Scalable Nearest Neighbor Algorithms for High Dimensional Data

Marius Muja (UBC), David G. Lowe (Google)

IEEE
2014

Presenter: Derrick Blakely

Department of Computer Science, University of Virginia

https://qdata.github.io/deep2Read/

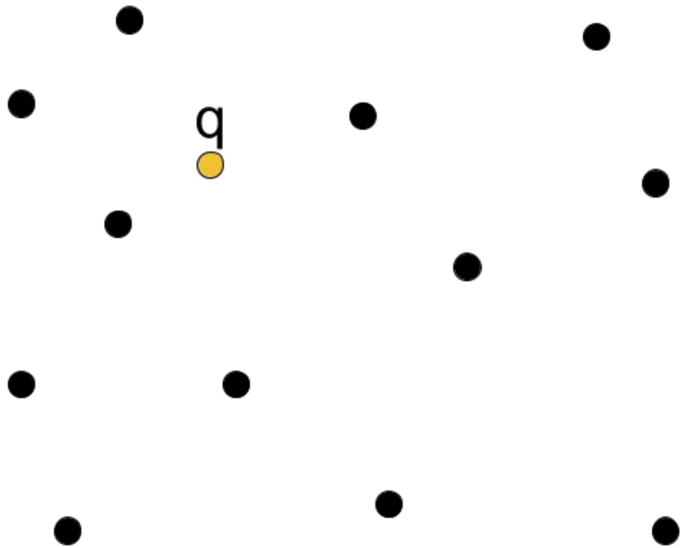
Roadmap

1. Background
2. Motivation
3. Scalable Nearest Neighbor Algorithms for High Dimensional Data
4. Results
5. Conclusion and Take-Aways

Roadmap

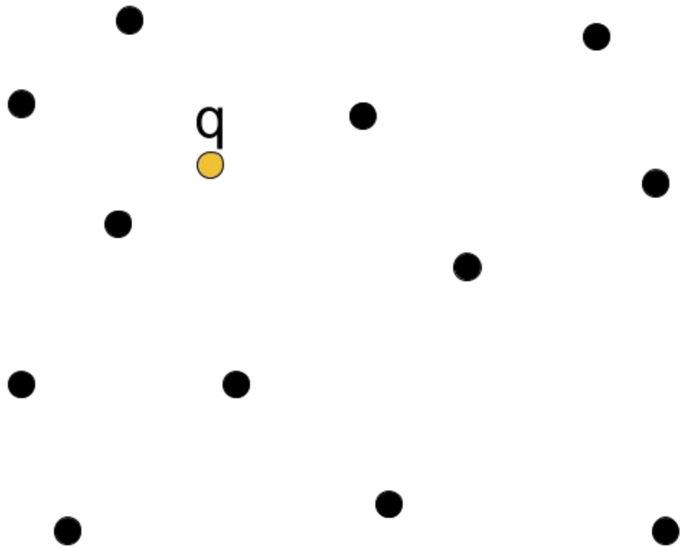
1. Background
2. Motivation
3. Scalable Nearest Neighbor Algorithms for High Dimensional Data
4. Results
5. Conclusion and Take-Aways

Nearest Neighbor Search

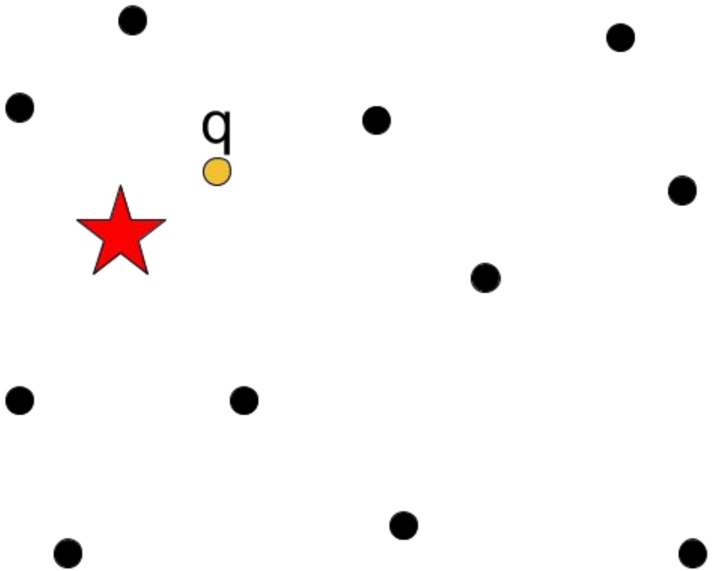


Input: Set of points P, d dimensions,
and query q

Nearest Neighbor Search



Input: Set of points P, d dimensions, and query q



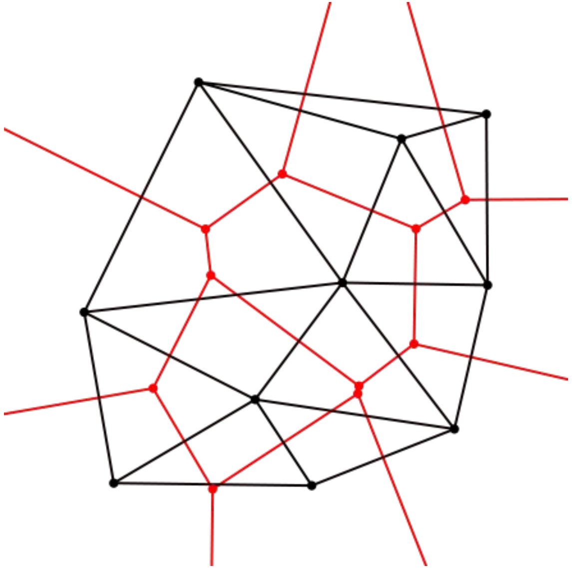
Output: Nearest neighbor p*

Exact Solutions

- Linear search: $O(nd)$, where $n = |P|$

Exact Solutions

- Linear search: O(nd), where n = |P|
- Voronoi diagrams/Delaunay triangulation for d = 2 -- O(logn) querytime



Exact Solutions

- Linear search: O(nd), where n = |P|
- Voronoi diagrams/Delaunay triangulation for d = 2 -- O(logn) querytime
- kd-Trees and other partitioning trees

Exact Solutions

- Linear search: O(nd), where n = |P|
- Voronoi diagrams/Delaunay triangulation for d = 2 -- O(logn) querytime
- kd-Trees and other partitioning trees
    - BSP-trees (binary space partitions)

