Case Study 2: Document Retrieval

Task Description: Finding Similar Items

Machine Learning for Big Data
CSE547/STAT548, University of Washington
Sham Kakade
April 13, 2017

Announcements:

- HW2 posted
- Project Milestones
  - Start early
  - Lit. review (>= 3 papers read carefully)
  - First rounds of experiments
- Today:
  - Review: Sim search, k-NNs, KD-trees
  - Today: KD-trees (cont.), ball trees, cover trees

Task 1: Find Similar Documents

- **To begin…**
  - **Input:** Query article
  - **Output:** Set of *k* similar articles

Document Representation

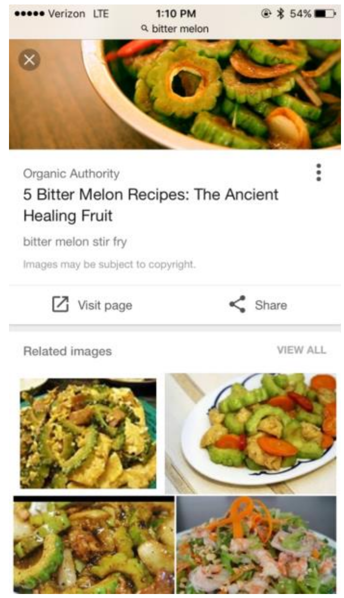
- Bag of words model

$$X = \begin{bmatrix} wc_1 \\ \vdots \\ wc_d \end{bmatrix} \in \mathbb{R}^d$$

"Bag of words" wc_i - word count of word i

ignore word order.

Image Search…



5

Where is FAST similarity search important?

- web search
- image seach
- sky maps / location
- shazam / song identification
- physics simulators
  - Robotics

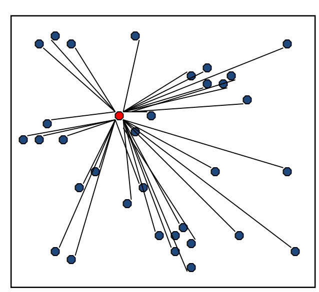
6

1-Nearest Neighbor

- Articles = $\mathcal{X} = \{x^1 \ldots x^N\}$
  $x^i \in \mathbb{R}^d$
- Query:
  x
- 1-NN
  - Goal: find $x \in \mathcal{X} \setminus x$ "closest" to x.
    query
  - Formulation:
    $$x^{NN} \in \underset{x^i \in \mathcal{X} \setminus x}{\arg\min} \; d(x^i, x)$$

Issues with Search Techniques

- Naïve approach:
  Brute force search
  - Given a query point x
  - Scan through each point x^i
  - O(N) distance computations per 1-NN query!
  - O(Nlog$k$) per k-NN query!



33 Distance Computations

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

Think about Web Search/Image Search

- How big is N?

# of web pages

N → # of images

- How fast do we desire to do recall?

Intuition (?): NN in 1D and Sorting

- How do we do 1-NN searches in 1 dim?

or How do we sort?

approx.

- Pre-processing time:

O(N)

bins

sorting

O(N log N)

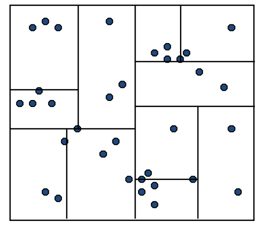
- Query time:

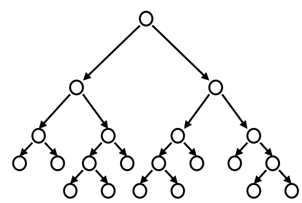
O(1)

O(log N)

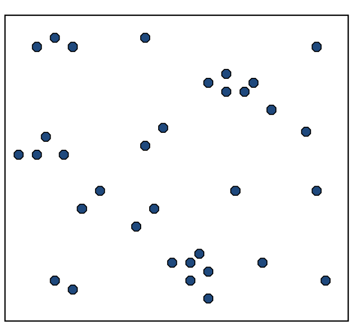
KD-Trees

- Smarter approach: ***kd-trees***
  - Structured organization of documents
    - Recursively partitions points into axis aligned boxes.
  - Enables more efficient pruning of search space
    - Examine nearby points first.
    - Ignore any points that are further than the nearest point found so far.
- ***kd-trees*** work “well” in “low-medium” dimensions
  - We’ll get back to this…





