20152410 배형준 Data Mining HW1

Contents

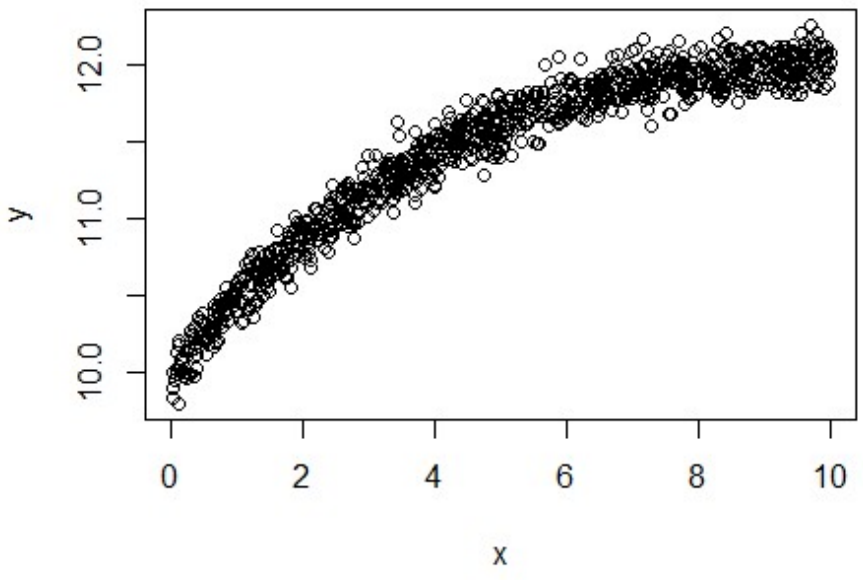
Use the attached “data2.txt”. Use set.seed(1) to make a test set of 300 observations from the data. Use the remaining 700 observations and apply (1) validation set, (2) LOOCV, and (3) 10-fold cross validation approaches setting the seed with your university ID to answer the following subproblems:

load dataset and split train and test

```
student = 20152410
dataset = read.csv('./data2.txt', header=TRUE, sep=' ')
head(dataset)
```

```
##          y        x
## 1 11.06616 2.655087
## 2 11.33982 3.721239
## 3 11.60248 5.728534
## 4 11.93088 9.082078
## 5 10.88337 2.016819
## 6 12.02761 8.983897
```

```
# reorder variables
data = data.frame(x = dataset[, 2], y = dataset[, 1])

# 데이터의 분포 시각적으로 확인
plot(data[, 1], data[, 2], xlab='x', ylab='y')
```



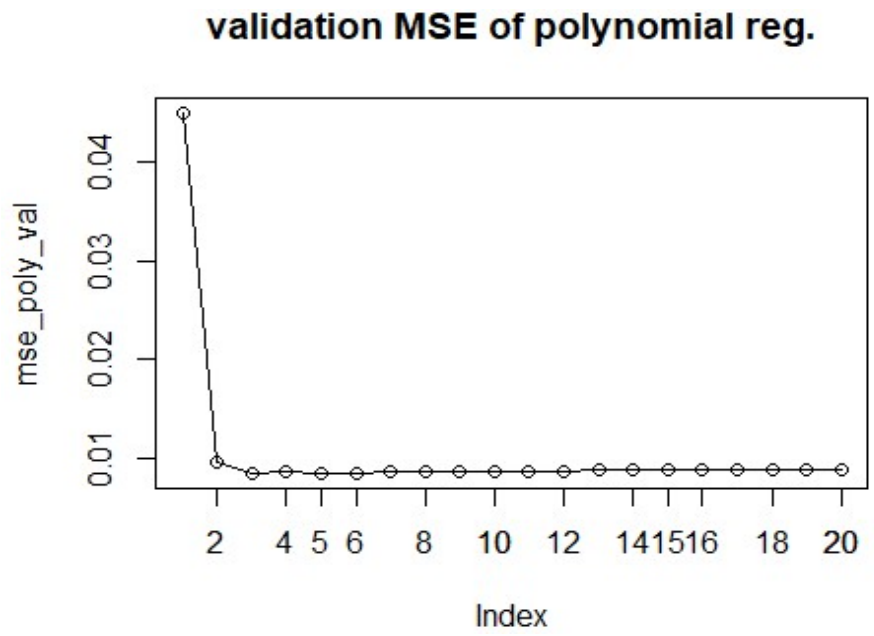
```r
# split trainset and testset
set.seed(1)
n = dim(data)[1]
train_size = 0.7
train_index = sample(1:n, n*train_size, replace=FALSE)
trainset = data[train_index, ]
testset = data[-train_index, ]

# for fitted curve in graph
x = seq(0, 10, 0.001)
x_linspace= data.frame(x = x)
```

(a) Polynomial regression

Determine the best polynomial regression model to predict y by x. Draw a scatter plot with the fitted curve. Report the test MSE of your final model using the test set.

