STATS 790 - Statistical Learning
Assignment 2

Tommy Flynn

13 February, 2023

Contents

Question 1: 2
Question 2: 8
Question 3: (ELS 3.6) 10
Question 4: (ELS 3.19) 11
Question 5: (ELS 3.28) 12
Question 6: (ELS 3.30) 13

Question 1:

a) Below we compute the full rank linear regression coefficients using 4 different methods.

(i) Naive Linear Algebra: Given full rank $\mathbf{X}_{n\times p}$ and $\vec{Y}_{n\times 1}$, the naive linear algebra approach finds the coefficients $\hat{\vec{\beta}}$ by minimizing the sum of squared residuals $L = (\vec{Y} - \mathbf{X}\vec{\beta})'(\vec{Y} - \mathbf{X}\vec{\beta})$ as follows: $\frac{\partial L}{\partial \vec{\beta}} = 0 \iff -2\mathbf{X}'\vec{Y} + 2\mathbf{X}'\mathbf{X}\vec{\beta} = 0 \iff \hat{\vec{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\vec{Y}$.

(ii) QR Decomposition: Decompose the design matrix into $\mathbf{X} = \mathbf{QR}$ where $\mathbf{Q}_{n\times p}$ is orthogonal and $\mathbf{R}_{p\times p}$ is upper triangular. This allows us to calculate the coefficients as follows: $\hat{\vec{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\vec{Y} = [(\mathbf{QR})'(\mathbf{QR})]^{-1}(\mathbf{QR})'\vec{Y} = [\mathbf{R}'\mathbf{Q}'\mathbf{QR}]^{-1}\mathbf{R}'\mathbf{Q}'\vec{Y} = \mathbf{R}^{-1}(\mathbf{R}')^{-1}\mathbf{R}'\mathbf{Q}'\vec{Y} = \mathbf{R}^{-1}\mathbf{Q}'\vec{Y}$. Note: this involves solving the upper triangular system using back subsitution.

(iii) SVD: Decompose the design matrix into $\mathbf{X} = \mathbf{UDV}'$ where $\mathbf{U}_{n\times p}, \mathbf{V}_{p\times p}$ are orthogonal and $\mathbf{D}_{p\times p}$ is diagonal. Then the coefficients are: $\hat{\vec{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\vec{Y} = [(\mathbf{UDV}')'(\mathbf{UDV}')]^{-1}(\mathbf{UDV}')'\vec{Y} = [\mathbf{VDU}'\mathbf{UDV}']^{-1}(\mathbf{VDU}')\vec{Y} = (\mathbf{V}')^{-1}\mathbf{D}^{-1}\mathbf{D}^{-1}\mathbf{V}^{-1}\mathbf{VDU}'\vec{Y} = \mathbf{VD}^{-1}\mathbf{U}'\vec{Y}$. Note: the inverse of the diagonal matrix is easily calculated using reciprocals.

(iv) Cholesky Decomposition: Decompose the design matrix and its transpose into $(\mathbf{X}'\mathbf{X}) = \mathbf{LL}'$ where $\mathbf{L}_{p\times p}$ is lower triangular. Then the coefficients are: $\hat{\vec{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\vec{Y} = (\mathbf{LL}')^{-1}\mathbf{X}'\vec{Y} = (\mathbf{L}')^{-1}\mathbf{L}^{-1}\mathbf{X}'\vec{Y}$. Note: this involves solving the lower and upper triangular systems using forward and back substitution.

b) Below is my code to benchmark the first 3 algorithms and lm from base R.