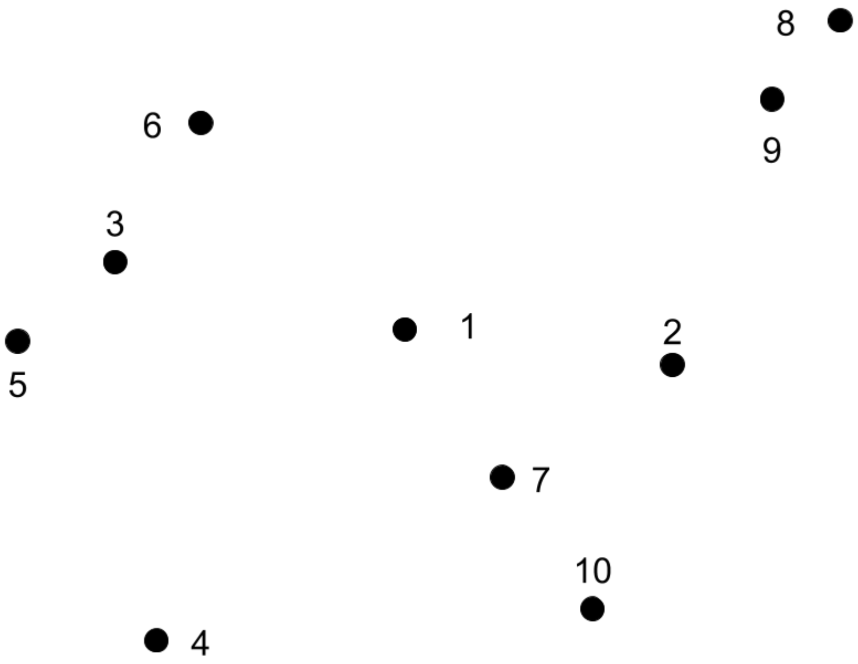
Exact Solutions

- Linear search: O(nd), where n = |P|
- Voronoi diagrams/Delaunay triangulation for d = 2 -- O(logn) querytime
- kd-Trees and other partitioning trees
    - BSP-trees (binary space partitions)
    - R-Trees (overlapping boxes)

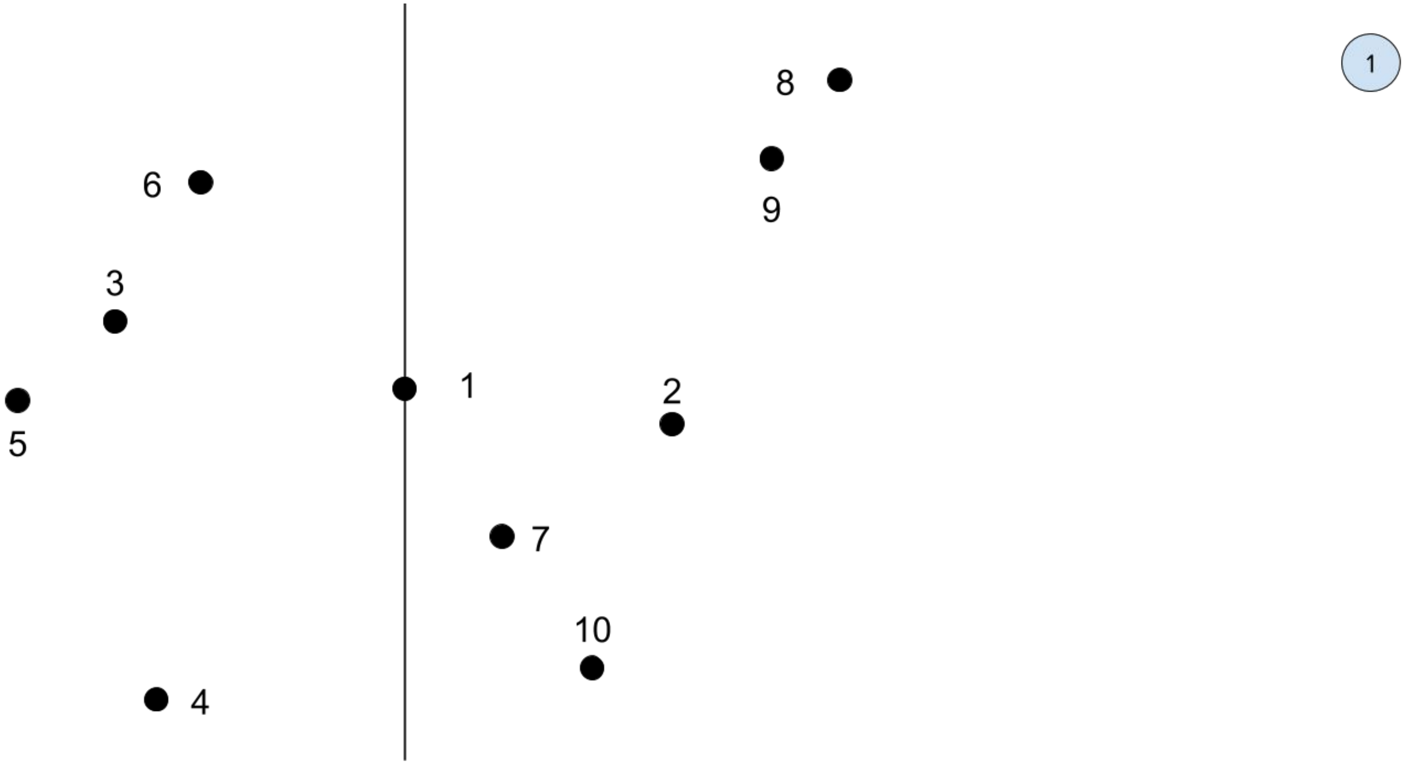
Exact Solutions

- Linear search: O(nd), where n = |P|
- Voronoi diagrams/Delaunay triangulation for d = 2 -- O(logn) querytime
- kd-Trees and other partitioning trees
  - BSP-trees (binary space partitions)
  - R-Trees (overlapping boxes)
  - Ball trees (partition into hyperspheres)

