CPS296.2 Geometric Optimization **February 27, 2007**

Lecture 14: k-center Problem

Lecturer: Pankaj K. Agarwal *Scribe: Sharath Kumar*

14.1 Clustering

Given a set P of n points in $\mathbb{R}^d$, a typical clustering problem asks for partitioning P into a family of k subsets $\{P_1, \ldots, P_k\}$, so that a certain objective function is minimized. The objective function depends on the application and varies. A typical clustering problem is the so-called centered clustering problem in which a representative point c_i (or a q-flat) is chosen for each cluster P_i and the cost of the cluster p_i is a function of the distances from c_i to points in P_i. (e.g. max or average distance). Another example is in which the cost of a cluster P_i is a function of pairwise distances in P_i. The overall objective function is $\max_i \mu(P_i)$ or $\sum_i \mu(P_i)$. In general, a clustering objective function can be viewed as

$$\bigoplus \mu(P_i),$$

where $\bigoplus$ is typically the L_p-norm operator.

We now define a few widely used clustering problems.

Definition 1 *k-center: Given a set P of points in $\mathbb{R}^d$ and an integer k, compute a set $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$ of k points in $\mathbb{R}^d$ such that*

$$\max_{p \in P} \min_{\sigma \in \Sigma} ||p - \sigma||$$

is minimized.

Definition 2 *k-median: Given a set P of points in $\mathbb{R}^d$ and an integer k, compute a set $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$ of k points in $\mathbb{R}^d$ such that*

$$\frac{1}{n} \sum_{p \in P} \min_{\sigma \in \Sigma} ||p - \sigma||$$

is minimized.

Definition 3 *k-means: Given a set P of points in $\mathbb{R}^d$ and an integer k, compute a set $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$ of k points in $\mathbb{R}^d$ such that*

$$\left(\frac{1}{n} \sum_{p \in P} \min_{\sigma \in \Sigma} ||p - \sigma||^2 \right)^{1/2}$$

is minimized.

In this lecture, we focus on exact and approximation algorithms for the k-center problem.

14.2 Exact Solutions to k-center

Here is another formulation of the k-center problem. Given a set P of n points in $\mathbb{R}^d$, determine the minimum radius r such that k congruent balls of radius r cover the set P. The center of these balls form the k centers. The decision version of this problem is NP-Hard for $d \geq 2$ if either d or k is part of the input.

14.2.1 2-center in $\mathbb{R}^2$

Let us look at the exact algorithm for the 2-center problem in $\mathbb{R}^2$. The algorithm is based on the observation that, in the optimal solution, the two clusters are linearly separable. Given n points, we generate every possible linearly separable pair of clusters. This can be done by taking a line passing through a pair of points in P as the line separating the clusters. Since there are $O(n^2)$ possible such lines, there will be $O(n^2)$ possible linearly separable clusters. For each of these clusters P_i, we can compute the minimum enclosing ball in $O(n)$ time.

Algorithm 1 2-center(P)

1: **for** each pair $p_i : (a, b)$, $a, b \in P$ and $a \neq b$ **do**
2: Let P_1 and P_2 be clusters separated by line passing through (a, b)
3: Let $r_i = \max$ (MEB(P_1), MEB(P_2)).
4: **end for**
5: **return** $\min_i r_i$.

Thus, in $O(n^3)$ time, we can solve the 2-center problem.

We can generate linearly separable partitions by making lines around each point such that two consecutive pairs of partitions differ by a single point. We can use dynamic data structures with $\mathrm{polylog}$ update time to maintain clusters thereby reducing the running time to $O(n^2 \,\mathrm{polylog}(n))$.

14.2.2 k-center in $\mathbb{R}^2$

Algorithm 1 can be generalized to solve the k-center problem in $\mathbb{R}^2$ in $O(n^{O(k)})$ time. This is because the clusters are formed by the voronoi partitions of the k center points. Since there are $n^{O(k)}$ distinct voronoi partitions all of which can be generated in $O(n^{O(k)})$ time, we can in $O(n^{O(k)})$ time determine the k-centers. The running time of the best known algorithm for computing the optimal k-center in $\mathbb{R}^2$ is $O(n^{\sqrt{k}})$.

14.3 2-approximation for the k-center problem

Now we look at polynomial time approximation algorithm for the k-center problem.

Consider the following greedy algorithm. Select an arbitrary point as the first center. Then, at every step, select the new center as the point farthest away from all existing centers. Repeat this k times. Return these k points as the k-centers. This greedy algorithm has an approximation ratio 2.

Let, Σ: Set of k centers chosen. Then the radius would be

$r = \max_{x \in P}(\min_{\sigma \in \Sigma}(||x\sigma||))$

P: set of n input points.

Algorithm 2 Greedy-k-center(P)

1: Initialize Σ to an arbitrary point $p \in P$
2: **for** $i = 2$ to k **do**
3: $\quad x = \text{argmax}_{x \in P} \min_{\sigma \in \Sigma} ||x - \sigma||$
4: $\quad \Sigma = \Sigma \cup \{x\}$
5: **end for**
6: **return** Σ and Radius $r = \max_{x \in P}(\min_{\sigma \in \Sigma} ||x - \sigma||)$

Running Time of the greedy algorithm is $O(nk)$. This can be improved to $O(n \log n)$.

