STAT 542: Homework 3

Spring 2020, by Yifan Shi (yifans16)

Due: Monday, Mar 2 by 11:59 PM

Contents

Question 1 [70 Points] Lasso and Coordinate Descent

We will write our own lasso algorithm. You should only use functions in the R base package to write your lasso code. However, you can use other packages to generate data and check your answers. Perform the following steps.

a. [15 points]

Let's start with the objective function

$$\ell(\boldsymbol{\beta}) = \frac{1}{2n}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda|\boldsymbol{\beta}|_1$$

Since we will utilize coordinate descent, derive the solution of β_j while holding all other parameters fixed. In particular, you should write the solution as a "soft-thresholding" function, which is in a similar format as Page 36 in the lecture note `Penalized`. Finally, write an R function for soft-thresholding in the form of `soft_th <- function(b, lambda)`. The function outputs * $b - \lambda$ if $b > \lambda$ * $b + \lambda$ if $b < -\lambda$ * 0 otherwise

Answer:

$$Loss(\boldsymbol{\beta}) = \frac{1}{2n}\|\mathbf{y} - \mathbf{X}_{(-j)}\boldsymbol{\beta}_{(-j)} - \mathbf{X}_j\boldsymbol{\beta}_j\|_2^2 + \lambda|\boldsymbol{\beta}_j|_1 + \lambda\sum_{i\neq j}^{p}\boldsymbol{\beta}_i$$

$$Then\ set\ \frac{\partial L}{\partial \boldsymbol{\beta}_j} = 0\ and\ \mathbf{r} = \mathbf{y} - \mathbf{X}_{(-j)}\boldsymbol{\beta}_{(-j)}$$

$$if\ \boldsymbol{\beta}_j > 0 : \frac{1}{2n}(2) - \mathbf{X}_j^T(\mathbf{r} - \mathbf{X}_j\boldsymbol{\beta}_j) + \lambda = 0$$

$$\boldsymbol{\beta}_j = \frac{1}{n}\mathbf{X}_j^T\mathbf{r} - \lambda\ with\ \lambda < \frac{1}{n}\mathbf{X}_j^T\mathbf{r}$$

$$if\ \boldsymbol{\beta}_j < 0 : \frac{1}{2n}(2) - \mathbf{X}_j^T(\mathbf{r} - \mathbf{X}_j\boldsymbol{\beta}_j) - \lambda = 0$$

$$\boldsymbol{\beta}_j = \frac{1}{n}\mathbf{X}_j^T\mathbf{r} + \lambda\ with\ -\lambda > \frac{1}{n}\mathbf{X}_j^T\mathbf{r}$$

$$Thus\ \boldsymbol{\beta}_j = \begin{cases} \frac{1}{n}\mathbf{X}_j^T\mathbf{r} - \lambda & \lambda < \frac{1}{n}\mathbf{X}_j^T\mathbf{r} \\ \frac{1}{n}\mathbf{X}_j^T\mathbf{r} + \lambda & -\lambda > \frac{1}{n}\mathbf{X}_j^T\mathbf{r} \\ 0 & otherwise \end{cases}$$

```r
soft_th <- function(b,lambda){
  if (b > lambda) {
    return(b - lambda)
  }
  else if (b < -lambda) {
    return(b + lambda)
  }
  else {return(0)}
}
```

b. [15 points]

Let's write one iteration of the coordinate descent algorithm that goes through variables 1 to p and update their parameter values. First, we will generate the following data:

```r
  library(MASS)
  set.seed(1)
  n = 200
  p = 500

  # generate data
  V = matrix(0.2, p, p)
  diag(V) = 1
  X_org = as.matrix(mvrnorm(n, mu = rep(0, p), Sigma = V))
  true_b = c(runif(10, -1, 1), rep(0, p-10))
  y_org = X_org %*% true_b + rnorm(n)
```

For this question, you should pre-process (center and scale) the data such that $\sum_i X_{ij} = 0$ and $X_j^\mathrm{T} X_j = n$ for each j, and center (but not scale) the outcome, so that the intercept term is not needed. You should record the mean and sd of these transformations such that the parameter estimates on the original scale can be recovered (like HW2). Please be aware that the `scale()` function in R gives you $X_j^\mathrm{T} X_j = n-1$ instead of n. After you transform the data, proceed with one iteration of the coordinate descent algorithm that updates each parameter once. You should consider two things (both were mentioned during our lectures) that may improve the efficiency of your algorithm:

- After standardizing the X variables, how your updates can be simplified?
- When calculating the partial residuals (you should have this term in your formula), inherit the residuals from the previous update to save calculations.

