Computing Nearest Neighbors for Moving Points and Applications to Clustering

Tapas Kanungo* David M. Mount† Nathan S. Netanyahu‡ Christine Piatko§
Ruth Silverman¶ Angela Y. Wu‖

1 Introduction

Clustering is an important problem, with applications in areas such as data mining and knowledge discovery [6], data compression and vector quantiation [8], and pattern recognition and pattern classification [5]. One important class of clustering problems is given a set of n data points in real d-dimensional space, $\mathbf{R}^d$, and an integer k. The problem is to determine a set of k points $\mathbf{R}^d$, called *centers*, that minimizes the mean squared distance from each data point to its nearest center. This problem is closely related to other clustering problems, such as the *k-medians* problem [2], in which the objective is to minimize the sum of distances, and the *k-center* problem [1], in which the objective is to minimize the maximum distance. The results of Arora, Raghavan and Rao [2] can be extended to this problem, implying the existence of a polynomial time approximation scheme, but this algorithm is quite complex. Also see Hochbaum [9] and Jain and Dubes [11] for a more general description of clustering problems.

One of the most popular heuristics for computing centers for minimizing the squared-error distortion is called the *k-means* algorithm [12, 7]. It is a simple iterative algorithm, and it may be used either for computing an initial set of centers or for producing a local improvement to a given set of centers. Here is how it works. Given any set of k centers, for each center c_i, let R_i denote the set of data points for which c_i is the nearest neighbor. For the next iteration of the algorithm, replace c_i with the centroid of R_i. These two steps are repeated until some some convergence conditions have been met. (See [14] for discussion of its statistical and convergence properties).

One problem with k-means is that it takes a long time to run. There are two reasons for this. First it is often applied in moderate to high dimensional spaces, and computing nearest neighbors efficiently in such spaces (without resorting to brute-force search) is not a trivial problem. The second reason is that often many iterations are needed until the algorithm's termination conditions are satisfied.

We present an approach for improving the efficiency of the k-means algorithm. Our approach is motivated by an observation made by some researchers using the k-means algorithm [10]. After an initial phase of rapid movement of the center points, the algorithm tends to settle into a long phase where the center points move only very slowly. This suggests that a smart algorithm should attempt to update nearest neighbors incrementally after the centers move, rather than recompute them from scratch each time. We present an algorithm for doing this, and we analyze this algorithm as a function of the distances that the centers move. We have implemented this algorithm, and will present empirical results in a more complete paper.

2 Algorithm Overview

In normal nearest neighbor searching, it is common to preprocess a set of data points so that nearest neighbor queries can be answered for a set of query points. In our context, our center points play the role of the data point in the nearest neighbor search, our data points play the role of the query points in nearest neighbor searching. Rather than design a dynamic data structure in which to store the moving center points, our approach will be to reverse the standard roles, and instead to build a data structure for the larger static set of data points, and to move the smaller set of center points through this structure.

*Center for Automation Research, University of Maryland College Park, Maryland.
†Department of Computer Science, University of Maryland, College Park, Maryland.
‡Center for Automation Research, University of Maryland, College Park, and CESDIS, NASA Goddard Space Flight Center.
§The Johns Hopkins University Applied Physics Laboratory, Laurel, Maryland.
¶Department of Computer Science, University of the District of Columbia, Washington, DC, and Center for Automation Research, University of Maryland, College Park, Maryland.
‖Department of Computer Science and Information Systems, The American University, Washington, DC.

The data structure in which the n data points are stored is a *balanced box-decomposition tree* (or BBD-tree) [3]. Each node of the tree is naturally associated with a rectangular region of space, called a *cell*. The tree and the associated decomposition of space can be constructed in $O(n \log n)$ time. As part of this preprocessing, for each node in the tree, we compute the centroid of the points in the cell, weighted by the number of points in the cell.