(1) validation set

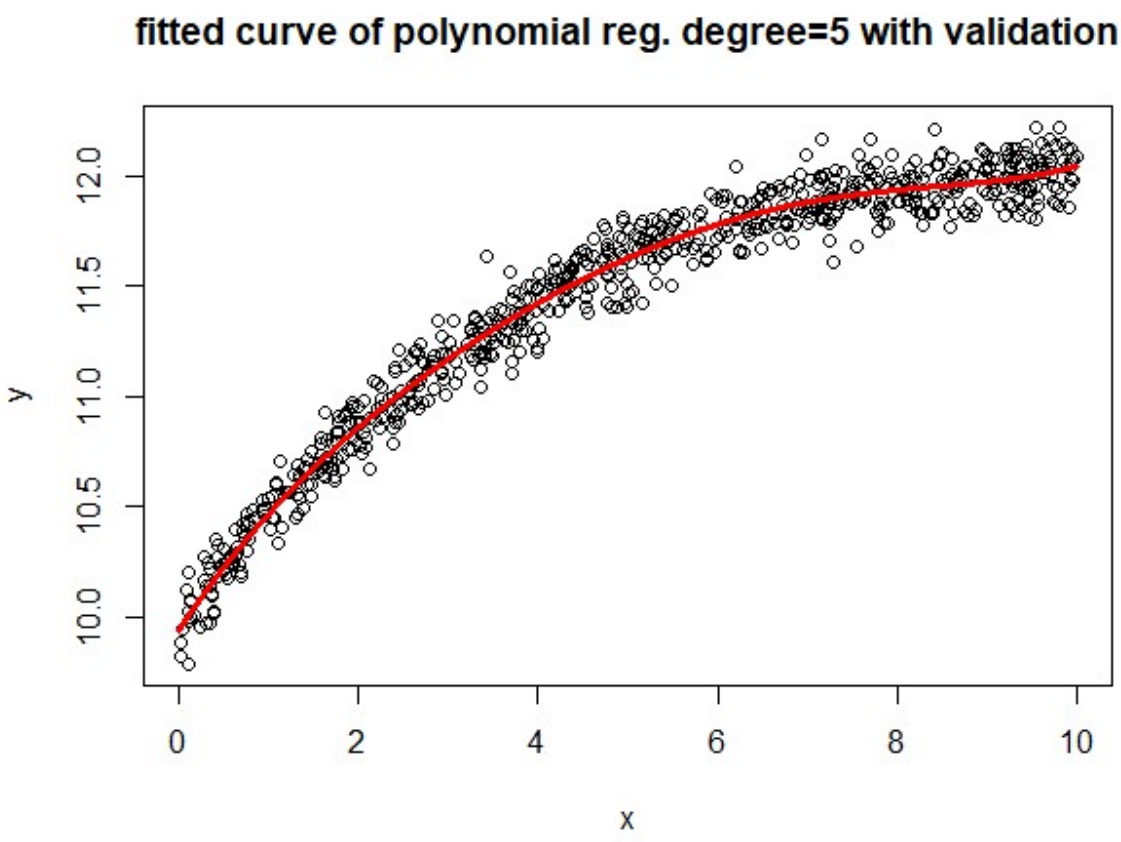
```
# make validation set
set.seed(student)
m = dim(trainset)[1]
val_size = 0.3
val_index = sample(1:m, m*val_size, replace=FALSE)
train = trainset[-val_index, ]
val = trainset[val_index, ]

mse_poly_val = c()


# model learning
for (i in 1:20) {
  model_poly = lm(y ~ poly(x, i), data=train)
  predict_value = predict(model_poly, newdata = val)
  mse = mean((val[, 'y'] - predict_value)^2)

  mse_poly_val[i] = mse
}

# 차수가 2 부터 validation mse 가 급격하게 감소하는 것을 확인할 수 있다.
plot(mse_poly_val, type='o', xlim=c(1, 20), main='validation MSE of polynomial reg.
')
axis(side=1, at=seq(0, 20, 2))
```



```r
# calculate test mse
degree = which.min(mse_poly_val)
model_poly = lm(y ~ poly(x, degree), data=trainset)
predict_value = predict(model_poly, newdata=testset)
test_mse_poly_val = mean((testset[, 'y'] - predict_value)^2)
cat('validation method 를 이용해 구한 polynomial regression 의 차수는',
    degree, '이고 test MSE 는', test_mse_poly_val, '이다.')
```

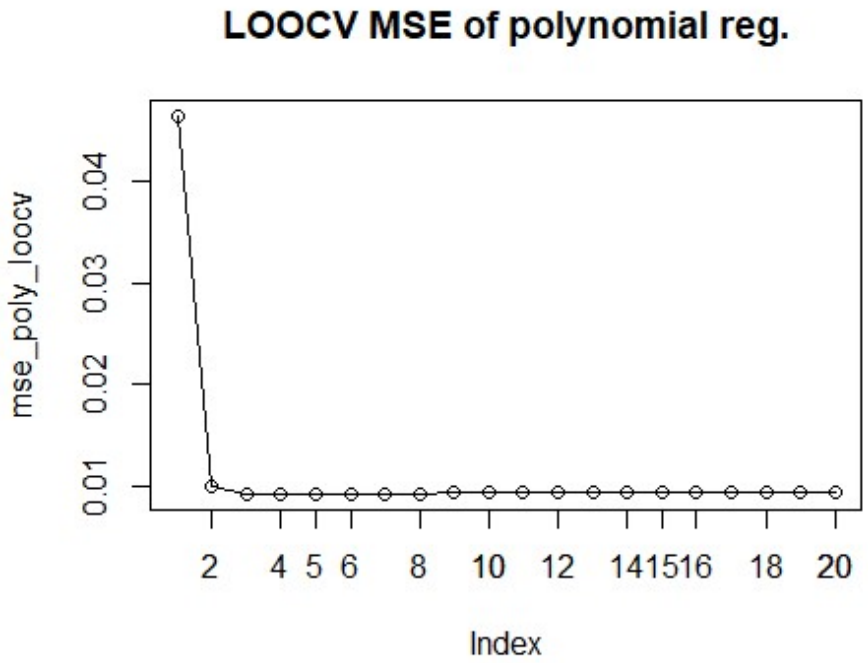
```
## validation method 를 이용해 구한 polynomial regression 의 차수는 5 이고 test MSE 는
0.01142621 이다.
```

```r
# fitted curve
plot(trainset[, 'x'], trainset[, 'y'], xlab='x', ylab='y',
     main='fitted curve of polynomial reg. degree=5 with validation')
points(x, predict(model_poly, newdata=x_linspace), col='red', type='l', lwd=3)
```