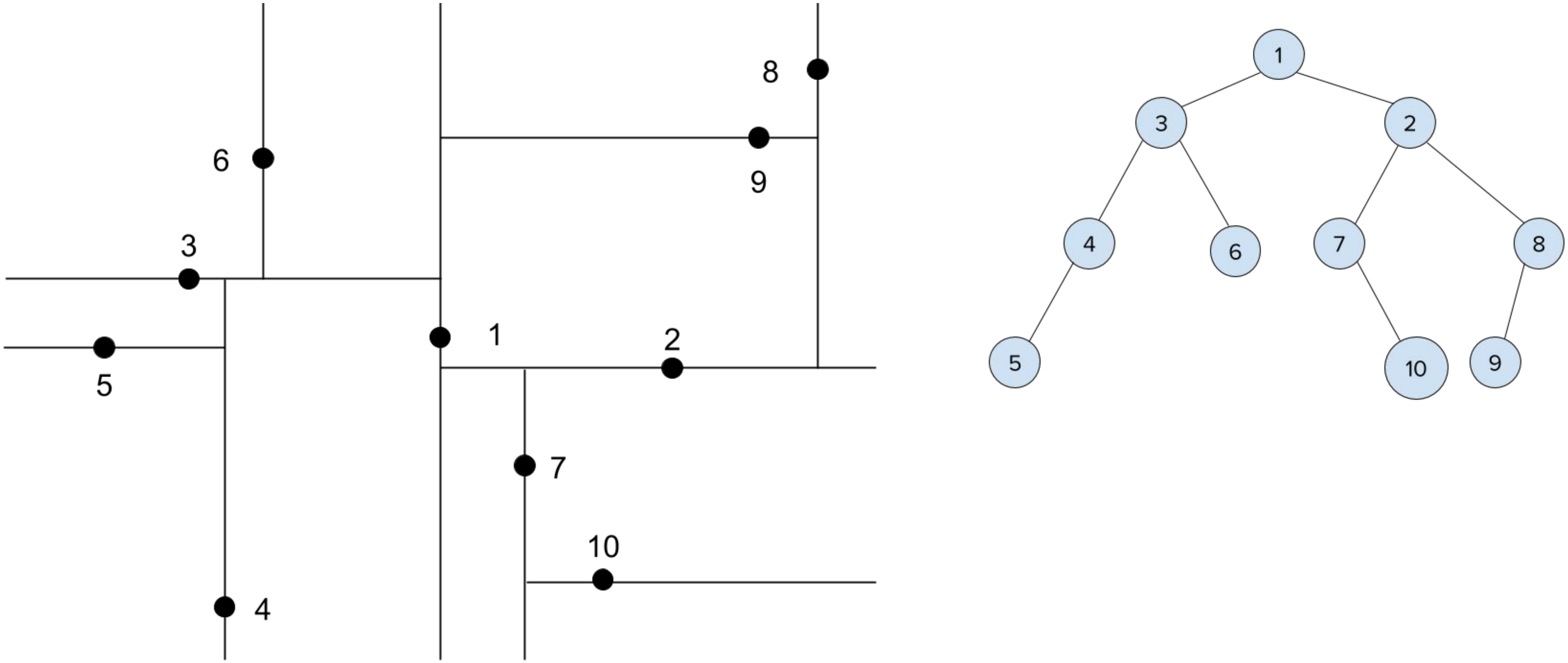
Exact Solutions: kd-Trees



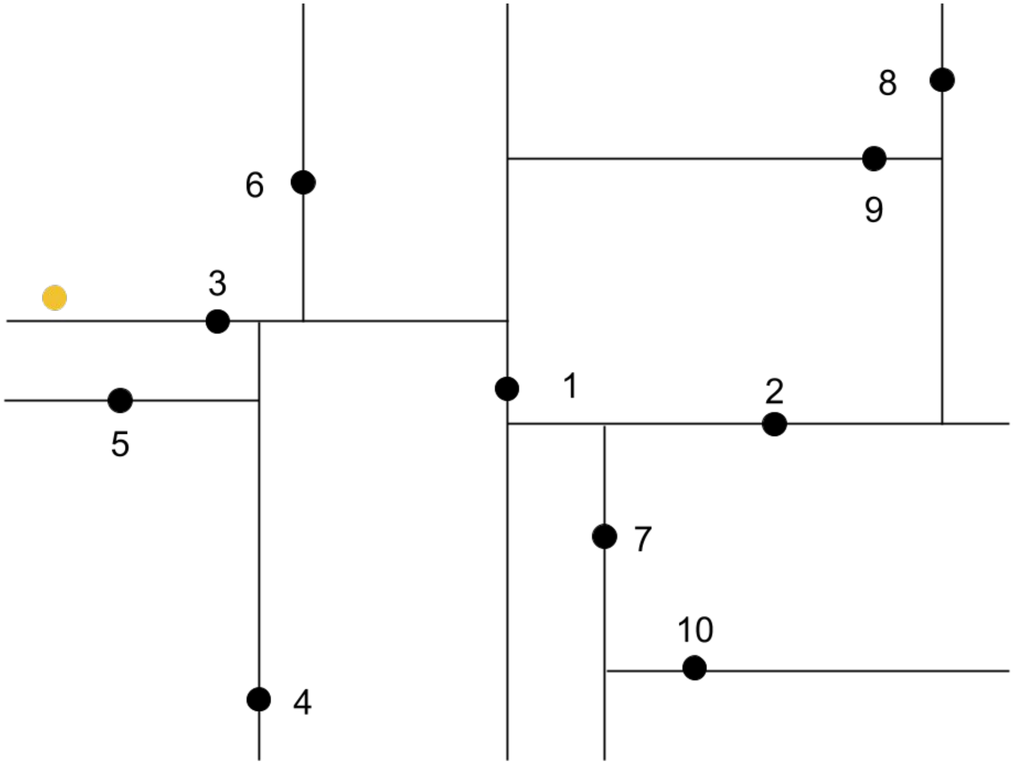
Exact Solutions: kd-Trees

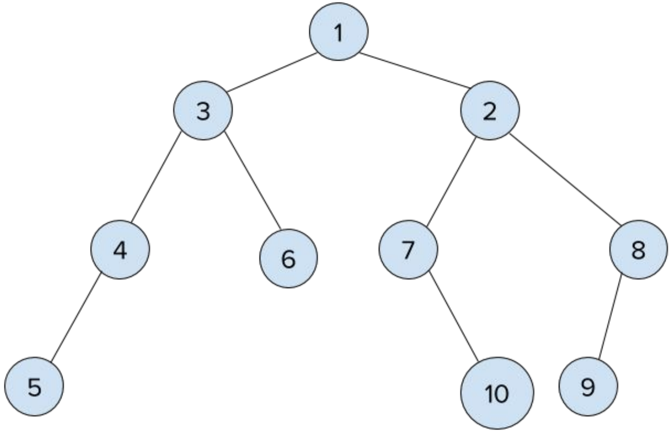


Exact Solutions: kd-Trees



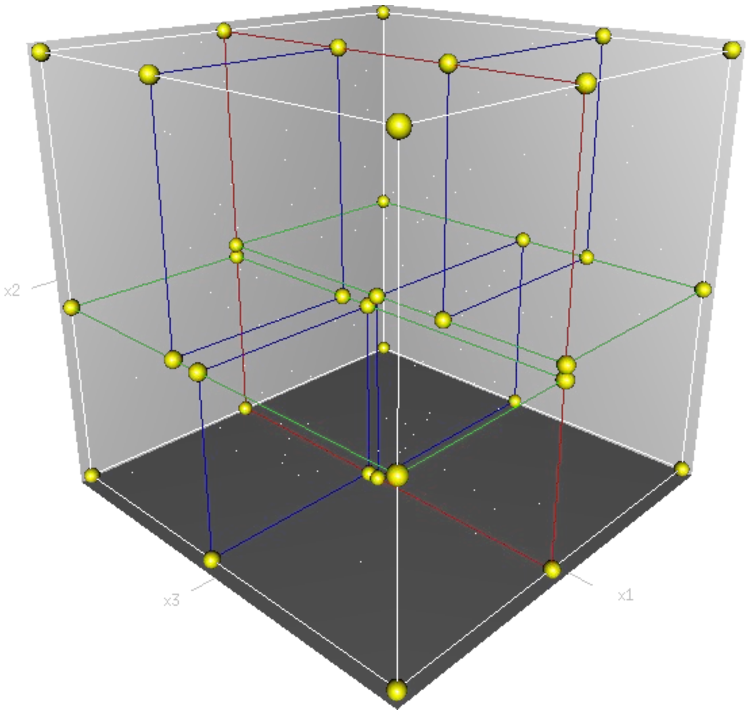
Exact Solutions: kd-Trees





Exact Solutions: kd-Trees

- Linear once roughly d > 20



Approximate Nearest Neighbor (ANN)

1. Partitioning Trees (kd-Trees, etc.)
2. Locality Sensitive Hashing
3. Nearest Neighbor Graphs

Partitioning Trees

- All are different flavors of using a BST for point location

Partitioning Trees

- All are different flavors of using a BST for point location
- Kd-trees, BSP-trees, R-trees, ball trees, etc.

