ISLRCh5&6

Sky Liu

2/14/2019

5.7.8

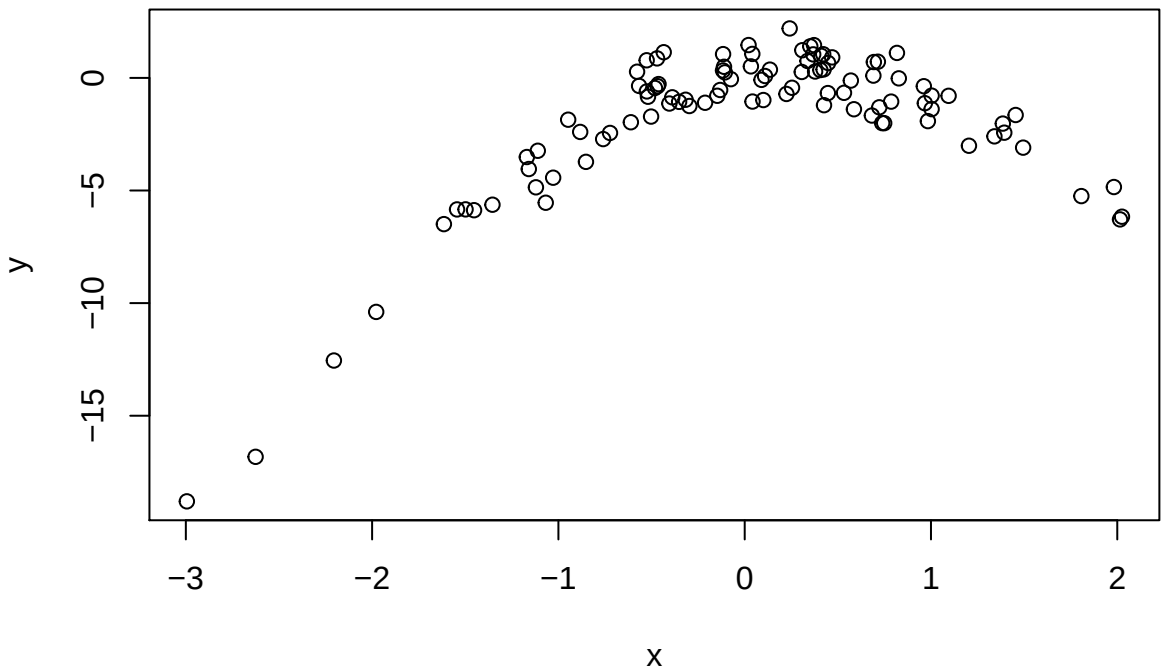
part a

```
set.seed(214)

x <- rnorm(100)
y <- x - 2 * x^2 + rnorm(100)
```

n=100 and p=2. The equation form of the model is $y = x - 2x_i^2 + \epsilon$

part b

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



The simulated data peaks at x = 0.25, which is also the maximum of the first derivative of the model equation

part c

```
set.seed(214)
DF <- data.frame(y = y, x = x)
m1 <- glm(y ~ x, data = DF)
```

```
cv.err <- cv.glm(DF, m1)
e1<-cv.err$delta[1]
```

```
m2 <- glm(y ~ x + I(x^2), data = DF)
cv.err <- cv.glm(DF, m2)
e2<-cv.err$delta[1]
```

```
m3 <- glm(y ~ x + I(x^2) + I(x^3), data = DF)
cv.err <- cv.glm(DF, m3)
e3<-cv.err$delta[1]
```

```
m4 <- glm(y ~ x + I(x^2) + I(x^3) + I(x^4), data = DF)
cv.err <- cv.glm(DF, m4)
e4<-cv.err$delta[1]
e1
```

```
## [1] 9.570498
```

```
e2
```

```
## [1] 0.9137375
```

```
e3
```

```
## [1] 0.9505579
```

```
e4
```

```
## [1] 0.839804
```

part d

```
set.seed(211)

DF <- data.frame(y = y, x = x)
m1 <- glm(y ~ x, data = DF)
cv.err <- cv.glm(DF, m1)
e1<-cv.err$delta[1]
```

```
m2 <- glm(y ~ x + I(x^2), data = DF)
cv.err <- cv.glm(DF, m2)
e2<-cv.err$delta[1]
```

```
m3 <- glm(y ~ x + I(x^2) + I(x^3), data = DF)
cv.err <- cv.glm(DF, m3)
e3<-cv.err$delta[1]
```

```
m4 <- glm(y ~ x + I(x^2) + I(x^3) + I(x^4), data = DF)
cv.err <- cv.glm(DF, m4)
e4<-cv.err$delta[1]
e1
```

```
## [1] 9.570498
```

```
e2
```

```
## [1] 0.9137375
```

```
e3
```

```
## [1] 0.9505579
```

```
e4
```

```
## [1] 0.839804
```

