Homework 3

Hengle Li

2/8/2020

```r
library(tidyverse)
library(rsample)
library(rcfss)
library(leaps)
library(yardstick)
```

Training/test error for subset selection

```r
#Assuming the features of X are unknown, each X, is just a vector
#of length 20, containing 20 random numbers
X_generate <- function(x){
  set.seed(x)
  sample(0:100, 20, replace = TRUE)
}

X_list <- map(1:1000, X_generate)
```

```r
#beta is a known vector of length 20 with a number of 0s.
set.seed(1)
beta <- sample(1:20, 20, replace = TRUE)
#examining the numbers within beta
beta
```

```
##  [1]  4  7  1  2 11 14 18 19  1 10 14 10  7  9 15  5  9 14  5  5
```

```r
#assume there are 3 0s in beta, choose three random places
set.seed(2)
sample(1:20, 3, replace = FALSE)
```

```
## [1] 15  6 19
```

```r
#the 6th, 15th, and 19th will be 0
beta[c(6, 15, 19)] <- 0
#examine beta again
beta
```

```
##  [1]  4  7  1  2 11  0 18 19  1 10 14 10  7  9  0  5  9 14  0  5
```

```r
#since there's no requirement for epsilon, just use a random vector
#of length 1000, one number for each Y
set.seed(66)
epsilon <- sample(0:100, 1000, replace = TRUE)
```

By the definition of Y, it's a vector of 20. Therefore, the entire dataset of Y's is what we are looking for.

```r
#use a for loop to create a list of Y
out <- vector(mode = "list",
              length = 1000)
  for(i in seq_along(out)){
    out[[i]] <- sum(X_list[[i]] * beta + epsilon[[i]])}
```

```r
#transform X_list to a proper dataset
df <- as.data.frame(X_list)
colnames(df) <- c(seq(1:1000))
df <- as.data.frame(t(df))
df <- cbind(df, Y = as.integer(out))
```

```r
#split the dataset
df_split <- initial_split(df, 0.1)
df_train <- training(df_split)
df_test <- testing(df_split)
```

```r
#create the model
#set size of subset to 20, because there are only 20 predictors
best_subset <- regsubsets(Y ~ .,
                          data = df_train,
                          nvmax = 20
                          )
```