(2) LOOCV

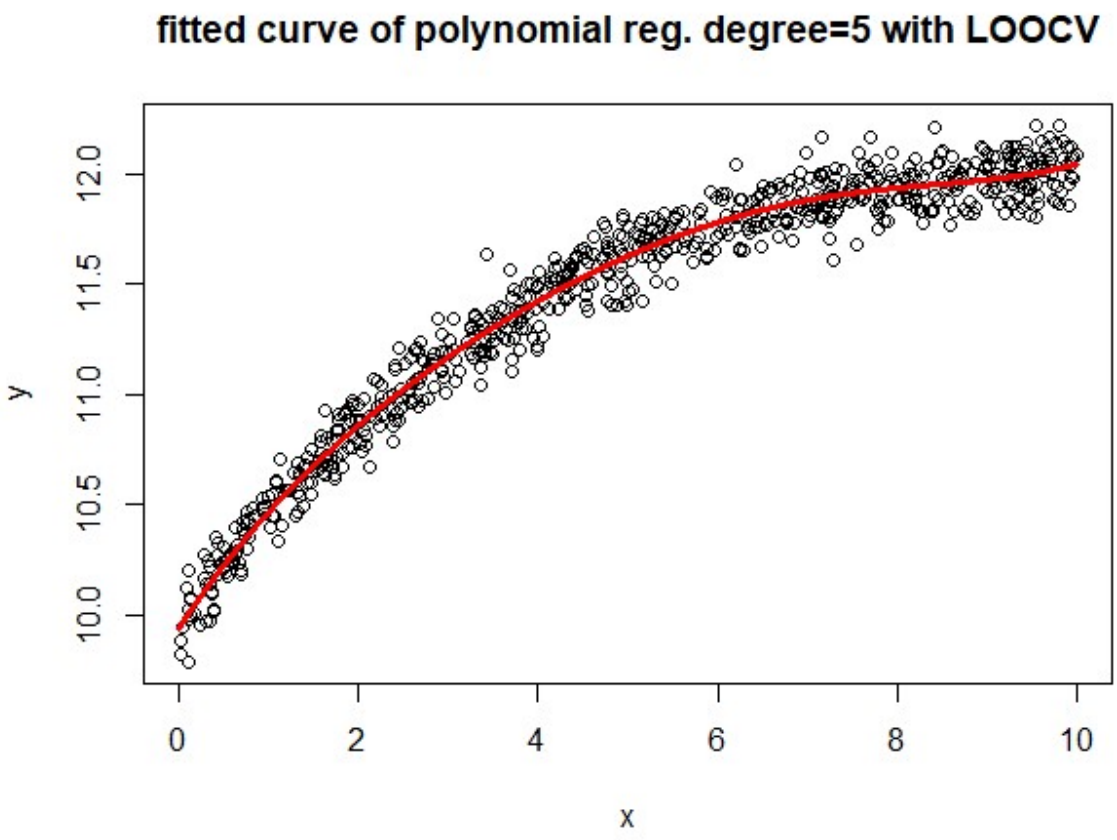
```
mse_poly_loocv = c()

# model learning
for (i in 1:20) {
  mse_loocv = 0

  for (j in 1:m) {
    temp_train = trainset[-j, ]
    temp_val = trainset[j, ]
    model_poly = lm(y ~ poly(x, i), data=temp_train)
    predict_value = predict(model_poly, newdata=temp_val)
    mse = (temp_val[, 'y'] - predict_value)^2

    mse_loocv = mse_loocv + mse
  }
  mse_poly_loocv[i] = mse_loocv / m
}

# 차수가 2 부터 validation mse 가 급격하게 감소하는 것을 확인할 수 있다.
plot(mse_poly_loocv, type='o', xlim=c(1, 20), main='LOOCV MSE of polynomial reg.')
axis(side=1, at=seq(0, 20, 2))
```



```r
# calculate test mse
degree = which.min(mse_poly_loocv)
model_poly = lm(y ~ poly(x, degree), data=trainset)
predict_value = predict(model_poly, newdata=testset)
test_mse_poly_loocv = mean((testset[, 'y'] - predict_value)^2)
cat('LOOCV method 를 이용해 구한 polynomial regression 의 차수는',
    degree, '이고 test MSE 는', test_mse_poly_loocv, '이다.')
```

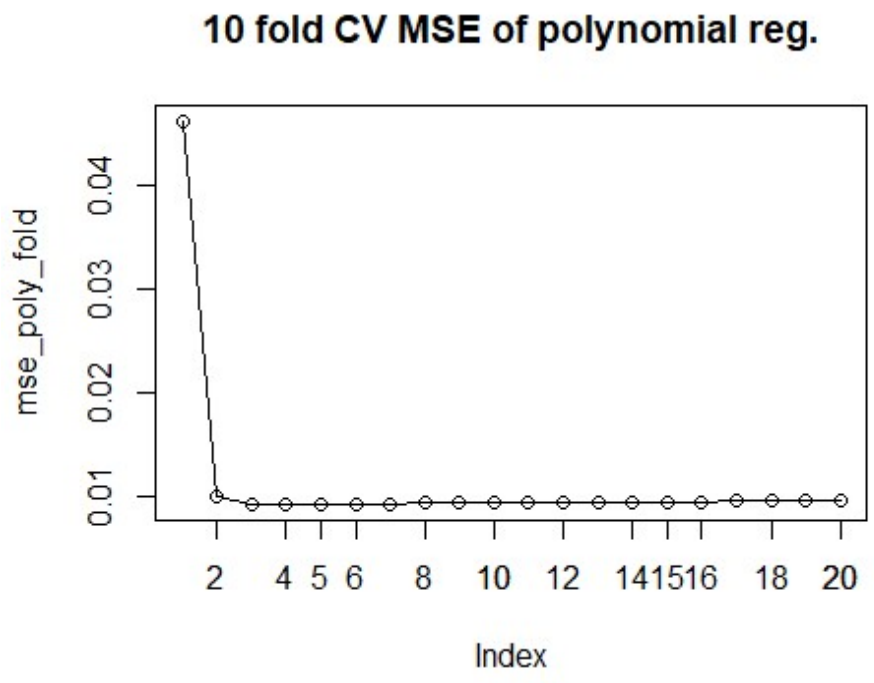
```
## LOOCV method 를 이용해 구한 polynomial regression 의 차수는 5 이고 test MSE 는 0.011
42621 이다.
```

```r
# fitted curve
plot(trainset[, 'x'], trainset[, 'y'], xlab='x', ylab='y',
     main='fitted curve of polynomial reg. degree=5 with LOOCV')
points(x, predict(model_poly, newdata=x_linspace), col='red', type='l', lwd=3)
```



