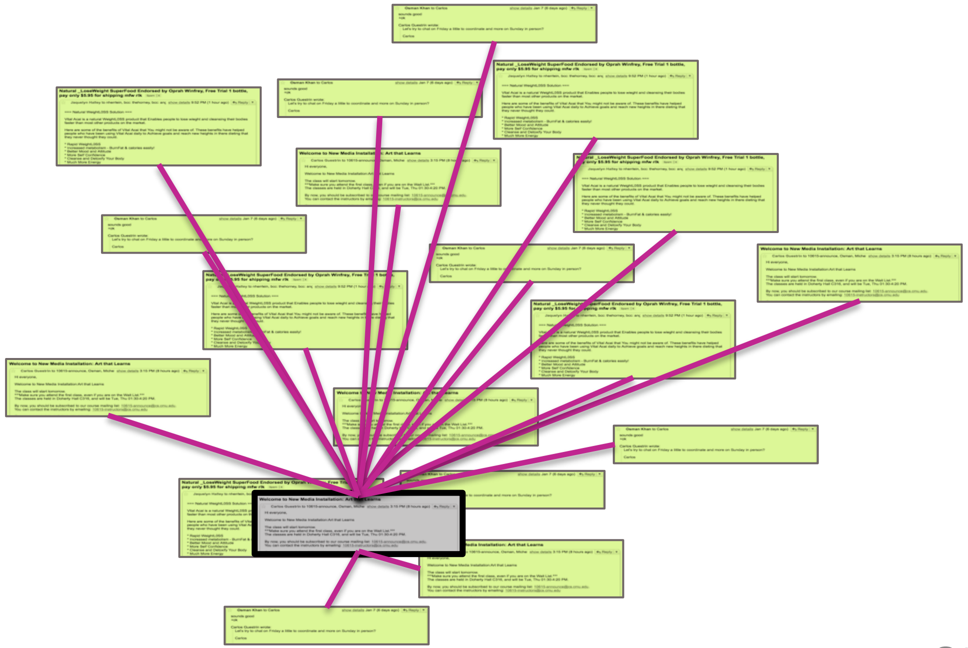
Scaling up k-NN search by storing data in a KD-tree

Complexity of search

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!



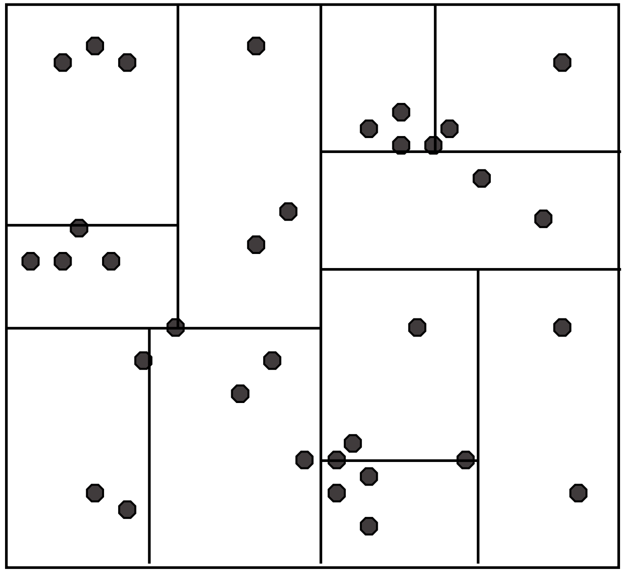
What if N is huge???
(and many queries)

KD-tree representation

KD-trees

Structured organization of documents

- Recursively partitions points into axis aligned boxes.

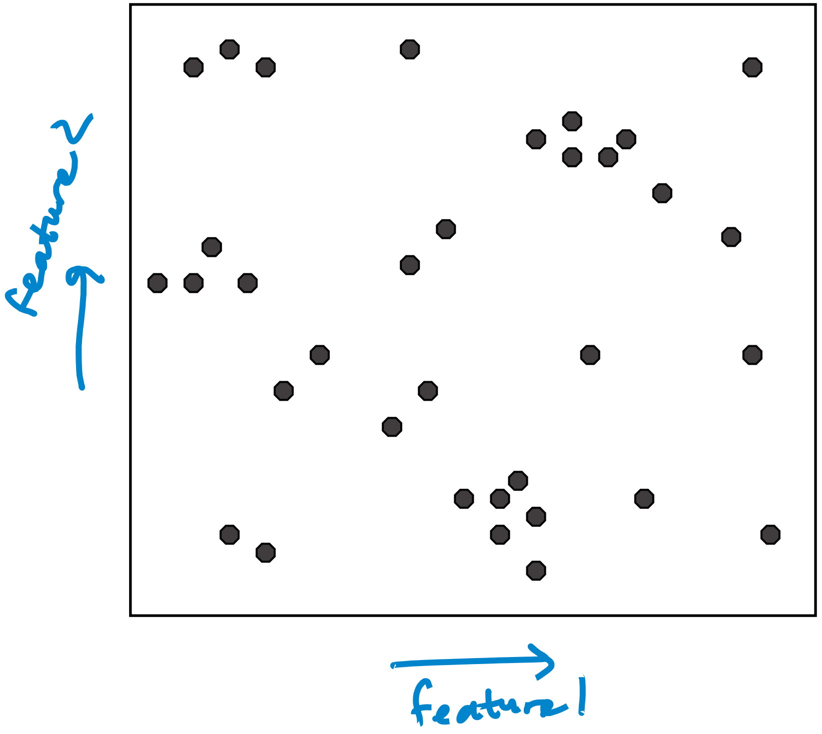


Enables more efficient pruning of search space

Works “well” in “low-medium” dimensions

- We’ll get back to this…

KD-tree construction

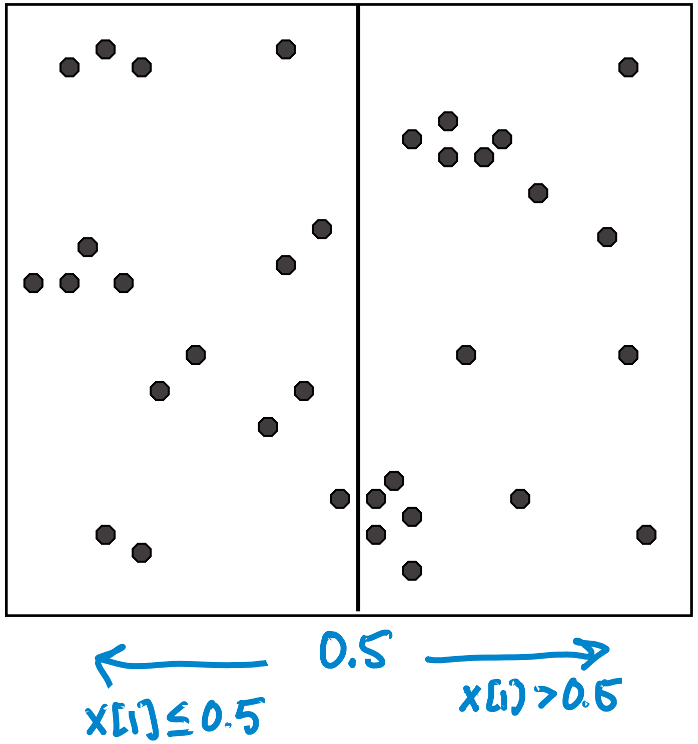


Start with a list of d-dimensional points.

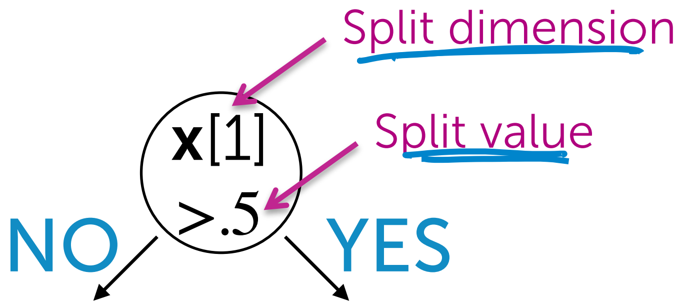
Pt	**x**[1]	**x**[2]
1	0.00	0.00
2	1.00	4.31
3	0.13	2.85
…	…	…

obs. indices　　feat. 1 (word 1)　　feat. 2 (word 2)