Partitioning Trees

- All are different flavors of using a BST for point location
- Kd-trees, BSP-trees, R-trees, ball trees, etc.
- Randomly perturb query point and return the point whose cell the query lands in

Partitioning Trees

- All are different flavors of using a BST for point location
- Kd-trees, BSP-trees, R-trees, ball trees, etc.
- Randomly perturb query point and return the point whose cell the query lands in
- Randomized forests, with trees searched in parallel

Kd-Tree Random Forests

- Build multiple randomized kd-Trees and search them in parallel

Kd-Tree Random Forests

- Build multiple randomized kd-Trees and search them in parallel
- Splitting dimension sampled from top N dimensions for each remaining subset whenever a split is made

Kd-Tree Random Forests

- Build multiple randomized kd-Trees and search them in parallel
- Splitting dimension sampled from top N dimensions for each remaining subset whenever a split is made
- Single priority queue shared among all trees

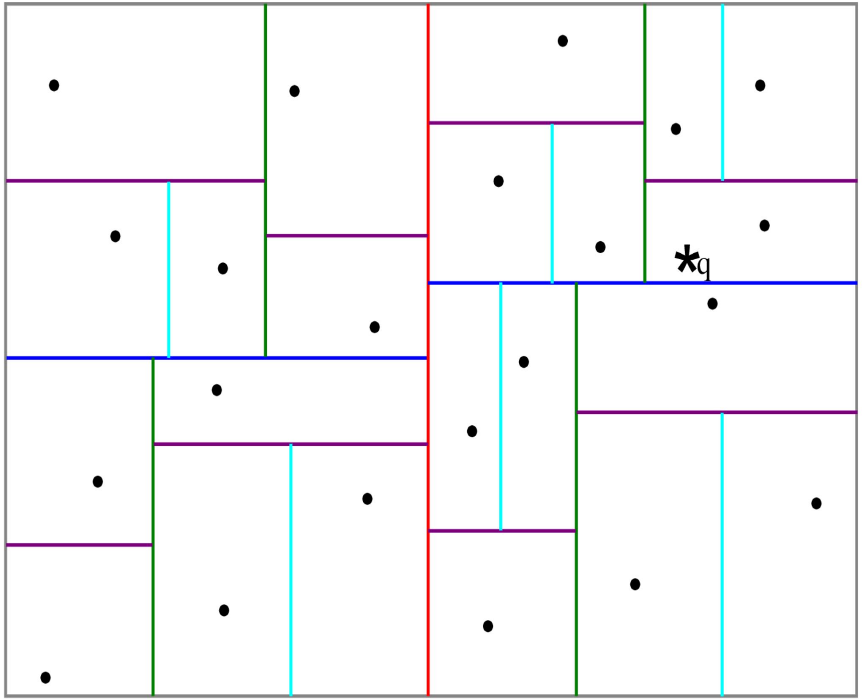
Kd-Tree Random Forests

- Build multiple randomized kd-Trees and search them in parallel
- Splitting dimension sampled from top N dimensions for each remaining subset whenever a split is made
- Single priority queue shared among all trees
- Ordered by increasing distance to decision boundary

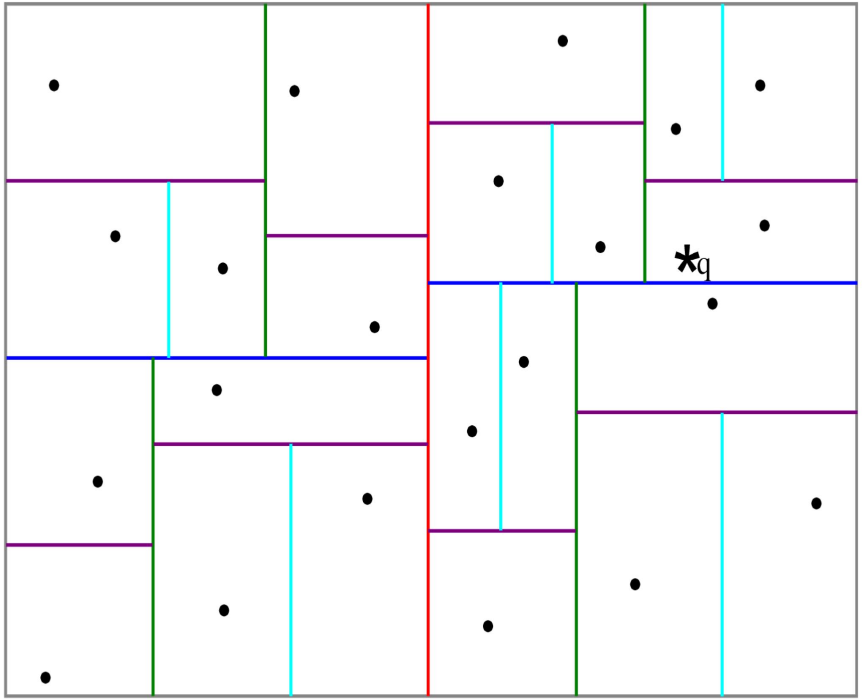
Kd-Tree Random Forests

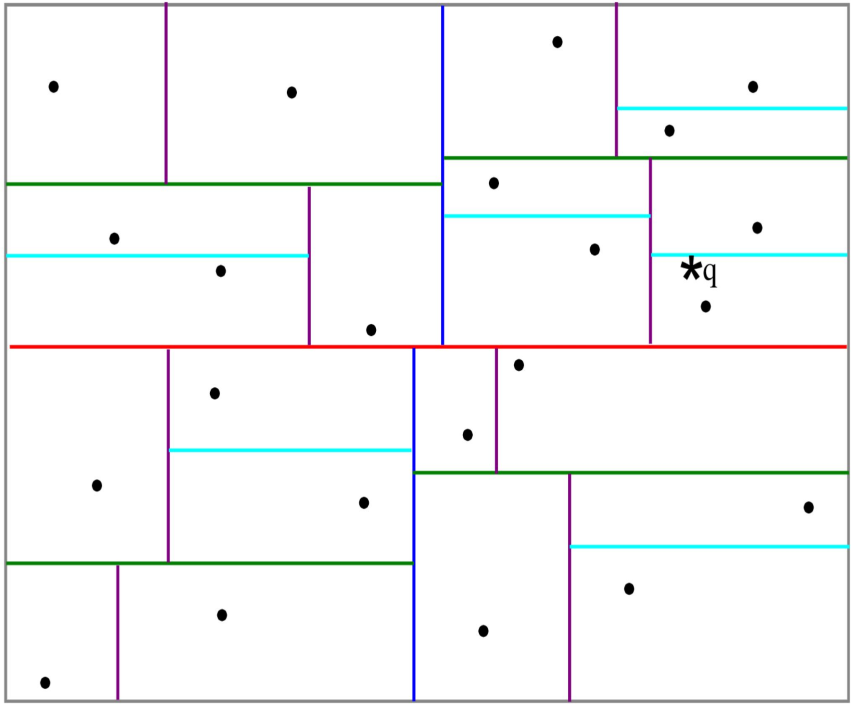
- Build multiple randomized kd-Trees and search them in parallel
- Splitting dimension sampled from top N dimensions for each remaining subset whenever a split is made
- Single priority queue shared among all trees
- Ordered by increasing distance to decision boundary
- Mitigates tendency of tree search to become linear as d increases

