

---

# Dimensionality Reduction

Many Machine Learning problems involve thousands or even millions of features for each training instance. Not only does this make training extremely slow, it can also make it much harder to find a good solution, as we will see. This problem is often referred to as the *curse of dimensionality*.

Fortunately, in real-world problems, it is often possible to reduce the number of features considerably, turning an intractable problem into a tractable one. For example, consider the MNIST images (introduced in [Chapter 3](#)): the pixels on the image borders are almost always white, so you could completely drop these pixels from the training set without losing much information. [Figure 7-6](#) confirms that these pixels are utterly unimportant for the classification task. Moreover, two neighboring pixels are often highly correlated: if you merge them into a single pixel (e.g., by taking the mean of the two pixel intensities), you will not lose much information.



Reducing dimensionality does lose some information (just like compressing an image to JPEG can degrade its quality), so even though it will speed up training, it may also make your system perform slightly worse. It also makes your pipelines a bit more complex and thus harder to maintain. So you should first try to train your system with the original data before considering using dimensionality reduction if training is too slow. In some cases, however, reducing the dimensionality of the training data may filter out some noise and unnecessary details and thus result in higher performance (but in general it won't; it will just speed up training).

Apart from speeding up training, dimensionality reduction is also extremely useful for data visualization (or *DataViz*). Reducing the number of dimensions down to two

(or three) makes it possible to plot a high-dimensional training set on a graph and often gain some important insights by visually detecting patterns, such as clusters.

In this chapter we will discuss the curse of dimensionality and get a sense of what goes on in high-dimensional space. Then, we will present the two main approaches to dimensionality reduction (projection and Manifold Learning), and we will go through three of the most popular dimensionality reduction techniques: PCA, Kernel PCA, and LLE.

## The Curse of Dimensionality

We are so used to living in three dimensions<sup>1</sup> that our intuition fails us when we try to imagine a high-dimensional space. Even a basic 4D hypercube is incredibly hard to picture in our mind (see [Figure 8-1](#)), let alone a 200-dimensional ellipsoid bent in a 1,000-dimensional space.

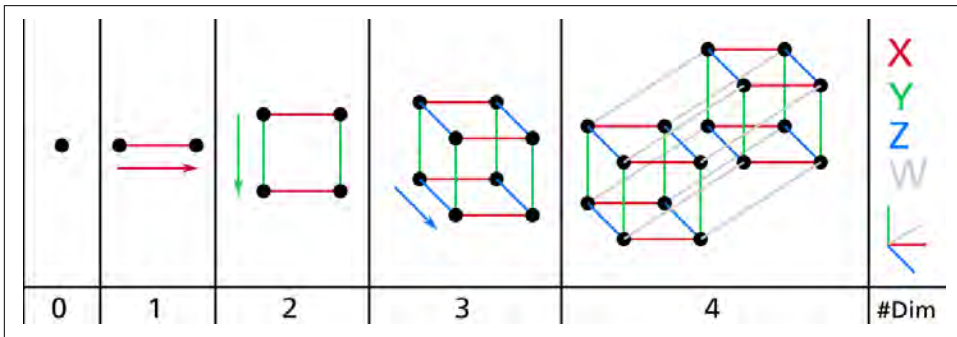


Figure 8-1. Point, segment, square, cube, and tesseract (0D to 4D hypercubes)<sup>2</sup>

It turns out that many things behave very differently in high-dimensional space. For example, if you pick a random point in a unit square (a  $1 \times 1$  square), it will have only about a 0.4% chance of being located less than 0.001 from a border (in other words, it is very unlikely that a random point will be “extreme” along any dimension). But in a 10,000-dimensional unit hypercube (a  $1 \times 1 \times \dots \times 1$  cube, with ten thousand 1s), this probability is greater than 99.999999%. Most points in a high-dimensional hypercube are very close to the border.<sup>3</sup>

<sup>1</sup> Well, four dimensions if you count time, and a few more if you are a string theorist.

<sup>2</sup> Watch a rotating tesseract projected into 3D space at <http://goo.gl/OM7ktJ>. Image by Wikipedia user Nerd-Boy1392 (Creative Commons BY-SA 3.0). Reproduced from <https://en.wikipedia.org/wiki/Tesseract>.

<sup>3</sup> Fun fact: anyone you know is probably an extremist in at least one dimension (e.g., how much sugar they put in their coffee), if you consider enough dimensions.