The Approximation-Generalization Trade-off

Nikolaj Nasenko

2021/09/27

1. Statistical background

The main purpose of statistical learning is to derive generalizations. Thus, looking only at a sample of data, we want to understand the relationship of covariates that is universal, not just the prevailing one in that particular data. Therefore, the analytical practice is to split the data into a training set, i.e., data on which the model is trained, and a test set, i.e., data on which the model is not trained. The training data are then used to estimate how well our model performs. The key metric we use to asses the amount of the generalization error is the difference between the training error (**in-sample error** or $\mathrm{E_{in}}$) and the test error (**out-of-sample error** or $\mathrm{E_{out}}$). The interrelation between $\mathrm{E_{in}}$ and $\mathrm{E_{out}}$ is explained in terms of the Hoeffding inequality.

$$P[|E_{in}(g(x)) - E_{out}(g(x))| > \epsilon] \leq 2Me^{-2\epsilon^2 N}, \quad \text{for any } \epsilon > 0$$

Where ϵ denotes the generalization error, $g(x)$ is the hypothesis we pick out of our hypothesis set H (our chosen model) such that $g(x)$ best approximates the target function[1] $f(x)$, where M is a measure of the size of the hypothesis set H (model complexity) and N is the amount of data.

The essentials of statistical learning are set to find:

- $\mathbf{E_{in}}$ **closest to** $\mathbf{E_{out}}$ ($E_{in} \approx E_{out}$), i.e. the **generalization** ability
- $\mathbf{E_{in}}$ **as small as possible** ($E_{in} \approx 0$), i.e. the **proximity**/accuracy

In more complex models, $\mathrm{E_{in}}$ tends to zero, but with respect to the Hoeffding inequality, it is also more likely to deviate from $\mathrm{E_{out}}$. Otherwise, in less complex models, $\mathrm{E_{in}}$ increases but is more likely to approach $\mathrm{E_{out}}$. Thus, there exists a trade-off between approximation and generalization. Therefore, we must strike a balance when choosing H.

Note, however, that the Hoeffding inequality also shows that the approximation-generalization problem disappears with large sets of data.

2. Bias and Variance

The bias-variance trade-off offers another angle on the trade-off between approximation and generalization. Assume we have noisy data.

$$y = f(x) + \epsilon, \quad \text{where } E(\epsilon) = 0$$

We can decompose out-of-sample error into the sum of the variance of $g(x)$, the squared bias of $g(x)$, and the variance of the error term ϵ.

$$E_D[E_{out}(g^{(D)}(x))] = E_D[E_x[(y - g^{(D)}(x))^2]] = Var(g^{(D)}(x)) + [Bias(g^{(D)}(x))]^2 + Var(\epsilon)$$

[1] In the equation $y = f(x) + \epsilon$, we call y the target and $f(x)$ the target function.

Where E_D denotes the expected value with respect to all data sets and E_x with respect to x. The dependence of the hypothesis $g(x)$ on a particular data set D is referred to as $g^{(D)}(x)$.

The variance of the hypothesis $g(x)$ tells us how our estimate of $g(x)$ changes if we use different training data. If $g(x)$ approximates well target function $f(x)$ on the training data ($E_{in} \approx 0$), then even small changes in the training data may cause $g(x)$ to differ substantially from $\bar{g}(x)$. Thus, the variance increases and we may not be able to generalize well.

$$Var(g^{(D)}(x)) = E_D[(g^{(D)}(x) - \bar{g}(x))^2], \quad \text{where } \bar{g}(x) = E_D[g^{(D)}(x)]$$

The bias term refers to the deviation of the average function $g(x)$ from y. In other words, it tells how our average estimate of y differs from the actual value of y. If our $g(x)$ is similar to $\bar{g}(x)$, then the bias becomes larger, but our generalization ability improves.

$$[Bias(g^{(D)}(x))]^2 = (y - \bar{g}(x))^2$$

Hence, we see that there is again a trade-off between approximation and generalization, but this time in terms of variance and bias.

Since $g^{(D)}(x)$ depends on the chosen model and learning algorithm, $g^{(D)}(x)$ will vary accordingly, so will the bias and variance.

Finally, even though bias and variance are measured by the squared error $\mathrm{E}_{\mathrm{out}}$, the learning algorithm itself does not have to minimize the squared error loss function, it can be any loss function.

3. Simulating the approximation-generalization trade-off

Now that we are familiar with the theoretical foundations of the approximation-generalization trade-off, let us see whether we can find such a relationship on simulated data. First, let us generate some polynomial target function f from which we create noisy data.

```
# Simulating data
x <- 1:100
f <- 100 + 0.0004*(x - 50)^3 # target function

set.seed(42)
y <- f + rnorm(100, 0, 10) # noisy data
data <- data.frame(x, y) # our dataset
```

We then split the data into training and test set such that the training set contains 60% of the data.

