K Means

1 VARIATIONS OF IMPLEMENTATION EXPLAINED HERE

- Lloyd's Algorithm (Default)
- Elkan's Algorithm (Triangle inequality bounded)
- Hamerly's Algorithm (Only 1 lower bound in comparison to k of them in Elkan's)

2 VARATIONS OF IMPLEMENTATIONS NOT EXPLAINED HERE YET

- Drake's and Hamerly's Algorithm (b lower bounds, where $1 < b < k$)
- Annular algorithm
- Heap-ordered k-means
- MacQueen's Algorithm (Streaming)
- Hartingan-Wong K means
- Coresets with K means

3 SPEEDING UP TRADITIONAL IMPLEMENTATION

This section is the rough summary of https://www.ccs.neu.edu/home/radivojac/classes/2021fallcs6220/hamerly_bookchapter_2014.pdf.

The main steps in the traditional Lloyd's implementation of k means optimisation are:

(1) Initialise the centres
(2) Until convergence do:
    (a) Assign each point to its currently closest cluster centre
    (b) Move each centre to the mean of its currently-assigned centres.

3.1 Reason for inefficiency

The main reason why the standard batch method for optimizing k-means is inefficient is that in each iteration it must identify the closest centre for each clustered point. To do this, the methods naively compute all nk distances between each of the n clustered points and each of the k centres. After each iteration, the centres move and these distances may all change, requiring recomputation. But typically the centres don't move much, especially after the first few iterations. Most of the time the closest centre in the previous iteration remains the closest centre. Thus, keeping track of the closest centre for each clustered point is made much more efficient with some caching. When the closest centre doesn't change, ideally we shouldn't need to compute the distance between that point and any cluster centre. And even when the closest centre for a point changes, it might be possible to avoid computing the distance from that point to all centres, instead looking only at a few centres that are guaranteed to be closer to that point than all other centres.

3.2 Initilasation step optimisation

https://www.sciencedirect.com/science/article/pii/S0957417412008767
https://www.ccs.neu.edu/home/radivojac/classes/2021fallcs6220/ch8.pdf#page=43
The convergence rate of the k-means algorithm can be affected by the initial positions of the cluster centres. Well-chosen initial centres that are close to the true cluster centroids or capture the data's inherent structure can lead to faster convergence. This is because the initial clusters are already relatively close to their final positions, reducing the number of iterations required for convergence.

The most effective current method is the k-means++ initialization, which randomly selects a good initialization with a high probability.

3.3 Moving centre step optimisation

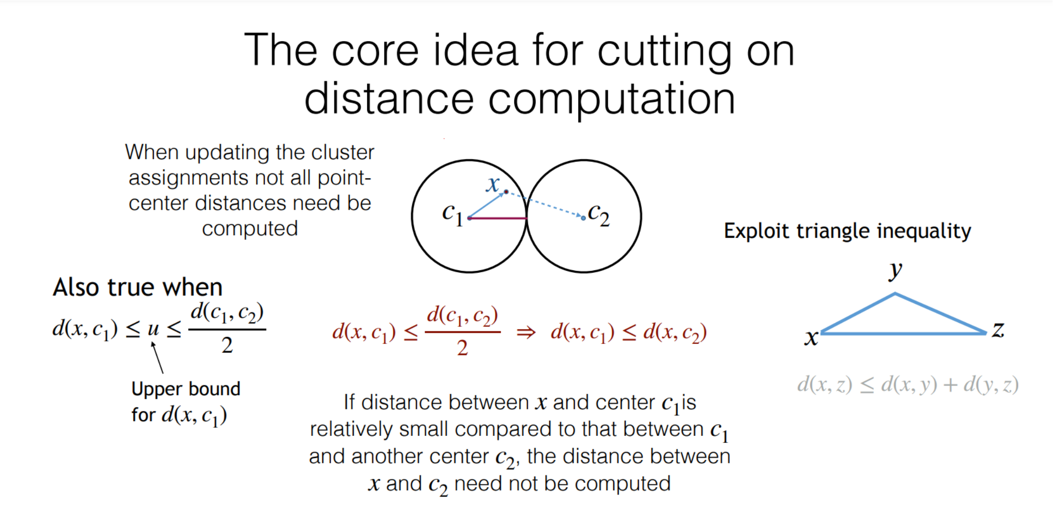
Step 2(b) can be optimized easily by caching sufficient statistics for each cluster, the vector sum of the points assigned to the cluster, and the number of points assigned to the cluster. Keeping these is inexpensive and avoids a sum over all points for each iteration. Each time a point changes cluster membership, the relevant sufficient statistics are updated. After the first few iterations, most points remain in the same cluster for many iterations. Thus these sufficient statistics updates become much cheaper than a sum over all points.