kd-Tree Random Forests

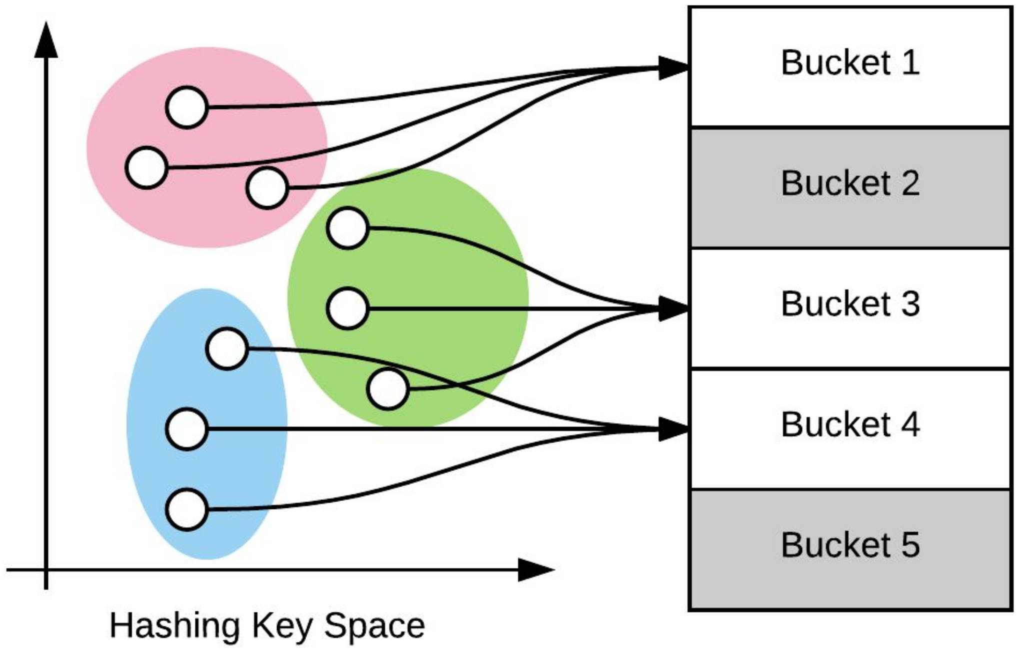


kd-Tree Random Forests

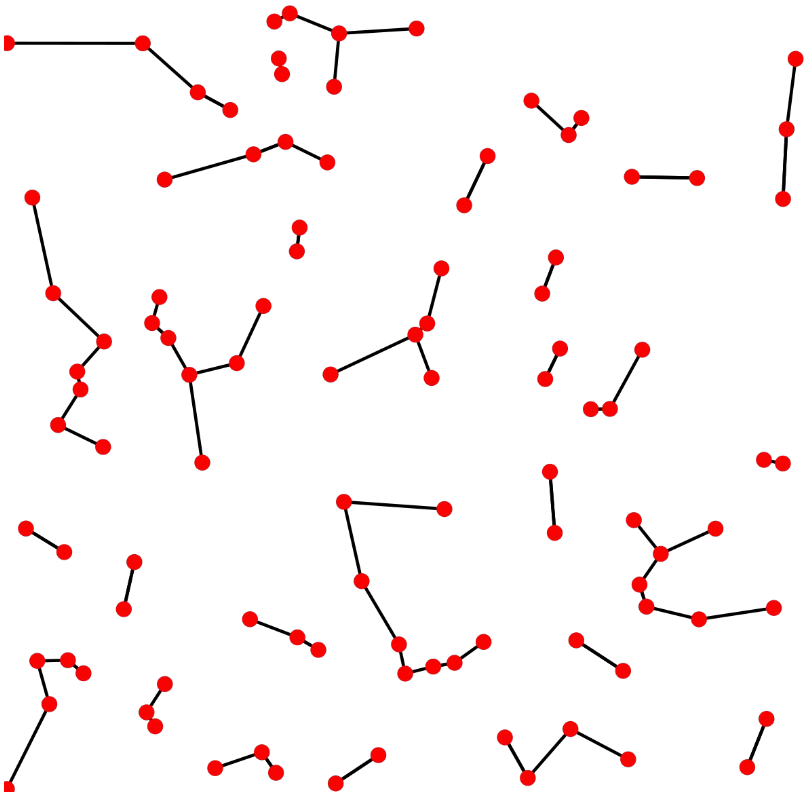




Locality Sensitive Hashing



Nearest Neighbor Graphs



Roadmap

1. Background
2. Motivation
3. Scalable Nearest Neighbor Algorithms for High Dimensional Data
4. Results
5. Conclusion and Take-Aways

Roadmap

1. Background
2. Motivation
3. Scalable Nearest Neighbor Algorithms for High Dimensional Data
4. Results
5. Conclusion and Take-Aways

Motivation

- Lots of different ANN algos

Motivation

- Lots of different ANN algos
- Which ones tend to work the best?

Motivation

- Lots of different ANN algos
- Which ones tend to work the best?
- Introduce a new algo that tends to work well

Motivation

- Lots of different ANN algos
- Which ones tend to work the best?
- Introduce a new algo that tends to work well
- Create an ANN library for C++: FLANN

Motivation

- Lots of different ANN algos
- Which ones tend to work the best?
- Introduce a new algo that tends to work well
- Create an ANN library for C++: FLANN
- Automatic algo selection

Motivation

- Lots of different ANN algos
- Which ones tend to work the best?
- Introduce a new algo that tends to work well
- Create an ANN library for C++: FLANN
- Automatic algo selection
- Distributing ANN with compute clusters and map reduce

Roadmap

1. Background
2. Motivation
3. Scalable Nearest Neighbor Algorithms for High Dimensional Data
4. Results
5. Conclusion and Take-Aways

Roadmap

1. Background
2. Motivation
3. Scalable Nearest Neighbor Algorithms for High Dimensional Data
4. Results
5. Conclusion and Take-Aways

Priority Search K-Means Tree

- Based on K-means clustering

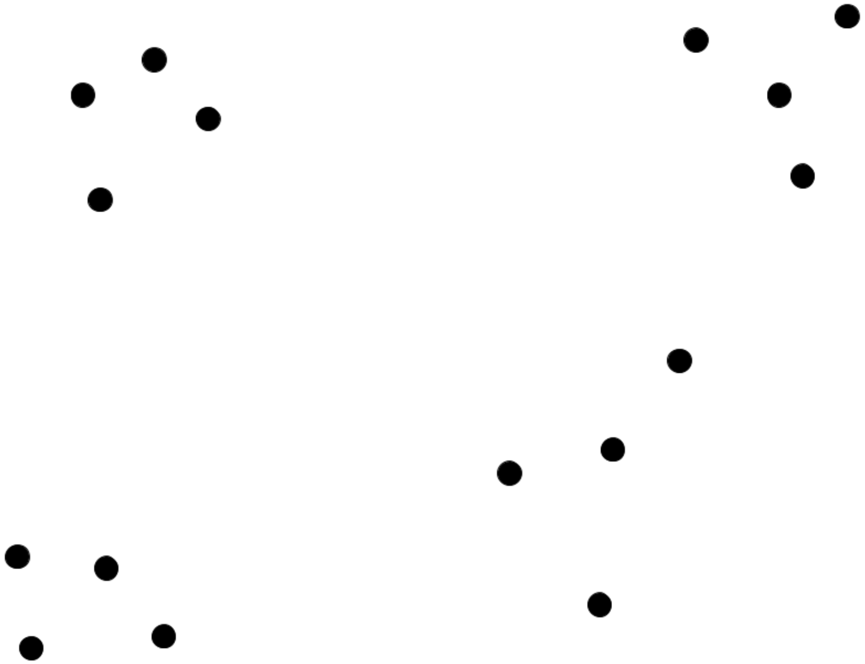
Priority Search K-Means Tree

- Based on K-means clustering
- Uses full distance metric for partitioning, rather than using one dimension at a time

