Understanding Bias-Variance trade-off using R

Shrusti Ghela

06-23-2022

The Bias-Variance trade-off is the most basic yet one of the most important (and sometimes misunderstood) concepts in the field of Machine Learning. We have frequently come across phrases like "high bias leads to under-fitting and high variance leads to over-fitting," or "simpler models have high bias and low variance, whereas more complicated or sophisticated models have low bias and high variance." But what do bias and variance really mean, and how do they impact a model's performance and accuracy?

In this article, we will go through the intuitive understanding and mathematical meaning of bias and variance, see the mathematical relation between bias, variance, and performance of a model, and finally, work on a demonstration to see the effects of model complexity on bias and variance using simulation in R.

Before getting started

Let us first discuss the assumptions and notations of this article

- Both bias and variance are statistical concepts that have a variety of uses. However, they will be discussed in this article in terms of an estimator that is trying to fit, explain, or estimate an unknown data distribution.
- There is a data generator $Y = f(X) + \epsilon$ which is generating data (X, Y) (Data can be repeatedly sampled from the generator, resulting in various sample sets)
  - f is some fixed, unknown, true data generating function. (f represents the systematic information that X provides about Y)
  - ϵ is a random error term. It is independent of X and $\mathbb{E}(\epsilon) = 0$ and $var(\epsilon) = \sigma^2$
  - X is the independent variable and Y is the dependent variable.
- We want to estimate f! (using the sample set we got from the generator)
- We have training Data $(X_1, y_1), ..., (X_n, y_n); X_i \in \mathbb{R}^p$
- We have test Observation (X_0, y_0) (not in our training data!)
- We use training data to obtain $\hat{f}$. Hopefully, $Y \approx \hat{f}(X)$

What is bias and variance in the current setting?

Bias of an estimator is the difference between the expected estimate and the true values in the data.

Intuitively, bias is a measure of how close or far is the estimator to the true data generator it is trying to estimate.

$$Bias[\hat{f}(x_0)] = \mathbb{E}[\hat{f}(x_0)] - f(x_0)$$

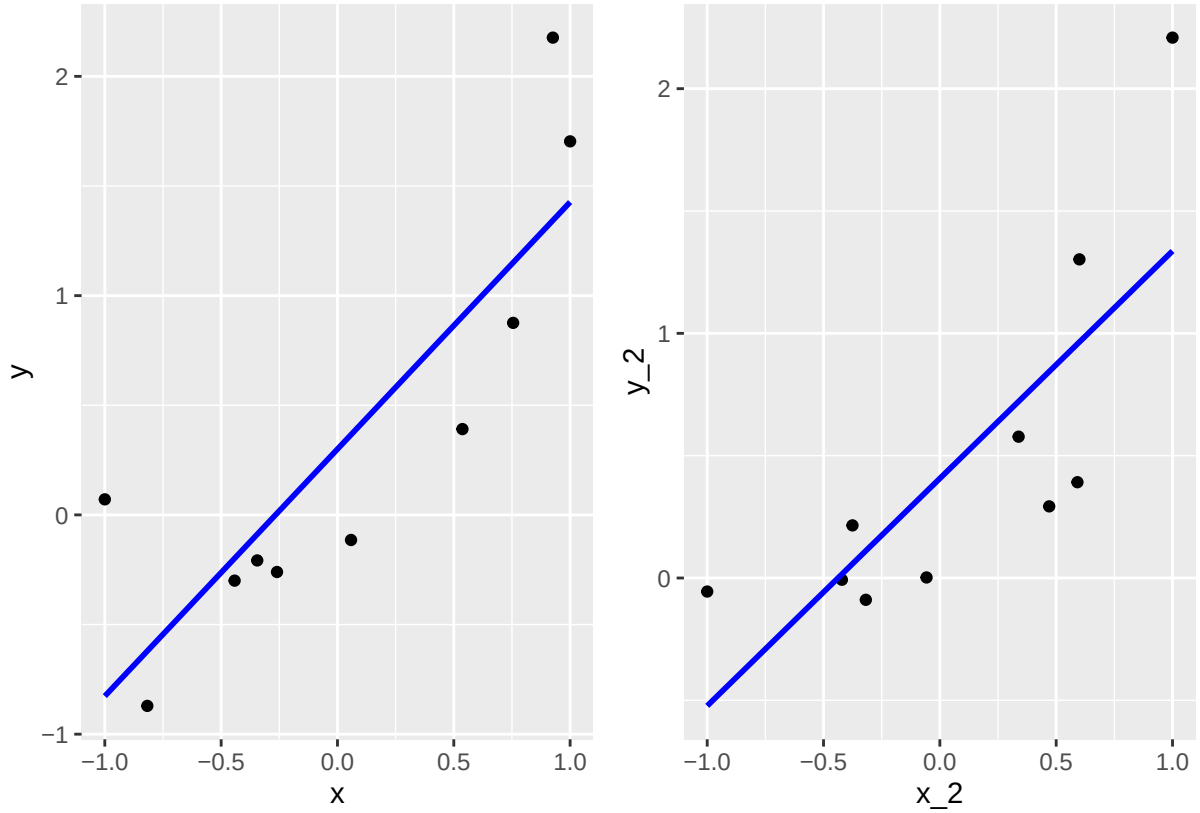
So, considering this, it is only natural to think that an estimator will have *high bias* if it does not change too much when a different sample set of the data is thrown at it. This will typically be the situation when an estimator does not have capacity to properly fit the true data generator. So, simpler models have higher bias as compared to more complex models.

