Parallel Computing Mid-Term: *k-means*

Giovanni Bindi

Università degli Studi di Firenze

24 February 2021

Introduction: *k-means*

K-means is a simple clustering algorithm developed during the 1950s [1]: aims to partition N observations into k clusters. Given a set of observations $\mathcal{S} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ with $\mathbf{x} \in \mathbb{R}^D$ the algorithm aims to find $k \leq N$ sets $\mathcal{C} = \{\mathcal{S}_1, \ldots, \mathcal{S}_k\}$ as to:

$$\arg\min_{\mathcal{C}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in \mathcal{S}_i} \|\mathbf{x}_i - \boldsymbol{\mu}_i\|_2^2 \tag{1}$$

where $\boldsymbol{\mu}_i$ is the mean of the points in $\mathcal{S}_i$:

$$\boldsymbol{\mu}_i = \frac{1}{|\mathcal{S}_i|} \sum_{\mathbf{x}_j \in \mathcal{S}_i} \mathbf{x}_j \tag{2}$$

Objective in Eq. 1 can be shown equal to:

$$\arg\min_{\mathcal{C}} \sum_{i=1}^{k} \frac{1}{2|\mathcal{S}_i|} \sum_{\mathbf{x}_j, \mathbf{y}_j \in \mathcal{S}_i} \|\mathbf{x}_j - \mathbf{y}_j\|_2^2 \tag{3}$$

Algorithm (Lloyd's version)

Drawback

Problem 1 is NP-hard [2] with a time complexity of $O(N^{ND+1})$ [3]

$\rightarrow$ Need for an heuristic.

Lloyd's algorithm: after initial sampling phase $\mathcal{C}^{(1)} = \{\mathcal{S}_k^{(1)}, \dots, \mathcal{S}_k^{(1)}\}$ for $t = 1, \dots, I$ iterations repeat two phases:

1. **Assignment** step: where each data point is assigned to the cluster with the nearest mean, i.e. building $\mathcal{S}_i^{(t)}$:

$$\mathcal{S}_i^{(t)} = \left\{\mathbf{x}_p : \|\mathbf{x}_p - \boldsymbol{\mu}_i^{(t)}\|^2 \leq \|\mathbf{x}_p - \boldsymbol{\mu}_j^{(t)}\|_2^2\right\} \tag{4}$$

   for all $\mathbf{x}_p \in \mathcal{S}$ and for all $j \neq i$.

2. **Update** step: where the centroids are recomputed, based on the previous step calculations:

$$\boldsymbol{\mu}_i^{(t+1)} = \frac{1}{|\mathcal{S}_i^{(t)}|} \sum_{\mathbf{x}_j \in \mathcal{S}_i^{(t)}} \mathbf{x}_j \tag{5}$$

Pseudo-code: Sequential Version

Input: $\mathcal{S} \subset \mathbb{R}^D$, $k, I \in \mathbb{N}$. Optional: $\epsilon \in \mathbb{R}$.
Output: Clusters $\mathcal{C}$ with centroids $\{\boldsymbol{\mu}_1, \dots \boldsymbol{\mu}_k\}$

$\boldsymbol{\mu}^{(1)} = \{\boldsymbol{\mu}_1^{(1)}, \dots, \boldsymbol{\mu}_k^{(1)}\} \leftarrow \texttt{sample}(\mathcal{S})$
for $t = 1, \dots, I$ **do**
    $\boldsymbol{\tau} = \{\mathbf{0}, \dots, \mathbf{0}\}$
    $\eta = \{0, \dots, 0\}$
    for $i = 1, \dots, N$ **do**
        $c = \texttt{closest\_centroid}(\mathbf{x}_i, \boldsymbol{\mu}^{(t)})$
        $\boldsymbol{\tau}_c = \boldsymbol{\tau}_c + \mathbf{x}_i$
        $\eta_c = \eta_c + 1$
    end for
    for $j = 1, \dots, k$ **do**
        $\boldsymbol{\mu}_j^{(t+1)} = \boldsymbol{\tau}_j / \eta_j$
    end for
    if $\|\boldsymbol{\mu}^{(t+1)} - \boldsymbol{\mu}^{(t)}\|_2^2 \leq \epsilon$ **then**
        return $\boldsymbol{\mu}^{(t+1)}$
    end if
end for
return $\boldsymbol{\mu}^{(I)}$