KD-tree construction



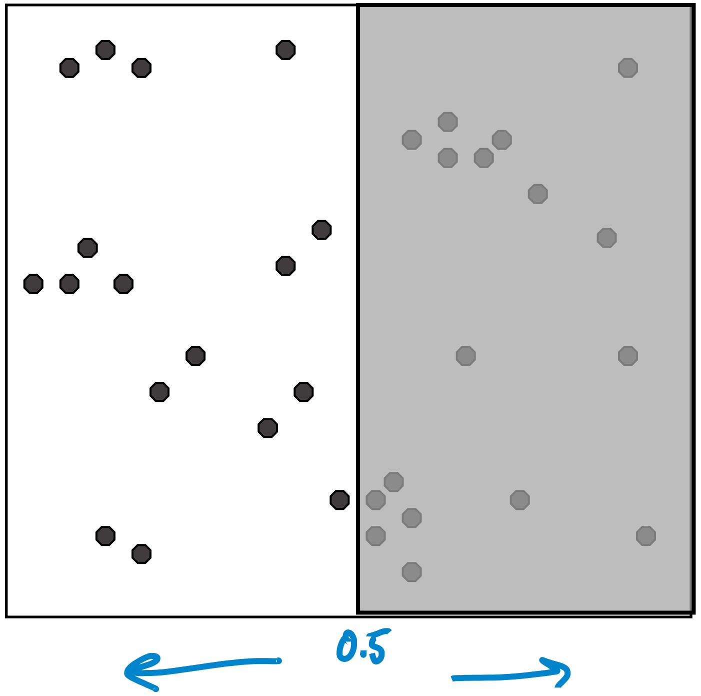
Split points into 2 groups



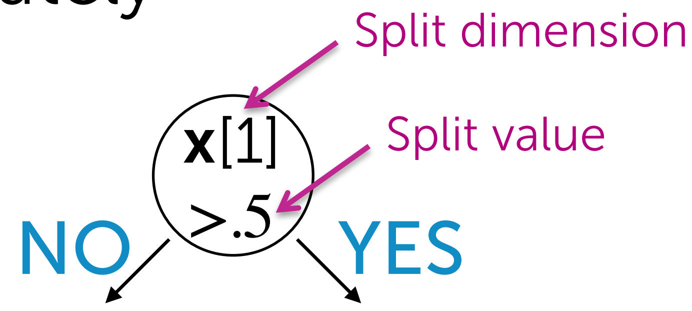
Pt	**x**[1]	**x**[2]
1	0.00	0.00
3	0.13	2.85
…	…	…

Pt	**x**[1]	**x**[2]
2	1.00	4.31
…	…	…

KD-tree construction



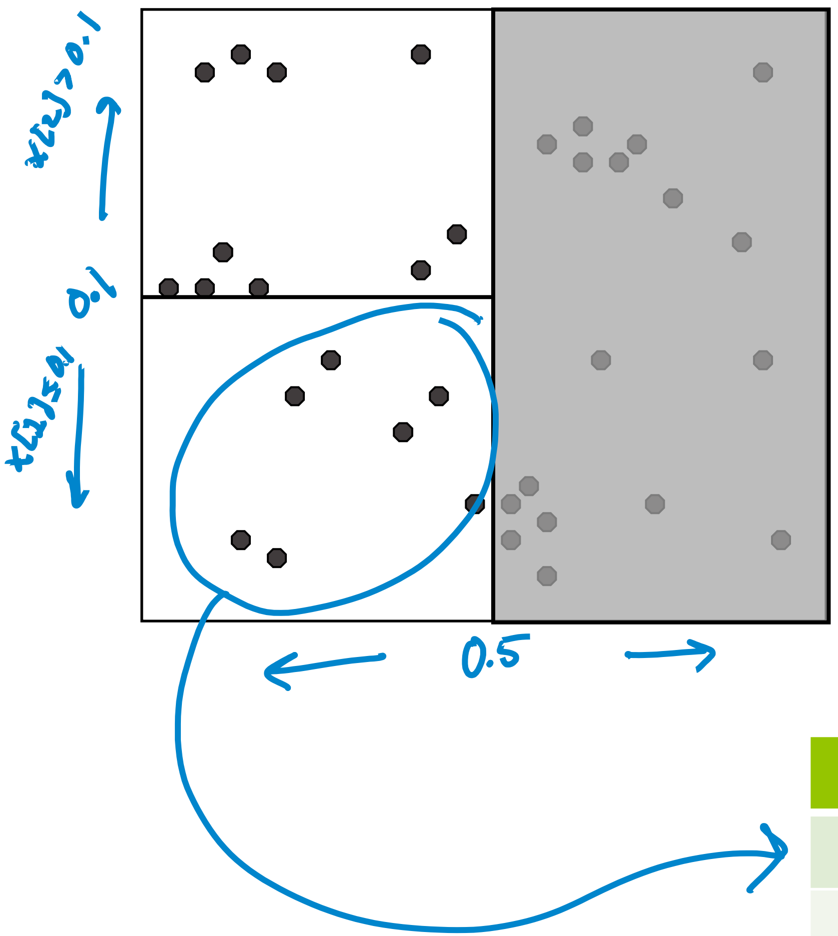
Recurse on each group separately



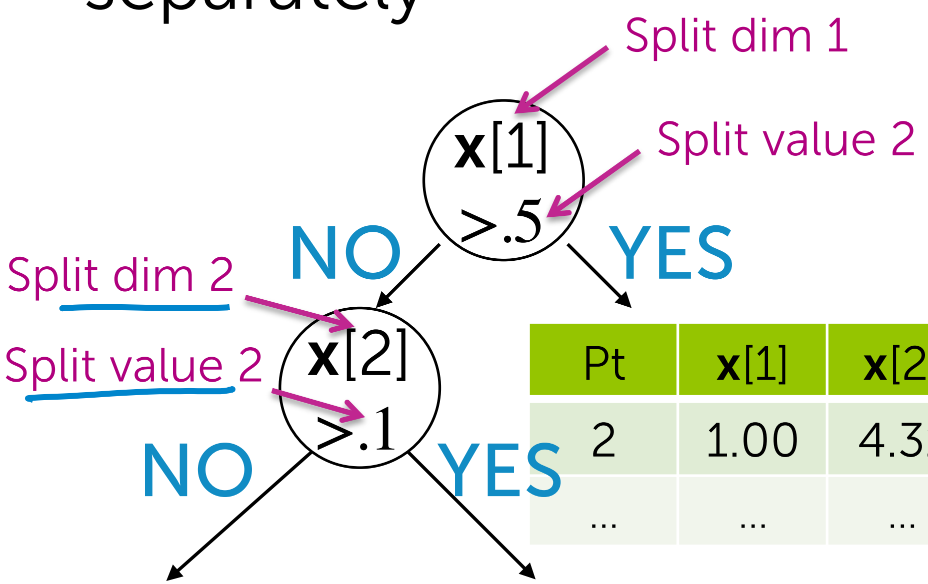
Pt	**x**[1]	**x**[2]
1	0.00	0.00
3	0.13	2.85
…	…	…

