

Approximate Nearest Neighbor (ANN) Search - I

Sanjiv Kumar, Google Research, NY

EECS-6898, Columbia University - Fall, 2010

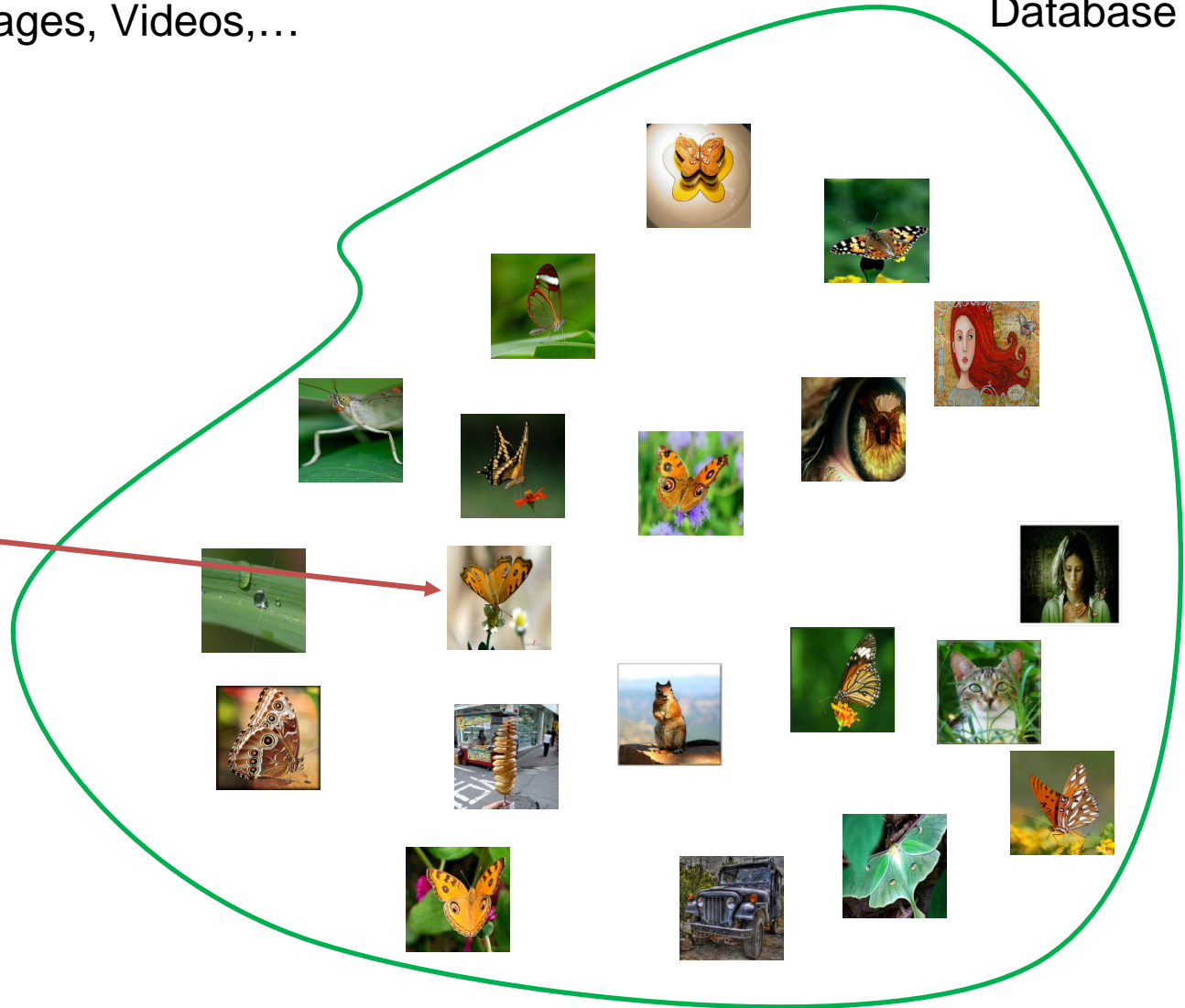
Matching and Retrieval

Documents, Images, Videos,...

Database



query



Matching and Retrieval

Typical Procedure

1. Convert the database items as well as query into vectors in an appropriate **feature space**
2. Define a **distance or similarity measure** for a pair of vectors
3. Find nearest neighbors by **explicit search** over entire database

Approximate !

* Vector space not necessary if similarity can be defined for items directly

Kernel Density Estimation

Given an unlabeled training set, $\{x_i\}_{i=1\dots n}$ learn a nonparametric density function $p(x)$

$$p(x) = \frac{1}{n} \sum_{i=1}^n \underbrace{\frac{1}{h} k\left(\frac{x - x_i}{h}\right)}_{N(x_i, \sigma^2 I)}$$



- Too expensive for large n
- Many kernels have rapid (exponential) fall off
- Nearest Neighbors sufficient but expensive for large n

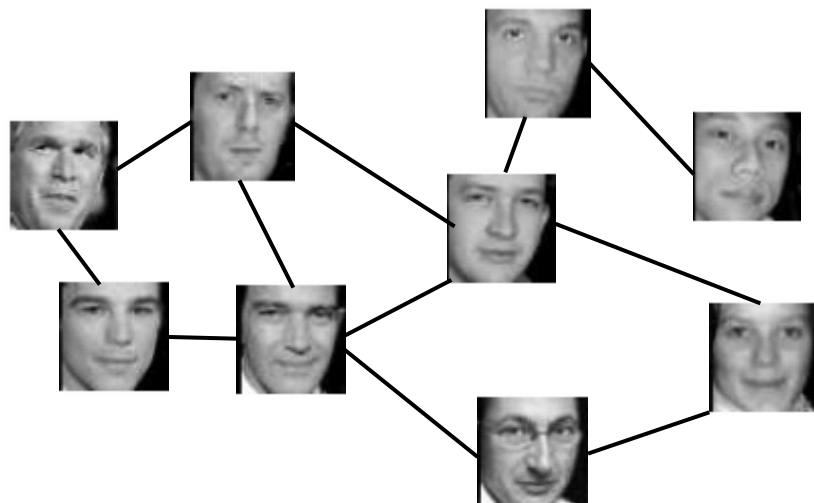
Large-scale Approximate NN search

- Similar arguments for training parametric models e.g., Gaussian Mixture Models with large number of components!

Graph-based methods

Commonly used by many machine learning techniques

- Spectral clustering, manifold learning, semi-supervised learning, ...



Similarity/Weight Matrix
$$W_{ij} = \begin{cases} \exp(-\|x_i - x_j\|^2 / \sigma^2) & \text{if } i \sim j \\ 0 & \text{otherwise} \end{cases}$$

Pairwise NN search problem – $O(n^2)$!

Nearest Neighbor Search

Formal Definition

Given a database $X = \{x_i\}_{i=1\dots n}$ $x_i \in \mathbb{R}^d$ and a query $q \in \mathbb{R}^d$

ε -Neighbor $\hat{x} \in X$ such that $d(q, \hat{x}) \leq \varepsilon$

Nearest Neighbor $\hat{x} \in X$ such that $d(q, \hat{x}) \leq d(q, x) \quad \forall x \in X$ $O(nd)$!

Nearest Neighbor Search

Formal Definition

Given a database $X = \{x_i\}_{i=1\dots n}$ $x_i \in \mathbb{R}^d$ and a query $q \in \mathbb{R}^d$

ε -Neighbor $\hat{x} \in X$ such that $d(q, \hat{x}) \leq \varepsilon$

Nearest Neighbor $\hat{x} \in X$ such that $d(q, \hat{x}) \leq d(q, x) \quad \forall x \in X \quad O(nd)!$

To find nearest neighbor of a query in $n = 1\text{B}$ database with $d = 10\text{K}$: 15 hrs

To build graphs with $n = 10\text{M}$, $d = 10\text{K}$: 155 yrs

Nearest Neighbor Search

Formal Definition

Given a database $X = \{x_i\}_{i=1\dots n}$ $x_i \in \mathbb{R}^d$ and a query $q \in \mathbb{R}^d$

ε -Neighbor $\hat{x} \in X$ such that $d(q, \hat{x}) \leq \varepsilon$

Nearest Neighbor $\hat{x} \in X$ such that $d(q, \hat{x}) \leq d(q, x) \quad \forall x \in X \quad O(nd)!$

To find nearest neighbor of a query in $n = 1\text{B}$ database with $d = 10\text{K}$: 15 hrs

To build graphs with $n = 10\text{M}$, $d = 10\text{K}$: 155 yrs

Speed-up possible

- Distance-preserving dimensionality reduction e.g., randomized projections ! \rightarrow usually not sufficient for large databases
- Fast distance evaluation or reduce the search space

What about storage?

Nearest Neighbor Search

Formal Definition

Given a database $X = \{x_i\}_{i=1\dots n}$ $x_i \in \mathbb{R}^d$ and a query $q \in \mathbb{R}^d$

ε -Neighbor $\hat{x} \in X$ such that $d(q, \hat{x}) \leq \varepsilon$

Nearest Neighbor $\hat{x} \in X$ such that $d(q, \hat{x}) \leq d(q, x) \quad \forall x \in X$ $O(nd)!$

To find nearest neighbor of a query in $n = 1\text{B}$ database with $d = 10\text{K}$: 15 hrs

To build graphs with $n = 10\text{M}$, $d = 10\text{K}$: 155 yrs

Speed-up possible

- Distance-preserving dimensionality reduction e.g., randomized projections ! \rightarrow usually not sufficient for large databases
- Fast distance evaluation or reduce the search space

What about storage?

