K-Means & Acceleration Methods

Summer 2018

Contents

Partitioning clustering with k-means

Clustering: Division of data set into subsets (clusters) such that:

- Elements in each subset are pairwise similar (w.r.t. some distance measure)
- Elements from differing subsets are pairwise dissimilar

Partitioning clustering: Clustering where each data element belongs to exactly one cluster

k-means:

- Chosen as one of the top 10 data mining algorithms in [1]
- Computes partitioning clustering - in short:
  1. Given k centers (points from the same space as the input data), assign each point to the closest center
  2. Recompute each center as mean of all points assigned to it
  3. Repeat from step 1 until convergence
- Most expensive part: Point-center distance computations
- Many methods to avoid unnecessary distance computations exist
- Here, we will introduce a chosen few along with the basic k-means method: Lloyd's algorithm[2]

Lloyd's algorithm - Notation

Definition 1 (Notation: k-means)

For convenience: $\forall m, n \in \mathbb{Z}$:

For $m \leq n$: $(m{:}n) := \{m, m+1 \ldots, n\}$ and for $m > n$: $(m{:}n) := \emptyset$

Input data $\mathrm{X} := (x(1), \ldots, x(n)), x(i) \in \mathbb{R}^D$:

n points from D-dimensional Euclidean vector space

The Euclidean distance d: $\mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}_0^+, (x, y) \mapsto \|x - y\|_2$:

$$d(x, y) := \|x - y\|_2 = \sqrt{(x - y)^t (x - y)}$$

Current centers $\mathrm{C} := (c(1), \ldots, c(k)), c(j) \in \mathbb{R}^D$:

k points from the same vector space $\mathbb{R}^D$.

Current point-to-center assignment a: $(1{:}n) \to (1{:}k)$:

$a(i) = j :\Leftrightarrow x(i)$ currently assigned to center $c(j)$.

Current cluster size $n(j) := \left|a^{-1}(j)\right|$: With $a^{-1}(j)$ being the set of points currently assigned to $c(j)$. Thus: $\sum_{j=1}^{k} n(j) = n$

Lloyd's algorithm - Objective function

Objective function: Minimize the intra-cluster variance (distortion):

Definition 2 (Distortion function)

Given a set X of n points $x(i)$, a set C of k centers $c(j)$ and an assignment function $a(i)$, the distortion function is:

$$J\left(\mathrm{C}, a(\cdot)|\mathrm{X}\right) := \sum_{i=1}^{n} d^2\left(x(i), c(a(i))\right)$$

$$= \sum_{j=1}^{k} \sum_{i \in a^{-1}(j)} d^2\left(x(i), c(j)\right)$$

Theorem 1 (NP-hardness)

Finding global minimum of $J\left(\mathrm{C}, a(\cdot)|\mathrm{X}\right)$ *is* ***NP-hard****.[3]*

k-means output: *Local* minimum and, in rare pathological cases, saddle point of J - see Subsection 4 (Convergence properties).

Lloyd's algorithm - Pseudocode I

Lloyd's algorithm[2]: Alternately optimize J w.r.t. a and C:

Algorithm 1: $\textsc{Lloyd}(\mathrm{X}, \mathrm{C})$

Input: n points $\mathrm{X} \in \mathbb{R}^{D \times n}$, k initial centers $\mathrm{C} \in \mathbb{R}^{D \times k}$
Output: A local minimum (or saddle point) $(\mathrm{C}, a(\cdot))$ of $J\left(\mathrm{C}', a'(\cdot)|\mathrm{X}\right)$

```
// Initialize objective function
J_new ← ∞
do
    J ← J_new
    // Step 1: Compute a(·) := arg min_{a'(·)} J(C, a'(·)|X)
    for i ← 1 to n do
        a(i) ← Update_Assignment(x(i), C)
    // Step 2: Compute C := arg min_{C'} J(C', a(·)|X)
    for j ← 1 to k do
        c(j) ← Update_Center(a^{-1}(j), X)
    // Recompute objective function
    J_new ← J(C, a(·)|X)
while J > J_new // Repeat as long as J decreases
return (C, a(·))
```

Lloyd's algorithm - Pseudocode II

Update_Assignment:

Solve $\arg\min_{a'(\cdot)} J\left(\mathrm{C}, a'(\cdot)|\mathrm{X}\right)$

$\Rightarrow$ For each $i \in (1{:}n)$, minimize $d^2\left(x(i), c(a(i))\right)$ w.r.t. $a(i)$

$\Rightarrow$ Assign each $x(i)$ to center that is closest w.r.t. d:

Algorithm 2: Update_Assignment(x, C)

Input: For some $i \in (1{:}n)$: Point $x := x(i)$, current centers C
Output: Index a of center closest to x

```
// Get index of center closest to x
```

1 $a \leftarrow \arg\min_{j' \in (1:k)} d(x, c(j'))$
2 **return** a

Handling ambiguities: If multiple closest centers have the same distance to a point (i.e., $\left|\arg\min_{j' \in (1:k)} d(x, c(j'))\right| > 1$), choose one according to an arbitrary scheme (e.g., simply at random or the candidate center with the smallest or biggest index)

Lloyd's algorithm - Pseudocode III

Update_Center:

Solve $\mathrm{C} := \arg\min_{\mathrm{C}'} J\left(\mathrm{C}', a(\cdot)|\mathrm{X}\right) = \arg\min_{\mathrm{C}'} \sum_{j=1}^{k} \sum_{i \in a^{-1}(j)} \sum_{l=1}^{D} \left(x_l(i) - c'_l(j)\right)^2$

$\Rightarrow$ For each $j \in (1{:}k), l \in (1{:}D)$, solve $\frac{\partial J}{\partial c_l(j)} \stackrel{!}{=} 0$ for $c_l(j)$:

$$\frac{\partial J}{\partial c_l(j)} = -2 \cdot \sum_{i \in a^{-1}(j)} \left(x_l(i) - c_l(j)\right) \stackrel{!}{=} 0$$

$$\Leftrightarrow \sum_{i \in a^{-1}(j)} x_l(i) = \sum_{i \in a^{-1}(j)} c_l(j) = n(j) \cdot c_l(j)$$

$$\Leftrightarrow c_l(j) = \frac{1}{n(j)} \sum_{i \in a^{-1}(j)} x_l(i)$$

$$\Rightarrow c(j) = \frac{1}{n(j)} \sum_{i \in a^{-1}(j)} x(i)$$

This is a unique minimum because: $\frac{\partial^2 J}{\partial c_l^2(j)} = 2 \cdot n(j) > 0$

Lloyd's algorithm - Pseudocode IV

Update_Center (cont'd):

Algorithm 3: Update_Center(I, X)

Input: For some $j \in (1{:}k)$, set $I = a^{-1}(j)$ of point indices assigned to center $c(j)$, input points X

Output: Recomputed center c

```
// Recompute center as mean of points assigned to it
```

1 $c \leftarrow \frac{1}{|I|} \sum_{i \in I} x(i)$

2 **return** c

Case $n(j) = 0$: Either leave $c(j)$ as it is or randomly assign new position.

Other vectorspaces & metrics: Solution of $\arg\min_{\mathrm{C}'} J\left(\mathrm{C}', a(\cdot)|\mathrm{X}\right)$ will, in general, be different in other vectorspaces and/or with other distance metrics

Lloyd's algorithm - k-medians I

k-medians: k-means variant using Manhattan distance and a slightly different objective function:

- Manhattan distance: $d_M(x, y) := \sum_{l=1}^{D} |x_l - y_l|$
- Objective function: $J_M\left(\mathrm{C}', a(\cdot)|\mathrm{X}\right) := \sum_{i=1}^{n} d_M\left(x(i), c(a(i))\right)$
  Note: Here, we use sum of distances while J uses sum of *squared* distances (absolute errors vs. squared errors)

k-medians - optimizing $\arg\min_{a'(\cdot)} J_M\left(\mathrm{C}, a'(\cdot)|\mathrm{X}\right)$:

- Same solution as before

k-medians - optimizing $\arg\min_{\mathrm{C}'} J_M\left(\mathrm{C}', a(\cdot)|\mathrm{X}\right)$

- $c_l(j) := median(\{x_l(i) \mid i \in a^{-1}(j)\})$
  (component-wise median of assigned points)
- Computing the centers via median instead of mean is more robust against outliers

Lloyd's algorithm - k-medians II

k-medians - optimizing $\arg\min_{\mathrm{C}'} J_M(\mathrm{C}', a(\cdot)|\mathrm{X})$ (cont'd)

Derivation: With $I_l^+(j) := \left\{ i \in a^{-1}(j) \mid x_l(i) \geq c_l(j) \right\}$ and $I_l^-(j) := a^{-1}(j) \setminus I_l^+(j) = \left\{ i \in a^{-1}(j) \mid x_l(i) < c_l(j) \right\}$, we have:

$$J_M(\mathrm{C}', a(\cdot)|\mathrm{X}) = \sum_{j=1}^{k} \sum_{i \in a^{-1}(j)} \sum_{l=1}^{D} |x_l(i) - c'_l(j)|$$

$$= \sum_{j=1}^{k} \sum_{l=1}^{D} \left[\sum_{i \in I_l^+(j)} (x_l(i) - c'_l(j)) + \sum_{i \in I_l^-(j)} (c'_l(j) - x_l(i)) \right]$$

$$\Rightarrow \frac{\partial J_M}{\partial c_l(j)} = \sum_{i \in I_l^+(j)} (-1) + \sum_{i \in I_l^-(j)} 1 \stackrel{!}{=} 0 \Leftrightarrow |I_l^-(j)| \stackrel{!}{=} |I_l^+(j)|$$

$$\Rightarrow c_l(j) = median(x_l(i) \mid i \in a^{-1}(j))$$

This is a unique minimum because:
$\frac{\partial J_M}{\partial c_l(j)}$ (non-strictly) monotonically increasing with $c_l(j)$

Lloyd's algorithm - Convergence properties I

Lemma 1 (Convergence)

Algorithm 1 (Lloyd) converges in a finite number of steps

Proof (Lemma 1)

1. Derivations of Algorithm 2 (Update_Assignment) and Algorithm 3 (Update_Center) directly imply that J is (non-strictly) monotonically decreasing with each iteration of main k-means loop.
2. For a finite number $n \in \mathbb{N}$ of points, there is only a finite number of possible partitionings.
3. Algorithm 1 (Lloyd) stops as soon as J does not decrease anymore.

1. & 2. & 3. $\Rightarrow$ Lemma 1 $\square$

Lloyd's algorithm - Convergence properties II

Lemma 2 (Local optimality)

Let $(\mathrm{C}_{out}, a_{out}(\cdot)) := \textsc{Lloyd}(\mathrm{X}, \mathrm{C})$. *Then the following holds:*
Final center assignment resulting in $a_{out}(\cdot)$ *was unique*
$\Rightarrow$ *Algorithm 1 (*LLOYD*) converges to local minimum of* J.

