Stat 542 HW2

Yizhou Zhang

Feb 21th, 2018

Problem 1a: Fixing all other parameters, write down the one-variable optimization problem of β_0 based on this objective function. Given the explicit form of the solution to this problem.

The one-variable optimization problem is:

$$\underset{\beta_0}{\operatorname{argmin}} f(\beta_0) = \frac{1}{2n}||y - \beta_0||_2^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Take derivative with respect to β_0 yields the first order condition $-\frac{1}{n}\sum_{n=1}^{n}(y_i - \beta_0 - X\beta) = 0$. Therefore the solution of F.O.C is :

$$\beta_0 = \frac{\sum_{n=1}^{n}(y_i - X\beta)}{n}$$

Problem 1b: Fixing all other parameters, write down the one-variable optimization problem of one parameter.

The basic setup of the Lasso problem is :

$$\underset{\beta_0}{\operatorname{argmin}} f(\beta) = ||y - X\beta||_2^2 + \lambda||\beta||_1$$

Assume orthogonal X matrix, or $X^T X = I_p$, then by the lecture notes the above optimization problem is equivalent to $||X\beta^{ols} \hat{} - X\beta||^2 + \lambda||\beta||_1$, which is further equivalent to

$$\hat{\beta}^{lasso} = \underset{\beta}{\operatorname{argmin}} \sum_{j=1}^{p} (\beta^{\hat{o}ls} - \beta_j)^2 + \lambda|\beta_j|$$

Since all other parameters are fixed,

$$\hat{\beta_j}^{lasso} = \underset{\beta_j}{\operatorname{argmin}} (\beta_j - \beta^{\hat{o}ls})^2 + \lambda|\beta_j|, \lambda > 0$$

if $\beta_j > 0$, then the above problem becomes

$$\underset{\beta_j}{\operatorname{argmin}} (\beta_j - \beta^{\hat{o}ls})^2 + \lambda\beta_j$$

Take derivative w.r.t β_j. The F.O.C is :

$$2(\beta - \hat{\beta}) + \lambda = 0$$

or

$$\beta_j = \hat{\beta} - \frac{\lambda}{2}, if \hat{\beta} > \frac{\lambda}{2}$$

if $\beta_j < 0$, the optimization problem becomes

$$\underset{\beta_j}{\operatorname{argmin}}(\beta_j - \hat{\beta^{ols}})^2 - \lambda\beta_j$$

Take derivative w.r.t β_j, the F.O.C reduces to

$$\beta_j = \hat{\beta} + \frac{\lambda}{2}, if \hat{\beta} < -\frac{\lambda}{2}$$

When $-\frac{\lambda}{2} < \hat{\beta} < \frac{\lambda}{2}$, I will show $\beta_j = 0$ is the best solution. Specifically, need to show $\beta_j(a) - \beta_j(0) > 0 for \forall a > 0$.

Note that

$$\beta_j(a) - \beta_j(0) = (a - \hat{\beta_j^{ols}})^2 + \lambda\beta - (\hat{\beta_j^{ols}})^2 = \beta_j^2 - \beta_j(2\hat{\beta_j^{ols}} + \lambda)$$

Given $-\frac{\lambda}{2} < \hat{\beta} < \frac{\lambda}{2}$, $\beta_j^2 - \beta_j(2\hat{\beta_j^{ols}} + \lambda) > 0$ will always hold.Following a similar pattern , it can be shown that $\beta_j(b) - \beta_j(0) > 0, \forall b < 0$. Therefore, $\beta_j = 0$ is the best solution when $-\lambda < 2\hat{\beta} < \lambda$.

To summarize,

$$\hat{\beta_j}^{lasso} = \begin{cases} \hat{\beta^{ols}} - \lambda/2, if \hat{\beta^{ols}} > \frac{\lambda}{2} \\ 0, if -\lambda < 2\hat{\beta^{ols}} < \lambda \\ \hat{\beta^{ols}} + \lambda/2, if \hat{\beta^{ols}} < -\frac{\lambda}{2} \end{cases}$$

Question 1c: Fixing all non-intercept parameters to be zero. Use part a) to update the intercept. After this update, find the smallest lambda such that none of the coefficients can be updated out of zero in the next iteration.

