Visualizing K-Means Clustering

January 19, 2014

Suppose you plotted the screen width and height of all the devices accessing this website. You'd probably find that the points form three clumps: one clump with small dimensions, (smartphones), one with moderate dimensions, (tablets), and one with large dimensions, (laptops and desktops). Getting an algorithm to recognize these clumps of points without help is called *clustering*. To gain insight into how common clustering techniques work (and don't work), I've been making some visualizations that illustrate three fundamentally different approaches. This post, the first in this series of three, covers the k-means algorithm. To begin, click an initialization strategy below:

How to pick the initial centroids?

I'll Choose

Randomly

Farthest Point

Restart

K-Means Algorithm

The k-means algorithm captures the insight that each point in a cluster should be near to the center of that cluster. It works like this: first we choose k, the number of clusters we want to find in the data. Then, the centers of those k clusters, called *centroids*, are initialized in some fashion, (discussed later).

The algorithm then proceeds in two alternating parts: In the *Reassign Points* step, we assign every point in the data to the cluster whose centroid is nearest to it. In the *Update Centroids* step, we recalculate each centroid's location as the mean (center) of all the points assigned to its cluster. We then iterate these steps until the centroids stop moving, or equivalently until the points stop switching clusters.

In the above visualization, the color of the small data points represents the cluster to which they have been assigned. The larger circles are the cluster centroids, with their color indicating the cluster that they represent. For illustrative purposes, I've also shaded subregions depending on which centroid they are closest to. This is called a Voronoi diagram.

In the Reassign Points step, for every point we determine which centroid is closest to it, and modify its color accordingly. In the Update Centroids step, the centroids move to the new mean of the points in their cluster, and I redraw the Voronoi diagram based on the new locations of the centroids. Until k-means converges, after each Update Centroids step there will be data points in the Voronoi cells belonging to a different cluster; these will be reassigned in the next Reassign Points step. When the algorithm converges, all the data points in each Voronoi cell will actually be assigned to the corresponding centroid, so that further Reassign Points steps and further Update Centroids steps will have no effect.

Properties of K-Means

It's not hard to prove that the k-means algorithm eventually converges. (Proof outline: The sum of squared distances between each point and its centroid strictly decreases in both the Reassign Points and Update Centroids steps, and there are only finitely many cluster configurations). Despite some contrived examples in which k-means takes exponential time to converge, as a matter of practice it converges reasonably quickly. Since the Reassign Points and Update Centroids steps each take linear time, the practical run time of k-means is basically linear, and exactly so if you limit the number of iterations.

Unfortunately, despite the fact that k-means is guaranteed to converge, the final cluster configuration to which it converges is not in general unique, and depends on the initial centroid locations. As a simple example of this, take a look at the "Gaussian Mixture" data, which consists of three clearly separate clumps. If you try to place four centroids, however, usually what will happen after you run k-means is that two of the point clumps will get one centroid each, and the final clump will get two centroids. Which clump it is that gets two centroids, however, depends on where you initialize the centroids to be. As a matter of practice, then, oftentimes people will try several initialization strategies, (or try a randomized initialization strategy multiple times), and pick the one that results in the best clustering.

One final interesting property of k-means, though one that isn't much talked about, is that it is actually possible for no data points to be assigned to a cluster in the Reassign Points step. For instance, if you initialize the centroids yourself, and place one centroid inside a ring of other centroids in some empty region, then the ring of centroids will "block off" the inner centroid from the data points. Consequently, no data points will be closest to the inner centroid, and so no points will be assigned to it. It can also happen, however, that no points are assigned to a centroid in a later Reassign Points step.

What happens in this situation is implementation dependent: In my visualization, the lonely centroid awkwardly stays where it is, and sometimes gets points assigned to it in a later Reassign Points step. Some implementations, however, delete the lonely centroid, and some randomly relocate it somewhere else. If an implementation does not specifically handle this situation, however, then it will occasionally fail with a 0/0 error when it tries to compute the lonely centroid's new location.

Initialization Strategies

So how do you initialize the centroids in k-means?

One straightforward way of doing it is to pick them randomly from among your data points. This is does not work particularly well, however, because it becomes overwhelmingly likely that many of initial centroids will end up in the same true cluster. Suppose you are lucky enough to pick k correctly, and that each of the k true clusters has about the same number of points. Then it's easy to show that the probability that the k centroids will each be initialized into unique clusters is about k!/k^k. By Stirling's Approximation, this is about e^-k sqrt(2 pi k), which goes to zero very fast.

A better idea, implemented in the "farthest" heuristic, is to initialize the first centroid randomly, but then to initialize the second centroid to be the data point farthest away from it. In general, we initialize the jth centroid to the point whose minimum distance to the preceding centroids is largest. This procedure initializes the centroids to be well spread-out from each other. Notice how well it works in the "Packed Circles" data, for example: With k=6, the farthest point heuristic will usually initialize each centroid into a different true cluster.

Perhaps an even better initialization strategy, (though not one implemented in the visualization), is called k-means++. It works similarly to the "farthest" heuristic, except that instead of choosing the jth centroid to be the point furthest from the preceding centroids, it selects a point with probability proportional to the square of its distance to the nearest preceding centroid. This makes some sense: If you pick the farthest point, you often get points that are at the edges of their true clusters, but if you choose randomly as described, then you're more likely to get one near the center of the true cluster. (Indeed, the authors even proved that this initialization results in a clustering that in expectation is suboptimal by at most a log(k) factor)!

Of course, there is still the problem of choosing k, the number of clusters. Usually, you don't know beforehand how many clusters the data contains, and usually you can't look at the data directly because it lies in a higher dimension than two or three. (Indeed, if you can look at your data and see obvious clusters like you can here, you may be better off clustering manually). So in practice, people often try different values of k and see how their results vary.

K-Means Performance

K-Means works best in datasets that have with clusters that are roughly equally-sized and shaped roughly regularly. So it works very well on the "Gaussian Mixture" data and the "Packed Circles" data if you the "Farthest" heuristic and the right number of centroids.

However, it is not able to capture the four different clusters of points in the "Smiley" and "Pimpled Smiley" datasets, due to the unusual shape of the face outline. The requirement of choosing k, and the dependence of the output on the initial cluster configuration, is also annoying. And k-means can only be applied when the data points lie in a Euclidean space, failing for more complex types of data.

Despite these disadvantages, the k-means algorithm is a major workhorse in clustering analysis: It works well on many realistic data sets, and is relatively fast, easy to implement, and easy to understand. Many clustering algorithms that improve on or generalize k-means, such as k-medians, k-medoids, k-means++, and the EM algorithm for Gaussian mixtures, all reflect the same fundamental insight, that points in a cluster ought to be close to the center of that cluster.

Next: Try out the DBSCAN algorithm on these datasets.

Post RSS

You might also enjoy...

- Visualizing DBSCAN Clustering
- Visualizing the James-Stein Estimator
- Markov Chain Implications Graph