Proof (Lemma 2)

Point x was **uniquely** assigned to a center
$\Rightarrow$ New center is either the same as before or is closer (otherwise the assignment would not be unique)
$\Rightarrow$ Each unique change of assignment decreases J
Thus: If final assignment was unique
$\Rightarrow$ Every other assignment $a(\cdot) \neq a_{\text{out}}(\cdot)$ would increase J
$\Rightarrow$ Lemma 2 □

Lloyd's algorithm - Convergence properties III

Ambiguous final assignment: We might get stuck in a saddle point:

Example 1

For $D := 1, \mathrm{X} := (-2, 0, 0, 2)$, initial $\mathrm{C} := (-1, 1), d(x, y) = |x - y|$:

- Unique assignment of $x(1)$ to $c(1)$ and $x(4)$ to $c(2)$, assignment of x(2)=x(3)=0 not unique
- Assume the first iteration assigns $x(2)$ to $c(1)$ and x(3) to $c(2)$.
- Then centers do not move and $J = 4$.

Now two possible outcomes for k-means:

1. Leave or swap the assignments of $x(2)$ and $x(3)$ and we still get the same centers and $J \Rightarrow k$-means terminates with $J = 4$
   - Saddle point of J as two "adjacent" solutions have the same J
2. Assign $x(2)$ and $x(3)$ to $c(1)$. Centers move to $c(1) = -\frac{2}{3}$ and $c(2) = 2$. This is the final solution with $J = \frac{8}{3} < 4$
   - Local and in this case even global minimum of J

Lloyd's algorithm - Convergence properties IV

Avoiding saddle points: Different methods possible, for example:

1. Temporarily add small amount of random noise to centers before reassigning points
   - Minimizes chance of two centers having same distance to a point
   - Cheap, but J is not necessarily monotone anymore
2. More expensive:
   As long as J does not decrease: Keep track of all encountered assignments and, in case of non-uniqueness, choose assignment not encountered yet. Abort main loop, if no non-encountered assignments left.
   - Guaranteed to find local minimum of J
   - Requires storing m assignments, where m is the number of consecutive main loop iterations where J does not decrease

Lloyd's algorithm - Final remarks I

Problem: k has to be known beforehand or be estimated

Comparing clusterings of different k:

- For $k = n$, optimal solution puts each point in its own cluster resulting in "perfect" distortion score of $J = 0$
- Obviously, this is not desirable
  $\Rightarrow$ Distortion function J not suitable for choosing "best" clustering among solutions with different number of clusters
- Popular approach: Choose solution that maximizes so called silhouette coefficient[4]

Choosing k:

- Execute k-means multiple times with different k
- Or: Start k-means with big k, periodically merge close centers
- And/or: Combine k-means with mean shift[5]
- Among resulting solutions, choose the one with biggest silhouette coefficient
- Overview of methods for determining k: [6]

Lloyd's algorithm - Final remarks II

Cluster shapes: Only finds non-overlapping clusters of convex shape

Main loop time complexity:

- Single Euclidean distance computation $\in \mathcal{O}(D)$
- Assignment updates $\in \mathcal{O}(n \cdot k \cdot D)$
- Center updates $\in \mathcal{O}(n \cdot D)$
- $\Rightarrow$ Single main loop iteration $\in \mathcal{O}(n \cdot k \cdot D)$

Number of main loop iterations: Highly depends on:

Data structure: The weaker the data's cluster structure, the longer until k-means converges

- As data gets closer to uniform distribution, surface of J gets "flatter" and has fever distinct minima to converge to

Choice of k: The bigger k, the more often center assignments change in the beginning and the longer it takes until assignments converge

Choice of initial centers: Choose initial centers evenly distributed over data (see k-means++[7])

Lloyd's algorithm - Final remarks III

Accelerating Algorithm 3 (Update_Center):

- Keep track of reassigned points
- Center updates can be accelerated to $\mathcal{O}(n' \cdot D)$ with n' being the number of reassigned points (therefore $n' \leq n$):

Proof

For some $i \in (1{:}n), j, j' \in (1{:}k), j \neq j'$, reassign $x(i)$ from center $c(j)$ to $c(j')$. Then (assuming $n_{\text{new}}(j) = n(j) - 1 > 0$, otherwise see remark on Slide 9):

$$c_{\text{new}}(j) = \frac{1}{n(j) - 1} \left(n(j) \cdot c(j) - x(i) \right)$$

$$c_{\text{new}}(j') = \frac{1}{n(j') + 1} \left(n(j') \cdot c(j') + x(i) \right)$$

$\Rightarrow$ Each movement requires $\mathcal{O}(D)$ to update affected centers $\square$

Lloyd's algorithm - Final remarks IV

Accelerating Algorithm 2 (Update_Assignment):

- Naïve assignment update involves a lot of unnecessary point-center distance computations
- In the following sections, we present a selection of methods that:
  - Prune a portion of unnecessary distance computations
  - Despite pruning, produce the same results as Lloyd's Algorithm 2 (Update_Assignment)

Contents

Accelerating distance calculations I

Partial distance search[8]: For sum-based metrics with non-negative terms (e.g., Minkowski distances[9] like Euclidean or Manhattan distance)

- Assume:
  - We know the distance $d(x, c)$ of a point x to center c
  - We want to decide whether a different center c' is closer to x: $d(x, c') \overset{?}{<} d(x, c)$
- If, for example, d is the Euclidean distance, then computing $d(x, c')$ involves sum over squared distances from each dimension
- As soon as sum gets bigger than $d^2(x, c)$, we know that $d(x, c') > d(x, c)$
  $\Rightarrow$ No need to finish summation!

Accelerating distance calculations II

Efficient Euclidean distance[10]:

- We have:

$$\begin{aligned}\arg\min_c \|x-c\| &= \arg\min_c \|x-c\|^2 \\ &= \arg\min_c (x^t x - 2x^t c + c^t c) \\ &= \arg\min_c (-2x^t c + c^t c)\end{aligned}$$

- $c^t c$ can be computed in advance for each center c
- $x^t x$ is irrelevant for solution of $\arg\min_c \|x-c\|^2$
  $\Rightarrow$ We only need to compute $2x^t c$
  $\Rightarrow$ We save D "minus"-operations and one "square root"-operation compared to computing
  $\|x-c\| = \sqrt{(x-c)^t (x-c)}$

Pruning via bounds

Avoid unnecessary point-distance computations:

- Suppose for every point $x(i)$ we knew:
  - An upper bound $u(i)$ on the distance to *closest* center
  - A lower bound $l(i, j)$ on the distance to center $c(j)$
- Then we can *prune* distance computations to all centers $c(j)$ with $l(i, j) > u(i)$
  - If we know at least one center $c(j')$ whose distance is at most $u(i)$, we may additionally prune distance computations to all $c(j)$ with $l(i, j) = u(i)$
  - If no unpruned centers left, $c(j')$ must be the closest one!
- In order to achieve acceleration, we want to:
  - Be able to efficiently update bounds after each center movement (faster than it would take to recompute exact distances)
  - Keep bounds as tight as possible in order to prune as many distance computations as possible
- In the following sections, we present a number of bound-based methods ...

Contents

Triangle inequality

Bound updates via triangle inequality:

- Popular class of very efficient methods[11, 10, 12, 13] for bound updates is based on triangle inequality:

Definition 3 (Triangle inequality)

Function $d\colon V \times V \to \mathbb{R}_0^+$ on some set V fulfills *triangle inequality* iff:

$$\forall x, y, z \in V : d(x, y) \leq d(x, z) + d(z, y)$$

- Intuition: "Direct path from x to y is always the shortest"
- Euclidean distance $d(x, y) = \|x - y\|_2$ is a metric and thus, by definition of a metric (cf. [14]), fulfills the triangle inequality
- In the following subsections, we present three complementary algorithms that:
  - Respectively perform best in high, low and medium dimensions
  - Most state-of-the art methods are extensions/improvements of these three methods[13, 15]

Elkan's algorithm - Introduction

General information:

- Introduced by Elkan [11] in 2003
- For each point, maintains one upper bound on the distance to closest center ...
- ... and k lower bounds on the distance to each center
- First k-means variant that uses lower bounds and carries over varying information from one iteration to the next
- [12] suggest that Elkan's algorithm is faster than both Hamerly's (Subsection 3) and Drake's (Subsection 4) method in high dimension settings with $D \geq \sim 120$

Elkan's algorithm - Notation

Definition 4 (Notation: Bounds & more I)

Index of currently closest center $a\colon (1{:}n) \to (1{:}k)$:

$a(i) := \arg\min_j d(x(i), c(j))$

Current upper bound $u\colon (1{:}n) \to \mathbb{R}_0^+$:

$u(i) :=$ upper bound on distance from $x(i)$ to closest center $c(a(i))$

Current lower bound $l\colon (1{:}n) \times (1{:}k) \to \mathbb{R}_0^+$:

$l(i,j) :=$ lower bound on distance from $x(i)$ to $c(j)$

Current pairwise center distance $s\colon (1{:}k) \times (1{:}k) \to \mathbb{R}_0^+$

$s(j,j') := d(c(j), c(j'))$

Distance to closest other center: $s_{\mathsf{min}}(j) := \min_{j' \in (1:k)\setminus\{j\}} s(j,j')$

Old value/mapping/position $*_{\mathsf{old}}$:

$*_{\mathsf{old}}$ refers to values/mappings/positions $* \in \{a, u, l, c, \dots\}$ **before** the respective last update of $*$

Last center movement $\delta\colon (1{:}k) \to \mathbb{R}_0^+$: $\delta(j) := d(c_{\mathsf{old}}(j), c(j))$

Elkan's algorithm - Upper bounds I

Updating upper bounds:

Lemma 3 (New upper bound $u(i)$)

Given $u_{old}(i)$ (upper bound on the distance from $x(i)$ to $c_{old}(a_{old}(i))$, the old closest center at its old position) and $\delta(a_{old}(i))$ (distance old closest center moved), we can update upper bound on the distance to the (unknown) new closest center $c(a(i))$ as:

$$d(x(i), c(a(i))) \leq d(x(i), c(a_{old}(i))) \leq u(i) := u_{old}(i) + \delta(a_{old}(i))$$

What it means:

1. Left inequality: New closest center $c(a(i))$ cannot be further away than $c(a_{\text{old}}(i))$, the old closest center at its new position
2. Right inequality: Old closest center at its new position cannot be further away than its old maximal distance plus the distance it has moved

Elkan's algorithm - Upper bounds II

Proof (Lemma 3)

1. Either old closest center at its new position is still closest one (i.e., $a(i) = a_{\text{old}}(i)$) or a different center is closer (i.e., $a(i) \neq a_{\text{old}}(i)$):

$$d(x(i), c(a(i))) \leq d(x(i), c(a_{\text{old}}(i)))$$

2. With $c_{\text{old}}(a_{\text{old}}(i))$ being the old closest center at its old position, $\delta(a_{\text{old}}(i)) := d(c_{\text{old}}(a_{\text{old}}(i)), c(a_{\text{old}}(i)))$ being its last moved distance, we get via the triangle inequality:

$$d(x(i), c(a_{\text{old}}(i))) \leq d(x(i), c_{\text{old}}(a_{\text{old}}(i))) + \delta(a_{\text{old}}(i))$$

