Bias - Variance Tradeoff and Cross-validation

Applied Machine Learning for Educational Data Science

true

09/27/2021

Contents

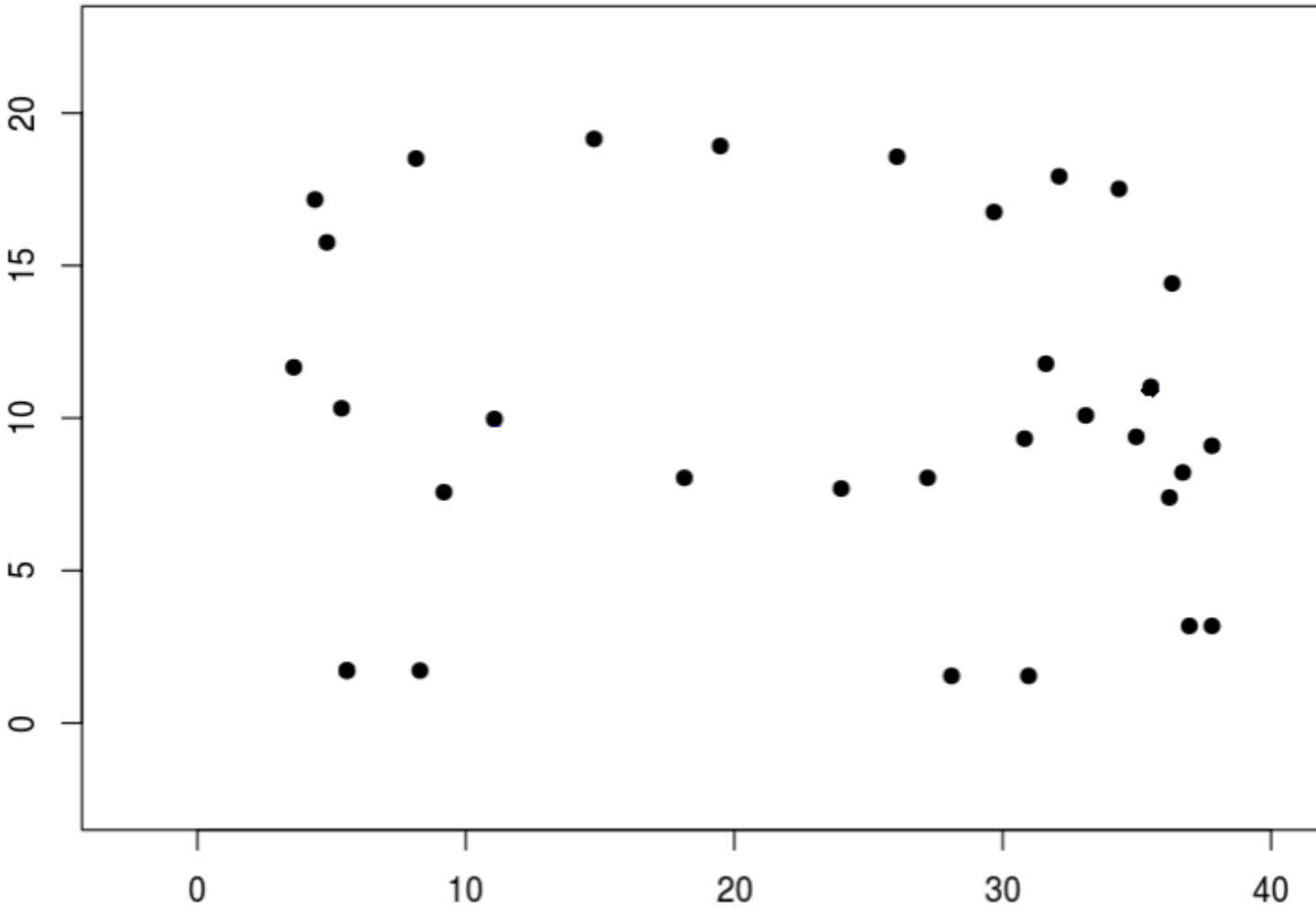
[Updated: Wed, Oct 06, 2021 - 16:57:23]

How many parameters does it take to draw an elephant?

Once upon a time, two physicists met and one convinced the other that the agreement between some model-based calculations and measured experimental numbers were only superficial. In that conversation, von Neumann was quoted as saying this famous phrase *"... with four parameters I can fit an elephant, and with five I can make him wiggle his trunk."* You can read the full story here.

Since then, a number of people tried to come up with mathematical models that can drawn an elephant with as few parameters as possible. This has become an interesting activity when people want to make a point about the how complex of a model one would need to understand what we observe in real world.

Now, we will join them. See the following plot that has a number of data points. Would you say there is an elephant there? If so, can you come up with a mathematical model to fit these data points? How complex would that model be? How many parameters would you need?



Below is a web application with such a model. You can increase the number of parameters in this model from 1 to 70, and the model predictions will start to look like an elephant. The purpose of our quick exploration is to find the number of parameters you would use to model an elephant. Start manipulating the **p** (number of parameters) and examine how the model predicted contour changes. Stop when you believe that you can convince someone else that it looks like an elephant.

https://kourentzes.shinyapps.io/FitElephant/

The Principle of Parsimony

Bias - Variance Tradeoff

When we use a model to predict an outcome, there are two main sources of error: model error and sampling error.