Pick $\lambda = 0.4$. Use the initial value of $\boldsymbol{\beta}$ being all zeros. However, your code should work for any initial values. Report the first ten entries of your $\boldsymbol{\beta}$ after this one iteration of updates. You should use the soft-thresholding `soft_th` during the update.

Answer:

This kind of standarization can remove the denominator in the equation of estimated beta.

Scaling:

```r
n <- nrow(X_org)
p <- ncol(X_org)
X_mean <- apply(X_org,2,mean)
X_sd <- apply(X_org,2,sd)*sqrt((n-1)/n)
ones <- matrix(1,200,1)
X_stan <- (X_org-ones%*%X_mean)/(ones%*%X_sd)
y_stan <- scale(y_org,center=T,scale=F)
```

Coordinate Descent:

```r
lambda <- 0.4
beta <- matrix(0,p,1)
for (j in 1:p){
  if (j == 1){
    r = y_stan - X_stan[,-1]%*%beta[-1]
  }
  else{
    r = r - X_stan[,j-1]*beta[j-1] + X_stan[,j]*beta[j]
  }
  b = 1/n*t(X_stan[,j])%*%r
  beta[j] = soft_th(b,lambda)
}
beta[1:10]
```

```
##  [1] 0.18330176 0.08744669 0.17079989 0.24813098 0.00000000 0.17953579 0.00000000
##  [8] 0.10104141 0.00000000 0.19245049
```

c. [**15 points**]

Now, let's finish this coordinate descent algorithm. Write a function `myLasso(X, y, lambda, tol, maxitr)`. Set the tolerance level `tol` = 1e-5, and `maxitr` = 100 as the default values. Use the "one loop" code that you just wrote part c), and integrate that into a grand for-loop code that will repeatedly updating the parameters up to `maxitr` runs. Check your parameter updates after each loop and stop the algorithm once the ℓ_1 distance between the parameter values before and after each iteration is less than the tolerance level. Again, use $\lambda = 0.4$, and initial value zero. You function should output the estimated parameter values.

Answer:

```r
oldbeta <- matrix(0,p,1)
newbeta <- matrix(0,p,1)
myLasso <- function(X,y,lambda,tol,maxitr){
  for (i in 1:maxitr){
    for (j in 1:p){
          if (j == 1){
            r = y - X[,-1]%*%newbeta[-1]
            }
      else{
        r = r - X[,j-1]*newbeta[j-1] + X[,j]*newbeta[j]
        }
        b = 1/n*t(X[,j])%*%r
        newbeta[j] = soft_th(b,lambda)
    }
    diff = max(abs(oldbeta-newbeta))
    if (diff >= tol) {oldbeta = newbeta}
    else {break}
  }
  return(newbeta)
}
beta <- myLasso(X_stan,y_stan,0.4,1e-5,100)
```

d. [10 points] Recover the parameter estimates on the original scale of the data. Report the nonzero parameter estimates. Check your results with the `glmnet` package on the first 11 parameters (intercept and the ten nonzero ones).

```
* If your result matches the `glmnet` package, report the $\ell_1$ distance between your solution and t
* If your result does not match the `glmnet` package, discuss potential issues and fixes of your code
```

Answer:

```
beta_opt <- c(mean(y_org)-sum((X_mean/X_sd)*beta) , beta/X_sd)
beta_opt <- beta_opt[1:11]
library(glmnet)
lasso.fit <- glmnet(X_org,y_org,family="gaussian",alpha=1,lambda=0.4,
                    standardize=T,thresh=1e-5,maxit=100)
beta_glm <- coef(lasso.fit)[1:11]
diff = abs(beta_glm-beta_opt)
knitr::kable(as.data.frame(cbind(beta_opt,beta_glm,diff)))
```

beta_opt	beta_glm	diff
0.0553022	0.0553719	0.0000697
0.0714599	0.0711116	0.0003482
0.0000000	0.0000000	0.0000000
0.0930431	0.0926459	0.0003973
0.1739594	0.1737718	0.0001876
0.0000000	0.0000000	0.0000000
0.1640543	0.1638625	0.0001917
0.0000000	0.0000000	0.0000000
0.0706774	0.0703112	0.0003662
0.0000000	0.0000000	0.0000000
0.2291848	0.2291852	0.0000005

e. [**15 points**]