(3) 10 fold cv

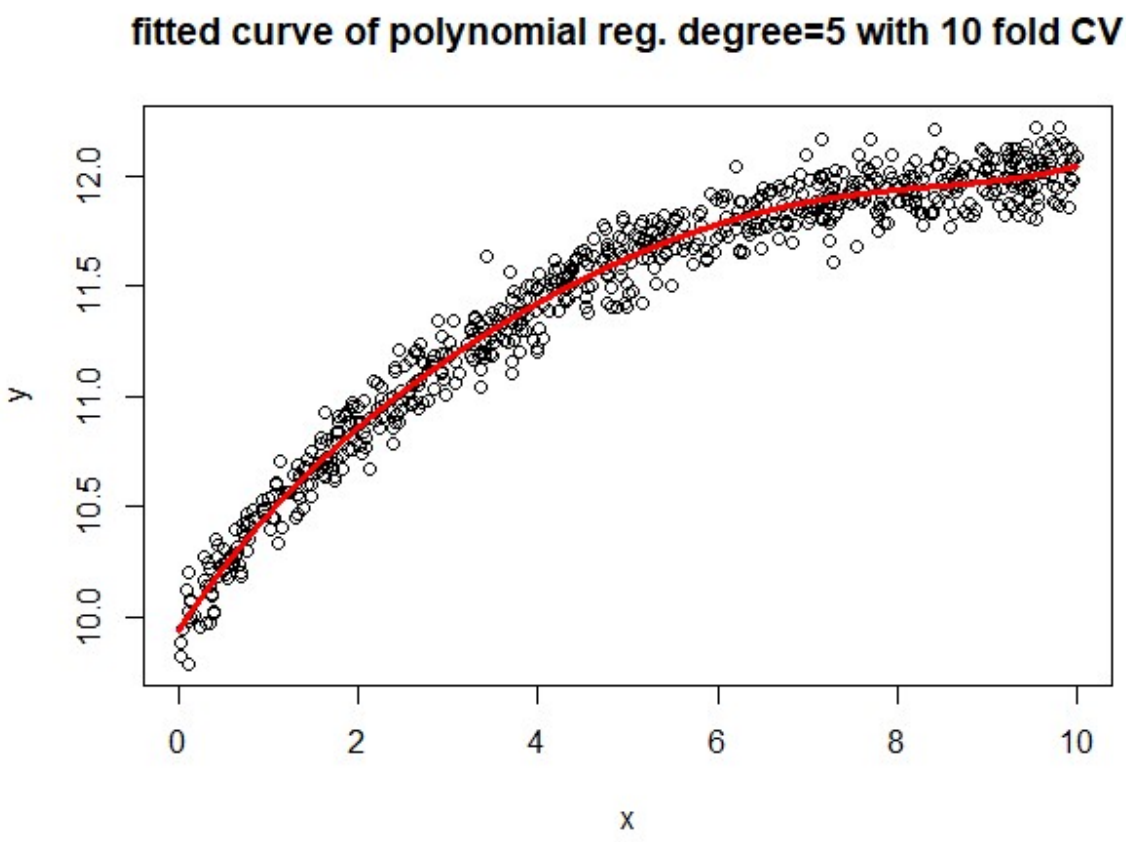
```
set.seed(student)
fold = 10
fold_index = sample(1:fold, m, replace=TRUE)

mse_poly_fold = c()

# model learning
for (i in 1:20) {
  mse_fold = 0

  for (j in 1:fold) {
    temp_train = trainset[fold_index!=j, ]
    temp_val = trainset[fold_index==j, ]
    model_poly = lm(y ~ poly(x, i), data=temp_train)
    predict_value = predict(model_poly, newdata=temp_val)
    mse = mean((temp_val[, 'y'] - predict_value)^2)

    mse_fold = mse_fold + mse
  }
  mse_poly_fold[i] = mse_fold / fold
}

# 차수가 2 부터 validation mse 가 급격하게 감소하는 것을 확인할 수 있다.
plot(mse_poly_fold, type='o', xlim=c(1, 20), main='10 fold CV MSE of polynomial re
g.')
axis(side=1, at=seq(0, 20, 2))
```



```r
# calculate test mse
degree = which.min(mse_poly_fold)
model_poly = lm(y ~ poly(x, degree), data=trainset)
predict_value = predict(model_poly, newdata=testset)
test_mse_poly_10foldcv = mean((testset[, 'y'] - predict_value)^2)
cat('10 fold CV method 를 이용해 구한 polynomial regression 의 차수는',
    degree, '이고 test MSE 는', test_mse_poly_10foldcv, '이다.')
```

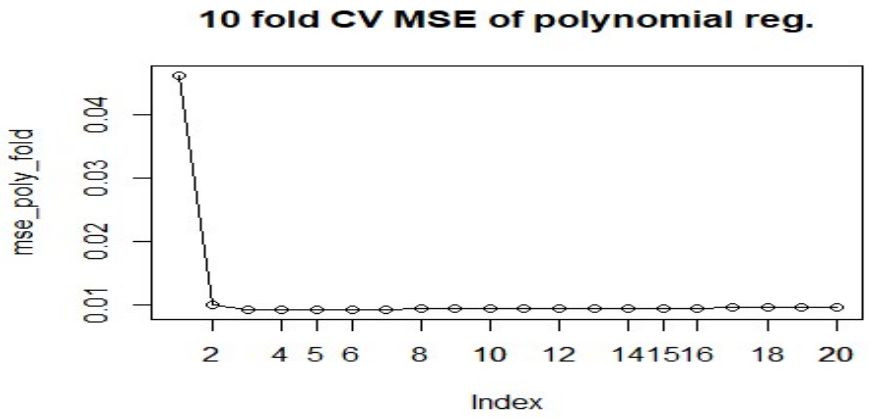
```
## 10 fold CV method 를 이용해 구한 polynomial regression 의 차수는 5 이고 test MSE 는
0.01142621 이다.
```

```r
# fitted curve
plot(trainset[, 'x'], trainset[, 'y'], xlab='x', ylab='y',
     main='fitted curve of polynomial reg. degree=5 with 10 fold CV')
points(x, predict(model_poly, newdata=x_linspace), col='red', type='l', lwd=3)
```



(4) Conclusion

validation, LOOCV, 10 fold CV 방법으로 Polynomial regression 을 학습하였다. x 와 y 의 scatter plot 을 확인한 것을 기반으로 모델의 hyperparameter 인 polynomial degree 를 1 부터 20 사이에서 선정하는 것으로 학습을 설계하였다. validation MSE 를 기준으로 모델 학습을 평가했을 때 validation, LOOCV, 10 fold CV 3 개의 방법 모두 최적 degree 로 5 를 선정하였다. degree 가 같기 때문에 최종 학습 모델이 다 같아졌고 test MSE 0.01142621 를 얻을 수 있었다.



10 fold CV MSE 그래프를 다시 살펴보도록 하겠다. Degree=1 일때는 val MSE 가 큰 것을 확인할 수 있는데 이는 MSE 를 구성하고 있는 요소 중 bias 의 값이 큰 것으로 볼 수 있다. 데이터를 더 잘 설명하는 모델을 만들기 위해서는 더 복잡한 모델을 사용할 필요가 있다. (more complex, flexible model) degree 가 2~20 인 부분에서는 MSE 를 구성하고 있는 요소 중 variance 와 bias 모두 작은 값인 것을 확인할 수 있다. 여기에선 학습 설계 상 degree 가 20 보다 큰 부분을 확인하지 않았는데 만약 degree 의 범위를 100 까지 늘려본다면 degree 가 증가할수록 bias 는 0 에 가까워지지만 오히려 variance 가 엄청나게 증가해버려 val MSE 가 다시 상승할 것으로 예상된다.