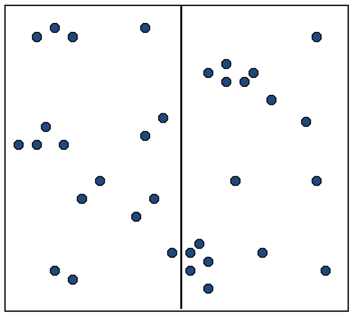
KD-Tree Construction



Pt	X	Y
1	0.00	0.00
2	1.00	4.31
3	0.13	2.85
…	…	…

- Start with a list of *d*-dimensional points.

KD-Tree Construction



X>.5

NO

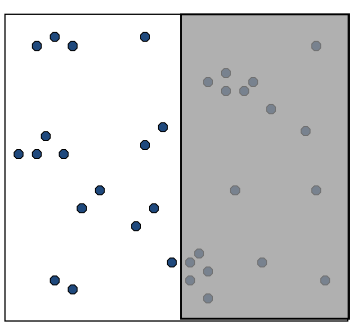
Pt	X	Y
1	0.00	0.00
3	0.13	2.85
…	…	…

YES

Pt	X	Y
2	1.00	4.31
…	…	…

- Split the points into 2 groups by:
  - Choosing dimension d_j and value V (methods to be discussed…)
  - Separating the points into $x^i_{d_j}$> V and $x^i_{d_j}$<= V.

KD-Tree Construction



X>.5

NO

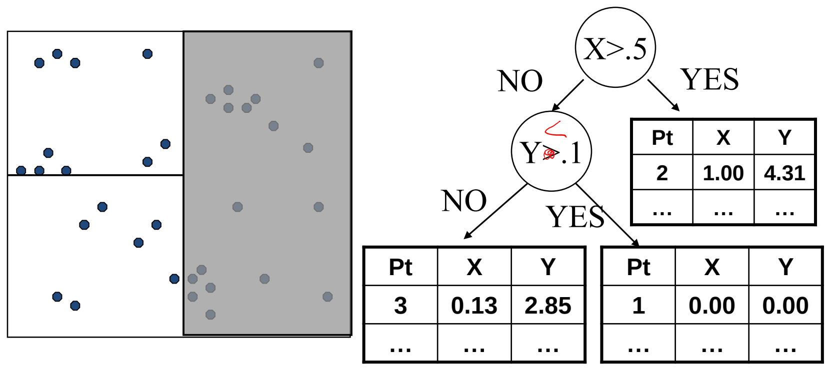
Pt	X	Y
1	0.00	0.00
3	0.13	2.85
…	…	…

YES

Pt	X	Y
2	1.00	4.31
…	…	…

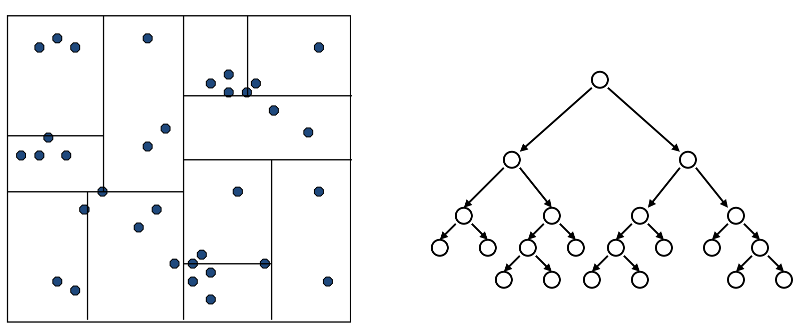
- Consider each group separately and possibly split again (along same/different dimension).
  - Stopping criterion to be discussed…

KD-Tree Construction



- Consider each group separately and possibly split again (along same/different dimension).
  - Stopping criterion to be discussed…