3. By definition $d(x(i), c_{\text{old}}(a_{\text{old}}(i))) \leq u_{\text{old}}(i)$ and we get:

$$\begin{aligned} d(x(i), c(a(i))) &\leq d(x(i), c(a_{\text{old}}(i))) \\ &\leq d(x(i), c_{\text{old}}(a_{\text{old}}(i))) + \delta(a_{\text{old}}(i)) \\ &\leq u_{\text{old}}(i) + \delta(a_{\text{old}}(i)) \end{aligned}$$

□

Elkan's algorithm - Lower bounds I

Updating lower bounds:

Lemma 4 (New lower bound $l(i,j)$)

Given $l_{old}(i,j)$ (lower bound on the distance from $x(i)$ to $c_{old}(j)$, the j-th center at its old position) and the j-th center's moved distance $\delta(j)$, we can update lower bound on the distance to $c(j)$ (the j-th center at its new position) as:

$$d(x(i), c(j)) \geq l(i,j) := l_{old}(i,j) - \delta(j)$$

What it means: $c(j)$ cannot be closer than its old distance minus the distance it moved

Elkan's algorithm - Lower bounds II

Proof (Lemma 4)

Via triangle inequality, we get:

$$\begin{aligned} d(x(i), c_{\text{old}}(j)) &\leq d(x(i), c(j)) + \delta(j) \\ \Leftrightarrow d(x(i), c(j)) &\geq d(x(i), c_{\text{old}}(j)) - \delta(j) \\ &\geq l_{\text{old}}(i, j) - \delta(j) \end{aligned}$$

□

Elkan's algorithm - Lower bounds III

Additional pruning:

Lemma 5 (Pruning via center-center distances)

Given $u(i)$, the index $a_{old}(i)$ of the old closest center to $x(i)$, some $j \in (1{:}k)$ and the center-center distances $s(\cdot,\cdot), s_{min}(\cdot)$, we have:

1. $u(i) \leq \frac{1}{2} s(a_{old}(i), j) \Rightarrow d(x(i), c(a_{old}(i))) \leq d(x(i), c(j))$
2. $u(i) \leq \frac{1}{2} s_{min}(a_{old}(i)) \Rightarrow a(i) = a_{old}(i)$

What it means:

1. $u(i) \leq \frac{1}{2} s(a_{\text{old}}(i), j)$ implies that old closest center at its new position is at least as close to $x(i)$ as $c(j)$
   $\Rightarrow$ No need to explicitly compute distance to $c(j)$!
2. $u(i) \leq \frac{1}{2} s_{\text{min}}(a_{\text{old}}(i))$ implies that $c(a_{\text{old}}(i))$ is still closest center
   $\Rightarrow$ We can directly infer that $a(i) = a_{\text{old}}(i)$ without any point-center distance calculations!

Elkan's algorithm - Lower bounds IV

Proof (Lemma 5 - Part I)

The second sublemma directly follows from the first one:

1. With the definition of $s_{\text{min}}(\cdot)$ and the precondition $u(i) \leq \frac{1}{2} s_{\text{min}}(a_{\text{old}}(i))$, we get:

$$u(i) \leq \frac{1}{2} s_{\text{min}}(a_{\text{old}}(i)) \leq \frac{1}{2} s(a_{\text{old}}(i), j) \qquad \forall j \in (1{:}k) \setminus \{a_{\text{old}}(i)\}$$

2. Using the first sublemma, the above implies:

$$\Rightarrow d(x(i), c(a_{\text{old}}(i))) \leq d(x(i), c(j)) \qquad \forall j \in (1{:}k) \setminus \{a_{\text{old}}(i)\}$$
$$\Rightarrow a(i) = a_{\text{old}}(i)$$

Elkan's algorithm - Lower bounds V

Proof (Lemma 5 - Part II)

1. To prove the first sublemma, we first prove:

$$\forall x, c, c' \in \mathbb{R}^D: \qquad d(x,c) \leq \frac{1}{2} d(c,c') \Rightarrow d(x,c) \leq d(x,c')$$

Apply the triangle inequality, rearrange and then use the precondition $2d(x,c) \leq d(c,c')$:

$$\begin{aligned} & d(c,c') \leq d(x,c) + d(x,c') \\ \Leftrightarrow\ & d(c,c') - d(x,c) \leq d(x,c') \\ \Rightarrow\ & d(x,c) \leq d(x,c') \end{aligned}$$

2. Set $x = x(i), c = c(a_{\text{old}}(i)), c' = c(j)$, use $u(i) \geq d(x(i), c(a_{\text{old}}(i)))$ and the result from above:

$$d(x(i), c(a_{\text{old}}(i))) \leq u(i) \leq \frac{1}{2} d(c(a_{\text{old}}(i)), c(j)) \overset{\text{by def.}}{=} \frac{1}{2} s(a_{\text{old}}(i), j)$$

$$\Rightarrow d(x(i), c(a_{\text{old}}(i))) \leq d(x(i), c(j))$$

□

Elkan's algorithm

We continue with integrating Lemma 3 to 5 into Lloyd's algorithm resulting in Elkan's algorithm[11] ...

Elkan's algorithm - Pseudocode I

Algorithm 4: Elkan(X, C)

Input: n points $\mathrm{X} \in \mathbb{R}^{D \times n}$, k initial centers $\mathrm{C} \in \mathbb{R}^{D \times k}$
Output: A local minimum (or saddle point) $(\mathrm{C}, a(\cdot))$ of $J\left(\mathrm{C}', a'(\cdot)|\mathrm{X}\right)$

```
// Initialize J, assignments (get recomputed anyways) and bounds
```
1 $(J_{\text{new}}, a(\cdot), u(\cdot), l(\cdot,\cdot)) \leftarrow (\infty, 1, \infty, 0)$
2 **do**
3 $\quad J \leftarrow J_{\text{new}}$
$\quad$ `// Update center-center distances`
4 $\quad s(j, j') \leftarrow d(c(j), c(j'))$ $\quad \forall j, j' \in (1{:}k)$
5 $\quad s_{\text{min}}(j) \leftarrow \arg\min_{j' \in (1:k) \setminus \{j\}} s(j, j')$ $\quad \forall j \in (1{:}k)$
6 $\quad$ **for** $i \leftarrow 1$ ***to*** n **do**
$\quad\quad$ `// Update assignment and, if possible, some bounds`
7 $\quad\quad (a(i), u(i), l(i,\cdot)) \leftarrow$ Update_Assignment_Elkan$(x(i), \mathrm{C}, a(i), u(i), l(i,\cdot), s_{\text{min}}(a(i)), s(a(i),\cdot))$
$\quad$ `// Update centers and store moved distance`
8 $\quad$ **for** $j \leftarrow 1$ ***to*** k **do**
9 $\quad\quad (c(j), \delta(j)) \leftarrow$ Update_Center_Elkan$\left(a^{-1}(j), \mathrm{X}, c(j)\right)$
$\quad$ `// Update all bounds`
10 $\quad (u(\cdot), l(\cdot,\cdot)) \leftarrow$ Update_Bounds_Elkan$(a(\cdot), u(\cdot), l(\cdot,\cdot), \delta(\cdot))$
11 $\quad J_{\text{new}} \leftarrow J\left(\mathrm{C}, a(\cdot)|\mathrm{X}\right)$
12 **while** $J > J_{new}$
13 **return** $(\mathrm{C}, a(\cdot))$

(Changes w.r.t. Algorithm 1 (Lloyd) are denoted in red)

Elkan's algorithm - Pseudocode II

Algorithm 6: UPDATE_ASSIGNMENT_ELKAN$(x, \mathrm{C}, a_{\mathrm{OLD}}, u, l(\cdot), s_{\mathrm{MIN}}, s(\cdot))$

Input: For some $i \in (1{:}n)$: Point $x := x(i)$, centers C, index $a_{\mathsf{old}} := a_{\mathsf{old}}(i)$ of last closest center, upper bound $u := u(i)$, lower bounds $l(\cdot) := l(i, \cdot)$, current distance $s_{\mathsf{min}} := s_{\mathsf{min}}(a_{\mathsf{old}}(i))$ of last closest center to its closest other neighbor, current distances $s(\cdot) := s(a_{\mathsf{old}}, \cdot)$ of the last closest center to each other center

Output: Index a of closest center, u and $l(\cdot)$ with some entries refined with exact distances

if $u \leq \frac{1}{2} s_{min}$ **then return** $(a_{old}, u, l(\cdot))$ // Lemma 5.2
$update\_old\_dist \leftarrow$ true
$a \leftarrow a_{\mathsf{old}}$
for $j \in (1{:}k)$ **do**
  $max\_lower\_bound \leftarrow \max\left\{l(j), \frac{1}{2} s(j)\right\}$ // Tighten lower bound via Lemma 5.1
  // Case $j = a_{\mathsf{old}}$: Either already found closer candidate than $c(a_{\mathsf{old}})$ or latter still
  // closest center considered so far - both cases: No exact distance to $c(a_{\mathsf{old}})$ needed
  if $(j = a_{old}$ **OR** $u \leq max\_lower\_bound)$ **then continue** with next j
  // Compute $d(x, c(a_{\mathsf{old}}))$ only if necessary and not more than once
  if $update\_old\_dist$ **then**
    $u \leftarrow d(x, c(a_{\mathsf{old}}))$
    $update\_old\_dist \leftarrow$ false
  $l(j) \leftarrow d(x, c(j))$
  // At this point: $u =$ exact dist. to closest center considered so far
  if $u > l(j)$ **then** // $\Leftrightarrow u > d(x, c(j))$
    $(a, u) \leftarrow (j, l(j))$
return $(a, u, l(\cdot))$

Elkan's algorithm - Pseudocode III

Update_Center_Elkan:

Similar to Algorithm 3 (Update_Center), but additionally returns moved distance:

Algorithm 8: Update_Center_Elkan$(I, \mathrm{X}, c_{\mathrm{OLD}})$

Input: For some $j \in (1{:}k)$: Set $I = a^{-1}(j)$ of point indices assigned to the center to be updated, input points X, old center position $c_{\mathsf{old}} := c_{\mathsf{old}}(j)$

Output: Recomputed center c and its moved distance δ

1 $c \leftarrow \frac{1}{|I|} \sum_{i \in I} x(i)$

// Compute moved distance

2 $\delta \leftarrow d(c, c_{\mathsf{old}})$

3 **return** (c, δ)

Elkan's algorithm - Pseudocode IV

Update_Bounds_Elkan:

Apply Lemma 3 and 4:

Algorithm 10: Update_Bounds_Elkan$(a_{\text{OLD}}(\cdot), u_{\text{OLD}}(\cdot), l_{\text{OLD}}(\cdot,\cdot), \delta(\cdot))$