3.4 Reassignment step optimisation

The algorithms implementations presented here use the idea of triangle inequality which states that:

$$\|a - c\| \le \|a - b\| + \|b - c\|$$

this means that the length of line segment (a, c) is at most the sum of the lengths of line segments (a, b) and (b, c). In other words, the shortest path between two points a and c is a straight line. Taking a path that goes through an intermediate point b cannot reduce the distance.



The core idea for cutting on distance computation

When updating the cluster assignments not all point-center distances need be computed

Exploit triangle inequality

Also true when $d(x, c_1) \le u \le \frac{d(c_1, c_2)}{2}$

Upper bound for $d(x, c_1)$

$d(x, c_1) \le \frac{d(c_1, c_2)}{2} \Rightarrow d(x, c_1) \le d(x, c_2)$

$d(x, z) \le d(x, y) + d(y, z)$

If distance between x and center c_1 is relatively small compared to that between c_1 and another center c_2, the distance between x and c_2 need not be computed

Fig. 1. Intuition behind optimising computation https://www.ccs.neu.edu/home/radivojac/classes/2021fallcs6220/advkmeans.pdf#page=17

Elkan's algorithm https://www.researchgate.net/publication/2480121_Using_the_Triangle_Inequality_to_Accelerate_K-Means

Preliminary intuition

Let x be a point and let c and c' be the centres. We need to know that $\|x - c\| \ge \|x - c'\|$ in order to avoid calculating the actual value of $\|x - c'\|$. There are two Lemmas useful for obtaining bounds

on distances:
Lemma 1: Let x be a point and let c and c' be the centres. If $\|c - c'\| \geq 2\|x - c\|$ then $\|x - c'\| \geq \|x - c\|$.
We use Lemma 1 as follows. Let x be any data point, let c be the centre to which x is currently assigned, and let c' be any other center. The lemma says that if $\frac{1}{2}\|c - c'\| \geq \|x - c\|$, then $\|x - c'\| \geq \|x - c\|$. In this case, it is not necessary to calculate $\|x - c'\|$. Suppose that we do not know $\|x - c\|$ exactly, but we do know an upper bound u such that $u \geq \|x - c\|$. Then we need to compute $\|x - c'\|$ and $\|x - c\|$ only if $u > \frac{1}{2}\|c - c'\|$.

Lemma 2: Let x be a point and let c and c' be the centres. Then $\|x - c'\| \geq \max\{0, \|x - c\| - \|c - c'\|\}$.
Lemma 2 is applied as follows. Let x be any data point, let c be any center, and let c' be the previous version of the same center. Suppose that in the previous iteration we knew a lower bound l' such that $\|x - c'\| \geq l'$. Then we can infer a lower bound l for the current iteration:

$$\begin{aligned} \|x - c\| &\geq \max\{0, \|x - c'\| - \|c - c'\|\} \\ &\geq \max\{0, l' - \|c - c'\|\} = l \end{aligned}$$

Informally, if l' is a good approximation to the previous distance between x and the center, and this center has moved only a small distance, then l is a good approximation to the updated distance.

Algorithm

Initialise the centres, set the lower bound $l(x, c) = 0$ for each point x and centre c. Assign each x to its closest centre. Each time $\|x - c\|$ is computed, set $l(x, c) = \|x - c\|$. Assign upper bounds $u(x) = \min_c \|x - c\|$.
Repeat until convergence:

```
(1) For all centres $c$ and $c'$ compute $\|c - c'\|$. For all centres $c$, compute $s(c) = \frac{1}{2}\min_{c \neq c'}\|c - c'\|$.
(2) Identify all points x such that $u(x) \leq s(c(x))$
(3) For all remaining points $x$ and centers $c$ such that
      (i) $c \neq c(x)$ and
     (ii) $u(x) > l(x, c)$ and
    (iii) $u(x) > \frac{1}{2}\|c(x) - c\|$
    Do:
      (a) If $r(x)$ then compute $\|x - c(x)\|$ and assign $r(x) =$ false. Otherwise $\|x - c(x)\| = u(x)$.
      (b) If $\|x - c(x)\| > l(x, c)$ or $\|x - c(x)\| > \frac{1}{2}\|c(x) - c\|$ then:
          Compute $\|x - c\|$
          If $\|x - c\| < \|x - c(x)\|$ then assign $c(x) = c$
(4) For each centre $c$, let $m(c)$ be the mean of the points assigned to $c$.
(5) For each point $x$ and centre $c$, assign
        $l(x, c) = \max\{l(x, c) - \|c - m(c)\|, 0\}$
(6) For each point $x$ assign
        $u(x) = u(x) + \|m(c(x)) - c(x)\|$
        $r(x) =$ true
(7) Replace each centre $c$ by $m(c)$.
```

Limitation of Elkan's implementation is that storing and updating the lower bounds ($N \times K$ dimension) can be a bottleneck for large K. Hammerly's algorithm comes as a solution to that.