Pt	**x**[1]	**x**[2]
2	1.00	4.31
…	…	…

KD-tree construction



Recurse on each group separately

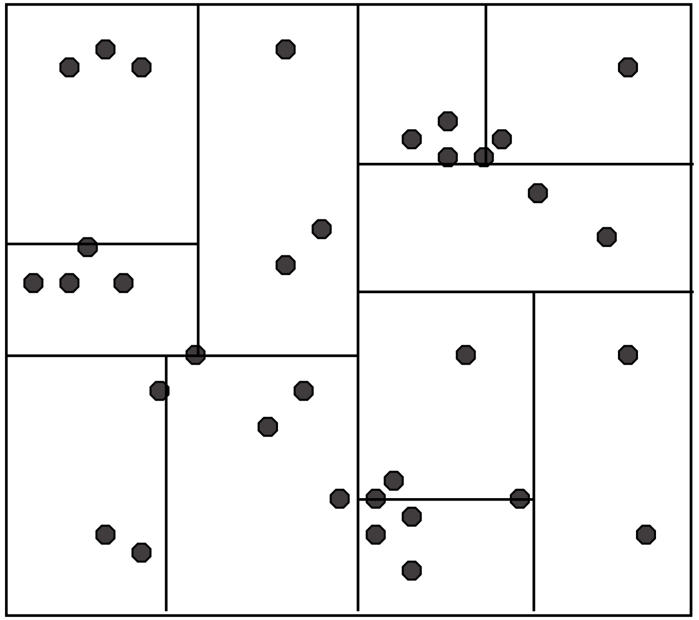


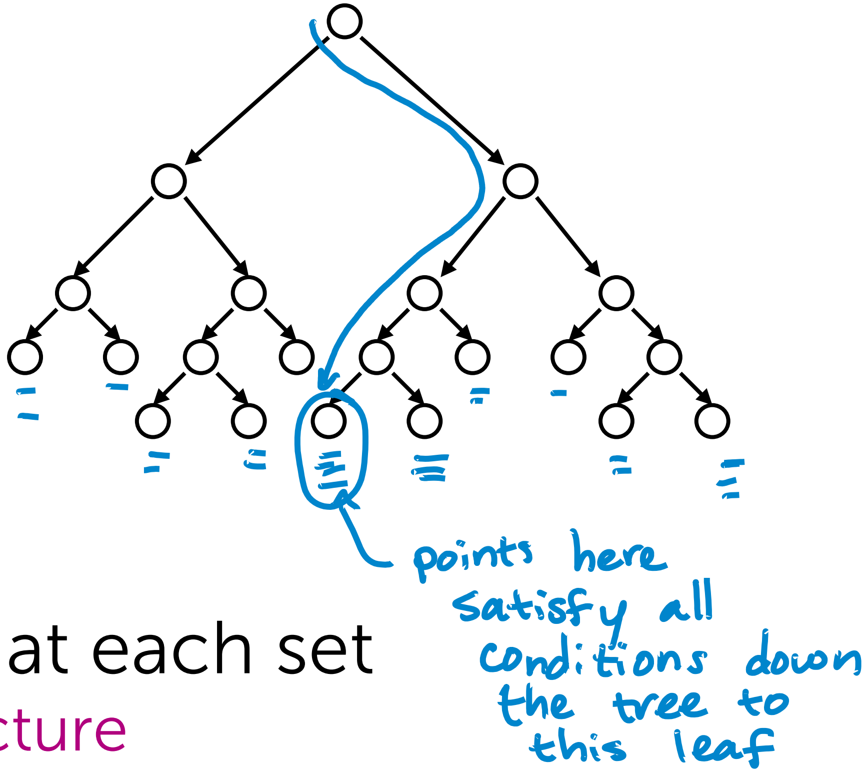
Pt	**x**[1]	**x**[2]
2	1.00	4.31
…	…	…

Pt	**x**[1]	**x**[2]
3	0.13	2.85
…	…	…

Pt	**x**[1]	**x**[2]
1	0.00	0.00
…	…	…

KD-tree construction



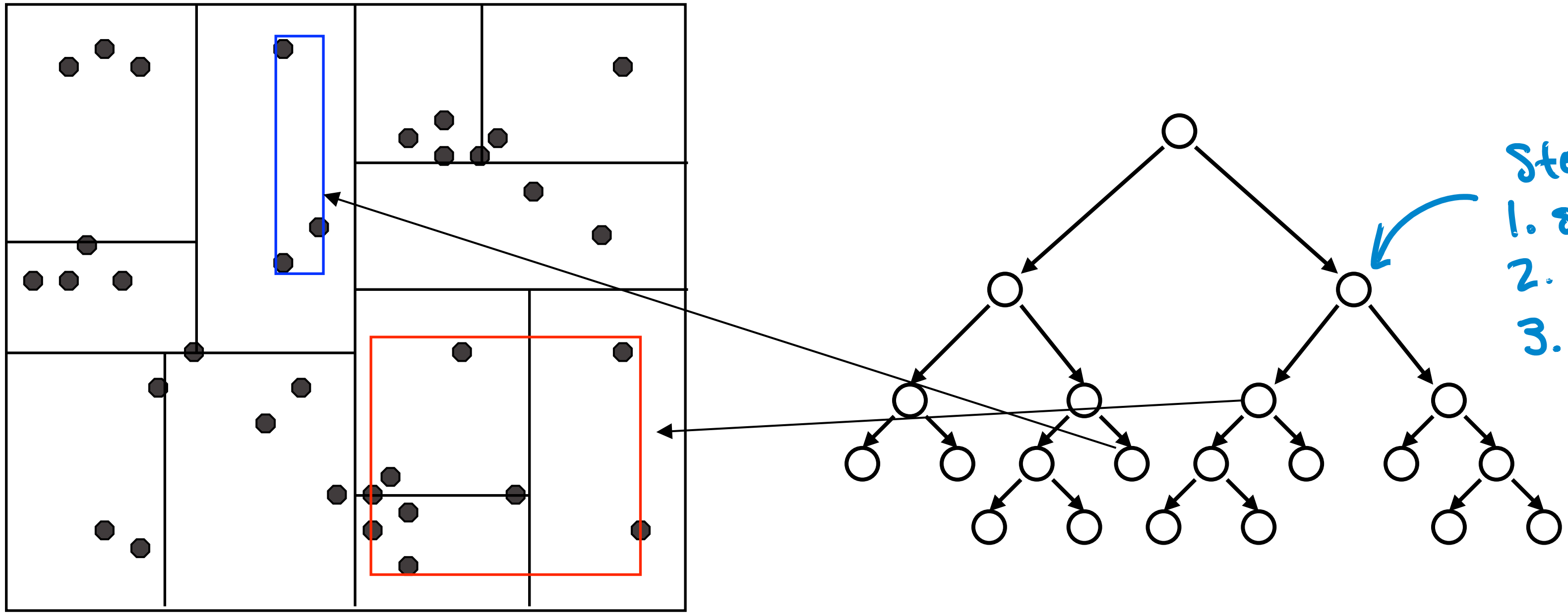


Continue splitting points at each set

- Creates a binary tree structure

Each leaf node contains a list of points

KD-tree construction



Store:
1. split dim?
2. split value
3. bounding box that is as small as possible while containing pts

Keep one additional piece of info at each node:

#3 – The (tight) bounds of points at or below node

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)

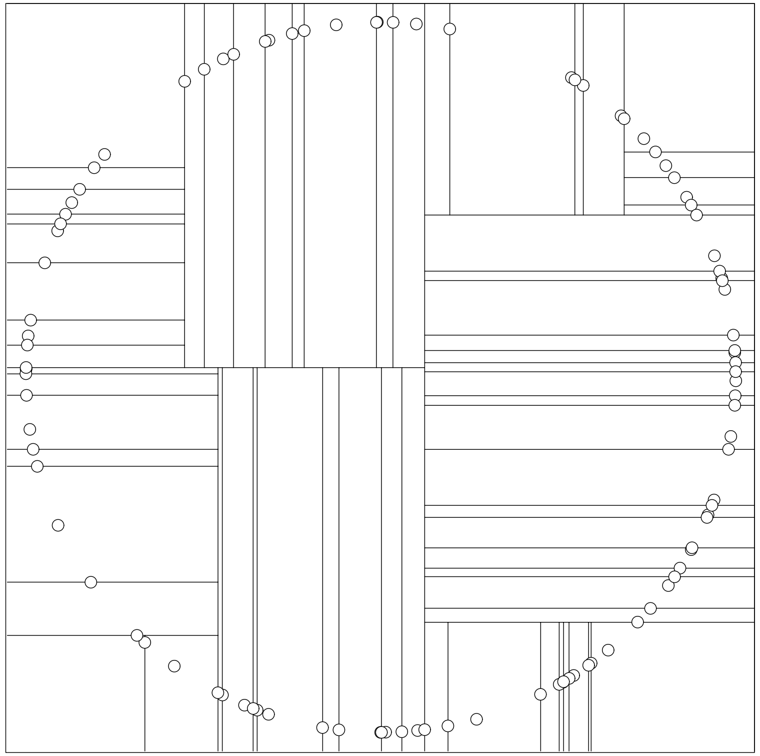
- When do we stop?

  fewer than m pts left

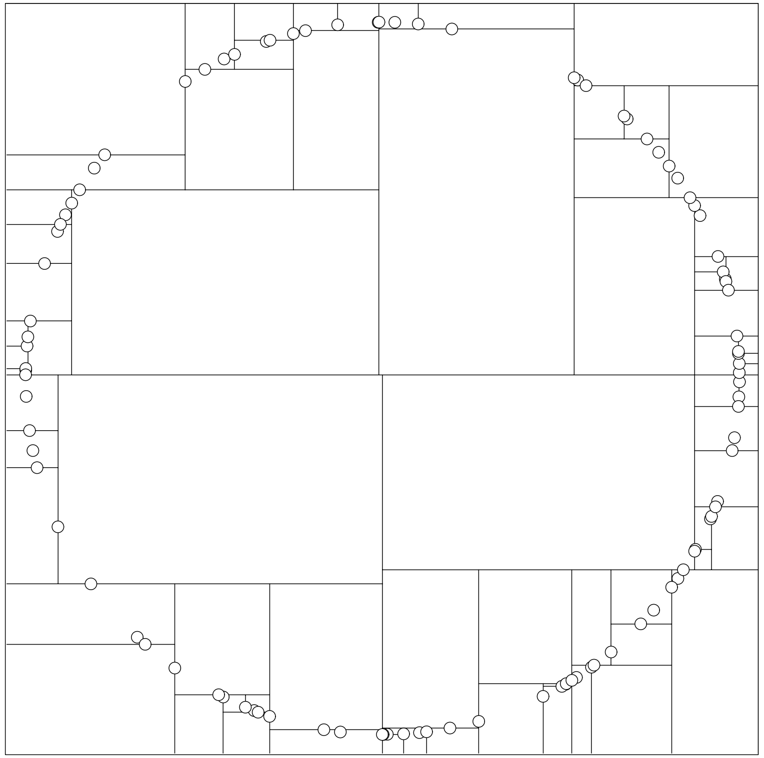
  or

  box hits minimum width

Many heuristics...



