

Here is a more troublesome difference: if you pick two points randomly in a unit square, the distance between these two points will be, on average, roughly 0.52. If you pick two random points in a unit 3D cube, the average distance will be roughly 0.66. But what about two points picked randomly in a 1,000,000-dimensional hypercube? Well, the average distance, believe it or not, will be about 408.25 (roughly $\sqrt{1,000,000/6}$)! This is quite counterintuitive: how can two points be so far apart when they both lie within the same unit hypercube? This fact implies that high-dimensional datasets are at risk of being very sparse: most training instances are likely to be far away from each other. Of course, this also means that a new instance will likely be far away from any training instance, making predictions much less reliable than in lower dimensions, since they will be based on much larger extrapolations. In short, the more dimensions the training set has, the greater the risk of overfitting it.

In theory, one solution to the curse of dimensionality could be to increase the size of the training set to reach a sufficient density of training instances. Unfortunately, in practice, the number of training instances required to reach a given density grows exponentially with the number of dimensions. With just 100 features (much less than in the MNIST problem), you would need more training instances than atoms in the observable universe in order for training instances to be within 0.1 of each other on average, assuming they were spread out uniformly across all dimensions.

Main Approaches for Dimensionality Reduction

Before we dive into specific dimensionality reduction algorithms, let's take a look at the two main approaches to reducing dimensionality: projection and Manifold Learning.

Projection

In most real-world problems, training instances are *not* spread out uniformly across all dimensions. Many features are almost constant, while others are highly correlated (as discussed earlier for MNIST). As a result, all training instances actually lie within (or close to) a much lower-dimensional *subspace* of the high-dimensional space. This sounds very abstract, so let's look at an example. In [Figure 8-2](#) you can see a 3D dataset represented by the circles.

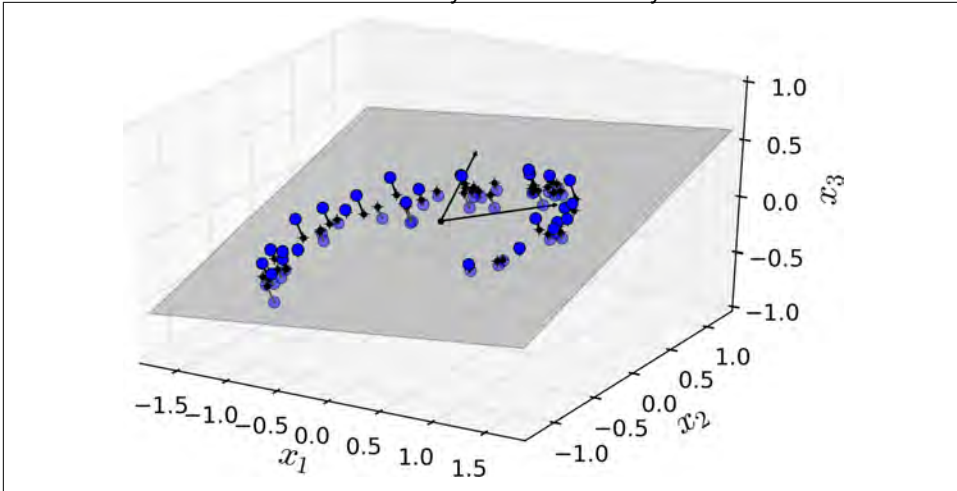


Figure 8-2. A 3D dataset lying close to a 2D subspace

Notice that all training instances lie close to a plane: this is a lower-dimensional (2D) subspace of the high-dimensional (3D) space. Now if we project every training instance perpendicularly onto this subspace (as represented by the short lines connecting the instances to the plane), we get the new 2D dataset shown in [Figure 8-3](#). Ta-da! We have just reduced the dataset's dimensionality from 3D to 2D. Note that the axes correspond to new features z_1 and z_2 (the coordinates of the projections on the plane).

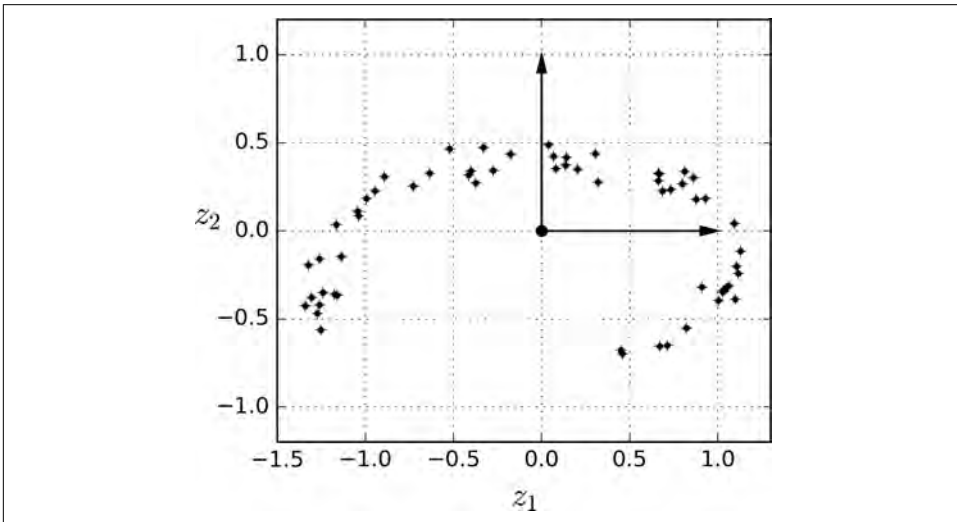


Figure 8-3. The new 2D dataset after projection

However, projection is not always the best approach to dimensionality reduction. In many cases the subspace may twist and turn, such as in the famous *Swiss roll* toy dataset represented in Figure 8-4.