The results are the same becasue it evaluates only one single observation.

part e

The second model has the smallest error as expected because the quadratic polynomial matchs with the original data.

```
summary(m2)
```

```
##
## Call:
## glm(formula = y ~ x + I(x^2), data = DF)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.26425  -0.68193   0.02306   0.55706   2.00606
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.06233    0.11085   0.562    0.575
## x            1.00760    0.09946  10.131   <2e-16 ***
## I(x^2)      -1.98615    0.06589 -30.142   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.8520698)
##
##     Null deviance: 1155.507  on 99  degrees of freedom
## Residual deviance:   82.651  on 97  degrees of freedom
## AIC: 272.73
##
## Number of Fisher Scoring iterations: 2
```

Both the linear and quadratic terms has small p value.

6.8.2

part a - lasso

iii. Less flexible and hence will give improved prediction accuracy when its increase in bias is less than its decrease in variance.

Lasso constraints non-zero coefficient estimates to zeros, that means an increase in bias (due to the reduced number of coefficient estimates) and a decrease in variance. If the increase in bias is less than the decrease in variance , the prediction accuracy will be improved by using lasso comparing to least squares method.

part b - ridge regression

iii. Less flexible and hence will give improved prediction accuracy when its increase in bias is less than its decrease in variance.

Same as part a

part c - non-linear methods

ii. More flexible and hence will give improved prediction accuracy when its increase in variance is less than its decrease in bias.

6.8.10

part a

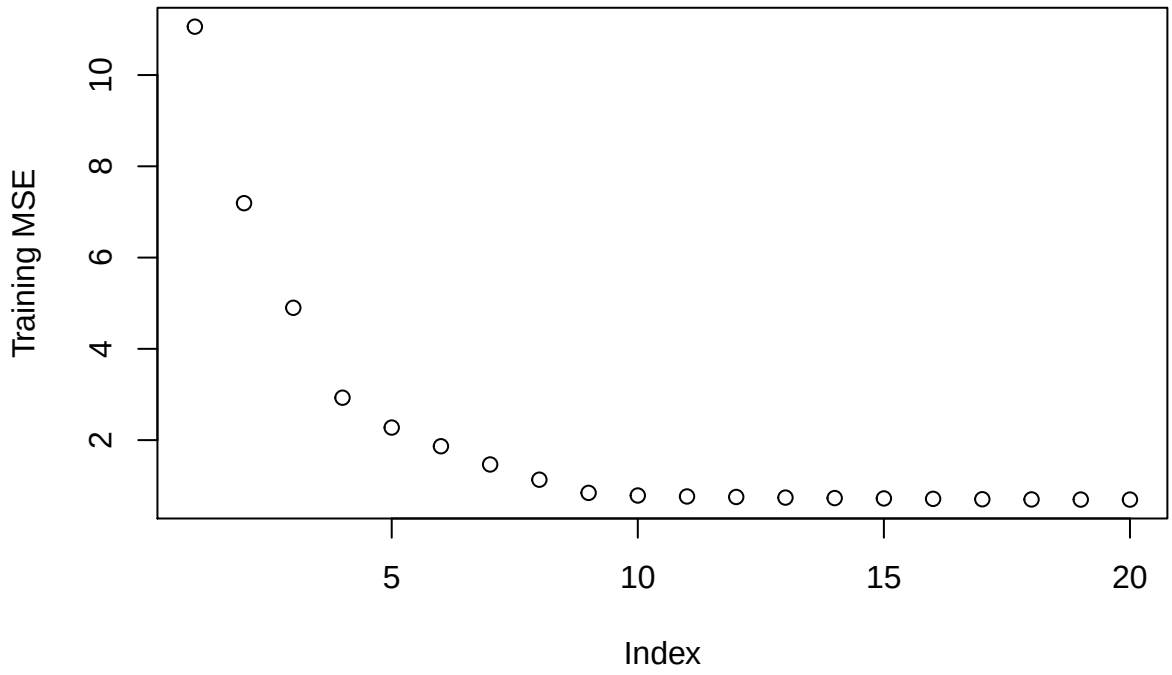
```
set.seed(192)
n = 1000
p = 20
x = matrix(rnorm(n * p), n, p)
betas = rnorm(p)
zerob = c(3,4,7,8,11,12,13,19,20)
betas[zerob] = 0

e = rnorm(p)
y = x %*% betas + e
```

