Model Selection Final Project Q1

Paul Nguyen

2024-10-04

In this problem, we have p covariates, $X_1, \dots, X_p$, and an outcome Y. We have 2^p possible regression models, one for each subset S of covariates, and each has its own parameter $\beta^{(S)}$. We wish to perform inference on $(S, \beta^{(S)})$.

We specify a uniform prior on S, so that $p(S) = 2^{-p}$ for all subsets.

1. Design and implement an algorithm that returns samples of $(S, \beta^{(S)})$.

Let's first generate a Uniform random variable R from [0,1] to determine what type of move we would like.

Now, we describe the model moves. There are 4 possible types of moves.

The first possible move is a "stay" move. In this move, we let $(S, \beta) \to (S, \beta')$. We will perform a standard Metropolis-Hasting move here, and let $\beta' \sim N_{|S|}(0, I_{|S|})$. We perform this move if $R \in [0, .25)$.

The second possible move is a cross model move, called "add". Here, $(S, \beta) \to (S', \beta')$, where $|S'| = |S| + 1$. Let $U \sim Unif[0, 1]$. Then let $h(\beta_1, U) = (\beta_1, \beta_2)$. We perform this move if $R \in [.25, .5)$.

The third possible move is "delete". Let $(S, \beta) \to (S', \beta')$, where $|S'| = |S| - 1$. Let $h(\beta_1, \beta_2) = (\beta_1, U)$. We perform this move if $R \in [0.5, .75)$.