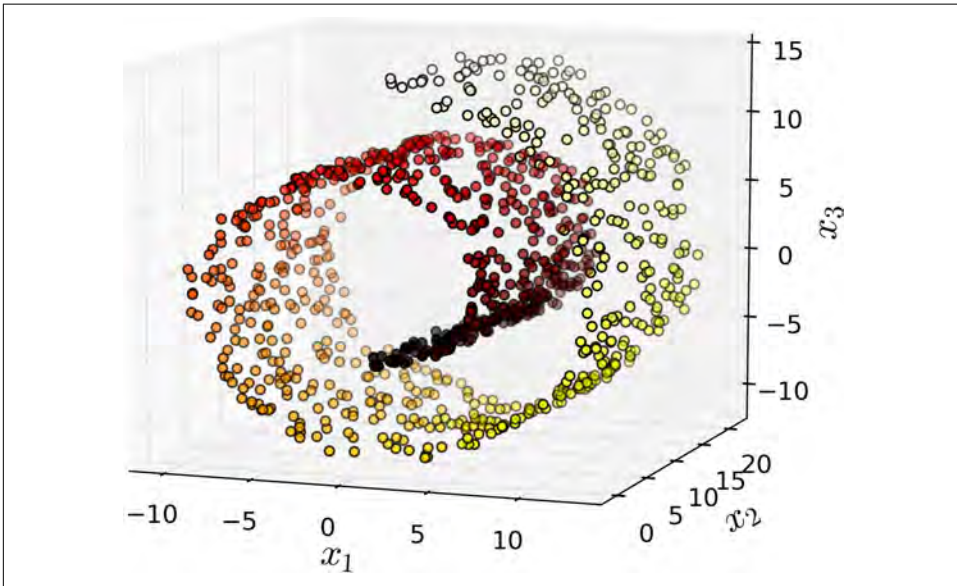


Figure 8-4. *Swiss roll dataset*

Simply projecting onto a plane (e.g., by dropping x_3) would squash different layers of the Swiss roll together, as shown on the left of Figure 8-5. However, what you really want is to unroll the Swiss roll to obtain the 2D dataset on the right of Figure 8-5.

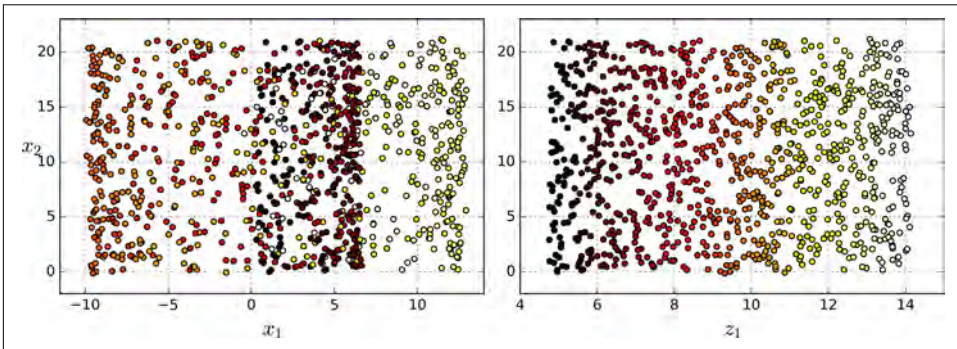


Figure 8-5. *Squashing by projecting onto a plane (left) versus unrolling the Swiss roll (right)*

Manifold Learning

The Swiss roll is an example of a 2D *manifold*. Put simply, a 2D manifold is a 2D shape that can be bent and twisted in a higher-dimensional space. More generally, a d -dimensional manifold is a part of an n -dimensional space (where $d < n$) that locally resembles a d -dimensional hyperplane. In the case of the Swiss roll, $d = 2$ and $n = 3$: it locally resembles a 2D plane, but it is rolled in the third dimension.

Many dimensionality reduction algorithms work by modeling the *manifold* on which the training instances lie; this is called *Manifold Learning*. It relies on the *manifold assumption*, also called the *manifold hypothesis*, which holds that most real-world high-dimensional datasets lie close to a much lower-dimensional manifold. This assumption is very often empirically observed.

Once again, think about the MNIST dataset: all handwritten digit images have some similarities. They are made of connected lines, the borders are white, they are more or less centered, and so on. If you randomly generated images, only a ridiculously tiny fraction of them would look like handwritten digits. In other words, the degrees of freedom available to you if you try to create a digit image are dramatically lower than the degrees of freedom you would have if you were allowed to generate any image you wanted. These constraints tend to squeeze the dataset into a lower-dimensional manifold.

The manifold assumption is often accompanied by another implicit assumption: that the task at hand (e.g., classification or regression) will be simpler if expressed in the lower-dimensional space of the manifold. For example, in the top row of [Figure 8-6](#) the Swiss roll is split into two classes: in the 3D space (on the left), the decision boundary would be fairly complex, but in the 2D unrolled manifold space (on the right), the decision boundary is a simple straight line.

However, this assumption does not always hold. For example, in the bottom row of [Figure 8-6](#), the decision boundary is located at $x_1 = 5$. This decision boundary looks very simple in the original 3D space (a vertical plane), but it looks more complex in the unrolled manifold (a collection of four independent line segments).

In short, if you reduce the dimensionality of your training set before training a model, it will definitely speed up training, but it may not always lead to a better or simpler solution; it all depends on the dataset.

Hopefully you now have a good sense of what the curse of dimensionality is and how dimensionality reduction algorithms can fight it, especially when the manifold assumption holds. The rest of this chapter will go through some of the most popular algorithms.