Let's modify our Lasso code to perform path-wise coordinate descent. The idea is simple: we will solve the solution on a grid of λ values, starting from the largest one. After obtaining the optimal $\boldsymbol{\beta}$ for a given λ, we simply use this solution as the initial value (instead of all zero) for the next (smaller) λ. This is referred to as a warm start in optimization problems. We will consider the following grid of λ:

```
  # lmax = max(t(X) %*% y) / n
  # where X is standardized and y is centered from the original data
  lmax = 0.765074054
  lambda_all = exp(seq(log(lmax), log(lmax*0.01), length.out = 100))
```

After finishing your algorithm, you should have a matrix that records all the fitted parameters on your λ grid. Provide a plot similar to page 41 (right panel) in the lecture note "Penalized" using the first 10 (nonzero) parameters. Use parameter values on the original scale of the data.

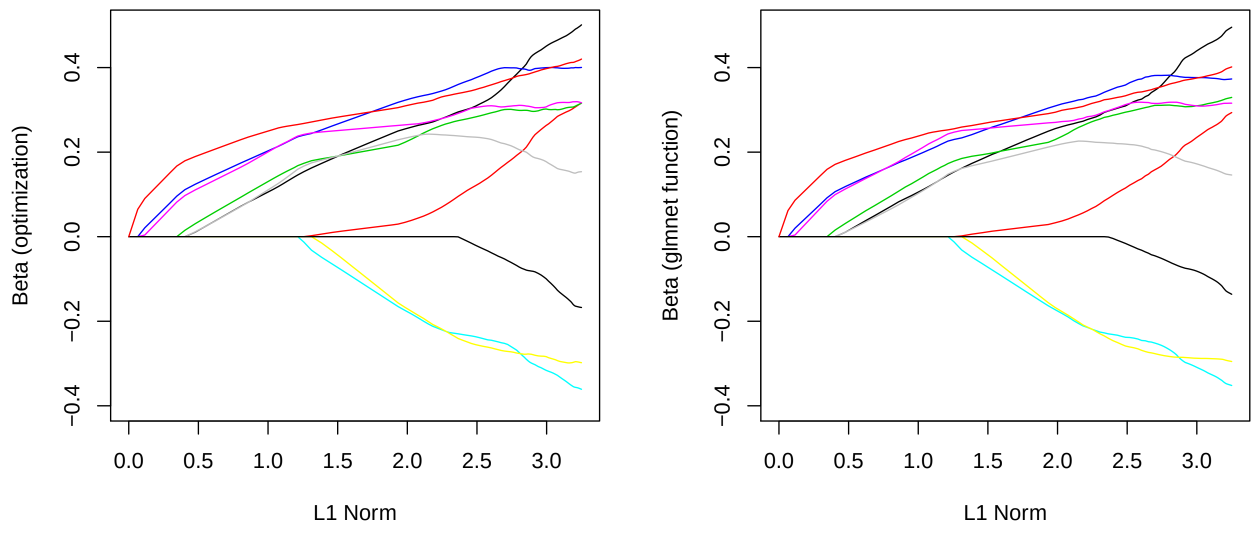
- Comment on the similarity between your solution and the `glmnet` solution.
- Does that vary across different λ values?
- Do you notice anything unusual about the plot? What could be the cause of such a phenomenon?

