Using multiple tables for even greater efficiency in NN search

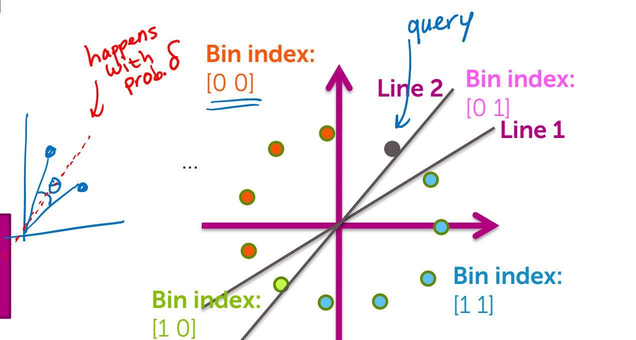
If I throw down 2 lines…

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

For simplicity, assume we **search bins 1 bit off** from query

Let δ be the probability of a line falling between points θ apart

Search 3 bins and do not find NN with probability δ^2

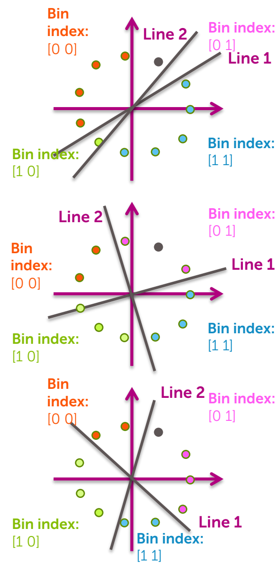


What if I repeat the 2-line binning?

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

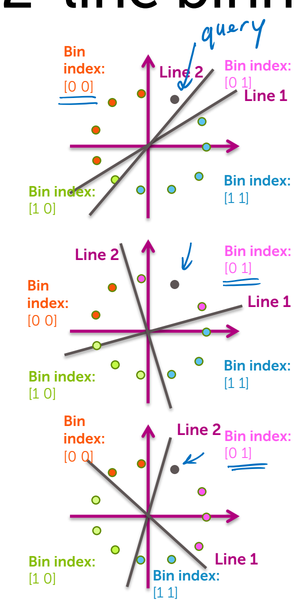


What if I repeat the 2-line binning?

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3



Now, search only query bin **per table**

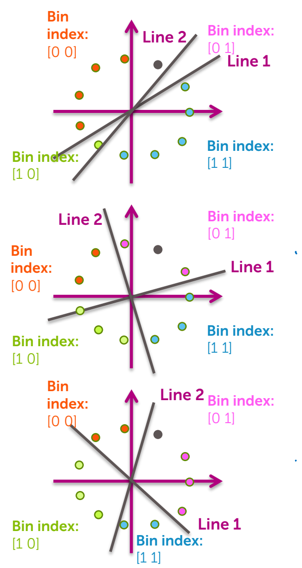
Still searching 3 bins, but what is chance of not finding NN?

What if I repeat the 2-line binning?

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3



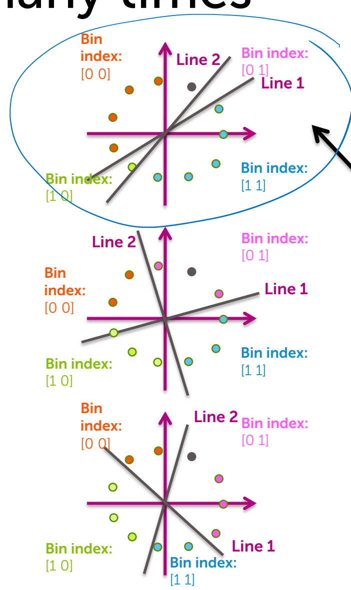
What is chance that query pt and NN are split in **all tables**?

Probability of splitting neighboring points many times

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3



Probability NN is in different bin:

Prob = 1-Pr(same bin)

= 1-(1-δ)^2

= 2δ - δ^2

$1-\delta$ = prob. that 1 line does not split query + NN

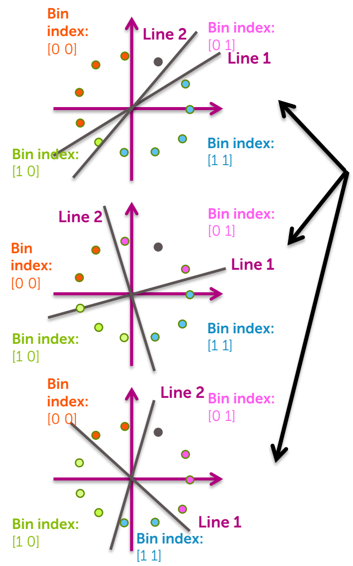
$(1-\delta)^2$ = prob. that 2 lines don't split pts

