Clustering

1 Description

Given some data and a similarity or distance function d, the objective in clustering is to obtain a partitioning function P_d which assigns data points to different groups or clusters. As such, we have that if $P_d(\mathbf{x}_i) = P_d(\mathbf{x}_j)$, then $\mathbf{x}_i$ and $\mathbf{x}_j$ are in the same cluster.

Two trivial partitioning functions would be to assign all points to the same cluster, or to have each point be its own partition. In both cases, no useful information is conveyed. Thus, a desirable property of clustering algorithms is to be somewhere in between those two extremes. Different algorithms are described in the sections below.

2 Single Linkage Clustering (SLC)

We initialize the algorithm by assigning each data point to its own cluster. We then repeatedly compute the Inter-Cluster Distance (ICD) between all clusters, and merge the two clusters with minimum ICD. We repeat these two steps $N - k$ times to obtain k clusters, where N is the number of data points.

The ICD is defined as the minimum distance between any two points in each cluster, thus

$$\text{ICD}(c_i, c_j) = \min_{k,l} d(\mathbf{x}_k, \mathbf{x}_l), \quad \mathbf{x}_k \in c_i, \quad \mathbf{x}_l \in c_j$$

where c_i and c_j are two different clusters.

Due to it being a hierarchical agglomerative clustering algorithm, the mergers at each iteration can be used to construct a "dendrogram", which is useful for visualizing how the data is related.

SLC has a finite number of possible steps, namely 1 to $N - 1$. With 1 step, we have the extreme case of each data point being its own cluster. With $N - 1$ steps, we have the opposite extreme case of all points belonging to the same cluster. Thus, one must choose an appropriate number of clusters to obtain any useful information.

In addition to ICD, other alternatives can be used, such as the mean or the median over all pairwise distances.

The running time of SLC is $O(N^3)$, due to the fact that k is bound by N and we iterate $N - k$ times, and for each iteration we need to compare distances between all pairs of points totaling $N(N - 1)$ pairs.

Due to the fact that SLC is greedy in the sense that once a point is assigned to a cluster after the first iteration it cannot be reassigned to a different cluster, SLC can generate

"undesirable" or "unnatural" clusters in certain situations. To address this, algorithms which ca re-assign clusters are necessary. k-means clustering is one such algorithm, and is described in the next section.

3 k-means

We initialize the algorithm by picking k random data points, which will be the initial centroids of our clusters. We then repeatedly assign each data point to the nearest centroid, and recompute the centroids based on these new assignments. The algorithm terminates when centroids no longer change.

More formally, we have that the initial centroids c_i at $t = 0$ are

$$\{c_1^0 = \mathbf{x}_{j_1}, c_2^0 = \mathbf{x}_{j_2}, \ldots, c_k^0 = \mathbf{x}_{j_k}\}$$

where $j_1, \ldots, j_k$ are distinct and uniformly drawn from $\{1, \ldots, N\}$, with N being the number of data points. We then assign each data point $\mathbf{x}_j$ to the nearest centroid like so

$$P_d^{t+1}(\mathbf{x}_j) \leftarrow \arg\min_i d(c_i^t, \mathbf{x}_j)$$

Afterwards we recompute the centroids based on the new assignments like so

$$c_i^t \leftarrow \frac{1}{|c_i^t|} \sum_{x_j \in c_i^t} \mathbf{x}_j$$

We then repeat the two previous steps until $c_i^{t+1} = c_i^t$ for all c_i at some timestep $t + 1$.

These steps can also be thought of as a repeated execution of two operations: the computation of centroids followed by the assignment of points to such centroids. At the end of each iteration, the assignment of points provides a new partitioning function. One observation is that in the first iteration, centroids are computed simply by the random assignment of points. Another observation is that, if the steps are viewed in this way, the last iteration would be interrupted after computing centroids since it would be observed that they have not changed. Thus, the final partitioning function is the one obtained in the previous iteration.

If we define a cost function as such

$$\sum_{x_j} \| c_{P_d^t(\mathbf{x}_j)}^t - \mathbf{x}_j \|^2$$

in other words, the sum of squared differences between points and their cluster centroids – we can think of k-means as a local optimization algorithm. In this context, we can think of P_d and all c_i's as parameters for which we are trying to minimize the cost. The neighborhood of any given P_d and c_i's would be

$$\left\{ \left(P_d(\mathbf{x}_j) = \arg\min_i d(c_i, \mathbf{x}_j), c_1, \ldots, c_k \right), \left(P_d, c_1 = \frac{1}{|c_1|} \sum_{x_j \in c_1} \mathbf{x}_j, \ldots, c_k = \frac{1}{|c_k|} \sum_{x_j \in c_k} \mathbf{x}_j \right) \right\}$$

In other words, a pair where the first element has an updated partitioning function based on the centroids, and the second element has updated centroids based on the partitioning function. Thus, only either the partitioning function or the centroids change.