Input: Indices $a_{\text{old}}(\cdot)$ of closest centers before last movement, upper and lower bounds $u_{\text{old}}(\cdot)$ and $l_{\text{old}}(\cdot,\cdot)$ referring to old center positions, distances $\delta(\cdot)$ moved by centers in last center update

Output: Updated upper and lower bounds $u(\cdot)$ and $l(\cdot,\cdot)$ guaranteed to be valid for new center positions

1 **for** $i \in (1{:}n)$ **do**
  // Lemma 3
2 $u(i) \leftarrow u_{\text{old}}(i) + \delta(a_{\text{old}}(i))$
  // Lemma 4
3 **for** $j \in (1{:}k)$ **do**
4 $l(i,j) \leftarrow l_{\text{old}}(i,j) - \delta(j)$
5 **return** $(u(\cdot), l(\cdot,\cdot))$

Elkan's algorithm - Final remarks

Overhead cost per iteration:

- Time: Dominated by computation of $s(\cdot,\cdot), s_{\text{min}}(\cdot) \in \mathcal{O}(k^2 \cdot D)$
- Space: Storage of $s(\cdot,\cdot), s_{\text{min}}(\cdot), u(\cdot), l(\cdot,\cdot)$
  $\Rightarrow \#\text{ values} = (k \cdot (k-1) + k + n + n \cdot k) \in \mathcal{O}(k^2 + n \cdot k)$

Speedup:

Hamerly [10] compares runtimes of Elkan's with the standard Lloyd algorithm and a few other methods:

- General observation: Elkan's speedup w.r.t. Lloyd increases with:
  - Dimensionality D: Pricier distance computations (Euclidean distance $\in \mathcal{O}(D)$) result in higher payoff when avoiding them
  - Number of centers k: Empirically, Elkan's number of required exact distance calculations grows *sublinearly* with k [11]
- Speedup factors of 4 and more for large D and k
- Speedups of < 1 only for low D ($< \sim 8$) **and** k ($< \sim 20$): Elkan's overhead then often outweighs distance calculation savings
- Next section: Hamerly's algorithm[10] which outperforms Elkan's method in low to medium dimensional cases

Hamerly's algorithm - Introduction

General information:

- Introduced by Hamerly [10] in 2010
- Same upper bound per point as Elkan's method
- Saves overhead by keeping only one lower bound per point
- Lower bound $l^+(i)$ on distance to second closest center
- Lemma 3: $d(x(i), c(a(i))) \leq d(x(i), c(a_{\text{old}}(i))) \leq u(i)$
  $\Rightarrow$ If $u(i) \leq l^+(i)$ then second closest center cannot be closer than $c(a_{\text{old}}(i))$ and we can derive $a(i) = a_{\text{old}}(i)$ without any distance calculations!
- While Elkan's algorithm aims at skipping single distance calculations, Hamerly's method aims at skipping main loop of Algorithm 2 (UPDATE_ASSIGNMENT) altogether
- [12] suggest that Hamerly's algorithm is faster than both Elkan's (Subsection 2) and Drake's (Subsection 4) method in low dimension settings with $D < \sim 120$

Hamerly's algorithm - Notation I

Definition 5 (Notation: Order and finite domain functions)

Given a function $f\colon (1{:}n) \to \mathbb{R}$, we define:

Index of z-th smallest element w.r.t. variable i, $\arg\min_i\text{-z}\, f(i)$:

$$\arg\min_i\text{-1}\, f(i) := \arg\min_{i \in (1:n)} f(i)$$

$$\forall z \in (2{:}n) \quad \arg\min_i\text{-z}\, f(i) := \arg\min_{\substack{i \in (1:n) \\ \forall z' \in (1:(z-1)) \\ i \neq \arg\min_{i'}\text{-[z']}\, f(i')}} f(i)$$

z-th smallest element $\min_i\text{-z}\, f(i)$:

$$\min_i\text{-z}\, f(i) := f(\arg\min_i\text{-z}\, f(i))$$

$\arg\max_i\text{-z}\, f(i)$ and $\max_i\text{-z}\, f(i)$: Analogous for z-th largest element

$\tilde{f}(\cdot)$ is $f(\cdot)$ ordered in ascending manner:

$$\forall z \in (1{:}n)\colon \tilde{f}(z) := \min_i\text{-z}\, f(i) = f(\arg\min_i\text{-z}\, f(i))$$

Hamerly's algorithm - Notation II

Definition 6 (Notation: Bounds & more II)

Index of currently 2nd closest center $a^+ : (1{:}n) \to (1{:}k)$:

$$a^+(i) := \arg\min_j\text{-}2\, d(x(i), c(j))$$

Current 2nd lower bound $l^+ : (1{:}n) \to \mathbb{R}_0^+$:

$l^+(i) \leq d(x(i), c(a^+(i))$, lower bound on distance from $x(i)$ to currently second closest center $c(a^+(i))$

Index of center that moved the z-th farthest $j_{\mathsf{max},z}$:

$$j_{\mathsf{max},z} := \arg\max_j\text{-z}\, \delta(j)$$

Hamerly's algorithm - Lower bounds I

Updating lower bounds:

Lemma 6 (New lower bound $l^+(i)$)

Given $\delta(j_{max,1})$, $\delta(j_{max,2})$ (two largest distances that centers moved), $l^+_{old}(i)$ (lower bound on the distance from $x(i)$ to $c_{old}(a^+_{old}(i))$, the old second closest center at its old position) and $a_{old}(i)$ (index of previously closest center), we can update the lower bound on the distance to $c(a^+(i))$ (the currently second closest center at its new position) as:

$$d(x(i), c(a^+(i))) \geq l^+(i) := l^+_{old}(i) - \delta(j_{max,1}) \quad \text{for } a_{old}(i) \neq j_{max,1}$$
$$d(x(i), c(a^+(i))) \geq l^+(i) := l^+_{old}(i) - \delta(j_{max,2}) \quad \text{for } a_{old}(i) = j_{max,1}$$

Note: Second case is a tighter bound since:

$$\delta(j_{\text{max},1}) \geq \delta(j_{\text{max},2})$$
$$\Rightarrow l^+_{\text{old}}(i) - \delta(j_{\text{max},2}) \geq l^+_{\text{old}}(i) - \delta(j_{\text{max},1})$$

Hamerly's algorithm - Lower bounds II

To prove Lemma 6, we first prove the following:

Lemma 7 (Point-wise order implies sorted point-wise order)

Given two functions $f, g\colon (1{:}n) \to \mathbb{R}$, *we have:*

$$\forall i \in (1{:}n)\colon \quad f(i) \geq g(i) \Rightarrow \forall j \in (1{:}n)\colon \quad \tilde{f}(j) \geq \tilde{g}(j)$$

Proof (Lemma 7 - Part I)

Let $I_f\colon (1{:}n) \to (1{:}n)$ be a *bijective* (and therefore invertible) mapping that uniquely assigns each index of $f(\cdot)$ to the corresponding index of its ascendingly sorted version $\tilde{f}(\cdot)$, i.e.:

$$\forall i \in (1{:}n)\colon \qquad I_f(i) := h_f^{-1}(i) \qquad \text{with } h_f(z) := \arg\min_{i'}\text{-z}\, f(i')$$

$$\Rightarrow f(i) = \tilde{f}(I_f(i))$$

Let $I_g(\cdot)$ be analogously defined for $g(\cdot)$.

Hamerly's algorithm - Lower bounds III

Proof (Lemma 7 - Part II)

Let $\forall j \in (1{:}n)\colon I_{\tilde{f}\to\tilde{g}}(j) := (I_g \circ I_f^{-1})(j)$ be the mapping that maps the index j assigned by $I_f(i)$ to the index assigned by $I_g(i) = I_g(I_f^{-1}(j))$. Since $I_f(\cdot)$ and $I_g(\cdot)$ are *bijective*, so is $I_{\tilde{f}\to\tilde{g}}(\cdot)$ and we have:

$$\forall j \in (1{:}n)\colon \quad f(I_f^{-1}(j)) \overset{\text{precondition}}{\geq} g(I_f^{-1}(j))$$

$$\overset{\text{by definition}}{\Leftrightarrow} \quad \tilde{f}(I_f(I_f^{-1}(j))) \geq \tilde{g}(I_g(I_f^{-1}(j)))$$

$$\tilde{f}(j) \geq \tilde{g}(I_{\tilde{f}\to\tilde{g}}(j)) \tag{1}$$

Hamerly's algorithm - Lower bounds IV

Proof (Lemma 7 - Part III)

Now we show via contradiction:
Assume $\forall i \in (1{:}n)\colon f(i) \geq g(i)$ but $\exists j' \in (1{:}n)$ with $\tilde{f}(j') < \tilde{g}(j')$. We show that the latter implies the contradiction $\tilde{f}(j') < \tilde{f}(j')$. Note that Equation (1) is still applicable as it only relies on $f(i) \geq g(i)$.

$$\forall j < j'\colon \quad \tilde{g}(I_{\tilde{f}\to\tilde{g}}(j)) \overset{Equation\ (1)}{\leq} \tilde{f}(j) \overset{\tilde{f}\text{ ascending}}{\leq} \tilde{f}(j') \overset{\text{precondition}}{<} \tilde{g}(j')$$
$$\overset{\tilde{g}\text{ ascending}}{\Rightarrow} I_{\tilde{f}\to\tilde{g}}(j) < j'$$

$I_{\tilde{f}\to\tilde{g}}(\cdot)$ is bijective $\Rightarrow I_{\tilde{f}\to\tilde{g}}(j') =: j''$ must map to an index $j'' \geq j'$:

$$\tilde{f}(j') < \tilde{g}(j') \overset{j''\geq j'}{\leq} \tilde{g}(j'') \overset{Equation\ (1)}{\leq} \tilde{f}(j') \quad \lightning$$

□

Hamerly's algorithm - Lower bounds V

Proof (Lemma 6 - Part I)

We first prove the $a_{\text{old}}(i) \neq j_{\text{max},1}$ case:

$$\forall j \in (1{:}k): \quad d(x(i), c(j)) \geq d(x(i), c_{\text{old}}(j)) - \delta(j) \tag{2}$$

$$\Rightarrow d(x(i), c(a^+(i))) = \min_j\text{-}2\, d(x(i), c(j))$$

$$\overset{\substack{Lemma\ 7 \\ \text{and Equation (2)}}}{\geq} \min_j\text{-}2\, [d(x(i), c_{\text{old}}(j)) - \delta(j)]$$

$$\geq \min_j\text{-}2\, [d(x(i), c_{\text{old}}(j)) - \delta(j_{\text{max},1})]$$

$$= d(x(i), c_{\text{old}}(a^+_{\text{old}}(i))) - \delta(j_{\text{max},1})$$

$$\geq l^+_{\text{old}}(i) - \delta(j_{\text{max},1})$$

Hamerly's algorithm - Lower bounds VI

Proof (Lemma 6 - Part II)

Now the $a_{\text{old}}(i) = j_{\text{max},1}$ case: Since we know that $c(a_{\text{old}}(i))$ moved the biggest distance $\delta(j_{\text{max},1})$, we know that $c(a^+(i))$ cannot have moved more than the second biggest distance $\delta(j_{\text{max},2})$:

$$\begin{aligned} \forall j \in (1{:}k) \setminus \{a_{\text{old}}(i)\}: \qquad & \delta(j_{\text{max},1}) \geq \delta(j_{\text{max},2}) \geq \delta(j) \\ \Rightarrow\ & d(x(i), c(j)) \geq d(x(i), c_{\text{old}}(j)) - \delta(j) \\ & \geq d(x(i), c_{\text{old}}(j)) - \delta(j_{\text{max},2}) \end{aligned} \tag{3}$$

Equation (3) and $\overset{Lemma\ 7}{\Rightarrow}$

$$\begin{aligned} d(x(i), c(a^+(i))) &= \min_j\text{-}2\, d(x(i), c(j)) \\ &\geq \min_j\text{-}2[d(x(i), c_{\text{old}}(j)) - \delta(j_{\text{max},2})] \\ &= d(x(i), c_{\text{old}}(a^+_{\text{old}}(i))) - \delta(j_{\text{max},2}) \\ &\geq l^+_{\text{old}}(i) - \delta(j_{\text{max},2}) \end{aligned}$$

□

Hamerly's algorithm

We continue with integrating Lemma 3, 5 and 6 into Lloyd's algorithm resulting in Hamerly's algorithm[10] ...