이 데이터에서는 변수 간의 관계 해석이 중요하지 않아 val MSE 를 가장 작게 만드는 (성능을 최적화하는) degree=5 를 사용했지만 해석이 중요한 경우에는 실제 변수 간 관계가 5 차 다항식인지를 검증해 볼 필요가 있다. 1 차 다항식은 bias 가 너무 크고, 2~5 차 다항식이 성능이 좋으면서 엄청나게 복잡하지는 않은 모델이다. 이런 경우엔 데이터가 없는 x 가 10 보다 큰 경우에 y 가 어떻게 분포할 것인가가 모델을 결정하는 요소라고 생각한다. x 가 10 보다 클 때 y 가 감소한다고 가정하면 2, 4 차 다항식을 고려할 수 있겠고 x 가 10 보다 클 때 y 가 증가한다고 가정하면 3, 5 차 다항식을 고려할 수 있겠다. 2, 3 차 다항식의 성능이 4, 5 차 다항식의 성능에 비해 크게 부족하지 않으므로 y 의 분포를 고려하여 2 차 또는 3 차 다항식 회귀를 선택하는 것이 좋아보인다.

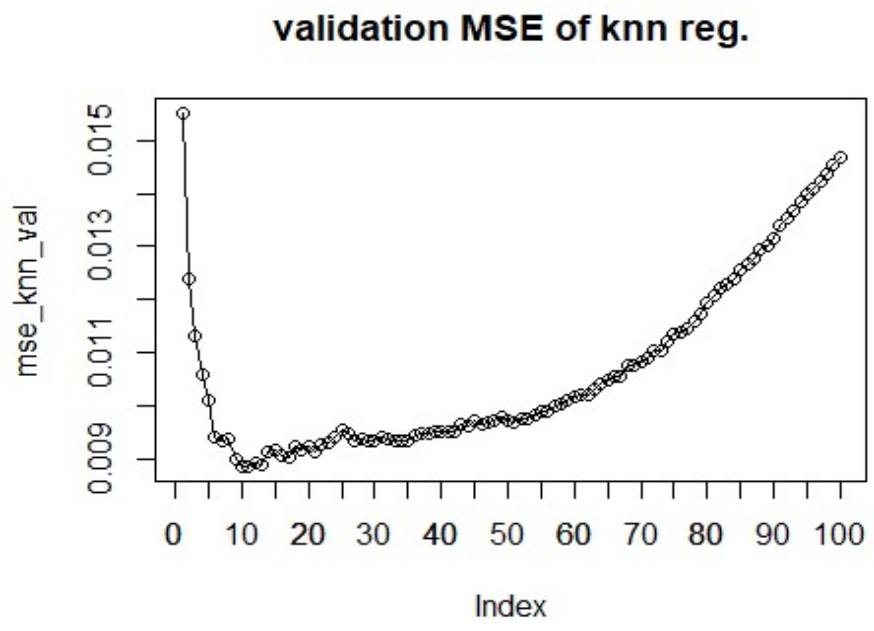
(b) KNN regression

Determine the best knn regression model to predict y by x. Draw a scatter plot with the fitted curve. Report the test MSE of your final model using the test set.

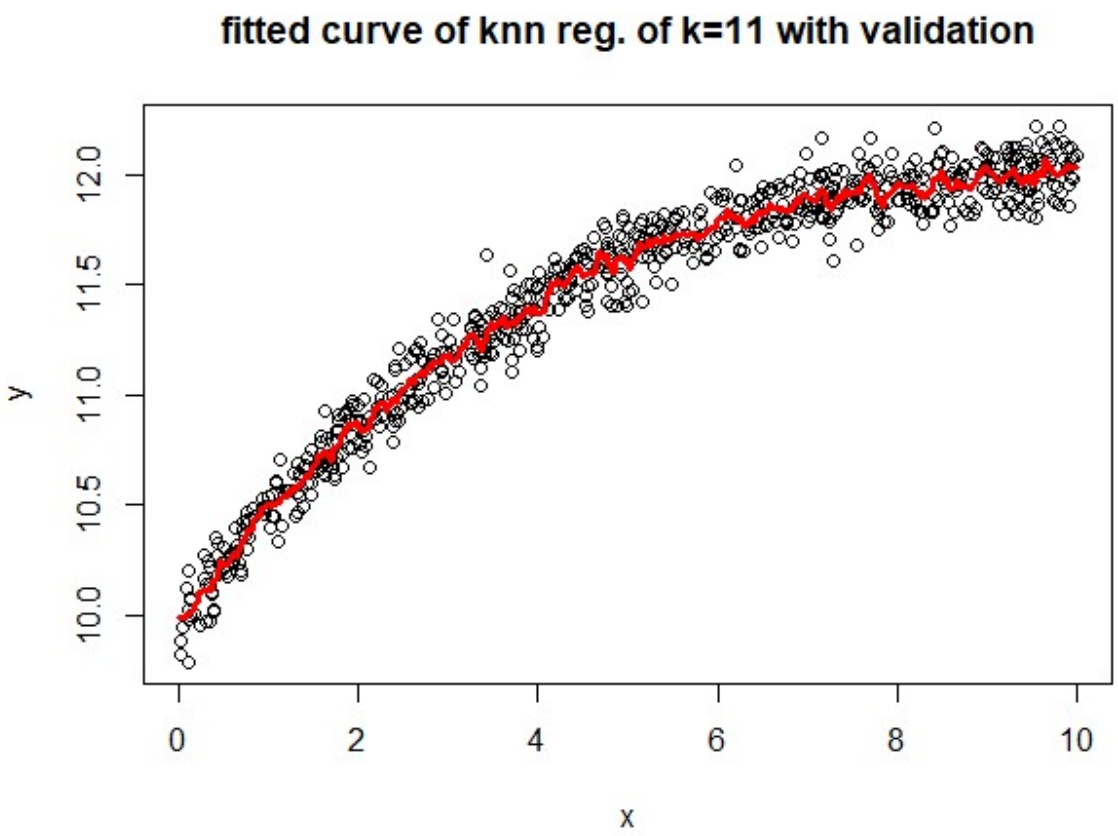
```
library(caret)
```

(1) validation set

```
# make validation set
set.seed(student)
m = dim(trainset)[1]
val_size = 0.3
val_index = sample(1:m, m*val_size, replace=FALSE)
train = trainset[-val_index, ]
val = trainset[val_index, ]

mse_knn_val = c()

# model learning
for (i in 1:100) {
  model_knn = knnreg(y ~ x, k=i, data=train)
  predict_value = predict(model_knn, newdata=val)
  mse = mean((val[, 'y'] - predict_value)^2)

  mse_knn_val[i] = mse
}

# k 가 증가하면서 val mse 가 감소하다가 10 근처부터 다시 증가하는 것을 확인할 수 있다.
plot(mse_knn_val, type='o', xlim=c(1, 100), main='validation MSE of knn reg.')
axis(side=1, at=seq(0, 100, 5))
```