For $n = 1\text{B}$ database with $d = 10\text{K}$: 40 TB

Nearest Neighbor Search

Formal Definition

Given a database $X = \{x_i\}_{i=1\dots n}$ $x_i \in \mathbb{R}^d$ and a query $q \in \mathbb{R}^d$

ε -Neighbor $\hat{x} \in X$ such that $d(q, \hat{x}) \leq \varepsilon$

Nearest Neighbor $\hat{x} \in X$ such that $d(q, \hat{x}) \leq d(q, x) \quad \forall x \in X \quad O(nd)!$

To find nearest neighbor of a query in $n = 1\text{B}$ database with $d = 10\text{K}$: 15 hrs

To build graphs with $n = 10\text{M}$, $d = 10\text{K}$: 155 yrs

Speed-up possible

- Distance-preserving dimensionality reduction e.g., randomized projections ! \rightarrow usually not sufficient for large databases
- Fast distance evaluation or reduce the search space

What about storage?

For $n = 1\text{B}$ database with $d = 10\text{K}$: 40 TB

Approximate Nearest Neighbor $y \in X$ such that

$$d(q, y) \leq (1 + \varepsilon)d(q, \hat{x})$$

Distance Metrics

Conditions to be a metric

1. Non-negative: $d(x, y) \geq 0$, $d(x, x) = 0$
2. Symmetric: $d(x, y) = d(y, x)$
3. Triangle Inequality: $d(x, y) + d(y, z) \geq d(x, z)$

Distance Metrics

Conditions to be a metric

1. Non-negative: $d(x, y) \geq 0$, $d(x, x) = 0$
2. Symmetric: $d(x, y) = d(y, x)$
3. Triangle Inequality: $d(x, y) + d(y, z) \geq d(x, z)$

Example of a distance measure which is **not** a metric:

KL-divergence $KL(p, q) = \int p(x) \log \frac{p(x)}{q(x)} dx$

“distance” between
two distributions

- **not symmetric**
- **does not satisfy triangle inequality**

Distance Metrics

Conditions to be a metric

1. Non-negative: $d(x, y) \geq 0$, $d(x, x) = 0$
2. Symmetric: $d(x, y) = d(y, x)$
3. Triangle Inequality: $d(x, y) + d(y, z) \geq d(x, z)$

L_p – metric $d(x, y) = \left(\sum_{j=1}^d |x_j - y_j|^p \right)^{1/p}$

Distance Metrics

Conditions to be a metric

1. Non-negative: $d(x, y) \geq 0$, $d(x, x) = 0$
2. Symmetric: $d(x, y) = d(y, x)$
3. Triangle Inequality: $d(x, y) + d(y, z) \geq d(x, z)$

L_p – metric $d(x, y) = \left(\sum_{j=1}^d |x_j - y_j|^p \right)^{1/p}$ What is L_0, L_∞ ?

Distance Metrics

Conditions to be a metric

1. Non-negative: $d(x, y) \geq 0$, $d(x, x) = 0$
2. Symmetric: $d(x, y) = d(y, x)$
3. Triangle Inequality: $d(x, y) + d(y, z) \geq d(x, z)$

L_p – metric $d(x, y) = \left(\sum_{j=1}^d |x_j - y_j|^p \right)^{1/p}$ What is L_0, L_∞ ?

cosine distance

$$\cos(x, y) = x^T y / \|x\| \|y\|$$

If vectors are unit L_2 -norm $L_2(x, y) = 2 - 2\cos(x, y)$

Distance Metrics

Conditions to be a metric

1. Non-negative: $d(x, y) \geq 0$, $d(x, x) = 0$
2. Symmetric: $d(x, y) = d(y, x)$
3. Triangle Inequality: $d(x, y) + d(y, z) \geq d(x, z)$

Mahalanobis distance $d(x, y) = (x - y)^T A (x - y)$

 symmetric positive semi-definite matrix

Distance Metrics

Conditions to be a metric

1. Non-negative: $d(x, y) \geq 0$, $d(x, x) = 0$
2. Symmetric: $d(x, y) = d(y, x)$
3. Triangle Inequality: $d(x, y) + d(y, z) \geq d(x, z)$

Mahalanobis distance $d(x, y) = (x - y)^T A (x - y)$

 symmetric positive semi-definite matrix

$$A = U \Sigma U^T = (U \Sigma^{1/2})(U \Sigma^{1/2})^T = BB^T$$

$$\tilde{x} = B^T x \implies d(x, y) = (\tilde{x} - \tilde{y})^T (\tilde{x} - \tilde{y})$$

Equivalent to L_2 distance in linearly transformed space

Learning Distance Metric

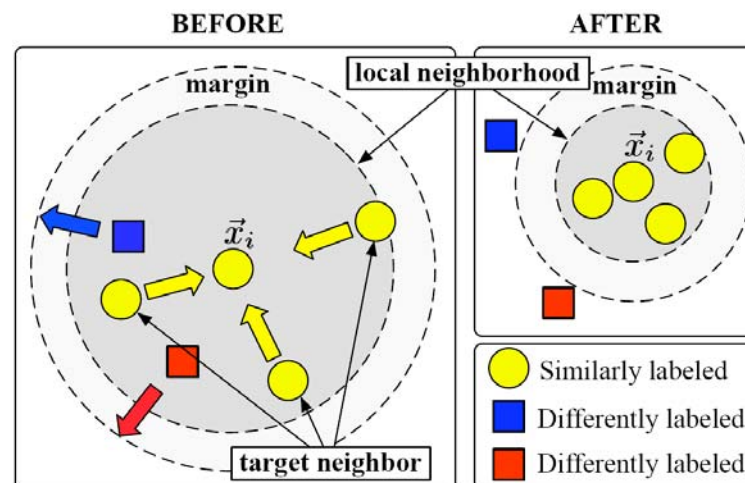


Given a large collection of items and associated features, learn a distance metric!

Learning Distance Metric

Many applications

- kNN-classification, clustering, density estimation, graph construction



$$\min \sum_{(x_i, x_j) \in S} d_A(x_i - x_j)$$

$$d_A(x_i - x_k) - d_A(x_i - x_j) \geq 1 \quad \forall (i, j, k)$$

$$A \succ 0$$

Now onwards, focus on ANN techniques

[Weinberger K. et al.]

Two popular ANN approaches

Tree approaches

- Recursively partition the data: **Divide and Conquer**
- Expected query time: $O(\log n)$
- Many variants: KD tree, Ball tree, PCA-tree, Vantage Point tree...
- Shown to perform very well in relatively low-dim data

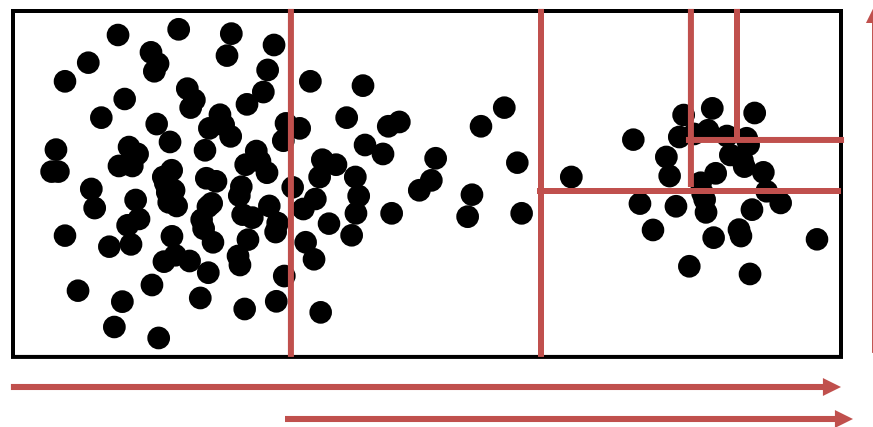
Hashing approaches

- Each image in database represented as a code
- Significant reduction in storage
- Expected query time: $O(1)$ or $O(n)$
- Compact codes preferred

KD-Tree

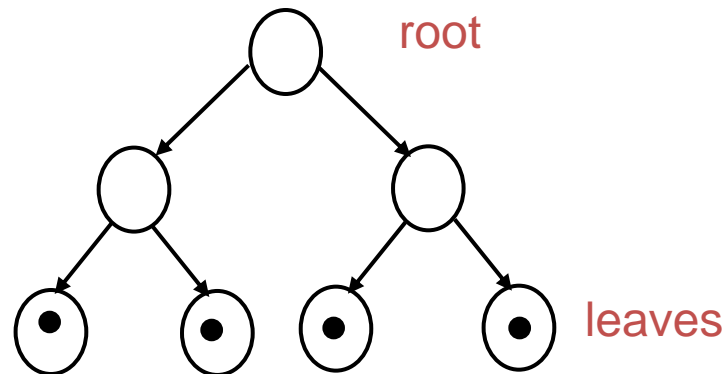
Building K-Dimensional Tree

- Axis-parallel splits
- Find the dimension of **largest variance** (remove outliers)
- Binary partitioning: Split the data along **median** → balanced partitioning
- Split recursively until each node has only one point (leaves)