Hamerly's algorithm - Pseudocode I

Algorithm 11: Hamerly(X, C)

Input: n points $\mathrm{X} \in \mathbb{R}^{D \times n}$, k initial centers $\mathrm{C} \in \mathbb{R}^{D \times k}$
Output: A local minimum (or saddle point) $(\mathrm{C}, a(\cdot))$ of $J\left(\mathrm{C}', a'(\cdot)|\mathrm{X}\right)$

// Initialize J, assignments (get recomputed anyways) and bounds
$(J_{\text{new}}, a(\cdot), u(\cdot), l^+(\cdot)) \leftarrow (\infty, 1, \infty, 0)$
do
  $J \leftarrow J_{\text{new}}$
  // Update center-center distances
  $s_{\text{min}}(j) \leftarrow \arg\min_{j' \in (1:k) \setminus \{j\}} d(c(j), c(j'))$ $\quad \forall j \in (1{:}k)$
  for $i \leftarrow 1$ ***to*** n **do**
    // Update assignment and, if possible, some bounds
    $(a(i), u(i), l^+(i)) \leftarrow$ Update_Assignment_Hamerly$(x(i), \mathrm{C}, a(i), u(i), l^+(i), s_{\text{min}}(a(i)))$
  // Update centers and store moved distance (same as in Elkan)
  for $j \leftarrow 1$ ***to*** k **do**
    $(c(j), \delta(j)) \leftarrow$ Update_Center_Elkan$\left(a^{-1}(j), \mathrm{X}, c(j)\right)$
  $j_{\text{max},1} \leftarrow \arg\max\text{-}1_j\, \delta(j)$
  $j_{\text{max},2} \leftarrow \arg\max\text{-}2_j\, \delta(j)$
  // Update all bounds
  $(u(\cdot), l^+(\cdot)) \leftarrow$ Update_Bounds_Hamerly$(a(\cdot), u(\cdot), l^+(\cdot), \delta(\cdot), j_{\text{max},1}, j_{\text{max},2})$
  $J_{\text{new}} \leftarrow J\left(\mathrm{C}, a(\cdot)|\mathrm{X}\right)$
while $J > J_{new}$
return $(\mathrm{C}, a(\cdot))$

(Changes w.r.t. Algorithm 1 (Lloyd) are denoted in red)

Hamerly's algorithm - Pseudocode II

Algorithm 13: Update_Assignment_Hamerly$(x, \mathrm{C}, a_{\mathrm{OLD}}, u, l^+, s_{\mathrm{MIN}})$

Input: For some $i \in (1{:}n)$: Point $x := x(i)$, centers C, index $a_{\mathsf{old}} := a_{\mathsf{old}}(i)$ of last closest center, upper bound $u := u(i)$ on distance to closest center, lower bound $l^+ := l^+(i)$ on distance to second closest center, current distance $s_{\mathsf{min}} := s_{\mathsf{min}}(a_{\mathsf{old}}(i))$ of last closest center to its closest other neighbor

Output: Index a of closest center, u and l^+ (either old ones or one or both refined with exact distances)

```
// Tighten lower bound via Lemma 5.2
```
$max\_lower\_bound \leftarrow \max\left\{l^+, \frac{1}{2}s_{\mathsf{min}}\right\}$
if $u \leq max\_lower\_bound$ **then return** (a_{old}, u, l^+)
```
// First pruning failed?  Try again with refined u:
```
$u \leftarrow d(x, c(a_{\mathsf{old}}))$
if $u \leq max\_lower\_bound$ **then return** (a_{old}, u, l^+)
```
// Pruning failed!  Recompute all point-center distances:
```
$a \leftarrow \arg\min\text{-}1_j\, d(x, c(j))$
$a^+ \leftarrow \arg\min\text{-}2_j\, d(x, c(j))$
if $a \neq a_{old}$ **then** // Avoid computing exact u twice
    $u \leftarrow d(x, c(a))$
$l^+ \leftarrow d(x, c(a^+))$
return (a, u, l^+)

Hamerly's algorithm - Pseudocode III

Update_Bounds_Hamerly:

Apply Lemma 3 and 6:

Algorithm 15: Update_Bounds_Hamerly$(a_{\text{OLD}}(\cdot), u_{\text{OLD}}(\cdot), l^+_{\text{OLD}}(\cdot), \delta(\cdot), j_{\text{MAX},1}, j_{\text{MAX},2})$

Input: Indices $a_{\text{old}}(\cdot)$ of closest centers before last movement, upper and lower bounds $u_{\text{old}}(\cdot)$ and $l^+_{\text{old}}(\cdot)$ referring to old center positions, distances $\delta(\cdot)$ moved by centers in last center update, indices $j_{\text{max},1}$ and $j_{\text{max},2}$ of centers that moved the furthest and 2nd furthest

Output: Updated upper and lower bounds $u(\cdot)$ and $l^+(\cdot)$ guaranteed to be valid for new center positions

1 **for** $i \in (1{:}n)$ **do**
  // Lemma 3
2 $u(i) \leftarrow u_{\text{old}}(i) + \delta(a_{\text{old}}(i))$
  // Lemma 6
3 **if** $a_{old}(i) = j_{max,1}$ **then**
4 $l^+(i) \leftarrow l^+_{\text{old}}(i) - \delta(j_{\text{max},2})$
5 **else**
6 $l^+(i) \leftarrow l^+_{\text{old}}(i) - \delta(j_{\text{max},1})$
7 **return** $(u(\cdot), l^+(\cdot))$

Hamerly's algorithm - Final remarks

Overhead cost per iteration:

- Time: Dominated by computation of $s_{\text{min}}(\cdot) \in \mathcal{O}(k^2 \cdot D)$
- Space: Storage of $s_{\text{min}}(\cdot), u(\cdot), l^+(\cdot)$

  $\Rightarrow \#\text{ values} = (k + n + n) \in \mathcal{O}(k + n) \overset{k \leq n}{=} \mathcal{O}(n)$

Speedup: Hamerly [10] compares runtimes of his algorithm with Elkan's, the standard Lloyd method and a few others:

- General observations: Faster than Elkan's algorithm for roughly up to $D = 50$:
  - Curse of dimensionality: In higher dimensions, data tends to be less well separable via distance measures
  - Single bounds become less useful - having multiple increases chance of avoiding at least a few distance computations
  - With low D, single lower bound often enough to skip the center assignment loop altogether
- Speedup factors of up to 10 and more w.r.t to Lloyds
- Next section: Drake's algorithm[12] which outperforms Elkan's and Hamerly's method in medium dimensional cases

Drake's algorithm - Introduction I

General information:

- Introduced by Drake and Hamerly [12] in 2012
- Same upper bound per point as Elkan's method
- Keeps $b \in (1{:}(k-1))$ lower bounds
- Periodically adapts $b \Rightarrow$ Middle ground between Elkan's and Hamerley's method
- Drake and Hamerly [12] suggest that their algorithm is faster than both Elkan's (Subsection 2) and Hamerly's (Subsection 3) method in medium dimension settings with $\sim 20 \leq D < \sim 120$

Drake's algorithm - Introduction II

Choice of lower bounds I:

- Hamerley's method: For each point $x(i)$, keep k lower bounds $l(i,j)$ on distance to all centers $c(j)$ without any particular ordering of $l(i,j)$
- Now: For $z \in (1{:}(b-1))$, z-th lower bound initially refers to $(z+1)$-th closest center
- In later iterations this property might get lost and is only reestablished if no pruning was possible and distances to *all* centers had to be computed
  - But: High probability that *current* closest center will be among those centers that have been among the closest at some point in the past and for those we keep bounds
- No lower bound for last closest center required (i.e., for $c(a_{\text{old}}(i))$ - see next slide)
- b-th lower bound has special meaning: Refers to the whole set of $(k-b)$ remaining centers (again, except the last closest one) and must therefore be a valid lower bound for all of them

Drake's algorithm - Introduction III

Choice of lower bounds II:

- z-th lower bound is $\hat{l}^+(i,z)$
- For each point $x(i)$, prune distance calculations to all centers corresponding to lower bounds with $\hat{l}^+(i,z) \geq u(i)$
- In particular, if $\hat{l}^+(i,z) \geq u(i)$ for all $1 \leq z \leq b$, then $a(i) \stackrel{!}{=} a_{\text{old}}(i)$ and no distance calculation required for $x(i)$

Drake's algorithm - Introduction IV

Choice of b:

- Higher b: Higher bound-keeping overhead
- Lower b: Lower pruning efficiency
- Observation: In the beginning greater movement of centers and therefore individual bounds less effective
  $\Rightarrow$ Start with a larger number b of bounds and reduce b as centers move less and less

Drake's algorithm - Notation I

Definition 7 (Notation: Bounds & more III)

Number b of lower bounds to keep: $b \in (1{:}(k-1))$

Index $a^+(i,z)$ of currently $(z+1)$-th closest center:

$$a^+ \colon (1{:}n) \times (1{:}(k-1)) \to (1{:}k)$$

$$a^+(i,z) := \arg\min_j\text{-}[z+1]\, d(x(i), c(j))$$

Index $\hat{a}^+(i,z)$ of center to which z-th lower bound refers to:

$$\hat{a}^+ \colon (1{:}n) \times (1{:}b) \to (1{:}k)$$

$$\hat{a}^+(i,z) \neq a_{\text{old}}(i) \quad \forall (i,z) \in (1{:}n) \times (1{:}b)$$

$\hat{a}^+(i,\cdot)$ should "approximate" first b entries of $a^+(i,\cdot)$ and be *identical* in the case no pruning was possible and we had to compute all point-center distances for a given point $x(i)$

