Chapter 10

Generalization and Overfitting

Throughout this course so far we have described various approaches to learn a model from training data. However, we have not given deep attention to the challenge of ensuring that our model will also *generalize* well to unseen test data. We have hinted at the challenge of generalization, but we have not analyzed it in detail. In this chapter, we will formalize some notions surrounding generalization, and—in the next chapter—we will introduce some general strategies that can promote generalization capabilities in machine learning models.

10.1 Features and Complexity

Imagine we are working on a regression problem with a single input feature (see Figure 10.1 for an example). Examining our training data, we might decide that a simple linear function is not complex enough (e.g., the data is clearly non-linear).

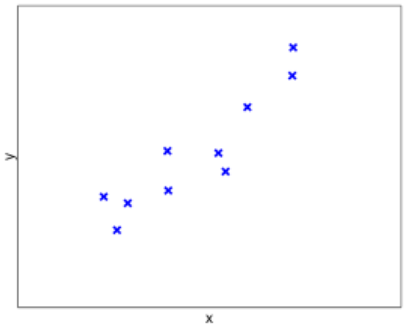

Figure 10.1: A simple example of a regression dataset.

A primer on feature design

As discussed in Chapter 3, one common way to increase the complexity of a linear model is to add more complicated features. For example, we might add higher-order polynomial features to our linear model by replacing our original model

$$f(x) = \mathbf{w}^\top \mathbf{x} + b \tag{10.1}$$

with a higher-order variant

$$f(x) = b + +w_1 x + w_2 x^2 ... + w_k x^k. \tag{10.2}$$

This is equivalent to replacing the original feature inputs x with k-dimensional feature vectors $[x, x^2, ..., x^k]^\top$. There are many ways to add new features to our model. We might add polynomial features as in the above example, we might add the logarithm of our original input, or–depending on the application—we might even collect and define new features from our raw data (e.g., new text features for spam classification). In general, this idea is called *feature design*, and we will discuss it in more detail in Chapter 16.

Adding features increases complexity

In this chapter, we will focus on the example of adding polynomial features, but our discussion will apply to adding new features in general. The key idea is that adding new features is the most common way to increase the complexity of a machine learning model. In fact, nearly every machine learning model used in practice can be interpreted as a linear model acting on some complex set of features! As long as the features we add have some correlation with the target, adding these features will improve the ability for a model to fit the training data. Figure 10.2 illustrates this idea using our simple regression dataset. It shows the best fit line generated by a regression model as we add more and more high-order polynomial features.

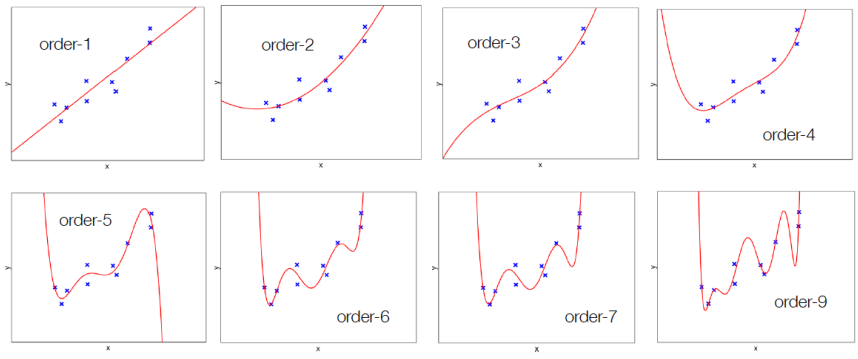

Figure 10.2: Regression fit using a linear model with increasing high-order polynomial features.

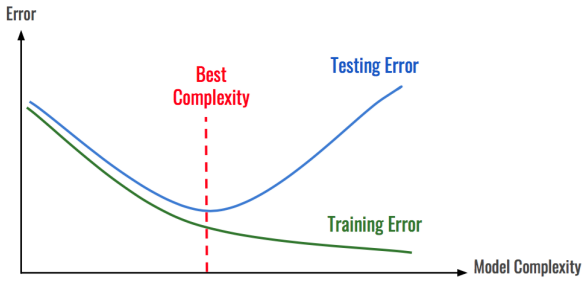

Figure 10.3: The training error decreases as we increase the model complexity, but at some point the testing error begins to increase.[1]

A key point to recognize is that adding these more complicated features improves the fit on the training data, but this does not guarantee that the model will achieve better performance on unseen data. In terms of empirical risk minimization, adding more complicated features will achieve a lower minimum for the empirical risk, but this might not actually reflect an improvement on the true underlying distribution. If we look at the order-9 fit in Figure 10.2 it seems obvious that this regression model is far too erratic and will not actually generalize well—despite the fact that it can fit the training data near perfectly.