The last type of move is "swap". Again, $(S, \beta) \to (S', \beta')$, with $|S'| = |S|$. We let $h(\beta^S) = (\beta^{S'})$. We perform this move if $R \in [.75, 1]$.

```r
set.seed(4)
gen_data <- function(n, p){
  sigma <- 1
  num_coef <- ceiling(runif(1) * p)
  coef_index <- sample(1:p, size = num_coef)
  coef <- rnorm(num_coef)

  data_mat <- matrix(nrow = n,
                     ncol = p)
  for (i in 1:nrow(data_mat)) {
    for (j in 1:ncol(data_mat)) {
      data_mat[i,j] = runif(1, -1, 1)
    }
  }
  y <- vector(length = n)
  for (i in 1:length(y)) {
    y[i] <- sum(data_mat[i,coef_index] * coef) + rnorm(1, 0, sigma)
  }
  return(list(data_mat = data_mat,
              coef = coef,
              coef_index = coef_index,
              y = y))
}
```

```r
sample = function(x, size, replace = F, prob = NULL) {
  if (length(x) == 1) return(x)
  base::sample(x, size = size, replace = replace, prob = prob)
}

update_beta <- function(beta){
  S <- beta != 0
  u <- runif(1)
  p <- length(beta)
if (u < .25) { # stay at S
  # print("stay")
  beta = S * rnorm(p)
} else if ((u >= .25) & (u < .5)) { # add
  # print("add")
  if (sum(S) < p) { # if we havent already hit max predictors
    false_indices <- as.vector(which(!S))
    sampled_index <- sample(false_indices,1)
    S[sampled_index] <- T # takes a "F" from S, turns to T
    beta[sampled_index] <- rnorm(1) # adds new value to that index
  }
} else if ((u >= .5) & (u < .75)) { # delete
  # print("delete")
  if (sum(S) > 0){ # checks to see we have at least one predictor
    true_indices <- as.vector(which(S))
    S[sample(true_indices,1)] <- F # takes a "T" from S, turns to F
  }
  beta = S * beta
} else if (u >= .75){ # swap
  # print("swap")
    true_indices <- as.vector(which(S))
    swap_indices <- sample(1:p, length(true_indices))
    old_beta <- beta
    beta[swap_indices] <- old_beta[true_indices]
    beta[-swap_indices] <- 0
}
  return(beta)
}

# now need to accept or not accept transition
get_ratio <- function(y, beta1, beta2, data_mat){
  post_beta1 <- -sum(((y - (data_mat %*% as.matrix(beta1)))^2) / 2)
  post_beta2 <- -sum(((y - (data_mat %*% as.matrix(beta2)))^2) / 2)
  ratio <- min(1, exp(post_beta1 - post_beta2))
  return(ratio)
}



#try to run many times
run_sim <- function(y, data_mat, m){
  p <- ncol(data_mat)
  S <- (runif(p) >= .5)
  beta <- S*(rnorm(p))
  beta_list <- vector("list", m)
  for (i in 1:m) {
    beta_candidate <- update_beta(beta)
    ratio <- get_ratio(y, beta_candidate, beta, data_mat)
    u <- runif(1)
    if (u < ratio) {
      beta <- beta_candidate
    }
    beta_list[[i]] <- beta
  }
  return(beta_list)
}

get_sim_post_prob <- function(sim_list){ #get the posterior probabilities for each set S
  p <- length(sim_list[[1]])
  model_probs <- matrix(nrow = length(sim_list),
                        ncol = p)
  for (i in 1:length(sim_list)) {
    for (j in 1:p) {
       model_probs[i,j] <- toString(sim_list[[i]][j] != 0)
    }
  }
  model_probs <- data.frame(model_probs) %>%
    group_by_all() %>%
    count() %>%
    arrange(desc(n))
  return(model_probs)
}

# n = 500, p = 5
data <- gen_data(500, 5)
sim1 <- run_sim(y = data$y, data_mat = data$data_mat, m = 50000)
s_probs <- get_sim_post_prob(sim1)
#posterior prob
s_probs %>%
  mutate(prob = n / 50000)
```

```
## # A tibble: 7 x 7
## # Groups:   X1, X2, X3, X4, X5 [7]
##   X1    X2    X3    X4    X5        n    prob
##   <chr> <chr> <chr> <chr> <chr> <int>   <dbl>
## 1 FALSE FALSE TRUE  TRUE  TRUE  36416 0.728
## 2 TRUE  FALSE TRUE  TRUE  TRUE   6111 0.122
## 3 FALSE TRUE  TRUE  TRUE  TRUE   4949 0.0990
## 4 TRUE  TRUE  TRUE  TRUE  TRUE   2512 0.0502
## 5 FALSE FALSE FALSE TRUE  TRUE      8 0.00016
## 6 TRUE  FALSE FALSE FALSE TRUE      3 0.00006
## 7 FALSE FALSE FALSE FALSE TRUE      1 0.00002
```

```r
# true S (non zero)
sort(data$coef_index)
```

```
## [1] 3 4 5
```

```r
#looks like it works pretty well

# try again for p = 10
data10 <- gen_data(1000, 10)
sim10 <- run_sim(y = data10$y, data_mat = data10$data_mat, m = 50000)
s_probs10 <- get_sim_post_prob(sim10)
#posterior prob
s_probs10 %>%
  mutate(prob = n / 50000)
```

```
## # A tibble: 82 x 12
## # Groups:   X1, X2, X3, X4, X5, X6, X7, X8, X9, X10 [82]
##    X1    X2    X3    X4    X5    X6    X7    X8    X9    X10       n   prob
##    <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <int>  <dbl>
##  1 FALSE FALSE TRUE  FALSE FALSE FALSE FALSE FALSE FALSE FALSE 41218 0.824
##  2 FALSE FALSE TRUE  FALSE FALSE FALSE FALSE FALSE TRUE  FALSE  1742 0.0348
##  3 FALSE FALSE TRUE  FALSE TRUE  FALSE FALSE FALSE FALSE FALSE  1159 0.0232
##  4 FALSE FALSE TRUE  FALSE FALSE TRUE  FALSE FALSE FALSE FALSE   819 0.0164
##  5 FALSE FALSE TRUE  FALSE FALSE FALSE TRUE  FALSE FALSE FALSE   756 0.0151
##  6 FALSE FALSE TRUE  FALSE FALSE FALSE FALSE FALSE FALSE TRUE    662 0.0132
##  7 TRUE  FALSE TRUE  FALSE FALSE FALSE FALSE FALSE FALSE FALSE   596 0.0119
##  8 FALSE FALSE TRUE  FALSE FALSE FALSE FALSE TRUE  FALSE FALSE   547 0.0109
##  9 FALSE TRUE  TRUE  FALSE FALSE FALSE FALSE FALSE FALSE FALSE   539 0.0108
## 10 FALSE FALSE TRUE  TRUE  FALSE FALSE FALSE FALSE FALSE FALSE   521 0.0104
## # i 72 more rows
```

```r
# true S (non zero)
sort(data10$coef_index)
```

```
## [1] 3
```