To compute the nearest neighbors efficiently, we apply the following simple recursive algorithm. For each cell of tree we maintain a set of *candidate centers*. These are the center points that might serve as the nearest neighbor for some point lying within the cell. These candidates can be computed as follows. For each cell, we compute the candidate c closest to its midpoint. Then for each other candidate, if the cell lies entirely on c's side of the bisecting hyperplane between c and this candidate, we remove this candidate. If a node is associated with a single candidate, then this center is the nearest neighbor of all the associated data points. The weighted node centroid is assigned to this center. Otherwise we recurse on its children. If a leaf node is reached, then we compute the distances to all its candidates, and assign the point to its nearest center.

To update nearest neighbors after the centers have moved with each iteration, we model this as a *kinetic process*, as if the centers move continuously in time (see, e.g. [4]). Following their general approach, we maitain a set of *certificates*, whose validity at any time implies the correctness of the current structure. As centers move, certificates may be violated. Since the centers do not necessarily move predictably (e.g. along algebraic curves), we cannot predict when a certificate will be violated. So we compute an upper bound the movement of each point before any of its certificates is violated. As each of these events is *triggered*, we test the validity of the certificate, and if violated we make appropriate updates to the data structure.

3 Data Sensitive Analysis

The amount of computation performed in each iteration of the above k-means algorithm will depend on the distances by which the centers move. Traditional worst-case analysis is inappropriate here, since we are interested in the case where the movement is very small. Instead, our analysis is based in part on a parameter r, which is defined to be the maximum distance that any center moves. We show that number of cells in the BBD-tree that are visited by the algorithm, is proportional to an intrinsic geometric quantity, which intuitively will be small for typical applications.

This quantity is defined as follows. For each center c_i, define $V_i(r)$ to be the set of points in $p \in \mathbf{R}^d$ such that $\mathrm{dist}(p, c_i) + r \leq \mathrm{dist}(p, c_j) - r$ for all $j \neq i$. Notice that $V_i(0)$ is just the Voronoi cell of c_i, and $V_i(r)$ is a subset of this cell. Let $V(r) = R^d \setminus \cup_i V_i(r)$, be the complement of all these sets. The main observation is that only those nodes of the BBD-tree whose cells intersect $V(r)$ need to be visited. Following the same sort of analysis used in [13], it can be shown that the number of nodes visited in the BBD-tree is proportional to the *simple-cover complexity* of $V(r)$.

References

[1] Pankaj K. Agarwal and Cecilia M. Procopiuc. Exact and approximation algorithms for clustering. In *Proc. 9th ACM-SIAM Sympos. Discrete Algorithms*, pages 658–667, 1998.

[2] S. Arora, P. Raghavan, and S. Rao. Approximation schemes for Euclidean k-median and related problems. In *Proc. 30th Annu. ACM Sympos. Theory Comput.*, 1998. 106–113.

[3] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching. In *Proc. 5th Ann. ACM-SIAM Symposium on Discrete Algorithms*, pages 573–582, 1994. (Revised version to appear in *JACM*).

[4] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. In *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, pages 747–756, 1997.

[5] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.

[6] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1996.

[7] E. Forgey. Cluster analysis of multivariate data: Efficiency vs. interpretability of classification. *Biometrics*, 21:768, 1965.

[8] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic, Boston, 1992.

[9] D. S. Hochbaum. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, Boston, 1997.

[10] M. Iwana. Personal communication, 1997.

[11] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[12] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of the Fifth Berkeley Symposium on Math. Stat. and Prob.*, volume 1, pages 281–296, 1967.

[13] Joseph S. B. Mitchell, D. M. Mount, and Subhash Suri. Query-sensitive ray shooting. *Internat. J. Comput. Geom. Appl.*, 7(4):317–347, August 1997.

[14] S. Z. Selim and M. A. Ismail. K-means-type algorithms: a generalized convergence theorem and characterization of local optimality. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:81–87, 1984.