Current z-th lower bound $\hat{l}^+ \colon (1{:}n) \times (1{:}b) \to \mathbb{R}_0^+$:

$$\hat{l}^+(i,z) \leq d(x(i), c(\hat{a}^+(i,z)))$$

Drake's algorithm - Notation II

Definition 8 (Notation: Bounds & more IV)

Set $\hat{B}(\cdot)$ of center indices corresponding to current b-th lower bound:

$$\hat{B}\colon (1{:}n) \to 2^{(1:k)}$$

$$\hat{B}(i) := \{j \in (1{:}k) \mid j \neq \hat{a}^+(i,z) \forall z \in (1{:}(b-1)) \wedge j \neq a_{old}(i)\}$$

$$\hat{l}^+(i,b) \leq d(x(i), c(j)) \quad \forall j \in \hat{B}(i)$$

Current set of candidate center indices $A\colon (1{:}n) \to 2^{(1:k)}$:

$$A(i) := \{a_{\text{old}}(i)\} \cup \{\hat{a}^+(i,z) \mid z \in (1{:}(b-1)) \wedge \hat{l}^+(i,z) < u(i)\} \cup \begin{cases} \emptyset & \text{if } \hat{l}^+(i,b) \geq u(i) \\ \hat{B}(i) & \text{if } \hat{l}^+(i,b) < u(i) \end{cases}$$

- Current $a(i)$ must be in $A(i)$
- Also: $|A(i)| = 1 \Rightarrow A(i) = \{a_{\text{old}}(i)\} \Rightarrow a(i) = a_{\text{old}}(i)$ (in that case, no distance calculations required for $x(i)$)

Drake's algorithm - Lower bounds I

Lemma 8 (New z-th lower bound $\hat{l}^+(i,z)$)

Given $\hat{l}^+_{old}(i,z)$ (lower bound on distance to $c_{old}(\hat{a}^+_{old}(i,z))$) and the other values below, we can update the lower bound on distance to $c(\hat{a}^+(i,z))$ as:

1. *For $z < b$: $d(x(i), c(\hat{a}^+(i,z))) \geq \hat{l}^+(i,z)$ with ...*

$$\hat{l}^+(i,z) := \begin{cases} \hat{l}^+_{old}(i,z) - \delta(\hat{a}^+(i,z)) & \text{if } \hat{a}^+(i,z) = \hat{a}^+_{old}(i,z) \\ d(x(i), c_{old}(\hat{a}^+(i,z))) - \delta(\hat{a}^+(i,z)) & \text{else} \end{cases}$$

2. *For $z = b$: $\forall j \in \hat{B}(i)$: $d(x(i), c(j)) \geq \hat{l}^+(i,b)$ with ...*

$$\hat{l}^+(i,b) := \begin{cases} \hat{l}^+_{old}(i,b) - \delta(j_{max,1}) & \text{if } \hat{B}(i) = \hat{B}_{old}(i) \\ \min\limits_{j \in \hat{B}(i)} d(x(i), c_{old}(j)) - \delta(j_{max,1}) & \text{else} \end{cases}$$

- [12] uses $\delta(j_{\text{max},1})$ to compute $\hat{l}^+(i,b)$ but we may also use $\max\limits_{j \in \hat{B}(i)} \delta(j)$ for a tighter bound (but: overhead of explicitly determining $\hat{B}(i)$ may not pay off!)

Drake's algorithm - Lower bounds II

Proof (Lemma 8 - Part I)

The $z < b$ case follows directly from the triangle inequality:

$$\forall j \in (1{:}n) \quad d(x(i), c(j)) \geq d(x(i), c_{\text{old}}(j)) - \delta(j)$$
$$\Rightarrow d(x(i), c(\hat{a}^+(i,z))) \geq d(x(i), c_{\text{old}}(\hat{a}^+(i,z))) - \delta(\hat{a}^+(i,z))$$

In case of $\hat{a}^+(i,z) = \hat{a}^+_{\text{old}}(i,z)$, we can further derive:

$$\ldots = d(x(i), c_{\text{old}}(\hat{a}^+_{\text{old}}(i,z))) - \delta(\hat{a}^+(i,z))$$

By definition, $\hat{l}^+_{\text{old}}(i,z) \leq d(x(i), c_{\text{old}}(\hat{a}^+_{\text{old}}(i,z)))$:

$$\ldots \geq \hat{l}^+_{\text{old}}(i,z) - \delta(\hat{a}^+(i,z))$$

Drake's algorithm - Lower bounds III

Proof (Lemma 8 - Part II)

The $z = b$ case: $\forall j \in \hat{B}(i)$, we have ...

$$\begin{aligned} d(x(i), c(j)) &\geq d(x(i), c_{\text{old}}(j) - \delta(j) \\ &\geq \min_{j \in \hat{B}(i)} [d(x(i), c_{\text{old}}(j)) - \delta(j)] \\ &\geq \min_{j \in \hat{B}(i)} d(x(i), c_{\text{old}}(j)) - \max_{j \in \hat{B}(i)} \delta(j) \end{aligned}$$

In case of $\hat{B}(i) = \hat{B}_{\text{old}}(i)$, we can further derive:

$$\ldots = \min_{j \in \hat{B}_{\text{old}}(i)} d(x(i), c_{\text{old}}(j)) - \max_{j \in \hat{B}(i)} \delta(j)$$

By definition, $\hat{l}^{+}_{\text{old}}(i, b) < d(x(i), c_{\text{old}}(j)) \ \forall j \in \hat{B}_{\text{old}}(i)$

$$\ldots \geq \hat{l}^{+}_{\text{old}}(i, b) - \max_{j \in \hat{B}(i)} \delta(j)$$

With $\max_{j \in \hat{B}(i)} \delta(j) \leq \delta(j_{\text{max},1})$, the proof is complete □

Drake's algorithm - Computing candidate set $A(i)$ I

Computation of candidate set $A(\cdot)$:

Given $i \in (1{:}n)$, we may compute $A(i)$ (Definition 8) in three ways:

1. Iterate over all lower bounds $\hat{l}^+(i,\cdot)$ and store indices $\hat{a}^+(i,z)$ where $\hat{l}^+(i,z) < u(i)$
   - Note: $\hat{l}^+(i,b) < u(i)$ implies $\hat{B}(i) \subset A(i)$ and we also *always* have $a_{old}(i) \in A(i)$
   - Time overhead (per point): $\mathcal{O}(b)$
   - Memory overhead (p. p.): $\mathcal{O}(b')$, with $b' \leq b$ being the number of indices z where $\hat{l}^+(i,z) < u(i)$
2. Sort $\hat{l}^+(i,\cdot)$ and $\hat{a}^+(i,\cdot)$ in an ascending manner w.r.t. $\hat{l}^+(i,\cdot)$ and determine maximal index $z' \in (0{:}b)$ with $(z' = 0) \vee [(z' > 0) \wedge (\hat{l}^+(i,z') < u(i))]$
   - Knowing z' and having $\hat{l}^+(i,\cdot)$ sorted ascendingly, we can easily retrieve our candidates by iterating over $\hat{l}^+(i,z)$ up until $z = z'$ (or derive that $a_{\text{old}}(i)$ is the only candidate if $z' = 0$)
   - Time overhead (p. p.): $\mathcal{O}(b \cdot \log(b))$ (sorting $\hat{l}^+(i,\cdot)$ and $\hat{a}^+(i,\cdot)$)
   - Memory overhead (p. p.): $\mathcal{O}(1)$ (storing z')

Drake's algorithm - Computing candidate set $A(i)$ II

Computation of candidate set $A(\cdot)$ (cont'd):

3. Method used in [12]: Sacrifice additional tightness of lower bounds to avoid overhead of previous two methods:
   1. Update lower bounds $\hat{l}^+(i,z)$ according to Lemma 8 in descending z-order (i.e., $z = b, b-1, \ldots, 1$)
   2. $\forall z \in (1{:}(b-1))$, after updating $\hat{l}^+(i,z)$, set $\hat{l}^+(i,z) \leftarrow \min\{\hat{l}^+(i,z), \hat{l}^+(i,z+1)\}$

   - Obviously, new lower bound is less tight and therefore also valid
   - Resulting sequence of lower bounds is already sorted in ascending manner!
   - Time overhead (p. p.): $\mathcal{O}(z'+1)$ (single additional $\min$ operation after each $\hat{l}^+(i,z)$ update and determining z')
   - Memory overhead (p. p.): $\mathcal{O}(1)$ (storing z' as in second method)

Drake's algorithm - Setting number of lower bounds I

Setting b:

- [12] uses adaptive tuning mechanism based on observations in experiments on uniformly distributed random datasets:
  1. Best performance for $b \in (\lceil \frac{k}{8} \rceil : \lceil \frac{k}{4} \rceil)$
  2. For majority of points, first lower bound enough (i.e., $\hat{l}^+(i,1) \geq u(i)$)
     - For the majority of remaining bounds, second lower bound enough (i.e., $\hat{l}^+(i,2) \geq u(i)$)
     - And so on ...
  3. These majorities increase with time meaning that later on some of the larger bounds do not get used at all

Drake's algorithm - Setting number of lower bounds II

Setting b (cont'd):

- Adaptive tuning mechanism incorporates these observations:
  1. Start with $b = \lceil \frac{k}{4} \rceil$
  2. In each iteration, compute maximal number m of bounds required to decide upon the candidate set by using the fact that $\hat{l}^+(i, \cdot)$ is sorted in ascending manner:

$$m = \max_{i \in (1:n)} \left\{ \min\{z \in (1:b) \mid z = b \vee \hat{l}^+(i,z) \geq u(i)\} \right\}$$
$$= \max_{i \in (1:n)} \{\min\{b, |A(i)|\}\}$$

  3. Set new b as: $b \leftarrow \max\{\lceil \frac{k}{8} \rceil, m\}$
- Note: b is guaranteed to monotonically decrease over time:
  - By definition: $m \leq b$ and $\lceil \frac{k}{8} \rceil \leq b$
  - For the new number of bounds b_{new}, we therefore have:

$$b_{\text{new}} = \max\{\lceil \frac{k}{8} \rceil, m\} \overset{m \leq b}{\leq} \max\{\lceil \frac{k}{8} \rceil, b\} \overset{\lceil \frac{k}{8} \rceil \leq b}{\leq} b$$

Drake's algorithm

We continue with integrating Lemma 3, 8 and the remarks from Slide 65 and 67 into Lloyd's algorithm resulting in Drake's algorithm[12] ...

