Lasso

Lasso

This article applies subgradient descent, proximal gradient descent and accelerated proximal gradient descent methods to lasso, and The convergence speed and convergence accuracy of the three methods are compared.

The objective function of lasso can be written as

$$min_\beta \frac{1}{2}||Y - X\beta||^2 + \lambda||\beta||_1$$

(a) Algorithm steps

1. Subgradient descent applied to Lasso

Algorithm 1 Subgradient descent applied to Lasso

Initialization: X^0, ρ;
for $k = 0, 1, 2, ...$ **do**
    Compute t_k and $\partial\ \frac{1}{2}||Y - X\beta^{(k)}||^2 + \lambda||\beta||_1\ = -X(Y - X\beta^{(k)}) + \lambda sgn(\beta)$;
    Update $\beta^{(k+1)}$, $\beta^{(k+1)} = \beta^{(k)} - t_k\ f(\beta^k)) = \beta^{(k)} - t_k X(X\beta^{(k)} - Y)$;
    If $||\beta^{(k+1)} - \beta^{(k)}||^2 < \rho$, break
end for

Where

$$sgn(\beta) = \begin{cases} -1 & if \quad \beta < 0 \\ 0 & if \quad \beta = 0 \\ 1 & if \quad \beta > 0 \end{cases}$$

2. Proximal gradient descent applied to Lasso

Algorithm 2 Proximal gradient descent applied to Lasso

Initialization: X^0, ρ;
for $k = 0, 1, 2, ...$ **do**
    Compute $t_k = \frac{1}{M}$ and $\frac{1}{2}||Y - X\beta^{(k)}||^2 = -X(Y - X\beta^{(k)})$;
    Compute $\beta_{temp}^{(k+1)} = \beta^{(k)} - t_k\ f(\beta^{(k)})) = \beta^{(k)} - t_k\beta(Y - X\beta^k)$;
    Update $\beta^{(k+1)}$, $\beta^{(k+1)} = Prox_{\lambda||||_1}(\beta_{temp}^{(k+1)})$
    If $||\beta^{k+1} - \beta^k||^2 < \rho$, break
end for

3. Accelerated proximal gradient descent applied to Lasso by using Nestorov's Method

Algorithm 3 Accelerated proximal gradient descent applied to Lasso by using Nestorov's Method

Initialization: X^0, ρ;
for $k = 0, 1, 2, ...$ **do**
    Compute $t_k = \frac{1}{M}$ and $\frac{1}{2}||Y - X\beta^{(k)}||^2 = -X(Y - X\beta^{(k)})$;
    Compute $\beta_{temp}^{(k+1)} = \beta^{(k)} - t_k\, f(\beta^{(k)})) = \beta^{(k)} - t_k\beta(Y - X\beta^k)$;
    $\beta_x^{k+1} = \beta_{temp}^{(k+1)} + \frac{k}{k+3}(\beta_{temp}^{(k+1)} - \beta_{temp}^{(k)})$
    Update $\beta^{(k+1)}$, $\beta^{(k+1)} = Prox_{\lambda||||_1}(\beta_x^{(k+1)})$;
    If $||\beta^{(k+1)} - \beta^{(k)}||^2 < \rho$, break
end for

In these algorithm, the proximal operator of L1 norm is

$$Prox_{\lambda||||_1}(\beta) = \begin{cases} \beta_i - \lambda & \beta_i > \lambda \\ 0 & |\beta_i| > \lambda \\ \beta_i + \lambda & \beta_i < -\lambda \end{cases}$$

(b) R code

1.

```
hess.loss.max <- function(X){
hess.loss = 2 * (t(X) %*% X)
hess.ev = eigen(hess.loss)$values
max(hess.ev)

}


sub_lasso<-function(beta)
{
  new_beta=0
  for (i in 1:length(beta)){
    if (beta[i]>0){
      new_beta[i]=1
    }else if(beta[i]<0){
      new_beta[i]=-1
    }else{
      new_beta[i]=0
    }
  }
  return(new_beta)
}


SubGradientDescent<-function(X,y,maxiter,lambda,beta_s)
{

  rownumber<-nrow(X)
  colnumber<-ncol(X)
  iter<-1
  error_sum=rep(0,maxiter)
```

```
  t=1/hess.loss.max(X)
  beta=as.matrix(rep(1,colnumber))

while(iter<maxiter){
    iter<-iter+1
    grad= t(X) %*% (X%*%beta-y)+sub_lasso(beta)
    beta_new=beta-t*grad
    error_sum[iter-1]=log(sqrt(sum((beta_new-beta_s)^2)))
    beta=beta_new

}
      out=list(beta_output=beta,error_output=error_sum)
      return(out)
}
```

