Linear Regression

Linear regression example

Optimization of $w$

Figure 5.1
Underfitting and Overfitting in Polynomial Estimation

Underfitting

Appropriate capacity

Overfitting

Figure 5.2
Generalization and Capacity

![Diagram showing the relationship between capacity and error](image)

Figure 5.3: Typical relationship between capacity and error. Training and test error behave differently. At the left end of the graph, training error and generalization error are both high. This is the underfitting regime. As we increase capacity, training error decreases, but the gap between training and generalization error increases. Eventually, the size of this gap outweighs the decrease in training error, and we enter the overfitting regime, where capacity is too large, above the optimal capacity.

The concept of non-parametric models. So far, we have seen only parametric models, such as linear regression. Parametric models learn a function described by a parameter vector whose size is finite and fixed before any data is observed. Non-parametric models have no such limitation. Sometimes, non-parametric models are just theoretical abstractions (such as an algorithm that searches over all possible probability distributions) that cannot be implemented in practice. However, we can also design practical non-parametric models by making their complexity a function of the training set size. One example of such an algorithm is nearest neighbor regression. Unlike linear regression, which has a fixed-length vector of weights, the nearest neighbor regression model simply stores the \( X \) and \( y \) from the training set. When asked to classify a test point \( x \), the model looks up the nearest entry in the training set and returns the associated regression target. In other words, \( \hat{y} = y_i \) where \( i = \arg \min ||X_i - x||^2 \).

The algorithm can also be generalized to distance metrics other than the \( L^2 \) norm, such as learned distance metrics (Goldberger et al., 2005). If the algorithm is allowed to break ties by averaging the \( y_i \) values for all \( X_i \) that are tied for nearest, then this algorithm is able to achieve the minimum possible training error (which might be greater than zero, if two identical inputs are associated with different outputs) on any regression dataset.
Training Set Size

Figure 5.4: The effect of the training dataset size on the train and test error, as well as on the optimal model capacity. We constructed a synthetic regression problem based on adding a moderate amount of noise to a degree-5 polynomial, generated a single test set, and then generated several different sizes of training set. For each size, we generated 40 different training sets in order to plot error bars showing 95 percent confidence intervals.

(Top) The MSE on the training and test set for two different models: a quadratic model, and a model with degree chosen to minimize the test error. Both are fit in closed form. For the quadratic model, the training error increases as the size of the training set increases. This is because larger datasets are harder to fit. Simultaneously, the test error decreases, because fewer incorrect hypotheses are consistent with the training data. The quadratic model does not have enough capacity to solve the task, so its test error asymptotes to a high value. The test error at optimal capacity asymptotes to the Bayes error. The training error can fall below the Bayes error, due to the ability of the training algorithm to memorize specific instances of the training set. As the training size increases to infinity, the training error of any fixed-capacity model (here, the quadratic model) must rise to at least the Bayes error.

(Bottom) As the training set size increases, the optimal capacity (shown here as the degree of the optimal polynomial regressor) increases. The optimal capacity plateaus after reaching sufficient complexity to solve the task.
Weight Decay

Underfitting (Excessive \( \lambda \))

Appropriate weight decay (Medium \( \lambda \))

Overfitting (\( \lambda \rightarrow 0 \))

Figure 5.5: We fit a high-degree polynomial regression model to our example training set from figure 5.2. The true function is quadratic, but here we use only models with degree 9. We vary the amount of weight decay to prevent these high-degree models from overfitting. (Left) With very large \( \lambda \), we can force the model to learn a function with no slope at all. This underfits because it can only represent a constant function. (Center) With a medium value of \( \lambda \), the learning algorithm recovers a curve with the right general shape. Even though the model is capable of representing functions with much more complicated shape, weight decay has encouraged it to use a simpler function described by smaller coefficients. (Right) With weight decay approaching zero (i.e., using the Moore-Penrose pseudoinverse to solve the underdetermined problem with minimal regularization), the degree-9 polynomial overfits significantly, as we saw in figure 5.2.
Bias and Variance

The MSE measures the overall expected deviation—in a squared error sense—between the estimator and the true value of the parameter. As is clear from equation 5.54, evaluating the MSE incorporates both the bias and the variance. Desirable estimators are those with small MSE and these are estimators that manage to keep both their bias and variance somewhat in check.

![Image of Bias and Variance graph]

Figure 5.6: As capacity increases (x-axis), bias (dotted) tends to decrease and variance (dashed) tends to increase, yielding another U-shaped curve for generalization error (bold curve). If we vary capacity along one axis, there is an optimal capacity, with underfitting when the capacity is below this optimum and overfitting when it is above. This relationship is similar to the relationship between capacity, underfitting, and overfitting, discussed in section 5.2 and figure 5.3.

The relationship between bias and variance is tightly linked to the machine learning concepts of capacity, underfitting and overfitting. In the case where generalization error is measured by the MSE (where bias and variance are meaningful components of generalization error), increasing capacity tends to increase variance and decrease bias. This is illustrated in figure 5.6, where we see again the U-shaped curve of generalization error as a function of capacity.

5.4.5 Consistency

So far we have discussed the properties of various estimators for a training set of fixed size. Usually, we are also concerned with the behavior of an estimator as the amount of training data grows. In particular, we usually wish that, as the number of data points \( m \) in our dataset increases, our point estimates converge to the true value.
Decision Trees

Figure 5.7: Diagrams describing how a decision tree works.