Probability of splitting neighboring points many times

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3



Probability NN is in different bin in **all 3 tables**:

Prob = $(2\delta - \delta^2)^3$

\# of hash tables

Comparing approaches for 2-bit tables

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

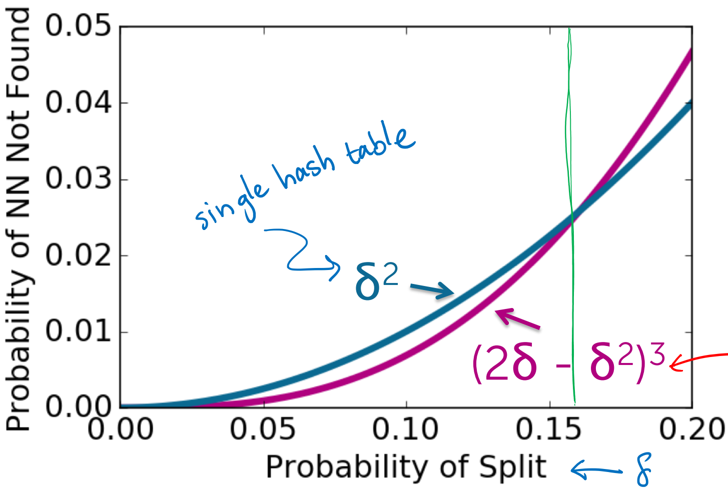
Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

# bins searched	**prob. of no NN**
3	δ^2
3	$(2\delta - \delta^2)^3$

Comparing probabilities



– throw down 2 lines
– searching 3 bins

multiple hash table

If I throw down h lines…

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
indices	L0	L1	L2	L3	L4	L5	L6	L7

Still assume we **search bins 1 bit off** from query

Prob. of being > 1 bit away

= 1-Pr(same bin)-Pr(1 bin away)

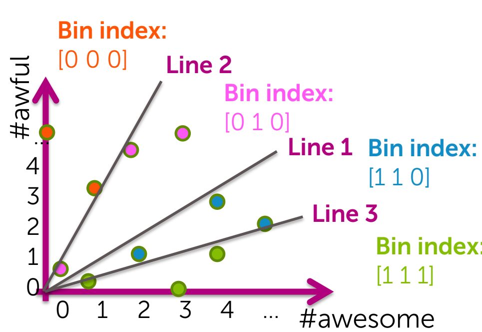
= 1-Pr(no split lines)-Pr(1 split line)

= $1-(1-\delta)^h-h\delta(1-\delta)^{h-1}$

Prob. of 1 line splitting

prob. of h-1 lines not splitting

Prob. of no split is 1-δ + we have h lines ⇒ Prob. none of h lines split is $(1-\delta)^h$



If I throw down h lines…

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
indices	L0	L1	L2	L3	L4	L5	L6	L7

Still assume we **search bins 1 bit off** from query

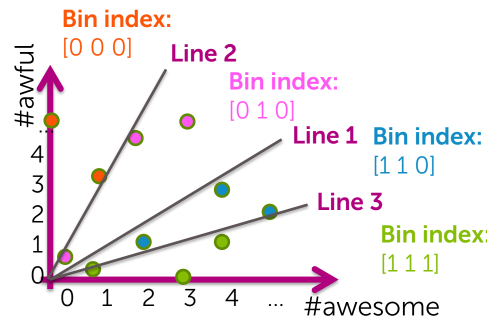
Prob. of being > 1 bit away

= 1-Pr(same bin)-Pr(1 bin away)

= 1-Pr(no split lines)-Pr(1 split line)

= $1-(1-\delta)^h-h\delta(1-\delta)^{h-1}$

Search h+1 bins and do not find NN with probability $1-(1-\delta)^h-h\delta(1-\delta)^{h-1}$



Probability of splitting neighboring points many times

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
indices	L0	L1	L2	L3	L4	L5	L6	L7

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
indices	L0	L1	L2	L3	L4	L5	L6	L7

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
indices	L0	L1	L2	L3	L4	L5	L6	L7

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
indices	L0	L1	L2	L3	L4	L5	L6	L7

Probability NN is in different bin in **all h+1 tables**

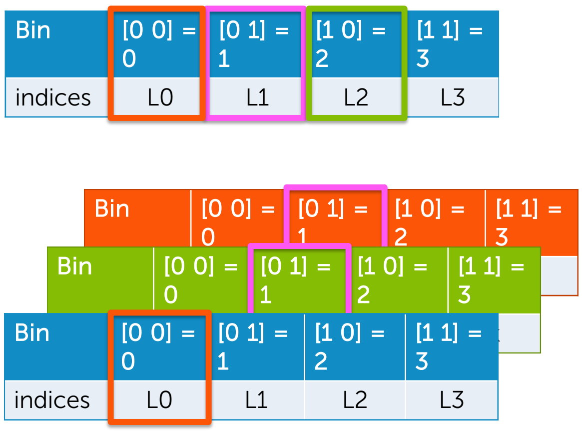
= $(1-\Pr(\text{same bin}))^{\underline{h+1}}$