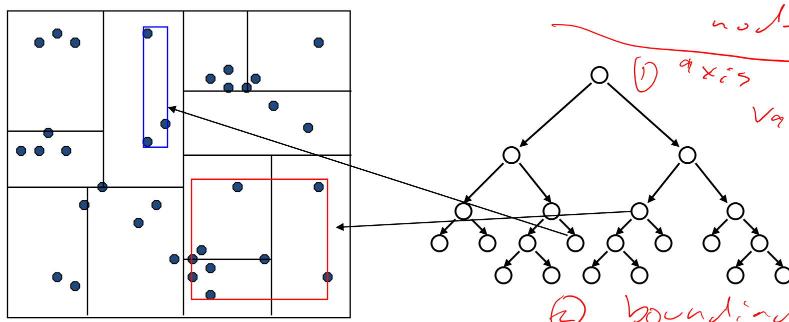
KD-Tree Construction



- Continue splitting points in each set
  - creates a binary tree structure
- Each leaf node contains a list of points

KD-Tree Construction

maintain at all nodes



(1) axis & value

(2) bounding box

- Keep one additional piece of information at each node:
  - The (tight) bounds of the points at or below this node.

KD-Tree Construction

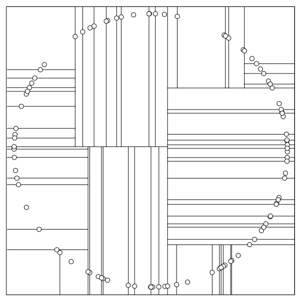
- Use heuristics to make splitting decisions:
- Which dimension do we split along?

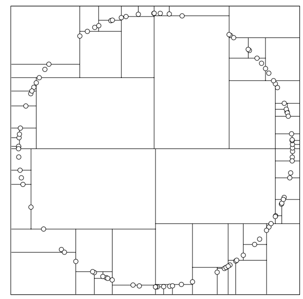
  widest (some "variance" measure)
- Which value do we split at?

  median "center-of-range"
- When do we stop?

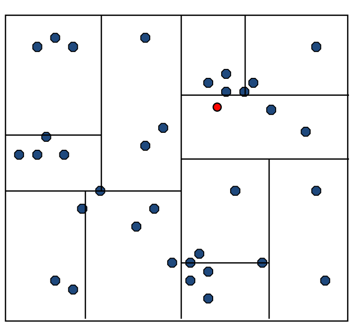
  stop when each box has ≤ m points.

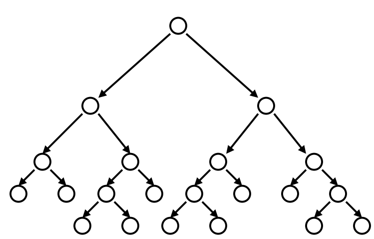
Many heuristics…


median heuristic


center-of-range heuristic

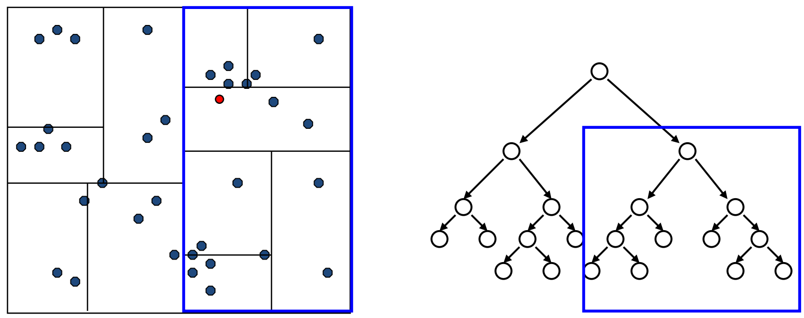
Nearest Neighbor with KD Trees





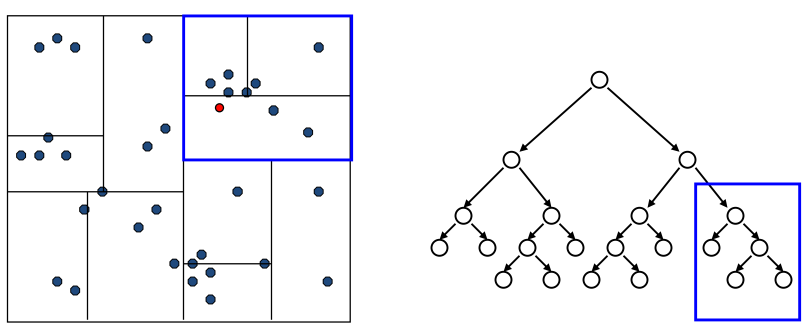
- Traverse the tree looking for the nearest neighbor of the query point.