The one-variable optimization problem is:

$$\underset{\beta_0}{\operatorname{argmin}} f(\beta_0) = \frac{1}{2n}||y - \beta_0||_2^2$$

Take derivative with respect to β_0 yields the first order condition $-\frac{1}{n}\sum_{n=1}^{n}(y_i - \beta_0) = 0$. The solution of F.O.C is

$$\beta_0 = \frac{\sum_{n=1}^{n} y_i}{n}$$

By the soft-thresholding solution derived in part b),

$$\hat{\beta_j}^{lasso} = 0, if|\hat{\beta_j^{ols}}| \leq \frac{\lambda}{2}$$

. Therefore, the minimum λ value, denoted as λ_{max}, is equal to

$$\lambda_{max} \geq 2 * |\hat{\beta_{max}^{ols}}|$$

where

$$|\hat{\beta_{max}^{ols}}| = \max abs(\hat{\beta_j^{ols}}), j = 1, 2...p$$

Problem 1d: Implement the above procedures to get a path-wise coordinate descent solution of the lasso problem.Generate an independent set of 1000 observations under the same model as the test dataset.Report details of your tuning procedure and the results.

```
#search over 100 lambdas, and define the function
lambda_max = 2 *max(abs(coef(ols)))
lambda = exp(seq(log(lambda_max), log(0.01), length.out = 100))
LassoFit = function(X, y, lambda, tol = 1e-5, maxitr = 100){

    # initiate objects to record the values
    mybeta = matrix(NA, ncol(X), length(lambda))
    mybeta0 = rep(NA, length(lambda))
    mylambda = rep(NA, length(lambda))

    # initiate values

    current_beta = matrix(0, P, 1)

    current_beta0 = 0

for (l in 1:length(lambda)){
        # reduce the current lambda value to a smaller one
        current_lambda = lambda[l]

for (k in 1:maxitr){
  # update the intercept term based on the current beta values.

  old = current_beta
  current_beta0 = mean( y - X %*% current_beta)

  # compute residuals (this is with all variables presented)
  r = y - current_beta0 - X %*% current_beta

  # start to update each beta_j
  for (j in 1:ncol(X)){
    # remove the effect of variable j from model, and compute the residual
    r = r + X[,j]*current_beta[j]
    # update beta_j using the results in part b)
    xr = sum(X[,j]*r)
    xx = sum(X[,j]^2)
    current_beta[j] = abs(xr/xx)-current_lambda/2
    current_beta[j] = sign(xr)*ifelse(current_beta[j]>0,current_beta[j],0)

    # add the effect of variable j back to the model, and compute the residual
    r = r - X[,j]*current_beta[j] }

  # check if beta changed more than the tolerance level in this iteration
  #(use tol as the threshold)
  # if not, break out of this loop k
  # you will need to record the beta values at the start of this
  #iteration for comparison.
```

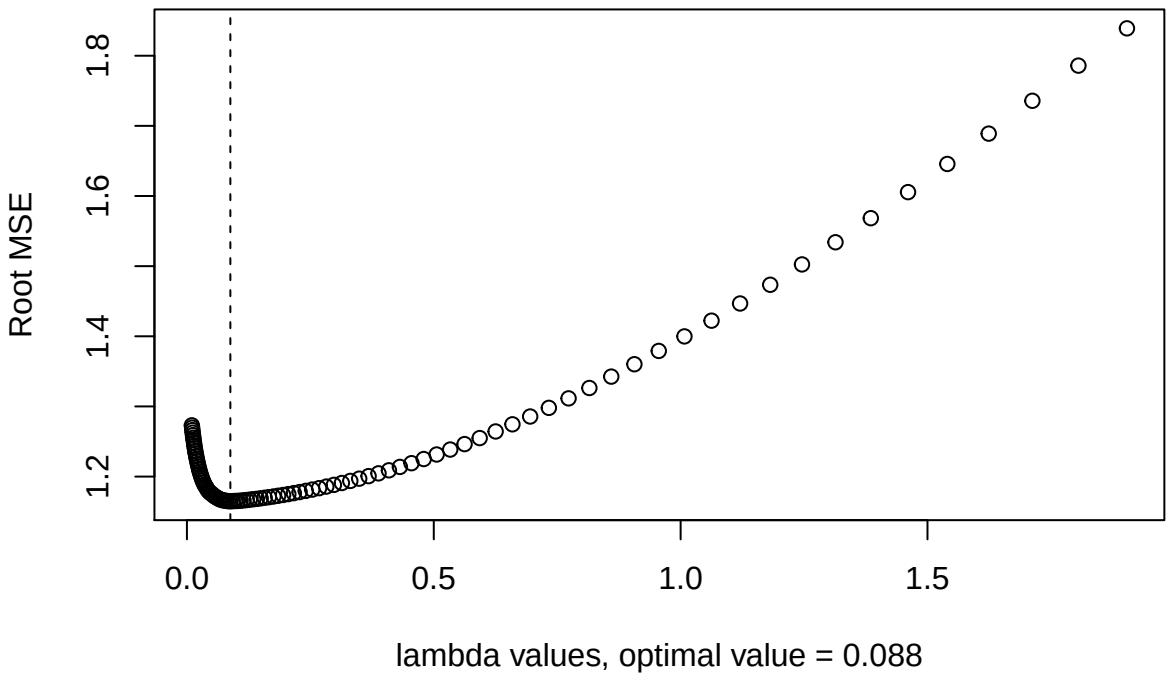
```
  if (all(current_beta - old < tol))  break;}
        # record the beta_j and beta_0 values
        mybeta[, l] = current_beta
        mybeta0[l] = current_beta0
    }
    return(list("beta" = mybeta, "b0" = mybeta0, "lambda" = lambda))}
a =  LassoFit(X,y, lambda)
```