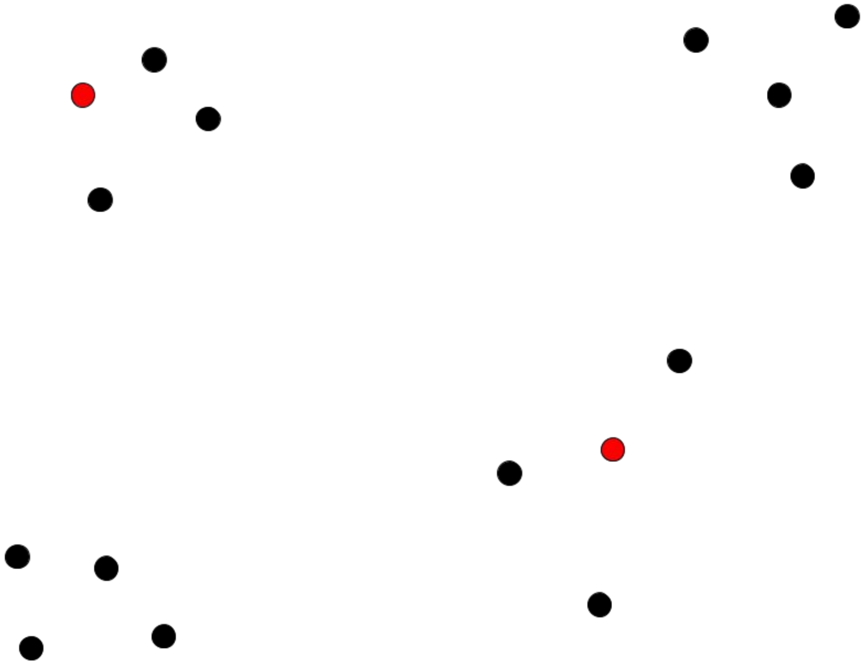
Priority Search K-Means Tree

- Based on K-means clustering
- Uses full distance metric for partitioning, rather than using one dimension at a time
- Goal: better exploit distribution/structure of data points in P

