0.1 Algorithm complexity analysis

Let n be the number of data points, k be the number of clusters (number of means), d be the dimensionality of the data, and i be the number of iterations until convergence. The runtime of the k-means algorithm (including spherical k-means) is $O(nkdi)$.

Let us assume that the cost of initializing the means and splitting empty clusters is negligible, i.e. $O(1)$. In practice, the two are related: we find that explicitly picking good starting means for the agglomerative algorithm (e.g. randomly, or by maximizing distance between the means) makes the probability of empty clusters small, whereas it becomes excessive and unnecessary for the divisive algorithm. See **??** for more discussion.

We will also assume that d and i are constants. Though in degenerate data sets i can be an exponential function of n, in practice i is a small constant on datasets with clustering structuring.

Let us also assume that we choose a branching factor of 2, i.e. a binary hierarchical tree structure. The following analysis does not depend on the branching factor, assuming it is constant.

0.1.1 Agglomerative complexity

Theorem.

$$T_{aggl}(n) = \Theta(n^2 di)$$

Proof. Our agglomerative algorithm has a recurrence relation of the following form:

$$T_{aggl}(n) = T(\frac{n}{2}) + nkdi$$

for some constant c, since we divide the problem size into two for every recursive call. But we choose $k = \frac{n}{2}$ each time, since the number of means we choose to cluster with is also half of n.

We use the master theorem [**?**] to solve this recurrence relation:

Master theorem variable	**Value**
a	1
b	2
$f(n)$	$\frac{n^2 di}{2}$
c	2
$log_b(a)$	0

This satisfies case 3 of the master theorem:

$$f(n) \in \Omega(n^c) \text{ s.t. } c > log_b(a)$$

since $f(n) \in \Omega(n^2)$ s.t. $c = 2 > log_b(a) = 0$, and

$$af(\frac{n}{b}) \leq k_0 f(n) \text{ for some } k_0 < 1 \text{ and sufficiently large } n$$

since when $k_0 = \frac{1}{4}$, $\left\{ f(\frac{n}{2}) = (\frac{1}{4})\frac{n^2 di}{2} \right\} \leq \left\{ k_0 f(n) = (\frac{1}{4})\frac{n^2 di}{2} \right\}$

and so, by the master theorem: $T_{aggl}(n) = \Theta(n^2 di)$. $\square$

0.1.2 Divisive complexity

Theorem.

$$T_{div}(n) = \Theta(n \log(n) di)$$

Proof. Our divisive algorithm has a recurrence relation of the following form:

$$T_{div}(n) = 2T(\frac{n}{2}) + nkdi$$

for some constant c, since we divide the problem size into two for every recursive call and we have two subproblems each time. But we choose $k = 2$ each time, since the number of means we choose to cluster with is always the branching factor.

Problems of this form are solvable using a highly prominent and useful algorithm called the master theorem[**?**], which (given certain assumptions about the corresponding recurrence relation) gives the time complexity of many common recursive algorithms.

We use the master theorem to solve this recurrence relation:

This satisfies case 2 of the master theorem:

Master theorem variable	Value
a	2
b	2
$f(n)$	$n(2)di$
c	1
$log_b(a)$	1

$$f(n) \in \Theta(n^c \log_{k_0} n) \text{ s.t. } c = log_b(a)$$

by letting $k_0 = 0$, since $f(n \log^0(n)) = f(n) \in \Theta(n)$ s.t. $c = 1 = log_b(a) = 1$.
and so, by the master theorem: $T_{div}(n) = \Theta(n \log(n) di)$. □

0.1.3 Conclusion

As we demonstrate with our complexity proofs, the divisive complexity algorithm is actually faster (linearithmic versus quadratic), which guides our use of the divisive algorithm in our experiments.