Pseudo-code: Parallel Version

Input: $\mathcal{S} \subset \mathbb{R}^D$, $k, I \in \mathbb{N}$. $T \in \mathbb{N}$. Optional: $\epsilon \in \mathbb{R}$.
Output: Clusters $\mathcal{C}$ with centroids $\{\boldsymbol{\mu}_1, \dots \boldsymbol{\mu}_k\}$

$\boldsymbol{\mu}^{(1)} = \{\boldsymbol{\mu}_1^{(1)}, \dots, \boldsymbol{\mu}_k^{(1)}\} \leftarrow \texttt{sample}(\mathcal{S})$
$\mathcal{P} \leftarrow \texttt{init\_pool}(T)$
for $t = 1, \dots, I$ **do**
  $\boldsymbol{\tau} = \{\mathbf{0}, \dots, \mathbf{0}\}$
  $\eta = \{0, \dots, 0\}$
  for each thread $p \in \mathcal{P}$ **do**
    $(s, e) \leftarrow \texttt{compute\_start\_end}(p, N)$
    for $i = s, \dots, e$ **do**
      $c = \texttt{closest\_centroid}(\mathbf{x}_i, \boldsymbol{\mu}^{(t)})$
      $\boldsymbol{\tau}_c = \boldsymbol{\tau}_c + \mathbf{x}_i$
      $\eta_c = \eta_c + 1$
    end for
  end for
  $\texttt{wait\_all\_threads()}$
  for $j = 1, \dots, k$ **do**
    $\boldsymbol{\mu}_j^{(t+1)} = \boldsymbol{\tau}_j / \eta_j$
  end for
  if $\|\boldsymbol{\mu}^{(t+1)} - \boldsymbol{\mu}^{(t)}\|_2^2 \leq \epsilon$ **then**
    return $\boldsymbol{\mu}^{(t+1)}$
  end if
end for
return $\boldsymbol{\mu}^{(I)}$

Thread Pool

```
class thread_pool {

    private:

        std::vector<std::thread> workers;
        std::deque<std::function<void()>> tasks;
        std::mutex mtx;
        std::condition_variable task_done;
        std::condition_variable all_done;

        unsigned int running_tasks;
        bool stop;

        void run_task() {
            while (true) {
                std::unique_lock<std::mutex> lock(mtx);
                task_done.wait(lock, [this](){ return stop || !tasks.empty(); });
                if (!tasks.empty()) {
                    ++running_tasks;
                    auto func = tasks.front();
                    tasks.pop_front();
                    lock.unlock();
                    func();
                    lock.lock();
                    --running_tasks;
                    all_done.notify_one();
                }
                else if (stop)
                    break;
            }
            return;
        }
    ...
```

Thread Pool - 2

```
    ...
    public:

        thread_pool(const unsigned int T = std::thread::hardware_concurrency()) :
            running_tasks(),
            stop() {
                for (unsigned int i = 0; i < T; ++i)
                    workers.emplace_back(std::bind(&thread_pool::run_task, this));
            }

        ~thread_pool() {
            std::unique_lock<std::mutex> lock(mtx);
            stop = true;
            task_done.notify_all();
            lock.unlock();
            for (auto& w : workers)
                w.join();
        }

        template<class F>
        void enqueue(F&& f) {
            std::unique_lock<std::mutex> lock(mtx);
            tasks.emplace_back(std::forward<F>(f));
            task_done.notify_one();
            return;
        }

        void wait_all_threads() {
            std::unique_lock<std::mutex> lock(mtx);
            all_done.wait(lock, [this](){ return tasks.empty() && (running_tasks == 0); });
            return;
        }
};
```

Experiments

- 4C / 8T CPU (Intel i7-8550U).
- No ϵ-stop $\rightarrow I = 1000$ iterations.
- Ten runs for each experiment:

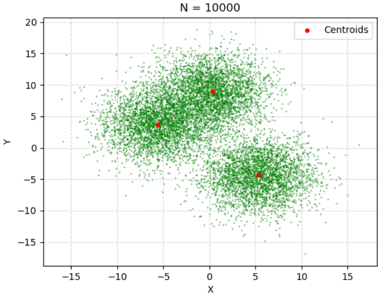
$$\bar{t}_A = \frac{1}{10} \sum_{i=1}^{10} t_A^{(i)} \tag{6}$$

- Speedup of B over A:

$$S = \bar{t}_A / \bar{t}_B \tag{7}$$

- Variance:

$$\sigma^2 \approx S^2 \left[\frac{\sigma_A^2}{\bar{t}_A^2} + \frac{\sigma_B^2}{\bar{t}_B^2} \right] \tag{8}$$



- Four dataset sizes $N = \{10000, 50000, 100000, 500000\}$.
- Two dimensionalities $D = \{2, 3\}$.
- Three centroids.