Nearest Neighbor with KD Trees



- Examine nearby points first:
  - Explore branch of tree closest to the query point first.

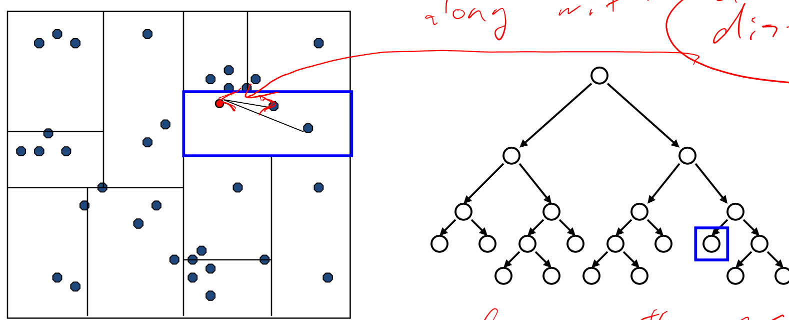
Nearest Neighbor with KD Trees



- Examine nearby points first:
  - Explore branch of tree closest to the query point first.

Nearest Neighbor with KD Trees

guess at NN
along with distance

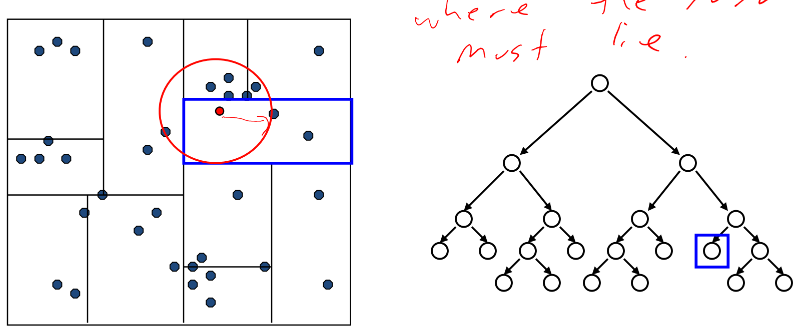


does the NN
have to be in
this box??

- When we reach a leaf node:
  - Compute the distance to each point in the node.

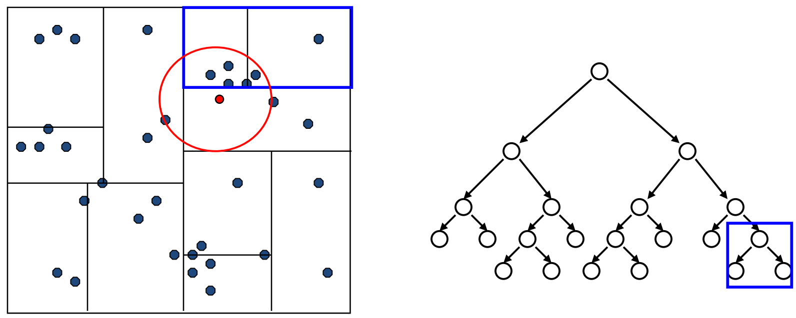
Nearest Neighbor with KD Trees

guaranteed region
where the NN
must lie.



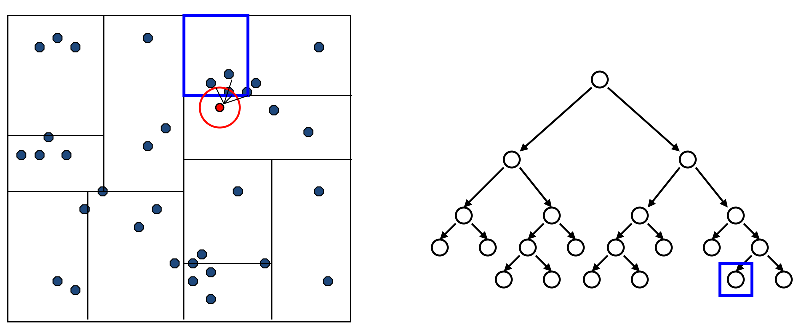
- When we reach a leaf node:
  - Compute the distance to each point in the node.