2.

```
proximal_lasso<-function(beta,lambda)
{
  new_beta=0
  for (i in 1:length(beta)){
    if (beta[i]>lambda){
      new_beta[i]=beta[i]-lambda
    }else if(beta[i]<(-lambda)){
      new_beta[i]=beta[i]+lambda
    }else{
      new_beta[i]=0
    }
  }
  return(new_beta)
}


ProxiGradientDescent<-function(X,y,maxiter,lambda,beta_s)
{

  rownumber<-nrow(X)
  colnumber<-ncol(X)
  iter<-1
  error_sum=rep(0,maxiter)
  t=1/hess.loss.max(X)
  beta=as.matrix(rep(1,colnumber))

while(iter<maxiter){
    iter<-iter+1
    grad= t(X) %*% (X%*%beta-y)
    beta_new=beta-t*grad
    beta_new=proximal_lasso(beta_new,lambda)
    error_sum[iter-1]=log(sqrt(sum((beta_new-beta_s)^2)))
    beta=beta_new

}
      out=list(beta_output=beta,error_output=error_sum)
```

```
      return(out)
}
```

3.

```
ACCProxiGradientDescent<-function(X,y,maxiter,lambda,beta_s)
{

  rownumber<-nrow(X)
  colnumber<-ncol(X)
  iter<-1
  error_sum=rep(0,maxiter)
  t=1/hess.loss.max(X)
  beta=as.matrix(rep(1,colnumber))

while(iter<maxiter){
    iter<-iter+1
    grad= t(X) %*% (X%*%beta-y)
    beta_new=beta-t*grad
    beta_new=beta_new+(iter/(iter+3))*(beta_new-beta)
    beta_new=proximal_lasso(beta_new,lambda)
    error_sum[iter-1]=log(sqrt(sum((beta_new-beta_s)^2)))
    beta=beta_new

}
      out=list(beta_output=beta,error_output=error_sum)
      return(out)
}
```

(c) Results

Firstly, we generate data.

```
generate_data <- function(n=200, d=100, s=5, seed =3) {
set.seed(seed)
beta <- numeric(length = d)
beta [1:s] <- 1
X <- matrix(rnorm(n*d), nrow=n, ncol=d)
y <- c(X %*% beta) + rnorm(n)
list(X=X, y=y, beta=beta)
}
data <- generate_data(n=2000, d=300, seed =123)
beta_s=data$beta
```

In order compare the differences in convergence speed, I set $t = \frac{1}{M}$, then

```
maxit=1000
output_sub=SubGradientDescent(data$X,data$y,maxit,0.1,beta_s)
output_prox=ProxiGradientDescent(data$X,data$y,maxit,0.1,beta_s)
output_ACC=ACCProxiGradientDescent(data$X,data$y,maxit,0.1,beta_s)
plot(output_sub$error_output[-maxit],type="l",col=2,ylab="log-MSE",main="lambda=0.1")
```

```
lines(output_prox$error_output[-maxit],type="l",col=3)
lines(output_ACC$error_output[-maxit],type="l",col=4)
legend("topright", legend=c("Subgradient", "Proximal gradient","Accelerate gradient"),col=c(2,3,4),lty=
```