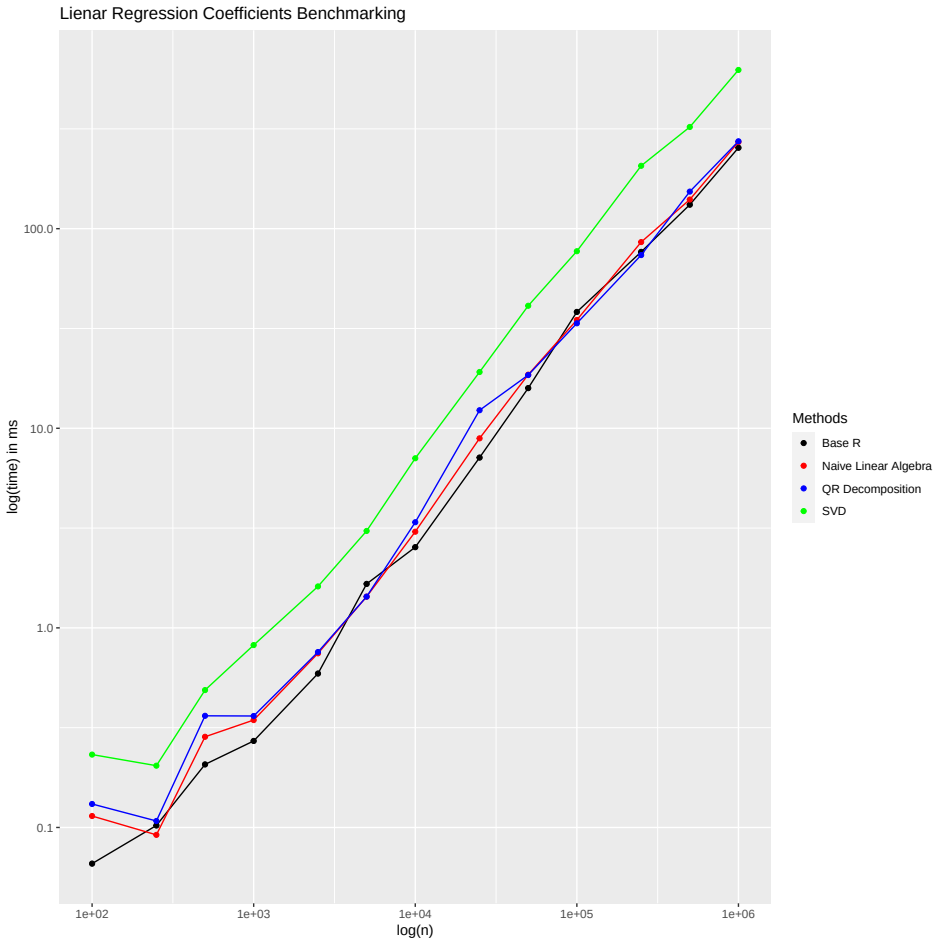
```
library(microbenchmark)
library(ggplot2)
library(tictoc)
set.seed(13)


# coefficients via naive linear algebra
```

```r
naive.lm <- function(X, Y){
  beta <- solve(t(X) %*% X) %*% t(X) %*% Y
  return(coefficients=as.vector(beta))
}

# coefficients via QR decomposition
qr.lm <- function(X, Y){
  # X = QR
  decomp <- qr(X)

  # solve R*beta = Q'*Y
  beta <- solve.qr(decomp, Y)
  return(coefficients=as.vector(beta))
}

# coefficients via SVD
svd.lm <- function(X, Y){
  # X = UDV'
  decomp <- svd(X)
  U <- decomp$u
  D.inv <- diag(1/decomp$d)
  V <- decomp$v

  beta <- V %*% D.inv %*% t(U) %*% Y
  return(coefficients=as.vector(beta))
}

# generates (n x p) design matrix and (1 x n) response vector
generate.data <- function(n, p){
  Y <- rnorm(n)
  X <- matrix(rnorm(p*n), ncol = p)
```

```
  list(X = X, Y = Y)
}

# set-up variables
sizes <- c(100, 250, 500, 1000, 2500, 5000, 10^4, 10^5/4,
           10^5/2, 10^5, 10^6/4, 10^6/2, 10^6)
n <- length(sizes)
init <- rep(0, n)
lm.times <- init
naive.times <- init
qr.times <- init
svd.times <- init

# main loop
tic()
for (i in 1:n){
  # benchmark
  D <- generate.data(sizes[i], p = 10)
  times <- microbenchmark(
    as.vector(lm.fit(D$X, D$Y)$coefficients),
    naive.lm(D$X, D$Y),
    qr.lm(D$X, D$Y),
    svd.lm(D$X, D$Y),
    times= 50,
    check='equal',
    unit='ms')
  s <- summary(times)$mean

  # append
  lm.times[i] <- s[1]
  naive.times[i] <- s[2]
```

```
  qr.times[i] <- s[3]
  svd.times[i] <- s[4]
}

# plotting variables
times.df <- list(data.frame(x=sizes, y=lm.times),
                 data.frame(x=sizes, y=naive.times),
                 data.frame(x=sizes, y=qr.times),
                 data.frame(x=sizes, y=svd.times))
c <- c('black', 'red', 'blue', 'green')

plot.times <- function(data, c){
  # set-up ggplot
  pl <- ggplot() +
    geom_point(data = times.df[[1]], aes(x, y, color = 'black')) +
    scale_x_log10() +
    scale_y_log10() +
    ggtitle('Lienar Regression Coefficients Benchmarking') +
    xlab('log(n)') +
    ylab('log(time) in ms') +
    scale_color_manual(name='Methods',
                       values=c('Base R'=c[1],
                                'Naive Linear Algebra'=c[2],
                                'QR Decomposition'=c[3],
                                'SVD'=c[4]))

  # plot lines and points
  for (i in 1:length(data)){
    pl <- pl +
      geom_line(data = data[[i]], aes(x, y, color = c[i]), color=c[i]) +
      geom_point(data = data[[i]], aes(x, y, color = c[i]), color=c[i])
```

```
  }
  return(pl)
}

plot.times(times.df, c)

# log-log models
model.lm <- lm(log(y) ~ log(x), data=times.df[[1]])
model.naive <- lm(log(y) ~ log(x), data=times.df[[2]])
model.qr <- lm(log(y) ~ log(x), data=times.df[[3]])
model.svd <- lm(log(y) ~ log(x), data=times.df[[4]])

model.lm
model.naive
model.qr
model.svd
toc()  # 158.244 seconds
```



