta
  grad = -t(x) %*% res

  return (grad)
}
```

function to calculate soft threshold

```
soft_threshold = function(y,lambda)
{
  z = abs(y) - lambda
  theta = sign(y)*z*(z > 0)
  return (theta) # this is max(z, 0) element wise
}
```

proximal gradient algorithm

```r
prox_grad = function(x, y, beta0, step=0.01, lambda, iter_max=10000, precision=1e-10)
{
  p = length(beta0)
  beta = matrix(0, nrow = iter_max, ncol = p)
  beta[1,] = beta0
  nll_track = rep(0, iter_max)
  nll_track[1] = nll(beta0, x, y)
  nll_track_lasso = rep(0, iter_max)
  nll_track_lasso[1] = nll_lasso(beta0, x, y, lambda)
  delta = 1
  iter = 1

  while ( (delta >= precision) && (iter <= iter_max) )
  {
    u = beta[iter,] - step*grad(beta[iter,], x, y)
    beta[iter+1,] = soft_threshold(u, lambda*step)
    nll_track_lasso[iter+1] = nll_lasso(beta[iter+1,], x, y, lambda)
    nll_track[iter+1] = nll(beta[iter+1,], x, y)
    delta = abs(nll_track_lasso[iter+1] - nll_track_lasso[iter])/nll_track_lasso[iter]
    iter = iter + 1
  }

  return (list(iter, nll_track_lasso, nll_track, beta))
}
```

accelerated proximal gradient algorithm

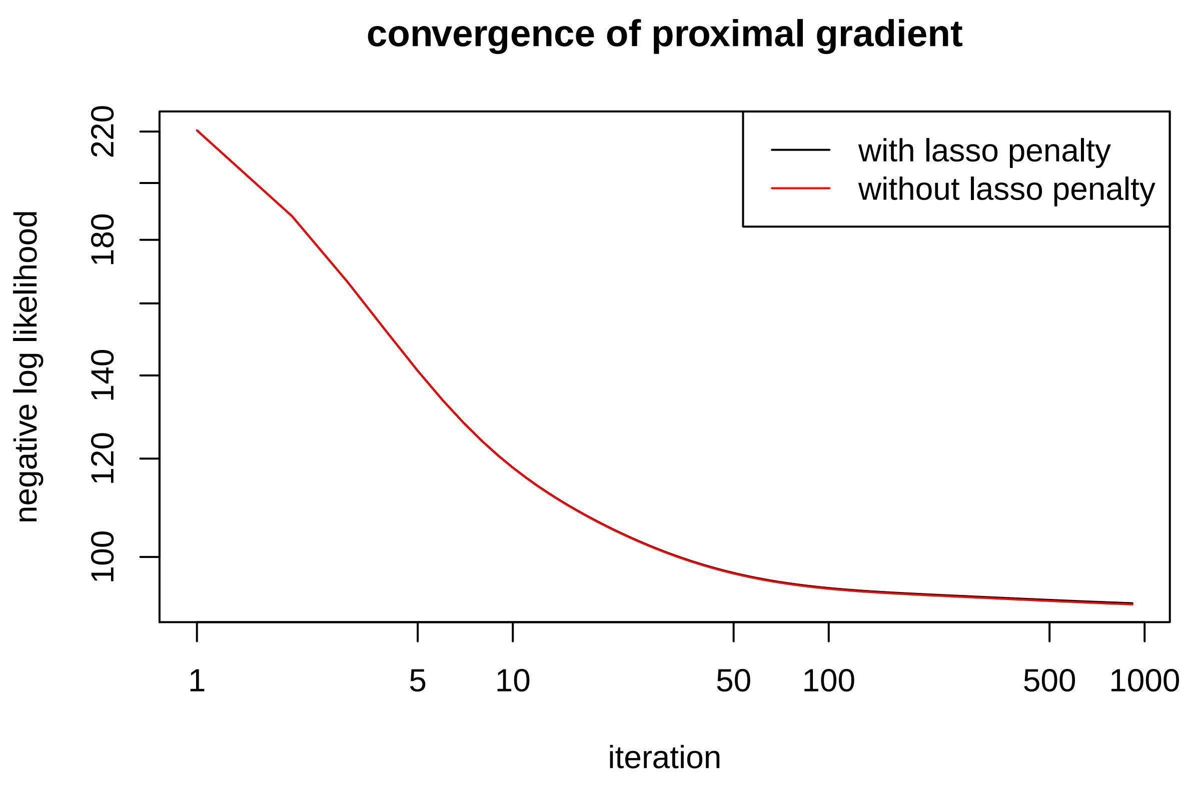
```r
acc_prox_grad = function(x, y, beta0, step=0.01, s=2, lambda, iter_max=10000, precision=1e-10)
{
  p = length(beta0)
  beta = matrix(0, nrow = iter_max, ncol = p)
  beta[1,] = beta0
  nll_track = rep(0, iter_max)
  nll_track[1] = nll(beta0, x, y)
  nll_track_lasso = rep(0, iter_max)
  nll_track_lasso[1] = nll_lasso(beta0, x, y, lambda)
  delta = 1
  iter = 1
  z = beta0

