Exhaustive kNN Algorithm

exhaustive k-NN algo: given query pt. q,

1) Scan through all n sample pts and compute squared distances to q
2) maintain a max-heap w/ k-shortest distances so far
   - remove top pt. and insert new pt. when we discover a sample pt. closer to q than top of heap

* heap not necessary but more efficient for large k values

- O training time for algorithm } only O(0) algo this semester
  - all computation done during query

- O(nd+nlogk) query time
  - O(nd+klognlogk) expected query time for random pt. order
    - ↳ Slightly better runtime → only O(klogn) expected heap operations occur
      - not recommended in practice since you'd lose more from cache misses than you'll gain from fewer heap operations

Can an algorithm preprocess training pts. to get sublinear query time?
- Voronoi diagrams for 2-5 dimensions
- k-d trees better for d → 30
- exhaustive k-nn best for d > 30
  - PCA or random projection also work
  - locality-sensitive hashing works but not widely used

Cont. below

Voronoi Diagrams

always convex polyhedron or polytope

Let X be a pt. set

Then the Voronoi cell of $\vec{u} \in X$ is $Vor(\vec{u}) = \{\vec{p} \in \mathbb{R}^d \mid \forall \vec{v} \in X, \|\vec{p} - \vec{u}\|_2 \leq \|\vec{p} - \vec{v}\|_2\}$

all pts. in vector space whose closest pt. is $\vec{u}$

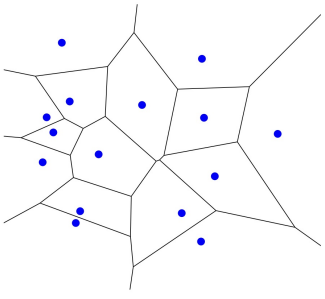

Figure 25.1: Example Voronoi diagram for 15 points

- Size of Voronoi diagram (# vertices) is $O(n^{\lceil d/2 \rceil})$
  - d = dimension of pts.
  - tight upper bound when d is a small constant
  - as d grows, tightest asymptotic bound is smaller than $O(n^{\lceil d/2 \rceil})$, but complexity still grows exponentially w/ d

We can use Voronoi diagrams for querying:
- given query pt. $\vec{q} \in \mathbb{R}^d$,
  - find $\vec{u} \in X$ for which $\vec{q} \in Vor(\vec{u})$
    - new data structure needed for this to be efficient
- In 2D, takes $O(n \log n)$ time to get the Voronoi diagram & trapezoidal map for pt. location
  - query then takes $O(\log n)$ after map computed
    - much better than linear query.
- In d dimensions, use a binary space partition tree (BSP tree) for pt. location
  - reasonably fast in 3-5 dimensions

* Voronoi diagram only supports 1-nearest neighbor queries
- for k-nearest neighbors, use order-k Voronoi diagram
  - cell for each possible k-nearest neighbors of $\vec{q}$
    - NOBODY USES THESE
      - ↳ $O(k^2 n)$ for order-k diagram in 2D, worse in higher dimensions
      - ↳ no software available to compute this
- V-diagrams...
  - exist for other distance metrics
  - good for 1-NN in 2-3D or 4-5D but k-d trees much simpler and faster for $d > 6$

k-d trees: decision trees for nearest neighbor search

Differences when compared to decision trees:

- choose splitting feature w/ largest width, i.e. feature i in $\max_{i,j,k}(X_{ji} - X_{ki})$
  - nearest neighbor search doesn't care about entropy
    - want sphere around query pt. to not intersect many boxes of decision tree instead
      → helps if boxes nearly cubical rather than long & thin
  - cheap alternative: rotate through features:
    1) Split on 1st feature at depth 1
    2) Split 2nd at depth 2, etc
    - faster by $O(d)$
- Splitting value is median pt. for feature i, or $\frac{X_{ji} + X_{ki}}{2}$
  - median guarantees $\lfloor \log_2 n \rfloor$ tree depth & $O(nd \log n)$ tree building time
    - $O(n \log n)$ time if rotating through features
  - alternative to median: split at box's center
    - improves aspect ratio of boxes
    - might make tree unbalanced
  - or, median at odd depth & box center at even depth
    - $O(\log n)$ depth
- each internal node of k-d tree stores a sample pt. that lies in node's box
  - this pt. is usually the splitting pt.
  - some implementations have pts. only at leaves
    - preferred w/ internal pts → when searching through tree, we often stop searching sooner.

Algo: given query $\vec{q}$, find pt. $\vec{v}$ s.t. $\|\vec{q} - \vec{v}\| \leq (1+\varepsilon)\|\vec{q} - \vec{s}\|$, $\vec{s}$ is closest sample pt.

- $\varepsilon = 0 \Rightarrow$ exact nearest neighbors
  $\varepsilon > 0 \Rightarrow$ approx. nearest neighbors

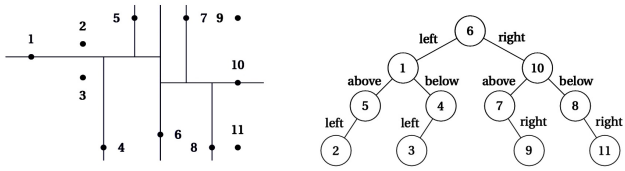

Figure 25.2: Example of a k-d tree. To step through the animation, you must use a PDF reader that supports animations, eg. Adobe Acrobat or Okular.

Query algo. maintains 2 things:

1) nearest neighbor found so far (or k nearest)
   - successive improvements decrease w/ distance
2) maintains binary min-heap of unexplored subtrees keyed by dist. from $\vec{q}$
   - successive pops increase min. distance in heap

each subtree represents an axis-aligned box

- query searches for boxes closest to $\vec{q}$ first
  → avoid searching unnecessary boxes
- measure distance from $\vec{q}$ to a box
  - distance is key for subtree in heap
- search stops when distance from $\vec{q}$ to kth nearest neighbor found so far is less than distance from $\vec{q}$ to nearest unexplored box
  (÷ times $1+\varepsilon$ if approx. NN)

1-NN query algo:

```
1  Q = heap containing root node with key zero
2  r = ∞
3  while Q is not empty and (1 + ε) * min(Q) < r
4      // pop min and update nearest neighbor
5      B = Q.removemin()
6      w = sample point at B
7      r = min(r, dist(q, w))
8
9      // add child boxes to heap
10     B1, B2 = child boxes of B
11     if (1 + ε) * dist(q, B1) < r then
12         Q.insert(B1, dist(q, B1)) // key for B1 is dist(q, B1)
13     if (1 + ε) * dist(q, B2) < r then
14         Q.insert(B2, dist(q, B2)) // key for B2 is dist(q, B2)
```

works w/ any Lp for which $p \in [1, \infty]$

- If k-NN, replace r w/ max-heap that holds k nearest neighbors (like exhaustive search algo.)

Summary:

- Start w/ heap initialized w/ root of k-d tree
- pop from min heap & compare distance from $\vec{q}$ to pt. in current node
  - update nearest neighbor if necessary
- look at shortest distance from $\vec{q}$ to children <u>boxes</u> of current node
  - if these distances to boxes are less than current nearest neighbor, then node in box might be new nearest neighbor → add child to heap
- repeat until min. element in heap is too far (over ($\vec{q}$ → nearest neighbor < $\vec{q}$ → closest unvisited box)

Why ε-approx. NN?

↳ don't need to visit each box