General Behaviour

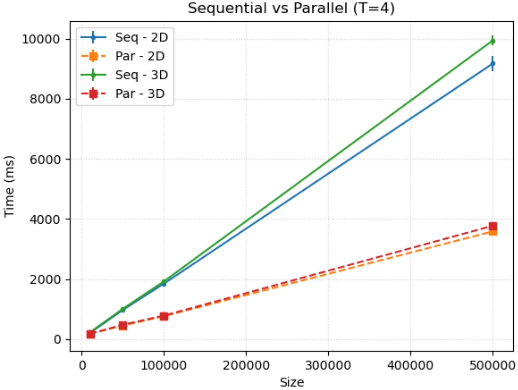


Figure 1: Execution time (ms): Sequential vs Parallel (with $T = 4$)

Against Sequential

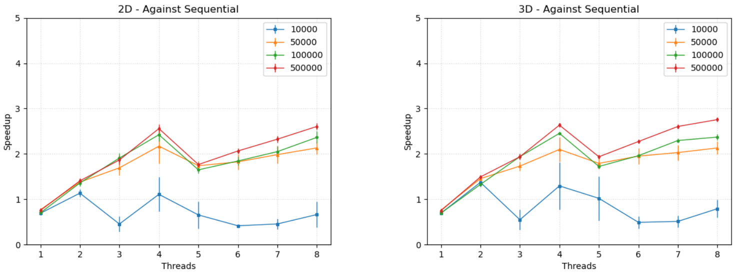


Figure 2: Speedup: Parallel vs Sequential for $2D$ (*left*) and $3D$ (*right*) datasets

- **Maximum** speedup: $S^{max} = 2.76$ for $N = 5 \cdot 10^5$, $D = 3$ and $T = 8$.
- **Minimum** speedup: $S^{min} = 0.42$ for $N = 1 \cdot 10^4$, $D = 2$ and $T = 6$.

Against $T = 1$

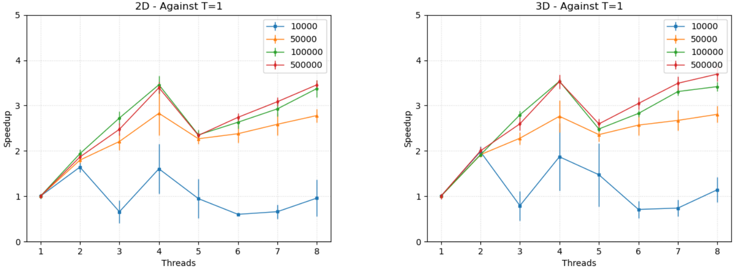


Figure 3: Speedup: Parallel vs $T = 1$ for $2D$ (*left*) and $3D$ (*right*) datasets

- **Maximum** speedup: $S^{max} = 3.69$ for $N = 5 \cdot 10^5$, $D = 3$ and $T = 8$.
- **Minimum** speedup: $S^{min} = 0.60$ for $N = 1 \cdot 10^4$, $D = 2$ and $T = 6$

Conclusions

Two implementations of *k-means*: sequential and parallel.

Pros:

- Parallelization method: thread pool & task queue.
- Header-only C++17 libraries, with simple test cases.
- Obtained a perceivable speedup.

Cons:

- Speedup is small.

Code

https://github.com/w00zie/kmeans

References I

[1] S. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982. DOI: 10.1109/TIT.1982.1056489.

[2] D. Aloise, A. Deshpande, P. Hansen, and P. Popat, "Np-hardness of euclidean sum-of-squares clustering," *Mach. Learn.*, vol. 75, no. 2, pp. 245–248, May 2009, ISSN: 0885-6125. DOI: 10.1007/s10994-009-5103-0. [Online]. Available: https://doi.org/10.1007/s10994-009-5103-0.

[3] M. Inaba, N. Katoh, and H. Imai, "Applications of weighted voronoi diagrams and randomization to variance-based k-clustering: (extended abstract)," in *Proceedings of the Tenth Annual Symposium on Computational Geometry*, ser. SCG '94, Stony Brook, New York, USA: Association for Computing Machinery, 1994, pp. 332–339, ISBN: 0897916484. DOI: 10.1145/177424.178042. [Online]. Available: https://doi.org/10.1145/177424.178042.