We can then note that each operation in each iteration is moving the parameters to either element of the above pair. Additionally, each operation can in fact only decrease the error: centroids are computed as the mean of points in a cluster over all features, and such a mean is in fact what minimizes the distance to each point in the cluster; and points are only reassigned if the distance to another centroid is shorter than the distance to the centroid of the cluster they currently belong to. Thus, at each iteration the error monotonically decreases.

Due to this, it can be shown that k-means always converges. Since the error decreases or stays the same, it will eventually reach a local minima, and as long as ties are broken consistently so that the algorithm doesn't flip-flop between two configurations with the same local minima, no previous configuration will be revisited apart from the final optimal configuration which indicates we should stop.

This also shows that k-means can terminate at a local optima which may not be the global optima. As with other randomized optimization algorithms, a solution to this is randomized restarts with different initial cluster choices.

For any given data and choice for k, there are $O(k^N)$ possible configurations of clusters. However, in practice k-means takes a lot fewer iterations to converge since the vast majority of configurations are never visited. Ignoring the time it takes to perform computations, each iteration takes $O(k \cdot N)$ time, since it takes $O(N)$ time to recompute the centroids and $O(k \cdot N)$ time to reassign points to clusters.

4 Expectation Maximization (EM)

Sometimes it may be useful to assign probabilities for each point belonging in a cluster rather than strictly assigning each point to a single cluster. This is called "soft clustering", and EM is an algorithm which can be used to derive such a clustering.

To accomplish this, we assume that there are K distributions which generated the data, and that each data point was generated by picking one of these distributions uniformly at random, and then sampling from the chosen distribution. This is known as a "mixture model", and if all distributions are Gaussian, it is called a Gaussian mixture model (GMM).

Before utilizing EM, we must introduce hidden variables to each data point which will denote which distribution generated the point. Namely, for each $\mathbf{x}_i$, we will have

$$[x_i^{(1)}, \ldots, x_i^{(D)}, z_i^{(1)}, \ldots, z_i^{(K)}]$$

where each $z_i^{(k)}$ represents the probability that $\mathbf{x}_i$ was generated by distribution k.

We then initialize the algorithm with random values for the different attributes of each of the K distributions (μ and Σ in the case of Gaussians, for example).

Afterwards, we repeatedly compute the expectations of the hidden variables, and update the attributes of the distributions from those expectations. These two steps are known as "E" (corresponding to expectation) and "M" (corresponding to maximization).

The E step derives the expectations for the probabilities of the data points belonging to each distribution (i.e. the expectations of the hidden variables). In the case of a GMM, for example, it would compute

$$E[z_i^{(k)}]^{t+1} \leftarrow \frac{p(\mathbf{x}_i \mid \mu_k^t, \sigma_k^t)}{\sum_{j=1}^{K} p(\mathbf{x}_i \mid \mu_j^t, \sigma_j^t)}$$

where the denominator serves to normalize the probabilities of a point belonging to each cluster so that they sum up to 1.

The M step recomputes the attributes of the distributions which maximize the probabilities of the data points based on the new expectations. In the case of a GMM, for example, it would compute

$$\mu_k^t = \frac{\sum_{i=1}^{N} E[z_i^{(k)}]^t \mathbf{x}_i}{\sum_{i=1}^{N} E[z_i^{(k)}]^t}, \quad \Sigma_k^t = \frac{\sum_{i=1}^{N} E[z_i^{(k)}]^t (\mathbf{x}_i - \mu_k^t)(\mathbf{x}_i - \mu_k^t)^\top}{\sum_{i=1}^{N} E[z_i^{(k)}]^t}$$

where the denominators again are normalizing terms.

These two steps have a strong resemblance to the k-means steps, except we are performing probabilistic assignments to clusters instead. If fact, if we only allowed $E[z_i^{(k)}] \in \{0, 1\}$ (i.e. if $E[z_i^{(k)}]$ were binary), and if we ignored Σ_k (or considered it to be the identity matrix, for example), the two algorithms would be practically identical.