Clustering & Dimension Reduction

Alex Sverdlov
alex@theparticle.com

1 Clustering

Clustering is a mechanism to group similar items together—primarily so we could work with groups instead of individual samples. There are two problems with this: what does similarity mean, and what exactly is a group? We need to define *both*.

The choices are often domain specific, for example, clustering locations might use Euclidean distance, but clustering financial instruments like Bonds might require some heuristic 'distance' measures. If distance measure is not appropriate to the task, the clusters will often be meaningless.

Next we need to decide how clusters will be formed, and how many we want. If we have one unlabeled sample, then obviously it is in the group of one, all by itself. There really is no need for such clusters. If we have *two* samples, we suddenly have a choice to make: do we create one cluster or two? It turns out that this choice is arbitrary—if we prefer to have one cluster, we create one cluster. Or we could create two.

1.1 k-Means

k-Means is perhaps the most popular clustering algorithm. It was first introduced by Hugo Steinhaus in 1956 [**?**], and later revised and extended by [**?**, **?**, **?**]. The algorithm is designed to cluster data into k buckets, usually based on Cartesian distance between input samples. It is an example of an *Expectation Maximization* algorithm, which is a general class of two step algorithms, where first step projects our expectations into the model, and second step adjusts the model so as to maximize our projections.

The algorithm starts with k random means. During each iteration, each training sample gets the label of the closest mean. Once all training samples are labeled, we recalculate k means using the samples assigned with each label. This repeats while training samples are changing labels from iteration to iteration.

The output of this procedure is obviously the k mean points. To find a cluster for any new input sample, we compute its distance to every mean, and assign it to the closest one.

1.2 Hierarchical Clustering

Hierarchical clustering looks for a whole hierarchy of clusters. This can be Agglomerative or Divisive.

Agglomerative clustering starts with each sample point in a cluster of its own. Each iteration merges two closest points or clusters (to form bigger clusters). The end product is one cluster that contains everything—along with the sequence of what falls into what cluster when.

Divisive clustering is the reverse of Agglomerative: starts off with one big cluster, and starts splitting off farthest things out.

1.3 Manifold Clustering

Traditional clustering method implicitly assume that clusters are centered around points, and even when customized, distance measures often assume coordinate independence. When trying to cluster abstract things like Bonds, for example, some of those assumptions may not apply. For example, for bonds, time to maturity, yield, and current price are all related and coordinates have different scale and units. It is not unreasonable to assume that individual bonds sit on some higher dimensional structure—not a single point—that represents relationships among different bonds.

Manifold clustering attempts to discover these higher dimensional structures. The methods and the types of manifolds detected differ.

Linear manifold clustering [**?**] attempts to find linear manifolds by sampling a certain number of points (for example, 3 for a plane), constructing the manifold, and then seeing if other points are also on the manifold. Using the distance-to-manifold histogram for all samples, this method determines if this really is a genuine manifold (the histogram indicates a separation of points on the manifold and not on the manifold).

Other methods to find manifolds involve looking for sparse manifolds near sample points. The idea is that neighboring sample points are very likely part of the same manifold and can be linked (with a weight). This appears to work for sparse manifolds without much clutter. [**?**]

2 Dimension Reduction

Imagine we take a picture of a 3D room—we get a million pixels. We now have a million dimensional "sample" of that 3D room. We can take a great many of such pictures of the same room—each one a million dimensional sample—all representing the same 3D room. It quickly becomes clear that all these pictures are really representing a 3D room (that is what we are photographing), and the variation is in lighting, direction of camera, etc., perhaps adding a few more dimensions—but certainly not a million. A great many problems have this ridiculously high dimensionality.

2.1 Principal Component Analysis

Perhaps the simplest and most overused method of dimension reduction is PCA, or Principal Component Analysis. Intuitively, principle components represent directions of variation. If we had a point cloud the shape of an average car, the first principle component would be along the length of the car (the longest dimension). The second principle component would be along the height or width, depending which was the second longest dimension of the car. For a 3D car, we would find three principle components—even if the original dataset had 500 dimensions. PCA is an automated mechanism to find such dimensions—for arbitrary data.

Practically this often means taking high dimensional data, and arranging it as columns of a matrix. If we are dealing with pictures of faces, where each picture is perhaps 256x256 pixels, then each column of a matrix will be 65536 numbers, and there will be a column for every training picture. A huge matrix. Applying PCA on such a matrix will result in a set of orthogonal vectors, that are called eigenvectors. For the 'pictures of faces', such eigenvectors (these are 65536 numbers, or 256x256 'images') represent typical components of faces—all other faces are linear combinations of these eigenfaces. The key here is that there are only perhaps a dozen such eigenfaces. To compare two face images, we project both of them onto these few eigenfaces, and compare distance within this low dimensional space. This is exactly the method used by Sirovich & Kirby [**?**], and Turk & Pentland [**?**] for face recognition, and it turned out that faces are relatively low dimensional—the Turk paper only used 7 eigenfaces.

The PCA approach is a lot more general than face recognition. It is a by product of one of the most useful matrix factorization methods ever, the Singular Value Decomposition: $X = U\Sigma V^*$