# Q21

Rahul Atre

2023-10-30

## Q21 [Regularization]

We have seen that as the number of features used in a model increases, the training error will necessarily decrease, but the test error may not. We will now explore this in a simulated data set.

1. Generate a data set with p = 20 features, n = 1,000 observations, and an associated quantitative response vector generated according to the model

$$Y = X\beta + \epsilon,$$

where $\beta$ has some elements that are exactly equal to zero.

We can generate the data using the following command:

```
set.seed(144) #Ensure reproducible outputs for simulation

p = 20 # Declare feature and observation values
n = 1000

x = matrix(rnorm(n * p), n, p) #rnorm generates a vector which is reshaped to matrix (obs. vs. features

b = rnorm(p) # Beta vector for model
randomCoefficients <- sample(1:length(b), 5)
b[randomCoefficients] = 0 #Randomly select 5 indices of b and set them to 0

eps = rnorm(n)

y = x %*% b + eps #Create linear model
```

2. Split your dataset into a training set containing 100 observations and a test set containing 900 observations.

```
training_set = sample(seq(n), 100, replace = FALSE) #Create training set of 100 obs.
test_set = sample(seq(n), 900, replace = FALSE) #Create test set of 900 obs.

x.training = x[training_set, ]
x.test = x[test_set, ]

y.training = y[training_set]
y.test = y[test_set]
```

3. Perform best subset selection on the training set, and plot the training set MSE associated with the best model of each size.

```
library(leaps) #regsubsets import
```

```
## Warning: package 'leaps' was built under R version 4.3.2
```

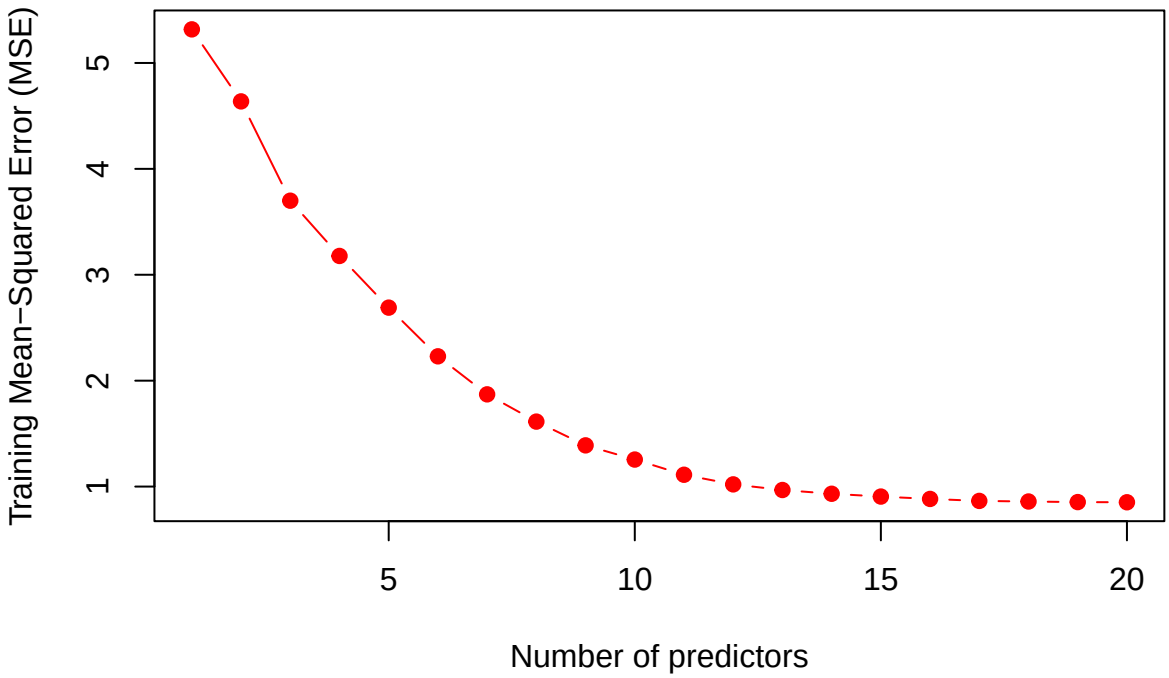
```
#Contain training predictor and response into data frame
df_training <- data.frame(x=x.training, y=y.training)

training_matrix <- model.matrix(y ~ ., data=df_training, nvmax = p) #Construct matrix
best_subset <- regsubsets(y ~ ., data=df_training, nvmax = p) #Model selection

value_errs <- rep(NA, p) #Variable to store training MSE

for(i in 1:p) {
 coefficients <- coef(best_subset, id = i) #Obtain coefficients of model from i predictors
 pred <- training_matrix[,names(coefficients)] %*% coefficients #prediction value
 value_errs[i] <- mean((pred - y.training)^2) #training MSE
}

# Plot training MSE
plot(value_errs, xlab = "Number of predictors", ylab = "Training Mean-Squared Error (MSE)", pch = 19, ty
```