= $(1-\Pr(\text{no split line}))^{h+1}$

= $(1-\underbrace{(1-\delta)^h}_{\text{prob. of no split from any of h lines thrown down}})^{\underline{h+1}}$ holds for all hash tables

Comparing approaches for h-bit tables

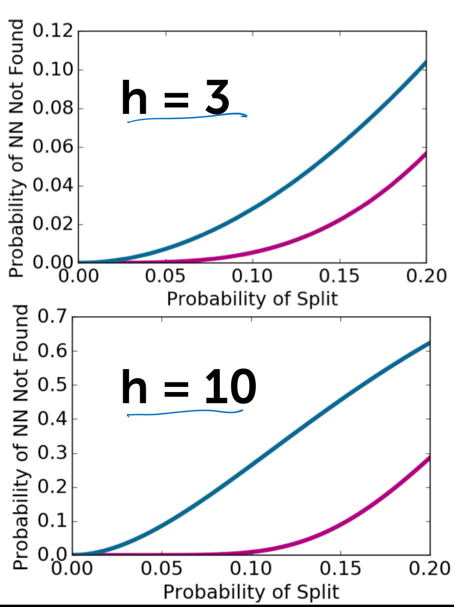
throw down h lines



	# bins searched	prob. of no NN
one table	$h+1$	$1-(1-\delta)^h-h\delta(1-\delta)^{h-1}$
multiple tables	$h+1$	$(1-(1-\delta)^h)^{h+1}$

13 ©2018 Emily Fox STAT/CSE 416: Intro to Machine Learning

Comparing probabilities



one hash table

$1-(1-\delta)^h-h\delta(1-\delta)^{h-1}$

$(1-(1-\delta)^h)^{h+1}$

multiple hash table

14 ©2018 Emily Fox STAT/CSE 416: Intro to Machine Learning

Fix #bits and increase depth

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
indices	L0	L1	L2	L3	L4	L5	L6	L7

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
indices	L0	L1	L2	L3	L4	L5	L6	L7

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
indices	L0	L1	L2	L3	L4	L5	L6	L7

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
indices	L0	L1	L2	L3	L4	L5	L6	L7

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
indices	L0	L1	L2	L3	L4	L5	L6	L7

Probability NN is in different bin in **all tables** falls off exponentially fast

hash tables

Prob

$= (1-\Pr(\text{same bin}))^{m}$

$= (1-\Pr(\text{no split line}))^{m}$

$= (1-(1-\delta)^{h})^{m}$

# of lines (bits)

15 ©2018 Emily Fox STAT/CSE 416: Intro to Machine Learning

Fix #bits and increase depth

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
indices	L0	L1	L2	L3	L4	L5	L6	L7

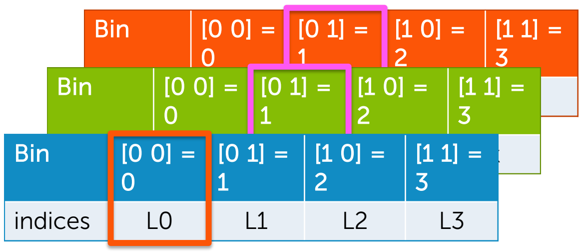
Typically higher probability of finding NN than searching m bins in 1 table

16 ©2018 Emily Fox STAT/CSE 416: Intro to Machine Learning

Summary of LSH approaches

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

Cost of binning points is **lower**, but likely need to search **more bins** per query



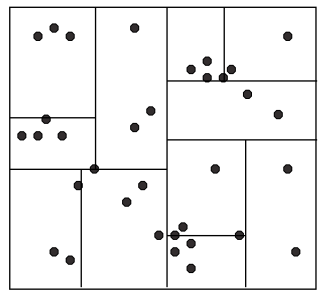
Cost of binning points is **higher**, but likely need to search **fewer bins** per query

KD-trees

KD-trees

Structured organization of documents

- Recursively partitions points into axis aligned boxes.