Answer:

```r
myLasso2 <- function(X,y,lamb,tol,maxitr){
  beta2 <- matrix(0,p,length(lamb))
  newbeta2 <- matrix(0,p,1)
  oldbeta2 <- matrix(0,p,1)
  for (k in 1:length(lamb)){
    lam = lamb[k]
    for (i in 1:maxitr){
      for (j in 1:p){
        if (j == 1){
          r2 = y - X[,-1]%*%newbeta2[-1]
          }
        else{
          r2 = r2 - X[,j-1]*newbeta2[j-1] + X[,j]*newbeta2[j]
        }
        b2 = 1/n*t(X[,j])%*%r2
        newbeta2[j] = soft_th(b2,lam)
      }
      diff2 = max(abs(oldbeta2-newbeta2))
      if (diff2 >= tol) {oldbeta2 = newbeta2}
      else {break}
    }
    beta2[,k] = newbeta2
    oldbeta2 = newbeta2
  }
  return(beta2)
}
beta_all = myLasso2(X_stan,y_stan,lambda_all,1e-5,100)
```

```r
lasso.fit2 <- glmnet(X_org,y_org,family="gaussian",alpha=1,lambda=lambda_all,
                     standardize=T,thresh=1e-5)
X_std <- matrix(X_sd,length(X_sd),100)
l1norm <- apply(abs(beta_all[1:10,]/X_std[1:10,]),2,sum)
```

```r
par(mfrow = c(1, 2))
plot(l1norm,beta_all[1,]/X_std[1,],type="l",col=1,
     ylim=c(-.4,.5),xlab="L1 Norm",ylab="Beta (optimization)")
for (l in 2:10){
  lines(l1norm,beta_all[l,]/X_std[l,],type="l",col=l)
}
beta_glm <- coef(lasso.fit2)[-1,]
plot(l1norm,beta_glm[1,],type="l",col=1,ylim=c(-.4,.5),
     xlab="L1 Norm",ylab="Beta (glmnet function)")
for (m in 2:10){
  lines(l1norm,beta_glm[m,]/X_std[m,],type="l",col=m)
}
```

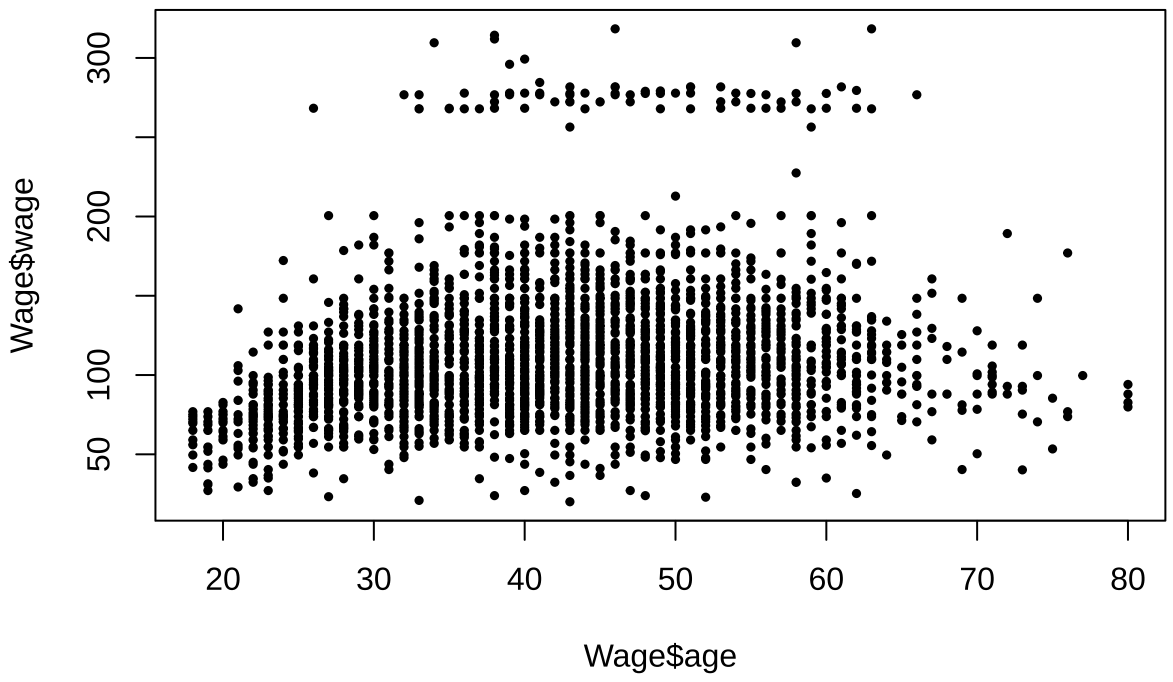


The entire shape of the two plots look similar; The points that each beta becomes non-zero seem to be same from the two plots; However, the graph of beta from glmnet looks a little bit smoother; There might be some situations that the direction of the coordinate is orthogonal to the direction of the gradient.