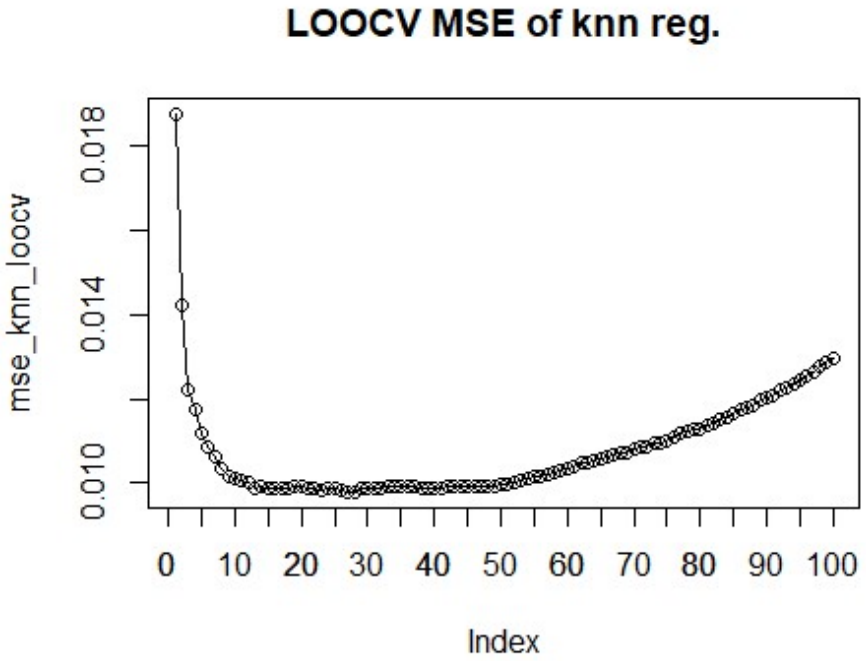


```r
# calculate test mse
k = which.min(mse_knn_val)
model_knn = knnreg(y ~ x, k=k, data=trainset)
predict_value = predict(model_knn, newdata=testset)
test_mse_knn_val = mean((testset[, 'y'] - predict_value)^2)
cat('validation method 를 이용해 구한 k 는',
    k, '이고 test MSE 는', test_mse_knn_val, '이다.')

## validation method 를 이용해 구한 k 는 11 이고 test MSE 는 0.01234136 이다.

# fitted curve
plot(trainset[, 'x'], trainset[, 'y'], xlab='x', ylab='y',
     main='fitted curve of knn reg of k=11 with validation')
points(x, predict(model_knn, newdata=x_linspace), col='red', type='l', lwd=3)
```

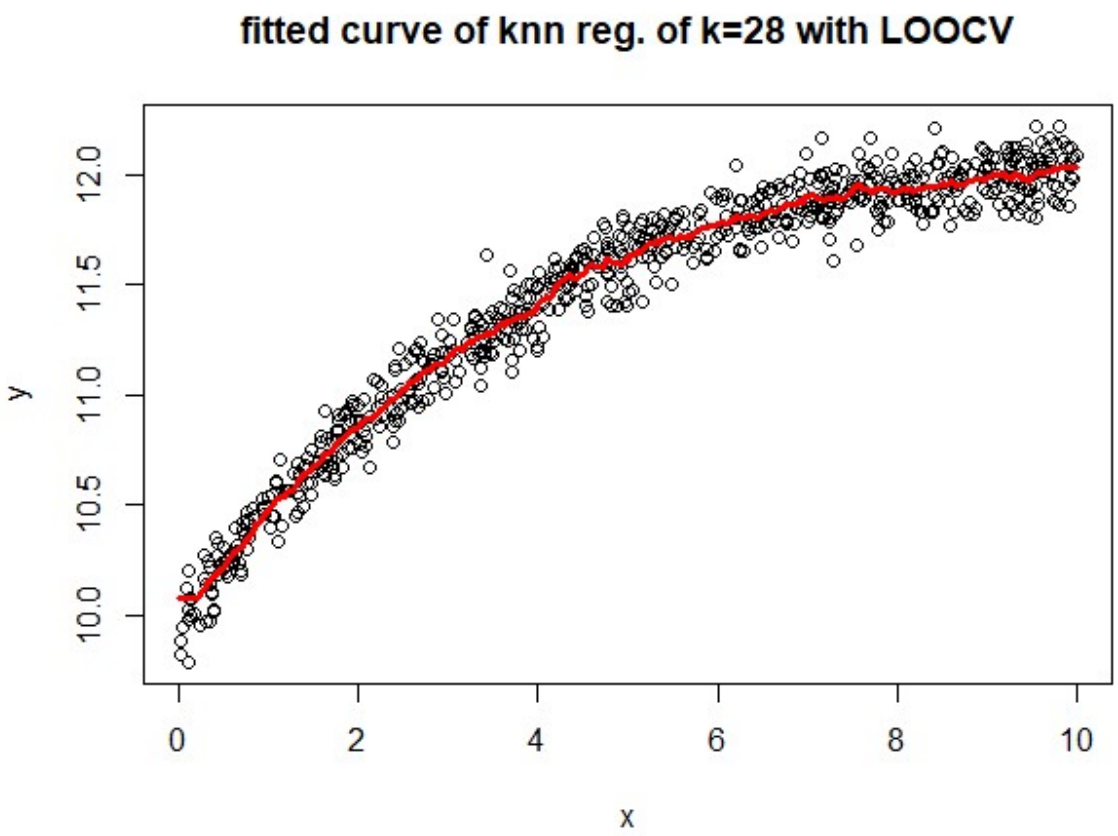


(2) LOOCV

```
mse_knn_loocv = c()

# model learning
for (i in 1:100) {
  mse_loocv = 0

  for (j in 1:m) {
    temp_train = trainset[-j, ]
    temp_val = trainset[j, ]
    model_knn = knnreg(y ~ x, k=i, data=temp_train)
    predict_value = predict(model_knn, newdata=temp_val)
    mse = (temp_val[, 'y'] - predict_value)^2

    mse_loocv = mse_loocv + mse
  }
  mse_knn_loocv[i] = mse_loocv / m
}

# k 가 감소하다가 50 을 지나면서부터 증가하는 것을 확인할 수 있다.
plot(mse_knn_loocv, type='o', xlim=c(1, 100), main='LOOCV MSE of knn reg.')
axis(side=1, at=seq(0, 100, 5))
```