reasonings

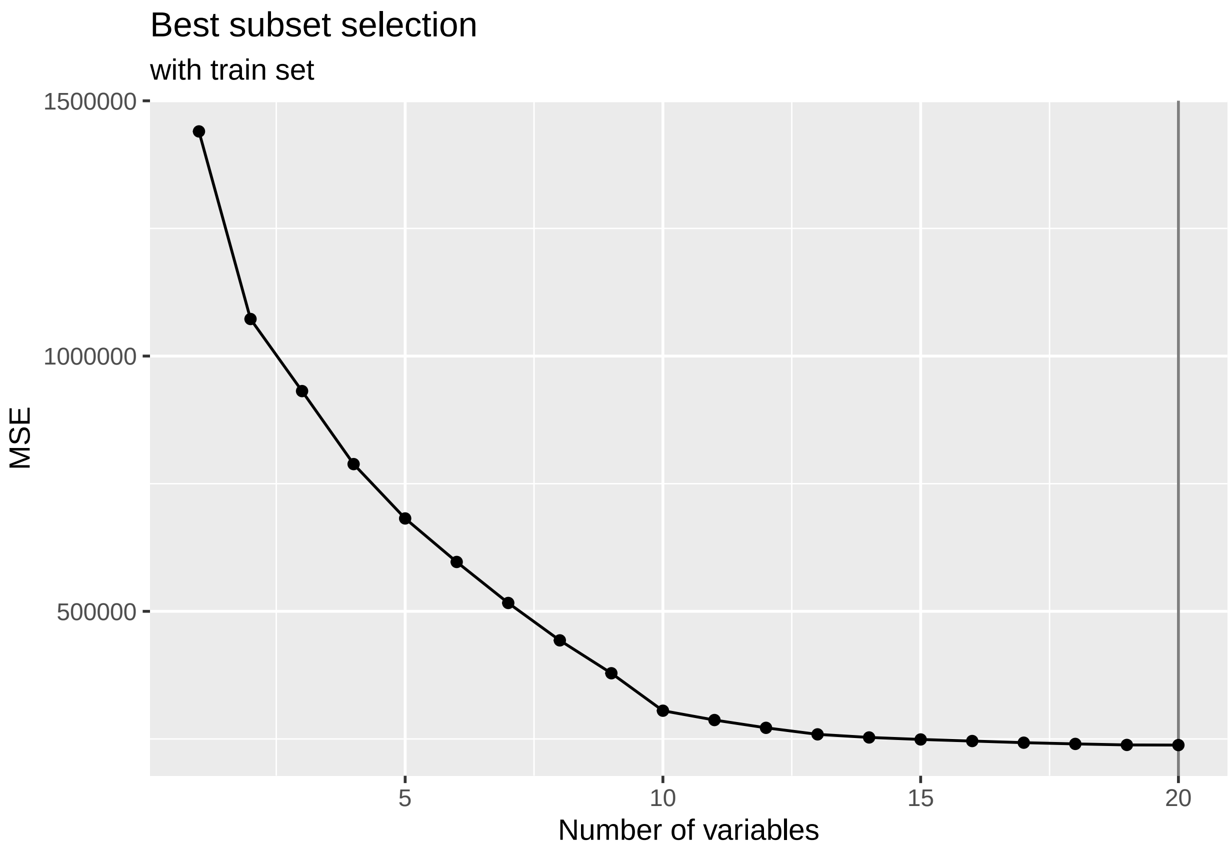
- best_subset contains a list of models
- coefficients of models in best_subset can be extracted by `coef()`
- values of X can be extracted from the dataset by `model.matrix()` with the training set
- for each possible number of predictors, there are 100 results, because there are 100 observations in the training set

```r
#construct functions for prediction
predict_best <- function(model, newdata, id){
  formulae <- as.formula(Y ~ .) #set formula for model.matrix()
  matrix_X <- model.matrix(formulae, newdata) #set model.matrix()
  variables <- names(coef(model, id = id)) #decide which variables to use
  #multiply X with coefficients to produce predictions
  as.vector(matrix_X[, variables] %*% coef(model, id = id), mode = "integer")
}
```

```r
#there will be 100 results for each possible n
result_train <- tibble(n = 1:20, pred = map(1:20, ~ predict_best(best_subset, df_train, .x))) %>%
  unnest(pred) %>%
  mutate(truth = rep(df_train$Y, 20))
#repeating the sequence of Y to match it with the results

#calculate MSE
train_mse <- result_train %>%
  group_by(n) %>%
  mse(truth = truth, estimate = pred)
```

```
train_mse %>%
  ggplot(aes(x = n, y = .estimate)) +
  geom_line() +
  geom_point() +
  geom_vline(xintercept = which.min(train_mse$.estimate), alpha = .5) +
  labs(title = "Best subset selection",
       subtitle = "with train set",
       x = "Number of variables",
       y = "MSE")
```

