Resampling and Regularization Homework

Megha Pandit

February 14, 2019

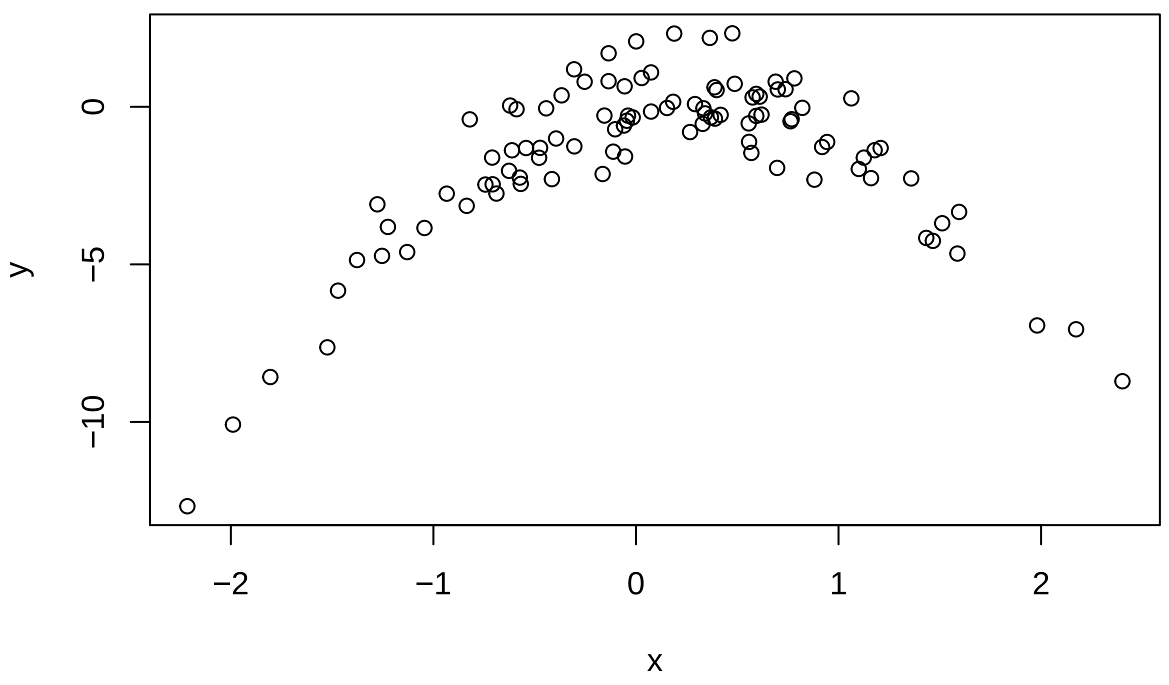
5.8. We will now perform cross-validation on a simulated data set. (a) Generate a simulated data set as follows:

```r
set.seed (1)
x <- rnorm (100)
y <- x-2* x^2+ rnorm (100)
```

In this data set, what is n and what is p? Write out the model used to generate the data in equation form. (b) Create a scatterplot of X against Y . Comment on what you find.

The model used to generate the data is $y = (x-2)x^2 + \epsilon$ In this data set, n = 100 and p = 2.

```r
plot(x,y)
```



From the scatterplot, we see a clear non-linear relationship between x and y. The plot looks more like an inverted parabola implying a quadratic relationship between x and y. And, the range of x seems to be approximately from -2 to 2.

(c) Set a random seed, and then compute the LOOCV errors that result from fitting the following four models using least squares:

i. Y = $\beta_0 + \beta_1 X + \epsilon$
ii. Y = $\beta_0 + \beta_1 X + \beta_2 X^2 + \epsilon$

iii. Y $= \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \epsilon$

iv. Y $= \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \beta_4 X^4 + \epsilon$. Note you may find it helpful to use the data.frame() function to create a single data set containing both X and Y .

```r
# i)
set.seed(1)
df <- data.frame(x,y)
fit1 <- glm(y ~ x, data = df)
print(paste0("LOOCV error for i): ", cv.glm(df, fit1)$delta[1]))
```

```
## [1] "LOOCV error for i): 7.28816160667281"
```

```r
# ii)
fit2 <- glm(y ~ poly(x,2), data = df)
print(paste0("LOOCV error for ii): ", cv.glm(df, fit2)$delta[1]))
```

```
## [1] "LOOCV error for ii): 0.937423637615552"
```

```r
# iii)
fit3 <- glm(y ~ poly(x,3), data = df)
print(paste0("LOOCV error for iii): ", cv.glm(df, fit3)$delta[1]))
```

```
## [1] "LOOCV error for iii): 0.95662183010894"
```

```r
# iv)
fit4 <- glm(y ~ poly(x,4), data = df)
print(paste0("LOOCV error for iv): ", cv.glm(df, fit4)$delta[1]))
```

```
## [1] "LOOCV error for iv): 0.953904892744804"
```

