Analysis of Lloyds k-Means Clustering Algorithm using kd-Trees

ERIC WENGROWSKI, Rutgers University

k-means is a commonly-used classification algorithm, but is difficult to implement because it is computationally NP-hard. However, many heuristic algorithms, such as Lloyd's k-means algorithm provide locally optimal results with drastically less computational difficulty. In this paper, we will investigate one particular implementation of Lloyd's algorithm that employs kd-trees. We explore k-means clustering using an implementation of Lloyd's algorithm within $O(dn * \log(n))$ time using a kd-tree that has $O(n)$ nodes and $O(log(n))$ depth.

General Terms: k-Means Clustering Algorithm, Lloyd Algorithm, Algorithm Design and Analysis, Machine Learning Algorithms, Clustering Algorithms, pattern clustering, filtering theory, covariance matrices, data structure, k-d tree, nearest-neighbor searching

1. INTRODUCTION

k-means is a well-known clustering algorithm. It is used to segment N data points in n-dimensional space are grouped into K mutually exclusive clusters, where K is already known. k-means is an iterative approach to classification. At each iteration, each data point is assigned to the cluster with the nearest mean point. The myriad of applications where k-means serves as a useful method of classification are well-known and established.

2. *K*-MEANS

The k-means algorithm partitions N data points into K mutually exclusive subsets S_j each containing N_j points. The goal of k-means is to minimize the sum-of-squared distances between each data point and its corresponding geometric mean point μ_j. [MacQueen et al. 1967]

$$J = \sum_{j=1}^{K} \sum_{n \in S_j} |x_n - \mu_j|^2$$

Ideally, k-means produces a global minimum sum-of-squared distances J. However, this is difficult to practically implement, and is computationally NP-hard [Mahajan et al. 2009] [Aloise et al. 2009]. Instead, the local minimum J is computed with the following recursive procedure: [Nigrin 1993]

3. LLOYD'S HEURISTIC ALGORITHM

Lloyd's Generalized algorithm is one such iterative k-means approximation that converges at a local optimum.

Author's addresses: E. Wengrowski, Electrical and Computer Engineering Department, Rutgers University.

DOI:http://dx.doi.org/10.1145/0000000.0000000

ALGORITHM 1: k-means

```
1: procedure KMEANS
2:     the data points are assigned at random to the K sets
3: while the centroid position changes do
4:     recompute the centroid
5:     reassign data points to their nearest centroid
       end
```

> Lloyd's algorithm is based on the simple observation that the optimal placement of a center is at the centroid of the associated cluster. Given any set of k centers Z, for each center $z \in Z$, let $V(z)$ denote its neighborhood, that is, the set of data points for which z is the nearest neighbor. In geometric terminology, $V(z)$ is the set of data points lying in the Voronoi cell of z. Each stage of Lloyd's algorithm moves every center point z to the centroid of $V(z)$ and then updates $V(z)$ by recomputing the distance from each point to its nearest center. These steps are repeated until some convergence condition is met. [Kanungo et al. 2002]

Lloyd's algorithm can actually also be described by **Algorithm 1** in the pseudo code listed above.

However, a straightforward implementation of Lloyd's algorithm is computationally heavy. This is principally due to the cost of computing the nearest center adjacent to each data point [Kanungo et al. 2002]. Knuth referred to this as the *post office problem* [Knuth 1973].

4. KD-TREE IMPLEMENTATION

A kd-tree is a binary tree. Nodes within the tree are organized into k-dimensional space. Each non-leaf represents a hierarchical subdivision of the into two hyperspaces known as half-spaces using a hyperplane. The hyperplane is orthogonal to each dimensional axis. Therefore, every node in the tree is associated with one of the k-dimensions. According to [De Berg et al. 2000], a kd-tree is subject to the following two constraints:

(1) The depth of each node corresponds to axis that its hyperplane separates
   — Each step down the tree increments the axis that is being described from $axis_1$ to $axis_k$
   — The axis-depth cycles repeats after $axis_k$
(2) Points are inserted into the kd-tree using the median value associated with the axis-determined hyperplane
   — This allows the entire data set to be inserted into the tree initially

The authors of [Kanungo et al. 2002] design an algorithm employs a kd-tree to store data points. The intuition is that a kd-tree need be built only once for a set of data points. However, the idea of storing points in a kd-tree for clustering is not new. This idea has been previously presented for classification purposes by A. Moore [Moore 1999], Alsabti [Alsabti et al. 1997], Kanungo in previous works [Kanungo et al. 1999], and many others.

This paper explores the implementation of Lloyd's algorithm by [Kanungo et al. 2002], where a kd-tree is used to maintain a subset of candidate centers for each node of the tree. The advantage of the kd-tree is principally the storing of multidimensional data points. Each data point belongs to a axis-aligned hyper-rectangle known as a *box*.

Each of the k clusters is defined by its box. A *bounding box* refers to the smallest box that contains all of the respective points. The axis-aligned splitting hyperplanes found in kd-trees define each point set's bounding box. Given textitn points, this produces a kd-tree with $O(n)$ nodes and $O(log(n))$ depth.