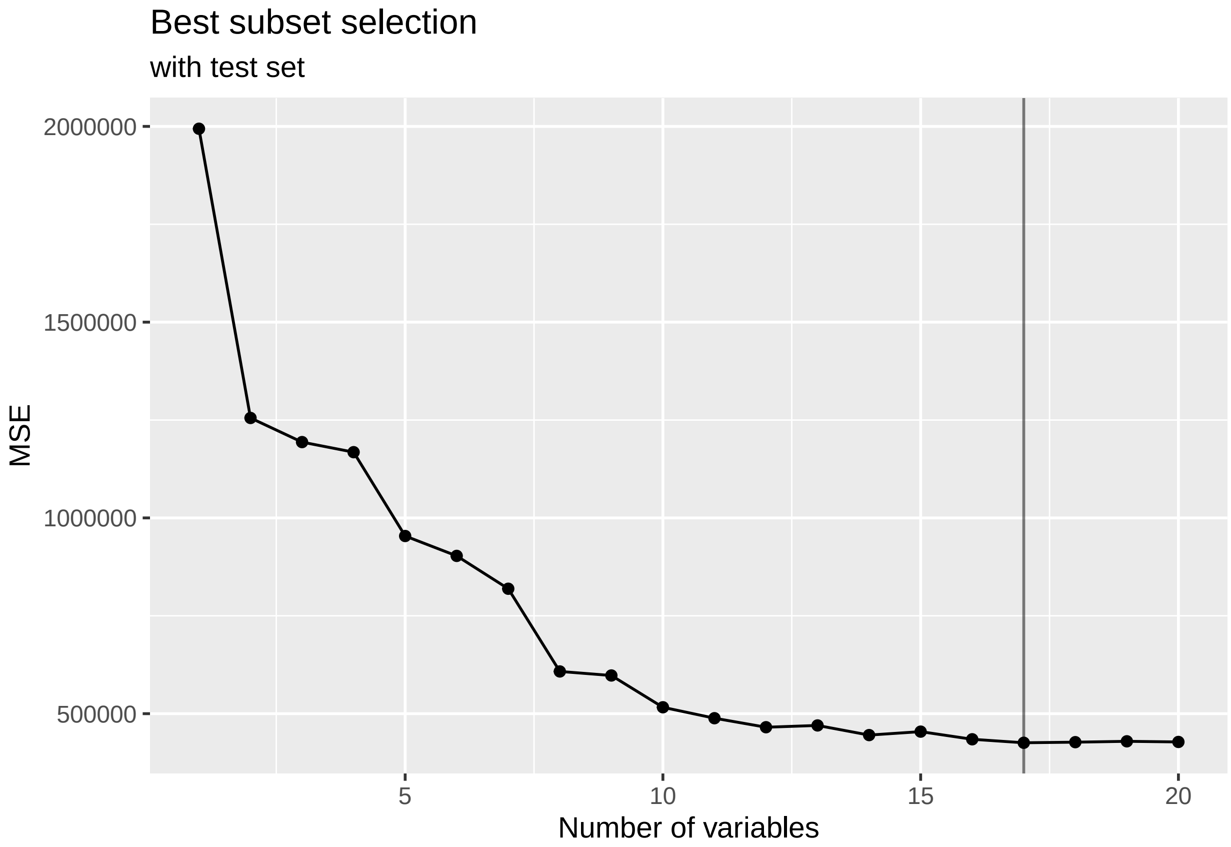


It is not surprising that when n = 20, MSE reaches minimum. It's a feature of machine learning in dealing with the training set: more predictors, higher accuracy.

```
#there will be 900 results for each possible n
result_test <- tibble(n = 1:20, pred = map(1:20, ~ predict_best(best_subset, df_test, .x))) %>%
  unnest(pred) %>%
  mutate(truth = rep(df_test$Y, 20))
#again, repeating the sequence of Y to match it with the results
```

```
#calculate MSE
test_mse <- result_test %>%
  group_by(n) %>%
  mse(truth = truth, estimate = pred)
```

```
test_mse %>%
  ggplot(aes(x = n, y = .estimate)) +
  geom_line() +
  geom_point() +
  geom_vline(xintercept = which.min(test_mse$.estimate), alpha = .5) +
  labs(title = "Best subset selection",
       subtitle = "with test set",
       x = "Number of variables",
       y = "MSE")
```



The graph indicates when 17 variables are included, MSE reaches minimum for the test set. It's noteworthy that 20 is no longer the best subset size here. On one hand, it's natural that when a feature is added, the MSE decreases due to the increased explanation power. On the other hand, one can see that for n equals 16 or above, the differences in MSE are quite small compared with those for n smaller than 16, i.e. the curve is flattened. Moreover, since there are 3 0s in beta, it's intuitively understandable that MSE reaches minimum at n=17: 20-3=17.

```
#coefficients of the model when n is 17
coef(best_subset, 17)
```

```
## (Intercept)          V1          V2          V4          V5          V7 
##  842.156469    5.892131    2.561686    2.036580   12.194403   17.724406 
##          V8          V9         V10         V11         V12         V13 
##   18.909786    4.244702    9.429593   11.411053   13.453363    9.668298 
##         V14         V16         V17         V18         V19         V20 
##    9.870179    5.382199    8.947430   11.574887    2.583881    2.329960 
```

```r
#true value of the coefficients
beta
```

```
##  [1]  4  7  1  2 11  0 18 19  1 10 14 10  7  9  0  5  9 14  0  5
```

```r
#the mean value of epsilon
mean(epsilon)
```

```
## [1] 49.929
```