(d) Repeat (c) using another random seed, and report your results. Are your results the same as what you got in (c)? Why?

```r
set.seed(99)
df <- data.frame(x,y)

# i)
fit1 <- glm(y ~ x, data = df)
print(paste0("LOOCV error for i): ", cv.glm(df, fit1)$delta[1]))
```

```
## [1] "LOOCV error for i): 7.28816160667281"
```

```r
# ii)
fit2 <- glm(y ~ poly(x,2), data = df)
print(paste0("LOOCV error for ii): ", cv.glm(df, fit2)$delta[1]))
```

```
## [1] "LOOCV error for ii): 0.937423637615552"
```

```r
# iii)
fit3 <- glm(y ~ poly(x,3), data = df)
print(paste0("LOOCV error for iii): ", cv.glm(df, fit3)$delta[1]))
```

```
## [1] "LOOCV error for iii): 0.95662183010894"
```

```r
# iv)
fit4 <- glm(y ~ poly(x,4), data = df)
print(paste0("LOOCV error for iv): ", cv.glm(df, fit4)$delta[1]))
```

```
## [1] "LOOCV error for iv): 0.953904892744804"
```

The results in (c) and (d) are the same because LOOCV trains the model on all the observations except one. Therefore, each time, the model is trained with the same set of observations for each cross validation set.

(e) Which of the models in (c) had the smallest LOOCV error? Is this what you expected? Explain your answer.

Model ii) has the least LOOCV error. This is expected as we saw in (b) that x and y share a quadratic relationship.

(f) Comment on the statistical significance of the coefficient estimates that results from fitting each of the models in (c) using least squares. Do these results agree with the conclusions drawn based on the cross-validation results?

```r
for (i in 1:4) {
  print(summary(glm(y ~ poly(x,i), data = df)))
}
```

```
##
## Call:
## glm(formula = y ~ poly(x, i), data = df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -9.5161  -0.6800   0.6812   1.5491   3.8183
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -1.550      0.260  -5.961 3.95e-08 ***
## poly(x, i)     6.189      2.600   2.380   0.0192 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 6.760719)
##
##     Null deviance: 700.85  on 99  degrees of freedom
## Residual deviance: 662.55  on 98  degrees of freedom
## AIC: 478.88
##
## Number of Fisher Scoring iterations: 2
##
##
## Call:
## glm(formula = y ~ poly(x, i), data = df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9650  -0.6254  -0.1288   0.5803   2.2700
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.5500     0.0958  -16.18  < 2e-16 ***
## poly(x, i)1   6.1888     0.9580    6.46 4.18e-09 ***
## poly(x, i)2 -23.9483     0.9580  -25.00  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## (Dispersion parameter for gaussian family taken to be 0.9178258)
##
##     Null deviance: 700.852  on 99  degrees of freedom
## Residual deviance:  89.029  on 97  degrees of freedom
## AIC: 280.17
##
## Number of Fisher Scoring iterations: 2
##
##
## Call:
## glm(formula = y ~ poly(x, i), data = df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9765  -0.6302  -0.1227   0.5545   2.2843
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.55002    0.09626 -16.102  < 2e-16 ***
## poly(x, i)1   6.18883    0.96263   6.429 4.97e-09 ***
## poly(x, i)2 -23.94830    0.96263 -24.878  < 2e-16 ***
## poly(x, i)3   0.26411    0.96263   0.274    0.784
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.9266599)
##
##     Null deviance: 700.852  on 99  degrees of freedom
## Residual deviance:  88.959  on 96  degrees of freedom
## AIC: 282.09
##
## Number of Fisher Scoring iterations: 2
##
##
## Call:
## glm(formula = y ~ poly(x, i), data = df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0550  -0.6212  -0.1567   0.5952   2.2267
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.55002    0.09591 -16.162  < 2e-16 ***
## poly(x, i)1   6.18883    0.95905   6.453 4.59e-09 ***
## poly(x, i)2 -23.94830    0.95905 -24.971  < 2e-16 ***
## poly(x, i)3   0.26411    0.95905   0.275    0.784
## poly(x, i)4   1.25710    0.95905   1.311    0.193
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.9197797)
##
##     Null deviance: 700.852  on 99  degrees of freedom
## Residual deviance:  87.379  on 95  degrees of freedom
## AIC: 282.3
##
## Number of Fisher Scoring iterations: 2
```