part b

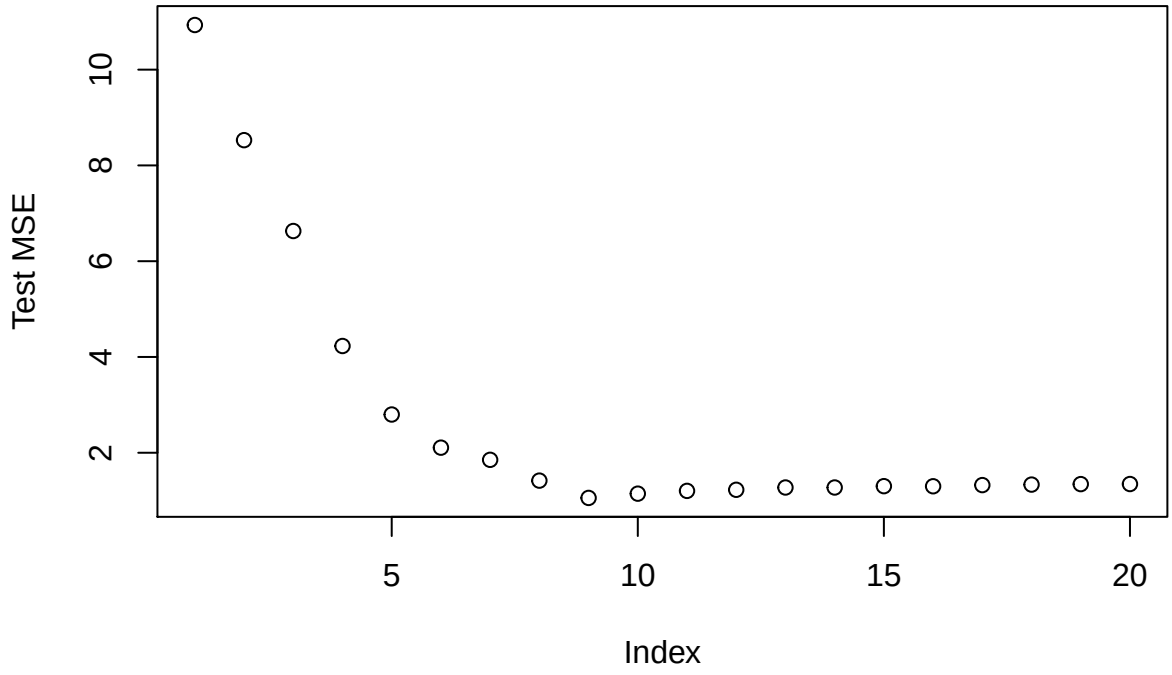
```
train = sample(seq(1000), 100, replace = FALSE)
y_train = y[train, ]
y_test = y[-train, ]
x_train = x[train, ]
x_test = x[-train, ]
```

part c

```
m_full = regsubsets(y ~ ., data = data.frame(x = x_train, y = y_train),nvmax = p)
val_errors = rep(NA, p)
x_cols = colnames(x, do.NULL = FALSE, prefix = "x.")
for (i in 1:p) {
    coei = coef(m_full, id = i)
    pred = as.matrix(x_train[, x_cols %in% names(coei)]) %*% coei[names(coei) %in%
        x_cols]
    val_errors[i] = mean((y_train - pred)^2)
}
plot(val_errors, ylab = "Training MSE")
```




```
val_errors1 = rep(NA, p)
for (i in 1:p) {
    coe = coef(m_full, id = i)
    pred = as.matrix(x_test[, x_cols %in% names(coe)]) %*% coe[names(coe) %in%
        x_cols]
    val_errors1[i] = mean((y_test - pred)^2)
}
plot(val_errors1, ylab = "Test MSE")
```



part e

```
which.min(val_errors1)
```

```
## [1] 9
```

Model with 9+1 terms has the smallest MSE, but from the plot we can see that, from the 9th to the 20th, the MSE are about the same, while the rest have large variance in MSE. Thus, this model is not a very good fit, maybe using more data to train the model will provide a better result.