For the tuning process, first generate a new set of 1000 test data under the same model.

```
N = 1000
P = 200

Beta = c(seq(1, 0, length.out = 21), rep(0, P-21))
Beta0 = 0.5

# you must set this seed for generating the data
set.seed(1)
# generate X_test
V = matrix(0.5, P, P)
diag(V) = 1
X_test = as.matrix(mvrnorm(N, mu = rep(0, P), Sigma = V))
# generate Y_test
y_test = Beta0 + X_test %*% Beta + rnorm(N)
```

The next step is tuning over the lambda values to find the one value that gives the smallest Root mean-square-error (rmse) on the test set. An ideal tuning curve should exhibit an U shape with the optimal lambda value at the bottom.

```
tst_err = rep(0,length(lambda))

for (i in 1:100){
  tst_err[i] =sqrt(mean((y_test - X_test %*% a$beta[,i])^2))
}
plot(x = lambda,y = tst_err, xlab = "lambda values, optimal value = 0.088",
     ylab = "Root MSE")
abline(v = 0.088, lty =2)
```



The figure above displays an U-shape curve, and the best lambda value is 0.088

Problem 2a Run the glmnet package and a 10 fold cross-validation to select the best lambda. Record the number of nonzero parameters using this lambda. For the rest of this question,fix this lambda value.

```
lso_cv = cv.glmnet(X, y, family = "gaussian", alpha = 1)
lfix_lso =lso_cv$lambda.min
#extract nonzero parameters
nonzero = which(coef(lso_cv) != 0)
names = coef(lso_cv)
```

The number of nonzero parameters is 52

Question 2b:Calculate the degrees of freedom.

```
N = 500
P = 200

Beta = c(seq(1, 0, length.out = 21), rep(0, P-21))
Beta0 = 0.5

# you must set this seed for generating the data
```

```r
set.seed(1)

# generate X
V = matrix(0.5, P, P)
diag(V) = 1
X = as.matrix(mvrnorm(N, mu = rep(0, P), Sigma = V))

deg_lso = rep(0,20)

for (i in 1:20){

  y = Beta0 + X %*% Beta + rnorm(N)
  lso_cv = glmnet(X, y, lambda = rep(lfix_lso,1), alpha = 1)
  fitted = predict(lso_cv, X)[,1]
  deg_lso[i] =  sum(diag(cov(y, fitted)))
}

deg_lso
```

```
##  [1] 58.92912 59.16701 58.32830 59.27116 59.50736 59.82628 58.95468
##  [8] 57.75069 58.68214 58.99254 59.09794 60.45476 59.39435 58.47452
## [15] 58.20727 58.54288 59.89937 58.78993 57.90003 60.36408
```

Repeating the DF calculation 20 times yields the above vector. The value range is (57.75, 60.45), and it is very close to the number of nonzero parameters in a). Note that no σ term is needed because the error variance is known as one.

Question 2c: Repeat part a) and b) for the ridge regression. Use the generalized cross validation criteria to select the best lambda. What is the theoretical value of the degrees of freedom for this lambda value? Does your estimation matches (or close to) the theoretical value?

The process is similar to b). First fit a ridge regression model `rdg_cv = cv.glmnet(X, y, family = "gaussian", alpha = 0)`, and extract the optimal λ value `rdg_cv$lambda.min`. Then generate 20 sets of `y` and fit a `glmnet` model with fixed lambda values `glmnet(X, y, lambda = rep(lfix_rdg,1), alpha = 0)`. Thirdly, calculate the degrees of freedom for each of the 20 iterations by the formula `sum(diag(cov(y, fitted)))`, which is the trace of the covariance matrix between $\hat{y}$ and y. No sigma term here because the error variance is known as one. The simulated DF of ridge regression is between (58.08, 60.53).

```r
pca = prcomp(X)
eigv = pca$sdev^2
eff_dg = sum(eigv^2/(eigv^2 + lfix_rdg))
eff_dg
```

```
## [1] 56.02834
```

The theoretical degrees of freedom of ridge regression is calculated above. First I fit `X` to principal component analysis. Then I extract the eigenvalues through `pca$sdev^2`. Lastly I calculate the degrees of freedom through `sum(eigv^2/(eigv^2 + lfix_rdg))`. There is no sigma terms here because the s.d of error is one as defined.

The theoretical effective DF is 56.03. This is slightly lower than the estimated DF above.