Model Error: Given that no model is a full representation of truth underlying observed data, every model is misspecified to some degree. Conceptually, we can define the model error as the distance between the

model and true generating mechanism underlying data. Technically, for a given set of predictors, it is the difference between the expected value predicted by the model and the true value underlying data. The term **bias** is also commonly used for model error.

Sampling Error: Given that the amount of data is fixed during any modeling process, it will decrease the stability of parameter estimates for models with increasing complexity across samples drawn from the same population. Consequently, this will increase the variance of predictions (more variability of a predicted value across different samples) for a given set of same predictors. The term **estimation error** or **variance** is also used for sampling error.

The essence of any modeling activity is to balance these two sources of error and find a stable model (generalizable across different samples) with the least amount of bias.

Bias and Variance of Model Predictions

We will do a simple Monte Carlo experimentation to better understand these two sources of error. Suppose that there is a true generating model underlying some observed data. This model is

$$y = e^{(x-0.3)^2} - 1 + \epsilon,$$

where x is a predictor variable that is equally spaced and ranges from 0 to 1, ϵ is a random error component and follows a normal distribution with a mean of zero and standard deviation of 0.1, and y is the outcome variable. Suppose that we simulate a small observed data following this model with a sample size of 20. Then, we use a very simple linear model to represent the observed simulated data.

$$y = \beta_0 + \beta_1 x + \epsilon$$

```
set.seed(09282021)

N = 20

x <- seq(0,1,length=20)

x
```

```
 [1] 0.00000000 0.05263158 0.10526316 0.15789474 0.21052632 0.26315789
 [7] 0.31578947 0.36842105 0.42105263 0.47368421 0.52631579 0.57894737
[13] 0.63157895 0.68421053 0.73684211 0.78947368 0.84210526 0.89473684
[19] 0.94736842 1.00000000
```

```
e <- rnorm(20,0,.1)

e
```

```
 [1]  0.07372726  0.08253427  0.13678980 -0.04993081  0.10368134  0.28473311
 [7] -0.03402811 -0.01834963 -0.02296964  0.02782503 -0.15425785 -0.13371024
[13]  0.03465939  0.21786527 -0.09607842  0.07927619  0.11618340 -0.19217742
[19] -0.07000210 -0.05165884
```

```
y <- exp((x-0.3)^2) - 1 + e

y
```

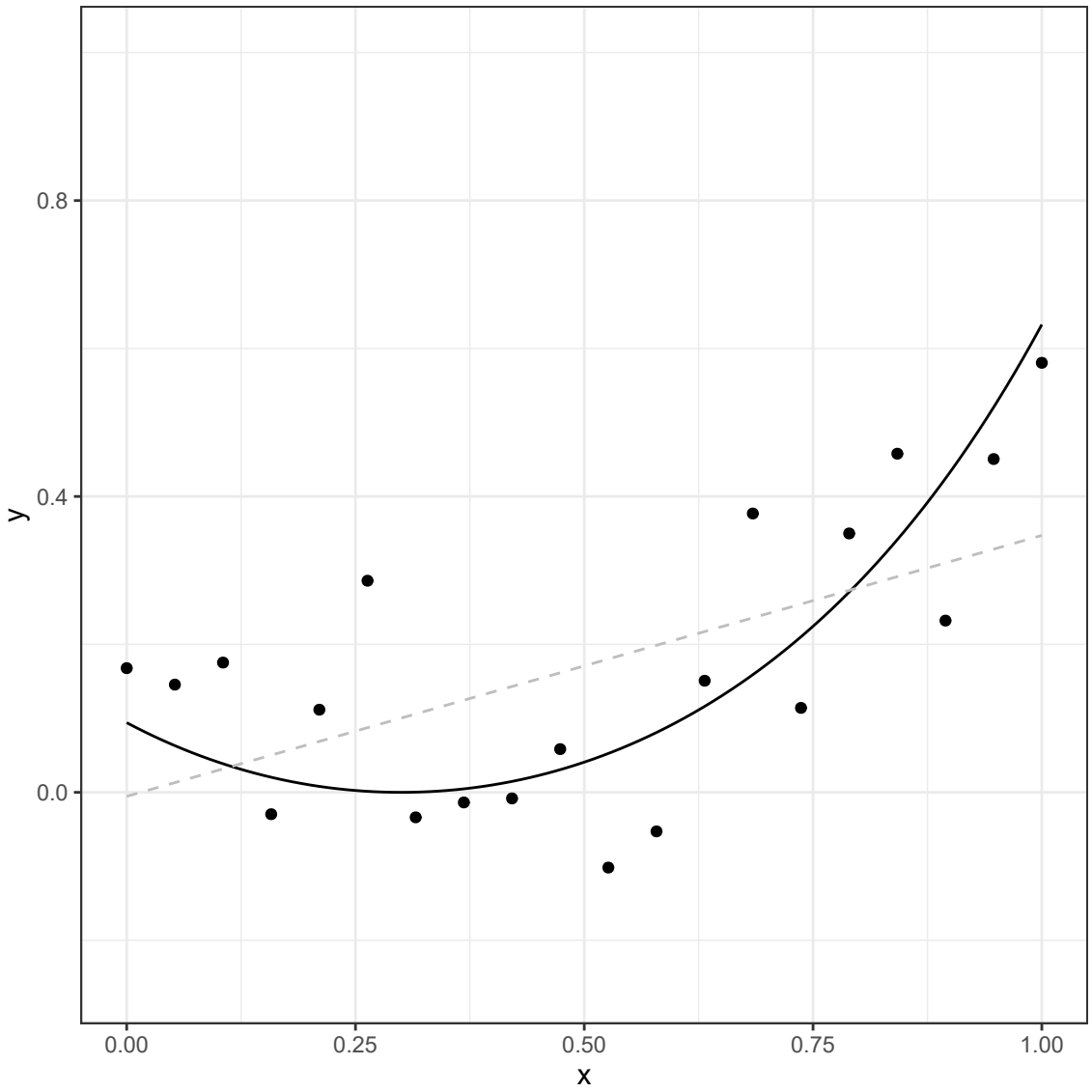
```
 [1]  0.167901540  0.145636360  0.175440469 -0.029531625  0.111719007
 [6]  0.286091377 -0.033778768 -0.013657214 -0.008208013  0.058450844
[11] -0.101704649 -0.052791207  0.150875614  0.376935001  0.114176511
[16]  0.349997226  0.457803769  0.232167369  0.450568829  0.580657384
```

```
mod <- lm(y ~ 1 + x)
mod
```

```

Call:
lm(formula = y ~ 1 + x)

Coefficients:
(Intercept)            x
   -0.00542      0.35272
```

```
predict(mod)
```

```
          1           2           3           4           5           6
-0.005420071  0.013143883  0.031707837  0.050271791  0.068835745  0.087399698
          7           8           9          10          11          12
 0.105963652  0.124527606  0.143091560  0.161655514  0.180219468  0.198783422
         13          14          15          16          17          18
 0.217347376  0.235911330  0.254475284  0.273039238  0.291603192  0.310167146
         19          20
 0.328731099  0.347295053
```