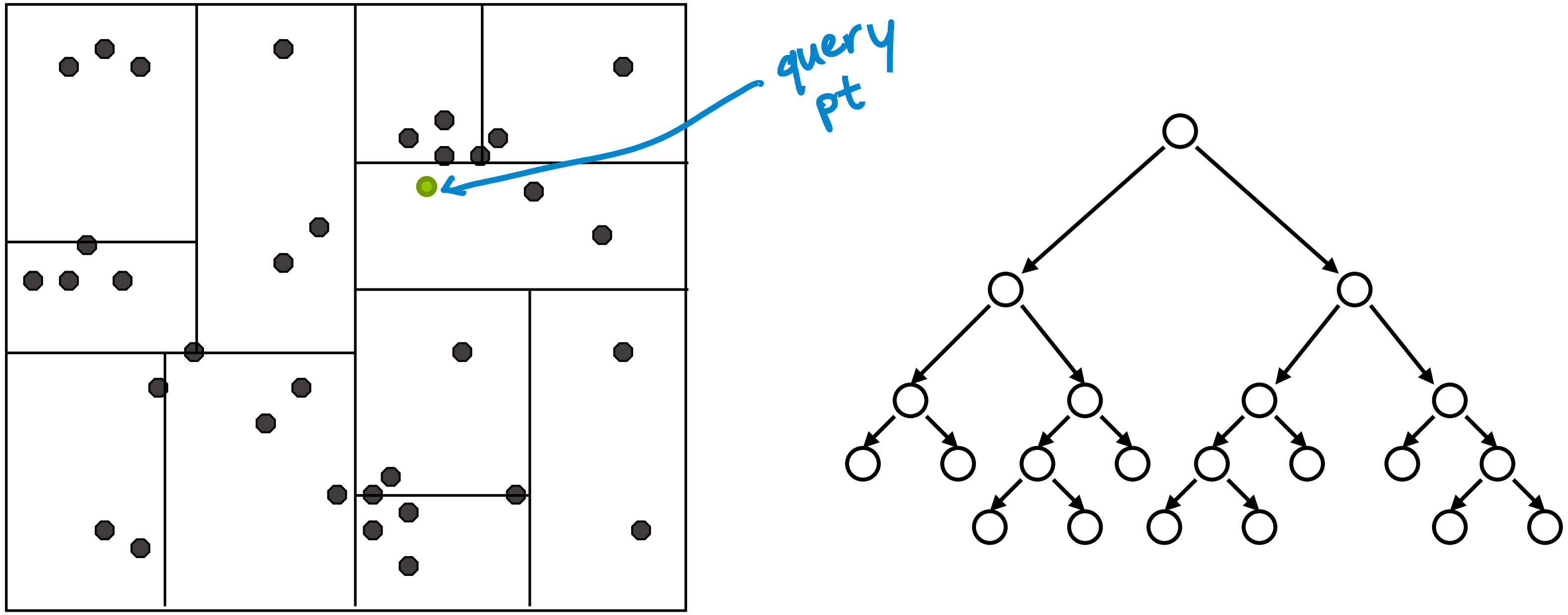
median heuristic



center-of-range heuristic

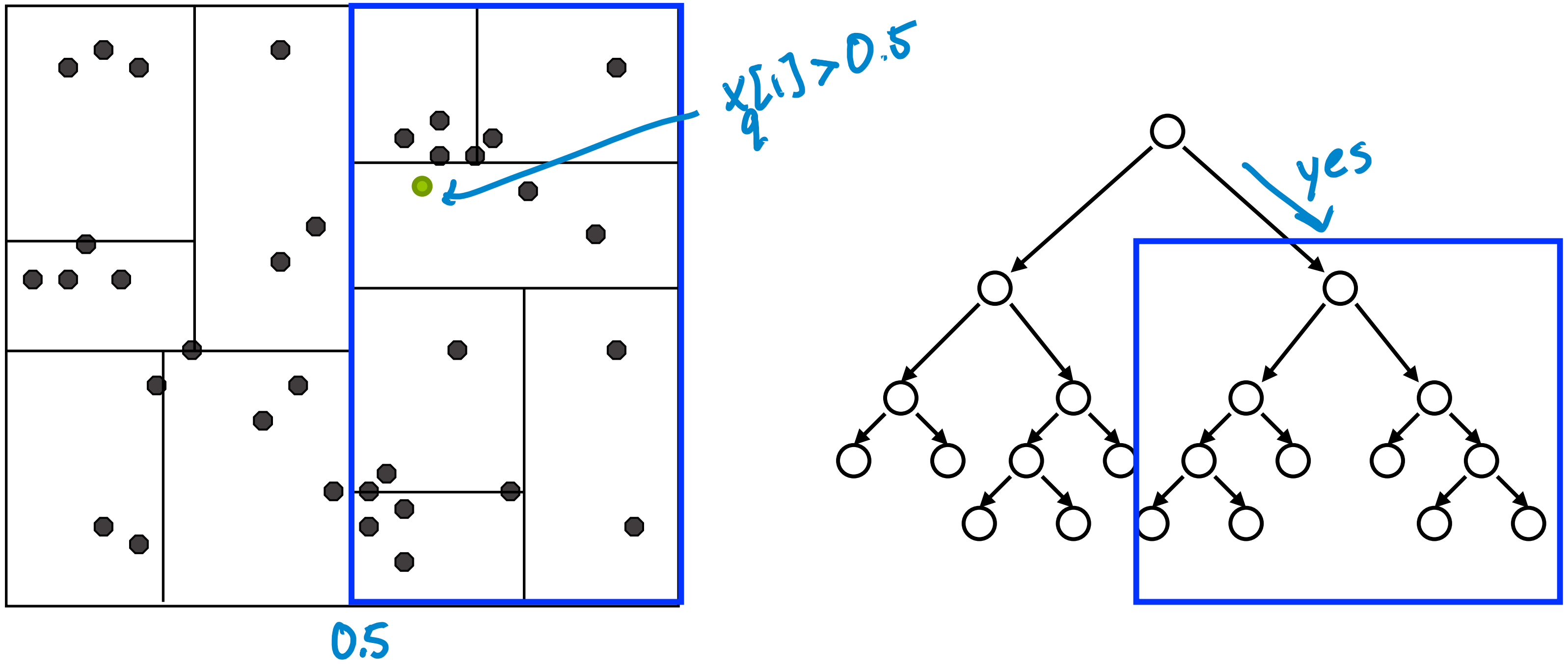
NN search with KD-trees

Nearest neighbor with KD-trees



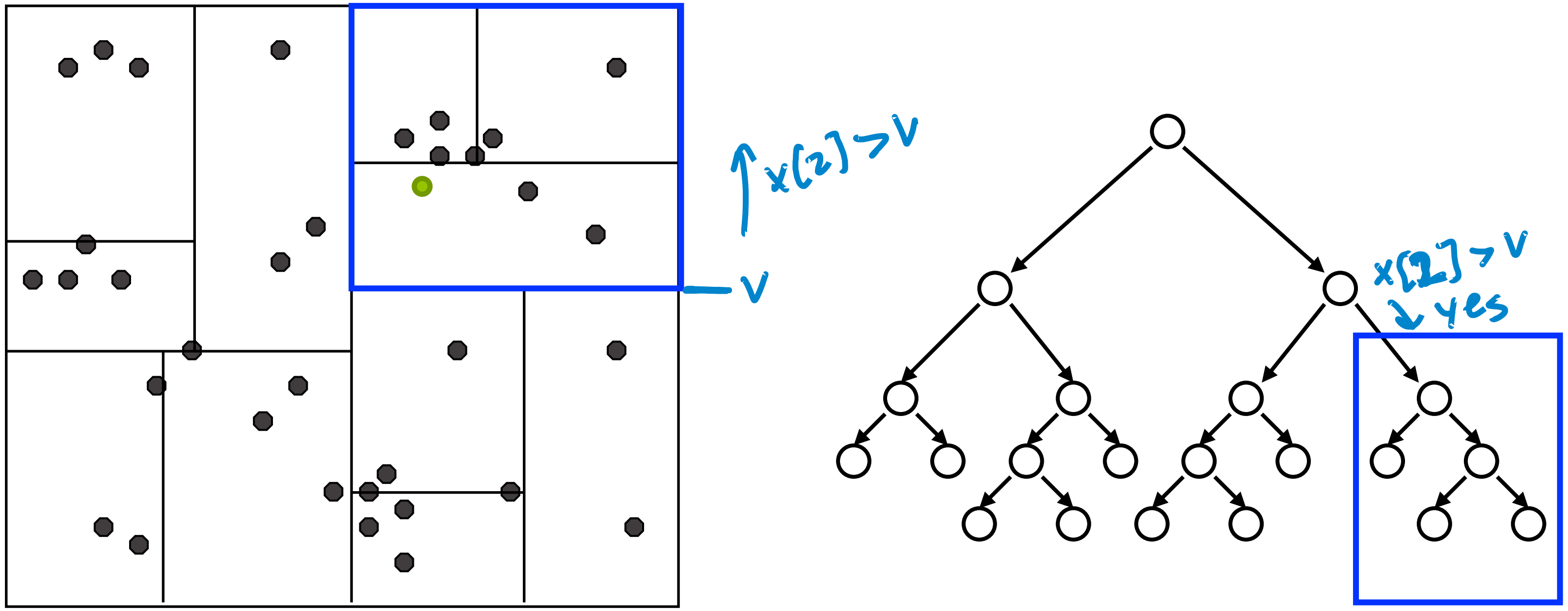
Traverse tree looking for nearest neighbor to query point

Nearest neighbor with KD-trees



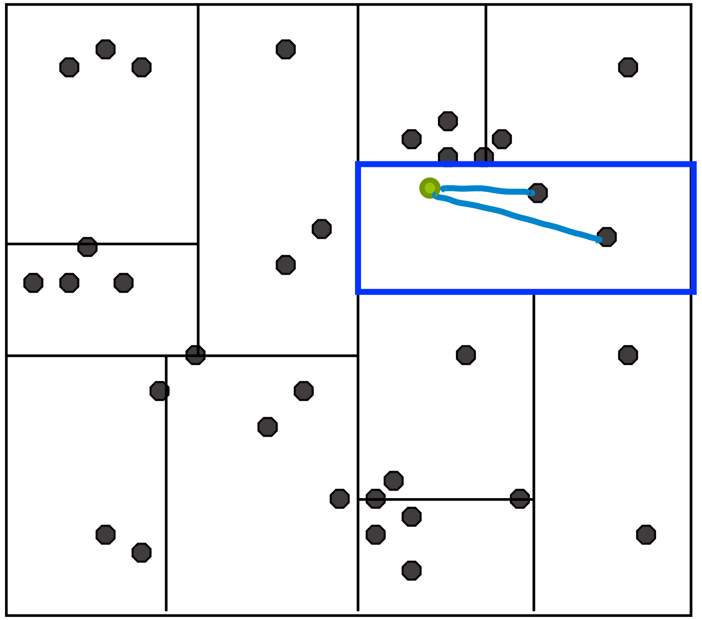
1. Start by exploring leaf node containing query point

