k-Nearest Neighbors and the Curse of Dimensionality

Why the k-nearest neighbors algorithm is especially sensitive to additional dimensions

Peter Grant Jul 23, 2019 · 6 min read ★

In my last article, we learned about the k-nearest neighbors modeling algorithm. That algorithm makes classification predictions about your desired data points based on the assumption that nearby data points are similar to your test point. In order to function it needs two things:

- First, it needs you to trust the assumption that it's nearby neighbors are similar to it. There are some cases where this will be true. For instance, pieces of produce in a grocery store are commonly sorted into similar groups. Gala apples are stored with

gala apples, which are stored next to fuji apples. It's pretty safe to assume that the item next to a gala apple is another gala apple, and very safe to assume that the neighboring item is an apple. However, in other cases the assumption that neighboring items are similar is not useful. Just because one item in my drunk drawer is a battery doesn't mean that the item next to it is also a battery. It very well could be a marker, a paperclip, or a rubber band.

- Second, you need to have a way to measure distance. Note that this is dimensional distance, not necessarily physical distance. It refers to the differs between the two data points you're trying to compare. In the above examples I used physical distance as the distance, but it could also be the difference in square footage of two houses, the difference in efficiency between two devices, and so on.

This means that your success when using the k-nearest neighbors algorithm is very dependent on having a dense data set. This makes it especially vulnerable to the "Curse of Dimensionality".

What is the "Curse of Dimensionality"?

The "Curse of Dimensionality" is a tongue in cheek way of stating that there's a ton of *space* in high-dimensional data sets. The size of the data space grows exponentially with the number of dimensions. This means that the size of your data set must also grow exponentially in order to keep the same density. If you don't, then data points start getting farther and farther apart.

Why is this especially problematic for k-nearest neighbors?

At face value it doesn't seem that k-nearest neighbors would be especially sensitive to this problem. Every machine learning algorithm needs a dense data set in order to accurately predict over the entire data space. Errors arise in all algorithms if there are gaps between the data. So what makes k-nearest neighbors special?

The special challenge with k-nearest neighbors is that it requires a point to be close in every single dimension. Some algorithms can create regressions based on single dimensions, and only need points to be close together along that axis. k-nearest neighbors doesn't work that way. It needs all points to be close along every axis in the data space. And each new axis added, by adding a new dimension, makes it harder and harder for two specific points to be close to each other in every axis.

Joel Grus does a good job of describing this issue in Data Science from Scratch. In that book he calculates the average and minimum distances between two points in a dimension space as the number of dimensions increases. He calculated 10,000 distances between points, with the number of dimensions ranging from 0 to 100. He then

proceeds to plot the average and minimum distance between two points, as well as the ratio of the closest distance to the average distance (Distance_Closest / Distance_Average).

In those plots, Joel showed that the ratio of the closest distance to the average distance increased from 0 at 0 dimensions, up to ~0.8 at 100 dimensions. And this shows the fundamental challenge of dimensionality when using the k-nearest neighbors algorithm; as the number of dimensions increases and the ratio of closest distance to average distance approaches 1 the predictive power of the algorithm decreases. If the nearest point is almost as far away as the average point, then it has only slightly more predictive power than the average point.

Think of how this problems applies to our apple example from above. It's a bit counter-intuitive, since the apple example is inherently in 3 dimensions, but imagine that the distance to the next closest item in the produce section is about the same as the average distance. Suddenly you can't be sure if the nearest item is another gala apple, or a fuji apple, or an orange, or a bunch of parsley. As far as the k-nearest neighbors algorithm would be concerned, the entire produce section would be jumbled together.

How do I overcome the curse of dimensionality when using the k-nearest neighbors algorithm?

The problem is fundamentally that there isn't enough data available for the number of dimensions. As the number of dimensions increases the size of the data space increases, and the amount of data needed to maintain density also increases. Without dramatic increases in the size of the data set, k-nearest neighbors loses all predictive power. And this makes one possible solution for the issue straightforward: Add more data. It's entirely possible to add more and more data to ensure that you have enough data density even as you add more dimensions. And if you have the hardware to handle that volume of data, it's a perfectly legitimate solution.

Of course, you can't always have the hardware necessary to add that much data. Not every data scientist can afford access to a supercomputer. And even then, it's possible to have a large enough data set that not even a super computer could handle it in a reasonable amount of time. This is where the concept of dimensionality reduction comes into play. Dimensionality reduction is a topic beyond the scope of this article, but I'll cover it in the next one. Essentially, it refers to identifying trends in the data set that operate along dimensions that are not explicitly called out in the data set. You can then create new dimensions matching those axes and remove the original axes, thus reducing the total number of axes in your data set.

As an example, say that you plot the location of garden gnomes in a city. You get the GPS co-ordinates of each garden gnome, and plot them on a map. You have the two dimensions of North-South and East-West. Now imagine that, for some reason, all of the garden gnomes are placed in a near diagonal line running across the town. There's one in the Southeast, one in the Northwest corner, and a nearly straight line between the two. Now you can create a new axis called the Southeast-Northeast axis (Or, if you want to be silly, "The Garden Gnome Demarcation Line") and remove the North-South and East-West axis. In this way you've reduced your data set from two dimensions two one, and made it easier to the k-nearest neighbors algorithm to succeed.

Wrapping It Up

The k-nearest neighbors algorithm hinges on data points being close together. This becomes challenging as the number of dimensions increases, referred to as the "Curse of Dimensionality." It's especially hard for the k-nearest neighbors algorithm it requires two points to be very close on *every* axis, and adding a new dimension creates another opportunity for points to be farther apart. As the number of dimensions increases, the closest distance between two points approaches the average distance between points, eradicating the ability of the k-nearest neighbors algorithm to provide valuable predictions.

To overcome this challenge, you can add more data to the data set. By doing so you add density to the data space, bringing the nearest points closer together and returning the ability of the k-nearest neighbors algorithm to provide valuable predictions. This is a valuable solution so long as you have the hardware needed to perform computations on your data set. As your data set gets larger and larger, you need more and more computing power to process it. Eventually the size of your data set will surpass your computing power. At that point, you need to use dimensionality reduction to present all of the valuable information in fewer dimensions. That topic will be described in more detail in a future article.

Machine Learning Towards Data Science Data Science Algorithms