10.2 Overfitting

In machine learning terms, we would say that the order-9 model in Figure 10.2 has *overfit* the training data. The term overfitting is used to describe a situation in which our training error is much lower than our error on held out (i.e., test) data. Our risk of overfitting increases as the complexity of our model increases (e.g., as we add more data). This idea is illustrated in Figure 10.3. They key point is that our training error will always decrease as we increase the model complexity, but at some point our generalization performance (i.e., our test error) will start to increase due to overfitting.

10.3 Bias and Variance

In the case of regression using the mean-squared error, we can gain some statistical insight into overfitting based on the notions of bias and variance.

10.3.1 Decomposing the mean-squared error

Suppose we want to compute the *expected* mean-squared error on a test point $(\mathbf{x}, y)$. One way of doing this is computing the expectation assuming that our

[1] Image credit: https://hackernoon.com/memorizing-is-not-learning-6-tricks-to-prevent-overfitting-in-machine-learning-820b091dc42

model $\hat{y} = f(\mathbf{x})$ is trained on a dataset sampled from the true distribution $\mathcal{D}_{\text{trn}}$. In other words, we want to compute the expected risk of our trained model, assuming that we draw a random training set from the underlying data distribution:

$$\mathbb{E}_{\mathcal{D}_{\text{trn}},(\mathbf{x},y)\sim P_{\mathcal{D}}}[(y-\hat{y})^2] = \mathbb{E}_{\mathcal{D}_{\text{trn}},(\mathbf{x},y)\sim P_{\mathcal{D}}}[(y-f(\mathbf{x}))^2]. \tag{10.3}$$

We use $\mathcal{D}_{\text{trn}}, (\mathbf{x}, y) \sim P_{\mathcal{D}}$ in the denominator to denote that this expectation is taken with respect to two random variables: the training dataset $\mathcal{D}_{\text{trn}}$ and test datapoint $(\mathbf{x}, y)$, both of which are independently sampled from the underlying data distribution $P_{\mathcal{D}}$. In terms of empirical risk minimization, Equation 10.3 gives the expected risk under the assumption that we are optimizing our model f using a finite training set. By incorporating the sampling of training dataset into this expectation, this measure of risk captures two sources of stochasticity: the stochasticity from sampling test points and the stochasticity from training our model on a finite data sample.

A key insight is that we can use some algebra and the properties expected values, variances, and covariances[2] to rearrange the expectation in Equation 10.3 in a pleasing form. Note that for notational simplicity, we drop the subscript from the expectations, variances, and covariances in what follows. However, it is important to note that all the distributional measures are computed with respect to sampling both a training dataset and a testing point. [3]

$$\begin{aligned}
\mathbb{E}[(y-\hat{y})^2] &= \mathbb{E}[y^2] - \mathbb{E}[2y\hat{y}] + \mathbb{E}[\hat{y}^2] \\
&= \mathbb{E}[y^2] + \mathbb{E}[\hat{y}^2] - 2\mathbb{E}[y\hat{y}] \\
&= \mathbb{E}[y^2] + \mathbb{E}[\hat{y}^2] - 2(\text{Cov}(y,\hat{y}) + \mathbb{E}[\hat{y}]\mathbb{E}[y]) \\
&= \mathbb{E}[y^2] + \mathbb{E}[\hat{y}^2] - 2\text{Cov}(y,\hat{y}) - 2\mathbb{E}[\hat{y}]\mathbb{E}[y] \\
&= \text{Var}(y) + \mathbb{E}[y]^2 + \text{Var}(\hat{y}) + \mathbb{E}[\hat{y}]^2 - 2\text{Cov}(y,\hat{y}) - 2\mathbb{E}[\hat{y}]\mathbb{E}[y] \\
&= \text{Var}(y-\hat{y}) + \mathbb{E}[y]^2 + \mathbb{E}[\hat{y}]^2 - 2\mathbb{E}[x]\mathbb{E}[y] \\
&= \text{Var}(y-\hat{y}) + (\mathbb{E}[y] - \mathbb{E}[\hat{y}])^2.
\end{aligned} \tag{10.4}$$