We can see from the plot that for fixed p the time increases slightly less than linearly (slope 1) for n on the log-log scale. SVD consistently under-performs and base R is generally the fastest for this range; occasionally, naive linear algebra and QR decomposition will outperform it. The log-log models are as follows:

$y_{\text{base}} = -7.5241 + 0.9449x$

$y_{\text{naive}} = -7.1401 + 0.9183x$

$y_{\text{qr}} = -6.8913 + 0.8993x$

$y_{\text{svd}} = -6.4979 + 0.9336x$

c) ESL page 93 claims that the computational complexity of various algorithms for linear regression for fixed p are $\mathcal{O}(n)$. However, the models show slopes near 0.9 indicating that our simulation is performing faster than theory.

Question 2:

To implement ridge regression via data augmentation, we append the design matrix with $\sqrt{\lambda}\mathbf{I}_p$ and the response vector with $\vec{0}_p$ and then calculate the OLS estimate. Below is the R code to perform augmented ridge and native ridge (glmnet) on the ESL prostate data.

```
library(glmnet)
library(microbenchmark)
set.seed(13)

# data augmentation
augment <- function(X, Y, lambda){
  p <- ncol(X)
  X.aug <- rbind(X, sqrt(lambda)*diag(p))
  Y.aug <- c(Y, rep(0, p))
  return(list(X=X.aug, Y=Y.aug))
}

# augmented fit
aug.fit <- function(X, Y, lambda){
  D = augment(scale(X, center=TRUE, scale=FALSE), Y, lambda)
  return(lm.fit(D$X, D$Y)$coefficients)
}

# load data
prostate <- read.delim('prostate.txt')
X <- as.matrix(prostate[,2:9])
Y <-  prostate$lpsa

# augmented vs native fits
lambda1 <- 0.1
lambda2 <- 0.1/(4*nrow(X)) # looks like they used different formula
aug <- aug.fit(X,Y, lambda1)
```

```
native <- glmnet(X, Y, family='gaussian', alpha=0, lambda=lambda2)


as.vector(aug)          # intercept = 0.181560845
```

```
## [1]  0.564345074  0.617932167 -0.021155263  0.096857787  0.753238561
## [6] -0.104072523  0.048108998  0.004473005
```

```
as.vector(native$beta) # intercept = 0.178745157
```

```
## [1]  0.563995964  0.622175486 -0.021234188  0.096637107  0.761317212
## [6] -0.105791392  0.049544550  0.004449972
```

```
# test run-times on large matrix
generate.data <- function(n, p){
  Y <- rnorm(n)
  X <- matrix(rnorm(p*n), ncol = p)
  list(X = X, Y = Y)
}


D5 <- generate.data(10^5, 10)
m <- microbenchmark(aug.fit(D5$X,D5$Y, lambda1),
                glmnet(D5$X, D5$Y, family='gaussian', alpha=0, lambda=lambda2),
                times= 100,
                unit='ms')
summary(m)$mean
```

```
## [1] 43.02440 16.61833
```

We can see that the coefficients match very closely:

$$\vec{\beta}'_{\text{Aug}} = (0.1816, 0.5643, 0.6179, -0.0212, 0.0969, 0.7532, -0.1041, 0.0481, 0.0045)$$

$$\vec{\beta}'_{\text{ridge}} = (0.1787, 0.5640, 0.6222, -0.0212, 0.0966, 0.7613, -0.1058, 0.0495, 0.0044)$$

However, the native approach was over twice as fast on a random $10^5 \times 10$ matrix.