  while ( (delta >= precision) && (iter < iter_max) )
  {
    u = z - step*grad(z, x, y)
    beta[iter+1,] = soft_threshold(u, lambda*step)
    s_update = (1 + sqrt(1 +4*s^2))/2
    z = beta[iter+1,] + (s - 1)/s_update * (beta[iter+1,] - beta[iter,])
    #z = beta[iter,] + (s - 1)/s_update * (beta[iter,] - beta[iter+1,])
    nll_track_lasso[iter+1] = nll_lasso(beta[iter+1,], x, y, lambda)
    nll_track[iter+1] = nll(beta[iter+1,], x, y)
    delta = abs(nll_track_lasso[iter+1] - nll_track_lasso[iter])/nll_track_lasso[iter]
    iter = iter + 1
    s = s_update
  }
  #nll_track_lasso[1:100]

  return (list(iter, nll_track_lasso, nll_track, beta))
}
```

run and compare beta with glmnet output

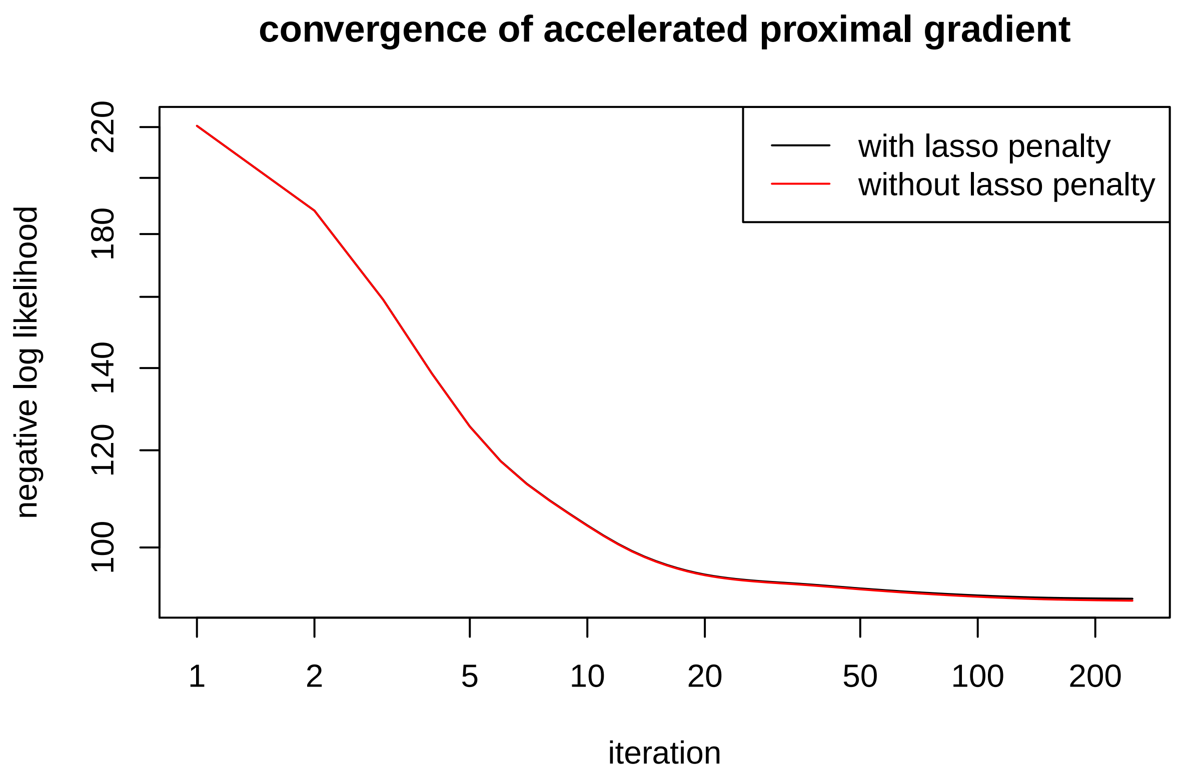
```
diabetesX = read.csv("C:/Users/schen/Dropbox/toChensu/Stats/2016Fall/Big Data/Assignment5/diabetesX.csv
diabetesY = read.csv("C:/Users/schen/Dropbox/toChensu/Stats/2016Fall/Big Data/Assignment5/diabetesY.csv
X = as.matrix(diabetesX)
Y = as.matrix(diabetesY)
Xs = scale(X)
Ys = scale(Y)
beta0 = rep(0, ncol(Xs))