4. Plot the test set MSE associated with the best model of each size.

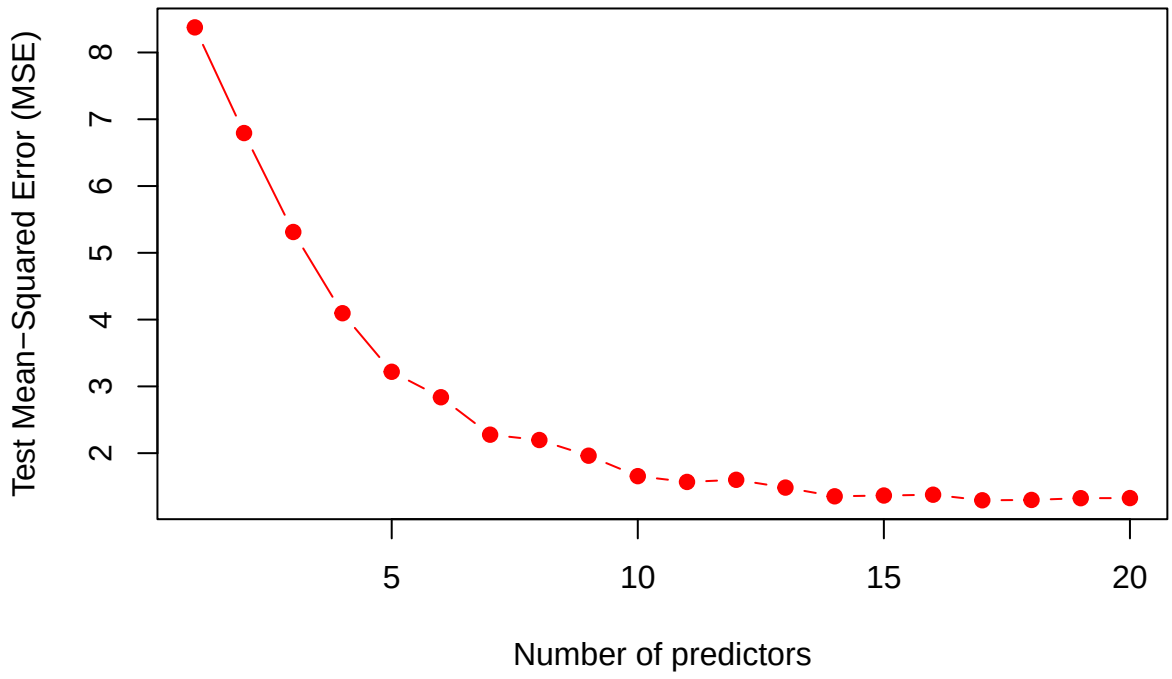
```r
#Contain test predictor and response into data frame
df_test <- data.frame(x=x.test, y=y.test)

test_matrix <- model.matrix(y ~ ., data=df_test, nvmax = p) #Construct matrix
best_subset <- regsubsets(y ~ ., data=df_training, nvmax = p) #Model selection

value_errs <- rep(NA, p) #Variable to store test MSE

for(i in 1:p) {
 coefficients <- coef(best_subset, id = i) #Obtain coefficients of model from i predictors
 pred <- test_matrix[,names(coefficients)] %*% coefficients #prediction value
 value_errs[i] <- mean((pred - y.test)^2) #training MSE
}

# Plot test MSE
plot(value_errs, xlab = "Number of predictors", ylab = "Test Mean-Squared Error (MSE)", pch = 19, type =
```




5. For which model size does the test set MSE take on its minimum value? Comment on your results. If it takes on its minimum value for a model containing only an intercept or a model containing all of the features, then play around with the way that you are generating the data in 1. until you come up with a scenario in which the test set MSE is minimized for an intermediate model size.

```r
which.min(value_errs)
```

```
## [1] 17
```

For a model size with the current given parameters (i.e. n=1000, p=20, etc), the test set MSE takes on a minimum value of **17**. However, while experimenting with different feature values, it appears that a more common value is **14**. We can say that this is the minimum test MSE for an intermediate model size.