To understand this above concept a little bit better, let's see what happens when we sample from a 3rd-order data generating function, and use a linear estimator to estimate it.

```
library(patchwork)
```

```
## Warning: package 'patchwork' was built under R version 3.6.2
```

```
plot1 + plot2
```



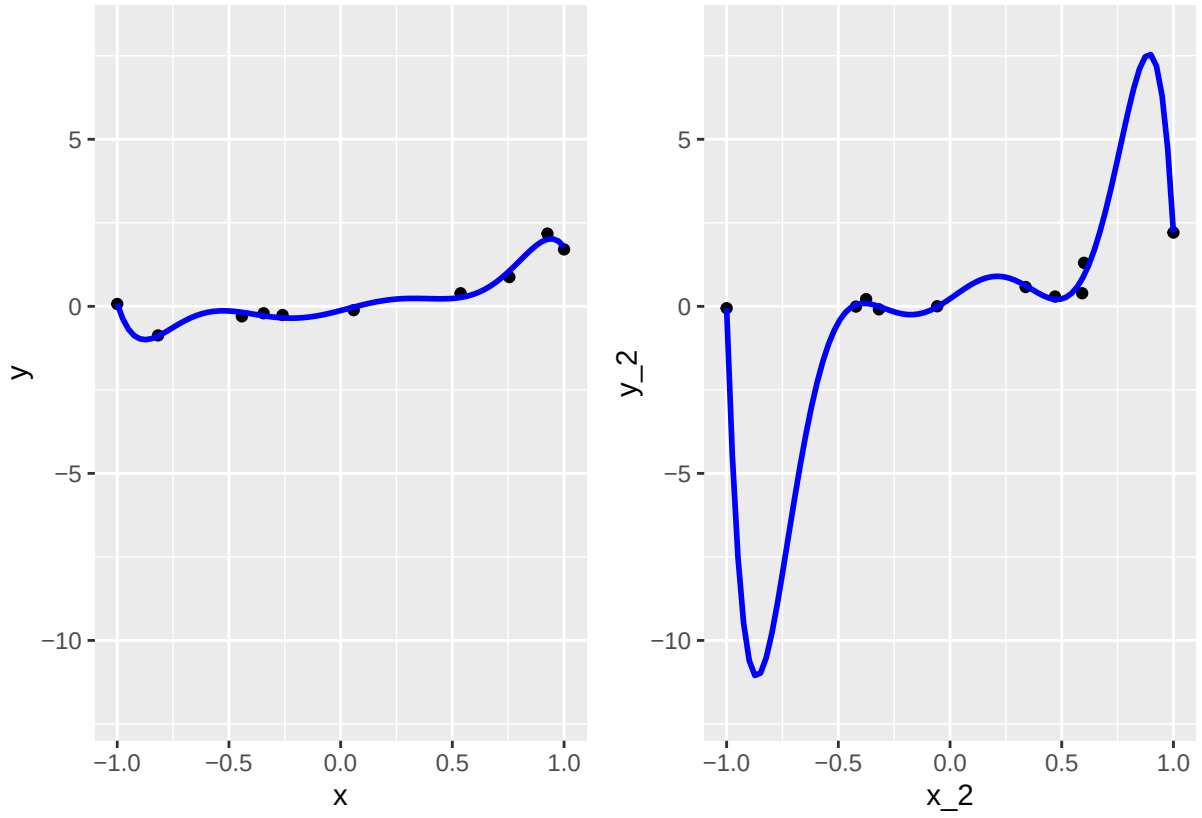
We can see that the estimator does not change too much when a different sample set of data is thrown at it. This happens because the linear estimator does not have the capacity to properly fit the 3rd-order function.

Variance of an estimator is the amount that the estimator will change given different samples of training data.

$$var[\hat{f}(x_0)] = \mathbb{E}[(\hat{f}(x_0) - \mathbb{E}[\hat{f}(x_0)])^2]$$

Considering the above equation, we can say that an estimator has high variance when the estimator changes its estimate a lot when its trained over multiple samples of the data. Simply putting it this way, estimator is flexible or complex enough to perfectly fit the training sample and because of that there is a significant amount of change in the estimate for different samples of the same data.

Let's use the same data as in the previous part, and try to fit a 7th order polynomial to it.



We can see that the 7th order polynomial is complex enough to perfectly fit the training data generated using a 3rd order polynomial. And because of this, even a small change in training data leads to high variance.