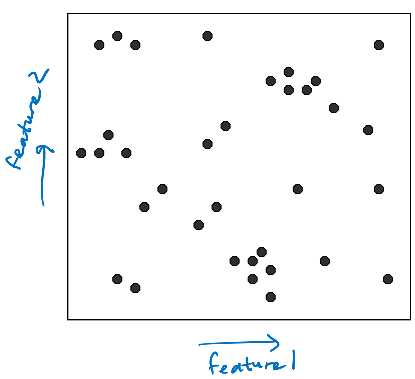


Enables more efficient pruning of search space

Works "well" in "low-medium" dimensions

- We'll get back to this…

KD-tree construction

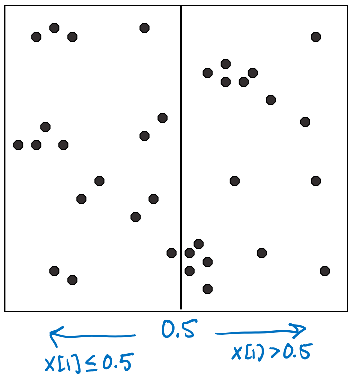


Start with a list of d-dimensional points.

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

Split dimension

Split value

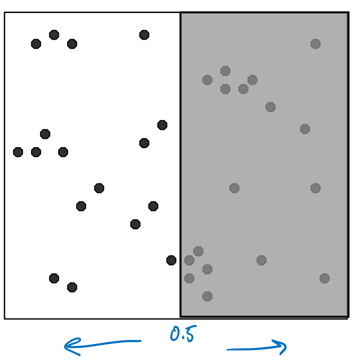
x[1] > .5

NO | YES

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

Split dimension

Split value

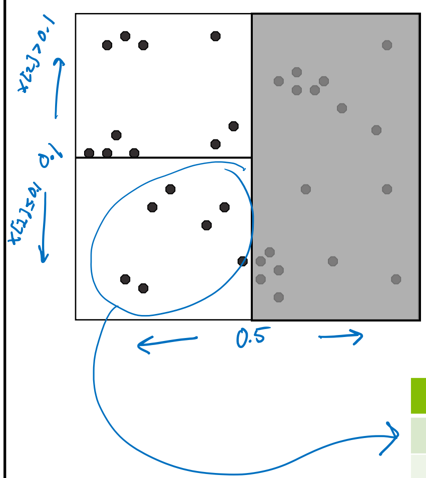
x[1] > .5

NO | YES

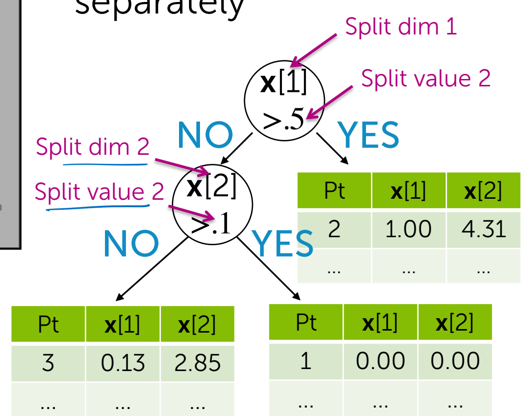
Pt	**x**[1]	**x**[2]
1	0.00	0.00
3	0.13	2.85
…	…	…

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

KD-tree construction



Recurse on each group separately

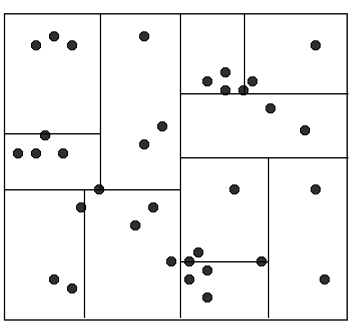


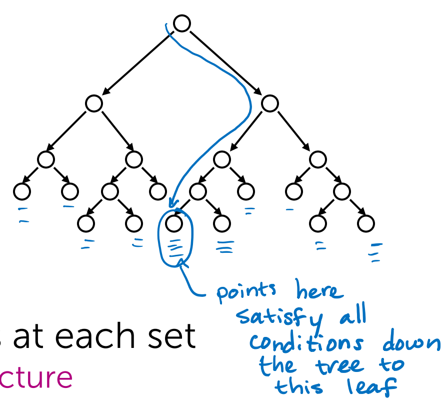
Pt	x[1]	x[2]
2	1.00	4.31
…	…	…

Pt	x[1]	x[2]
3	0.13	2.85
…	…	…

Pt	x[1]	x[2]
1	0.00	0.00
…	…	…

KD-tree construction

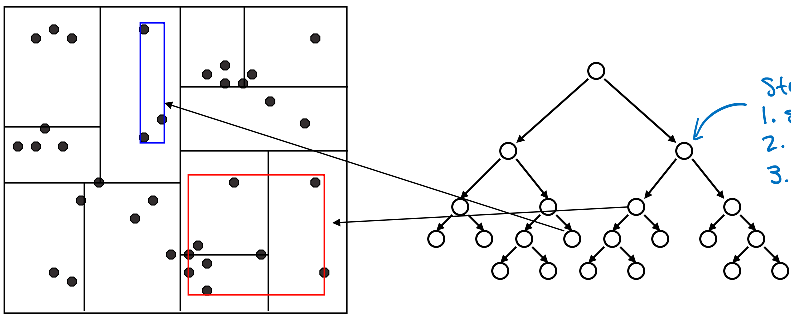




Continue splitting points at each set
- Creates a binary tree structure

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)