6. How does the model at which the test set MSE is minimized compare to the true model used to generate the data? Comment on the coefficient values.

```
coef(best_subset, which.min(value_errs))
```

```
## (Intercept)         x.1         x.2         x.3         x.4         x.5
##  -0.1297521   0.5421452   1.0454305   0.3677487  -0.2098152  -0.1679244
##         x.6         x.8         x.9        x.10        x.12        x.13
##   0.1744438   1.3894826  -0.4263537  -0.8139130  -0.9904887  -0.1488341
##        x.14        x.15        x.16        x.18        x.19        x.20
##  -0.2494174  -1.1252237   0.8418976  -0.6109398  -0.4637915  -1.2374532
```

```
print(paste("The coefficients that we set to 0 in the true model are: ", list(randomCoefficients)))
```

```
## [1] "The coefficients that we set to 0 in the true model are:  c(6, 17, 5, 7, 11)"
```

In the true model, the X parameters that have gone to zero are (6, 17, 5, 7, 11). In the minimized test set, the values that have gone to zero are (7, 11, 17).

The model was able to remove 3 out of the 5 zero'd out coefficients, but missed 2, which are $X_5$ and $X_6$.

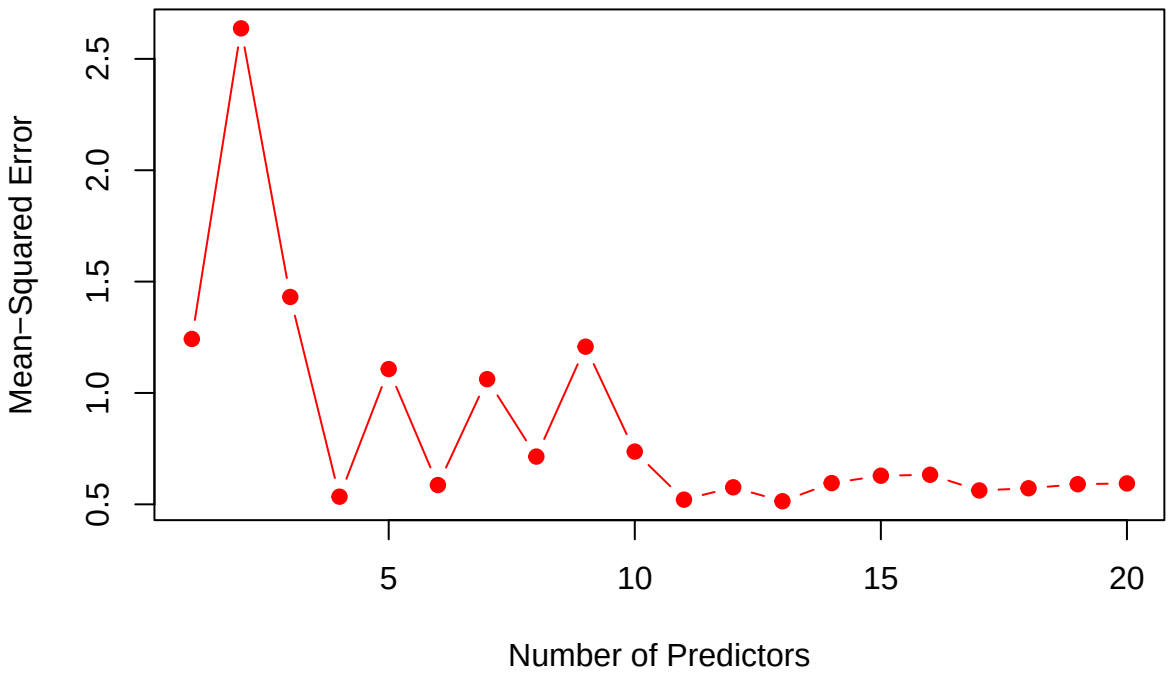
The coefficient values for these two are very small and close to 0, so it is understandable why the minimized test MSE model was not able to identify these two predictors and remove them.

7. Create a plot displaying $\sqrt{\sum_{j=1}^{p}(\beta_j - \hat{\beta}_j^r)^2}$ for a range of values of $r$, where $\hat{\beta}_j^r$ is the $j^{th}$ coefficient estimate for the best model containing $r$ coefficients. Comment on what you observe. How does this compare to the test MSE plot from 4.?

```
value_errs = rep(NA, p) #Reset value errors
x_cols = colnames(x, do.NULL = FALSE, prefix = "x.") #Obtain column names of x matrix

for (i in 1:p) { # Iterate over models to find best one
coefs = coef(best_subset, id = i)
value_errs[i] = sqrt(sum((b[x_cols %in% names(coefs)] - coefs[names(coefs) %in% x_cols])^2) + sum(b[!(x_
}

#Display graph for minimized error
plot(value_errs, xlab = "Number of Predictors", ylab = "Mean-Squared Error", pch = 19, type = "b", col=
```




```
which.min(value_errs)
```

```
## [1] 13
```

As we can see from the above calculations, the model for the estimated coefficients returns a different minimum value than the test set MSE. This indicates that a better fit for the coefficients does not imply a lower test set MSE.