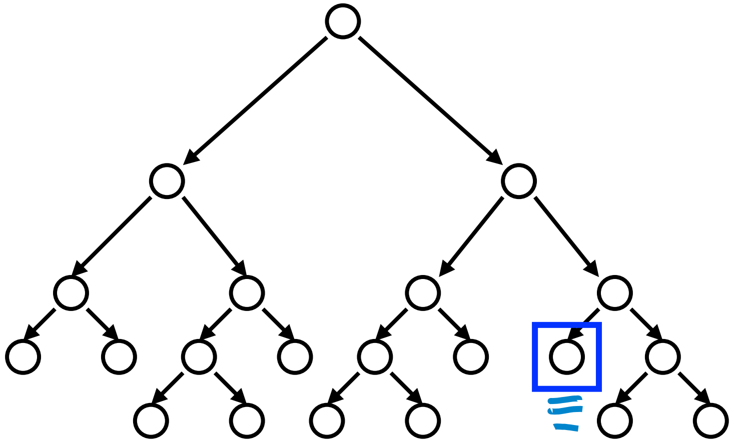
Nearest neighbor with KD-trees



1. Start by exploring leaf node containing query point

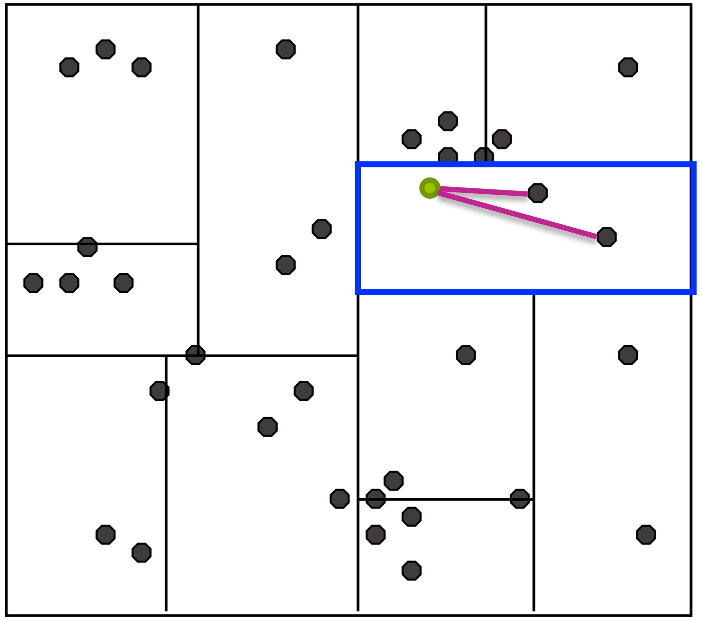
Nearest neighbor with KD-trees

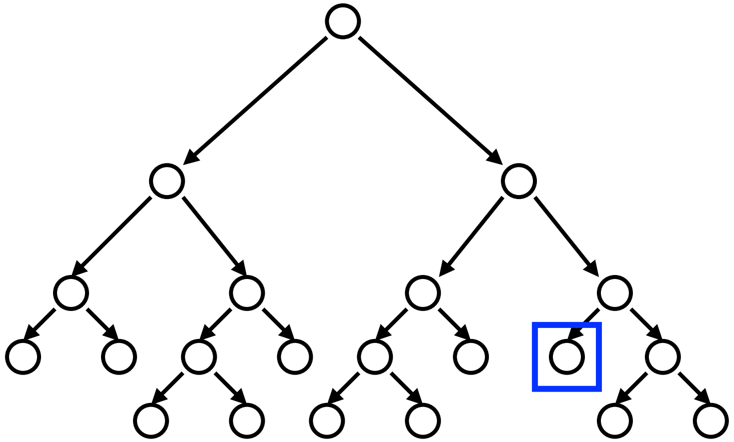




1. Start by exploring leaf node containing query point

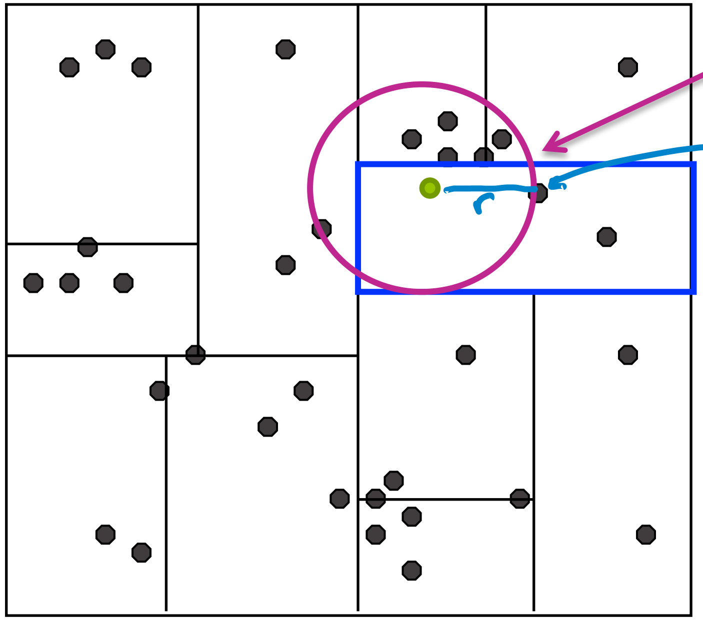
Nearest neighbor with KD-trees





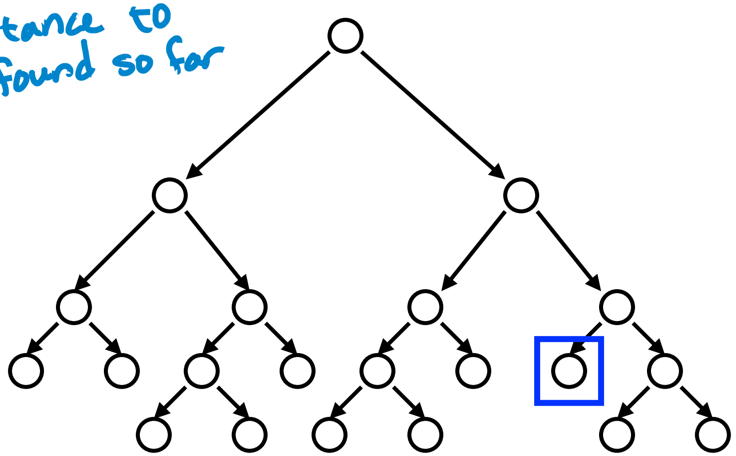
1. Start by exploring leaf node containing query point
2. Compute distance to each other point at leaf node

Nearest neighbor with KD-trees



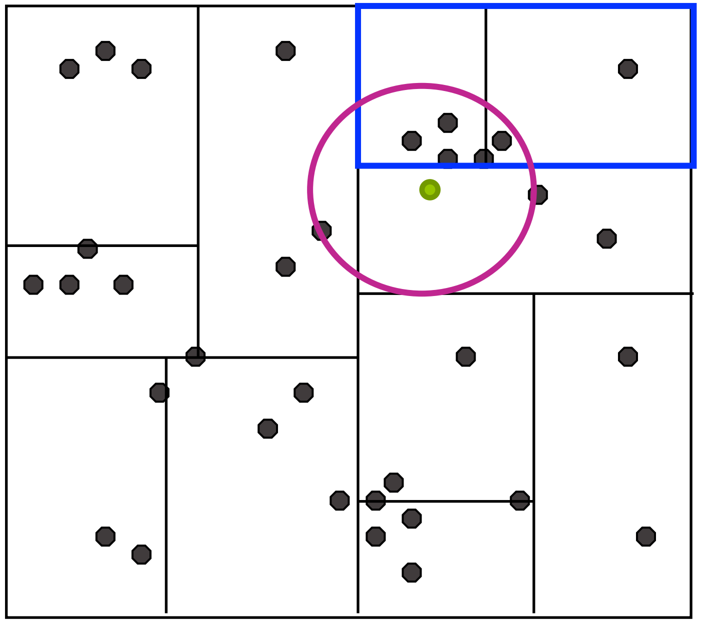
Does nearest neighbor have to live at leaf node containing query point?

