Survey of the k-Means Clustering Problem

MICHAEL PÉREZ, University of Florida, USA

Abstract The k-means clustering problem was first described in the 1950s, and since then numerous heuristic algorithms have been developed that converge to a local optimum. The problem is computationally difficult, or NP-Hard. This paper will survey significant milestones in the development of heuristic algorithms for the problem: Lloyd's algorithm, the 'Blacklisting' algorithm, and several improved algorithms that use the triangle inequality. The proofs which establish the problem's NP-hardness when k is fixed and in the plane will be also described.

Additional Key Words and Phrases: k-means, Clustering, Data Mining, sum-of-squares, NP-hardness

1 INTRODUCTION

Given a dataset and a constant k, the k-means clustering problem is the intuitive optimization problem of how to partition the data into k subsets so that each subset optimizes some measure. [13] [8] For example, one might want to cluster two-dimensional data points into k subsets such that the variance between each subset is minimized. This problem was first studied in the domain of signal processing; however, it has numerous applications today such as image segmentation and unsupervised learning. [10] [8] Today, the computer science, statistics, and mathematics literature on this subject is quite dense.

Throughout this paper, unless otherwise noted, the number of points are denoted by n, the number of dimensions by m, and the number of centers by k. Let the Euclidean distance between two points x and y in m dimensions be $distance(x, y) = \sqrt{\sum_{i=1}^{m} (y_i - x_i)^2}$. Now to more formally define the Euclidean k-means clustering problem:

Definition 1.1. Given a set of n points in m dimensions, $S = \{p_1, p_2, ..., p_n\}$, as input, find a set of k points, $C = \{c_1, c_2, ..., c_n\}$, that minimizes the distortion $D(S, C) = \sum_{i=1}^{n} (\min_{1 \le j \le k} distance(p_i, c_j))^2$. [11]

Lloyd invented an algorithm for clustering signals in pulse-code modulation (PCM) in 1955, entitled Lloyd's algorithm in 1955. (though he did not publish it until 1982) [10] Like all (known) algorithms that can solve this problem, Lloyd's algorithm computes a local optimum, but not necessarily a global optimum. [11] In 1992, Pelleg and Moore used the k-d tree data structure to improve the search for each point's center and avoid many distance calculations, thereby accelerating Lloyd's algorithm. [13] [8] In 2015 Hamerly and Drake surveyed previous algorithms and described two new algorithms that can speed up Lloyd's algorithm by over 30-50x. [8] In 2008, Aloise et al. provided a valid proof of the NP-hardness of the k-means clustering problem when the dimension m is part of the input and $k = 2$. [1] In 2009, Mahajan et al. established the NP-hardness of the problem when the dimension m is 2, or in the plane, and k is part of the input. [11]

2 BRUTE FORCE APPROACH, OR LLOYD'S ALGORITHM

The brute force k-means algorithm $LLOYD(S, C)$ clusters the n points into k clusters by partitioning the data into k subsets, so that all points in a subset correspond to a center. [10] [13] The algorithm progresses iteratively and keeps track of the centers of the subsets. [10] [13] Let the set of centers after the ith iteration be C^i. The algorithm can be summarized in three basic steps:

(1) initialize the centers k
(2) while $C^i \neq C^{i+1}$:
   (a) assign each data point x to its nearest center in C^i
   (b) compute C^{i+1} by finding the center of mass of points corresponding to each center [10] [13] [8]

Author's address: Michael Pérez, University of Florida, Gainesville, USA.

The initialization step is important; poor initialization can lead to empty clusters and a higher likelihood of getting stuck in bad local minima. [6] A comprehensive study by Celebi et al. showed that the k-means++ method and Bradley and Fayyad's method are the best initialization strategies. [2] [6] [4] When $C^i = C^{i+1}$, the centers do not change after an iteration and the algorithm converges. [13] The algorithm is guaranteed to converge because steps 2a and 2b always decrease the distortion $D(S, C)$, and the number of possible partitions of n points into k clusters is finite. [8] [3]

Input : S, C
Output : C
initialize the centers C
while $C^i \neq C^{i+1}$ **do**
  foreach $i \in S$ **do**
    $temp\_center[i] \leftarrow 1$
    foreach $j \in C$ **do**
      if $distance(s(i), c(j)) < distance(s(i)), c(temp\_center[j])$ **then**
        $center[i] \leftarrow j$
      end
    end
  end
  foreach $j \in C$ **do**
    set each center $c(j)$ to mean of its assigned points $\{x(i) | temp\_center(i) = j\}$
  end
end