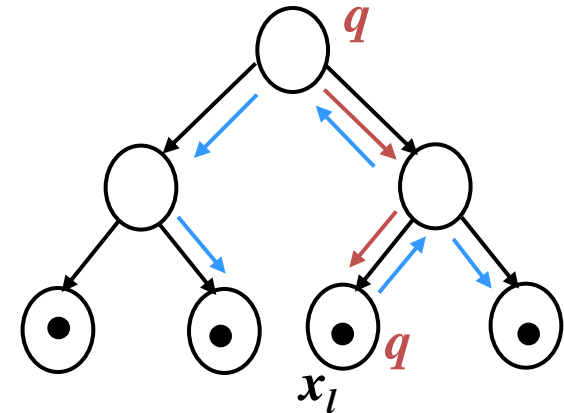
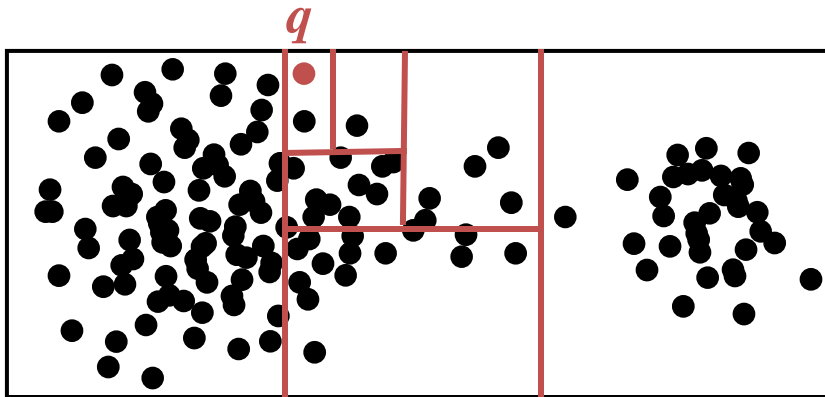
KD-Tree: Properties

- Binary tree of depth $O(\log(n))$
- Total nodes: $(2n-1)$ ($n-1$ internal and n leaves)
- Construction time: $O(nd \log(n))$
- Memory: nonleaf node – (dim, threshold), leaf node – data id
- Need to store the original data also



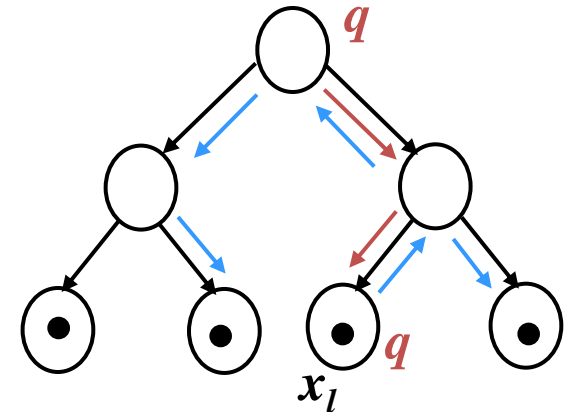
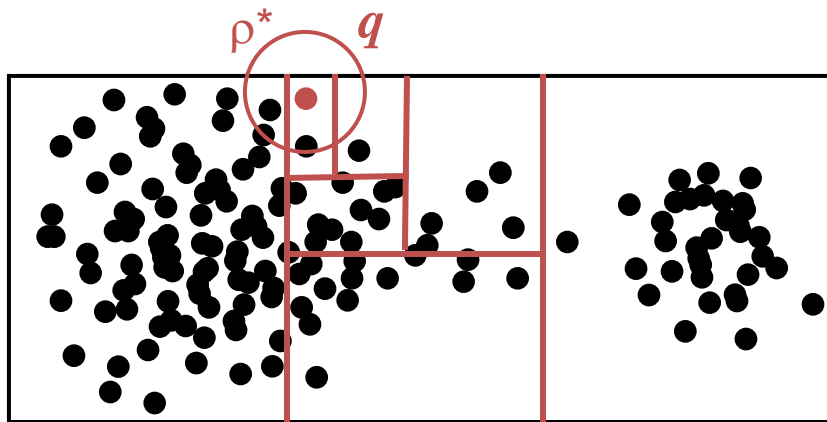
KD-Tree: Search

- Given a query q , push down the tree to a leaf: $O(\log(n))$
- Backtracking: search all potential leaves that may contain NN of q
- Maintain the nearest neighbor and min distance seen so far
- Branch-and-bound to check if leaves under a node may have smaller distance than seen so far



KD-Tree: Branch-and-Bound Verification

- In the beginning, best guess of NN: x_l and $\rho^* = d(q, x_l)$, $x^* = x_l$
- Draw a ball of radius ρ^* around q and see which hyper-rectangles are intersected by the ball
- If during search, $\rho_t = d(q, x_t) < \rho^*$, $\rho^* = \rho_t$, $x^* = x_t$
- Easy to modify search for k-nearest neighbors

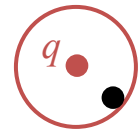


Order in which nodes are searched: Last-Bin-First or Best-Bin-First?
How many nodes do we need to look at during backtracking?

KD-Tree: Search complexity

- Suppose $X = \{x_i\}_{i=1\dots n}$ $x_i \sim p(x)$, *i.i.d.*
- Let q be a query, and $S(q, k)$ be the smallest ball centered at q containing exactly k nearest neighbors

Want to find expected number of leaves intersected by ball



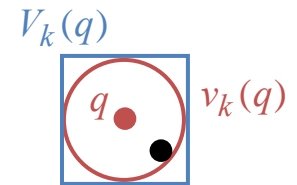
KD-Tree: Search complexity

- Suppose $X = \{x_i\}_{i=1\dots n}$ $x_i \sim p(x)$, *i.i.d.*
- Let q be a query, and $S(q, k)$ be the smallest ball centered at q containing exactly k nearest neighbors

Want to find expected number of leaves intersected by ball

Volume of the ball $v_k(q) = \int_{S(q,k)} dx$

Volume of hypercube containing the ball $V_k(q) = G(d)v_k(q)$



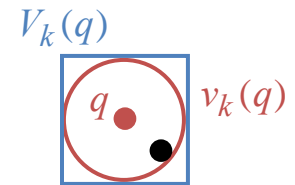
KD-Tree: Search complexity

- Suppose $X = \{x_i\}_{i=1\dots n}$ $x_i \sim p(x)$, *i.i.d.*
- Let q be a query, and $S(q, k)$ be the smallest ball centered at q containing exactly k nearest neighbors

Want to find expected number of leaves intersected by ball

Volume of the ball $v_k(q) = \int_{S(q,k)} dx$

Volume of hypercube $V_k(q) = \underbrace{G(d)}_{\text{increases with } d} v_k(q)$



KD-Tree: Search complexity

- Suppose $X = \{x_i\}_{i=1\dots n}$ $x_i \sim p(x)$, *i.i.d.*
- Let q be a query, and $S(q, k)$ be the smallest ball centered at q containing exactly k nearest neighbors

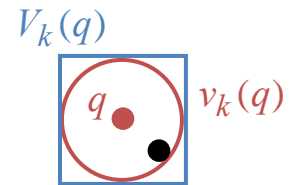
Want to find expected number of leaves intersected by ball

Volume of the ball $v_k(q) = \int_{S(q,k)} dx$

Volume of hypercube containing the ball $V_k(q) = \underbrace{G(d)}_{\text{increases with } d} v_k(q)$

increases with d

Expected Volume $E[V_k(q)] = G(d)E[v_k(q)]$
 $= kG(d)/(n+1)p(q)$



Assuming constant density in ball, and
 $E[\pi_k(q)] = k/(n+1)$

↑
Probability mass in ball

KD-Tree: Search complexity

- Suppose $X = \{x_i\}_{i=1\dots n}$ $x_i \sim p(x)$, *i.i.d.*
- Let q be a query, and $S(q, k)$ be the smallest ball centered at q containing exactly k nearest neighbors

Want to find expected number of leaves intersected by ball

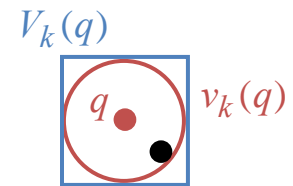
Volume of the ball $v_k(q) = \int_{S(q,k)} dx$

Volume of hypercube $V_k(q) = \underbrace{G(d)}_{\text{increases with } d} v_k(q)$
containing the ball

increases with d

Expected Volume $E[V_k(q)] = G(d)E[v_k(q)]$
 $= kG(d)/(n+1)p(q)$

Expected edge-length $= (kG(d)/(n+1)p(q))^{1/d}$



Assuming constant density in ball, and
 $E[\pi_k(q)] = k/(n+1)$

↑
Probability mass in ball

KD-Tree: Search complexity

Expected edge-length of hypercube containing the k-NN ball around query

$$= (kG(d)/(n+1)p(q))^{1/d}$$

Assuming leaf hyper-rectangle to be a hypercube, its expected edge length

$$= (1/(n+1)p(x_l))^{1/d}$$

KD-Tree: Search complexity

Expected edge-length of hypercube containing the k-NN ball around query

$$= (kG(d)/(n+1)p(q))^{1/d}$$

Assuming leaf hyper-rectangle to be a hypercube, its expected edge length

$$= (1/(n+1)p(x_l))^{1/d}$$

Suppose, $p(q) \approx p(x_l)$

Expected number of leaves overlapping the hypercube around query

$$= \left[(kG(d))^{1/d} + 1 \right]^d$$

KD-Tree: Search complexity

Expected edge-length of hypercube containing the k-NN ball around query

$$= (kG(d)/(n+1)p(q))^{1/d}$$

Assuming leaf hyper-rectangle to be a hypercube, its expected edge length

$$= (1/(n+1)p(x_l))^{1/d}$$

Suppose, $p(q) \approx p(x_l)$

Expected number of leaves overlapping the hypercube around query

$$= \left[(kG(d))^{1/d} + 1 \right]^d$$

For L_∞ , $G(d) = 1$, and suppose $k = 1 \Rightarrow 2^d$ upper bound !