Question 2 [30 Points] Splines

We will fit and compare different spline models to the `Wage` dataset form the `ISLR` package.

```r
library(ISLR)
data(Wage)
plot(Wage$age, Wage$wage, pch = 19, cex = 0.5)
```



a. [20 points]

Let's consider several different spline methods to model `Wage` using `age`. For each method (except the smoothing spline), report the degrees of freedom. To test your model, use the first 300 observations of the data as the training data and the rest as testing data. Use the mean squared error as the metric. * Write your own code (you cannot use `bs()` or similar functions) to implement a continuous piecewise linear spline fitting. Pick 4 knots using your own judgment. * Write your own code to implement a quadratic spline fitting. Your spline should have a continuous first derivative. Pick 4 knots using your own judgment. * Use existing functions (`bs()` or similar) to implement a cubic spline with 4 knots. Use their default knots. * Use existing functions to implement a natural cubic spline with 6 knots. Choose your own knots.
* Use existing functions to implement a smoothing spline. Use the built-in generalized cross-validation method to select the best tuning parameter.

Answer:

```
library(splines)
wage_trn <- Wage[1:300,]
wage_tst <- Wage[-c(1:300),]
mse <- function(y,yhat){
  return(sum((y-yhat)^2)/length(y))
}
```

Method1:

The degree of freedom is 6 and the mse is:

```
myknots <- c(30,40,55,65)
pos <- function(x) x*(x>0)
mybasis1 <- function(d){
  return(cbind("int" = 1, "x_1" = d$age,
               "x_2" = pos(d$age - myknots[1]),
               "x_3" = pos(d$age- myknots[2]),
               "x_4" = pos(d$age - myknots[3]),
               "x_5" = pos(d$age - myknots[4])))
}
wage_trn1 <- mybasis1(wage_trn)
wage_tst1 <- mybasis1(wage_tst)
fit1 <- lm(wage_trn$wage~.-1, data = data.frame(wage_trn1))
mse(wage_tst$wage,predict(fit1,data.frame(wage_tst1)))
```

```
## [1] 1593.59
```

Method2:

The degree of freedom is 7 and the mse is:

```
mybasis2 <- function(d){
  return(cbind("int" = 1, "x_1" = d$age, "x_2" = d$age^2,
               "x_3" = pos(d$age - myknots[1])^2,
               "x_4" = pos(d$age- myknots[2])^2,
               "x_5" = pos(d$age - myknots[3])^2,
               "x_6" = pos(d$age - myknots[4])^2))
}
wage_trn2 <- mybasis2(wage_trn)
wage_tst2 <- mybasis2(wage_tst)
```

```r
fit2 <- lm(wage_trn$wage~.-1, data = data.frame(wage_trn2))
mse(wage_tst$wage,predict(fit2,data.frame(wage_tst2)))
```

```
## [1] 1596.128
```

Method3:

The degree of freedom is 8 and the mse is:

```r
fit3 <- lm(wage~bs(age,degree=3,df=8),data=wage_trn)
mse(wage_tst$wage,predict(fit3,wage_tst))
```

```
## [1] 1597.686
```

Method4:

The degree of freedom is 10 and the mse is:

```r
fit4 <- lm(wage~ns(age,knots=c(30,35,40,45,55,65)),data=wage_trn)
mse(wage_tst$wage,predict(fit4,wage_tst))
```

```
## [1] 1595.863
```

Method5:

The mse and the degree of freedom are:

```r
fit5 <- smooth.spline(wage_trn$age,wage_trn$wage,cv=F)
mse(wage_tst$wage,predict(fit5,as.vector(wage_tst$age))$y)
```

```
## [1] 1590.561
```

```r
fit5$df
```

```
## [1] 4.14233
```

b. [10 points]

After writing these models, evaluate their performances by repeatedly doing a train-test split of the data. Randomly sample 300 observations as the training data and the rest as testing data. Repeat this process 200 times. Use the mean squared error as the metric.

```
* Record and report the mean, median, and standard deviation of the errors for each method. Also, provi
* Based on these results, what method would you prefer?
```