Algorithm 1: $LLOYD(S, C)$ [10] [8]

2.1 Running Time Analysis

$LLOYD(S, C)$ calculates the nearest center for each of the n points, during which each of the points' distance to k centers in m dimensions is calculated. [13] $distance(x, y)$ has a running time complexity of $O(m)$. Therefore, the running time complexity of each iteration is $O(nmk)$. [10] [13] If there are i iterations, the total running time of Lloyd's algorithm is $O(nmki)$. [8]

3 IMPROVEMENTS USING GEOMETRIC REASONING

Pelleg and Moore accelerated the brute force algorithm by utilizing the k-d tree data structure. [13] A k-d tree is a data structure that organizes points in k dimensional space. [13] (the k in k-d trees represents the dimension, while the k in k-means algorithm represents the number of centers) k-d trees are binary search trees in which every leaf node is a k-dimensional point and every non-leaf node represents a hyperplane which recursively splits the space into two parts. [13] [8] The root node corresponds to a hyperplane which contains all points. [13] The depth of each node corresponds to the axis that its hyperplane separates on. Each node is associated with one of the k dimensions.

3.1 The "Blacklisting" Algorithm

Pelleg and Moore's 'Blacklisting' algorithm $BLACKLISTING(S, C)$ first creates a k-d tree T from the n points. [13] The algorithms then proceeds iteratively until the centers converge, similar to Lloyd's algorithm. [13] In contrast, at each iteration $BLACKLISTING(S, C)$ calls a helper function $UPDATE(h, C)$. [8] This function performs a traversal of the k-d tree T to find subtrees that are owned by one center, which corresponds to eliminating

distance calculations between the furthest points and centers. [13] [8] The algorithm uses the tree's structure to avoid unnecessary point-center distance calculations, or 'blacklist' the centers. [13] [8]

Input : S, C
Output : C
create a k-d tree T from the data S
while $C^i \neq C^{i+1}$ **do**
  $UPDATE(T.root, C)$
  set each center $c(j)$ to mean of its assigned points $\{x(i)|center(i) = j\}$
end

Algorithm 2: $BLACKLISTING(S, C)$ [13] [8]

$UPDATE(h, C)$ is first called on the root node of the k-d tree; all k centers start there. [13] [8] The algorithm then recursively descends the tree and at each non-leaf node, it checks if any center in C is closer than one or more other centers to every point in the hyperplane corresponding to the current node, or if any centers are 'dominated' by other centers. [13] [8] The dominated centers are subsequently removed from C. [13] [8] When one center remains in C, the points below it in the tree are assigned to that center because it dominates the other centers with respect to those points. [13] [8] Otherwise, $UPDATE(h, C)$ is called recursively on the node's right and left nodes in the k-d tree until one dominating center remains in C. [13] [8] The recursive algorithm $UPDATE(h, C)$ is called iteratively by $BLACKLISTING(S, C)$ until the centers converge. [13] [8]

Input : h, C
Output : C
if *h is a leaf* **then**
  foreach *point* $x \in h$ **do**
    find the closest center to x and update its counters
  end
else
  compute $distance(c, h)$ for all centers $c \in C$
  remove dominated centers from C
  if $|C| = 1$ **then**
    update counters for the dominating center in C using data in h
  else
    $UPDATE(h.left, C)$
    $UPDATE(h.right, C)$
  end
end

Algorithm 3: $UPDATE(h, C)$ [13] [8]

3.2 Running Time Analysis

$BLACKLISTING(S, C)$ avoids unnecessary distance calculations in iterations when it uses the tree's structure to identify dominating centers near the root of the recursion, thereby accelerating the brute force method. [8] Kanungo et al. independently developed a similar algorithm, and determined the running time complexity is a complex expression dependent upon the cluster separation. [9] Although $BLACKLISTING(S, C)$ is faster than $LLOYD(S, C)$ in low dimensions, $BLACKLISTING(S, C)$ performs poorly in dimensions above 8 since the cost of using k-d trees increases exponentially with dimension. [13] [8]

4 OPTIMIZATIONS USING TRIANGLE INEQUALITY

The algorithms in this section use the triangle inequality from geometry to optimize Lloyd's algorithm for the k-means clustering problem. [8] The triangle inequality can be understood intuitively as stating that the sum of any two sides of a triangle is at least the length of the third side. More formally, the triangle inequality states that for three points a, b, and c in m dimensions, $distance(a, c) \leq distance(a, b) + distance(b, c)$. [8] The triangle inequality can be used to improve the k-means algorithm in various ways. [8]