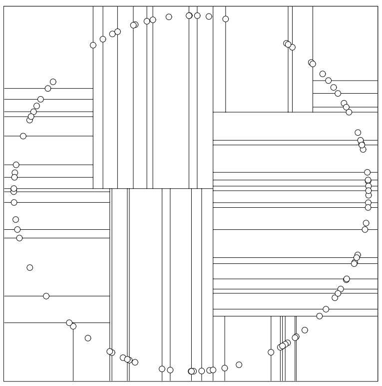
- When do we stop?

  fewer than m pts left

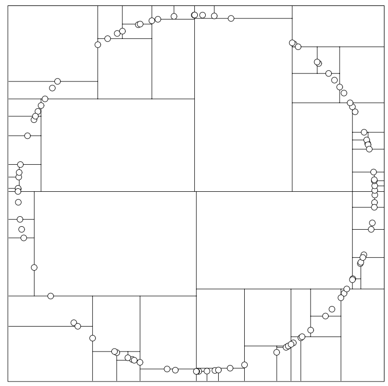
  or

  box hits minimum width

Many heuristics…



median heuristic

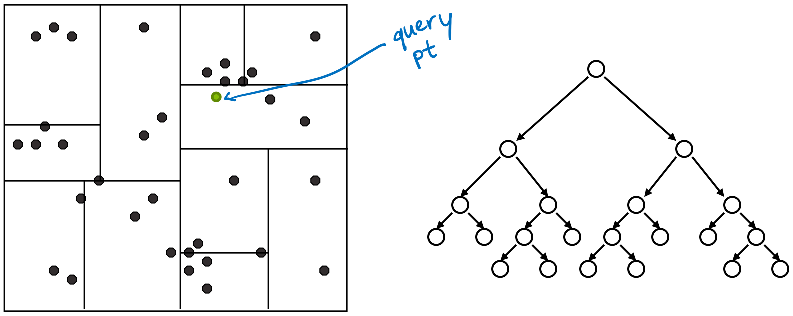


center-of-range heuristic

NN search with KD-trees

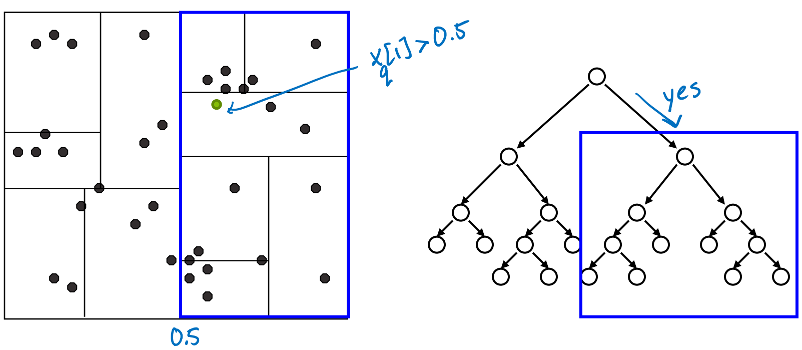
©2018 Emily Fox
STAT/CSE 416: Intro to Machine Learning

Nearest neighbor with KD-trees



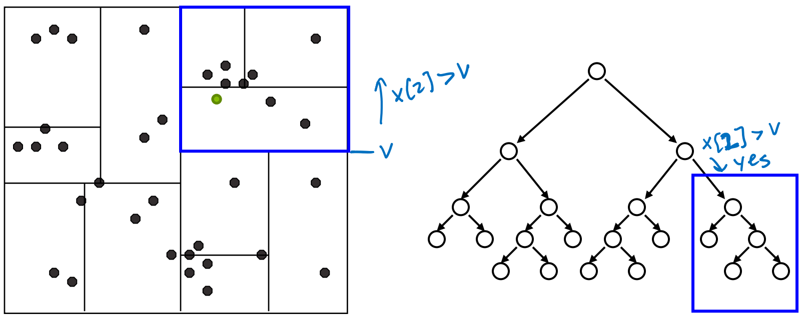
Traverse tree looking for nearest neighbor to query point