KD-Tree: Search complexity

Expected edge-length of hypercube containing the k-NN ball around query

$$= (kG(d)/(n+1)p(q))^{1/d}$$

Assuming leaf hyper-rectangle to be a hypercube, its expected edge length

$$= (1/(n+1)p(x_l))^{1/d}$$

Suppose, $p(q) \approx p(x_l)$

Expected number of leaves overlapping the hypercube around query

$$= \left[(kG(d))^{1/d} + 1 \right]^d$$

For L_∞ , $G(d) = 1$, and suppose $k = 1 \Rightarrow 2^d$ upper bound !

- $G(d)$ increases as p decreases \rightarrow above number is lower bound for other L_p metrics
- As d increases, the number of leaves to search grows exponentially
- Very conservative estimate, empirically one needs to test with given data
- As a rule-of-thumb, KD-trees work fine up to 15-20 dims, (have been shown to work well up to 100-150 dim))

Randomized KD-Trees

Performance of a single KD-tree is usually low

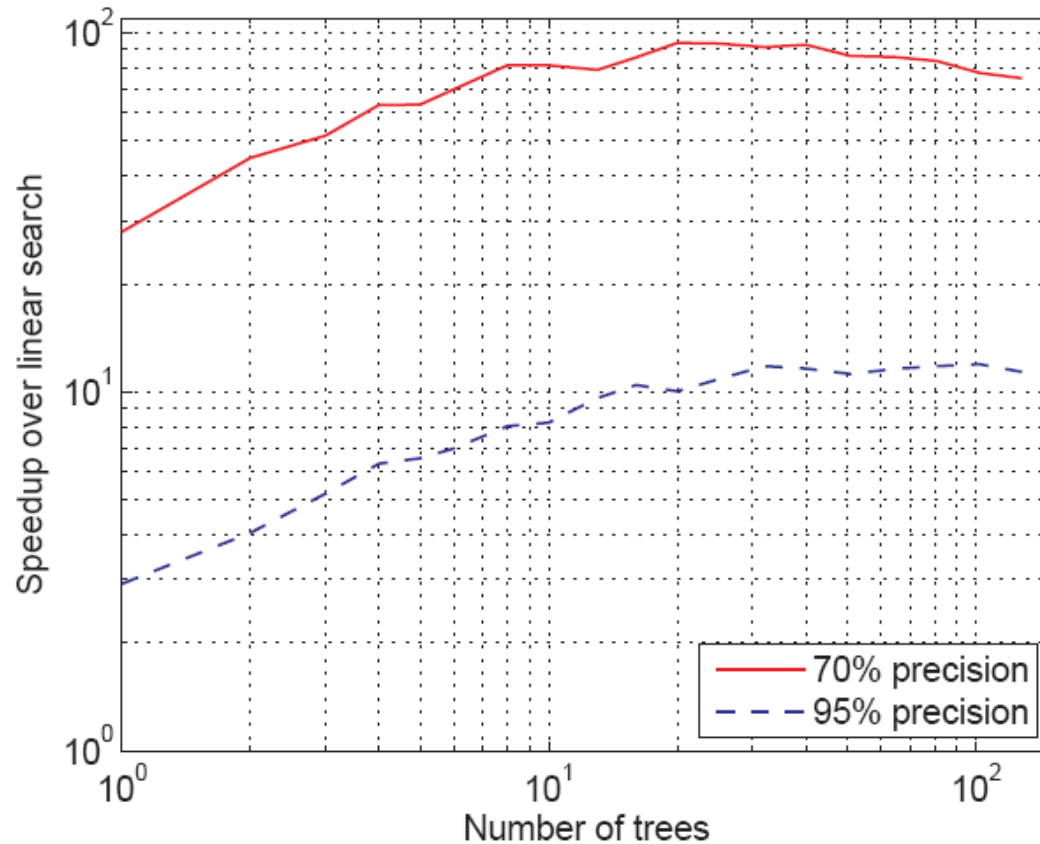
Build several KD Trees

- Find top few dimensions of largest variance at each node
- Randomly select one dimension from these and split on median
- construct many trees, each built completely i.e., one point per leaf
- More memory
- Additional parameter to tune: number of trees

Search

- Descend through each tree until leaf is reached
- Maintain a single priority queue for all the trees
- For approximate search, stop after a certain number of nodes have been examined

Experiments: Randomized KD-Trees



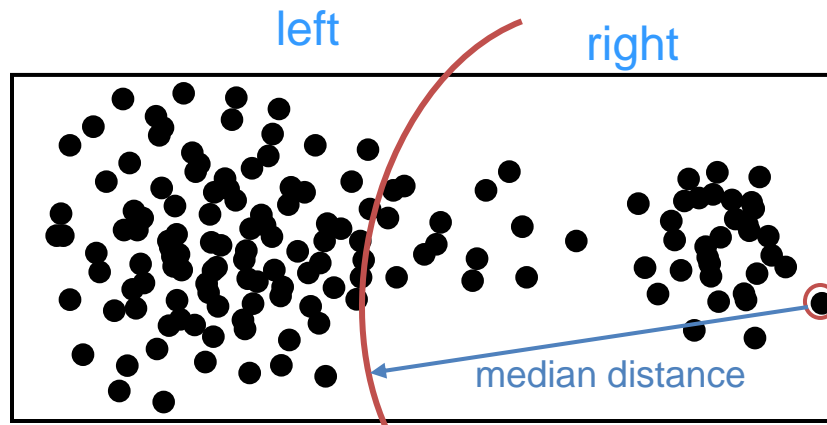
$d = 128, n = 100K$

Muja&Lowe[10]

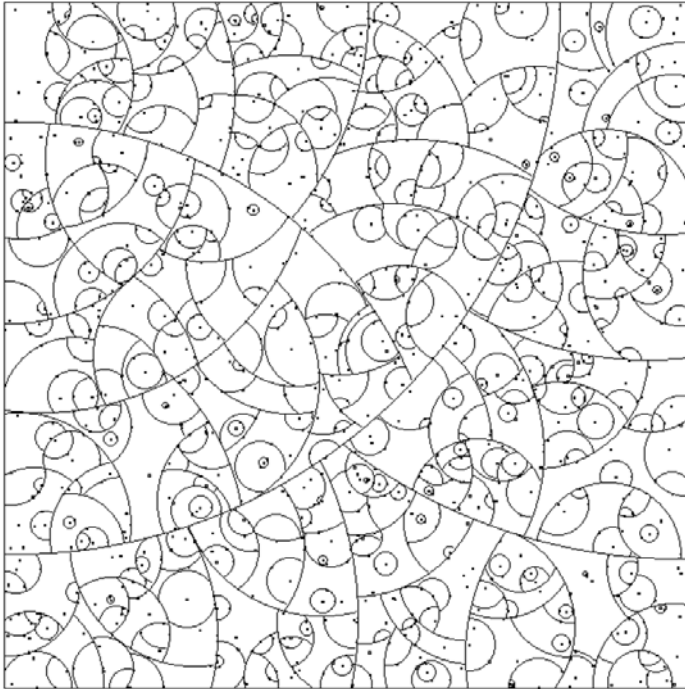
Vantage Point (VP)-Tree

Building VP-Tree

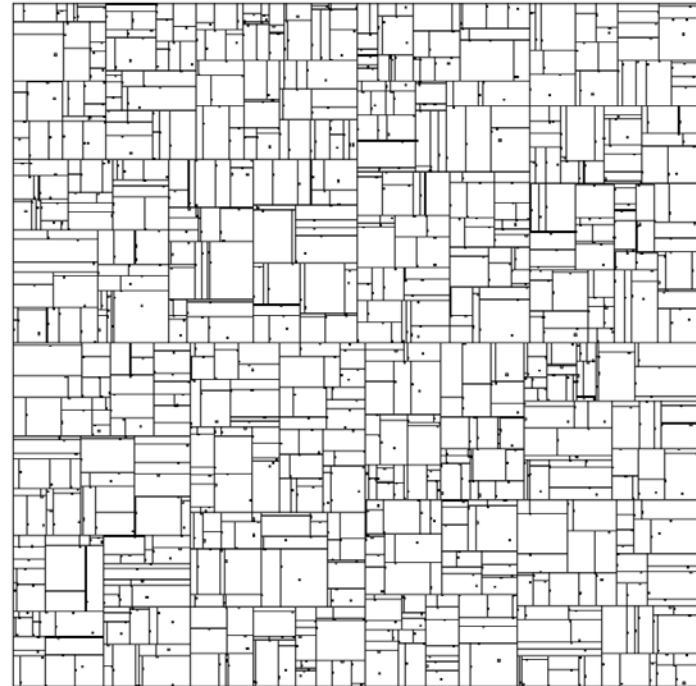
- Select a vantage point randomly (usually from data periphery)
- Compute distance from all other points and pick median distance
- Binary tree: split data using median distance from vantage point
- Split recursively until each node has only one point (leaves)



VP-Tree vs KD-Tree



VP-Tree



KD-Tree

Yianilos[3]

VP-Tree: Properties

- Binary tree of depth $\log(n)$ and total nodes $(2n-1)$
- Construction time: $O(nd\log(n))$
- Memory: nonleaf node – (vantage point, threshold), leaf node – data id
- Need to store the original data also

Backtracking

- Compare against the vantage point at each node to decide whether to explore subtree rooted at that node
- Upper bound on number of nodes to be explored


$$M \lceil \log(n) + 1 \rceil$$

VP-Tree: Properties

- Binary tree of depth $\log(n)$ and total nodes $(2n-1)$
- Construction time: $O(nd\log(n))$
- Memory: nonleaf node – (vantage point, threshold), leaf node – data id
- Need to store the original data also