Nearest Neighbor with KD Trees



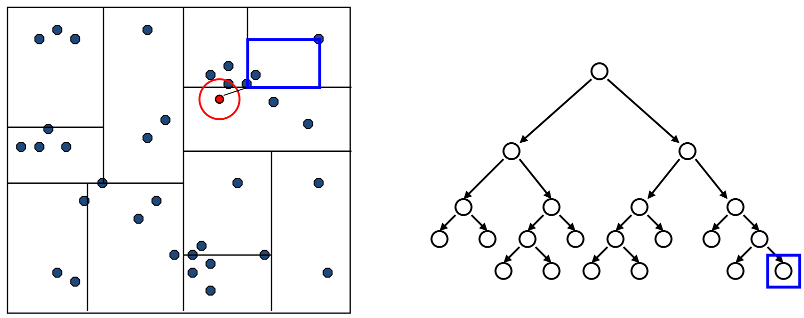
- Then backtrack and try the other branch at each node visited

Nearest Neighbor with KD Trees



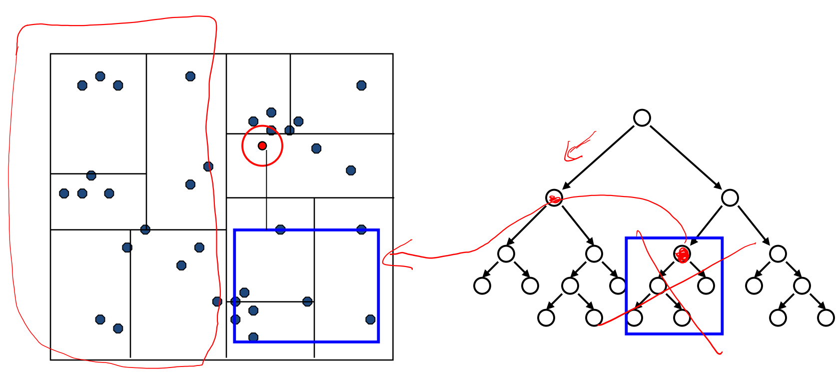
- Each time a new closest node is found, update the distance bound

Nearest Neighbor with KD Trees



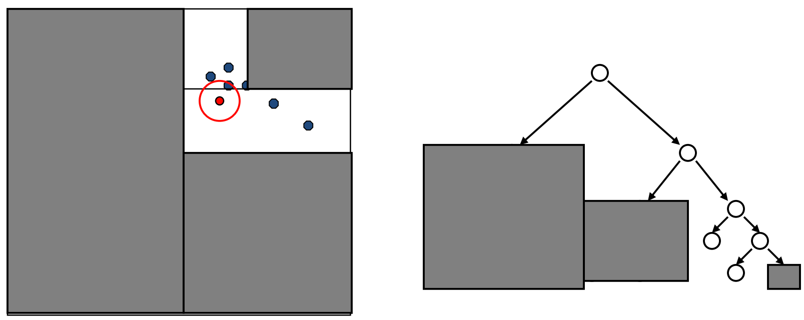
- Using the distance bound and bounding box of each node:
  - Prune parts of the tree that could NOT include the nearest neighbor

Nearest Neighbor with KD Trees



- Using the distance bound and bounding box of each node:
  - Prune parts of the tree that could NOT include the nearest neighbor

Nearest Neighbor with KD Trees



- Using the distance bound and bounding box of each node:
  - Prune parts of the tree that could NOT include the nearest neighbor