This suggests that an estimator's bias and variance are complementary to each other. (An estimator with high bias will have low variance and an estimator with high variance will have low bias)

Evaluating the performance of a Learning method

To evaluate the performance of a Statistical Learning method on any given data set, we need some way to measure how well its predictions actually match the observed data. We need to quantify the extent to which the predicted response value for a given observation is close to the true response value for that observation.

This is where the concept of MSE (Mean-Squared Error) comes into picture. MSE is a commonly-used measure. And it is given by the following formula.

$$Training\ MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(X_i))^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

We don't really care about how well the method works on the training data. We are interested in how well the method performs on the test (previously unseen) data.

We want to choose the method that gives the lowest test error, as opposed to the lowest training error.

Test Mean-Squared Error

We want to estimate the test error i.e,

$$\mathbb{E}[(\hat{f}(X_0) - y_0)^2]$$

How can we select a method the minimizes the test MSE?

- Sometimes, we might have access to a set of observations that were not used in training a learning method. We can then simply evaluate the test MSE using the above equation.
- However, when no test observations are available, selecting a method that minimizes the training MSE might seem to be a reasonable approach (because it looks like training MSE and test MSE are closely related). **BUT ITS ACTUALLY NOT!!!**

There is no guarantee that the method with the lowest training MSE will also have the lowest test MSE. (Because, many statistical learning methods specifically estimate coefficients so as to minimize the training MSE. For such methods, the training MSE can be quite small, but the test MSE is often much larger)

Bias-variance decomposition

Note: This section contains the mathematical derivation of the test MSE into bias and variance components. I highly recommend working through this. But if it's not something you'd want to do, feel free to skip to the end of this section.

We know that the test error is

$$MSE = \mathbb{E}[(y_0 - \hat{f}(X_0))^2]$$

This could be re-written as

$$MSE = \mathbb{E}[(y_0 - f(X_0) + f(X_0) - \mathbb{E}[\hat{f}(X_0)] + \mathbb{E}[\hat{f}(X_0)] - \hat{f}(X_0))^2]$$

Let us consider

$$\begin{aligned} A &= y_0 - f(X_0) \\ B &= f(X_0) - \mathbb{E}[\hat{f}(X_0)] \\ C &= \mathbb{E}[\hat{f}(X_0)] - \hat{f}(X_0)] \end{aligned}$$

Simply writing the test error in terms of A, B, C

$$\begin{aligned} MSE &= \mathbb{E}[(A + B + C)^2] \\ &= \mathbb{E}[A^2 + B^2 + C^2 + 2AB + 2AC + 2BC] \\ &= \mathbb{E}[A^2] + \mathbb{E}[B^2] + \mathbb{E}[C^2] + 2\mathbb{E}[AB] + 2\mathbb{E}[AC] + 2\mathbb{E}[BC] \end{aligned}$$

We know that $y_0 - f(X_0) = \epsilon_0$ and $\mathbb{E}[\epsilon_0] = 0$. So,

$$\begin{aligned} \mathbb{E}[AB] &= \mathbb{E}[(y_0 - f(X_0)) \cdot (f(X_0) - \mathbb{E}[\hat{f}(X_0)])] \\ &= \mathbb{E}[\epsilon_0] \cdot (f(X_0) - \mathbb{E}[\hat{f}(X_0)]) \\ &= 0 \cdot f(X_0) - \mathbb{E}[\hat{f}(X_0)] \\ &= 0 \end{aligned}$$

Also, A is the randomness from the test data and C is the randomness from the training data. So,

$$\begin{aligned} \mathbb{E}[AC] &= \mathbb{E}[(y_0 - f(X_0)) \cdot (\mathbb{E}[\hat{f}(X_0)] - \hat{f}(X_0)])] \\ &= \mathbb{E}[(y_0 - f(X_0))] \cdot \mathbb{E}[\mathbb{E}[\hat{f}(X_0)] - \hat{f}(X_0)] \\ &= 0 \end{aligned}$$

And

$$\begin{aligned}\mathbb{E}[BC] &= \mathbb{E}[(f(X_0) - \mathbb{E}[\hat{f}(X_0)]) \cdot (\mathbb{E}[\hat{f}(X_0)] - \hat{f}(X_0)])] \\ &= (f(X_0) - \mathbb{E}[\hat{f}(X_0)]) \cdot \mathbb{E}[\mathbb{E}[\hat{f}(X_0)] - \hat{f}(X_0)]] \\ &= (f(X_0) - \mathbb{E}[\hat{f}(X_0)]) \cdot [\mathbb{E}[\hat{f}(X_0)] - \mathbb{E}[\hat{f}(X_0)]] \\ &= 0\end{aligned}$$