Thus, in the end we can decompose the expected mean-squared error as a combination of two terms. A squared bias term

$$(\mathbb{E}[y] - \mathbb{E}[\hat{y}])^2 \tag{10.5}$$

that measures the difference between the expected model prediction and true expected value of the distribution. And a variance term

$$\text{Var}(y-\hat{y}) = \mathbb{E}[((y-\hat{y}) - \mathbb{E}[y-\hat{y}])^2], \tag{10.6}$$

[2] We will make use of the following well-known properties: $\text{Var}(x) = E[x^2] - E[x]^2$, $E[xy] = E[x]E[y] + \text{Cov}(x,y)$, and $\text{Var}(x-y) = Var(x) + Var(y) - 2Cov(x,y)$.

[3] Many online references—including Wikipedia—do not properly deal with the distributional assumptions when deriving their bias-variance trade-off expressions for prediction models (e.g., they ignore the fact that both the true value y and the prediction $\hat{y}$ depend on the sampled training point). Caution is urged when referring to online references for this topic.

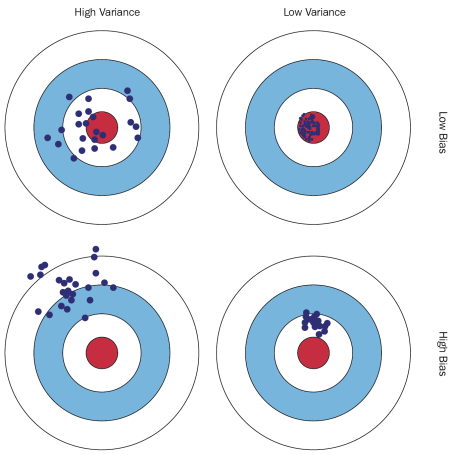

Figure 10.4: Illustration of bias and variance.[4]

that measures how much the prediction errors vary, depending on both the random sample of training data used and what the prediction target is. A model with high bias tends to make systematic errors when predicting for the test point. A model with high variance will have prediction errors that vary wildly. An intuitive visualization of bias and variance is given Figure 10.4. High bias and low variance predictions are consistently off target. In contrast, high variance and low bias predictions are erratic but tend to average out to a correct prediction.

Breaking down the variance term The variance term (Equation 10.6) measures how drastically our prediction errors tend to vary. It is important to note that such a variance can come from two sources. On the one hand, our model might be overly complicated and tend to make erratic predictions. On the other hand—however—it might simply be that the prediction task is inherently noisy, meaning that our model will occasionally have large or small errors for reasons that are outside of our control. Indeed, for many prediction tasks it is unreasonable to assume that achieving perfectly consistent generalization performance is possible. Thus it is important to distinguish between modeling variance and inherent noise in the data.

We can formalize this distinction by assuming that *some* optimal prediction function f^* exists, which is able to consistently predict the targets up to some noise level

$$y = f^*(\mathbf{x}) + \epsilon, \tag{10.7}$$

where ϵ is a random noise variable with zero mean. This assumption implies that the target values in our data distribution $P_{\mathcal{D}}$ can be generated

[4]Image credit: https://www.oreilly.com/library/view/hands-on-transfer-learning/

by combining a deterministic prediction function f^* to the input features and adding a small amount of noise ϵ. Note that $\mathbb{E}[\epsilon] = 0$ implies that $\mathbb{E}[f^*(\mathbf{x})] = y$, as expected. Now, we can further decompose the variance term in Equation 10.6 relative to the optimal predictor f^*:

$$\begin{aligned} \text{Var}(y - \hat{y}) &= \text{Var}(f^*(\mathbf{x}) + \epsilon - \hat{y}) && (10.8) \\ &= \text{Var}(f^*(\mathbf{x}) - \hat{y}) + \text{Var}(\epsilon) + 2\text{Cov}((f^*(\mathbf{x}) - \hat{y}), \epsilon) && (10.9) \\ &= \text{Var}(f^*(\mathbf{x}) - \hat{y}) + \text{Var}(\epsilon), && (10.10) \end{aligned}$$

where we used the fact that the noise is assumed to be independent from our prediction function. Thus, we see that variance comes from two sources. First, we have a variance due to our model

$$\text{Var}(f^*(\mathbf{x}) - \hat{y}) = \text{Var}(f^*(\mathbf{x}) - f(\mathbf{x})) \tag{10.11}$$

which assesses how drastically our predictions vary, compared to an optimal model. Second, we have an intrinsic noise term

$$\text{Var}(\epsilon), \tag{10.12}$$

which measures how intrinsically noisy our prediction problem is. Generally, when we discussing the bias-variance trade-off in machine learning, we are referring to the model variance and are ignoring the intrinsic noise.