Drake's algorithm - Pseudocode I

Algorithm 16: Drake(X, C)

Input: n points $\mathrm{X} \in \mathbb{R}^{D \times n}$, k initial centers $\mathrm{C} \in \mathbb{R}^{D \times k}$
Output: A local minimum (or saddle point) $(\mathrm{C}, a(\cdot))$ of $J\left(\mathrm{C}', a'(\cdot)|\mathrm{X}\right)$

```
// Initialize J, assignments (get recomputed anyways), lower bound number and bounds
```
$(J_{\text{new}}, a(\cdot), u(\cdot), b, \hat{a}^+(\cdot,\cdot), \hat{l}^+(\cdot,\cdot)) \leftarrow (\infty, 1, \infty, \lceil \frac{k}{4} \rceil, 1, 0)$
do
  $(J, m) \leftarrow (J_{\text{new}}, 1)$
  for $i \leftarrow 1$ ***to*** n **do**
    `// Update assignments, current number m of required lower bounds`
    `// (see Slide 67) and, if possible, some bounds`
    $(a(i), u(i), \hat{a}^+(i,\cdot), \hat{l}^+(i,\cdot), m') \leftarrow$ Update_Assignment_Drake$\left(x(i), \mathrm{C}, a(i), u(i), b, \hat{a}^+(i,\cdot), \hat{l}^+(i,\cdot)\right)$
    $m \leftarrow \max\{m', m\}$
  $b \leftarrow \max\{\lceil \frac{k}{8} \rceil, m\}$
  `// Update centers and store moved distance (same as in Elkan)`
  for $j \leftarrow 1$ ***to*** k **do**
    $(c(j), \delta(j)) \leftarrow$ Update_Center_Elkan$\left(a^{-1}(j), \mathrm{X}, c(j)\right)$
  $j_{\text{max},1} \leftarrow \arg\max\text{-}1_j\, \delta(j)$
  $(u(\cdot), \hat{l}^+(\cdot)) \leftarrow$ Update_Bounds_Drake$\left(a(\cdot), u(\cdot), b, \hat{a}^+(\cdot,\cdot), \hat{l}^+(\cdot,\cdot), \delta(\cdot), j_{\text{max},1}\right)$
  $J_{\text{new}} \leftarrow J\left(\mathrm{C}, a(\cdot)|\mathrm{X}\right)$ `// Update all bounds`
while $J > J_{new}$
return $(\mathrm{C}, a(\cdot))$

(Changes w.r.t. Algorithm 1 (Lloyd) are denoted in red)

Drake's algorithm - Pseudocode II

Algorithm 18: UPDATE_ASSIGNMENT_DRAKE$\left(x, \mathrm{C}, a_{\mathrm{OLD}}, u, b, \hat{a}^+(\cdot), \hat{l}^+(\cdot)\right)$

Input: For some $i \in (1{:}n)$: Point $x := x(i)$, centers C, index $a_{\mathsf{old}} := a_{\mathsf{old}}(i)$ of last closest center, upper bound $u := u(i)$, number of lower bounds b, indices $\hat{a}^+(z) := \hat{a}^+(i,z) \neq a$ of centers corresponding to b lower bounds $\hat{l}^+(z) := \hat{l}^+(i,z)$, with $z \in (1{:}b)$

Output: Index a of closest center, u (updated if $m > 1$), $\hat{a}^+$ and $\hat{l}^+$ (with first $m-1$ entries refined), number $m = \min\{b, |A(i)|\}$ of required bounds

// Compute m for the current point (see Slide 67)

$m \leftarrow \min\{z \in (1{:}b) \mid z = b \vee \hat{l}^+(i,z) \geq u(i)\}$

if $m = 1$ **then return** $(a_{\mathsf{old}}, u, \hat{a}^+(\cdot), \hat{l}^+(\cdot), m)$

// Compute set of center candidate indices (see Definition 8)

if $m < b$ **then**

  // All centers $c(\hat{a}^+(z))$ with $\hat{l}^+(i,z) \geq u(i)$ may be pruned

  $A \leftarrow \{a_{\mathsf{old}}\} \cup \{\hat{a}^+(z) \mid z \in (1{:}(m-1))\}$

else $A \leftarrow (1{:}k)$ // No pruning possible

// Sort candidate centers in ascending distance order

// and update corresponding bounds and indices

$a \leftarrow \arg\min_{j \in A} d(x, c(j))$

$u \leftarrow d(x, c(a))$

$\hat{a}^+(z) \leftarrow \arg\text{min-}[\mathrm{z}+1]_{j \in A}\, d(x, c(j)) \qquad \forall z \in (1{:}m)$

$\hat{l}^+(z) \leftarrow d(x, c(\hat{a}^+(z))) \qquad \forall z \in (1{:}m)$

return $(a, u, \hat{a}^+(\cdot), \hat{l}^+(\cdot), m)$

Drake's algorithm - Pseudocode III

Update_Bounds_Drake:

Apply Lemma 3, 8 and remark on Slide 65:

Algorithm 20: Update_Bounds_Drake$\left(a_{\text{OLD}}(\cdot), u_{\text{OLD}}(\cdot), b, \hat{a}^+(\cdot,\cdot), \hat{l}^+_{\text{OLD}}(\cdot,\cdot), \delta(\cdot), j_{\text{MAX},1}\right)$

Input: Closest centers before last movement $a_{\text{old}}(\cdot)$, number of lower bounds b, indices $\hat{a}^+(\cdot,\cdot)$ of centers corresponding to lower bounds $\hat{l}^+(\cdot,\cdot)$, distance $\delta(\cdot)$ moved by centers in last center update, index $j_{\text{max},1}$ of center that moved the furthest

Output: Updated upper and lower bounds $u(\cdot)$ and $\hat{l}^+(\cdot,\cdot)$ guaranteed to be valid for new center positions

1 **for** $i \in (1{:}n)$ **do**
  // Lemma 3
2 $u(i) \leftarrow u(i) + \delta(a_{\text{old}}(i))$
  // Note: At this point, $\hat{l}^+_{\text{old}}(i,\cdot)$ are guaranteed to contain the exact
  // distances of the "else"-cases in Lemma 8
  // Lemma 8
3 $\hat{l}^+(i,b) \leftarrow \hat{l}^+_{\text{old}}(i,b) - \delta(j_{\text{max},1})$
4 **for** $z \leftarrow (b-1)$ ***to*** 1 **do**
5 $\hat{l}^+(i,z) \leftarrow \min\{\hat{l}^+_{\text{old}}(i,z) - \delta(\hat{a}^+(i,z)), \hat{l}^+(i,z+1)\}$
6 **return** $(u(\cdot), \hat{l}^+(\cdot,\cdot))$

Drake's algorithm - Final remarks

Overhead cost per iteration:

- Time: Dominated by sorting $|A(i)| \leq k$ values after computing exact distances to all centers in $A(i)$, repeated for each point $\in \mathcal{O}(n \cdot k \cdot \log(k))$
- Space: Storage of $u(\cdot)$, $\hat{l}^+(\cdot,\cdot)$ and $\hat{a}^+(\cdot,\cdot)$
  $\Rightarrow \#$ values $= (n + 2 \cdot n \cdot b) \in \mathcal{O}(n \cdot b) \subseteq \mathcal{O}(n \cdot k)$

Speedup: Drake and Hamerly [12] compare runtimes of their algorithm with Elkan's, Hamerly's and standard Lloyd method:

- General observations: Outperforms Elkan's and Hamerly's algorithm in medium dimensions of roughly $\sim 20 \leq D < \sim 120$:
  - Middleground between Elkan's and Hamerly's method
  - Medium dimensionality: Having more than one bound is often beneficial but we usually do not need all of them
  - $\Rightarrow$ keep $1 < b < k$ bounds to centers that have been close by at some point
- Highest speedup factor of 2 w.r.t. next best method on uniformly distributed random data with $D = 50, k = 200$

Contents

Blacklisting and filtering centers with k-d trees I

K-d tree[16]:

- Data structure for points in k-dimensional space
- Recursively split space via a hyperplane perpendicular to some axis
- Represent splits as binary tree
- Root: Represents whole space
- Given a node and its assigned points, its children are:
  1. Select an axis (e.g., current depth modulo k plus one, i.e. cycle through axes)
  2. Find median of points in the selected axis' dimension
  3. Assign points left of this median to left child node and the other points to the right child node
  4. Repeat procedure on children nodes until only one point (or some minimal number of points) left

Blacklisting and filtering centers with k-d trees II

Pruning centers via k-d trees:

- Independently introduced by Pelleg and Moore [17] and Kanungo et al. [18]
- When constructing k-d tree, also compute smallest hyperrectangle enclosing points at each node
- Repeat each time centers have moved:
  1. Assign all centers to root node
  2. For each node, starting at the root:
     1. For each assigned center: Compute minimal and maximal distance to the node's hyperrectangle (i.e., the minimal and maximal distance that any point in that hyperractangle might possibly have to that center)
     2. These are also bounds on the distance to each enclosed point
     3. Remove centers from set of assigned centers with a lower bound greater than the minimal upper bound
     4. Only one center is left $\Rightarrow$ It is closest center to all enclosed points
     5. Otherwise: Assign remaining centers to both child nodes and repeat process on them
  3. Each time we reach a leaf, we only have to compute distances between leaf's points and its remaining assigned centers

Blacklisting and filtering centers with k-d trees III

Analysis:

- In Hamerly [10], competitive performance only in very low dimensions of $D \ll 8$
- Higher dimensions: Rapidly degrading performance due to curse of dimensionality!
- Constructing k-d tree is expensive and only worthwhile for big number of points
- Inflexible: Many new points $\Rightarrow$ Whole k-d tree has to be recomputed

Annular algorithm I

Annular algorithm - soring centers by norm[13]:

- Additional center pruning for methods from Section 3
- For some point x and center c, we get via triangle inequality:

$$\begin{aligned} d(c,0) - d(x,0) &\leq d(x,c) \\ \text{and } d(x,0) - d(c,0) &\leq d(x,c) \\ \Rightarrow |d(x,0) - d(c,0)| &\leq d(x,c) \end{aligned}$$

- Let $c(a)$ be the current closest center to x, $c(a_{\text{old}})$ the current position of the last closest center, c' be some candidate center and u an upper bound on $d(x, c(a_{\text{old}}))$ (and thus also on $d(x, c(a)) \leq d(x, c(a_{\text{old}}))$) and we have:

$$\begin{aligned} d(x,c') &\leq d(x, c(a_{\text{old}})) \\ \Rightarrow |d(x,0) - d(c',0)| &\leq d(x,c') \leq d(x, c(a_{\text{old}})) \end{aligned}$$

Annular algorithm II

Annular algorithm - soring centers by norm[13] (cont'd):

- Then the following must hold:

$$\begin{aligned} d(x,c') &\geq |d(x,0) - d(c',0)| \geq u \\ &\Rightarrow |d(x,0) - d(c',0)| \geq d(x, c(a_{\text{old}})) \\ &\Rightarrow d(x,c') \geq d(x, c(a_{\text{old}})) \end{aligned}$$

- We may therefore prune all centers c' with $|d(x,0) - d(c',0)| \geq u$
- Thus, only consider centers c' for point-center calculations where:

$$d(x,0) - u < d(c',0) < d(x,0) + u \tag{4}$$

- After each center movement:
  1. Recompute norm $d(c,0)$ for all centers and sort them w.r.t. their norm (overhead $\in \mathcal{O}(k \cdot D + k \cdot \log(k))$)
  2. Given a point x, use binary search to find candidate centers c' fulfilling Equation (4)
- $d(x,0)$ has to be computed only once during initialization