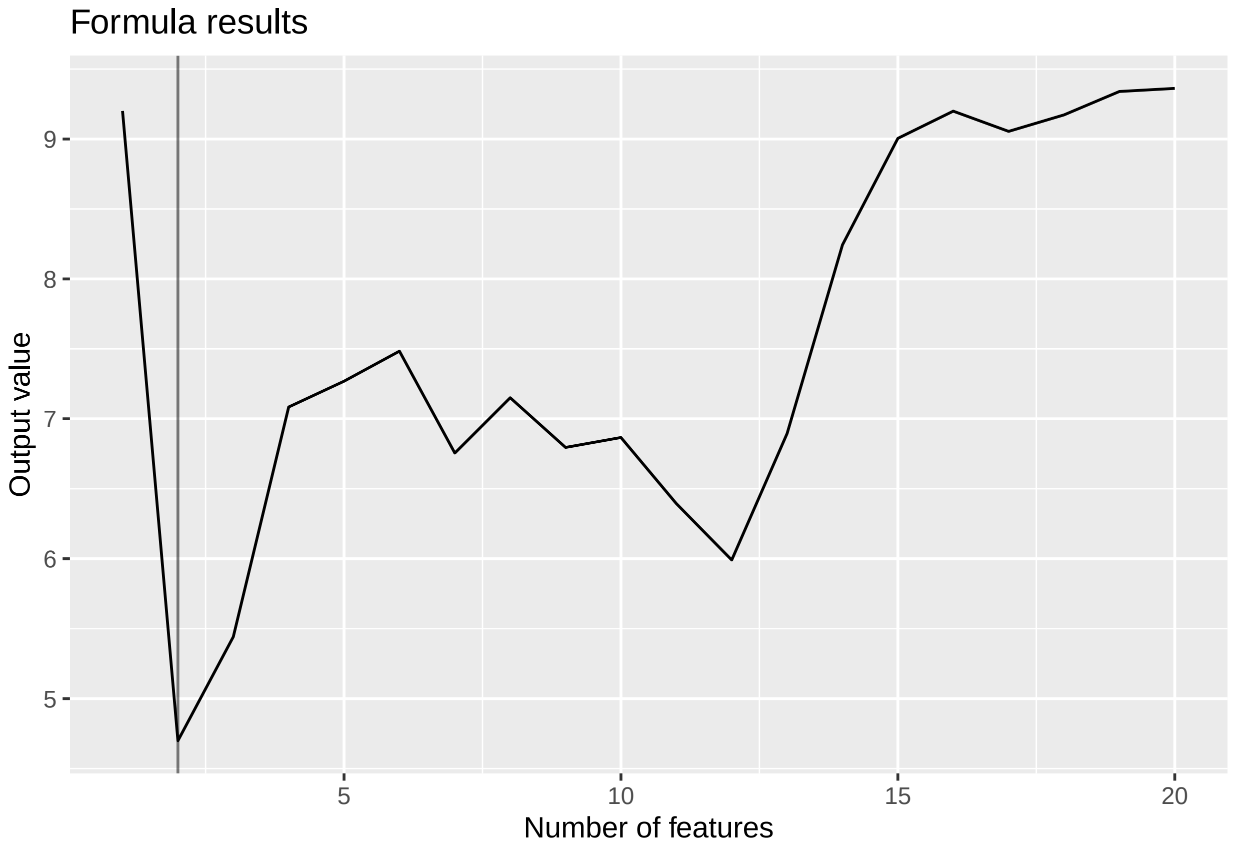
Comparing the coefficients of the model and the values in beta, we can see that the model leaves out V3, V6, V15, while the true null coefficients are V6, V15, V19, i.e. it misses only one coefficient. On the other hand, some coefficients are very close to the actual value of beta, including V4, V5, V7, V8, V10 etc. It should also be noted that the value of the intercept is very different from the mean value of epsilon, which shows the difference between high-dimensional space and the general understanding of 2d space.

```r
#transform beta into a dataset for processing
beta_frame <- as.data.frame(t(beta))

#function to calculate through the formula
fancy <- function(x){
  set <- as.data.frame(t(coef(best_subset, x)))
  set <- set[-c(1)]
  process <- beta_frame %>%
    dplyr::select(colnames(set))
  sqrt(sum((process[1,] - set[1,])^2))
}

#calculate the numbers
output <- map(1:20, fancy)
results_form <- tibble(n = 1:20, output = as.numeric(output))
```

```r
results_form %>%
  ggplot(aes(x = n, y = output)) +
  geom_line() +
  geom_vline(xintercept = which.min(results_form$output), alpha = .5) +
  labs(title = "Formula results",
       x = "Number of features",
       y = "Output value")
```



The graph shows when n = 2, the stat reaches its minimum. If we judge by this graph, the best subset selection should be when n is 2. However, when n is 2, there are only 2 features included in the model. By the formula, it's bound to be smaller than any other n: when n is 1, the model coefficient will be much different from its counterpart in beta for greater accuracy; when n is greater than 1, the more coefficients are included, the greater the stat will be in general. The intuitive of this stats is contradictory to that of best subset selection. Therefore, the stat is not a reliable way to tell which model best fits the datasets. Its accuracy is far worse than calculating MSE for the test set. If calculating test set MSE shows 17 is the best subset size, this stat shows 17 is a very poor size in performance.