From this, we get

$$\begin{aligned}MSE &= \mathbb{E}[A^2] + \mathbb{E}[B^2] + \mathbb{E}[C^2] \\ &= \mathbb{E}[(y_0 - \hat{f}(X_0))^2] \\ &= \mathbb{E}[(y_0 - f(X_0))^2] + \mathbb{E}[(f(X_0) - \mathbb{E}[\hat{f}(X_0)])^2] + \mathbb{E}[(\mathbb{E}[\hat{f}(X_0)] - \hat{f}(X_0)])^2]\end{aligned}$$

Taking the terms in the above equation, one at a time, and expanding them

$$\begin{aligned}\mathbb{E}[(y_0 - f(X_0))^2] &= \mathbb{E}[\epsilon_0^2] = \mathbb{E}[\epsilon^2] - (\mathbb{E}[\epsilon])^2 \\ &= var(\epsilon)\end{aligned}$$

$$\begin{aligned}\mathbb{E}[(f(X_0) - \mathbb{E}[\hat{f}(X_0)])^2] &= f(X_0) - \mathbb{E}[\hat{f}(X_0)])^2 \\ &= Bias^2 \hat{f}(X_0)\end{aligned}$$

$$\mathbb{E}[(\mathbb{E}[\hat{f}(X_0)] - \hat{f}(X_0)])^2] = var(\hat{f}(X_0))$$

So, finally we get

$$MSE = \mathbb{E}[(y_0 - \hat{f}(X_0))^2] = var(\epsilon) + Bias^2(\hat{f}(X_0)) + var(\hat{f}(X_0))$$

We also know that

$$Bias^2(\hat{f}(X_0)) \geq 0 \text{ and } var(\hat{f}(X_0)) \geq 0$$

So,

$$\mathbb{E}[(y_0 - \hat{f}(X_0))^2] \geq var(\epsilon)$$

Here, $var(\epsilon)$ is the irreducible error and $Bias^2(\hat{f}(X_0)) + var(\hat{f}(X_0))$ is the reducible error

So, from the above equation we understand that we can break down the test MSE(i.e the error of any estimator on previously unseen data) into two components:

- irreducible error (the variance of the noise in the data)
- reducible error (bias and variance of the estimator)

We only focus on minimizing the reducible error. Both bias and variance are the sources of this reducible error of an estimator.

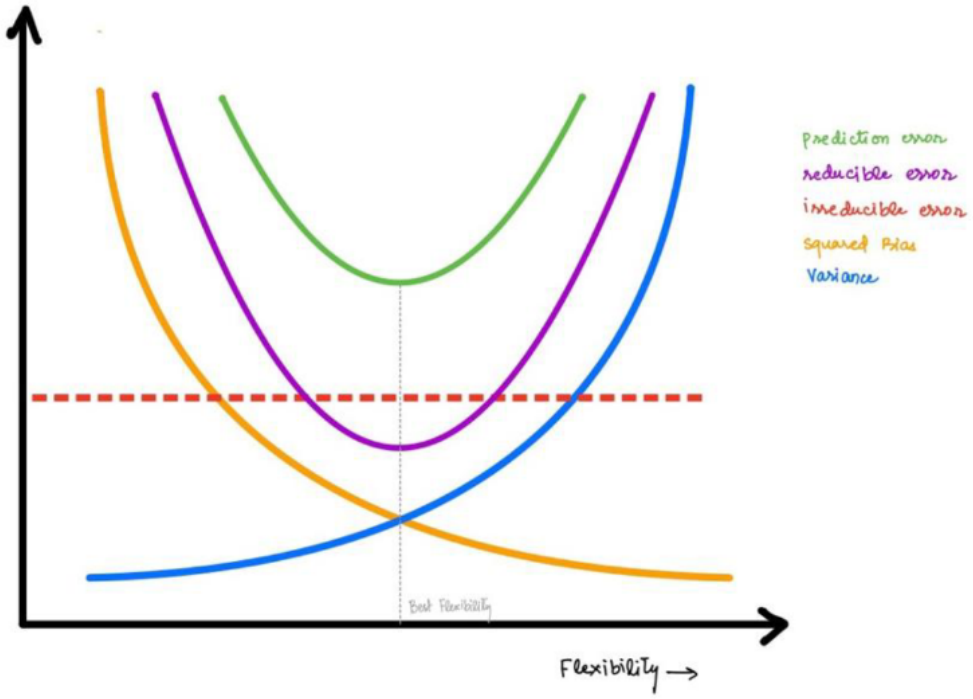
In the previous section, we had discussed that bias and variance of an estimator are complementary to each other (On increasing bias, variance decreases and vice versa). How does this property come into play while we are trying to minimize the reducible error?

Bias-Variance trade-off

Because the performance of an estimator depends on both the bias and variance, and the complementary nature of bias and variance, it is obvious that there is a trade-off between bias and variance.

Any estimator with little flexibility will not be able to explain all of the data points in the sample and will have high bias and low variance and by extension have high error. One the other hand, any estimator that is extremely flexible will explain all the data points too well (Fails to generalize the data on previously unseen sample and eventually fails to generalize the true data), and will have high variance and low bias and by extension have high error.