### use cv.glmnet to choose optimal lambda ###
library(glmnet)
```

```
## Loading required package: Matrix

## Loading required package: foreach

## Loaded glmnet 2.0-5
```

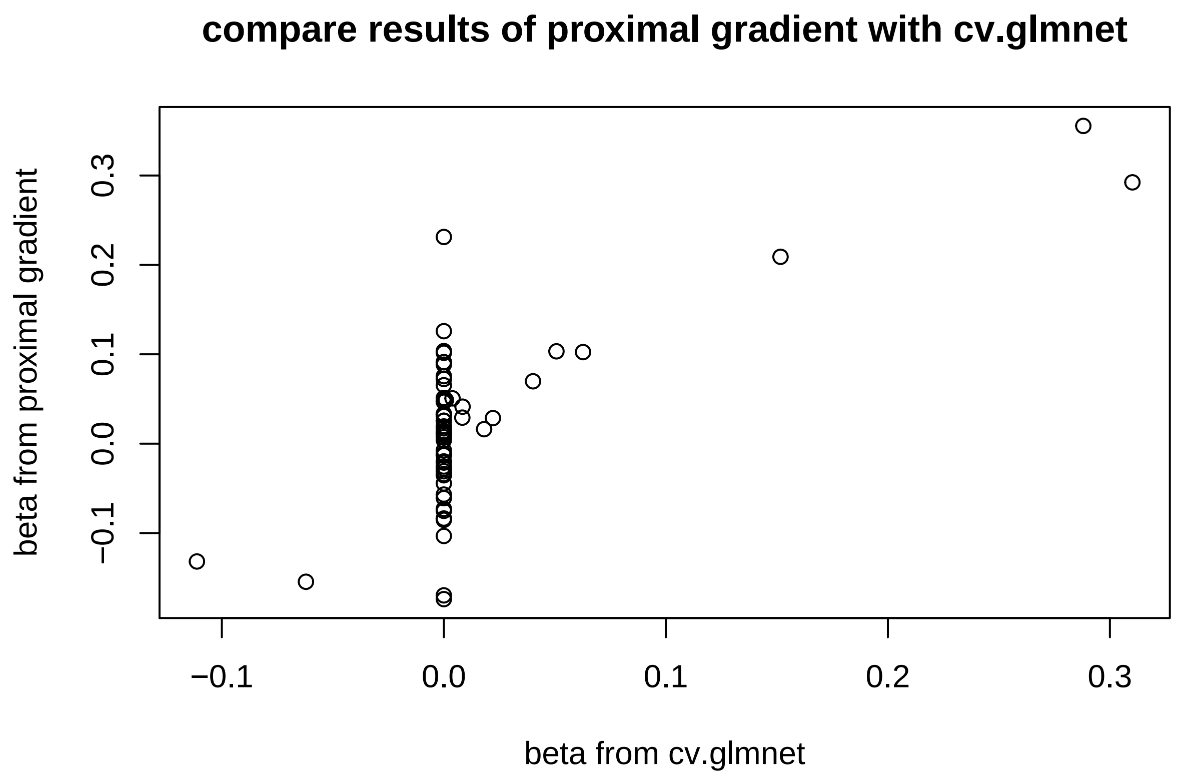
```
cv.fit = cv.glmnet(x = Xs, y = Ys)
cv.lambda = cv.fit$lambda.min
fit.cv.lambda = glmnet(x = Xs, y = Ys, lambda = cv.lambda)
glmnet_beta = fit.cv.lambda$beta