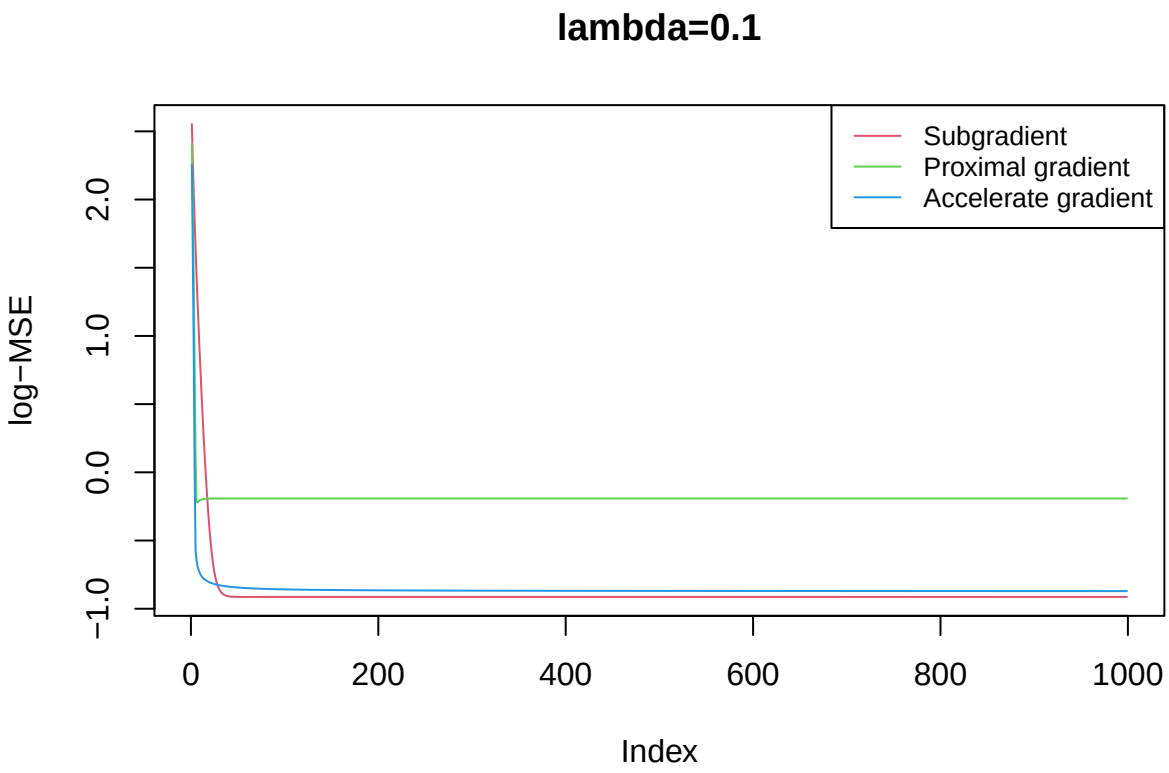

Table 1: For $n = 2000$, $\lambda = 0.1$, $d \in [100, 200, 300]$

d	Method	Subgradient	Proximal gradient	Accelarate gradient
100	Time per iteration (ms)	0.0033	0.006	0.02
100	terations until overfit	15	5	3
100	Time until overfit (ms)	0.05	0.03	0.06
200	Time per iteration (ms)	0.0104	0.0214	0.022
200	terations until overfit	22	7	5
200	Time until overfit (ms)	0.23	0.15	0.11
300	Time per iteration (ms)	0.0157	0.064	0.07
300	terations until overfit	28	7	5
300	Time until overfit (ms)	0.44	0.32	0.35

Table 2: For $\lambda = 0.1$, $d = 200$, $n \in [1000, 2000, 3000]$

n	Method	Subgradient	Proximal gradient	Accelarate gradient
1000	Time per iteration (ms)	0.0044	0.01125	0.01
1000	terations until overfit	25	8	5
1000	Time until overfit (ms)	0.11	0.09	0.05

n	Method	Subgradient	Proximal gradient	Accelarate gradient
2000	Time per iteration (ms)	0.0104	0.0214	0.022
2000	terations until overfit	22	7	5
2000	Time until overfit (ms)	0.23	0.15	0.11
3000	Time per iteration (ms)	0.0178	0.048	0.03
3000	terations until overfit	19	5	4
3000	Time until overfit (ms)	0.34	0.24	0.12

Table 3: For $n = 1000$, $d = 200$, $\lambda \in [0.1, 0.2, 0.3]$

λ	Method	Subgradient	Proximal gradient	Accelarate gradient
0.1	Time per iteration (ms)	0.0044	0.01125	0.01
0.1	iterations until overfit	25	8	5
0.1	Time until overfit (ms)	0.11	0.09	0.05
0.2	Time per iteration (ms)	0.0046	0.0067	0.022
0.2	iterations until overfit	28	15	10
0.2	Time until overfit (ms)	0.13	0.1	0.22
0.3	Time per iteration (ms)	0.0035	0.007	0.006
0.3	iterations until overfit	31	10	5
0.3	Time until overfit (ms)	0.11	0.07	0.03

(d) Discussion

From three tables above, we can conclude that

Tables above show that larger the number of features is, more "time per iteration" will be. The subgradient method has the least "time per iteration". Proximal gradient descent and Accelerate gradient descent method, these two methods have similar "time per iteration".

For "terations until overfit", subgradient gardient descent has the largest "iterations until overfit", Proximal gradient descent is the next, and Accelerate gradient descent method has the least number of iterations until overfit.

So, we can see that, Accelerate gradient descent method is the fastest method in these three method, Proximal gradient descent method comes next.