Report

Qi Yuchen

2020/2/11

Objectives

In this project, we attempt to compare two automated methods of variable selection, namely stepwise forward method and LASSO regression, in their ability to correctly identify relevant signals and estimating how missing weak signals impact coefficients of strong signals. The two tasks are (1) how well each of the two methods in identifying weak and strong predictors, and (2) how missing "weak" predictors impacts the parameter estimations.

Statistical methods to be studied

Methods of interest in this report are the step-wise forward method and automated LASSO regression which are two popular methods for the variable selection.

Step-wise forward method: Starting with the empty model, and iteratively adds the variables that best improves the model fit. In this report, it is done by sequentially adding predictors with the largest reduction in AIC, where

$$AIC = n\ln(\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2/n) + 2p,$$

where $\widehat{y}_i$ is the fitted values from a model, and p is the dimension of the model (i.e.,number of predictors plus 1).

Automated LASSO regression It estimates the model parameters by optimizing a penalized loss function:

$$\min_{\beta} \frac{1}{2n}\sum_{i=1}^{n}(y_i - x_i\beta)^2 + \lambda\|\sum_{k=1}^{p}|\beta_k|$$

where λ is a tunning parameter. Here cross-validation (CV) is the chosen selection criteria for LASSO.

Scenarios to be investigated

First we give the definitions of "strong", "weak-but-correlated" and "weak-andindependent" signals.

Definition of strong signals —

$$S_1 = \{j : |\beta_j| > c\sqrt{log(p)/n},\ \text{some } c > 0, 1 \le j \le p\}$$

Definition of weak-but-correlated signals —

$$S_2 = \{j : 0 < |\beta_j| \le c\sqrt{log(p)/n},\ \text{some } c > 0, \text{corr}(X_j, X_j') \ne 0, \text{for some } j' \in S_1, 1 \le j \le p\}$$

Definition of weak-and-independent signals —

$$S_3 = \{j : 0 < |\beta_j| \le c\sqrt{log(p)/n},\ \text{some } c > 0, \text{corr}(X_j, X_j') = 0, \text{for all } j' \in S_1, 1 \le j \le p\}$$

To narrow the scope of our simulations, we set the proportions of strong signals, weak and independent signals, and weak but correlated signals to be 10%, 20%, 20% respectively, then we have 50% null predictors. The coefficients of strong signals follows $Uniform(5, 10)$ which is sufficiently larger than the bound, and the

coefficients of strong signals follows $Uniform(1/2bound, bound)$, where the bound is the upper bound for the value of weak signal coefficients by definition. The theshold multiplier c is set to be 1.

Then, we vary the amount of total predictors from 10 to 100, with step to be 10. We also choose the correlation value to be 0.3, 0.5, 0.7. For each scenario, we generate 100 datasets. And in each dataset, the sample size is 200.

Methods for generating data

From the proportions of each type of signals and the number of total predictors, we get how many signals for each type. Then we generate a covariance matrix with the correlations set in this scenario following the definitions of each signal type. For each strong signal, there are two weak but correlated signals that are correlated to it. Whether the matrix is positive definite is also checked before passing it to the function `mvrnorm`, which produces random numbers from a multivariate normal distribution. Finally we generate the response Y as a linear combination of four types of signals and an error term. The code is shown below.

```
sim_beta_strong = function(n_strong, coef_strong){
  rep(coef_strong, n_strong) + runif(n_strong, min = 0, max = coef_strong)
}


sim_data = function(n_sample = 200, n_parameter = 50, prop_strong = 0.1, prop_wbc = 0.2, prop_wai = 0.2
  # Numbers of four signals
  n_strong = as.integer(n_parameter * prop_strong) # strong
  n_wbc = as.integer(n_parameter * prop_wbc) # weak but correlated
  n_wai = as.integer(n_parameter * prop_wai) # weak and independent
  n_null = n_parameter - n_strong - n_wbc - n_wai # null

  if (n_null < 0) {
    return("Given parameters' proportions are not valid.")
  }

  bound = c * sqrt(log(n_parameter) / n_sample) # threshold of weak/strong, the default is 0.14
  if (coef_strong < bound) {
    coef_strong = coef_strong + 2 * bound
  }

  cor_matrix = diag(n_parameter)

  # add correlation
  for (i in 1:n_strong) {
    cor_matrix[i, (n_strong + n_wai + i)] = cor
    cor_matrix[i, (n_strong + n_wai + n_wbc + 1 - i)] = cor
    cor_matrix[(n_strong + n_wai + i), i] = cor
    cor_matrix[(n_strong + n_wai + n_wbc + 1 - i), i] = cor
  }


  if (!is.positive.definite(cor_matrix)) {
    return("The correlation matrix is not valid.")
  }

  # simulate the data from multivariate normal
  X = mvrnorm(n = n_sample, mu = rep(0, n_parameter), Sigma = cor_matrix) # var = 1, correlation = cova
```

```
  beta = c(
    sim_beta_strong(n_strong, coef_strong),
    runif(min = bound/2, max = bound, n = n_wai),
    runif(min = bound/2, max = bound, n = n_wbc),
    rep(0, n_null)
  )

  Y = 1 + X %*% beta + rnorm(n_sample)
  data = as_tibble(data.frame(cbind(X, Y)))

  # Name the columns
  cols = c(
    str_c("strong", 1:n_strong, sep = "_"),
    str_c("wai", 1:n_wai, sep = "_"),
    str_c("wbc", 1:n_wbc, sep = "_"),
    str_c("null", 1:n_null, sep = "_"),
    "Y"
   )
   colnames(data) = cols
   data = data %>%
     dplyr::select(Y, everything())

  list(beta = beta,
       correlation = cor,
       n_parameter = n_parameter,
       prop_strong = prop_strong,
       prop_wbc = prop_wbc,
       prop_wai = prop_wbc,
       n_strong = n_strong,
       n_wai = n_wai,
       n_wbc = n_wbc,
       data = data
       )

}
```