Scaling up k-NN search using KD-trees

Complexity of brute force search

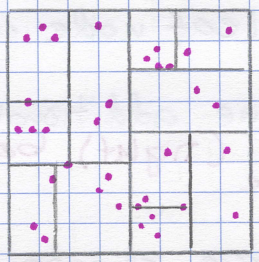
Given a query point, scan through each point
- $O(N)$ distance computations per 1-NN query!
- $O(N \log k)$ per k-NN query!

What if N is huge??? (and many queries)

KD-trees representation

KD-trees

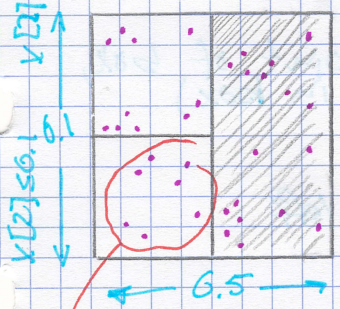
Structured organization of documents
- Recursively partitions points into axis aligned boxes



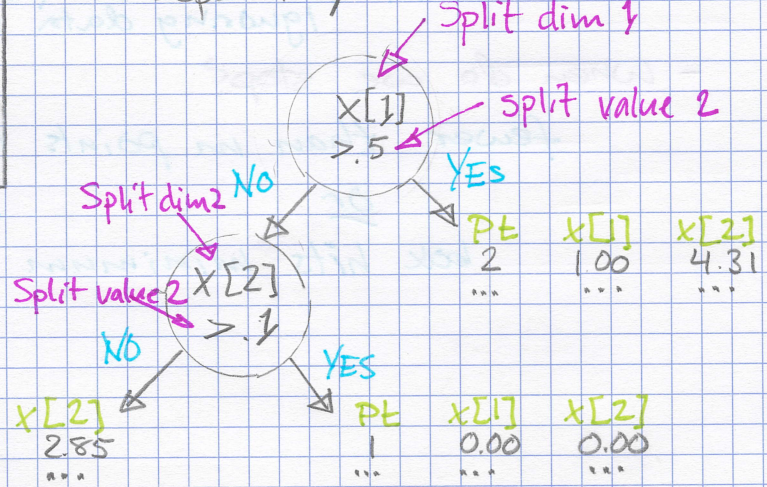
Enables more efficient pruning of search space

Works "well" in "low-medium" dimensions
- We'll get back to this...

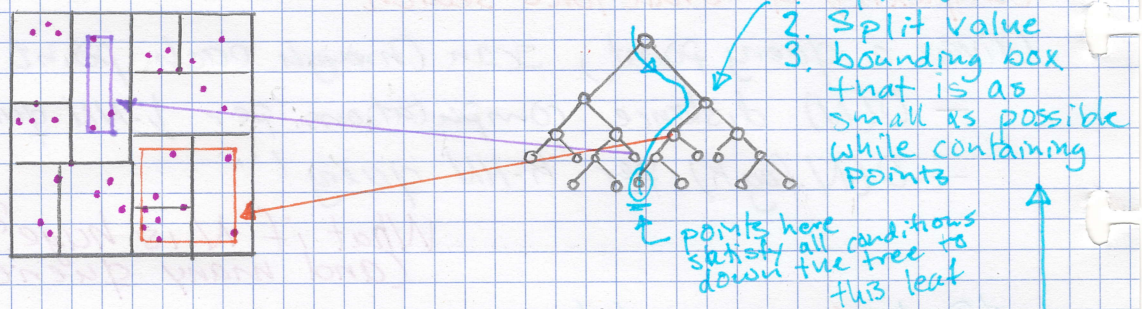
KD-tree construction



Recurse on each group separately



KD tree construction



Continue splitting points at each set
- Creates a binary tree structure

Each leaf node contains a list of points

Keep one additional piece of info at each node:
- The (tight) bounds of points at or below node