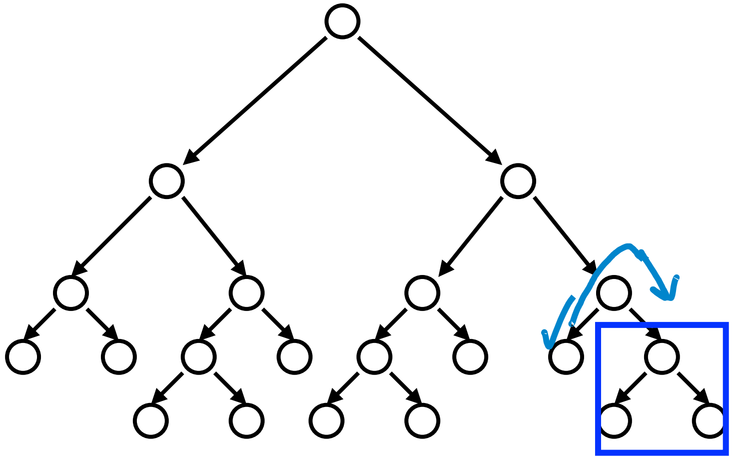
distance to NN found so far



1. Start by exploring leaf node containing query point
2. Compute distance to each other point at leaf node

Nearest neighbor with KD-trees

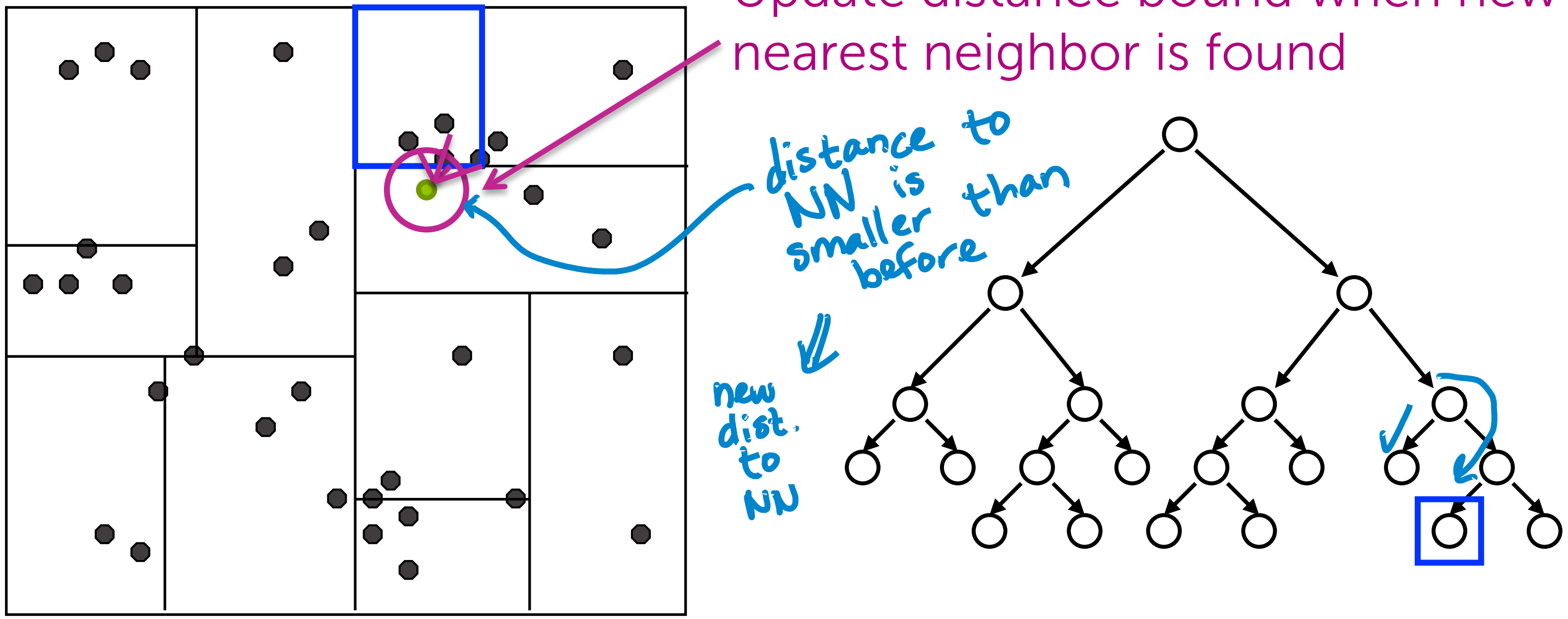




1. Start by exploring leaf node containing query point
2. Compute distance to each other point at leaf node
3. Backtrack and try other branch at each node visited

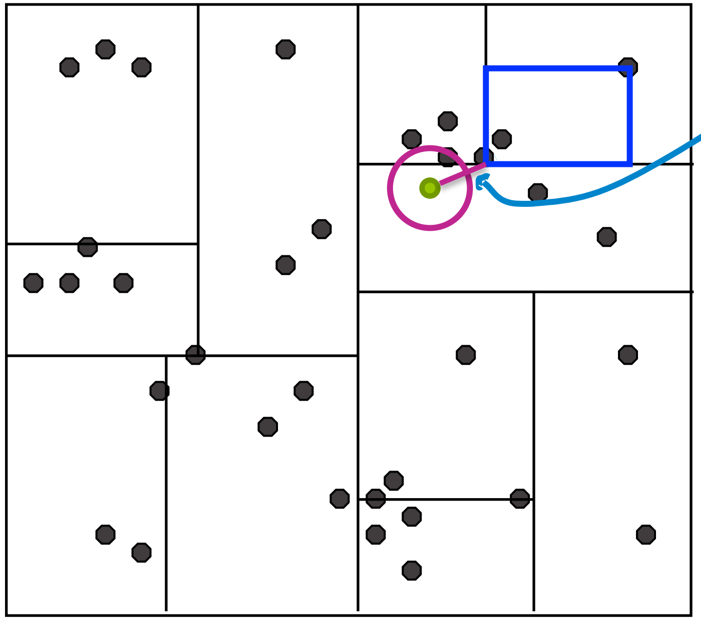
Nearest neighbor with KD-trees

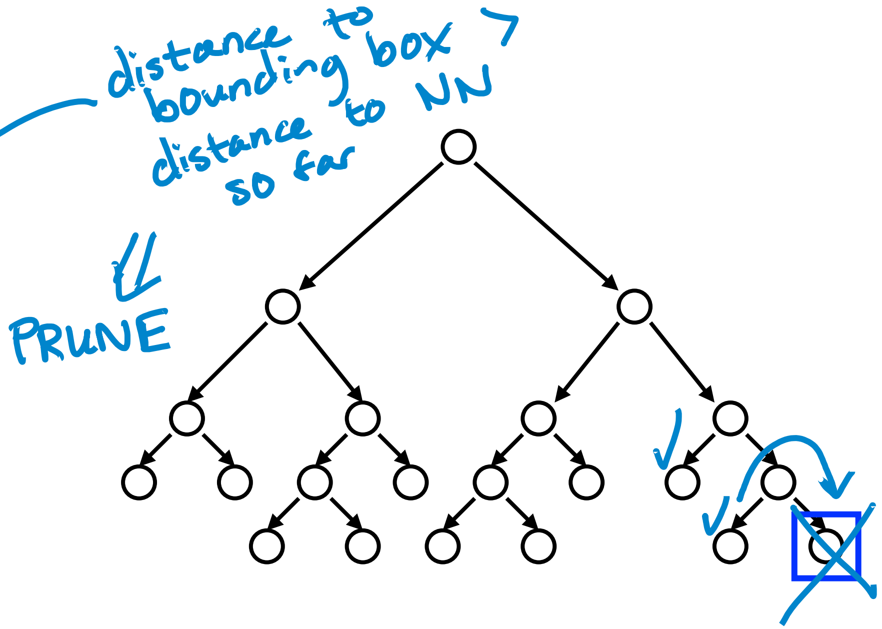
Update distance bound when new nearest neighbor is found



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

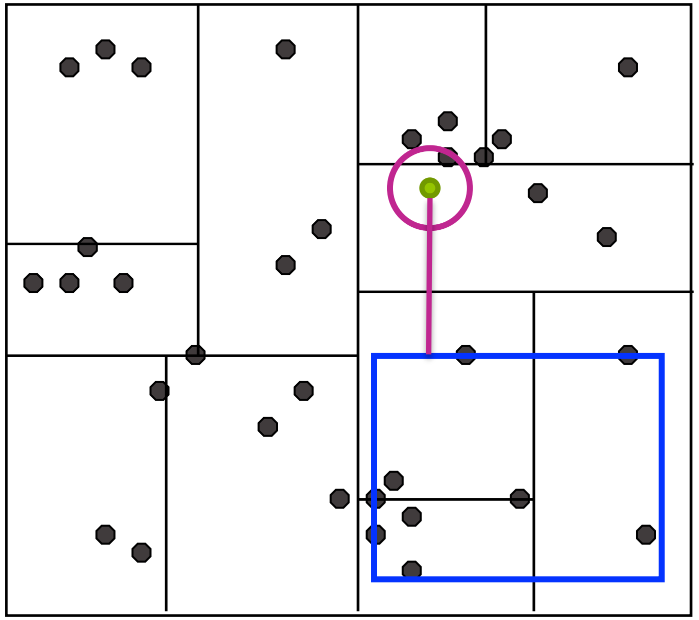
Nearest neighbor with KD-trees

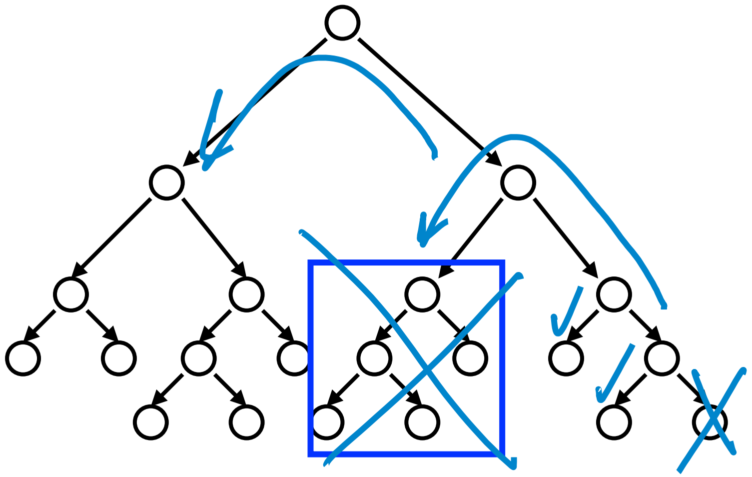




Use distance bound and bounding box of each node to prune parts of tree that cannot include nearest neighbor