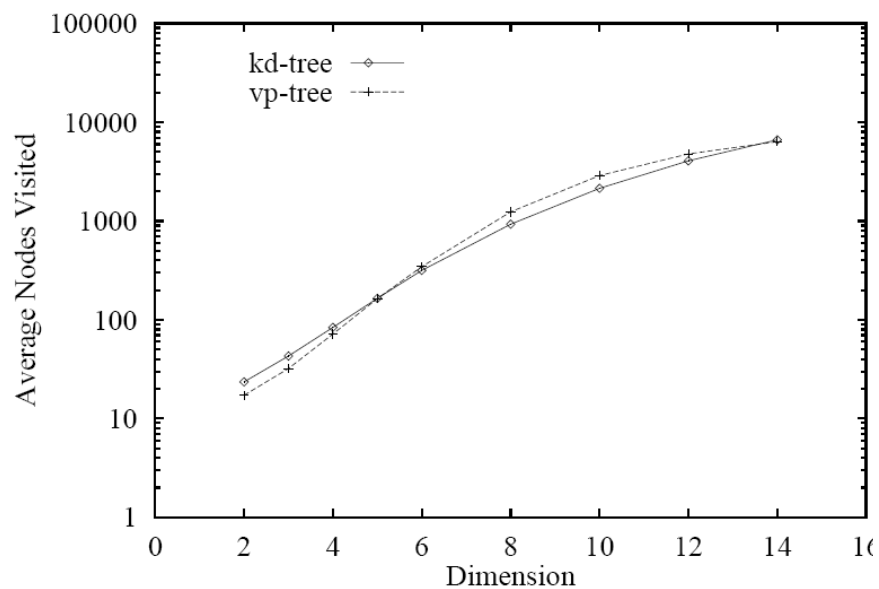
Backtracking

- Compare against the vantage point at each node to decide whether to explore subtree rooted at that node
- Upper bound on number of nodes to be explored

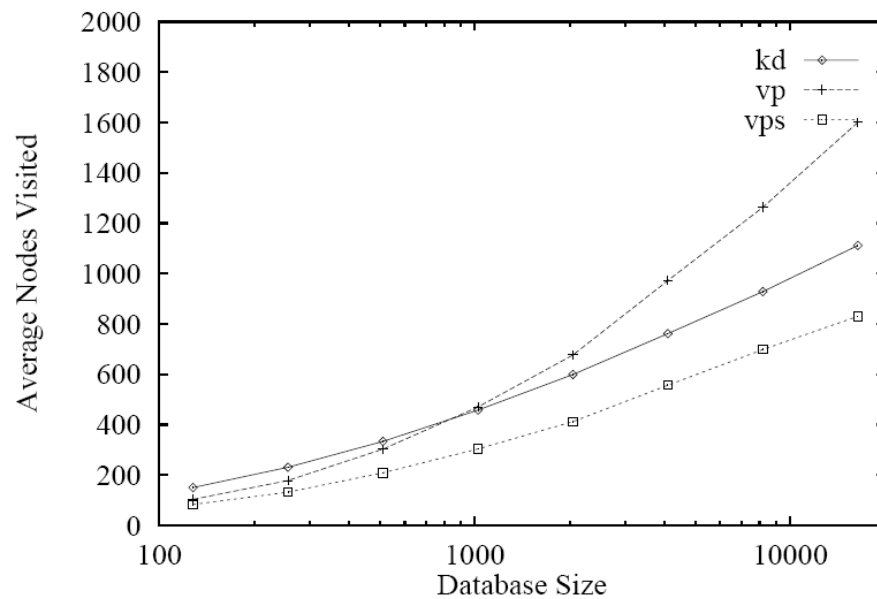
$$\eta^d M \lceil \log(n) + 1 \rceil$$


Exponential in d !

VP-Tree vs KD-Tree



$n = \sim 8K$



$d = 8$

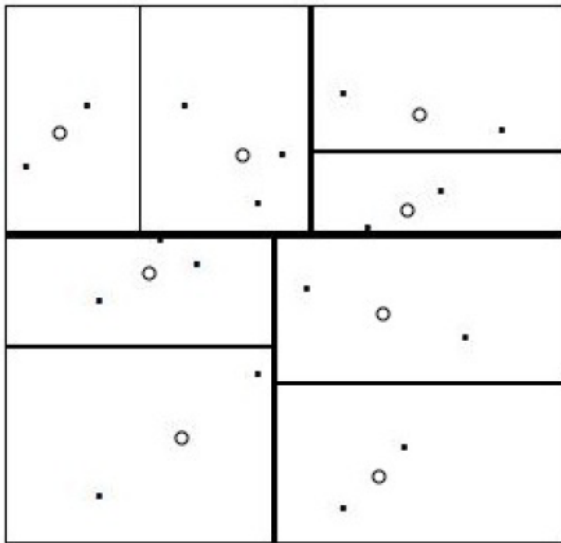
vps: vp-tree with upper/lower bound caching

Yianilos[3]

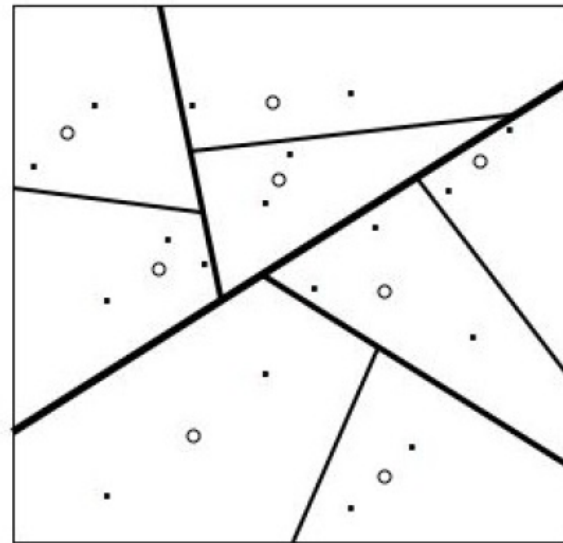
Projection-Based Trees

Project data on a hyperplane and threshold

- Different types of projection directions
- Binary Tree: usually threshold at median
- Split recursively until each node has only one point (leaves)
- Ball-Tree, PCA-Tree, Random-Projection Tree,...



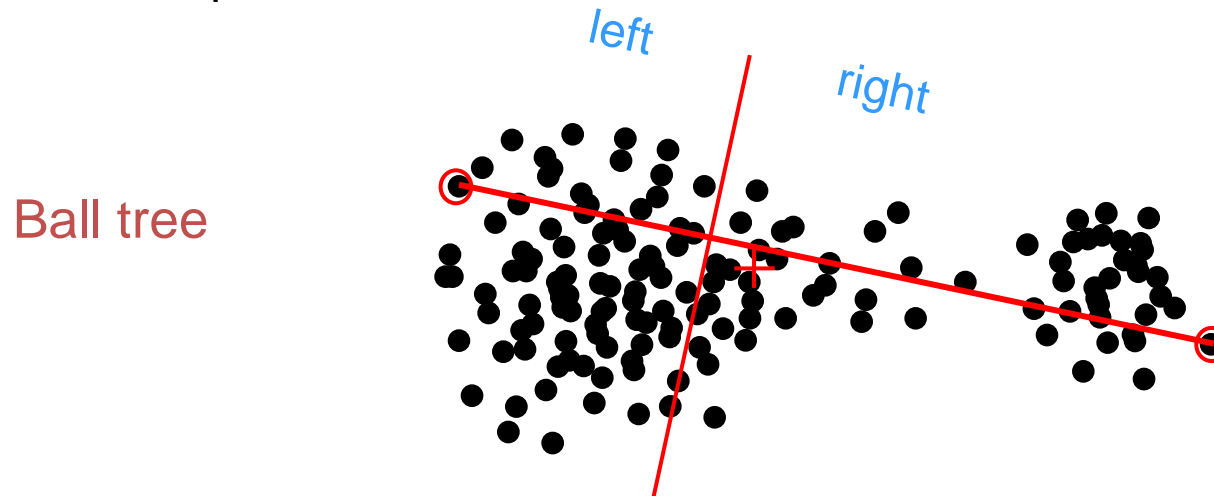
KD-Tree



Projection-Tree

Ball-Tree

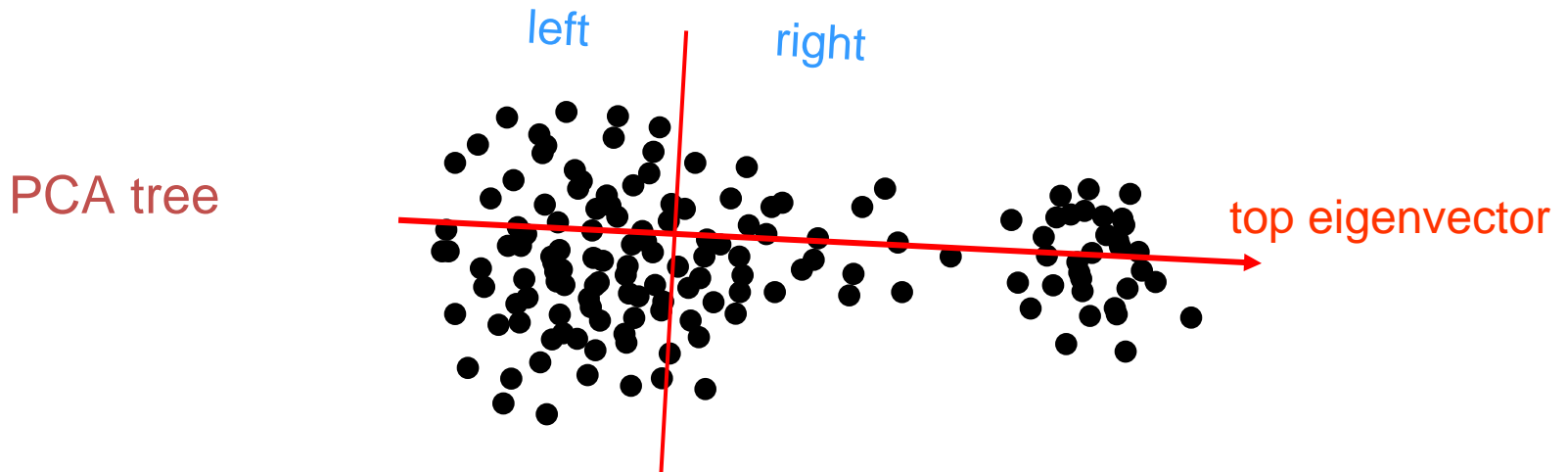
- Find the (approx) diameter of the given dataset
- Find the point farthest from mean and another farthest from it
- Threshold at median
- Another variation: split according to distance from two points (i.e., threshold at mid point of line joining two centers), need to store two vectors per node