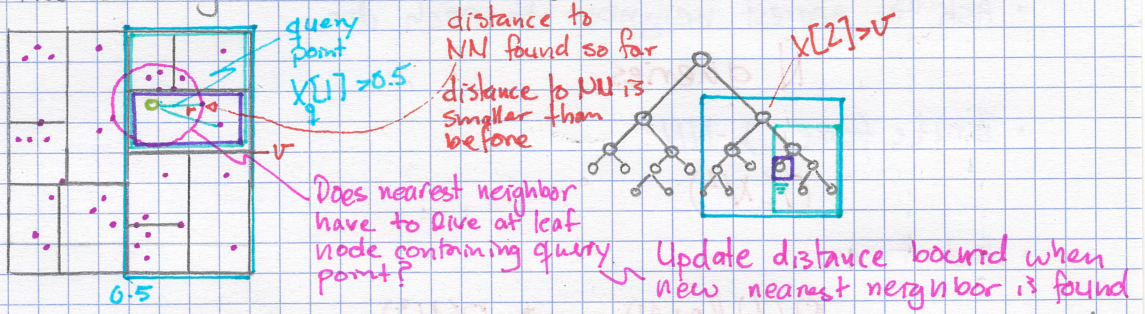
#3

KD-tree construction choices

Use heuristics to make splitting decisions:
- Which dimension do we split along?
  - widest (or alternate)
- Which value do we split at?
  - median (or center point of box, ignoring data in box)
- When do we stop?
  - fewer than m points left
  - or
  - box hits minimum width

NN search with KD-trees

Nearest neighbor with KD-trees



Traverse tree looking for nearest neighbor to query point

1. Start by exploring leaf node containing query point
2. Compute distance to each other point at leaf node
3. Backtrack and try other branch at each node visited

Use distance bound and bounding box of each node to prune parts of tree that cannot include nearest neighbor

Complexity of NN search with KD-trees

Complexity

For (nearly) balanced, binary trees...

- Construction
  - Size: $2N-1$ nodes if 1 data point at each leaf $\rightarrow O(N)$
  - Depth: $O(\log N)$
  - Median + send points left right: $O(N)$ at every level of the tree
  - Construction time: $O(N \log N)$

- 1-NN query
  - Traverse down tree to starting point: $O(\log N)$
  - Maximum backtrack and traverse: $O(N)$ in worst case
  - Complexity range: $O(\log N) \rightarrow O(N)$

Under some assumptions on distribution of points, we get $O(\log N)$ but <u>exponential in d</u>

Complexity for N queries

- Ask for nearest neighbor to each doc

N queries

- Brute force 1-NN:

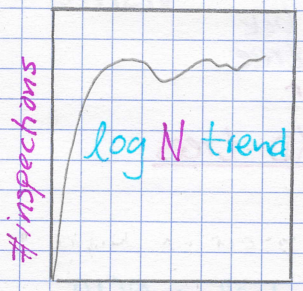
$O(N^2)$

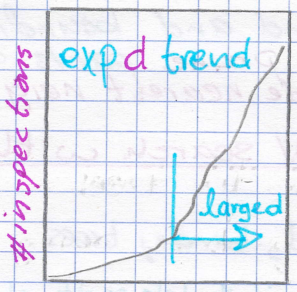
- kd-trees:

$O(N \log N) \longrightarrow O(N^2)$

potentially very large savings for large N!

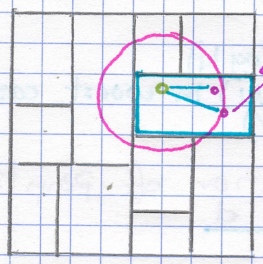
Inspections vs. N and d





$x_i \in \mathbb{R}^d$

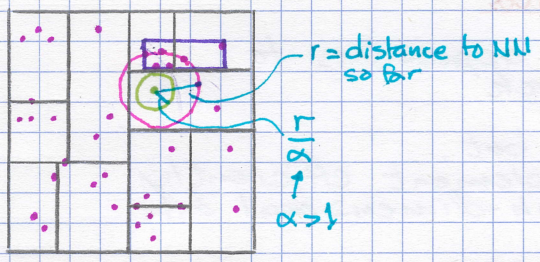
k-NN with KD-trees



distance to 2nd nearest neighbor in 2-NN example

Exactly the same algorithm, but maintain distance to furthest of current K nearest neighbors

Approximate k-NN search using KD-trees



Saves lots of search time at little cost in quality of NN!

Before: Prune when distance to bounding box > r

Now: Prune when distance to bounding box > r/α

Prunes more than allowed, but can guarantee that if we return a neighbor at distance r, then there is no neighbor closer than r/α

↳ Bound loose... In practice, often closer to optimal

Closing remarks on KD-trees

Tons of variants of kd-trees
- On construction of trees (heuristics for splitting, stopping, representing branches...)
- Other representational data structures for fast NN search (e.g., ball trees)

Nearest neighbor search
- Distance metric and data representation crucial to answer returned

For both, high-dim spaces are hard!
- Number of kd-tree search can be exponential in dimension
  - Rule of thumb... $N >> 2^d$ ... Typically useless for large d.
- Distances sensitive to irrelevant features
  - Most dimensions are just noise → everything is far away
  - Need technique to learn which features are important to given task