Heap-ordered k-means I

Heap-ordered k-means[13]:

- Restructured variant of Hamerly's method
- Inverts loop: Outer loop over centers and inner loop over assigned points
- Does not explicitly but rather implicitly keep $l^+(\cdot)$ and $u(\cdot)$
- Does not have to visit each single point
- For each center, keeps its assigned points in specifically designed min-heap data structures

Heap-ordered k-means[13] - lower memory use:

- Replace two bounds per point with single scalar value $v(i)$
- Keep another scalar value $w(j)$ for each center $c(j)$
- Both are defined, such that with $lu(i) := l^+(i) - u(i)$, we have:

$$v(i) \geq w(a_{\text{old}}(i)) \Leftrightarrow lu(i) \geq 0 \Leftrightarrow l^+(i) \geq u(i)$$
$$\Rightarrow a_{\text{old}}(i) = a(i)$$

Heap-ordered k-means II

Heap-ordered k-means[13] - computing $v(i)$ and $w(j)$:

- Initialize: $w(j) = 0$, $a(i) := 1$ and $v(i) = -1$ (latter forces algorithm to compute all distances in the first iteration)
- For each point where exact distance calculations were needed (i.e. where we also had to compute exact $l^+(i) := d(x(i), a^+(i))$ and $u(i) := d(x(i), a(i))$), update $v(i)$ as follows:

$$v(i) := lu(i) + w(a(i))$$

- After reassignments and center movements, update all $w(j)$ as:

$$w(j) := w_{\text{old}}(j) + \delta(j) + \delta(j_{\text{max},1})$$

Heap-ordered k-means III

Heap-ordered k-means[13] - properties of $v(i)$ and $w(j)$:

- During next reassignment phase, we have:

$$
\begin{aligned}
& v_{\text{old}}(i) \geq w(a_{\text{old}}(i)) \\
\Leftrightarrow\ & lu_{\text{old}}(i) + w_{\text{old}}(a_{\text{old}}(i)) \geq w(a_{\text{old}}(i)) \\
\Leftrightarrow\ & lu_{\text{old}}(i) + w_{\text{old}}(a_{\text{old}}(i)) \geq w_{\text{old}}(a_{\text{old}}(i)) + \delta(a_{\text{old}}(i)) + \delta(j_{\text{max},1}) \\
\Leftrightarrow\ & lu_{\text{old}}(i) - \delta(a_{\text{old}}(i)) - \delta(j_{\text{max},1}) \geq 0 \\
\overset{Lemma\ 3 \text{ and } 6}{\Leftrightarrow}\ & lu(i) \geq 0 \Rightarrow a_{\text{old}}(i) = a(i)
\end{aligned}
$$

- For points with $a_{\text{old}}(i) = a(i)$, we do not need to recompute $v(i)$:

$$
\begin{aligned}
v_{\text{old}}(i) &= lu_{\text{old}}(i) + w_{\text{old}}(a_{\text{old}}(i)) \\
&= lu_{\text{old}}(i) + w_{\text{old}}(a(i)) \\
&= lu_{\text{old}}(i) + w(a(i)) - \delta(a(i)) - \delta(j_{\text{max},1}) \\
&\overset{Lemma\ 3 \text{ and } 6}{=} lu(i) + w(a(i)) = v(i)
\end{aligned}
$$

Heap-ordered k-means IV

Heap-ordered k-means[13] - avoid examining all points:

- Invert loops - outer loop over centers and inner loop over points
- For each center $c(j)$, store points $x(i)$ previously assigned to it (i.e., with $a_{\text{old}}(i) = j$) in a min-heap that is sorted w.r.t. $v(i)$ (i.e., uses $v(i)$ as node key)
- Now: Exact distances, potential reassignments and updates of $v(i)$ only required for points $v(i) < w(j)$!
- Since points are sorted w.r.t. $v(i)$, no need to examine remaining points after the first encounter of $v(i) \geq w(j)$
- At the end of main loop: Update $w(j)$ for all centers

Additional improvements by Ryšavỳ and Hamerly I

General information:

- Recent publication by Ryšavỳ and Hamerly [15] (2016) introduces four improvements applicable to Elkan's, Hamerly's and the Heap method (and similar methods)
- Result in significant speedup factors of up to 8 in their experiments
- Here, we will only coarsely describe what each improvement does - see [15] for details

Additional improvements by Ryšavỳ and Hamerly II

1. Tighter lower bounds
   - Until now: When updating $l(i, j)$, implicit assumption that $c(j)$ moved directly towards $x(j)$, i.e. $l(i, j) := l_{\text{old}}(i, j) - \delta(j)$
   - [15] introduce tighter lower bounds that take direction of center movements into account
   - They derive new lower bounds for a special, 2 dimensional case based on circle and hyperbola geometry...
   - ...and a method to map general, D dimensional cases to the aforementioned special case

Additional improvements by Ryšavỳ and Hamerly III

2. Iteration over neighboring centroids:
   - Until now: If bounds fail, we have to compute distances to all k centers (or to those where Lemma 5.1 does not apply)
   - [15] introduce a method similar to Lemma 5.1 to additionally reduce the set of candidate centers
   - Given a center $c(j)$, it only requires the center's radius $r(j)$ (upper bound on the maximal distance between this center and its currently assigned points) and the center-center distances $s(j, \cdot), s_{\text{min}}(j)$
     $\Rightarrow$ No dependence on $u(i)$, $l(i, j)$ or $x(i)$ itself
     $\Rightarrow$ For each center, we may compute an initial candidate set for all its points before actually iterating over them!

Additional improvements by Ryšavỳ and Hamerly IV

3. Explicit upper bounds for Heap algorithm:
   - Original Hamerly's method: If first pruning failed, refine upper bound and try to prune again (see Algorithm 13)
   - Heap method: Does not store $u(i)$ explicitly and thus has to recompute all k point-center distances if first pruning fails
   - [15] introduce a method that is able to perform this upper bound refinement without any additional time complexity cost ...
   - ... and with an additional memory cost $\in \mathcal{O}(n + k)$

Additional improvements by Ryšavỳ and Hamerly V

4. Accelerate first iteration:

- All methods described so far: No bounds in first iteration
  $\Rightarrow$ Need to compute all point-center distances in first iteration
  $\Rightarrow$ Constitute up to 80% of total number of distance calculations in [15]'s experiments
- Given "good" assignment initialization $a_{\text{old}}(i)$ (acquired via some cheap heuristic), they propose the following:
  1. Initial upper bound: $u(i) := d(x(i), c(a_{\text{old}}(i)))$ (i.e., distance to initially assigned center)
  2. Initial second lower bound: Either zero or, if $u(i) \leq \frac{1}{2} s_{\text{min}}(a_{\text{old}})$, set $l^+(i) := s_{\text{min}}(a_{\text{old}}(i)) - u(i)$. Justification:

$$u(i) \leq \frac{1}{2} s_{\text{min}}(a_{\text{old}}(i)) \overset{Lemma\ 5}{\Rightarrow} a_{\text{old}(i)} = a(i) \tag{5}$$

Via triangle inequality:

$$\begin{aligned} d(x(i), c(a^+(i))) &\geq d(c(a(i)), c(a^+(i))) - d(x(i), c(a(i))) \\ &\geq s_{\text{min}}(a(i)) - u(i) \overset{Equation\ (5)}{=} s_{\text{min}}(a_{\text{old}}(i)) - u(i) \end{aligned}$$

References I

[1] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, Jan 2008. ISSN 0219-3116. doi: 10.1007/s10115-007-0114-2. URL https://doi.org/10.1007/s10115-007-0114-2.

[2] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28 (2):129–137, March 1982. ISSN 0018-9448. doi: 10.1109/TIT.1982.1056489.

[3] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. The planar k-means problem is np-hard. *Theoretical Computer Science*, 442:13 – 21, 2012. ISSN 0304-3975. doi: https://doi.org/10.1016/j.tcs.2010.05.034. URL http://www.sciencedirect.com/science/article/pii/S0304397510003269. Special Issue on the Workshop on Algorithms and Computation (WALCOM 2009).

[4] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53 – 65, 1987. ISSN 0377-0427. doi: https://doi.org/10.1016/0377-0427(87)90125-7. URL http://www.sciencedirect.com/science/article/pii/0377042787901257.

[5] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):790–799, Aug 1995. ISSN 0162-8828. doi: 10.1109/34.400568.

References II

[6] Wikipedia contributors. Determining the number of clusters in a data set — wikipedia, the free encyclopedia, 2018. URL https://en.wikipedia.org/w/index.php?title=Determining_the_number_of_clusters_in_a_data_set&oldid=826207721. [Online; accessed 4-April-2018].

[7] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[8] De-Yuan Cheng, A. Gersho, B. Ramamurthi, and Y. Shoham. Fast search algorithms for vector quantization and pattern matching. In *ICASSP '84. IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 9, pages 372–375, Mar 1984. doi: 10.1109/ICASSP.1984.1172352.

[9] Wikipedia contributors. Minkowski distance — Wikipedia, the free encyclopedia, 2018. URL https://en.wikipedia.org/w/index.php?title=Minkowski_distance&oldid=834553041. [Online; accessed 11-April-2018].

[10] Greg Hamerly. *Making k-means even faster*, pages 130–140. doi: 10.1137/1.9781611972801.12. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611972801.12.

[11] Charles Elkan. Using the triangle inequality to accelerate k-means. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 147–153, 2003.

[12] Jonathan Drake and Greg Hamerly. Accelerated k-means with adaptive distance bounds. In *5th NIPS workshop on optimization for machine learning*, pages 42–53, 2012.

References III

[13] Greg Hamerly and Jonathan Drake. Accelerating lloyd's algorithm for k-means clustering. In *Partitional clustering algorithms*, pages 41–78. Springer, 2015.

[14] Wikipedia contributors. Metric (mathematics) — wikipedia, the free encyclopedia, 2018. URL `https://en.wikipedia.org/w/index.php?title=Metric_(mathematics)&oldid=829118269`. [Online; accessed 29-March-2018].

[15] Petr Ryšavỳ and Greg Hamerly. Geometric methods to accelerate k-means algorithms. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pages 324–332. SIAM, 2016.

[16] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975. ISSN 0001-0782. doi: 10.1145/361002.361007. URL `http://doi.acm.org/10.1145/361002.361007`.

[17] Dan Pelleg and Andrew Moore. Accelerating exact k-means algorithms with geometric reasoning. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '99, pages 277–281, New York, NY, USA, 1999. ACM. ISBN 1-58113-143-7. doi: 10.1145/312129.312248. URL `http://doi.acm.org/10.1145/312129.312248`.

[18] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, Jul 2002. ISSN 0162-8828. doi: 10.1109/TPAMI.2002.1017616.