Susceptible to outliers!

PCA-Tree

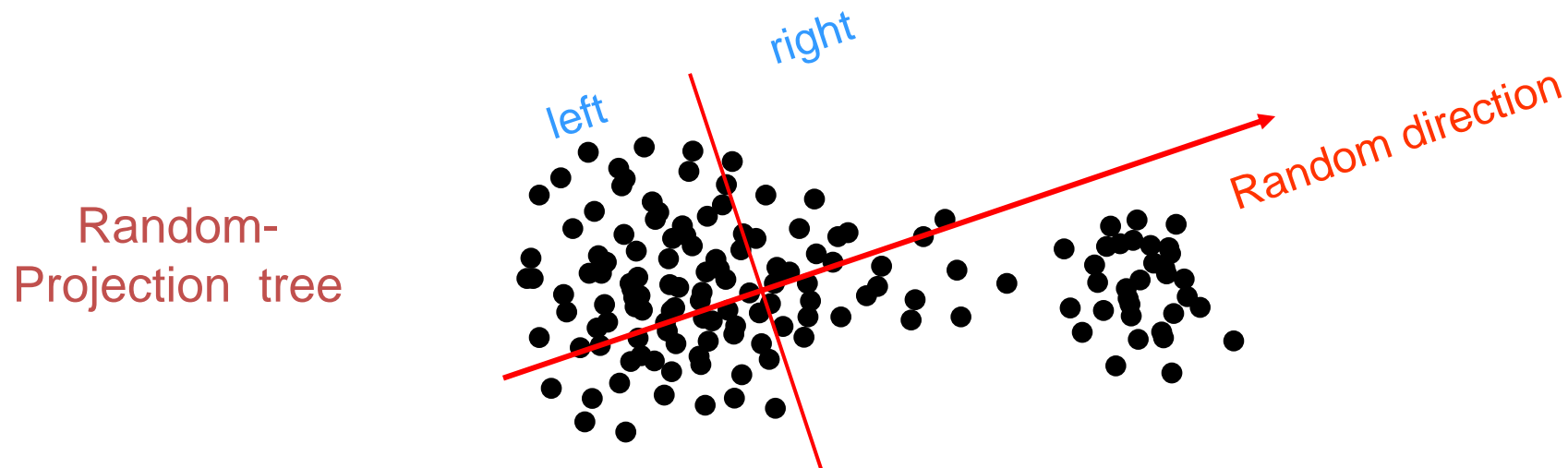
- Use top eigenvector of data covariance as projection direction
- Threshold at median
- More robust than ball tree in presence of outliers



Expensive, not enough data at lower levels to construct covariance!

Random-Projection (RP) Tree

- Randomly sample a projection direction from a fixed distribution
- Threshold at “adjusted” median
- Robust for high-dim data
- Can adapt to low-dimensional structure in the data well



Theoretical Guarantees!

Random-Projection (RP) Tree

Analysis: Show that every $O(D)$ levels, data diameter reduced by half

- Suppose D is the intrinsic dimensionality of the data at any cell, i.e.,

$$\sum_{i=1}^D \sigma_i^2 \geq (1 - \varepsilon) \sum_{i=1}^d \sigma_i^2 \quad \sigma_i^2 = eval(\Sigma_{X_C})$$

covariance of data in cell C

Random-Projection (RP) Tree

Analysis: Show that every $O(D)$ levels, data diameter reduced by half

- Suppose D is the intrinsic dimensionality of the data at any cell, i.e.,

$$\sum_{i=1}^D \sigma_i^2 \geq (1 - \varepsilon) \sum_{i=1}^d \sigma_i^2 \quad \sigma_i^2 = \text{eval}(\Sigma_{X_C})$$

covariance of data in cell C

Let Δ_{X_C} be the diameter of points in cell C (i.e. largest distance between any pair in C), and $\Delta_{X_C}^A$ be the **average** diameter of C :

$$\left(\Delta_{X_C}^A\right)^2 = \frac{1}{|X_C|^2} \sum_{(x,y) \in X_C} \|x - y\|^2 = \frac{2}{|X_C|} \sum_{x \in X_C} \|x - \mu_C\|^2$$

Random-Projection (RP) Tree

Analysis: Show that every $O(D)$ levels, data diameter reduced by half

- Suppose D is the intrinsic dimensionality of the data at any cell, i.e.,

$$\sum_{i=1}^D \sigma_i^2 \geq (1 - \varepsilon) \sum_{i=1}^d \sigma_i^2 \quad \sigma_i^2 = \text{eval}(\Sigma_{X_C})$$

covariance of data in cell C

Let Δ_{X_C} be the diameter of points in cell C (i.e. largest distance between any pair in C), and $\Delta_{X_C}^A$ be the **average** diameter of C :

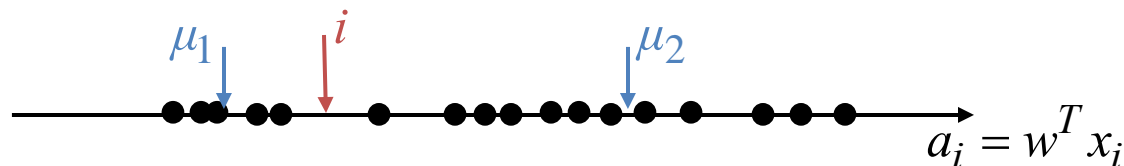
$$\left(\Delta_{X_C}^A\right)^2 = \frac{1}{|X_C|^2} \sum_{(x,y) \in X_C} \|x - y\|^2 = \frac{2}{|X_C|} \sum_{x \in X_C} \|x - \mu_C\|^2$$

$$\sum_{i=1}^d \sigma_i^2 = 1/2 \left(\Delta_{X_C}^A\right)^2$$

Random-Projection (RP) Tree

Analysis: Show that every $O(D)$ levels, data diameter reduced by half

- How to compute threshold?



$$c_i = \sum_{j=1}^i (a_j - \mu_1)^2 + \sum_{j=i+1}^n (a_j - \mu_2)^2$$

Threshold $t = \arg \max_i c_i$

Maximally decreases average squared inter-point distance !

Random-Projection (RP) Tree

Analysis: Show that every $O(D)$ levels, data diameter reduced by half

Theorem: Suppose data subset X_C in any cell C has intrinsic dimension D (for a given $0 < \varepsilon < 1$). Pick a point $x \in X_C$ at random and let C' be the cell that contains it in the next level down. Then,

$$E\left[\left(\Delta_{X_{C'}}^A\right)^2\right] \leq \left(1 - \frac{c}{D}\right)\left(\Delta_{X_C}^A\right)^2 \quad 0 < c < 1$$

Random-Projection (RP) Tree

Analysis: Show that every $O(D)$ levels, data diameter reduced by half

Theorem: Suppose data subset X_C in any cell C has intrinsic dimension D (for a given $0 < \varepsilon < 1$). Pick a point $x \in X_C$ at random and let C' be the cell that contains it in the next level down. Then,

$$E\left[\left(\Delta_{X_{C'}}^A\right)^2\right] \leq \left(1 - \frac{c}{D}\right) \left(\Delta_{X_C}^A\right)^2 \quad 0 < c < 1$$