Lemma 1 *The greedy algorithm has approximation ratio* 2

Proof: Let r^* be the optimal solution for the k-center problem. Thus the maximum distance of a point from its closest center is r^*. For the sake of contradiction, let us assume that the value returned by the greedy algorithm is $r > 2r^*$. Then, in particular, the k centers selected by the greedy algorithm and an additional point (which is the point $\text{argmax}_{x \in P}(\min_{\sigma \in \Sigma} ||x - \sigma||)$) should be at a mutual distance r from each other. Thus, there are $k+1$ points with all the pairwise distances greater than r. Now, in any disc of radius r^*, there cannot be two points at a distance $r > 2r^*$. Hence each of these $k + 1$ points must be in different discs in the optimal solution. This would mean that the algorithm should have chosen atleast $k + 1$ discs of radius r^* contradicting the assumption that r^* is a solution for the k-center problem. ■

The next question is whether there is an algorithm with a better approximation ratio. Theorem 1 addresses this issue.

Theorem 1 *[2] [3] Assuming $P \neq NP$, there is no polynomial time approximation algorithm achieving a factor of $2 - \epsilon$, $\epsilon > 0$, for the metric k-center problem.*

This proof shows that the dominating set problem can be solved in polynomial time if there is a polynomial time algorithm for a $2 - \epsilon$ approximation of the k-center problem. The dominating set problem is known to be NP-Complete and from there the theorem follows. Details can be read in [2].

Theorem 1 implies that there is no efficient algorithm with approximation factor $2-\epsilon$ for the k-center problem.

14.4 $(1 + \epsilon)$-approximation for k-center

Recall the coreset construction for the 1-center problem. We know the following theorem.

Theorem 2 *[1] For any point set P in R^d, and $1 > \epsilon > 0$, there is a subset $S \subset P$, $|S| = O(1/\epsilon)$, such that if c is the 1-center of S, then c is a $(1 + \epsilon)$-approximate 1-center for P.*

14.4.1 Coreset for the 2-center

Now, we describe a method to generate coreset for the 2-center problem. Let P be a set of n points and let P_1 and P_2 be the partition of these n points corresponding to the optimal solution to the 2-center problem. Our algorithm assumes the existance of an oracle which given a point $p \in P$, determines whether p is in P_1 or P_2. Later we show how this oracle can be simulated.

We start with two empty sets S_1 and S_2. For $j = 1, 2$, let c_j be the center of the smallest enclosing ball of S_j. In the ith iteration, we pick the point p_i farthest away from c_1 and c_2. To decide whether to put p_i in S_1 or S_2, we use the oracle which tells us whether p_i is in P_1 or P_2. After this, we recompute c_1 and c_2. After $O(1/\epsilon)$ iterations, we obtain a coreset for the 2-center provided that the oracle answers correctly.

Algorithm 3 $2 - center(P)$

Let P_1 and P_2 be the partitions corresponding to the optimal solution
Let $S_1 \cup S_2$ be the coreset of the 2-center .
for $i = 1$ to $O(1/\epsilon)$ **do**
    Let $B_1(c_1, r_1)$ =MEB(S_1)
    Let $B_2(c_2, r_2)$ =MEB(S_2)
    $p =$ argmax$_{p \in P}$(min($||p - c_1||$, $||p - c_2||$)
    if $p \in P_1$ **then**
        Add p to S_1
    else
        Add p to S_2
    end if
end for
return c_1, c_2 and radius $(1 + \epsilon)$ max(r_1, r_2)

To remove the oracle, we exhaustively enumerate all possible guesses. This would require running the above algorithm $2^{O(1/\epsilon)}$ times. Overall the algorithm would thus have a running time $O(dn2^{O(1/\epsilon)})$.

Theorem 3 *[1] For any point set $P \subset R^d$ and $1 > \epsilon > 0$, a $(1 + \epsilon)$- approximate 2-center for P can be found in $O(2^{O(1/\epsilon)}dn)$ time.*

14.4.2 Coreset for the k-center

Algorithm 3 can be generalized to obtain a coreset for the k-center.

Theorem 4 *[1] For any point set $P \subset R^d$ and $1 > \epsilon > 0$, a $(1 + \epsilon)$- approximate k-center for P can be found in $O(k^{O(k/\epsilon)}dn)$ time.*

Proof: A straight forward extension of the algorithm 3 is where each guess of the oracle now is a number between 1 and k and the number of iterations run by the algorithm is $O(k/\epsilon)$. The simulation of the oracle is equivalent to the running the Algorithm for all possible guesses. The total number of guesses is $k^{O(k/\epsilon)}$ Hence, the running time of the algorithm is $O(k^{O(k/\epsilon)}dn)$. ■

14.5 An application of clustering

We conclude this lecture by giving a very different application of clustering. A widely used data structure for answering range queries (given a rectangle σ, report all points in $P \cap \sigma$) is the so-called R-Tree [4]. It typically constructs a heirarchical clustering on P - given a parameter k partition P into k-clusters, and then recursively partition each cluster. However, the measure of the quality of cluster is very different - find a clustering so that the boundary of a rectangle intersects the bounding boxes of as few clusters as possible. Several heuristics have been developed for this clustering problem.

References

[1] M. Badoiu, S. Har-Peled and P. Indyk. Approximate Clustering via Core-Sets *Proceedings, Symposium on Theory of Computing*, 2002, 250–257. 14-4

[2] W. L. Hsu and G. L. Nemhauser. Easy and hard bottleneck location problems. *Discrete Applied Mathematics, 1*, 1979, 209–216. 14-3

[3] T. Feder and D. H. Greene. Optimal Algorithms for Approximate Clustering. *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, 1988, 434–444. 14-3

[4] H. Samet. *Fundamentals of Multidimensional and Metric data structure*, Morgan Kaufmann Publishers,San Francisco, 2006. 14-5