4.1 Phillips' Compare-Means and Sort-Means Algorithms

Phillips first used the triangle inequality to accelerate k-means clustering, with the development of two algorithms: Compare-Means and Sort-Means. [14] [8] The Compare-Means algorithm uses the triangle inequality to prove that given two centers c and c' and a point x, if one center c is close to x and another center c' is far from c, then c must be closer to x than to c'. [14] [8] This inequality is used in the innermost loop over the centers in k-means to eliminate some point-center calculations without calculating $distance(x, c)$. [14] [8]

The Sort-Means algorithm uses a k x k matrix to compute center-center distances. [14] [8] The matrix values are recomputed and each row is sorted at each iteration of the main loop whenever the centers move. [14] [8] This facilitates an efficient search for the nearest center c to a point x, by searching each center in order from the nearest to c first and the furthest from c last. [14] [8] The algorithm uses the same inequality as Compare-Means but searches the centers in a more efficient order. [14] [8] The extra overhead of sorting each row each time the centers are updated is a disadvantage compared to Compare-Means. [14] [8]

4.2 Elkan's Algorithm

The triangle inequality can be utilized further to maintain upper and lower bounds from a point to an updated center without calculating the point-center distance, for use in ruling out certain distance calculations. [8] [5] Elkan's algorithm uses one upper bound for each point, stored in $u(i)$, that represents the maximum distance between a point and its nearest center. [8] The algorithm also uses k lower bounds for each point, stored in $l(i, j)$, that represent the minimum distance from each point to each of the k centers. [8] The algorithm efficiently updates these bounds after each iteration of the main loop by adding or subtracting the distance that each center moves. [8] At each iteration, the algorithm also stores half the distance from each center $c(j)$ to its closest other center in $q(j)$. [8]

The if statement that tests if $u(i) \leq q(a(i))$ checks if any centers could be closer to $x(i)$ than the currently assigned center; if so, the algorithm continues to the next iteration. [8] In the innermost loop over the centers, the if statements check if $u(i) \leq max(l(i, j), distance(c(j), c(a(i)))/2)$, because if so, then it is not possible for the current center $c(j)$ to be the closest center to the current point and the algorithm continues to the next iteration. [8]

4.3 Other Algorithms

Hamerly and Drake describe several algorithms that are related to Elkan's and use distance bounds for the k-means problem in their survey. [8] Hamerly's algorithm keeps one upper bound and one lower bound for each point. [8] Drake's algorithm keeps one upper bound and b lower bounds per point, where b is chosen adaptively. [8] The Annular algorithm is like Hamerly's and has one upper bound and one lower bound per point, but each center is norm-ordered. [8] The Heap-ordered algorithm uses k heaps of assigned points, ordered by the bounded distance from the center, and the upper and lower bounds are combined into one value. [8]

Elkan stated that approaches to the k-means problems that use trees, like $BLACKLISTING(S, C)$ discussed in Section 3, are often asymptotically slower than the algorithms discussed in Section 4. [8] He attributed this to how $BLACKLISTING(S, C)$ initially creates a static tree from the points before beginning clustering, without any

```
Input  : S, C
Output : C
temp_center(i) ← 1, u(i) ← ∞, ∀ i ∈ N
l(i, j) ← 0, ∀ i ∈ N, j ∈ K
while C^i ≠ C^{i+1} do
    compute distance(c(j), c(j')), ∀j, j' ∈ K
    q(j) ← min_{j≠j'}{distance(c(j), c(j')/2}, ∀j ∈ K
    foreach point i ∈ N do
        if u(i) ≤ q(temp_center(i)) then
            continue
        else
            r ← True
            foreach center j ∈ K do
                z ← max(l(i, j), distance(c(j), c(a(i)))/2)
                if j = temp_center(i) or u(i) ≤ z then
                    continue
                else
                    if r then
                        u(i) ← distance(c(temp_center(i), s(i))
                        r ← False
                        if u(i) ≤ z then
                            continue
                        end
                    else
                        l(i, j) ← distance(c(j), s(i))
                        if l(i, j) ≤ u(i) then
                            temp_center(i) ← j
                        end
                    end
                end
            end
        end
    end
    foreach center j ∈ K do
        move c(j) to new location
        w(j) ← distance moved by c(j)
    end
    foreach i ∈ N do
        u(i) ← u(i) + w(temp_center(i))
        foreach center j ∈ K do
            l(i, j) ← l(i, j) − w(j)
        end
    end
end
```