result = prox_grad(x=Xs, y=Ys, beta0, step=0.0001, lambda=cv.lambda, iter_max=10000, precision=1e-5)
plot_iter = result[[1]]
result_nll_lasso = result[[2]][1:plot_iter]
result_nll = result[[3]][1:plot_iter]
result_beta = result[[4]][plot_iter,]
plot(x = 1:plot_iter, y = result_nll_lasso, type = "l",
     xlab = "iteration", ylab = "negative log likelihood",
     log = "xy", main = "convergence of proximal gradient")
lines(x = 1:plot_iter, y = result_nll, col = "red")
legend("topright", c("with lasso penalty","without lasso penalty"),
       col = c("black", "red"), lty = c(1,1))
```



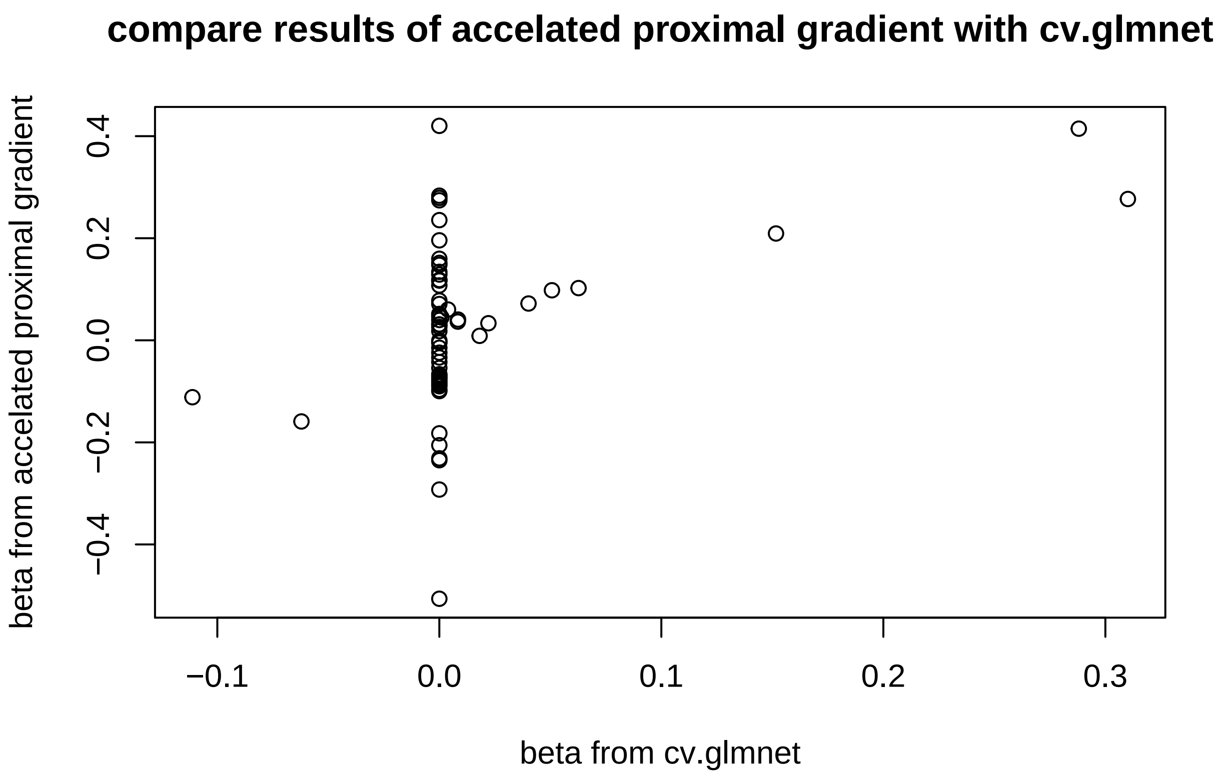
```r
result_acc = acc_prox_grad(x=Xs, y=Ys, beta0, step=0.0001, s=2, lambda=cv.lambda, iter_max=10000, preci
plot_iter_acc = result_acc[[1]]
result_nll_lasso_acc = result_acc[[2]][1:plot_iter_acc]
result_nll_acc = result_acc[[3]][1:plot_iter_acc]
result_beta_acc = result_acc[[4]][plot_iter_acc,]
plot(x = 1:plot_iter_acc, y = result_nll_lasso_acc, type = "l",
     xlab = "iteration", ylab = "negative log likelihood",
     log = "xy", main = "convergence of accelerated proximal gradient")
lines(x = 1:plot_iter_acc, y = result_nll_acc, col = "red")
legend("topright", c("with lasso penalty","without lasso penalty"),
       col = c("black", "red"), lty = c(1,1))
```



```r
plot(x = glmnet_beta, y = result_beta, xlab = "beta from cv.glmnet",
     ylab = "beta from proximal gradient",
     main = "compare results of proximal gradient with cv.glmnet")
```



```r
plot(x = glmnet_beta, y = result_beta_acc, xlab = "beta from cv.glmnet",
     ylab = "beta from accelated proximal gradient",
     main = "compare results of accelated proximal gradient with cv.glmnet")
```


compare results of accelated proximal gradient with cv.glmnet
beta from accelated proximal gradient
0.4
0.2
0.0
−0.2
−0.4
−0.1
0.0
0.1
0.2
0.3
beta from cv.glmnet