Nearest neighbor with KD-trees



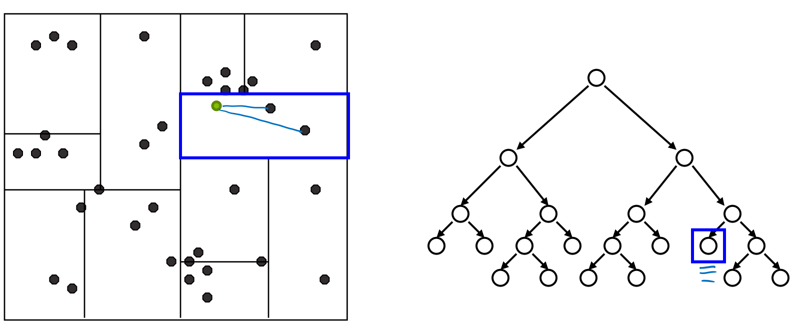
1. Start by exploring leaf node containing query point

Nearest neighbor with KD-trees



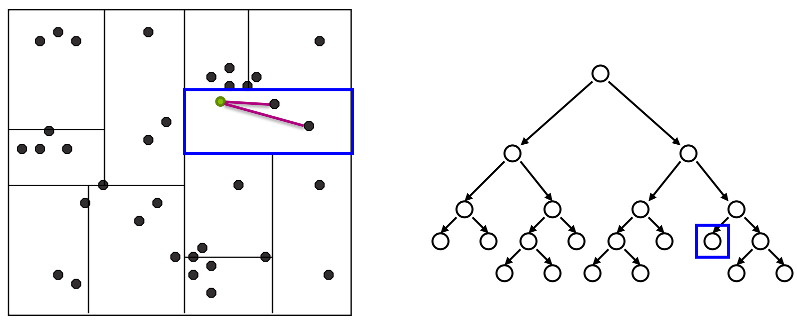
1. Start by exploring leaf node containing query point