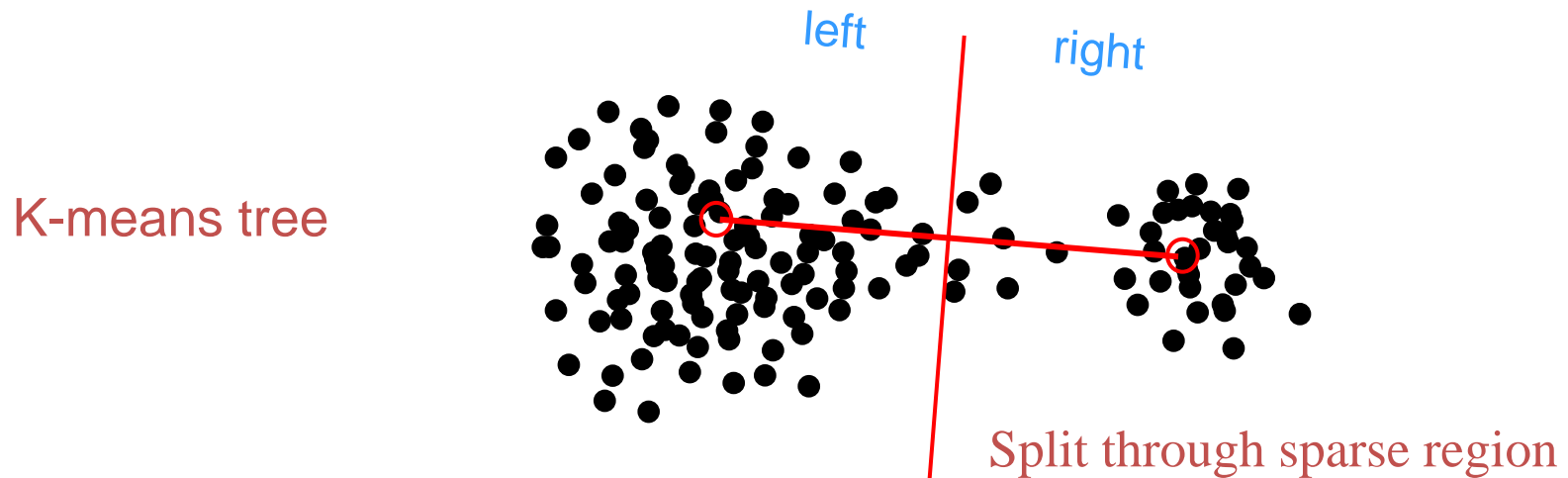
After D levels, reduction in expected average data diameter

$$\left(1 - \frac{c}{D}\right)^{D/2}$$

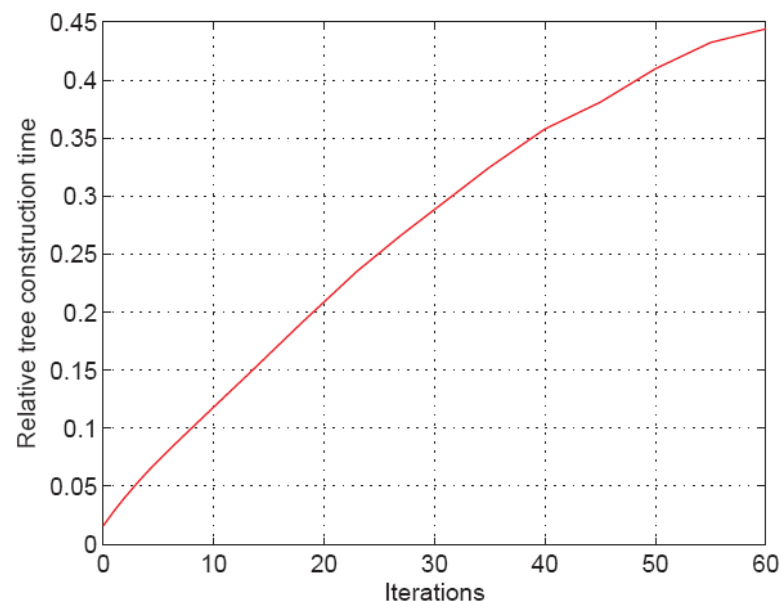
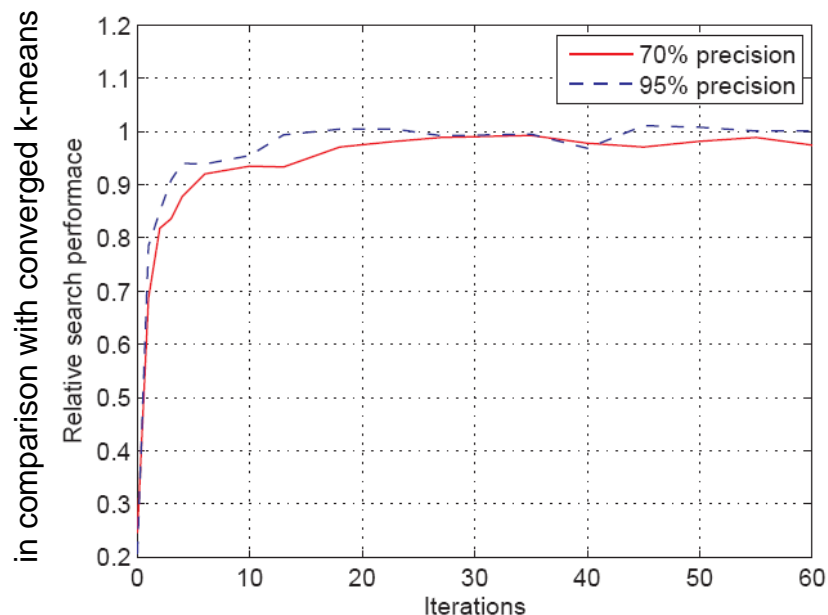
For moderate to large D , average diameter is halved every $O(D)$ levels !

K-Means Tree

- Many real-world datasets contain clusters
- Find the two end-points by iteratively finding centers using k-means
- Points are split based on closer center
- Unbalanced partitioning, more construction time



Number of K-means Iterations



$$d = 128, n = 100K$$

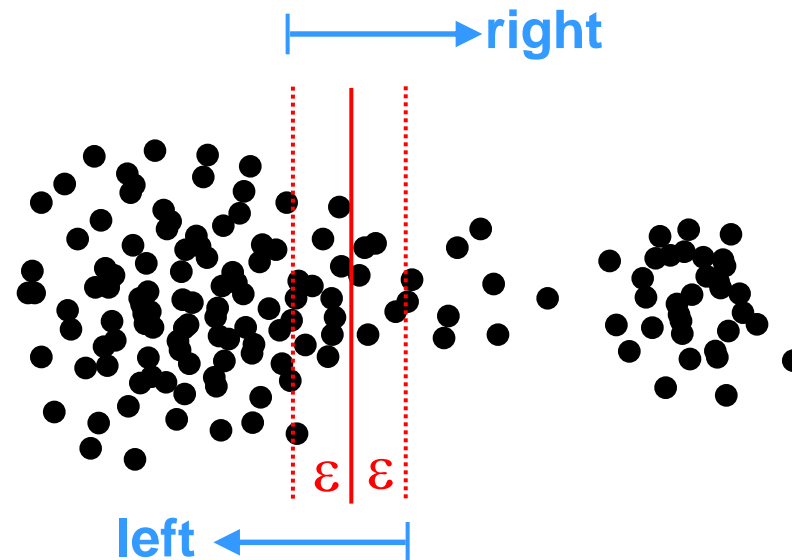
disjoint partitioning of data → boundary errors → backtracking

Muja&Lowe[10]

Spilling in Trees

Overlapped partitioning reduces boundary errors

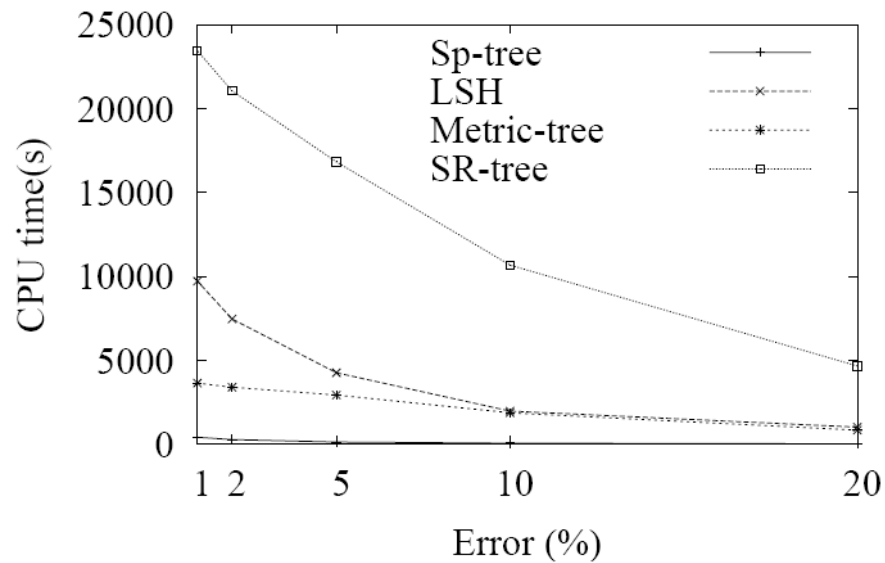
- no backtracking necessary



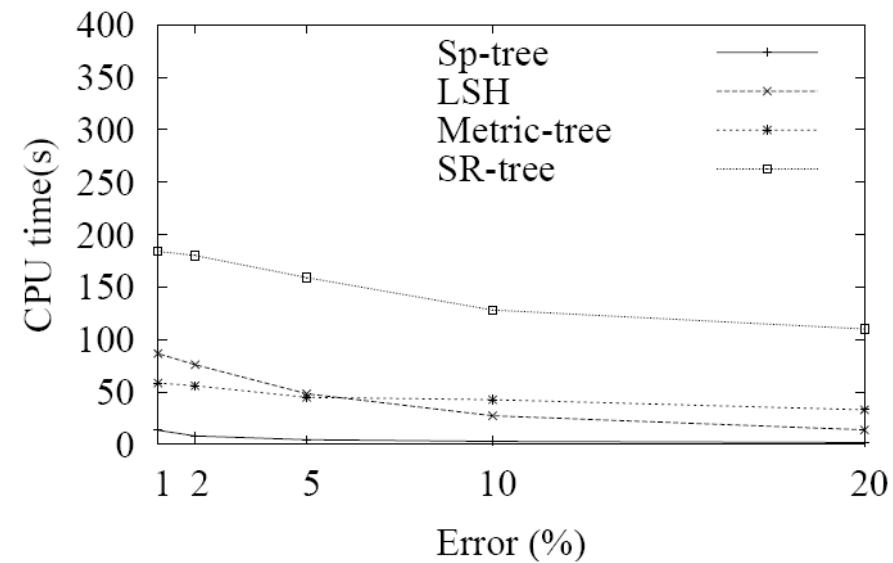
- Increases tree depth - more memory, slower to build
- Better when split passes through sparse regions (k-means)
- Lower nodes may spill too much → hybrid of spill and non-spill nodes
- Designing a good spill factor hard