The results from the least squares model for the four models above are consistent with the LOOCV results. In model i) and ii), all the coefficients are statistically significant. But, in models iii) and iv), the coefficients of $X^3 and X^4$ are not statistically significant, implying that y is second degree polynomial dependent on x.

6.2. For parts (a) through (c), indicate which of i. through iv. is correct. Justify your answer. (a) The lasso, relative to least squares, is: i. More flexible and hence will give improved prediction accuracy when its increase in bias is less than its decrease in variance. ii. More flexible and hence will give improved prediction accuracy when its increase in variance is less than its decrease in bias. iii. Less flexible and hence will give improved prediction accuracy when its increase in bias is less than its decrease in variance. iv. Less flexible and hence will give improved prediction accuracy when its increase in variance is less than its decrease in bias.

The lasso is: iii) less flexible and hence will give improved prediction accuracy when its increase in bias is less than its decrease in variance. This is because the lasso restricts the number of predictors by shrinking some of them to be exactly zero. This reduced flexibility results in increase in bias buta decrease in variance. Therefore, when the increase in bias is less than the decrease in variance, lasso will have better prediction accuarcy relative to least squares.

(b) Repeat (a) for ridge regression relative to least squares.

Similar to the lasso, ridge regression also is: iii) less flexible and hence will give improved prediction accuracy when its increase in bias is less than its decrease in variance.

(c) Repeat (a) for non-linear methods relative to least squares.

Since non-linear methods are more flexible relative to least squares, ii) is the correct answer here. They are more flexible and hence will give better prediction accuracy when the increase in variance is less than the decrease in bias.

6.10. 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. (a) 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 β has some elements that are exactly equal to zero.

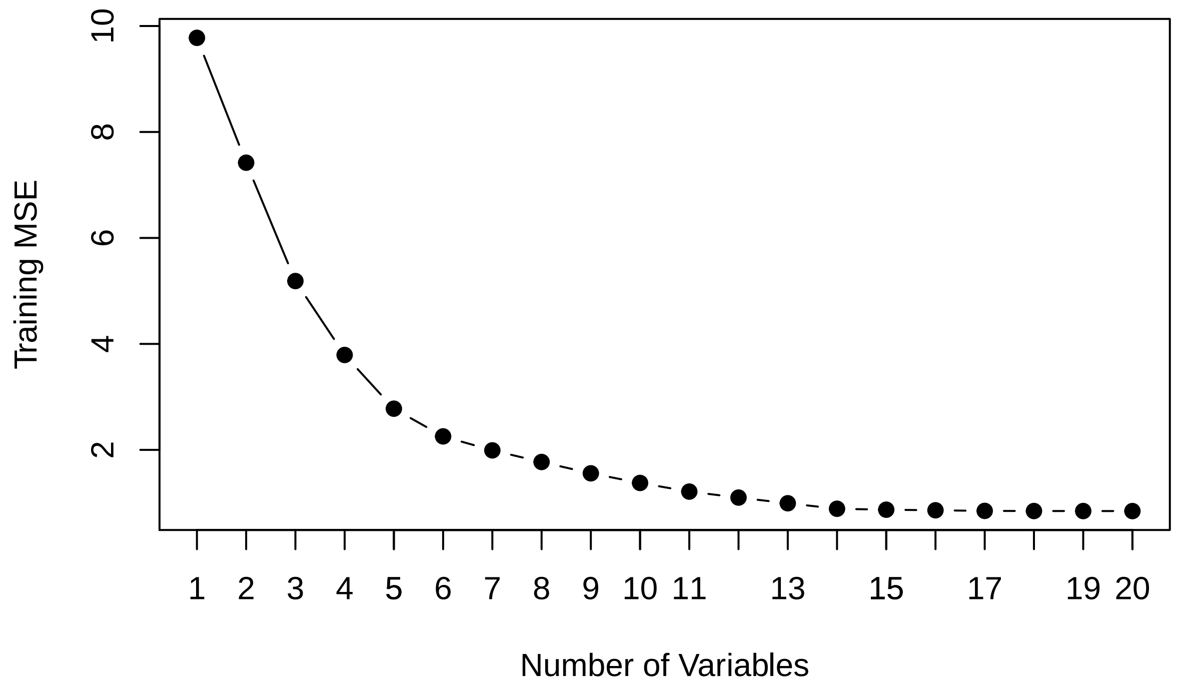
```r
set.seed(9)
x <- matrix(rnorm(1000*20), 1000, 20)
b <- matrix(rnorm(20), 20, 1)
b[2] <- 0
b[5] <- 0
b[9] <- 0
b[14] <- 0
b[18] <- 0

err <- rnorm(1000)

y <- x%*%b + err
```