Nearest neighbor with KD-trees



1. Start by exploring leaf node containing query point

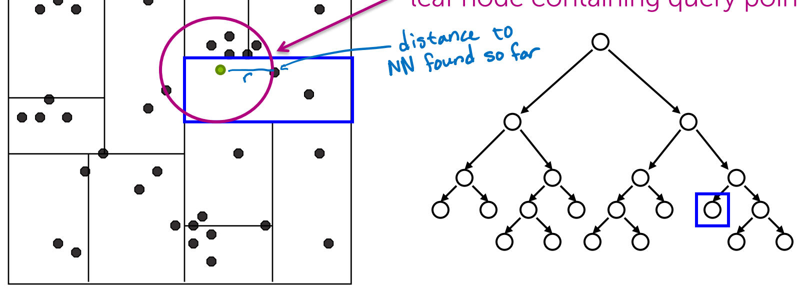
Nearest neighbor with KD-trees



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

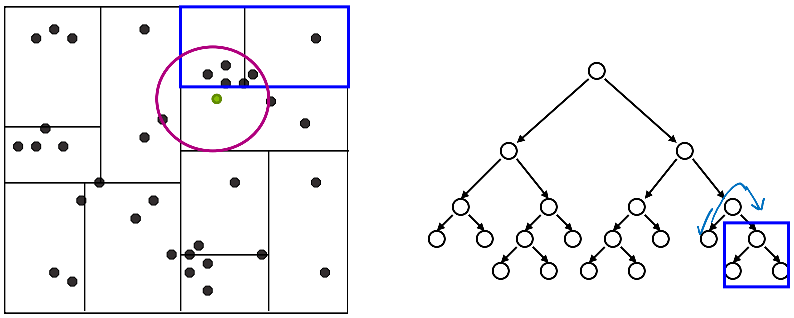
Nearest neighbor with KD-trees

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



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

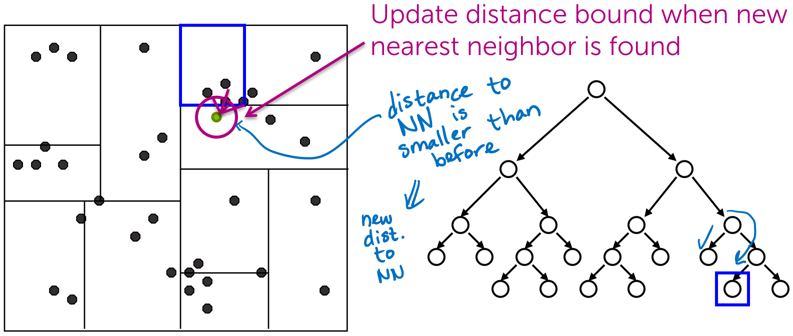
Nearest neighbor with KD-trees



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

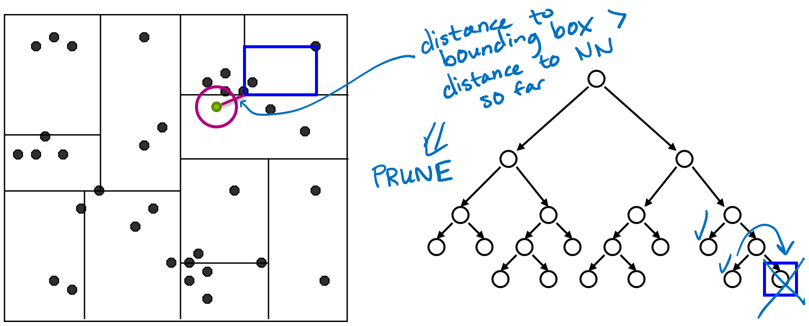
Nearest neighbor with KD-trees

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

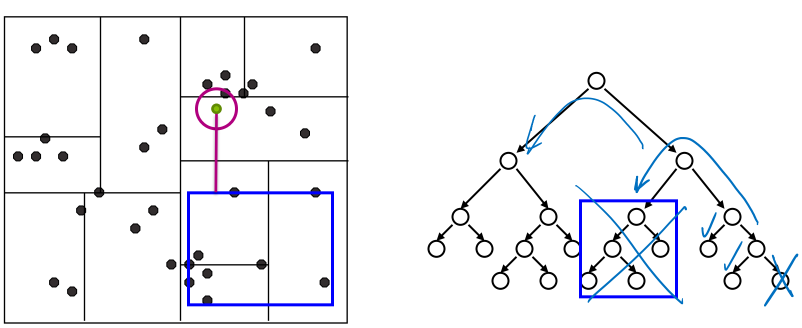
Nearest neighbor with KD-trees



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

37 ©2018 Emily Fox STAT/CSE 416: Intro to Machine Learning

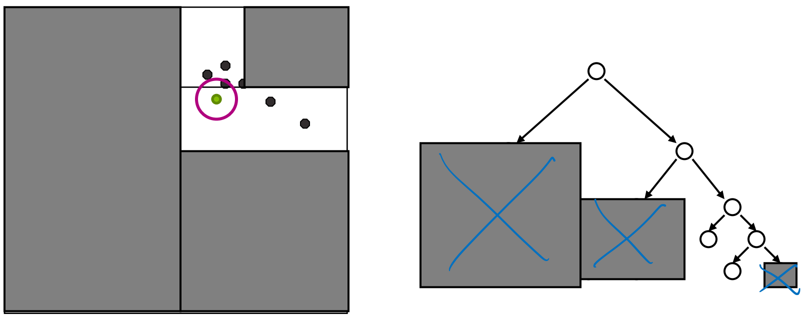
Nearest neighbor with KD-trees



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

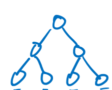
38 ©2018 Emily Fox STAT/CSE 416: Intro to Machine Learning

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

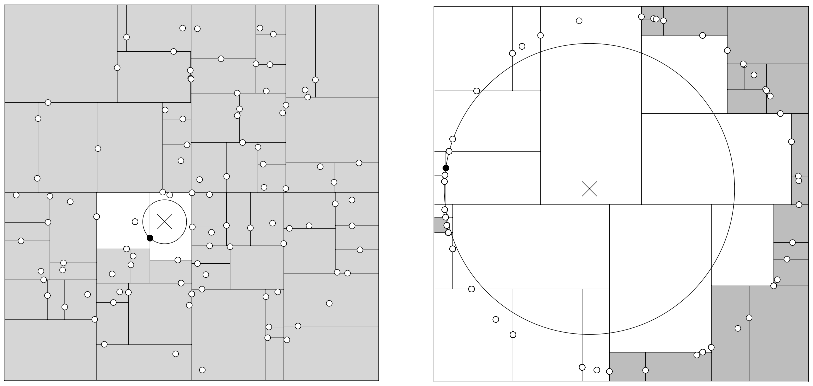


For (nearly) balanced, binary trees…

- 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:

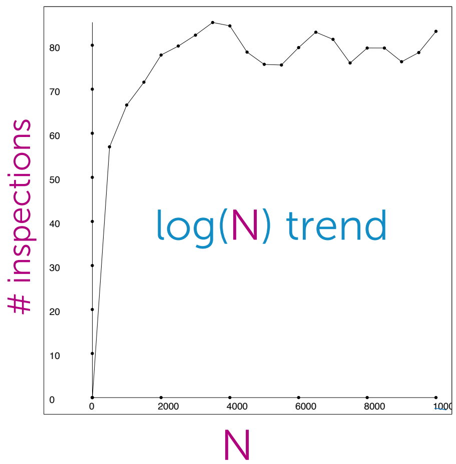
  $O(N^2)$

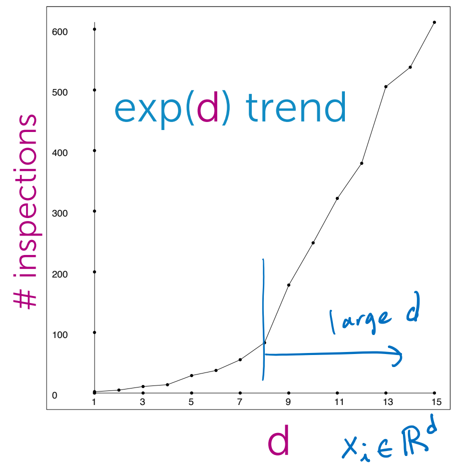
- kd-trees:

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

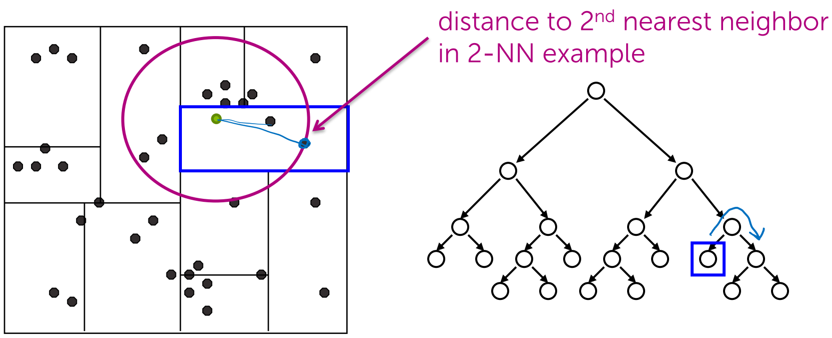
  potentially very large savings for large N!

Inspections vs. N and d





k-NN with KD-trees

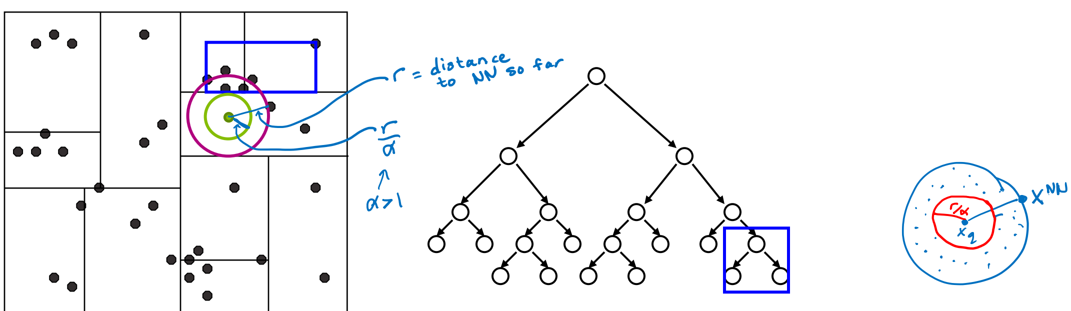


Exactly same algorithm, but maintain distance to furthest of current *k* nearest neighbors

Approximate k-NN search

OPTIONAL

Approximate k-NN with KD-trees



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.

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

Closing remarks on KD-trees

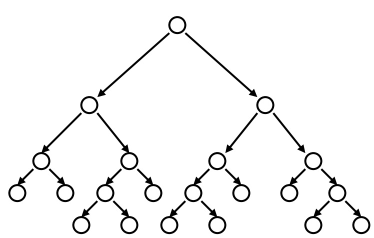
Tons of variants of kd-trees
- On construction of trees
  (heuristics for splitting, stopping, representing branches…)
- Other representational data structures for fast NN search
  (e.g., ball trees,…)

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

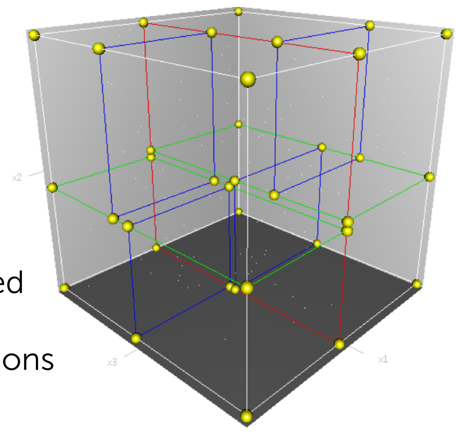
Motivating alternative approaches to approximate NN search

- KD-trees are cool, but…
  - Non-trivial to implement efficiently
  - Problems with high-dimensional data



KD-trees in high dimensions

- Unlikely to have any data points close to query point
- Once "nearby" point is found, the search radius is likely to intersect many hypercubes in at least one dim
- Not many nodes can be pruned
- Can show under some conditions that you visit at least 2^d nodes



Acknowledgements

Thanks to Andrew Moore (http://www.cs.cmu.edu/~awm/) for the KD-trees slide outline