Algorithm 4: ELKAN(S,C) [8] [5]

information about how many clusters there will be when the algorithm terminates. [8] The number of clusters changes throughout clustering so using a static tree is not the best approach to this problem. [8]

The accelerated algorithms (Elkan, Hamerly, Annular, Drake, and Heap) are much faster than Compare-Means and Sort-Means when the dimension is greater than 8, since the latter two do not use any distance bounds. [8] In experimental results, it is common to see the algorithms accomplish speedups of 30-50x times compared to the brute force approach. [8] Clearly, using the triangle inequality to maintain distance bounds and avoid unnecessary expensive computations is the best heuristic for the k-means problem. [8]

5 NP-HARDNESS WHEN $k = 2$

In 2004, Drineas et al. proposed a proof that the k-means clustering problem is NP-hard when $k = 2$ and for arbitrary dimension, via a reduction from the minimum bisection problem. [1] [7]

Definition 5.1. For a given graph $G = (V, E)$, the minimum bisection problem is how to bisect it into two parts such that each part has an equal number of nodes and the number of edges crossing the two parts is minimized. [1] [7]

In 2009, Aloise et al. showed that the NP-hardness proof by Drineas et al. was not valid, then provided an alternate proof of the same result. [1] Aloise et al. noticed in the original proof that when Drineas et al. calculate the sum of squares distances from each point to its center, they forget to add certain components in the expression. [1] After correcting the oversight, the cost does not depend on the cardinality of the clusters so the original proof is invalid. [1]

5.1 Proof by Reduction from Densest Cut Problem

Next, Aloise et al. provided an alternate valid proof of the problem's NP-hardness by a reduction from the densest cut problem. [1]

Definition 5.2. For a given graph $G = (V, E)$, the densest cut problem is how to cut the graph such that the ratio $E(P, Q)/|P| * |Q|$ is maximized over all bipartitions (P, Q) of the vertices in G, where $E(P, Q)$ represents the set of edges the cut crosses. [1]

The densest cut problem on G is equivalent to the sparsest cut problem on G's complement graph, which was shown to be NP-hard in 1990. [1] [12] The polynomial transformation that reduces the densest cut problem to the k-means problem has as input a graph $G = (V, E)$ with no parallel edges, and produces a matrix M of size $|V|$ x $|E|$. [1] If a vertex $v \in V$ is connected to an edge $e \in E$, then the entry in the matrix $M(v, e)$ is 1 for one endpoint of the edge and -1 for the other, arbitrarily. [1] Otherwise, $M(v, e)$ is 0. [1] After the transformation, each column in M has one cell that is 1, one that is -1, and the rest are 0. [1]

If each row of M is mapped to a point in $|E|$-dimensional space and $|P| = p$, $|Q| = q$, and $p + q = n$, when computing the cost of a bipartition into two clusters P and Q, the centers of each cluster have either $1/p$ or $-1/p$ as their e-th coordinate if $e \in E(P, Q)$, and 0 otherwise. [1] Aloise et al. show that the total cost simplifies to $2|E| - \frac{n*|E(P,Q)|}{p*q}$. [1] The k-means clustering problem with $k = 2$ minimizes the expression, which corresponds to maximizing the ratio $E(P, Q)/|P| * |Q|$ and finding the densest cut in graph G, concluding the reduction. [1]

6 NP-HARDNESS IN THE PLANE

In 2012, Mahajan et al. showed that the k-means problem, when $k \geq 1$ and in the plane, is NP-hard via a reduction from the planar 3-SAT problem. [11] The planar 3-SAT problem is an extension of the boolean 3-satisfiability problem discussed in class, when the boolean formula's incidence graph lies on a plane.