We want an estimator that balances bias and variance (It would be able to reduce error better than the ones that favor one extreme over the other)



The above graph is a pictorial representation of the bias-variance trade-off.

Demonstration

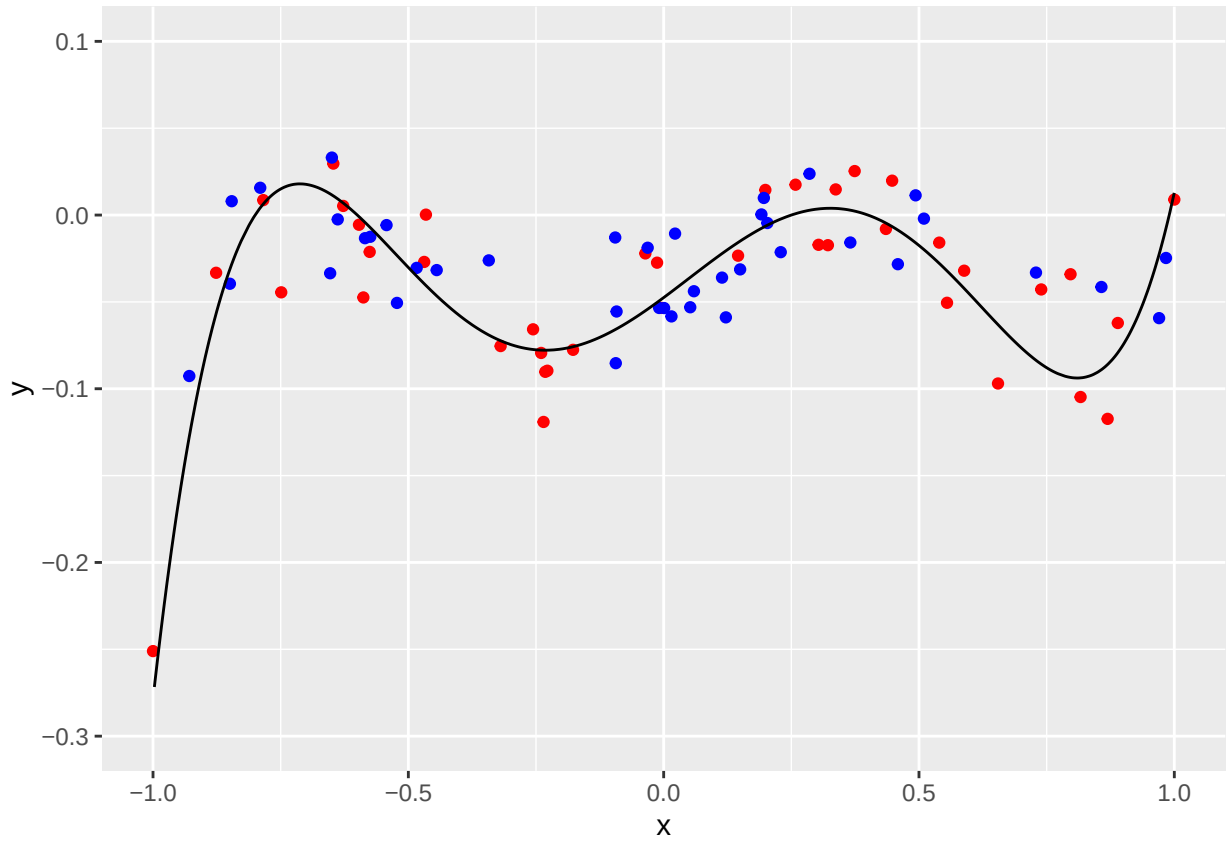
We understood the theoretical concepts of the Bias-Variance trade-off so far. Let us build on that and try out this simple demonstration.

```
set.seed(1)
# generate training data

n <- 40
x <- runif(n, -1,1)
x <- sort(x)
x[1] <- -1
x[n] <- 1

#the true data generating function is a 5-th order polynomial
#we add Gaussian noise to it

y <- (x-.99)*(x-.4)*(x-.25)*(x+.6)*(x+.8) + .03*rnorm(n)
```

```
#generate test data

n_test <- 40
x_test <- runif(n_test, -1,1)
y_test <- (x_test-.99)*(x_test-.4)*(x_test-.25)*(x_test+.6)*(x_test+.8)+
  .03*rnorm(n_test)
df_test <- as.data.frame(cbind(x_test, y_test))
```

```
# true data generating function
t <- runif(1000,-1, 1)
y_true <- (t-.99)*(t-.4)*(t-.25)*(t+.6)*(t+.8)
```

