Vector Search **Class #XXXXX Fall 2023**

Lecture number XX: kd-trees

Lectures: Edo Liberty

Warning: *Please do not cite this note as a peer reviewed source. If you find mistakes, please inform the authors.*

Nearest neighbor search is a fundamental computational building block in computer vision, graphics, data mining, machine learning, and many other sub-fields. As an example, consider a simple k-nearest-neighbor-classifier which, for each point predicts its class by the a majority vote over its neighbors' classes. As simplistic as this classifier sounds, it actually performs very well in many scenarios.

We could easily compute this by scanning the data points and finding those which minimize $\|q - x_i\|$. But, of course, it would be significantly better if we could reduce the dependence on n, or in other words, avoid scanning the data for every query point. One possible definition of the nearest neighbor problem is as follows:

Definition 0.1. Nearest Neighbor Search: *Given a set of points $\{x_1, \ldots, x_n\} \in \mathbb{R}^d$ preprocess them into a data structure X of size $\mathrm{poly}(n, d)$ in time $\mathrm{poly}(n, d)$ such that nearest neighbor queries can be performed in logarithmic time. In other words, given a search query point q a radius r and X one can return all x_i such the $||q - x_i|| \leq r$. The search for x_i should require at most $\mathrm{poly}(d, \log(n))$ time.*

1 kd-trees

First, we shall review a well known an widely used algorithm for this problem which is called kd-trees [1]. The data structure holds the points in a tree. Each subtree contains all the points in an axis aligned bounding box (maybe infinite). In each depth in the tree the bounding boxes are split along an axis (in a round robin order) at the median value of the points in the box. Splitting stops when the number of points in the box is sufficiently small, say one.

It is quite simple to prove that inserting and deleting points requires $O(log(n))$ time. Thus, the entire construction time is bounded by $O(n \log(n))$.

Searching however is quite a different matter. Let us assume w.o.l.g. that all points are inside the unit cube. A harmless assumption because we can scale the entire data set by a constant factor. Also, notice that each point in the date whose bounding box intersects the query sphere must be examined. Moreover, examining each data point can generate at most $O(log(n))$ computations. We will therefore only ask ourselves, how many point boxes have a non zero intersection with the query sphere.

To make this exercise simpler, consider a simpler case and a variation of kd-trees. First, all n data points are distributed uniformly and i.i.d. over $[0, 1]^d$. Also, the space partition splits every box exactly in the middle when the cut is made along the axes in a round robin order. Note that for such random data, our simple algorithm produces a very simple partition to the one kd-tree would have generated since the median point is approximately also in the middle of the box. (this holds as long as the number of points in a box is large enough)

Let's assume each box is split t times (the depth of the tree is t). That means that each axis was split in half t/d (assuming t/d is an integer) times. Therefore, the dimension of our box are $[2^{-t/d}, 2^{-t/d}, \ldots, 2^{-t/d}]$. What is the probability that a (randomly located) ball of radius r will hit this box?

This probability that the box is "hit" by a random query sphere is at most the volume of a sphere of radius $\sqrt{d}2^{-t/d-1} + r$ since if the query falls outside this ball it will not intersect the box. Assume, for now, that $\sqrt{d}2^{-t/d-1} \leq r$ and so the probability of this event is bounded by the volume of a ball of radius $2r$. The volume of such a ball is $\pi^{d/2}(2r)^d/(d/2)!$ (assuming, for convenience, that d is even). The dependance

on the radius is therefore $O(r^d)$. Thus, we can expect to inspect only a $O(r^d)$ fraction of the boxes (and thus points). Since $r \leq 1$ this can be a significant speedup.

We saw that separating the space into boxes enabled us to search in it more efficiently. We, however, made some heavy assumptions. One such is that $\sqrt{d}2^{-t/d-1} \leq r$. This assumption fails when the dimension grows. To see this, consider that $t \approx \log(n)$ to give boxes of volume roughly $1/n$. Requiring that $\sqrt{d}2^{-\log(n)/d} \leq \sqrt{d}/2$ we get that $\log(n) \geq d$ or that n must be exponential in d, $n \geq 2^d$. Therefore, we only gain if the number of points in exponential in the dimension. This is a result of the so called "curse of dimensionality" which is discussed next.

References

[1] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18:509–517, September 1975.