(b) Split your data set into a training set containing 100 observations and a test set containing 900 observations.

```
Data <- data.frame(x,y)
train <- Data[1:100,]
test <- Data[101:1000,]
```

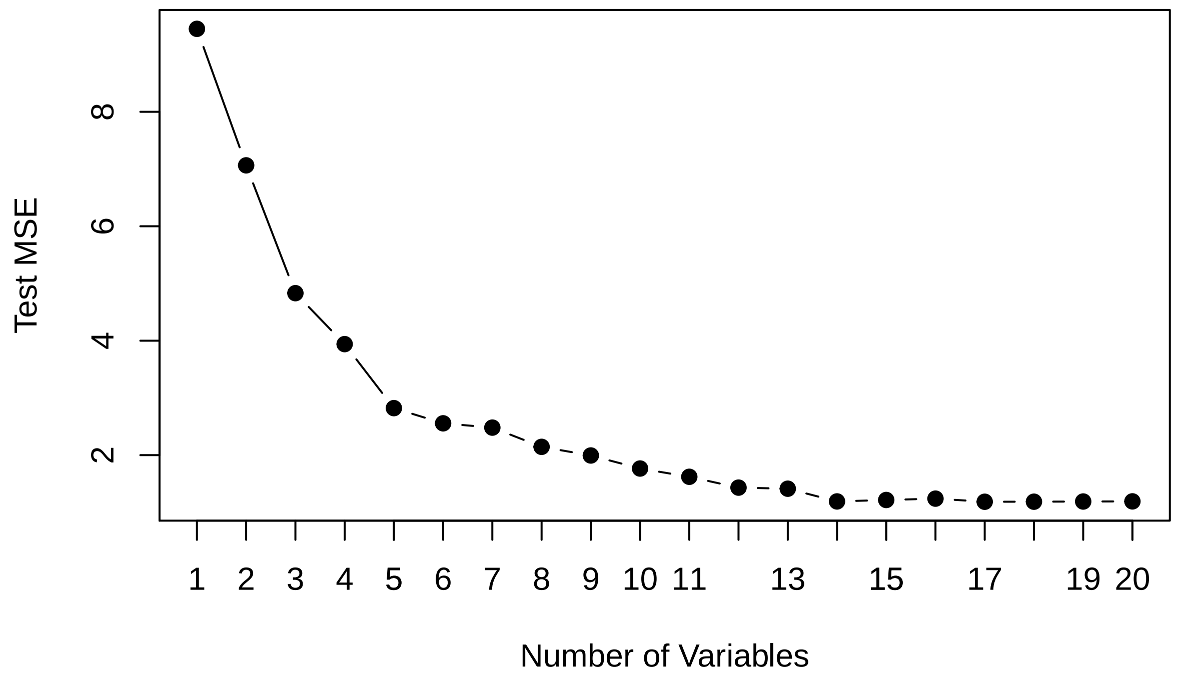
(c) Perform best subset selection on the training set, and plot the training set MSE associated with the best model of each size.

```
n <- 100
subset1 <- regsubsets(y ~., train, nvmax = 20)
plot((1/n)*summary(subset1)$rss, xlab = "Number of Variables", ylab = "Training MSE", type = "b", pch =
axis(1, at = seq(1, 20, 1))
```



(d) Plot the test set MSE associated with the best model of each size.

```
test.mat <- model.matrix(y ~., test, nvmax = 20)
errs <- rep(NA, 20)
for (i in 1:20) {
  coeff <- coef(subset1, id = i)
  pred <- test.mat[, names(coeff)] %*%coeff
  errs[i] <- mean((pred - test[,21])^2)
}
plot(errs, xlab = "Number of Variables", ylab = "Test MSE", type = "b", pch = 19)
axis(1, at = seq(1, 20, 1))
```



(e) 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 (a) until you come up with a scenario in which the test set MSE is minimized for an intermediate model size.

```
which.min(errs)
```

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

The model with 17 variables has the smallest MSE.