Example implementation in sklearn https://github.com/scikit-learn/scikit-learn/blob/364c77e047ca08a95862bec
sklearn/cluster/_kmeans.py#L437
Example of benchmark code between Elkan's and Lloyd's https://gist.github.com/amueller/dce0841b9bda4a5973

Hamerly's algorithm https://www.researchgate.net/publication/220906984_Making_k-means_Even_Faster

Benchmark between scikit-learn from the Intel distribution for Python https://colfaxresearch.com/cfxkmeans/
Implementation in C++ with Python wrapper https://github.com/ColfaxResearch/CFXKMeans

Preliminary intuition

Hamerly's algorithm uses the same upper bound $u(x)$ for each point x - for the distance between that point and its closest centre $c(x)$. But instead of k lower bounds, it uses only one lower bound per point, $l(x)$. This lower bound does not bound the distance from x to any particular cluster centre. Instead, it represents the minimum distance that any centre — except for the closest — can be to that point.
Consider the case where $u(x) \leq l(x)$. If this is true, it is not possible for any centre to be closer to x than its assigned centre. Thus, determining the assignment for x does not require knowing any exact distances, and the algorithm can skip the innermost loop that computes the distances between x and the k centres.
However if $u(x) > l(x)$ Hamerly's algorithm first tightens the upper bound by computing the exact distance $u(x) = \|x - c(x)\|$. If this reduces $u(x)$ significantly, then possibly $u(x) \leq l(x)$ and the algorithm can skip the innermost loop. If not, then it must compute the distances between x and all k cluster centres.

Algorithm

Initialise the centres, set the bounds $l(x) = 0$ and $u(x) = \infty$. Assign each x to its closest centre.
Repeat until convergence:

```
(1) For all centres c compute $s(c) = \frac{1}{2} \min_{c \neq c'} \|c - c'\|$.
(2) For points $x$ do:
    (a) $z = \max\{l(x), s(c)\}$
    (b) if $u(x) \leq z$ then continue with the next $x$
    (c) $u(x) = \|x - c(x)\|$    (Tighten the upper bound)
    (d) if $u(x) \leq z$ then continue with the next $x$
    (e) Find $c$ and $c'$, two closest centres to $x$, as well as the distances to each other
    (f) if $c \neq c(x)$ then:
         (i) $c(x) = c$
        (ii) $u(x) = \|x - c(x)\|$
    (g) $l(x) = \|x - c'\|$
(3) For each centre $c$ do:    (Update the centres)
    (a) Move centre $c$ to the new location
    (b) Let $\delta(c)$ be the distance moved by cluster $c$
(4) $\delta' = max_c \delta(c)$
(5) For points $x$ do:    (Update the bounds)
    (a) $u(x) = u(x) + \delta(c(x))$
    (b) $l(x) = l(x) - \delta'$
```

Hamerly's algorithm has several efficiency tradeoffs compared with Elkan's algorithm. With fewer lower bounds, Hamerly's algorithm uses less memory. It spends less time checking bounds (in the innermost loop) and updating bounds (when centres move). Having the single lower bound allows it to avoid entering the innermost loop more often than Elkan's algorithm. On the other hand, Elkan's algorithm computes fewer distances than Hamerly's, since Elkan's has more bounds to prune the required distance calculations. Also, Hamerly's algorithm works better in low dimensions than in high dimensions. Its single lower bound reduces by the maximum distance moved by any centre, and in high dimension, all centres tend to move a lot due to the curse of dimensionality.

Drake's and Hamerly's algorithm can be used to bridge the gap between the 1 and k lower bounds. It proposes the usage of b lower bounds where $1 < b < k$.

Drake's and Hamerly's algorithm http://cs.baylor.edu/~hamerly/papers/opt2012_paper_13.pdf

Annular algorithm https://www.ccs.neu.edu/home/radivojac/classes/2021fallcs6220/hamerly_bookchapter_2014.pdf

Heap-ordered k means algorithm https://www.ccs.neu.edu/home/radivojac/classes/2021fallcs6220/hamerly_bookchapter_2014.pdf