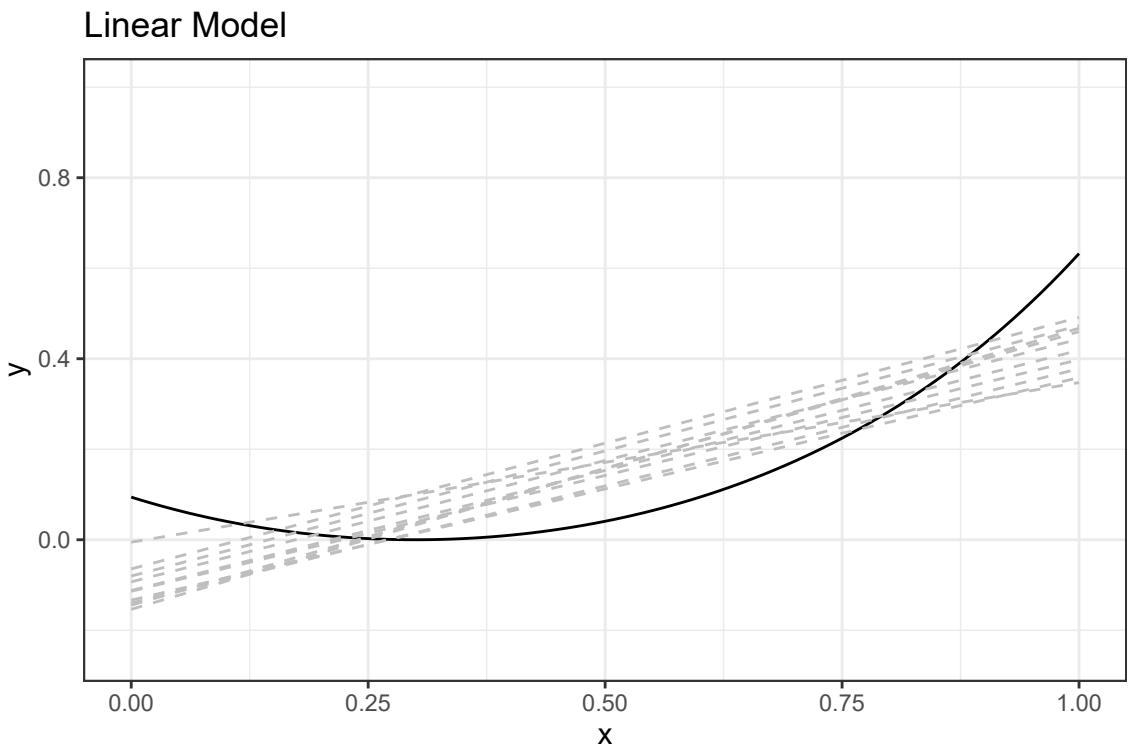


The solid line in this plot represents the true nature of the relationship between x and y. The observed data points do not lie on this line due to random error component (noise). The gray dashed line is how we represent the relationship between x and y if we use a simple linear model.