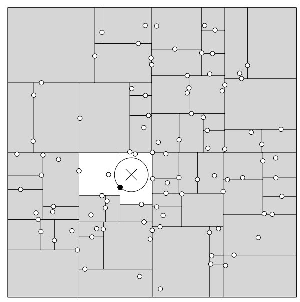
Complexity

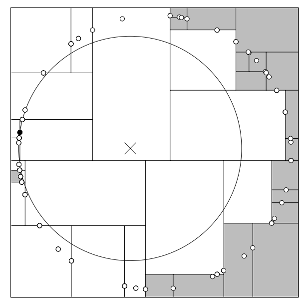
- For (nearly) balanced, binary trees...
- Construction
  - Size: O(N)
  - Depth: O(log N) (if can balance)
  - Median + send points left right:
  - Construction time: O(N log N)
- 1-NN query
  - Traverse down tree to ~~starting point~~ leaf node: log (N)
  - Maximum backtrack and traverse: O(N)
  - Complexity range:

  O(log N) ⟷ O(N)
- Under some assumptions on distribution of points, we get O(log*N*) but exponential in *d* (see citations in reading)

Complexity

Instead, as you set query near the center, rule out almost nothing





$O(\log N)$ $O(N)$

Complexity for *N* Queries

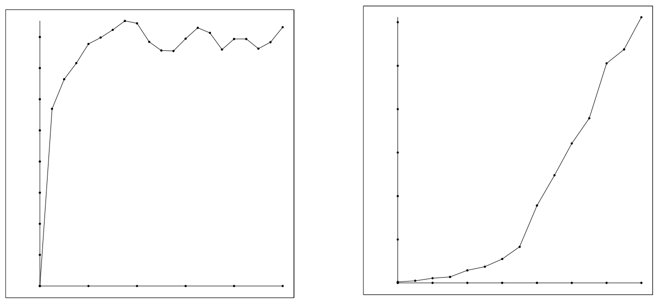
- Ask for nearest neighbor to each document
- Brute force 1-NN:

  $O(N^2)$
- kd-trees:

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

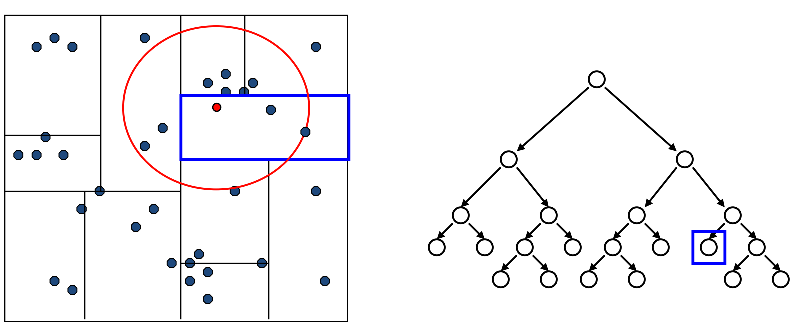
Inspections vs. *N* and *d*

complexity



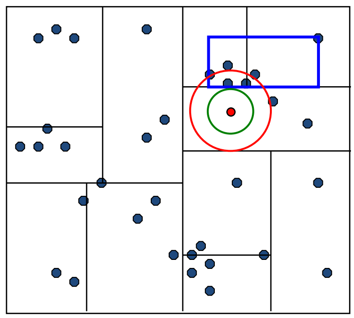
N d

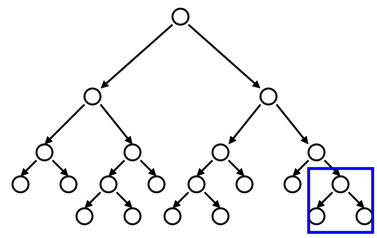
K-NN with KD Trees



- Exactly the same algorithm, but maintain distance as distance to furthest of current *k* nearest neighbors
- Complexity is: $O(K \log N)$