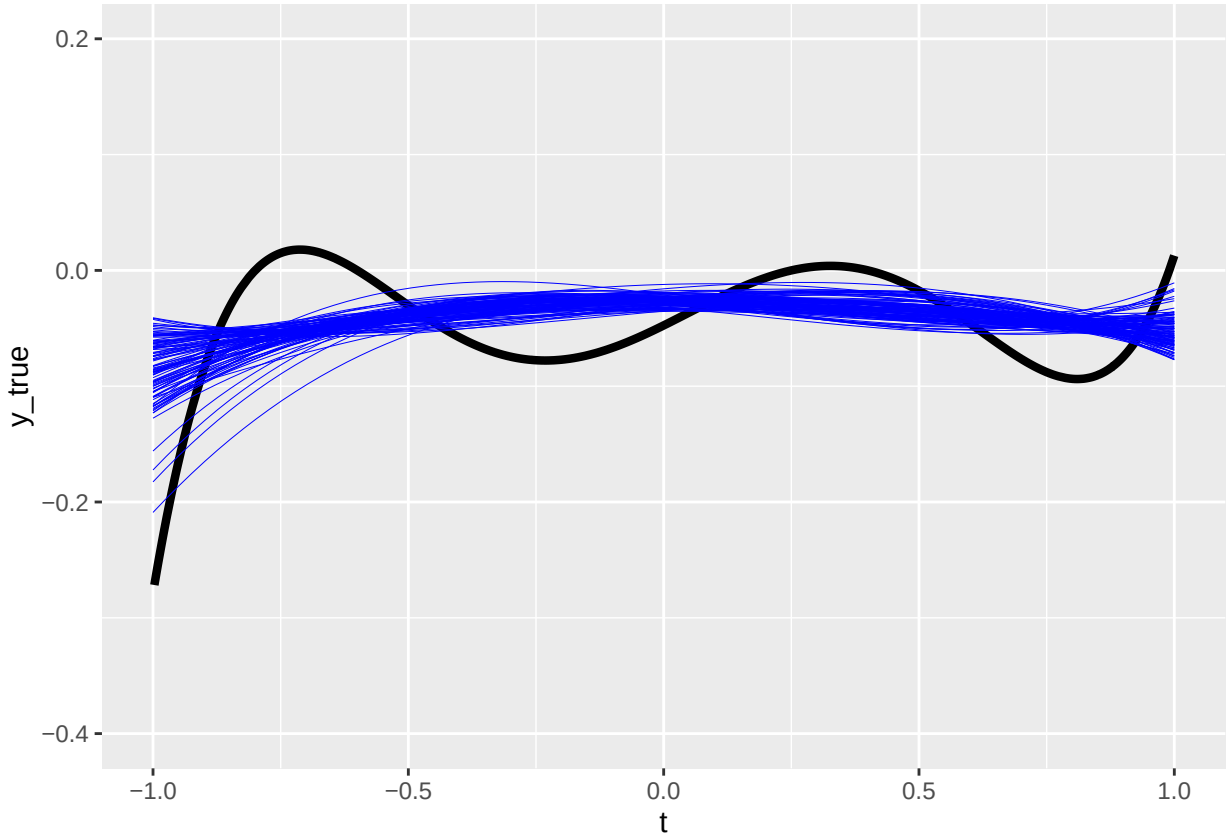
```
#plot the samples and the true data-generating function

library(ggplot2)

ggplot()+
  geom_point(aes(x,y), color="red")+
  geom_point(aes(x_test, y_test), color = "blue")+
  geom_line(aes(t, y_true))+
  ylim(-0.3,0.1)
```