This demonstration only represents a single dataset. Now, suppose that we repeat the same process 10 times. We will produce 10 different datasets with the same size (N=20) using the exact same predictor values (x) and true data generating model. Then, we will fit a simple linear model to each one of these 10 datasets.

```
set.seed(09282021)

E  <- vector('list',10)
Y  <- vector('list',10)
M1 <- vector('list',10)

N = 20

x <- seq(0,1,length=N)
```

```r
for(i in 1:10){

  E[[i]]  <- rnorm(N,0,.1)
  Y[[i]]  <- exp((x-0.3)^2) - 1 + E[[i]]

  M1[[i]] <- lm(Y[[i]] ~ 1 + x)
}
```



The solid line again represents the true nature of the relationship between x and y. There are 10 different lines (gray, dashed) and each lines represents a simple linear model fitted to a different dataset simulated by using the exact same data generating mechanism. The table below provides a more detailed look at the fitted values from each replication for every single x value.

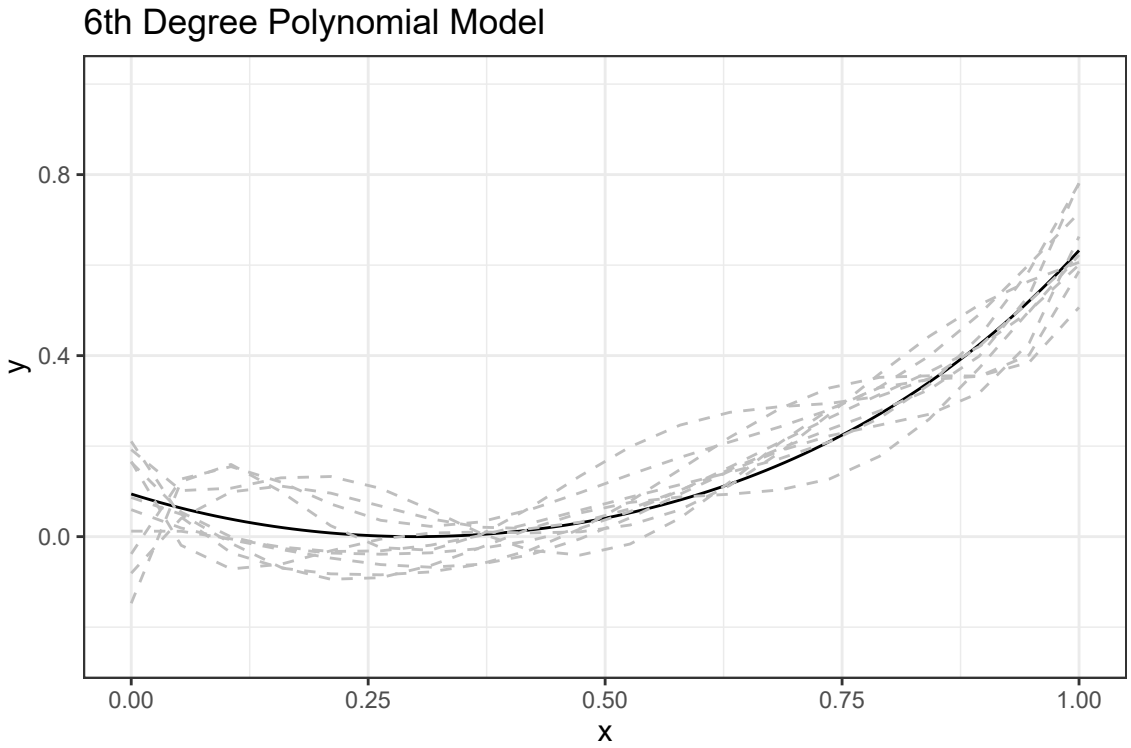
x	y.true	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	Mean	SD
0.000	0.094	0.192	0.166	0.164	0.012	-0.038	-0.081	0.211	0.087	0.060	-0.147	0.086	0.105
0.053	0.063	0.102	-0.019	0.054	0.012	0.127	0.040	0.043	0.049	0.019	0.122	0.048	0.045
0.105	0.039	0.107	-0.071	-0.001	-0.005	0.155	0.099	-0.036	-0.015	-0.007	0.160	0.025	0.076
0.158	0.020	0.130	-0.061	-0.026	-0.023	0.121	0.112	-0.069	-0.068	-0.028	0.100	0.010	0.085
0.211	0.008	0.133	-0.031	-0.036	-0.033	0.072	0.096	-0.082	-0.094	-0.047	0.024	-0.003	0.082
0.263	0.001	0.107	-0.006	-0.039	-0.031	0.036	0.068	-0.084	-0.091	-0.062	-0.024	-0.011	0.069
0.316	0.000	0.059	0.008	-0.036	-0.019	0.023	0.039	-0.078	-0.064	-0.068	-0.030	-0.015	0.050
0.368	0.005	0.008	0.010	-0.026	0.002	0.032	0.021	-0.061	-0.024	-0.061	0.004	-0.011	0.034
0.421	0.015	-0.030	0.008	-0.009	0.029	0.060	0.018	-0.029	0.018	-0.040	0.064	0.003	0.033
0.474	0.031	-0.041	0.011	0.018	0.058	0.097	0.032	0.017	0.052	-0.005	0.134	0.027	0.040
0.526	0.053	-0.016	0.026	0.056	0.087	0.137	0.061	0.078	0.076	0.042	0.198	0.061	0.043
0.579	0.081	0.041	0.059	0.102	0.115	0.175	0.102	0.148	0.088	0.094	0.246	0.103	0.041
0.632	0.116	0.121	0.113	0.154	0.142	0.209	0.148	0.220	0.094	0.145	0.275	0.150	0.042
0.684	0.159	0.207	0.186	0.209	0.172	0.243	0.195	0.283	0.103	0.190	0.287	0.198	0.049
0.737	0.210	0.281	0.271	0.263	0.210	0.280	0.237	0.329	0.128	0.223	0.294	0.247	0.057
0.789	0.271	0.329	0.360	0.314	0.262	0.330	0.278	0.352	0.178	0.246	0.310	0.294	0.059
0.842	0.342	0.347	0.443	0.364	0.336	0.399	0.325	0.355	0.260	0.270	0.350	0.345	0.057
0.895	0.424	0.354	0.512	0.420	0.442	0.491	0.399	0.355	0.373	0.318	0.421	0.407	0.065
0.947	0.521	0.402	0.565	0.497	0.587	0.602	0.533	0.385	0.499	0.428	0.520	0.500	0.080
1.000	0.632	0.586	0.606	0.622	0.781	0.715	0.781	0.507	0.601	0.663	0.622	0.651	0.092