Question 3: (ELS 3.6)

We are given Gaussian prior $\vec{\beta} \sim N(\vec{0}, \tau\mathbf{I})$ and Gaussian sampling model $\vec{Y} \sim N(\mathbf{X}\vec{\beta}, \sigma^2\mathbf{I})$. Bayes Theorem asserts that $P(\vec{\beta}|\vec{Y}) = \frac{P(\vec{Y}|\vec{\beta})P(\vec{\beta})}{P(\vec{Y})}$. Following ESL page 64, we take the negative log-posterior as follows:

$$-\log(P(\vec{\beta}|\vec{Y})) = \frac{1}{2\sigma^2}(\vec{Y} - \mathbf{X}\vec{\beta})'(\vec{Y} - \mathbf{X}\vec{\beta}) + \frac{1}{2\tau}\vec{\beta}'\vec{\beta} + \frac{(2\pi)^{-n/2}}{\sigma} + \frac{(2\pi)^{-(p+1)/2}}{\sqrt{\tau}} - \log(P(\vec{Y}))$$

If we let $\lambda = \sigma^2/\tau$ then we have:

$$-\log(P(\vec{\beta}|\vec{Y})) = \frac{1}{2\sigma^2}\left[(\vec{Y} - \mathbf{X}\vec{\beta})'(\vec{Y} - \mathbf{X}\vec{\beta}) + \lambda\vec{\beta}'\vec{\beta}\right] + \frac{(2\pi)^{-n/2}}{\sigma} + \frac{(2\pi)^{-(p+1)/2}}{\sqrt{\tau}} - \log(P(\vec{Y}))$$

Minimizing this with respect to $\vec{\beta}$ results in:

$$\frac{\partial(-\log(P(\vec{\beta}|\vec{Y})))}{\partial\beta} = 0 \iff (\vec{Y} - \mathbf{X}\vec{\beta})'(\vec{Y} - \mathbf{X}\vec{\beta}) + \lambda\vec{\beta}'\vec{\beta} = 0$$

Notice that this is exactly the condition for the ridge regression coefficients. Moreover, since we are minimizing the negative-log posterior, we get the MAP estimate or the posterior mode. In the Gaussian setting, this is also the mean. Therefore, the ridge regression estimate is equal to the posterior mean and mode.

Question 4: (ELS 3.19)

Let the ridge regression estimate be given by $\hat{\vec{\beta}}^{\text{ridge}} = (\mathbf{X}'\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}'\vec{Y}$ and decompose the design matrix using SVD as $\mathbf{X}_{n\times p} = \mathbf{U}\mathbf{D}\mathbf{V}'$ where $\mathbf{U}_{n\times p}, \mathbf{V}_{p\times p}$ are orthogonal and $\mathbf{D}_{p\times p}$ is diagonal. Then we have:

$\hat{\vec{\beta}}^{\text{ridge}} = [(\mathbf{U}\mathbf{D}\mathbf{V}')'(\mathbf{U}\mathbf{D}\mathbf{V}') + \lambda\mathbf{I}]^{-1}(\mathbf{U}\mathbf{D}\mathbf{V}')'\vec{Y}$

$= [\mathbf{V}\mathbf{D}\mathbf{U}'\mathbf{U}\mathbf{D}\mathbf{V}' + \lambda\mathbf{I}]^{-1}(\mathbf{V}\mathbf{D}\mathbf{U}')\vec{Y}$

$= [\mathbf{V}\mathbf{D}^2\mathbf{V}' + \lambda\mathbf{I}]^{-1}(\mathbf{V}\mathbf{D}\mathbf{U}')\vec{Y}$

$= \mathbf{V}(\mathbf{D}^2 + \lambda\mathbf{I})^{-1}\mathbf{V}'\mathbf{V}\mathbf{D}\mathbf{U}'\vec{Y}$

$= \mathbf{V}(\mathbf{D}^2 + \lambda\mathbf{I})^{-1}\mathbf{D}\mathbf{U}'\vec{Y}.$

This allows us to calculate:

$||\hat{\vec{\beta}}^{\text{ridge}}|| = [\mathbf{V}(\mathbf{D}^2 + \lambda\mathbf{I})^{-1}\mathbf{D}\mathbf{U}'\vec{Y}]'[\mathbf{V}(\mathbf{D}^2 + \lambda\mathbf{I})^{-1}\mathbf{D}\mathbf{U}'\vec{Y}]$

$= (\mathbf{U}'\vec{Y})'[\mathbf{D}(\mathbf{D}^2 + \lambda\mathbf{I})^{-2}\mathbf{D}](\mathbf{U}'\vec{Y})$

Thus $||\hat{\vec{\beta}}^{\text{ridge}}|| = \sum_{i=1}^{p}(\vec{u_i}'\vec{Y})'\frac{d_i^2}{(d_i^2+\lambda)^2}(\vec{u_i}'\vec{Y})$ where $\vec{u_i}$ is the ith column of $\mathbf{U}$ and d_i is the ith diagonal entry of $\mathbf{D}$. Now it is clear that if $\lambda \to 0$ then $||\hat{\vec{\beta}}^{\text{ridge}}||$ increases since the middle term grows. It is also the case that $||\hat{\vec{\beta}}^{\text{lasso}}||$ increases as $\lambda \to 0$. To see this, consider the context of the size constraint t. We have $\lambda \to 0 \implies ||\hat{\vec{\beta}}^{\text{ridge}}||$ increases $\implies t$ must increase for $\sum_{i=1}^{p}\beta_i^2 \leq t^2$ to hold. As ESL page 63 explains, there is a one-to-one correspondence between λ and t and it is also the case that in the space spanned by $\vec{\beta}$, the lasso constraint region $\sum_{i=1}^{p}|\beta_i| \leq t$ is always contained in the ridge constraint region $\sum_{i=1}^{p}\beta_i^2 \leq t^2$. Therefore, $\lambda \to 0 \implies t$ increases $\implies ||\hat{\vec{\beta}}^{\text{lasso}}||$ increases.

Question 5: (ELS 3.28)

Fix constraint t and let the fitted lasso coefficient for X_j be $\hat{\beta}_j = a$. If we augment our data with an identical copy $X_j^* = X_j$ then our model would take the following form:

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_j x_{ij} + \beta_j^* x_{ij}^* + \cdots + \beta_p x_{ip} + \epsilon_i, \;\; i = 1, \ldots, n$$

$$= \beta_0 + \beta_1 x_{i1} + \cdots + (\beta_j + \beta_j^*) x_{ij} + \cdots + \beta_p x_{ip} + \epsilon_i$$

$$= \beta_0 + \beta_1 x_{i1} + \cdots + \beta_{\text{Aug}} x_{ij} + \cdots + \beta_p x_{ip} + \epsilon_i, \;\; \beta_{\text{Aug}} = \beta_j + \beta_j^*$$

Thus, under constraint t the lasso coefficients with respect to the augmented data are identical with the exception $\hat{\beta}_{\text{Aug}} = a \implies (\hat{\beta}_j + \hat{\beta}_j^{\,*}) = a$. Therefore, there is no significant effect of this exact collinearity since both X_j parameters are thralled to the optimal value a while the others remain unchanged.

Question 6: (ELS 3.30)

The elastic-net optimization problem is given by:

$$\min_{\vec{\beta}} ||\vec{Y} - \mathbf{X}\vec{\beta}||_2^2 + \lambda[\alpha||\vec{\beta}||_2^2 + (1-\alpha)||\vec{\beta}||_1]$$

Rearranging we have:

$$\min_{\vec{\beta}} ||\vec{Y} - \mathbf{X}\vec{\beta}||_2^2 + ||\sqrt{\lambda\alpha}\vec{\beta}||_2^2 + \lambda(1-\alpha)||\vec{\beta}||_1$$

$$= \min_{\vec{\beta}} \left|\left| \begin{pmatrix} \vec{Y} \\ \vec{0}_p \end{pmatrix} - \begin{pmatrix} \mathbf{X} \\ \sqrt{\lambda\alpha}\mathbf{I}_p \end{pmatrix} \vec{\beta} \right|\right|_2^2 + \lambda(1-\alpha)||\vec{\beta}||_1$$

Therefore, elastic-net can be considered as lasso with augmented data using the identity matrix and zero vector as well as a penalty parameter $\lambda(1-\alpha)$.