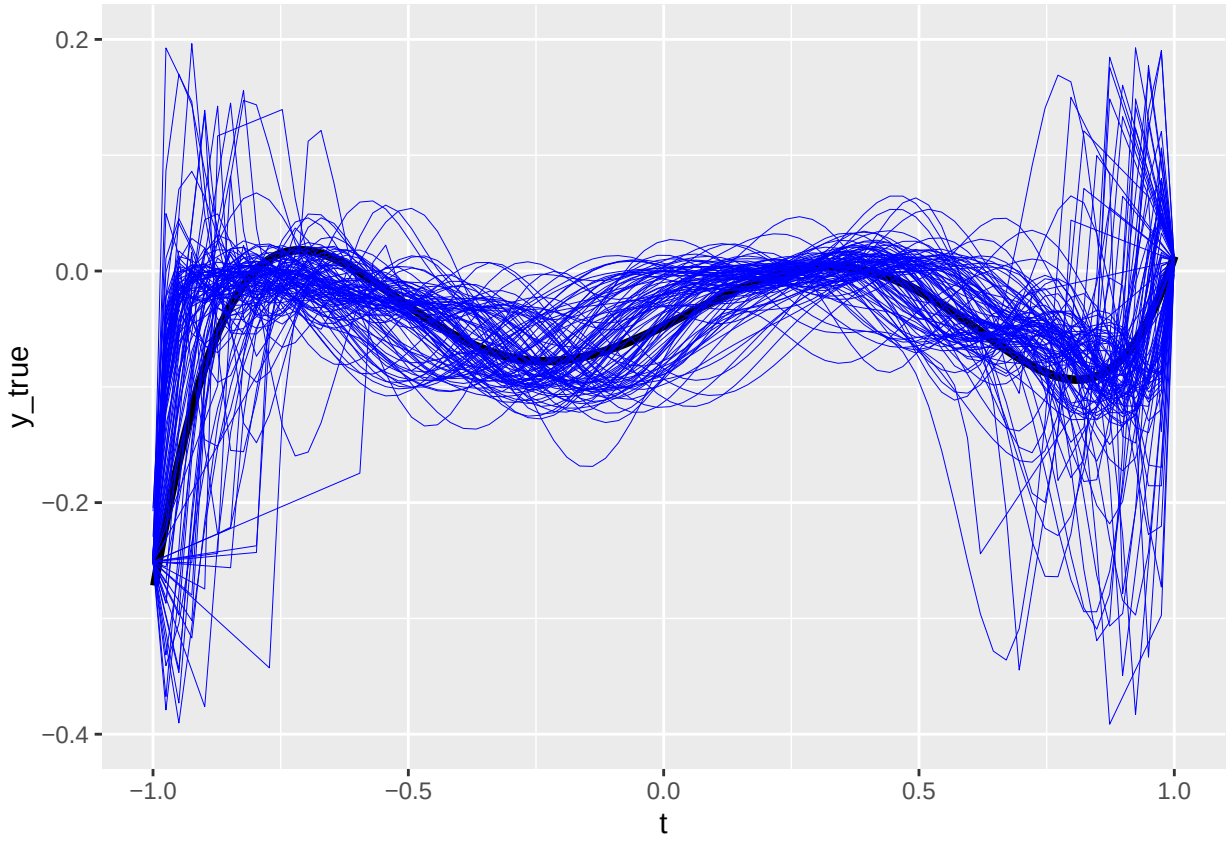

```
set.seed(2)
p <- ggplot()+
  geom_line(aes(x=t, y=y_true), size=1.5)+
  ylim(-0.4, 0.2)

q <- ggplot()+
  geom_line(aes(x=t, y=y_true), size =1.5)+
  ylim(-0.4,0.2)

r <- ggplot()+
  geom_line(aes(x=t, y=y_true), size=1.5)+
  ylim(-0.4, 0.2)

train.3_MSE <- rep(NA, 100)
test.3_MSE <- rep(NA, 100)
train.15_MSE <- rep(NA, 100)
test.15_MSE <- rep(NA, 100)
train.5_MSE <- rep(NA, 100)
test.5_MSE <- rep(NA, 100)

for (i in 1:100){
```

```r
n <- 40
x <- runif(n, -1,1)
x <- sort(x)
x[1] <- -1
x[n] <- 1
df <- as.data.frame(cbind(x,y))

# degree-3 polynomial linear regression
lm.fit.3 <- lm(y ~ poly(x,3), data = df)

y_hat_train.3 <- predict(lm.fit.3, data=df$x)
train.3_MSE[i] <- mean((y_hat_train.3-df$y)^2)
#print(train.3_MSE)

y_hat_test.3 <- predict(lm.fit.3, data=df_test$x_test)
test.3_MSE[i] <- mean((y_hat_test.3 - df_test$y_test)^2)

#print(test.3_MSE)

# degree-15 polynomial linear regression
lm.fit.15 <- lm(y ~ poly(x,15), data = df)

y_hat_train.15 <- predict(lm.fit.15, data=df)
train.15_MSE[i] <- mean((y_hat_train.15-df$y)^2)
#print(train.15_MSE)

y_hat_test.15 <- predict(lm.fit.15, data=df_test$x_test)
test.15_MSE[i] <- mean((y_hat_test.15-df_test$y_test)^2)
#print(test.15_MSE)

# degree-5 polynomial linear regression
lm.fit.5 <- lm(y ~ poly(x,5), data = df)

y_hat_train.5 <- predict(lm.fit.5, data=df$x)
train.5_MSE[i] <- mean((y_hat_train.5-df$y)^2)
#print(train.5_MSE)

y_hat_test.5 <- predict(lm.fit.5, data=df_test$x_test)
test.5_MSE[i] <- mean((y_hat_test.5 - df_test$y_test)^2)

#print(test.5_MSE)
```