part f

model 9 coefficients:

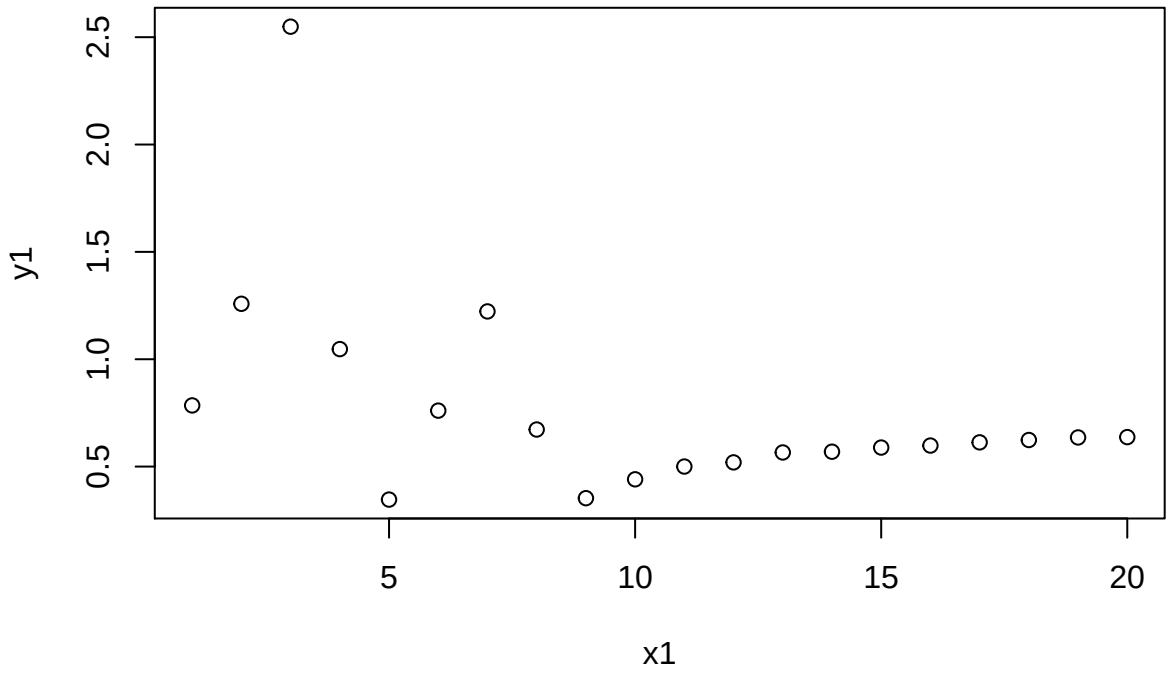
```
coef(m_full,id=9)
```

```
## (Intercept)         x.1         x.2         x.5         x.9        x.10
## -0.04108699  1.60499565 -1.36143010 -0.72375859  0.61009091  2.56219700
##        x.14        x.16        x.17        x.18
## -1.47997562 -0.77205853  0.54582835  1.04441198
```

we set 3,4,7,8,11,12,13,19,20 as zero at the begining. The 6th, 15 th does not match, others are fine.

part g

```
val_errors2 = rep(NA, p)
x1 = rep(NA, p)
y1 = rep(NA, p)
for (i in 1:p) {
    coe = coef(m_full, id = i)
    x1[i] = length(coe) - 1
    y1[i] = sqrt(sum((betas[x_cols %in% names(coe)] - coe[names(coe) %in% x_cols])^2) +
        sum(betas[!(x_cols %in% names(coe))])^2)
}
plot(x = x1, y = y1)
```




```
which.min(y1)
```

```
## [1] 5
```

The model with 5+1 terms is the best model.Still the increase of parameters leads to the increase of variance, that means larger gap between true betas and model betas.