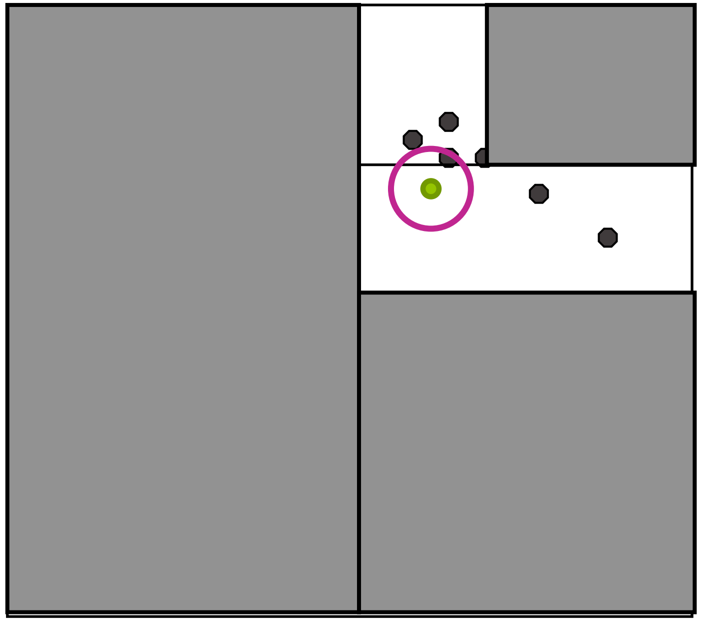
Nearest neighbor with KD-trees

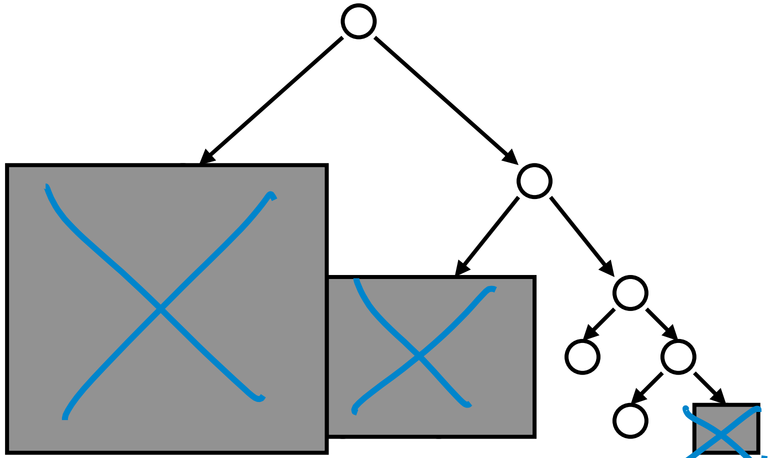




Use distance bound and bounding box of each node to
prune parts of tree that cannot include nearest neighbor

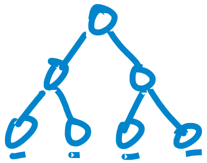
Nearest neighbor with KD-trees





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

Complexity

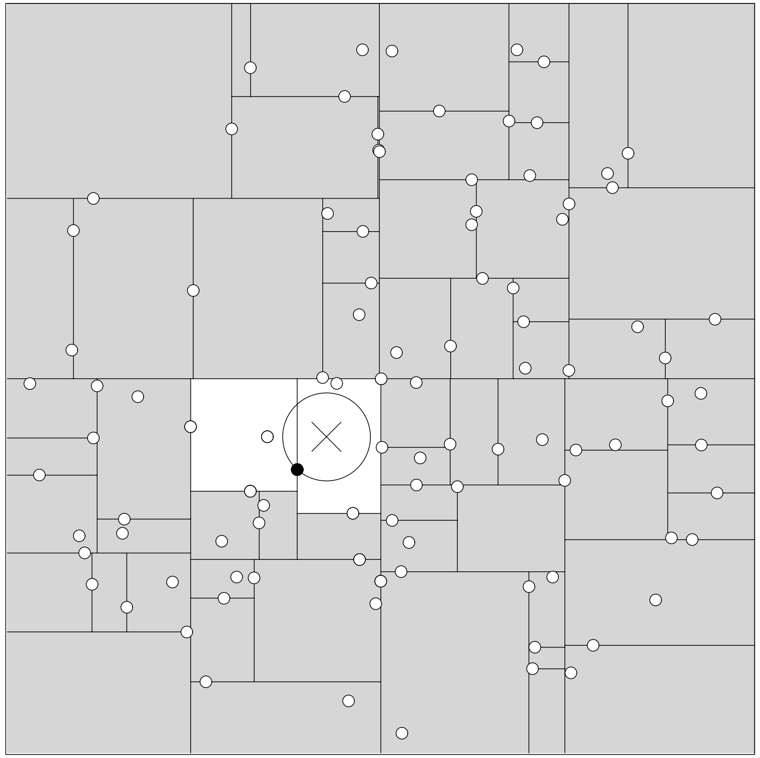


For (nearly) balanced, binary trees...

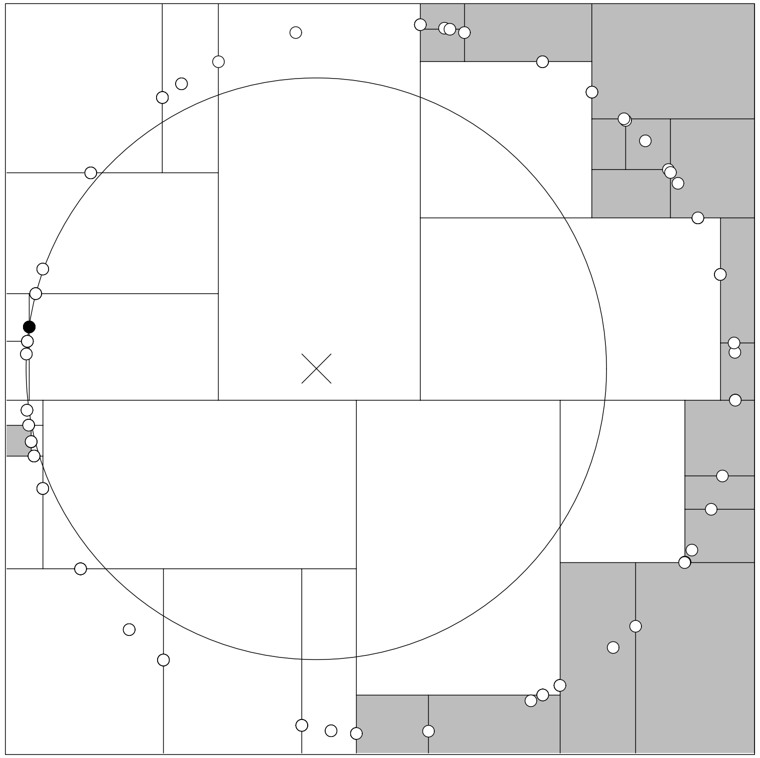
- Construction
  - Size: 2N-1 nodes if 1 datapt at each leaf → O(N)
  - Depth: O(log N)
  - Median + send points left right: O(N) at every level of the tree
  - Construction time: O(N log N)
- 1-NN query
  - Traverse down tree to starting point: O(log N)
  - Maximum backtrack and traverse: O(N) in worst case
  - Complexity range: O(log N) → O(N)

Under some assumptions on distribution of points, we get O(logN) but exponential in d

Complexity



pruned many
(closer to O(log N))



pruned few
(closer to O(N))

Complexity for N queries

- Ask for nearest neighbor to each doc

  N queries

- Brute force 1-NN:

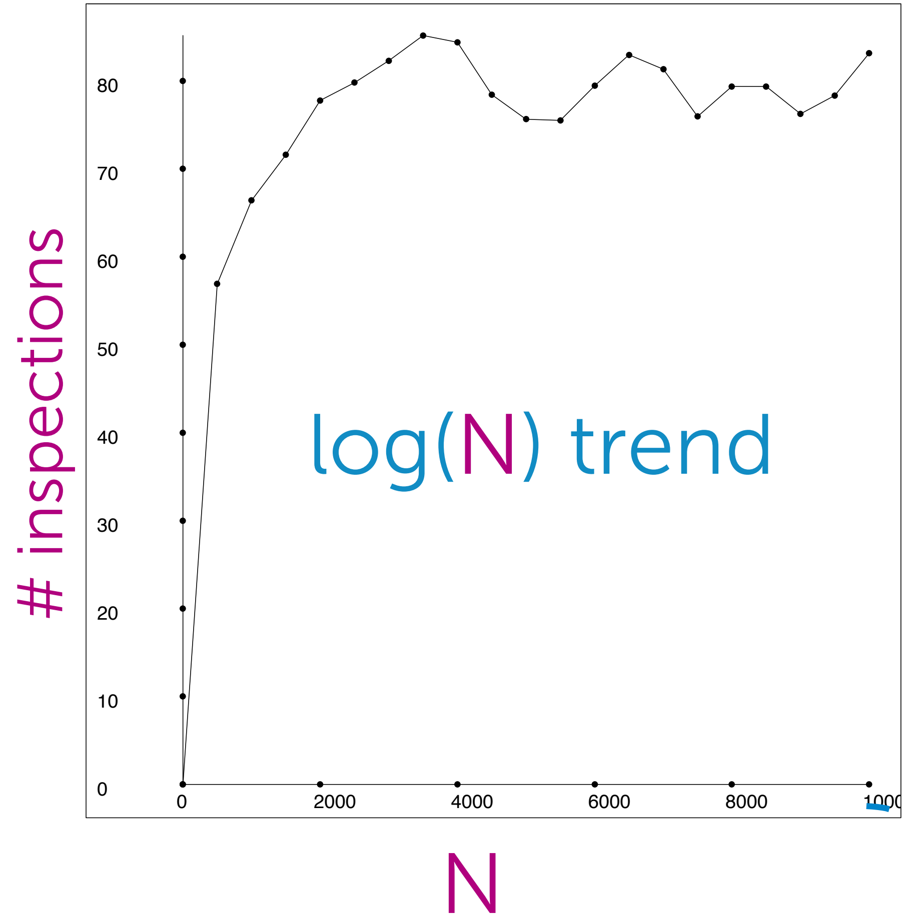
  $O(N^2)$

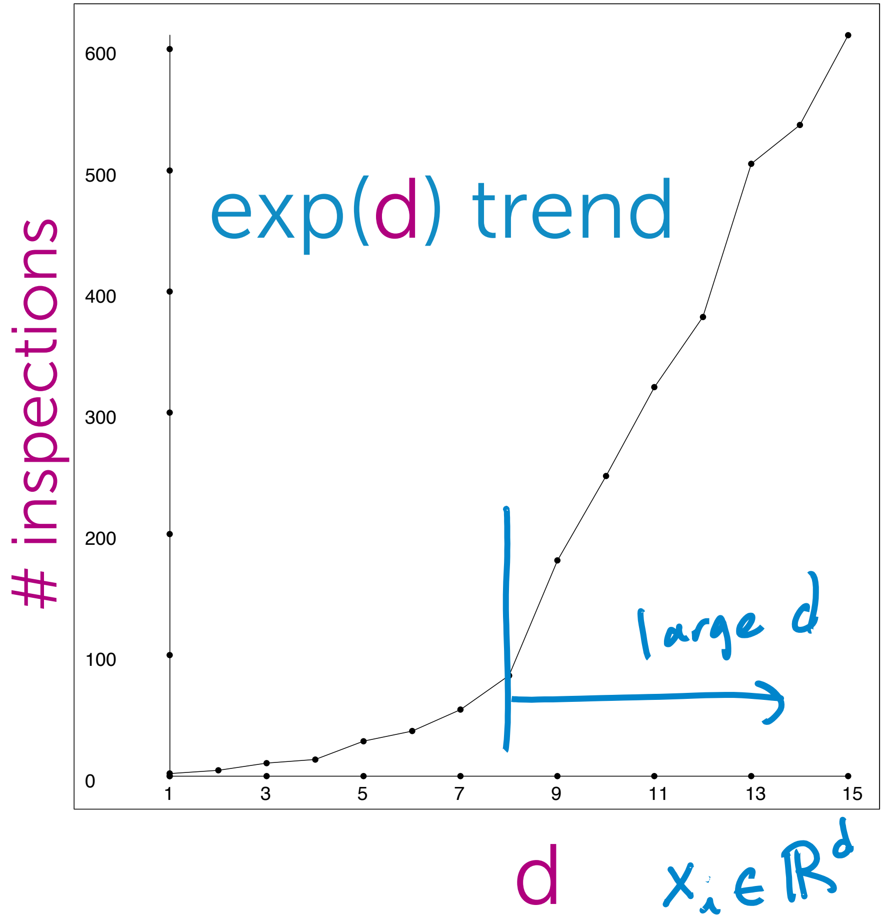
- kd-trees:

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

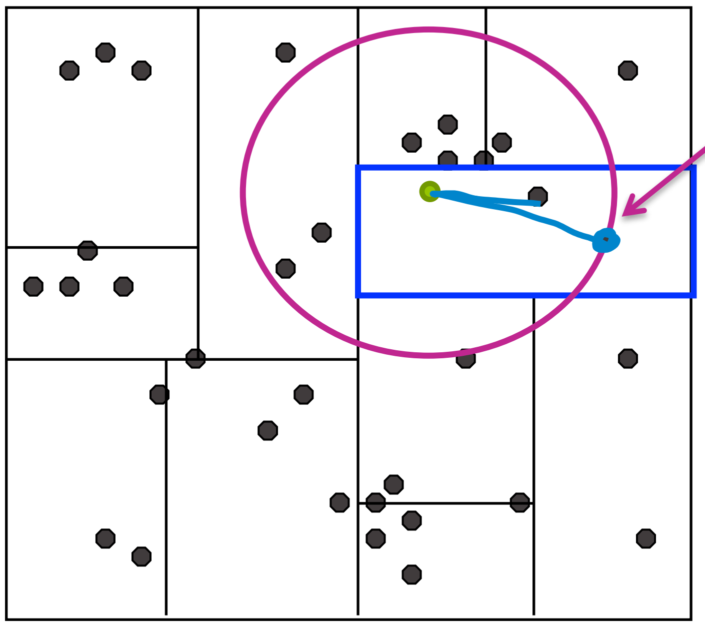
  potentially very large savings for large N!

Inspections vs. N and d

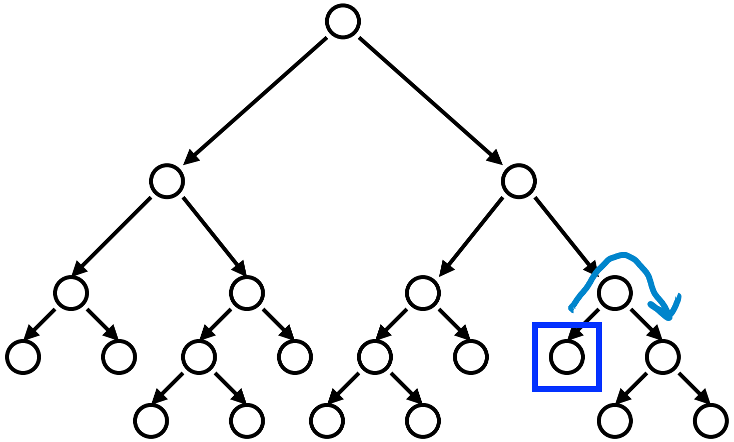




k-NN with KD-trees



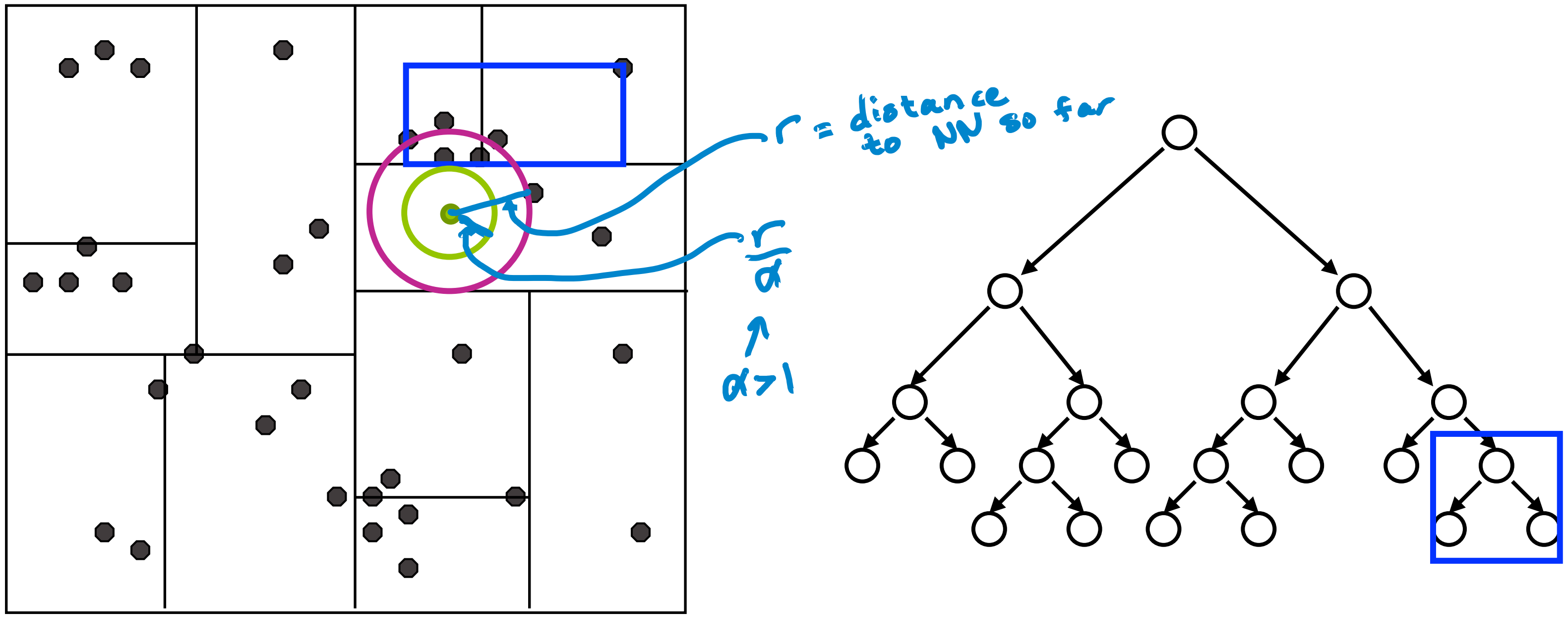
distance to 2nd nearest neighbor
in 2-NN example

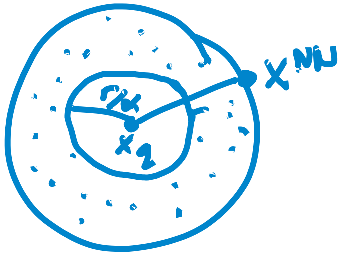


Exactly same algorithm, but maintain distance to
furthest of current *k* nearest neighbors

Approximate k-NN search

Approximate k-NN with 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 searches 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