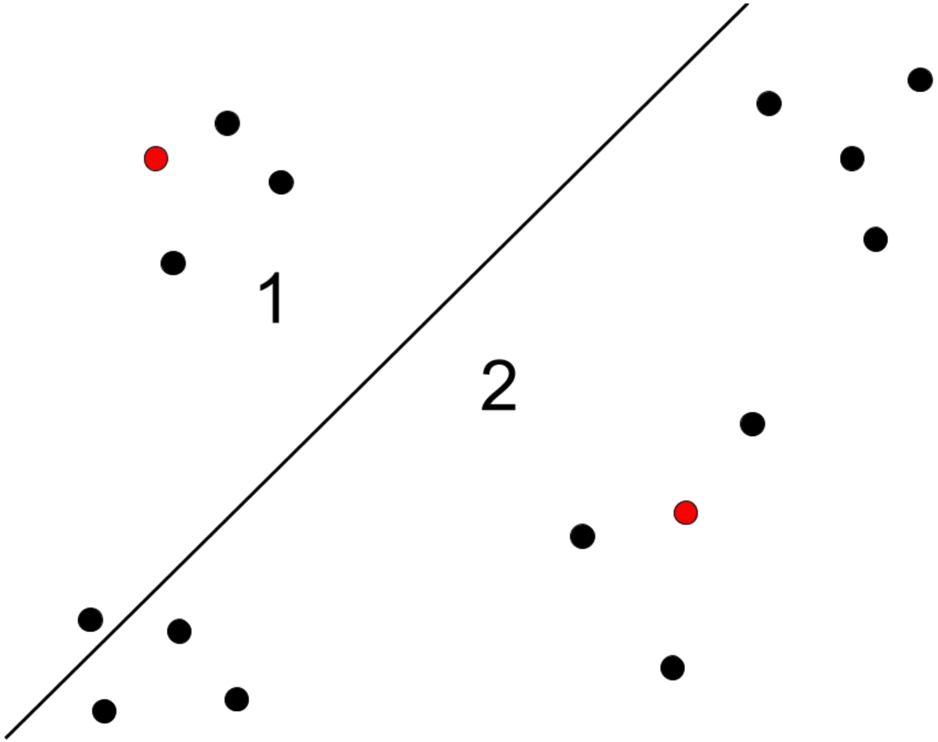
Priority K-means Tree Construction

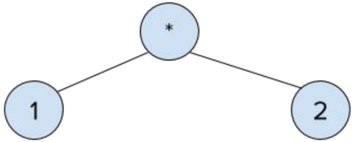


Priority K-means Tree Construction

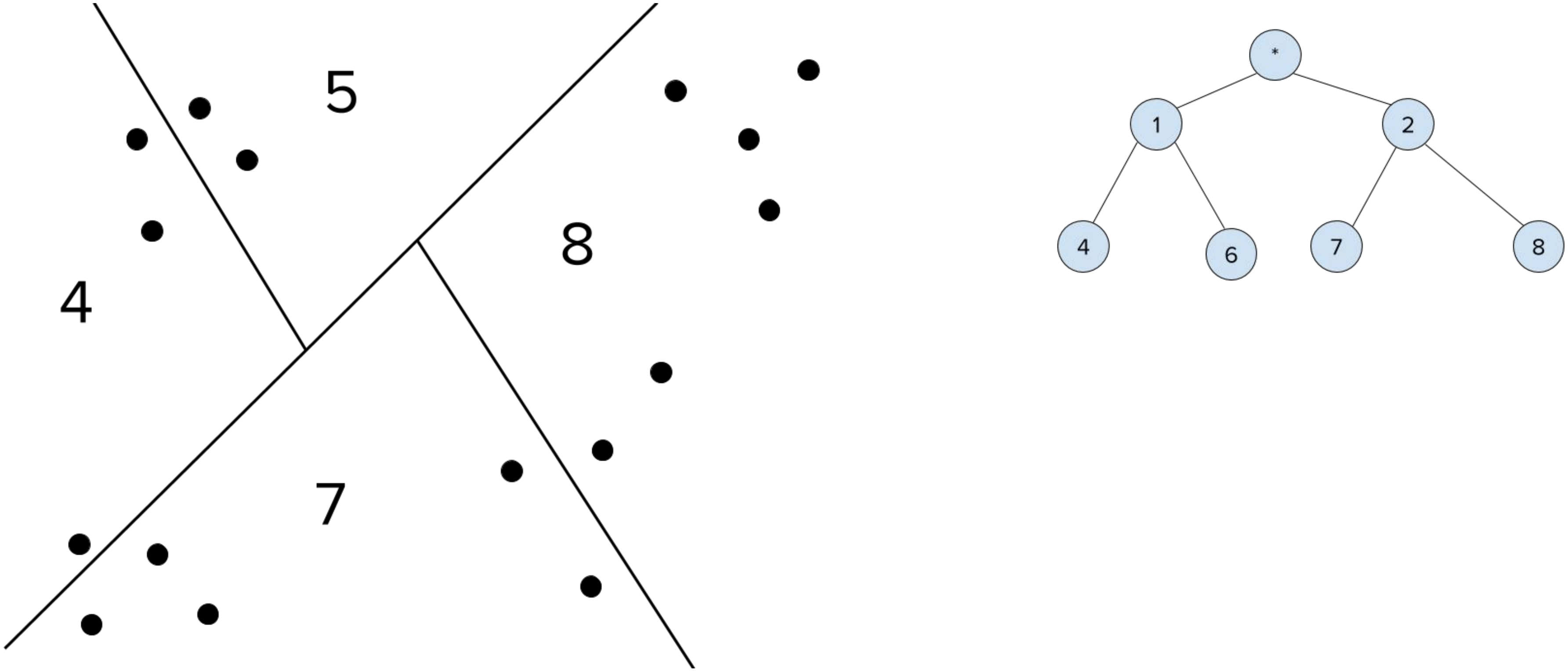


Priority K-means Tree Construction

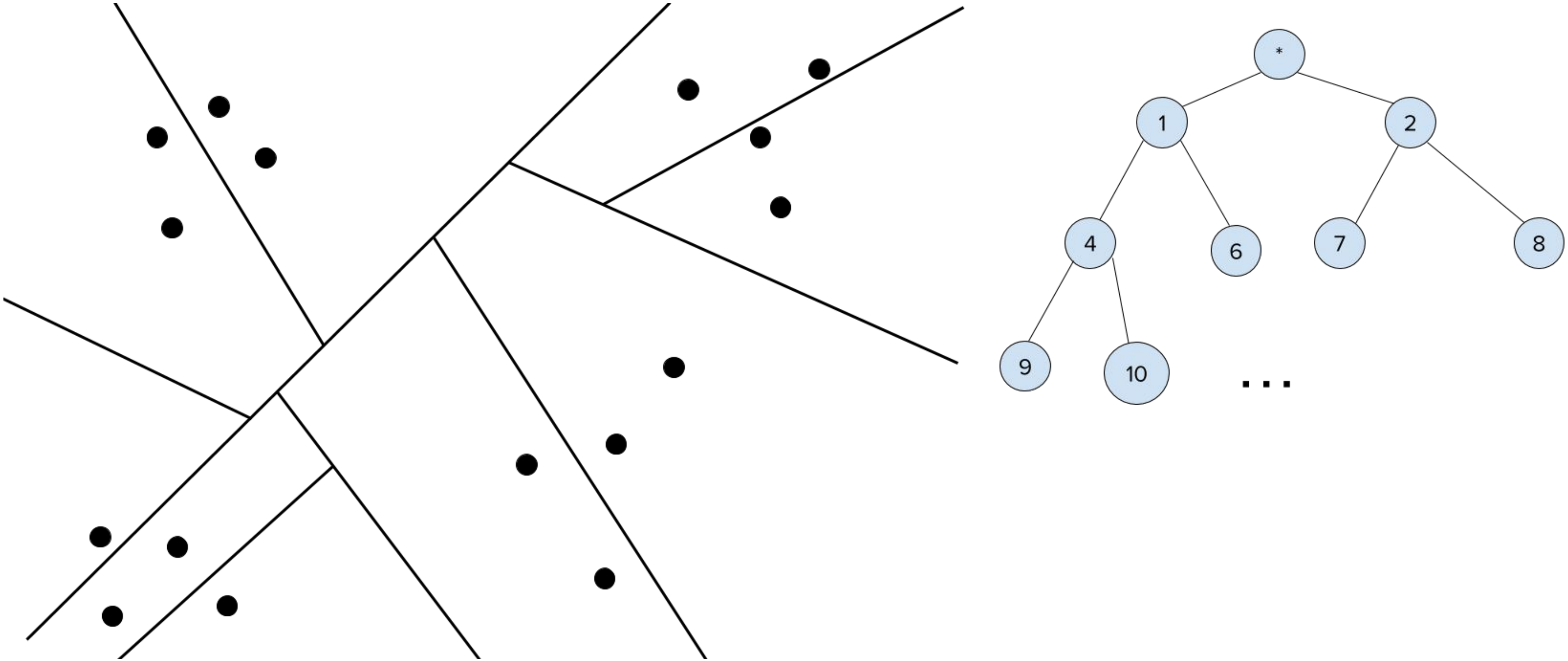




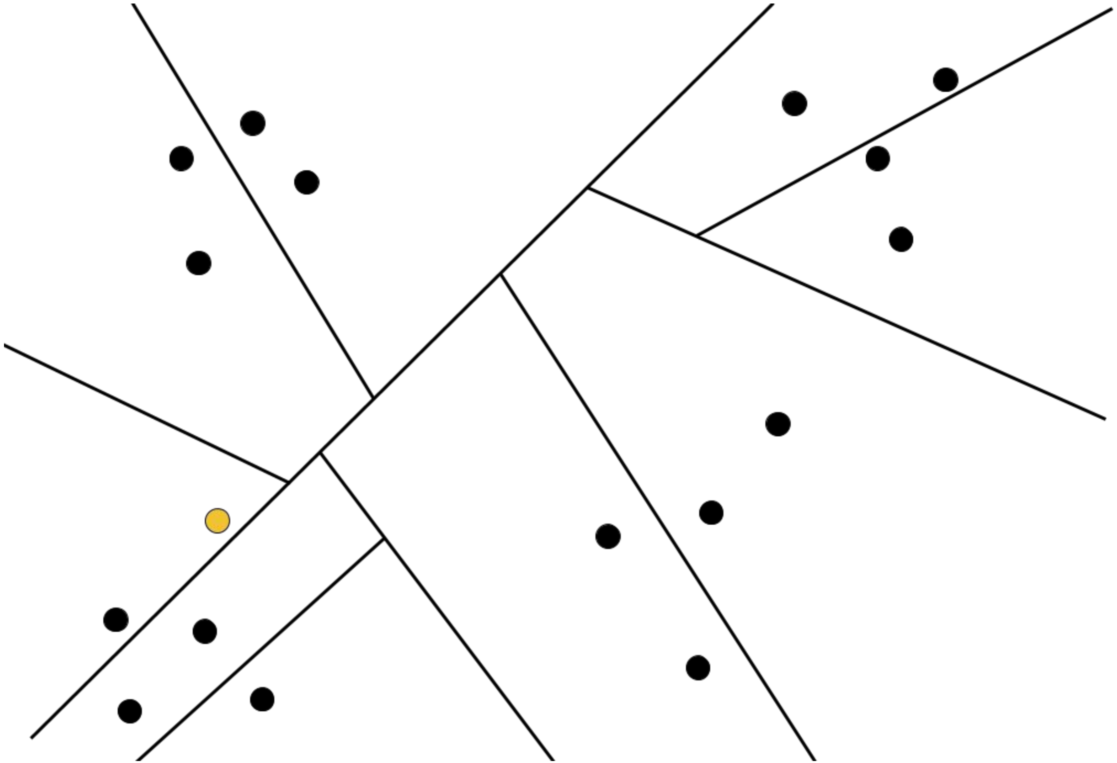
Priority K-means Tree Construction

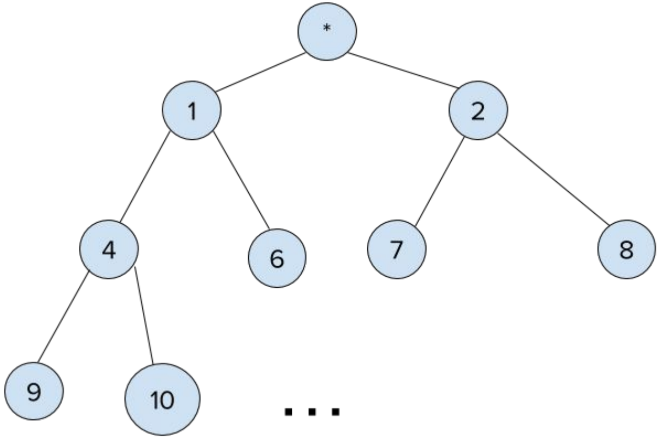


Priority K-means Tree Construction



Priority K-means Tree Querying





Time Complexity

	Construction	Query
Randomized kd-tree	$O(ndK^*\log_k n)$	Expected $O(d\log n)$
Priority k-means tree	Single level: $O(ndKI)$ Total tree: $O(ndKI^*\log_k n)$, where I = max iterations for k-means clustering	At each level, finding closest centroid: $O(Kd)$ Traversal: $O(d\log_k n)$

