wa_01

gaspard gaches

2025-02-10

Statistical Learning Weekly Assignment 1

Below, we create three functions: the first one generates n data points following the function f(x)=8*sin(x) + e~N(0,1), the secone one trains and tests the data with a polynomial regression model and also returns the test MSE, and the third one computes the second function for 2 different polynomial degrees.

```r
sineFunction <- function(n) {
  x <- runif(n, min=-3, max=3)
  y <- 8 * sin(x)+ rnorm(n)
  return(data.frame(x,y))
}

trainTest <- function(train_set, test_set, deg) {
  fit <- lm(y ~ poly(x, degree = deg), data = train_set)
  test_pred <- predict(fit, newdata=test_set)
  test_err <- mean((test_set$y - test_pred)^2)

  return(list(
    "data"=data.frame(
      x=test_set$x,
      y=test_set$y,
      yhat=test_pred
    ),
    "error"=test_err
    )
  )
}

calcTestError <- function(train_size) {
  train_set <- sineFunction(train_size)
  test_set <- sineFunction(10000)

  poly3 <- trainTest(train_set, test_set, 3)
  poly15 <- trainTest(train_set, test_set, 15)

  return(
    list(
      "data"=list(poly3$data, poly15$data),
      "errors"=list(poly3$error, poly15$error)
    )
  )
}

train_50 <- calcTestError(50)
train_10k <- calcTestError(10^4)

tibble(
  degree=c("degree 3", "degree 15"),
  "n=50"=as.numeric(train_50$errors),
  "n=10^4"=as.numeric(train_10k$errors)
)
```

```
## # A tibble: 2 x 3
##   degree    `n=50` `n=10^4`
##   <chr>      <dbl>    <dbl>
## 1 degree 3    1.22     1.19
## 2 degree 15   1.36     1.01
```

The table above contains the (test set) mean squared error values for both polynomials (degree 3 and 15), first trained on 50 observations only, and then on 10,000 observations, and tested on 10,000 unseen x values in both cases. I decided to also plot the test data for these four scenarios below, using the patchwork package, because a picture is better than a thousand words, and to put in perspective the numbers in the table above: plots:

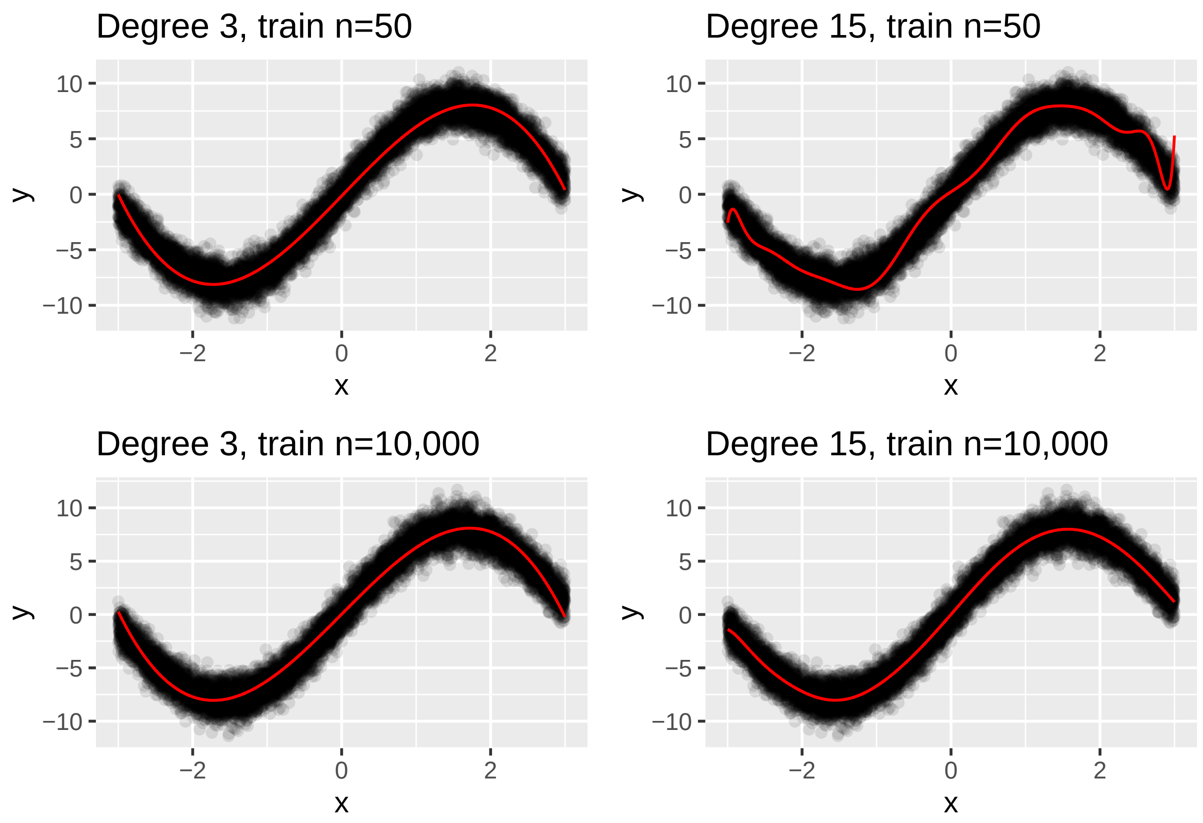
```
p1 <- ggplot(train_50$data[[1]], aes(x=x, y=y)) +
  geom_point(alpha=0.1) +
  geom_line(aes(y=yhat), color="red") +
  ggtitle("Degree 3, train n=50")

p2 <- ggplot(train_50$data[[2]], aes(x=x, y=y)) +
  geom_point(alpha=0.1) +
  geom_line(aes(y=yhat), color="red") +
  ggtitle("Degree 15, train n=50")

p3 <- ggplot(train_10k$data[[1]], aes(x=x, y=y)) +
  geom_point(alpha=0.1) +
  geom_line(aes(y=yhat), color="red") +
  ggtitle("Degree 3, train n=10,000")

p4 <- ggplot(train_10k$data[[2]], aes(x=x, y=y)) +
  geom_point(alpha=0.1) +
  geom_line(aes(y=yhat), color="red") +
  ggtitle("Degree 15, train n=10,000")

(p1 + p2) / (p3 + p4)
```



Note that the degree 15 polynomial looks much more wiggly than the degree 3 for n=50, which puts in perspective the MSE obtained in the table. This plot tells us that the MSE of the degree 15 is higher because that model is much more flexible and has higher variance.

The best prediction rule

Since we generated the data with a function that we came up with ourselves, we already know that the best prediction function is f(x) = 8sin(x). Let's compute its MSE along with the two other polynomials and compare them. I recreated a train and test set to make sure all three functions are trained and tested on the same data and took n=10,000 since we saw in the 2x2 table above that the MSE is lower in that case for both polyomials:

```r
train_set <- sineFunction(10000)
test_set <- sineFunction(10000)

poly3 <- trainTest(train_set, test_set, 3)
poly15 <- trainTest(train_set, test_set, 15)

yhat_sine <- 8*sin(test_set$x)
sine_mse <- mean((test_set$y - yhat_sine)^2)

tibble(
  degree=c("degree 3", "degree 15", "sine"),
  "MSE"=c(poly3$error, poly15$error, sine_mse)
)
```

```
## # A tibble: 3 x 2
##   degree      MSE
##   <chr>     <dbl>
## 1 degree 3  1.16
## 2 degree 15 0.988
## 3 sine      0.988
```

Here “sine” corresponds to the f(x) = 8sin(x) function. It’s worth noting that with this sample it barely beats the degree 15 polynomial. The MSE of the sine function is not a surprise because we generated residuals with rnorm() with a standard deviation of 1 (so variance = 1 too) and a mean of 0: we’ve already reduced the reducible error to 0, and only the residual variance remains.

Comparing the polynomial MSEs for n=10,000:

In the slides of week 1 we saw that flexible models have low bias but high variance. But here in this case, with our training set being so large it managed to “cover” the variance pretty well and combined with a low bias, it achieved a lower MSE than the less flexible degree 3 polynomial. I guess this might be part of the reason why large language models with billions of parameters need such huge amounts of training data. ### For n=50: On the other hand when the training test is relatively small, there isn’t enough training data for the model to properly learn and compensate for its high variance which leads to a higher MSE than the other polynomial.