For instance, when the x is equal to 0, the true value of y based on the model would be 0.094. However, when we fit a linear model to 10 different datasets with the underlying true model, the average predicted value was -.107 with a standard deviation of 0.047 across 10 replications. Similarly, when the x is equal to 0.316, the true value of y based on the model would be 0, but the average prediction was 0.059 with a standard deviatoin of 0.032 across 10 replications. It is clear that a linear model provides biased estimates such that there is an underestimation at the lower values of x and higher values of x while there is an overestimation in the middle of the range of x.

Now, let's do the same experiment by fitting a more complex 6th degree polynomial to the same datasets with the same underlying true model.

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4 + \beta_5 x^5 + \beta_6 x^6 + \epsilon$$

```
set.seed(09282021)

E  <- vector('list',10)
Y  <- vector('list',10)
M6 <- vector('list',10)

N = 20

x <- seq(0,1,length=N)

for(i in 1:10){

  E[[i]]  <- rnorm(N,0,.1)
  Y[[i]]  <- exp((x-0.3)^2) - 1 + E[[i]]

  M6[[i]] <- lm(Y[[i]] ~ 1 + x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6))
}
```



x	y.true	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	Mean	SD
0.000	0.094	0.192	0.166	0.164	0.012	-0.038	-0.081	0.211	0.087	0.060	-0.147	0.086	0.105
0.053	0.063	0.102	-0.019	0.054	0.012	0.127	0.040	0.043	0.049	0.019	0.122	0.048	0.045
0.105	0.039	0.107	-0.071	-0.001	-0.005	0.155	0.099	-0.036	-0.015	-0.007	0.160	0.025	0.076
0.158	0.020	0.130	-0.061	-0.026	-0.023	0.121	0.112	-0.069	-0.068	-0.028	0.100	0.010	0.085
0.211	0.008	0.133	-0.031	-0.036	-0.033	0.072	0.096	-0.082	-0.094	-0.047	0.024	-0.003	0.082
0.263	0.001	0.107	-0.006	-0.039	-0.031	0.036	0.068	-0.084	-0.091	-0.062	-0.024	-0.011	0.069
0.316	0.000	0.059	0.008	-0.036	-0.019	0.023	0.039	-0.078	-0.064	-0.068	-0.030	-0.015	0.050
0.368	0.005	0.008	0.010	-0.026	0.002	0.032	0.021	-0.061	-0.024	-0.061	0.004	-0.011	0.034
0.421	0.015	-0.030	0.008	-0.009	0.029	0.060	0.018	-0.029	0.018	-0.040	0.064	0.003	0.033
0.474	0.031	-0.041	0.011	0.018	0.058	0.097	0.032	0.017	0.052	-0.005	0.134	0.027	0.040
0.526	0.053	-0.016	0.026	0.056	0.087	0.137	0.061	0.078	0.076	0.042	0.198	0.061	0.043
0.579	0.081	0.041	0.059	0.102	0.115	0.175	0.102	0.148	0.088	0.094	0.246	0.103	0.041
0.632	0.116	0.121	0.113	0.154	0.142	0.209	0.148	0.220	0.094	0.145	0.275	0.150	0.042
0.684	0.159	0.207	0.186	0.209	0.172	0.243	0.195	0.283	0.103	0.190	0.287	0.198	0.049
0.737	0.210	0.281	0.271	0.263	0.210	0.280	0.237	0.329	0.128	0.223	0.294	0.247	0.057
0.789	0.271	0.329	0.360	0.314	0.262	0.330	0.278	0.352	0.178	0.246	0.310	0.294	0.059
0.842	0.342	0.347	0.443	0.364	0.336	0.399	0.325	0.355	0.260	0.270	0.350	0.345	0.057
0.895	0.424	0.354	0.512	0.420	0.442	0.491	0.399	0.355	0.373	0.318	0.421	0.407	0.065
0.947	0.521	0.402	0.565	0.497	0.587	0.602	0.533	0.385	0.499	0.428	0.520	0.500	0.080
1.000	0.632	0.586	0.606	0.622	0.781	0.715	0.781	0.507	0.601	0.663	0.622	0.651	0.092