10.3.2 The bias-variance trade-off and overfitting

The bias-variance decomposition is closely related to the notion of overfitting. Models that overfit have high variance and low bias. Due to their complexity, these models are able to perfectly fit the training data (i.e., they achieve low bias in training), but their complexity means that their predictions tend to vary drastically on the testing set. In contrast, models that are high bias and low variance are often said to *underfit*. These models make stable but incorrect predictions.

The bias-variance decomposition is referred to as a trade-off because these two error terms are inherently in conflict. In general, strategies that decrease bias will increase model complexity (and thus variance). On the other hand, strategies that seek to stabilize model predictions tend to induce bias. In fact, in many contexts it can be proved that asymptotically unbiased models will have unbounded variance.

10.4 Complexity and Generalization

Bias and variance provides one statistical perspective on the notion of overfitting, but it is rooted in the mean-squared error metric. There are also more

general notions of complexity in the context of empirical risk minimization. For example, the following theorem relates model complexity to the underlying risk of a statistical model.

Theorem 3. *Suppose we are optimizing over a finite model class $\mathcal{F}$. Let $R^*(f)$ denote the true risk of the model f (Equation 7.1) and let $R(f)$ denote the empirical risk (Equation 7.3). Assume that our loss function L is bounded in $[0, 1]$. Then with probability at least $1 - \delta$ we have that*

$$R^*(f) \leq R(f) + \sqrt{\frac{\log(|\mathcal{F}|) + \log(\frac{2}{\delta})}{2n}}, \tag{10.13}$$

where n is the number of examples used to compute the empirical risk $R(f)$.

The proof of Theorem 3 is beyond the scope of this course, but it can be found in various textbooks. The key idea is that we can *probabilistically* bound the gap between the empirical risk and the true risk, as long as our model comes from a finite model class $\mathcal{F}$. We can think of this as a *generalization gap*. If a model has a large generalization gap, then strong performance on a training set will not be a reliable indication of strong performance on unseen data.

Theorem 3 tells us that the generalization gap tends to increase as the size of the model class $|\mathcal{F}|$ increases, i.e., as the model class becomes more complex. Indeed, another way of interpreting the $\log(|\mathcal{F}|)$ term is that it measures how many bits of information it takes to encode our model. For example, one of the simplest classification models would be a *majority class* model, which simply predicts the majority class in the training data. Such a majority class model would require only one bit to encode (i.e., whether we are predicting true or false) and the size of the model class would be two. Thus, models that require more information to encode tend to have higher generalization gaps. On the other hand, generalization gaps tends to decrease as we get more training data. Intuitively, we can accommodate more complicated models as long as we have more data to train them.

Thus, this theorem illustrates two key points. It illustrates how more complex model classes tend to be noisier (i.e., higher variance) and prone to overfitting, in that they give us less reliable estimates of the true risk. It also illustrates how adding more training data can reduce this variance.

Of course, one issue in Theorem 3 is that no popular or practical machine learning models belong to a finite model class. For example, even linear models—which are some of the simplest machine learning models—have an infinitely large model class, since there are infinitely many lines (or hyperplanes) in any continuous space. Nonetheless, if we replace the $\log(|\mathcal{F}|$ term in Theorem 3 with some general notion of "model complexity", then the key idea of the theorem still holds in general: the more complex a model is, the harder it is to guarantee that it will generalize, especially when training on small datasets.

Theorem 3 is a simple example of a probabilistically approximately correct (PAC) bound. We obtain a bound that tells us that our estimate will be approximately correct, but only with a probability $1 - \delta$. PAC bounds are a primary

focus of statistical learning theory. Statistical learning theory also provides many ways to characterize the complexity of model classes, since most model classes (e.g., linear models) have continuous parameters and are thus infinite. For example, many of these notions build upon the idea of geometric margins and separating hyperplanes, which we discussed in Chapter 3.