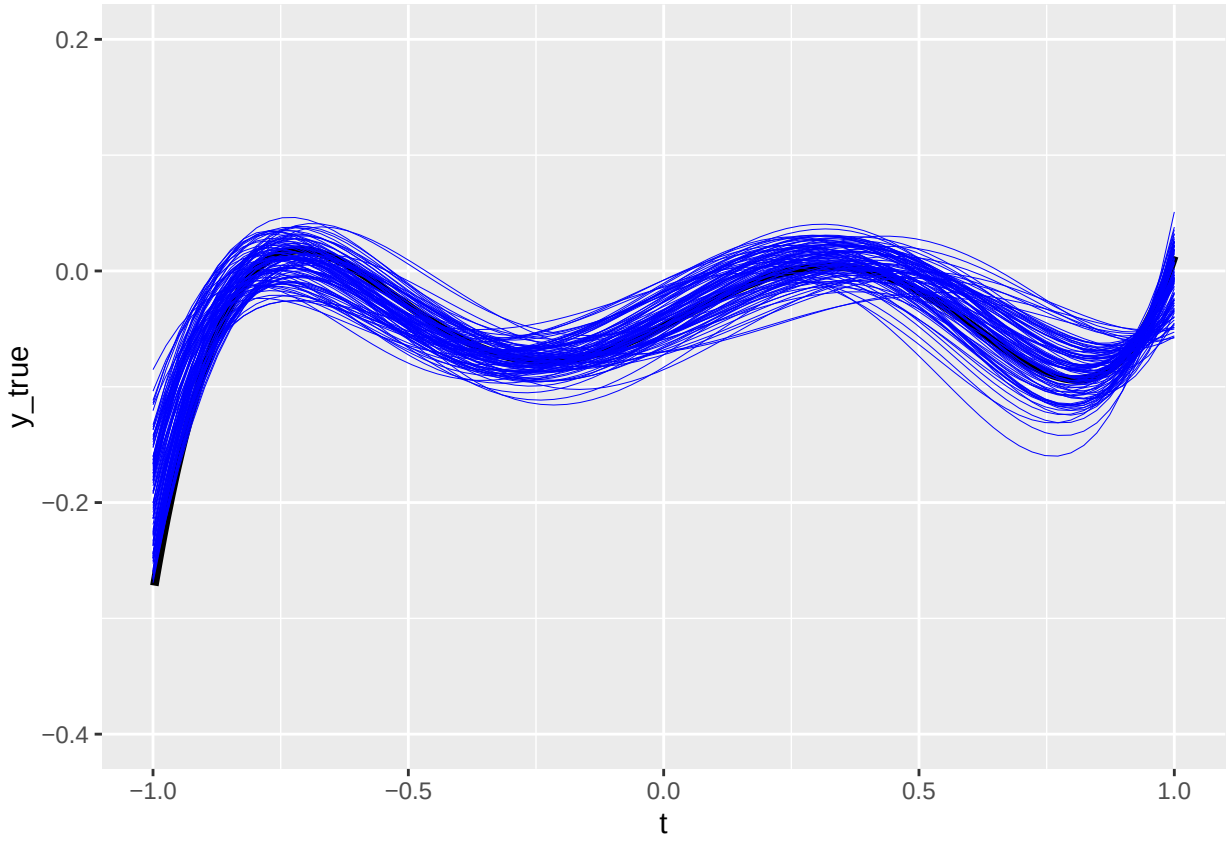
```
  r <- r +
          stat_smooth(method="lm", se=FALSE, fill=NA,
                      formula=y~poly(x,5, raw=TRUE),
                      aes_string(x, y), size = 0.1, color="blue")

  p <- p +
          stat_smooth(method="lm", se=FALSE, fill=NA,
                      formula=y~poly(x,3, raw=TRUE),
                      aes_string(x, y), size = 0.1, color="blue")


  q <- q +
    stat_smooth(method="lm", se=FALSE, fill=NA,
                formula=y~poly(x,15, raw=TRUE),
                aes_string(x, y), size = 0.1, color="blue")

}

print(p)
```



```
print(q)
```




```
print(r)
```




```
df_MSE <- as.data.frame(cbind(train.3_MSE, test.3_MSE, train.15_MSE,
                              test.15_MSE, train.5_MSE, test.5_MSE))
head(df_MSE)
```

```
##   train.3_MSE  test.3_MSE train.15_MSE test.15_MSE  train.5_MSE  test.5_MSE
## 1 0.002462239 0.001246556 0.0003545270 0.002819568 0.0005296431 0.002456169
## 2 0.002529331 0.001175276 0.0005429839 0.002688033 0.0011668327 0.001940450
## 3 0.002556723 0.001118329 0.0003544709 0.002853724 0.0009608675 0.001994569
## 4 0.002361671 0.001280452 0.0004756487 0.002604178 0.0010115080 0.001967647
## 5 0.002430272 0.001184620 0.0003841140 0.002844687 0.0011710448 0.001683357
## 6 0.002416164 0.001283760 0.0003976617 0.002745077 0.0006423343 0.002603158
```

Here, we see that the true data-generating function was a fifth-order polynomial. We try three estimators, a cubic polynomial, a 15th order polynomial, and a 5th order polynomial.

From this demonstration, we can verify our theory that the third order polynomial does not have the capacity to properly fit the true data generator and has high bias (and low variance). On the other end, the 15th order polynomial fits the data too well and fails to generalize the data and changes a lot when different samples are thrown at it. Hence, it does not generalize the true data generator and has high variance (and low bias).

Finally, we see that the 5-th order polynomial closely fits the true data-generating function. This is because it balances well between bias and variance, and by extention the test error.