Approximate K-NN with KD Trees





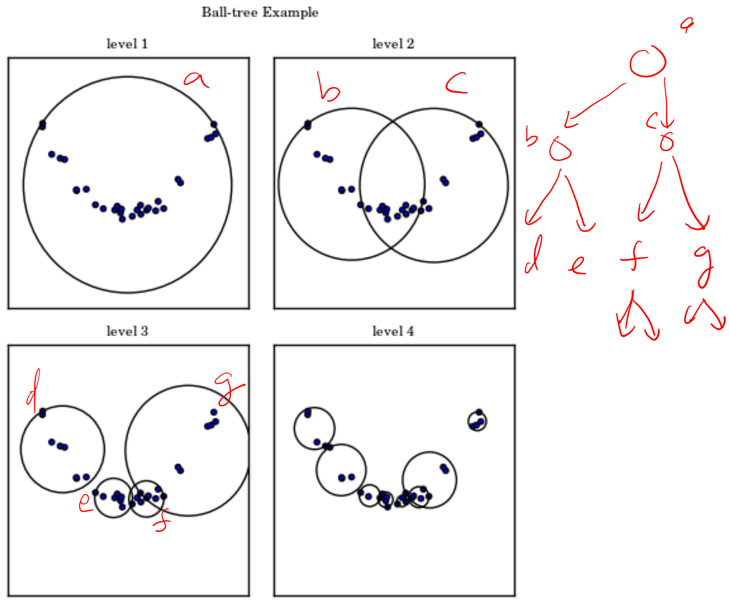
- **Before:** Prune when distance to bounding box > r
- **Now:** Prune when distance to bounding box > r/α
- Will prune more than allowed, but can guarantee that if we return a neighbor at distance r, then there is no neighbor closer than r/α.
- In practice this bound is loose…Can be closer to optimal.
- Saves lots of search time at little cost in quality of nearest neighbor.

What about NNs searches in high dimensions?

- KD-trees:
  - What is going wrong?

    axis aligned splits
  - Can this be easily fixed?
- What do have to utilize?
  - utilize triangle inequality of metric
  - New ideas: ball trees and cover trees

Ball Trees



Ball Tree Construction

- Node:
  - Every node defines a ball (hypersphere), containing
    - a subset of the the points (to be searched)
    - A center
    - A (tight) radius of the points
- Construction:
  - Root: start with a ball which contains all the data
  - take a ball and make two children (nodes) as follows:
    - Make two spheres, assign each point (in the parent sphere) to its closer sphere
    - Make the two spheres in a “reasonable” manner

Ball Tree Search

- Given point x, how do find its nearest neighbor quickly?
- Approach:
  - Start: follow a greedy path through the tree
  - Backtrack and prune: rule out other paths based on the triange inequality
    - (just like in KD-trees)

in high dimensions??

- How good is it?
  - Guarantees: → worst case complexity is bad;
  - Practice: one of best for exact NN-searches

Cover trees

- What about exact NNs in general metric spaces?
- Same Idea: utilize triangle inequality of metric (so allow for arbitrary metric)
- What does the dimension even mean?
- cover-tree idea: exploit the structure in the data

Intrinsic Dimension

- How does the volume grow, from radius R to 2R?

$$\frac{Vol(Ball_{2R})}{Vol(Ball_R)} = 2^d$$

- Can we relax this idea to get at the "intrinsic" dimension?
  - This is the "doubling" dimension:

Cover trees: data structure

- Ball Trees: each node had associated
  - Center:
  - (tight) Radius:
  - Points:
- Cover trees:
  - Center:
  - (tight) Radius:
  - Points:

Cover Tree Complexity

- Construction
  - Size:
  - Construction time:
- 1-NN query
  - Traverse down tree to starting point:
  - Maximum backtrack and traverse:
- Under assumptions that doubling dimension is D.

Wrapping Up – Important Points

kd-trees

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

Nearest Neighbor Search

- Distance metric and data representation are crucial to answer returned

For both…

- High dimensional spaces are hard!
  - Number of kd-tree searches can be exponential in dimension
    - Rule of thumb… $N >> 2^d$… Typically useless.
  - Distances are sensitive to irrelevant features
    - Most dimensions are just noise → Everything equidistant (i.e., everything is far away)
    - Need technique to learn what features are important for your task

What you need to know

- Document retrieval task
  - Document representation (bag of words)
  - tf-idf
- Nearest neighbor search
  - Formulation
  - Different distance metrics and sensitivity to choice
  - Challenges with large *N*
- kd-trees for nearest neighbor search
  - Construction of tree
  - NN search algorithm using tree
  - Complexity of construction and query
  - Challenges with large *d*