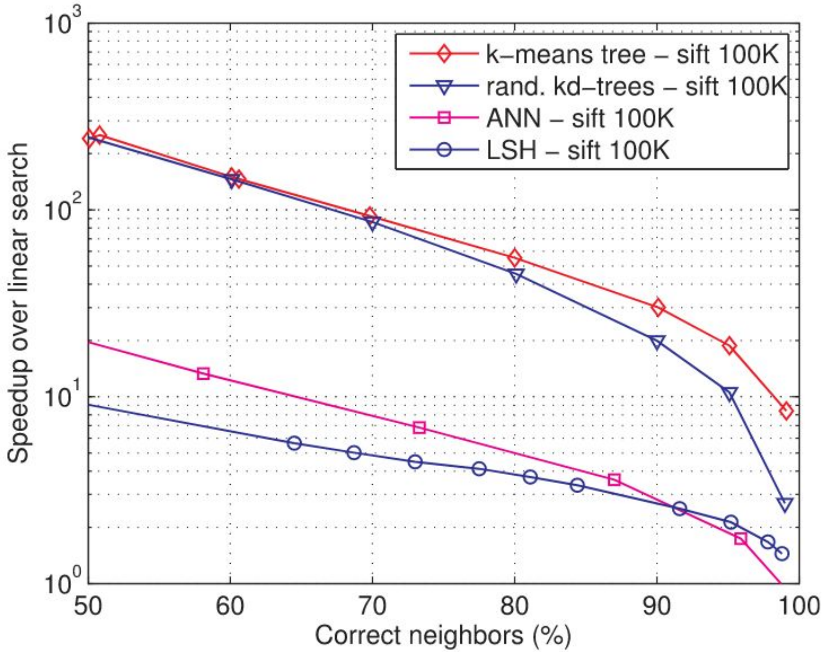
Roadmap

1. Background
2. Motivation
3. Scalable Nearest Neighbor Algorithms for High Dimensional Data
4. Results
5. Conclusion and Take-Aways

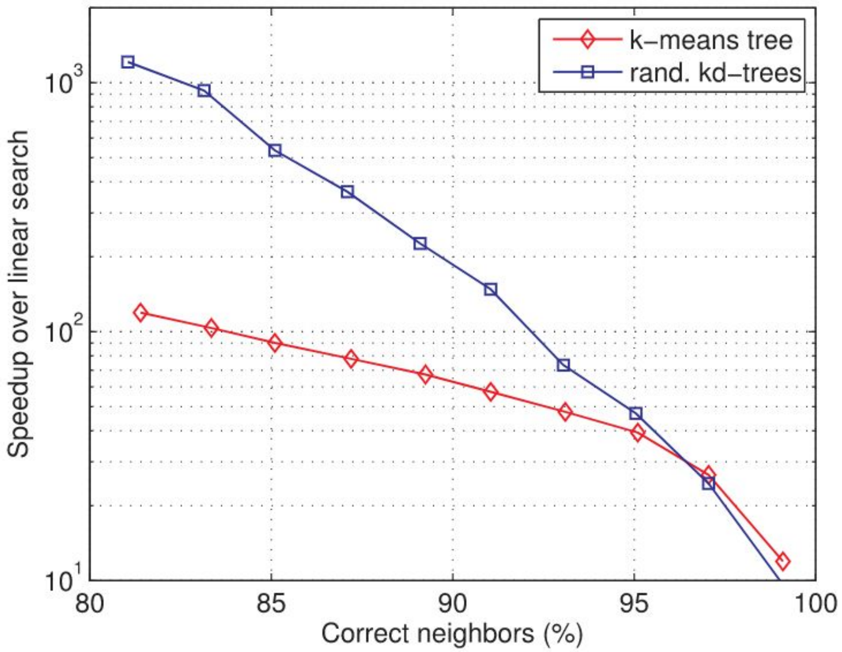
Roadmap

1. Background
2. Motivation
3. Scalable Nearest Neighbor Algorithms for High Dimensional Data
4. Results
5. Conclusion and Take-Aways

Results



k–means tree – sift 100K
rand. kd–trees – sift 100K
ANN – sift 100K
LSH – sift 100K
Speedup over linear search
Correct neighbors (%)
10^0
10^1
10^2
10^3
50
60
70
80
90
100



k–means tree
rand. kd–trees
Speedup over linear search
Correct neighbors (%)
10^1
10^2
10^3
80
85
90
95
100

Roadmap

1. Background
2. Motivation
3. Scalable Nearest Neighbor Algorithms for High Dimensional Data
4. Results
5. Conclusion and Take-Aways

Roadmap

1. Background
2. Motivation
3. Scalable Nearest Neighbor Algorithms for High Dimensional Data
4. Results
5. Conclusion and Take-Aways

Conclusions and Take-Aways

- Different ANN algos work better for different datasets

Conclusions and Take-Aways

- Different ANN algos work better for different datasets
- If data is naturally distributed into clusters ➜ k-means tree

Conclusions and Take-Aways

- Different ANN algos work better for different datasets
- If data is naturally distributed into clusters ➔ k-means tree
- Strong correlations between features ➔ kd-Trees

Conclusions and Take-Aways

- Different ANN algos work better for different datasets
- If data is naturally distributed into clusters ➔ k-means tree
- Strong correlations between features ➔ kd-Trees
- Generally, search tree-based algos are the most scalable

Conclusions and Take-Aways

- Different ANN algos work better for different datasets
- If data is naturally distributed into clusters ➔ k-means tree
- Strong correlations between features ➔ kd-Trees
- Generally, search tree-based algos are the most scalable
- A lot of potential for LSH; difficult obtaining good hash functions

Conclusions and Take-Aways

- Different ANN algos work better for different datasets
- If data is naturally distributed into clusters ➜ k-means tree
- Strong correlations between features ➜ kd-Trees
- Generally, search tree-based algos are the most scalable
- A lot of potential for LSH; difficult obtaining good hash functions
- Large scale ANN is a “big data” problem
    - Want good performance? Distribute with compute clusters and Map Reduce

FLANN

- Fast Library for Approximate Nearest Neighbor
- C++ source code
- Contains implementations of a bunch of nearest neighbor algos
- Included in the OpenCV package
- Very fast
- But has bugs, finicky, hard to use
- Annoy is more popular and is recommended by Radim Rehurek (Gensim creator)

Annoy

- Randomized forest of BSP-Trees
- Instead of splitting subsets based on a particular dimension it:
- Samples two points from the subset
- The boundary is chosen as the hyperplane equidistant from the two points
- Repeat the above k times (hyperparameter for making speed-precision tradeoffs)

What ANN Library Should We Use?

- See Gensim creator Radim Řehůřek's comparison
- FLANN is extremely fast (0.20-0.30 ms/query), but has bugs (reported 4 years ago but still unfixed), and is difficult to use
- Annoy is slower (6-7 ms/query), but much more usable; Radim declares it the “winner”
- LSH implementations (e.g., Yahoo, NearPy) very finicky; problem surrounding finding good hash functions; very low recall (~2 approx nearest neighbors per query)
- Important note: embedding spaces have clusters and
- ⇒ Trying out Annoy for the time being

Spotify