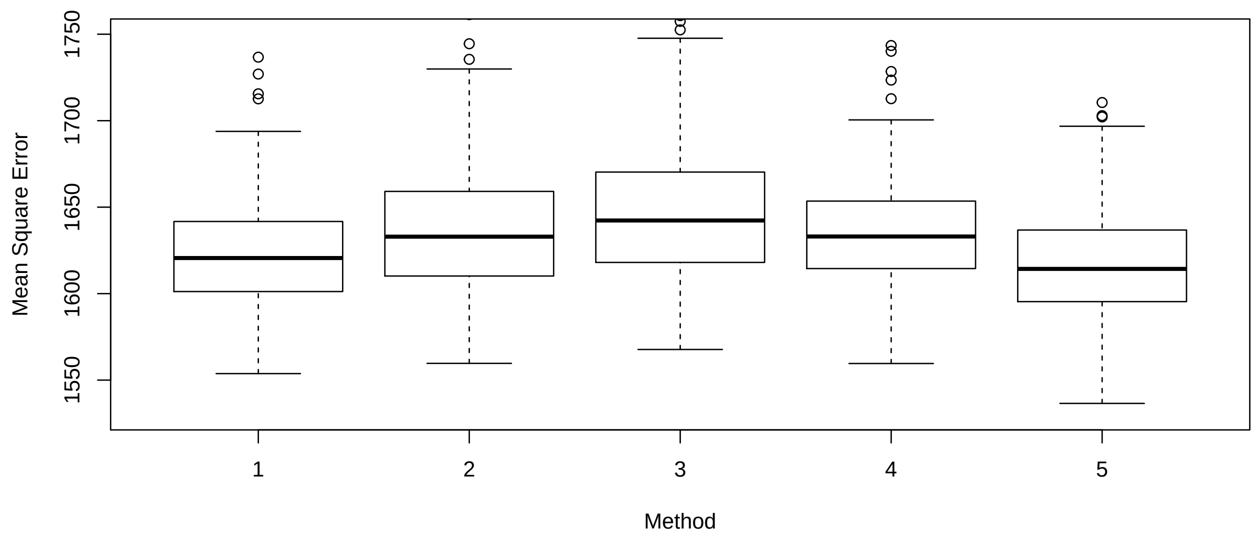
Answer:

```r
set.seed(10101)
mse_all <- matrix(0,200,5)
for (h in 1:200){
  trn_idx <- sample(1:3000,300)
  trn <- Wage[trn_idx,]
  tst <- Wage[-trn_idx,]
  trn1 <- mybasis1(trn)
  tst1 <- mybasis1(tst)
  trn2 <- mybasis2(trn)
  tst2 <- mybasis2(tst)
  sp1 = lm(trn$wage~.-1, data = data.frame(trn1))
  mse1 = mse(tst$wage,predict(sp1,data.frame(tst1)))
  sp2 = lm(trn$wage~.-1, data = data.frame(trn2))
  mse2 = mse(tst$wage,predict(sp2,data.frame(tst2)))
  sp3 = lm(wage~bs(age,degree=3,df=8),data=trn)
  mse3 = mse(tst$wage,predict(sp3,tst))
  sp4 = lm(wage~ns(age,knots=c(30,35,40,45,55,65)),data=trn)
  mse4 = mse(tst$wage,predict(sp4,tst))
  sp5 = smooth.spline(trn$age,trn$wage,cv=F)
  mse5 = mse(tst$wage,predict(sp5,as.vector(tst$age))$y)
  mse_all[h,] = c(mse1,mse2,mse3,mse4,mse5)
}
```

```r
mse_summary = as.data.frame(rbind(apply(mse_all,2,mean),apply(mse_all,2,median),apply(mse_all,2,sd)))
names(mse_summary)[1:5] = paste0(rep("Method",5),1:5)
rownames(mse_summary) = c("Mean","Median","SD")
knitr::kable(mse_summary)
```

	Method1	Method2	Method3	Method4	Method5
Mean	1623.28330	1653.59188	1654.93975	1635.51225	1620.12047
Median	1620.57876	1632.92672	1642.28712	1633.03597	1614.29627
SD	32.60698	96.93764	88.77342	32.47604	37.92482

```r
boxplot(mse_all,ylim=c(1530,1750),xlab="Method",ylab="Mean Square Error")
```



I would prefer smoothing spline. The mean and median of its mean square error are the lowest.