DISCUSSION

Compare the numbers in these two tables and discuss the differences you observe. What happened to predictions when you fit a more complex model (6th degree polynomial) instead of a simple regression

model? You can examine the following plot that displays the average prediction and range of predictions across 10 replications for every single value of x.

We can expand our experiment and examine a range of models starting from linear model up to the 6th degree polynomial. The following plots displays what you would see if you repeated this experiment by fitting a linear model, quadratic model, qubic model, a 4th degree polynomial, a 5th degree polynomial, and a 6th degree polynomial model to the same simulated datasets with the same underlying model. A table follows these plots that presents the bias and standard deviation of predictions across 10 replications for making comparisons.

$$y = \beta_0 + \beta_1 x + \epsilon$$
$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \epsilon$$
$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \epsilon$$
$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4 + \epsilon$$
$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4 + \beta_5 x^5 + \epsilon$$

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4 + \beta_5 x^5 + \beta_6 x^6 + \epsilon$$

x	y (TRUE)	Bias	SD	Bias	SD	Bias	SD	Bias	SD	Bias	SD	Bias	SD
0.000	0.094	-0.201	0.047	-0.010	0.051	-0.013	0.075	-0.001	0.073	-0.018	0.092	-0.008	0.105
0.053	0.063	-0.142	0.044	-0.012	0.046	-0.013	0.047	-0.013	0.049	-0.004	0.052	-0.016	0.045
0.105	0.039	-0.090	0.041	-0.013	0.042	-0.013	0.044	-0.020	0.064	-0.004	0.078	-0.013	0.076
0.158	0.020	-0.044	0.039	-0.014	0.039	-0.013	0.051	-0.022	0.075	-0.011	0.085	-0.011	0.085
0.211	0.008	-0.004	0.036	-0.014	0.036	-0.013	0.057	-0.020	0.076	-0.018	0.078	-0.011	0.082
0.263	0.001	0.030	0.034	-0.014	0.034	-0.012	0.059	-0.016	0.069	-0.023	0.064	-0.013	0.069
0.316	0.000	0.059	0.032	-0.012	0.033	-0.010	0.056	-0.010	0.057	-0.022	0.048	-0.015	0.050
0.368	0.005	0.082	0.031	-0.009	0.032	-0.007	0.050	-0.004	0.043	-0.017	0.035	-0.016	0.034
0.421	0.015	0.099	0.030	-0.004	0.032	-0.003	0.042	0.002	0.031	-0.007	0.024	-0.012	0.033
0.474	0.031	0.111	0.030	0.001	0.032	0.001	0.035	0.008	0.024	0.005	0.021	-0.004	0.040
0.526	0.053	0.117	0.031	0.006	0.033	0.006	0.031	0.013	0.024	0.017	0.027	0.008	0.043
0.579	0.081	0.116	0.031	0.012	0.033	0.011	0.033	0.017	0.030	0.027	0.035	0.022	0.041
0.632	0.116	0.108	0.033	0.018	0.034	0.017	0.038	0.020	0.036	0.032	0.042	0.033	0.042
0.684	0.159	0.093	0.035	0.023	0.036	0.021	0.042	0.021	0.042	0.032	0.045	0.039	0.049
0.737	0.210	0.070	0.037	0.026	0.037	0.024	0.044	0.020	0.047	0.027	0.047	0.037	0.057
0.789	0.271	0.037	0.039	0.027	0.039	0.025	0.042	0.018	0.050	0.016	0.051	0.024	0.059
0.842	0.342	-0.006	0.042	0.024	0.042	0.022	0.039	0.014	0.052	0.003	0.057	0.003	0.057
0.895	0.424	-0.062	0.045	0.015	0.044	0.015	0.042	0.008	0.053	-0.008	0.061	-0.017	0.065
0.947	0.521	-0.130	0.048	0.000	0.048	0.002	0.062	0.001	0.063	-0.009	0.064	-0.021	0.080
1.000	0.632	-0.214	0.051	-0.023	0.052	-0.020	0.101	-0.008	0.095	0.009	0.106	0.019	0.092