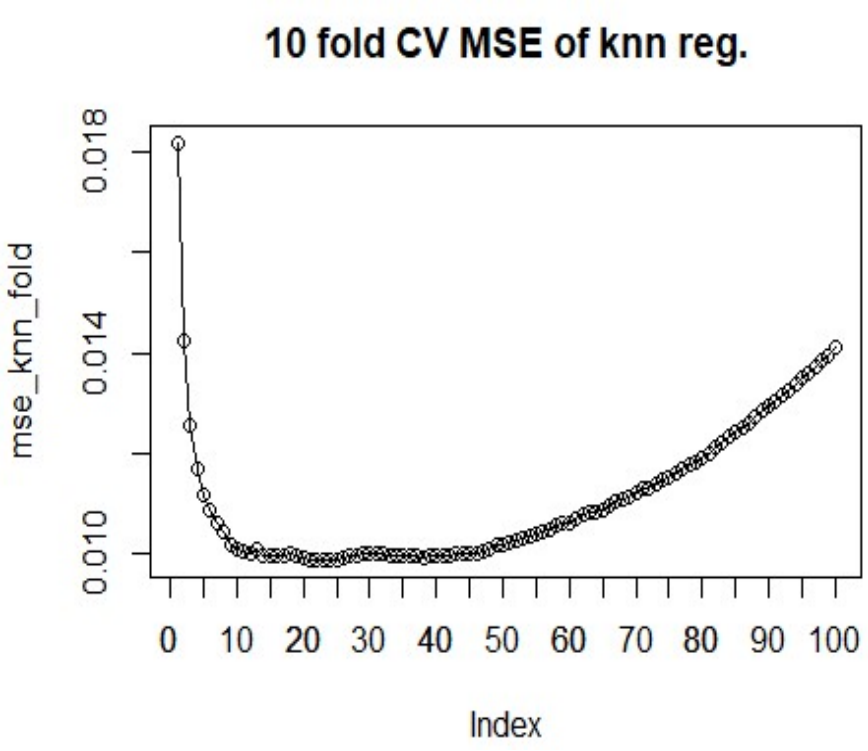


```r
# calculate test mse
k = which.min(mse_knn_loocv)
model_knn = knnreg(y ~ x, k=k, data=trainset)
predict_value = predict(model_knn, newdata=testset)
test_mse_knn_loocv = mean((testset[, 'y'] - predict_value)^2)
cat('validation method 를 이용해 구한 k 는',
    k, '이고 test MSE 는', test_mse_knn_loocv, '이다.')

## validation method 를 이용해 구한 k 는 28 이고 test MSE 는 0.01171555 이다.

# fitted curve
plot(trainset[, 'x'], trainset[, 'y'], xlab='x', ylab='y',
     main='fitted curve of knn reg of k=28 with LOOCV')
points(x, predict(model_knn, newdata=x_linspace), col='red', type='l', lwd=3)
```

