HW2 Data-driven statistical modelling

Fanny Bergström

2/8/2021

Excercise 3.2

For a response variable $y \in \{-1, +1\}$ and a linear classification function $f(x) = \beta_0 + \beta_T x$, suppose that we classify according to $\text{sign}(f(x))$. We will show that the signed Euclidean distance d of the point x with label y to the decision boundary is given by

$$d = \frac{1}{||\beta||_2} y f(x).$$

Since the classification is given by $\text{sign}(f(x))$, we know that the decision boundary is given by $f(x) = \beta_0 + \beta_T x = 0$. We also know that the shortest distance d from a point and a hyperplane is perpendicular to the plane. This means that in the direction of d, we have a unit vector given by

$$\beta^* = \frac{\beta}{||\beta||_2}.$$

We let x' denote the point on the decision boundary closest to x. Then this distance corresponds to the length of $x - x'$ in the direction of β^*. This distance equals

$$d = y\beta^{*T}(x - x').$$

Multiplying with y simply changes the sign depending on which side of the decision boundary x lies.

Since x' lies on the decision boundary $f(x') = \beta_0 + \beta_T x' = 0$, or equally $\beta_T x' = -\beta_0$. We can now simplifythe above expression as following

$$d = y\frac{\beta^T}{||\beta||_2}(x - x') = \frac{1}{||\beta||_2} y f(x),$$

which is what we wanted to show in this exercise.

Programming assignment

In this assignment we will do cross-validation for Lasso-penalised linear least squares using the Crime dataset (http://college.cengage.com/mathematics/brase/understandable_statistics/7e/students/datasets/mlr/frames/frame.html). Column 1 is the overall reported crime rate per 1 million residents; the response variable. Columns 3-7 are explanatory variables (covariates). We will not use column 2 at all.

```
library(tidyverse)
# Import data and remove col 2
data <- read_table2("CrimeData.txt",
                    col_names = FALSE) %>%
                select(-X2)
```

We centre the data to remove the constant β_0 from the model, and standardise the covariates (columns) to put them on equal scale.

```r
# Center data
data_standardised <- data %>%
                        scale(center = TRUE,
                        scale = TRUE)

X <- data_standardised[,-1]
y <- data_standardised[,1]
```

We write a function that solves the Lasso problem by cyclic coordinate descent (Section 2.4.2). The function uses a nested loop, where the inner loop iterates over covariates (one step updates a single β_j). The outer loop iterates over full cycles. Terminate the outer loop when the results β from two consecutive iterations of the other loop differ less than "small number" ϵ in 2-norm.

We start with writing a help function calculating the soft max threshold defined

$$S_\lambda(x) = \text{sign}(x)(|x| - \lambda)_+.$$

```r
# Soft threshold function
soft_threshold <- function(x, lamda){
    if(abs(x)-lamda > 0) {
        sign(x) * (abs(x)-lamda)
    }
    else { 0 }
}

# Cyclic coordinate gradient descent for lasso regression
coordinate_descent_lasso <- function(beta, X, y, lamda = 0.1, epsilon = 0.1){

    #Initialisation of values
    m <- dim(X)[1]
    n <- dim(X)[2]
    beta_norm <- t(beta) %*% beta
    beta_norm_new <- 0

    #Looping until convergence
    while(abs(beta_norm - beta_norm_new) > epsilon){
        beta_old <- beta

        #Looping through each coordinate
        for (j in 1:n){
            y_pred <- X %*% beta
            r <- t(X[,j]) %*% (y - y_pred  + beta[j] * X[,j])
            beta[j] <-  soft_threshold(r/m, lamda)
        }
    beta_norm <- t(beta_old) %*% beta_old
    beta_norm_new <- t(beta) %*% beta
    }

    return(beta)
}
```

Now, we will use this code to do a 5-fold cross-validation (15 mod 3 = 0) to find a suitable value for lambda. The candidate lambda's are $\lambda_{max} * (1e-4)^{[(m-k)/(m-1)]}$ for $i = 0, 1, ..., m$, where $m = 40$ and λ_{max} is the smallest lambda such that all β_j are 0 .

From Problem 2.1, we know that the smallest value of λ such that the regression coefficients estimated by the lasso are all equal to zero is given by