DISCUSSION

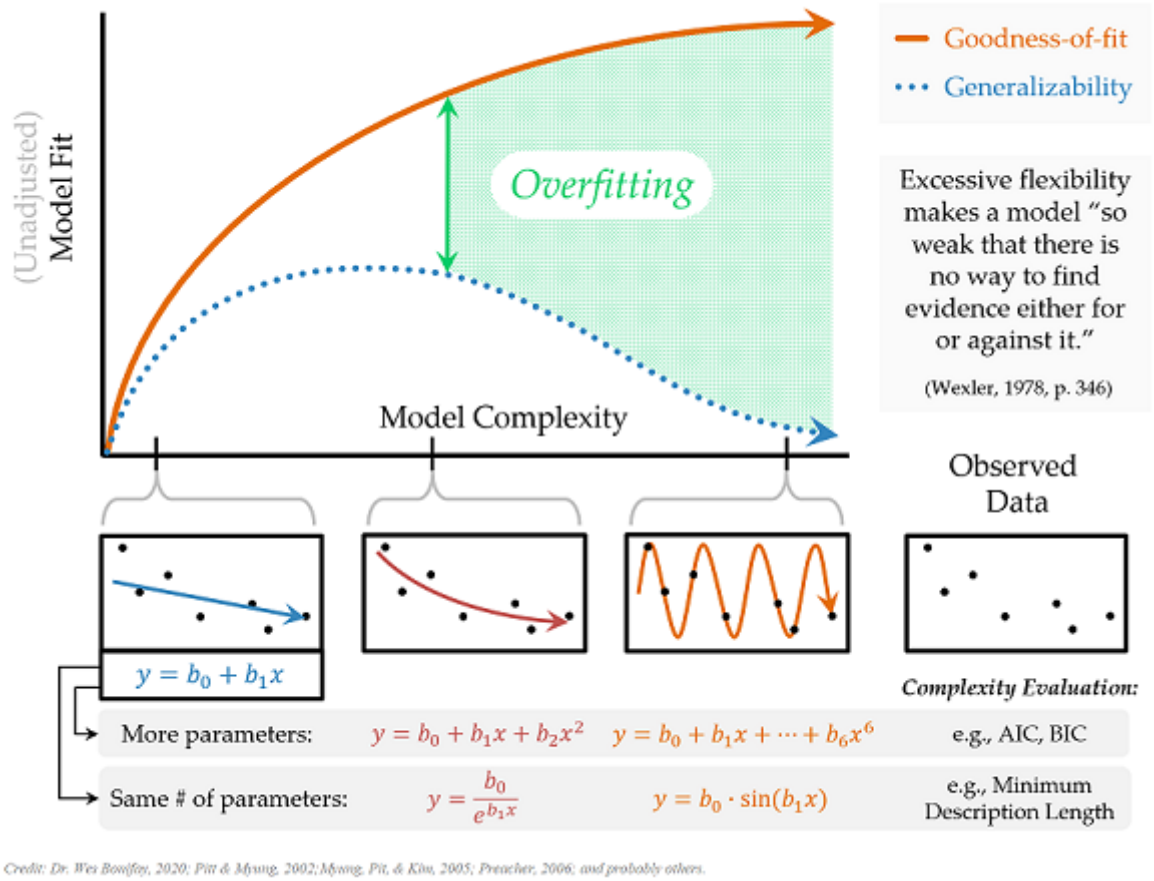
If you have to choose one of these models for one of the simulated datasets at hand, which one would you choose? Why?

Moral of the Story: Underfitting vs. Overfitting

Large model bias happens when we underfit and do not use all the information available in the dataset. An example of underfitting for the experimentation above would be using a linear model to represent the relationship between x and y for one of the sample datasets. Note that there is always a model bias to some degree for all these six models because none of them is the true model. However, it is the most obvious for the linear model that doesn't account for nonlinearity in the dataset. On the other hand, you can see that the linear model is the most robust to sampling variation. It is the most stable and provide the most consistent predictions for different datasets (smaller variation in predictions across 10 replications).

Large model variance happens when we overfit and try to extract more information than available in the dataset. An example of overfitting for the experimentation above would be using any model beyond the quadratic model. When this happens, we start modeling noise (error) in the sample dataset as if it provides some useful information while such information is unique to a specific sample dataset and there is no guarantee that it will be replicable for other samples from the same population. Notice that the bias does not improve much for models beyond the quadratic model; however, the variance of predictions keep increasing for more complex models. In other words, more complex models are less stable and not robust to sampling variation. The predictions from more complex models tend to vary more from sample to sample although they are less biased. In other words, there is less confidence that the model will be generalizable for observations outside of our sample.

Could you find that sweet model that provides a reasonable representation of data and also provides a reasonable amount of generalizability (consistent/stabile predictions for observations other than the ones you used to develop the model)?



Credit: Dr. Wes Bonifay, 2020; Pitt & Myung, 2002; Myung, Pitt, & Kim, 2005; Preacher, 2006; and probably others.

