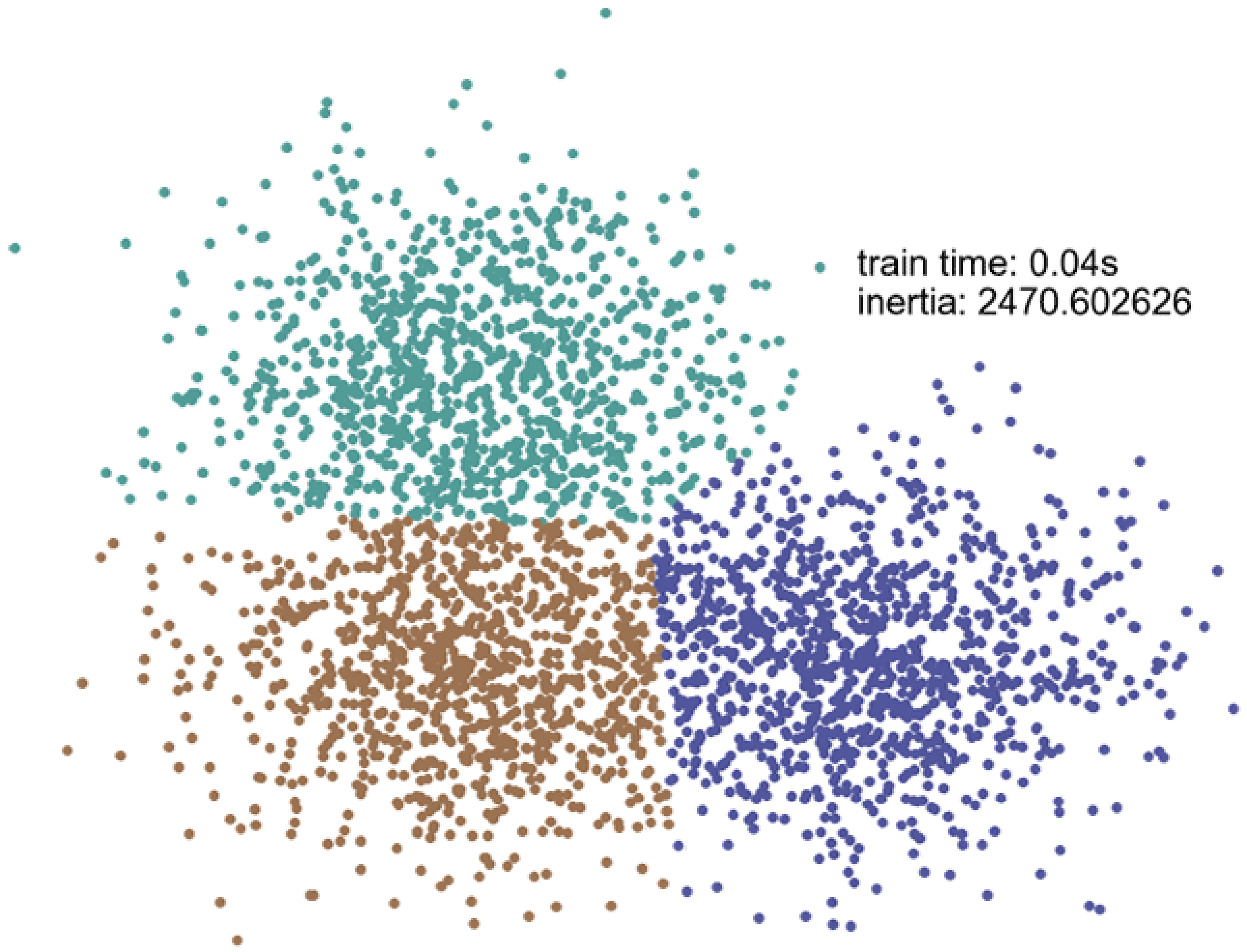
Kevin Davenport

Machine Learning & Statistics Blog

The Cost Function of K-Means

February 14, 2014



When exploring a novel dataset, I believe most analysts will run through the familiar steps of generating summary statistics and/or plotting distributions and feature interactions. Clustering and PCA are a great way to continue assessing data as they can explain “natural” grouping or ordering as well as attributing variance to certain features. When working with large datasets you’ll have to start considering scalability and complexity of the algorithm underlying a given method, that is to say that even if you were merely a “consumer of statistics”, you now have to consider the math. The many clustering approaches available today belong to either nonparametric/hierarchical or parametric methods, with differentiation in the former being agglomerative (bottom-up) versus divisive (top-down) and reconstructive versus generative methods for the latter.

A strength of the K-Means clustering algorithm (Parametric-Reconstructive) is it's efficiency (Big O Notation) with $O(nkt)$ where **n**, **k**, **t** equal the number of iterations, clusters, and data points. A potential strength or weakness is that K-Means converges to a local optimum which is easier than solving for the global optimum but can lead to less optimal convergence. Well known weaknesses of K-Means include required cluster (k) specification and sensitivity to initialization, however both have many options for mitigation (x-means, k-means ++, PCA, etc.)

The steps of K-Means tend to be intuitive, after random initialization of cluster centroids, the algorithm's inner-loop iterates over two steps:

1. Assign each observation x_i to the closest cluster centroid μ_j
2. Update each centroid to the mean of the points assigned to it.

Logically, K-Means attempts to minimize distortion defined by the the sum of the squared distances between each observation and its closest centroid. The stopping criterion is the change in distortion, once the change is lower than a pre-established threshold, the algorithm ends. In sci-kit learn the default threshold is 0.0001. Alternatively a max number of iterations can be set prior to initialization. More interesting approaches to threshold determination are outlined in Kulls and Jordan's paper: New Algorithms via Bayesian Nonparametrics.

$$\mathcal{L}(\Delta) = \sum_{i=1}^{n} ||x_i - \mu_{k(i)}||^2 = \sum_{k=1}^{K} \sum_{i \in C_k} ||x_i - \mu_k||^2$$

The notion isn't mentioned much in practical use, but K-Means is fundamentally a coordinate descent algorithm. Coordinate descent serves to minimize a multivariate function along one direction at a time. The inner-loop of k-means repeatedly minimizes the function with respect to k while holding μ fixed, and then minimizes with respect to μ while holding k fixed. This means the function must monotonically decrease and that values must converge. This distortion function is a non-convex function, meaning the coordinate descent is not guaranteed to converge to the global minimum and the algorithm can be susceptible to local optima. This is why it is optimal to run k-means many times using random initialization values for the clusters, then selecting the run with lowest distortion or cost.

Since there isn't a general theoretical approach to find the optimal number of k for a given data set, a simple approach is to compare the results of multiple runs with different k classes and choose the best one according to a given criterion (Elbow, BIC, Schwarz Criterion, etc.). Any approach must be taken with caution as increasing k results in smaller error function values, but also increases the chance of overfitting.

If you haven't already, I recommend taking a look at Mini-Batch K-Means and K-Means++ leveraging the built-in joblib parallelization functions (n_jobs). The joblib dispatch method works on K-Means by breaking down the pairwise matrix into n_jobs even slices and computing them in parallel.

Categorized in: Python