For any k-means estimation, the weighted centroid of each of the k regions must be calculated at each iteration. This corresponds to the k bounding boxes built into the kd-tree. Furthermore, for each of the k centers, the set of data points for which this centroid is closest must also be calculated. The algorithm described by [Kanungo et al. 2002] computes weighted centroid C_W defined to be the vector sum its associated data points, and the actual number of associated data points N_k. The actual centroid C_A is then simply C_W/N_k .

Each node in the kd-tree has an associated set of potential candidate centers. Each set contains the list of center points Z that could potentially serve as the nearest neighbor for a respective point. All k centers are candidate centers for the root. Let B denote the closed bounding box associated with each node. For each node, the candidate center z_0 that is closest to the midpoint of B is first computed and added to the set Z. Then, each other potential candidate center z is considered. If no part of the bounding box B is closer to each new candidate center than the original, they are not added to that node's candidate center set Z. This allows the authors of [Kanungo et al. 2002] to filter each impossible candidate center z from an entire set of nodes. This is enabled by the hierarchically-sorted nature of the kd-tree. Once a node contains only one candidate center z within its set of potential candidate centers Z, that node is assigned to the remaining z.

To compare the distances between candidate centers z to any part of B, the vertex of B that maximizes the vector distance between candidate centers is computed. The squared distance between each candidate center and the closest section of B are compared with one another. Whichever candidate center z has the minimal squared distance is added to Z, which the further center is removed. If two candidates lie within the cell, this approach will select the candidate that is closer to the cell's midpoint.

> Efficiency is achieved because the data points do not vary throughout the computation and, hence, this data structure does not need to be recomputed at each stage. [Kanungo et al. 2002]

The authors of [Kanungo et al. 2002] implement their algorithm using a modified kd-tree that subscribes to the following condition: *The cells have bounded aspect ratio and, with every 2d levels of descent in the tree, the sizes of the associated cells decrease by at least a factor of 1=2.*

5. ANALYSIS

A node is considered *visited* if and only if the candidate filtering process is invoked on it. The upper bounds on the computation time of the algorithm are based on the following:

(1) The time needed to process each node is proportional to the number of candidates for the node
    — This is at most k
        — but is typically much smaller

(2) Therefore, the total running time of the algorithm is larger by an upper bound factor of k
(3) The total run time includes the time needed to build the modified initial kd-tree.
— This upper bound is $O(dn*\log(n))$, where d represents the number of dimensions

The authors of [Kanungo et al. 2002] are able to perform k-means clustering using an implementation of Lloyd's algorithm within $O(dn * \log(n))$ time using a kd-tree that has $O(n)$ nodes and $O(log(n))$ depth.

6. CONCLUSION

True k-means clustering is a computationally expensive procedure. Many approximations such as Lloyd's algorithm are much more computationally attractive approximations that converge to a local optimum. The authors of [Kanungo et al. 2002] design and describe an implementation of Lloyd's algorithm that stores data points in a kd-tree. Initial centroid points are chosen at random. At each iteration, the nearest centroid to each data point must be computed. After that, a new centroid is computed based on the associated neighboring node set. In this implementation, a subset of candidate centers Z is maintained for each node in the kd-tree. The candidate centroids z for each node are filtered as they are propagated to the node's children. The real advantage of this approach lies in the fact that the kd-tree is computed for the data points rather than for the centers. Therefore, the kd-tree structure need not be updated at each iteration. This allows for an implementation of Lloyd's algorithm within $O(dn*\log(n))$ time using a kd-tree that has $O(n)$ nodes and $O(log(n))$ depth, where d represents the number of dimensions present in the data space.

REFERENCES

Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. 2009. NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning* 75, 2 (2009), 245–248.

Khaled Alsabti, Sanjay Ranka, and Vineet Singh. 1997. An efficient k-means clustering algorithm. (1997).

Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. 2000. *Computational geometry*. Springer.

Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine Piatko, Ruth Silverman, and Angela Y Wu. 1999. Computing nearest neighbors for moving points and applications to clustering. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 931–932.

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. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 24, 7 (2002), 881–892.

DE Knuth. 1973. The Art of Programming Vol. 3: Sorting and searching. *Addison-Wesley series in computer science* (1973).

James MacQueen and others. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Vol. 1. California, USA, 14.

Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. 2009. The planar k-means problem is NP-hard. In *WALCOM: Algorithms and Computation*. Springer, 274–285.

Andrew Moore. 1999. Very Fast EM-based Mixture Model Clustering Using Multiresolution KD-trees. In *Advances in Neural Information Processing Systems*, M. Kearns and D. Cohn (Eds.). Morgan Kaufman, 340 Pine Street, 6th Fl., San Francisco, CA 94104, 543–549.

Albert Nigrin. 1993. *Neural networks for pattern recognition*. MIT Press.

ACKNOWLEDGMENTS

The author would like to thank Shantenu Jha of Rutgers University for the great semester and extended submission deadline.