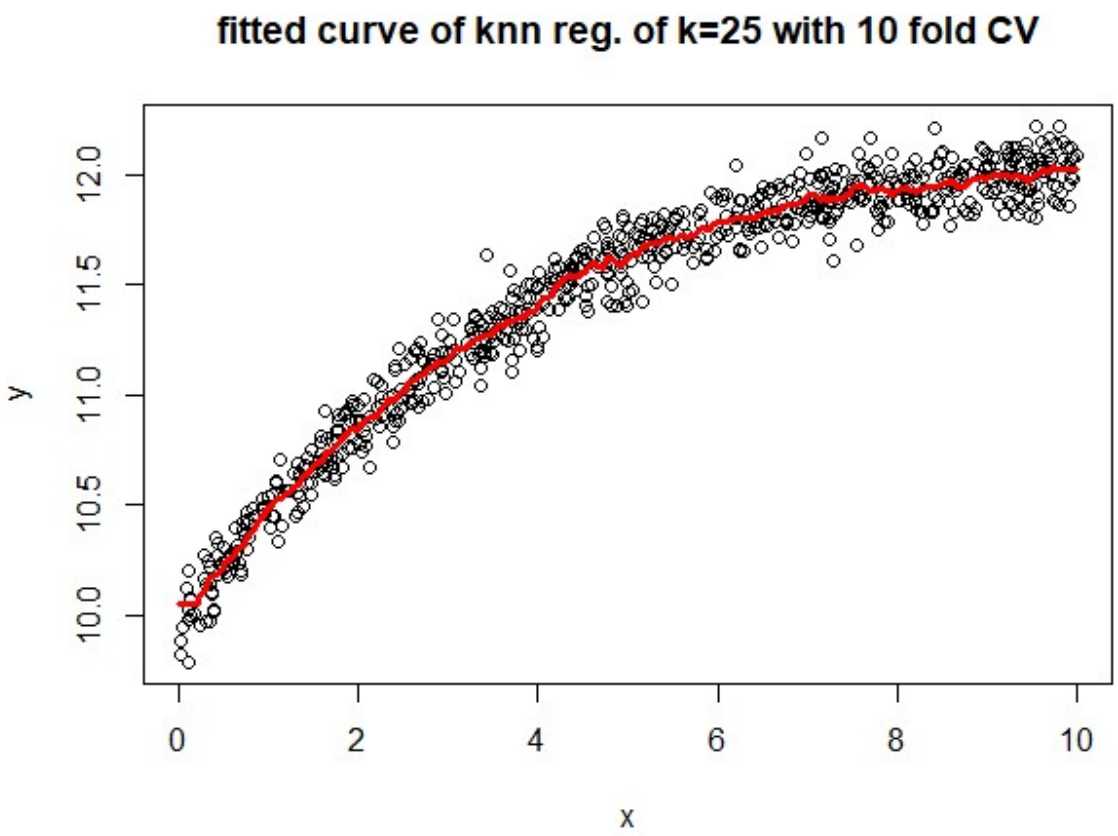


(3) 10 fold cv

```
set.seed(student)
fold = 10
fold_index = sample(1:fold, m, replace=TRUE)

mse_knn_fold = c()

# model learning
for (i in 1:100) {
  mse_fold = 0

  for (j in 1:fold) {
    temp_train = trainset[fold_index!=j, ]
    temp_val = trainset[fold_index==j, ]
    model_knn = knnreg(y ~ x, k=i, data=temp_train)
    predict_value = predict(model_knn, newdata=temp_val)
    mse = mean((temp_val[, 'y'] - predict_value)^2)

    mse_fold = mse_fold + mse
  }
  mse_knn_fold[i] = mse_fold / fold
}

# k 가 감소하다가 50 을 지나면서부터 증가하는 것을 확인할 수 있다.
plot(mse_knn_fold, type='o', xlim=c(1, 100), main='10 fold CV MSE of knn reg.')
axis(side=1, at=seq(0, 100, 5))
```

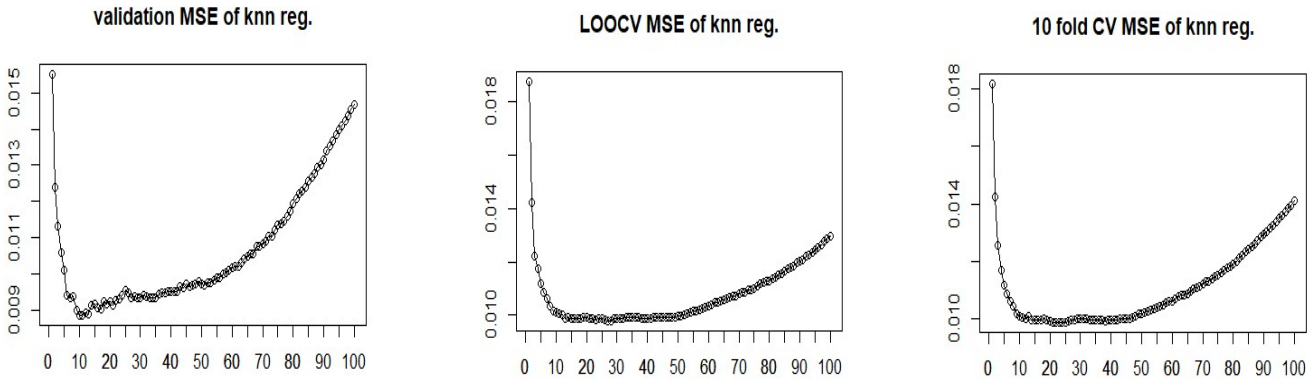


```r
# calculate test mse
k = which.min(mse_knn_fold)
model_knn = knnreg(y ~ x, k=k, data=trainset)
predict_value = predict(model_knn, newdata=testset)
test_mse_knn_10foldcv = mean((testset[, 'y'] - predict_value)^2)
cat('validation method 를 이용해 구한 k 는',
    k, '이고 test MSE 는', test_mse_knn_10foldcv, '이다.')

## validation method 를 이용해 구한 k 는 25 이고 test MSE 는 0.01172869 이다.

# fitted curve
plot(trainset[, 'x'], trainset[, 'y'], xlab='x', ylab='y',
     main='fitted curve of knn reg of k=25 with 10 fold CV')
points(x, predict(model_knn, newdata=x_linspace), col='red', type='l', lwd=3)
```



(4) Conclusion

validation, LOOCV, 10 fold CV 방법으로 KNN regression 을 학습하였다.



학습 곡선을 보면 k 가 1 에서 10 으로 변할 때 val MSE 가 급격하게 감소하는 것을 확인할 수 있다. 차이점은 k 가 적정 범위를 지나 증가할 때 (40~50 을 넘어갈 때) val MSE 가 증가하는 속도가 validation set 을 어떻게 만드냐에 따라 다른 것을 확인할 수 있다. 하나의 validation 만 정의한 경우엔 k 가 50 이상으로 증가할 때 val MSE 가 급격하게 증가하는 모습을 보이는데 이는 validation set 에 포함되는 case 가 어떻게 선정되느냐에 따라 결과가 달라질 수 있는 불안정성을 의미한다. K 가 50 이상으로 증가할 때 LOOCV 의 val MSE 가 가장 천천히 증가하는데 이는 가장 안정된 방법임을 의미한다. 하지만 데이터의 sample size 만큼 모델 학습을 하니까 엄청난 계산량을 필요로 한다. 컴퓨터 환경이 빠르지 않다면 최적 hyperparameter 를 구하기 위해 많은 시간이 소요된다는 단점이 있다. 안정성을 어느정도 확보하면서 LOOCV 보다 계산량을 감소시킨 방법이 fold CV 이다. 10 fold CV 의 val MSE 를 보면 k 가 50 이상 증가할 때 LOOCV 보다 증가하는 속도가 빠르다. 대신 validation MSE 처럼 val MSE 그래프가 불안정한 모습은 보이지 않는다. 또한 LOOCV 처럼 sample size 만큼 학습하지 않고 fold 개수만큼만 학습하므로 과도한 계산량을 요구하지도 않는다. 이 데이터의 경우에도 KNN reg 의 k 가 28, 25 로 비슷한 것으로 보아 최적 hyperparameter 를 구하는데 있어서 fold CV 의 성능이 부족하지 않은 것을 확인할 수 있다.

KNN regression 의 fitted curve 가 polynomial regression 의 curve 보다 구불구불한 것을 확인할 수 있는데 이는 non-parametric model 의 특징이라고 할 수 있다. 구불구불하긴 해도 데이터의 설명력이 좋은 편이라 curve 의 전체적인 추세는 비슷한 것도 확인할 수 있다.

KNN regression 의 경우 변수 간의 관계식을 정의하고 만드는 모델이 아니다보니 성능이 뒷받침되더라도 변수 간의 관계를 해석하기가 힘들다. 만약 이 데이터에서 변수 간 해석을 필요로 한다면 test MSE 성능이 비슷한 polynomial regression 과 KNN regression 중 parametric model 인 polynomial regression 을 사용할 것을 권장한다.