(f) 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.

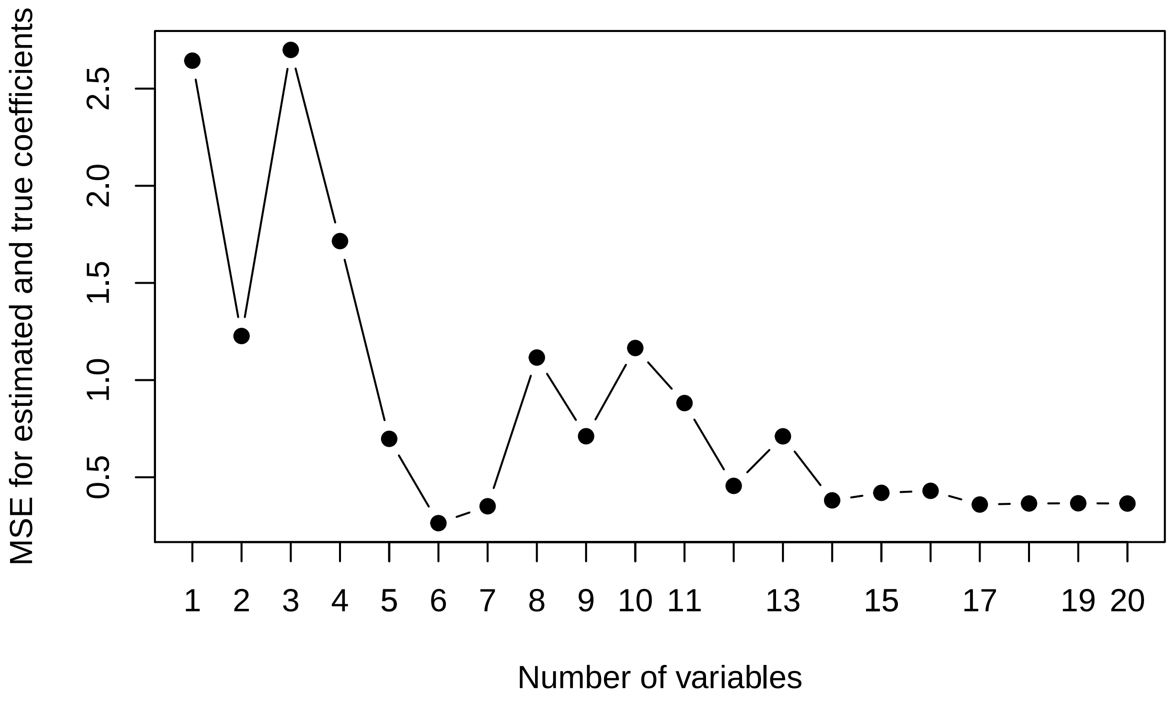
```
coef(subset1, which.min(errs))
```

```
## (Intercept)          X1          X2          X3          X4          X6
##   0.2055328  -0.1227184  -0.1237108  -0.4311793   0.3720967  -0.5984137
##          X7          X8         X10         X11         X12         X13
##  -0.4565580   1.4688580  -1.3445077   0.5176785  -1.6423076   0.4358270
##         X15         X16         X17         X18         X19         X20
##  -0.4302169  -0.3327596  -1.0936251   0.1259921  -1.0709751   0.5470251
```

The best model caught only three out of the five zero coefficients.

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

```
errors <- rep(NA, 20)
x_colname <- colnames(x, do.NULL = FALSE, prefix = "X")
for (i in 1:20) {
  coeff <- coef(subset1, id = i)
  errors[i] <- sqrt(sum((b[x_colname %in% names(coeff)] - coeff[names(coeff) %in% x_colname])^2) + sum(
}
plot(errors, xlab = "Number of variables", ylab = "MSE for estimated and true coefficients", type = "b"
axis(1, at = seq(1, 20, 1))
```



From the above plot, it is seen that the model with 6 variables has the least error. This implies that the model that gives coefficient estimates close to the true parameter values need not necessarily be the model that has the least MSE, i.e., it is not necessarily the best model.