LASSO

Rishabh Vaish

Coding Your Own Lasso

I will write our own Lasso code. First, we will generate simulated data. Here, only X_1, X_2 and X_3 are important, and we will not consider the intercept term.

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

# generate data
V = matrix(0.2, p, p)
diag(V) = 1
X = as.matrix(mvrnorm(n, mu = rep(0, p), Sigma = V))
y = X[, 1] + 0.5 * X[, 2] + 0.25 * X[, 3] + rnorm(n)

# we will use a scaled version
X = scale(X)
y = scale(y)
```

As we already know, coordinate descent is an efficient approach for solving Lasso. The algorithm works by updating one parameter at a time, and loop around all parameters until convergence.

Hence, we need first to write a function that updates just one parameter, which is also known as the soft-thresholding function. Construct the function in the form of `soft_th <- function(b, lambda)`, where `b` is a number that represents the one-dimensional linear regression solution, and `lambda` is the penalty level. The function should output a scaler, which is the minimizer of

$$(x - b)^2 + \lambda|b|$$

```
#Soft threshold function
soft_th <- function(b, lambda) {
  # Making three cases of beta_ols estimate and applying the shrinkage derived in class
  if (b > lambda / 2) {
    return(b - lambda / 2)
  }
  else if (b <= lambda / 2 && b >= -lambda / 2) {
    return(0)
  }
  else if (b > -lambda / 2) {
    return(b + lambda / 2)
  }
}
```

Now lets pretend that at an iteration, the current parameter $\boldsymbol{\beta}$ value is given below (as `beta_old`, i.e., $\boldsymbol{\beta}^{\text{old}}$). Apply the above soft-thresholding function to update all p parameters sequencially one by one to complete one "loop" of the updating scheme. Please note that we use the Gauss-Seidel style coordinate descent, in which the update of the next parameter is based on the new values of previous entries. Hence, each time a parameter is updated, you should re-calculate the residual

$$\mathbf{r} = \mathbf{y} - \mathbf{X}^{\mathrm{T}}\boldsymbol{\beta}$$

so that the next parameter update reflects this change. After completing this one entire loop, print out the first 3 observations of $\mathbf{r}$ and the nonzero entries in the updated $\boldsymbol{\beta}^{\text{new}}$ vector. For this, use `lambda` $= 0.7$ and

```
beta_old = rep(0, p)
```

```
lambda = 0.7
#Running one loop of updation of beta
for (i in 1:ncol(X)) {
  #Calculating residual with latest beta values
  residual <- y - X[,-i] %*% beta_old[-i]
  #Calculating beta_ols by considering one beta at a time
  beta_old[i] <- (t(X[, i]) %*% (residual)) / t(X[, i]) %*% X[, i]
  #applying shrinkage
  beta_old[i] <- soft_th(beta_old[i], lambda)
}
#residual
head(residual, 3)
```

```
##             [,1]
## [1,] -0.07604338
## [2,]  0.14677403
## [3,]  0.15625677
```

```
#Non zero beta values and their corresponding index :
which(beta_old != 0)
```

```
## [1] 1 2
```

```
beta_old[which(beta_old != 0)]
```

```
## [1] 0.3529634 0.0902926
```

Now, let us finish the entire Lasso algorithm. We will write a function `myLasso(X, y, lambda, tol, maxitr)`. Set the tolerance level `tol` $=$ 1e-5, and `maxitr` $= 100$ as the default value. Use the "one loop" code that you just wrote in the previous section, and integrate that into a grand for-loop that will continue updating the parameters up to `maxitr` runs. Check your parameter updates once in this grand loop and stop the algorithm once the ℓ_1 distance between $\boldsymbol{\beta}^{\text{new}}$ and $\boldsymbol{\beta}^{\text{old}}$ is smaller than `tol`. Use `beta_old = rep(0, p)` as the initial value, and `lambda` $= 0.3$. After the algorithm converges, I report the following: i) the number of iterations took; ii) the nonzero entries in the final beta parameter estimate, and iii) the first three observations of the residual.

```r
#Final function
myLasso <- function(X, y, lambda, tol, maxitr) {
  #initialize beta
  beta_old <- rep(0, p)
  #loop for max iterations
  for (n_iter in 1:maxitr) {
    #store the beta before update
    beta_before = beta_old
    #update loop in Part B
    for (i in 1:ncol(X)) {
      residual <- y - X[,-i] %*% beta_old[-i]
      beta_old[i] <-
        (t(X[, i]) %*% (residual)) / t(X[, i]) %*% X[, i]
      beta_old[i] <- soft_th(beta_old[i], lambda)
    }
    #store the beta after update
    beta_after <-  beta_old
    #calculate distance between beta
    distance <- sum(abs(beta_after - beta_before))
    #end function if tolerance recahed
    if (distance < tol) {
      ans_list <-
        list("n_iter" = n_iter,
             "beta" = beta_old,
             "residual" = residual)
      return(ans_list)
    }
  }
  # return result after max iteration
  ans_list <-
    list("n_iter" = n_iter,
         "beta" = beta_old,
         "residual" = residual)
  return(ans_list)
}

tol = 10 ^ -5
maxitr = 100
lambda = 0.3

result <- myLasso(X, y, lambda, tol, maxitr)

# No of iterations:")
result$n_iter #Iterations
```

```
## [1] 9
```

```r
#"Non zero beta dn their corresponding index"
which(!result$beta == 0)
```

```
## [1]   1   2   3  14 118 137
```

```r
result$beta[which(!result$beta == 0)]
```

```
## [1] 0.457802236 0.226116017 0.114399954 0.001018992 0.011551407 0.004669249
```

```r
#Top 3 residual values:
head(result$residual, 3)
```

```
##             [,1]
## [1,] -0.1757378
## [2,]  0.2262848
## [3,]  0.1912103
```

Now we have our own Lasso function, let's check the result and compare it with the `glmnet` package. Note that for the glmnet package, their `lambda` should be set as half of ours.

```r
library(glmnet)
#fit a lasso with lambda/2
lasso_fit <- glmnet(X, y, lambda = 0.15, alpha = 1)
#find non-zero beta values
beta_lasso <- lasso_fit$beta[which(!lasso_fit$beta == 0)]
#Calculate distance between two beta
l1_dist <-
  sum (abs(beta_lasso - result$beta[which(!result$beta == 0)]))
l1_dist
```

```
## [1] 0.001095256
```

The distance is less than 0.005. Hence the algorithm is quite accurate.