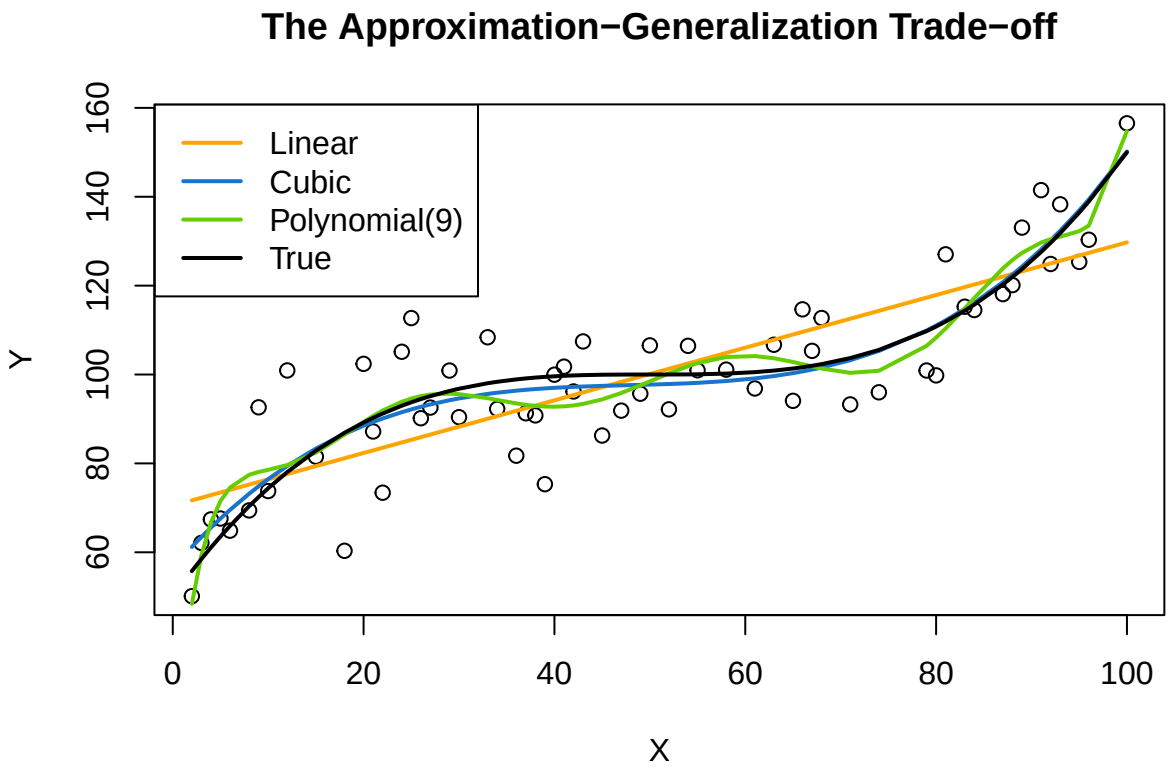
```
# Test and Training set
set.seed(42)
index <- sort(sample(1:nrow(data), 60)) # randomly choosing 60% of data
train <- data[index,]
test <- data[-index,]
```

In the next step, we choose three different models from less complex to more complex, i.e. linear, cubic and polynomial of degree 9. As for the learning algorithm, we apply the MSE estimator.

```
# Linear, Cubic and Polynomial(9) model
linear <- lm(y~x, data = train)
cubic <- lm(y~poly(x, 3, raw = TRUE), data = train)
polynomial <- lm(y~poly(x, 9, raw = TRUE), data = train)
```

We now have all we need to visualize all three models against the true model and to check how well these models approximate the target function that generated the given data.

```
plot(train$x, train$y, xlab = "X", ylab = "Y",
     main = "The Approximation-Generalization Trade-off")
lines(train$x, fitted(linear), col = "orange", lwd = 2)
lines(train$x, fitted(cubic), col = "dodgerblue3", lwd = 2)
lines(train$x, fitted(polynomial), col = "chartreuse3", lwd = 2)
lines(train$x, f[index], lwd = 2, col = "black")
legend("topleft", legend = c("Linear", "Cubic", "Polynomial(9)", "True"),
       col = c("orange", "dodgerblue3", "chartreuse3", "black"), lwd = 2)
```



Finally, we compute the in-sample and out-of-sample MSE errors for each model and put it into a table.

```
# In-sample and Out-of-sample error
(errors <- data.frame(
  Model = c("Linear", "Cubic", "Polynomial(9)"),
  Ein = c( # in-sample error
    mean(resid(linear)^2),
    mean(resid(cubic)^2),
    mean(resid(polynomial)^2)
  ),
  Eout = c( # out-of-sample error
    mean((test$y - predict(linear, newdata = test))^2),
    mean((test$y - predict(cubic, newdata = test))^2),
```

```
    mean((test$y - predict(polynomial, newdata = test))^2)
  )))
```

```
##           Model       Ein     Eout
## 1        Linear 139.65345 121.3381
## 2         Cubic  97.34535 115.1792
## 3 Polynomial(9)  85.55151 149.0443
```

We see that the in-sample error (Ein) decreases as the model becomes more complex, whereas the out-of-sample error (Eout) initially decreased and then increased again. Thus, a trade-off is indeed present. Based on the results, the most appropriate function that optimizes the approximation and generalization is cubic. That is a logical conclusion as the target function we used to generate this data has a cubic form. We can see from the plot that the true function and the cubic function we estimated are in fact very similar.

4. References

ABU-MOSTAFA, Y. S., MAGDON-ISMAIL, M. and LIN, H. (2012), *Learning from Data: a Short Course*, AMLbook.

JAMES, G., WITTEN, D., HASTIE, T. and TIBSHIRANI, R. (2021), *An Introduction to Statistical Learning: with Applications in R*, 2nd ed., Springer.