(Top) Each node of the tree chooses to send the input example to the child node on the left (0) or the child node on the right (1). Internal nodes are drawn as circles and leaf nodes as squares. Each node is displayed with a binary string identifier corresponding to its position in the tree, obtained by appending a bit to its parent identifier (0=choose left or top, 1=choose right or bottom).

(Bottom) The tree divides space into regions. The 2D plane shows how a decision tree might divide $\mathbb{R}^2$. The nodes of the tree are plotted in this plane, with each internal node drawn along the dividing line it uses to categorize examples, and leaf nodes drawn in the center of the region of examples they receive. The result is a piecewise-constant function, with one piece per leaf. Each leaf requires at least one training example to define, so it is not possible for the decision tree to learn a function that has more local maxima than the number of training examples.
Principal Components Analysis

Figure 5.8: PCA learns a linear projection that aligns the direction of greatest variance with the axes of the new space.

(Left) The original data consists of samples of $x$. In this space, the variance might occur along directions that are not axis-aligned.

(Right) The transformed data $z = x > W$ now varies most along the axis $z_1$. The direction of second most variance is now along $z_2$.

This representation that has lower dimensionality than the original input. It also learns a representation whose elements have no linear correlation with each other. This is a first step toward the criterion of learning representations whose elements are statistically independent. To achieve full independence, a representation learning algorithm must also remove the nonlinear relationships between variables.

PCA learns an orthogonal, linear transformation of the data that projects an input $x$ to a representation $z$ as shown in figure 5.8. In section 2.12, we saw that we could learn a one-dimensional representation that best reconstructs the original data (in the sense of mean squared error) and that this representation actually corresponds to the first principal component of the data. Thus we can use PCA as a simple and effective dimensionality reduction method that preserves as much of the information in the data as possible (again, as measured by least-squares reconstruction error). In the following, we will study how the PCA representation decorrelates the original data representation $X$.

Let us consider the $m \times n$-dimensional design matrix $X$. We will assume that the data has a mean of zero, $\mathbb{E}[x] = 0$. If this is not the case, the data can easily be centered by subtracting the mean from all examples in a preprocessing step.

The unbiased sample covariance matrix associated with $X$ is given by:

$$
\text{Var}[x] = \frac{1}{m-1} X > X.
$$

(5.85)
Chapter 5. Machine Learning Basics

Figure 5.9: As the number of relevant dimensions of the data increases (from left to right), the number of configurations of interest may grow exponentially.

(Left) In this one-dimensional example, we have one variable for which we only care to distinguish 10 regions of interest. With enough examples falling within each of these regions (each region corresponds to a cell in the illustration), learning algorithms can easily generalize correctly.

(A Center) With 2 dimensions it is more difficult to distinguish 10 different values of each variable. We need to keep track of up to 100 regions, and we need at least that many examples to cover all those regions.

(Right) With 3 dimensions this grows to 1000 regions and at least that many examples. For d dimensions and v values to be distinguished along each axis, we seem to need $O(v^d)$ regions and examples. This is an instance of the curse of dimensionality. Figure graciously provided by Nicolas Chapados.

The curse of dimensionality arises in many places in computer science, and especially so in machine learning. One challenge posed by the curse of dimensionality is a statistical challenge. As illustrated in figure 5.9, a statistical challenge arises because the number of possible configurations of $x$ is much larger than the number of training examples. To understand the issue, let us consider that the input space is organized into a grid, like in the figure. We can describe low-dimensional space with a low number of grid cells that are mostly occupied by the data. When generalizing to a new data point, we can usually tell what to do simply by inspecting the training examples that lie in the same cell as the new input. For example, if estimating the probability density at some point $x$, we can just return the number of training examples in the same unit volume cell as $x$, divided by the total number of training examples. If we wish to classify an example, we can return the most common class of training examples in the same cell. If we are doing regression we can average the target values observed over the examples in that cell. But what about the cells for which we have seen no example? Because in high-dimensional spaces the number of configurations is huge, much larger than our number of examples, a typical grid cell has no training example associated with it. How could we possibly say something...
Nearest Neighbor

Figure 5.10
An important concept underlying many ideas in machine learning is that of a manifold. Mathematically, it is a set of points, associated with a neighborhood around each point. From any given point, the manifold locally appears to be a Euclidean space. In everyday life, we experience the surface of the world as a 2-D plane, but it is in fact a spherical manifold in 3-D space.

The definition of a neighborhood surrounding each point implies the existence of transformations that can be applied to move on the manifold from one position to a neighboring one. In the example of the world’s surface as a manifold, one can walk north, south, east, or west.

Although there is a formal mathematical meaning to the term “manifold,” in machine learning it tends to be used more loosely to designate a connected set of points that can be approximated well by considering only a small number of degrees of freedom, or dimensions, embedded in a higher-dimensional space. Each dimension corresponds to a local direction of variation. See figure 5.11 for an example of training data lying near a one-dimensional manifold embedded in two-dimensional space. In the context of machine learning, we allow the dimensionality of the manifold to vary from one point to another. This often happens when a manifold intersects itself. For example, a figure eight is a manifold that has a single dimension in most places but two dimensions at the intersection at the center.

Figure 5.11: Data sampled from a distribution in a two-dimensional space that is actually concentrated near a one-dimensional manifold, like a twisted string. The solid line indicates the underlying manifold that the learner should infer.
Uniformly Sampled Images

Figure 5.12
QMUL Dataset

Figure 5.13