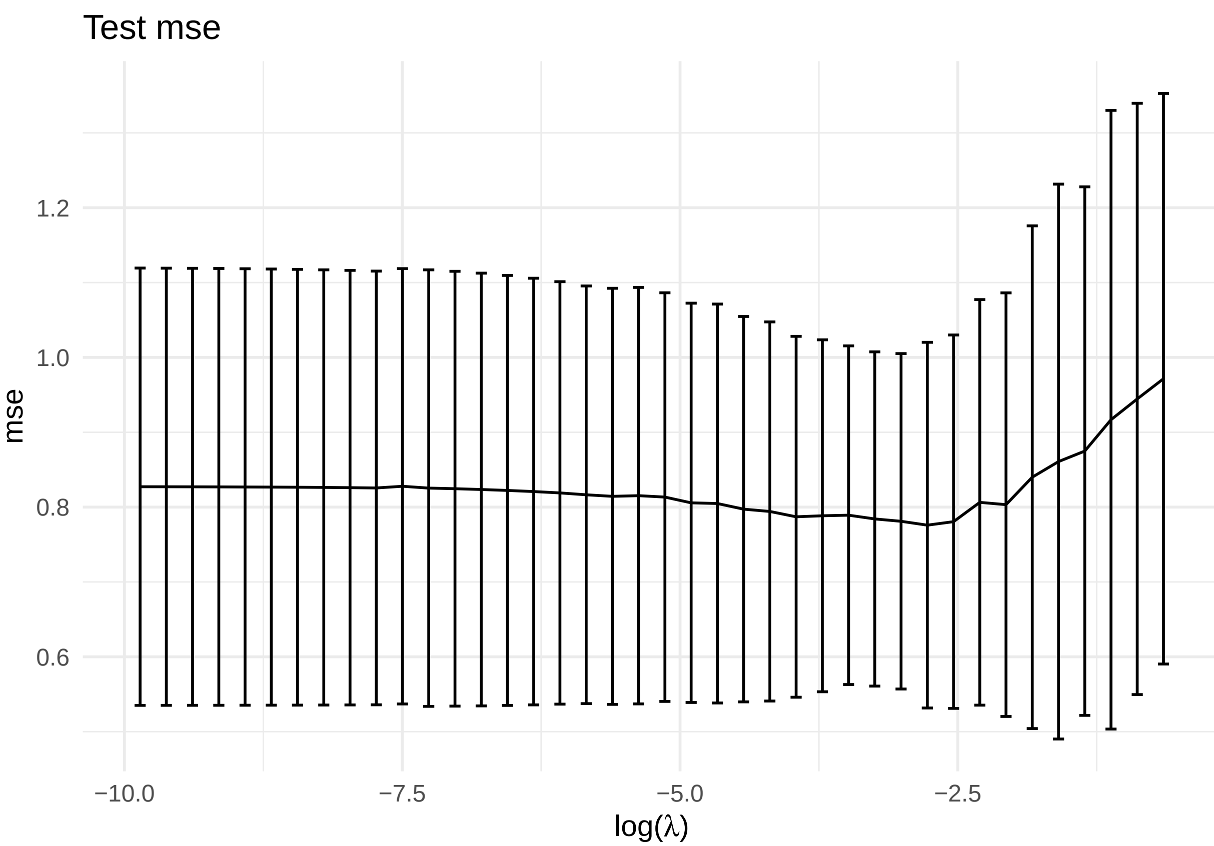
$$\lambda_{\max} = \max_j |\frac{1}{N}\langle \mathbf{x}_j, \mathbf{y}\rangle|.$$

```
# Finding lambda_max
lambda_max <- abs(1 / nrow(X) * (t(X) %*% y)) %>% max()

# Vector of candidate lambdas
# lamdas <- 1:40/40 * lambda_max

# Vector of candidate lambdas (log)
lamdas <- c()
for (k in 1:40) {
  lamdas[k] <- lambda_max * (1e-4)^((40 - k) / (40 - 1))
}

# Initial values of beta
dt1 <- data_standardised %>%
  as.data.frame() %>%
  mutate(X1 = data$X1)

beta_int <- beta <- lm("X1~.", dt1)$coefficients[-1]

# Mean squared error function
mse <- function(actual, predicted) {
  mean((actual - predicted)^2)
}


### 5-fold Cross validation

# Set seed for reproducability
set.seed(123)
# Randomly shuffle data
data_shuffle <- data_standardised[sample(nrow(X)), ]
# Create 5 folds (equal size)
folds <- cut(seq(1, nrow(X)), breaks = 5, labels = FALSE)

mse_res <- matrix(NA, 5, length(lamdas))

for (i in 1:5) {
  # Segement data
  index <- which(folds == i, arr.ind = T)
  test <- data_shuffle[index, ]
  train <- data_shuffle[-index, ]

  X_test <- test[, -1]
  y_test <- test[, 1]
  X_train <- train[, -1]
  y_train <- train[, 1]

  for (l in 1:length(lamdas)) {
    beta_l <- coordinate_descent_lasso(beta_int,
```

```
                                        X_train,
                                        y_train,
                                        lamda = lamdas[l]
    )
    y_pred <- X_test %*% beta_l
    mse_res[i, l] <- mse(y_pred, y_test)
  }
}
```

Below is a plot of the cross-validation mean square prediction error curve with error bars of the standard deviation.

```
mse_mean <- colMeans(mse_res)
mse_sd <- apply(mse_res, 2, sd)

bind_cols("lamdas" = lamdas, "mean" = mse_mean, "sd" = mse_sd) %>%
  ggplot(aes(x = log(lamdas), y = mean)) +
  geom_errorbar(aes(ymin = mean - sd, ymax = mean + sd), width = .1) +
  geom_line() +
  labs(
    title = "Test mse",
    y = "mse",
    x = expression(paste("log(", lambda, ")"))
  ) +
  theme_minimal()
```



We can now find the λ that gives us the smallest mse

```r
# Finding the lambda that gives the smallest mse.
ind <- mse_mean %>% which.min()
lamdas[ind]
```

```
## [1] 0.06237887
```

and the regression coefficients when using the particular λ

```r
# Regression coefficients
coordinate_descent_lasso(beta_int, X, y, lamda = lamdas[ind])
```

```
##          X3          X4          X5          X6          X7
##  0.46503343 -0.13547261  0.03387636  0.00000000  0.00000000
```

The smallest mse is found when $\lambda \approx 0.062$ which gives us three non-zero $beta_j$'s. The selection of the non-zero $beta_j$'s is consistent with the results in the SLS book.