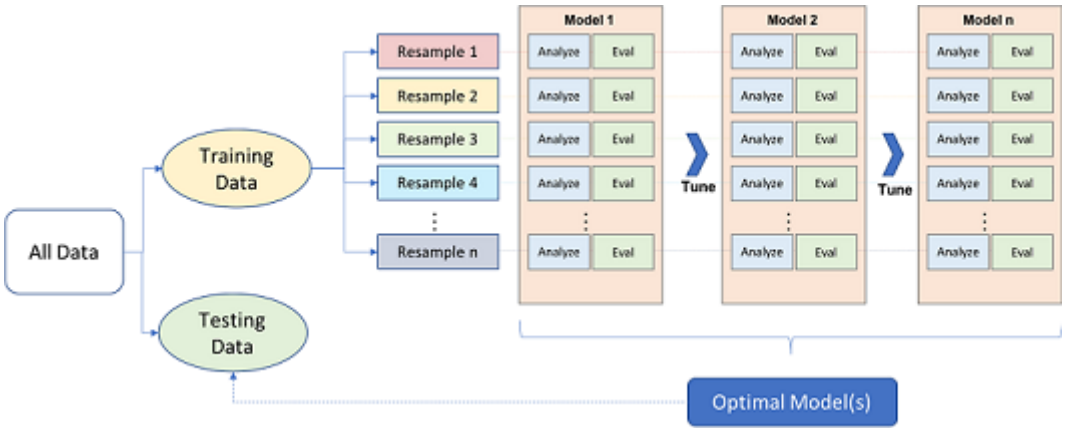
Facing the Reality

When try to understand/predict a phenomenon measured in some way in social and behavioral sciences, there is probably not a true model we can use as a reference to understand bias and variance of our predictions, like we had in our experimentation above. If there is such a thing that is a 'true model', it is probably a very complex system with many many variables that has an effect on the measured outcome variable. It would be reasonable to acknowledge that there are some variables with relatively larger important effects, there are many variables with small effects, and many others with tapering smaller effects, and the interactions among all these variables. Our models are just approximations to this full reality.

Since we have a fixed amount of data, we only have limited information and can reveal these effects only to certain degree. The more data we have, the more and smaller effects we can detect and separate from noise. So, the complexity of the model we can afford is limited to the amount of data and information available in data. The most challenging part of any modeling activity is to find the amount of complexity we can afford with the sample data at hand and also a model that can perform good enough for out of sample observations.

Use of Resampling Methods to Balance Model Bias and Model Variance

There are certain strategies applied in practice to avoid overfitting and finding the sweet spot between model bias and model variance. This process is nicely illustrated in Boehmke and Greenwell (2020, Figure 2.1)



We first split data into two main components: training dataset and test dataset. While there is no particular rule for the size of training and test datasets, it is common to see 80-20 or 70-30 splits based on the size of the original dataset. Training dataset is mainly used for exploring and model development while test set is mainly used for validating the performance of a final model. There may be different approaches being used while doing the initial split of training and test datasets such as **simple random sampling**, **stratified sampling**, or **down-sampling/up-sampling** for imbalanced data (typically happens for classification problem when there is a great imbalance among categories).

During the exploration and model development, it is also a good strategy to cross-validate the model performance within the training set. This is typically done by creating multiple partitions within the training set, and testing models on each partition while optimizing the parameters. There are different approaches for creating different partitions in the training dataset such as ***k*-fold cross-validation** or **bootstrapping**, but k-fold cross validation is the most common. In k-fold cross validation, the training sample is randomly partitioned into k sets of equal size. A model is fitted to k-1 folds and the remaining fold is used to test the model performance. This can be repeated k times by treating a different fold as an hold-out set. Finally,

the performance evaluation metric is aggregated (e.g., average) across k replications to get a k-fold cross-validation estimate of the performance evaluation metric. Once the model is optimized and a final model is developed as a result of this cross-validation process, then the final model is trained using the whole training data and evaluated one final time on test dataset to measure the generalizability of model predictions.

Back to the Elephant

If you are curious more about drawing an elephant, a more recent paper by Mayer, Khairy, and Howard (2010) provided a mathematical model that can draw an elephant with only four complex parameters (just like what von Neumann said). Below is an R code to reproduce their model using R.

Even more, this paper by Boué (2019) argues that you can approximate any dataset of any modality with a single parameter. Go figure!

```
# 4 complex parameters

  p1 <- 50-30i
  p2 <- 18+8i
  p3 <- 12-10i
  p4 <- -14-60i

  Cx <- c(0,50i,18i,12,0,-14)
  Cy <- c(0,-60-30i,8i,-10i,0,0)

# t, parameter that can be interpreted as the elapsed time while going along
# the path of the contour

  t <- seq(0,2*pi,length.out = 1000)

# X-coordinates

  x <- c()

  A <- c(0,0,0,12,0,-14)  # Real part of Cx
  B <- c(0,50,18,0,0,0)   # Imaginary part of Cx

  for(i in 1:length(t)){
    k <- 0:5
    x[i] <- sum(A*cos(k*t[i]) + B*sin(k*t[i])) # Eq 1
  }

# Y-coordinates

  y <- c()

  A <- c(0,-60,0,0,0,0)     # Real part of Cy
  B <- c(0,-30,8,-10,0,0)   # Imaginary part of Cy

  for(i in 1:length(t)){
    k <- 0:5
    y[i] <- sum(A*cos(k*t[i]) + B*sin(k*t[i])) # Eq 2
  }
```

```
# Function to draw the elephant

  plot(y,-x,type='l')
```