Definition 6.1. The input to the 3-SAT problem is a 3-CNF formula F with variables $v_1, v_2, ..., v_n$ and clauses containing those variables $c_1, c_2, ..., c_m$. Let the graph of F be $G(F) = (V, E)$, such that

$$V = \{v_i | 1 \leq i \leq n\} \cup \{c_i | 1 \leq j \leq m\}$$
$$E = \{(v_i, c_i)\} | v_i \in c_j \text{ or } \overline{v_i} \in c_j\} \cup \{(v_j, v_{j+1}\} | 1 \leq j \leq n\} \cup \{\{v_n, v_1\}\}. \text{ [11]}$$

The planar 3-SAT problem is to determine whether a given planar 3-CNF formula is satisfiable or not. [11]

The authors first describe several properties that the k-means problem instance constructed from a given planar 3-SAT problem will have, then prove a series of lemmas and claims that are later used in the reduction. [11] The authors describe a procedure of six steps to transform a planar 3-SAT formula into a desired instance of the k-means problem. [11] The reduction is particularly long and quite complex, so I leave it out for the sake of brevity. [11]

7 OPEN QUESTIONS

Since the k-means problem has been proven to be NP-Hard, every problem in NP can be reduced to it. [1] [11] Therefore, finding a polynomial time solution to any problem in NP would solve every problem in NP, including this problem. However, since it is suspected that $P \neq NP$, it is unlikely that a polynomial time algorithm for this problem exists and the algorithms discussed in this paper will likely continue to be the best methods.

8 CONCLUSION

The algorithms and proofs discussed suggest that although the k-means clustering problem has been proven to be NP-hard, efficient heuristic algorithms that converge quickly to a solution exist. Until the question "if $P = NP$?" is definitively answered, heuristic algorithms that use the triangle inequality from geometry will continue to be the most efficient approach for this problem.

REFERENCES

[1] Hansen P Aloise D, Deshpande A. 2009. NP-hardness of Euclidean sum-of-squares clustering. *Mach Learn* 75 (2009), 245–248. https://doi.org/10.1007/s10994-009-5103-0

[2] David Arthur and Sergei Vassilvitskii. 2007. K-Means++: The Advantages of Careful Seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (New Orleans, Louisiana) *(SODA '07)*. Society for Industrial and Applied Mathematics, USA, 1027–1035.

[3] Léon Bottou and Yoshua Bengio. 1994. Convergence Properties of the K-Means Algorithms. In *Proceedings of the 7th International Conference on Neural Information Processing Systems* (Denver, Colorado) *(NIPS'94)*. MIT Press, Cambridge, MA, USA, 585–592.

[4] Paul S. Bradley and Usama M. Fayyad. 1998. Refining Initial Points for K-Means Clustering. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML '98)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 91–99.

[5] Elkan C. 2003. Using the triangle inequality to accelerate k-means. *Proceedings of the twentieth international conference on machine learning (ICML)* (2003), 147–153.

[6] M. Emre Celebi, Hassan A. Kingravi, and Patricio A. Vela. 2012. A Comparative Study of Efficient Initialization Methods for the K-Means Clustering Algorithm. *CoRR* abs/1209.1960 (2012). arXiv:1209.1960 http://arxiv.org/abs/1209.1960

[7] Frieze A Kannan R et al. Drineas, P. 2004. 56 (07 2004), 9–33.

[8] Greg Hamerly and Jonathan Drake. 2015. Accelerating Lloyd's Algorithm for k-Means Clustering. (10 2015), 41–78. https://doi.org/10.1007/978-3-319-09259-1_2

[9] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. 2002. An Efficient K-Means Clustering Algorithm: Analysis and Implementation. *IEEE Trans. Pattern Anal. Mach. Intell.* 24, 7 (July 2002), 881–892. https://doi.org/10.1109/TPAMI.2002.1017616

[10] S. Lloyd. 1982. Least Squares Quantization in PCM. *IEEE Transactions on Information Theory* 28, 2 (1982), 129–137. https://doi.org/10.1109/TIT.1982.1056489

[11] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. 2012. The Planar k-means Problem is NP-Hard. *Theoretical Computer Science* 442 (2012), 13 – 21. https://doi.org/10.1016/j.tcs.2010.05.034 Special Issue on the Workshop on Algorithms and Computation (WALCOM 2009).

[12] D. W. Matula and F. Shahrokhi. 1990. Sparsest Cuts and Bottlenecks in Graphs. *Discrete Appl. Math.* 27, 1–2 (April 1990), 113–123. https://doi.org/10.1016/0166-218X(90)90133-W

[13] Dan Pelleg and Andrew Moore. 1999. Accelerating Exact k-means Algorithms with Geometric Reasoning. (07 1999). https://doi.org/10.1145/312129.312248

[14] Steven J. Phillips. 2002. Acceleration of K-Means and Related Clustering Algorithms. In *Algorithm Engineering and Experiments*, David M. Mount and Clifford Stein (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 166–177.