Effect of spilling

Aerial ($D=60$, $n=275,476$, $k=1$)



Corel_hist ($D=64$, $n=20,000$, $k=1$)



Liu et al.[7]

Tricks for Trees

- For high dimensional data, use randomized projections or PCA to do **dimensionality reduction**
- While backtracking, use **Best-Bin-First (BBF)** search instead of Last-Bin-First (LBF)
- BBF Search: make a **priority queue** of all the unexplored nodes and visit them in order of their “closeness” to the query
- In KD-trees, closeness is defined by distance to a cell boundary, while in k-means tree, it is distance to the center of a cell
- Space permitting, keep extra statistics on **lower and upper bound** for each cell and use **triangle inequality** to prune space
- Use **spilling** to avoid backtracking
- use **lookup tables** for fast distance computation, if possible

Two Main problems

- Memory needed is usually quite big (sometimes more than original data)
- Original data is always needed at search-time (may not be feasible for very large databases)

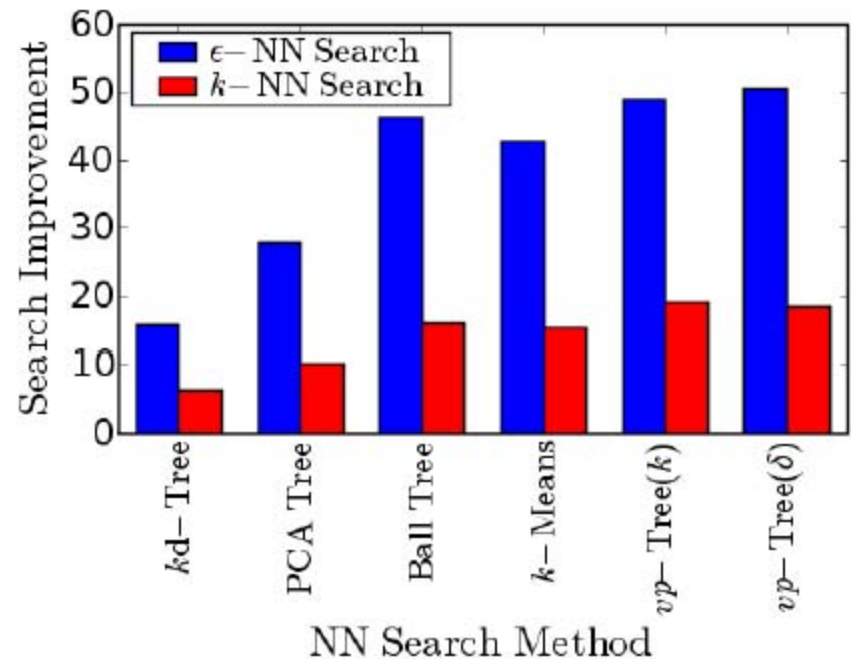
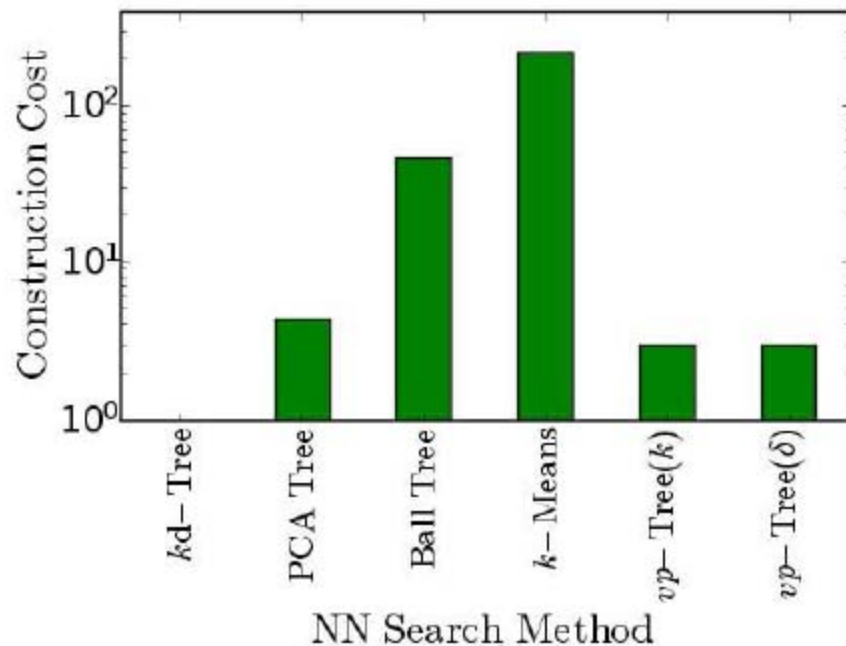
Experiments - Exact Search

Method	Leaf Size	Split	Cons. Cost	ϵ -NN Search Speed	k -NN Search Speed	Method	Leaf Size	Split	Cons. Cost	ϵ -NN Search Speed	k -NN Search Speed
Ball Tree	2	2	76.52	14.23	0.50	k d-Tree	2	1	0.00	4.45	1.88
		3	65.81	14.74	0.62		16	1	0.00	14.69	6.23
		4	65.20	14.19	0.68		128	1	0.00	15.82	6.31
	16	2	69.80	46.23	15.84	vp -Tree (k)	2	2	16.28	38.54	1.69
		3	56.61	45.37	15.41			4	9.07	38.48	1.89
		4	56.26	44.17	15.42			8	7.49	34.83	1.57
	128	2	53.30	44.77	15.37			16	6.28	31.55	1.50
		3	46.44	44.31	14.87		16	2	13.97	44.66	15.84
		4	46.33	44.64	15.14			4	7.13	46.07	16.49
k -Means	2	2	303.92	4.47	0.50			8	5.00	48.95	17.61
		4	289.93	6.17	0.69			16	4.02	41.95	19.12
	16	2	271.60	19.20	7.62		128	2	11.05	33.80	11.85
		4	270.40	20.98	8.36			4	5.95	36.75	12.72
	128	2	221.37	35.09	12.69			8	4.00	37.50	12.53
		4	228.73	38.90	13.89			16	3.00	38.13	12.28
		8	298.72	42.70	15.20	vp -Tree (δ)	2	2	5.71	45.29	10.27
PCA Tree	2	4	6.59	22.10	10.26			4	9.30	42.09	16.36
		8	8.25	25.50	9.81			8	9.60	36.96	16.94
	16	2	7.79	17.70	4.42		16	2	4.36	50.45	18.27
		4	5.50	26.60	10.13			4	7.94	49.36	18.21
		8	7.55	27.72	9.63			8	8.78	39.85	15.72
		8	7.61	17.41	4.24		128	2	3.01	44.25	13.05
	128	2	4.32	25.11	8.00			4	5.85	42.55	13.32
		4	6.24	25.80	8.37			8	6.64	36.23	12.31
		8	6.99	17.69	4.34						

$$d = 49, n = O(M)$$

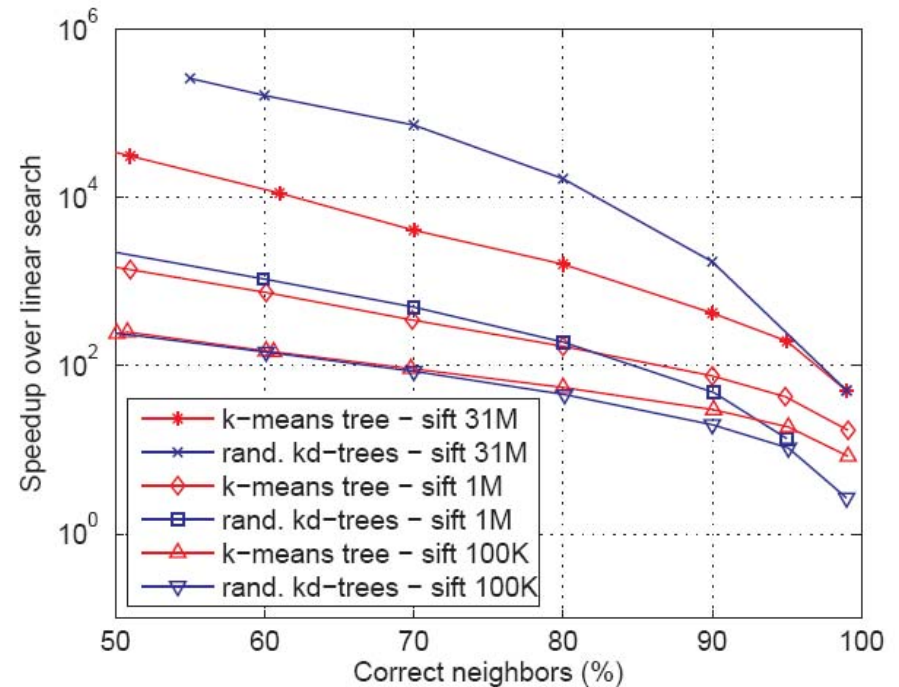
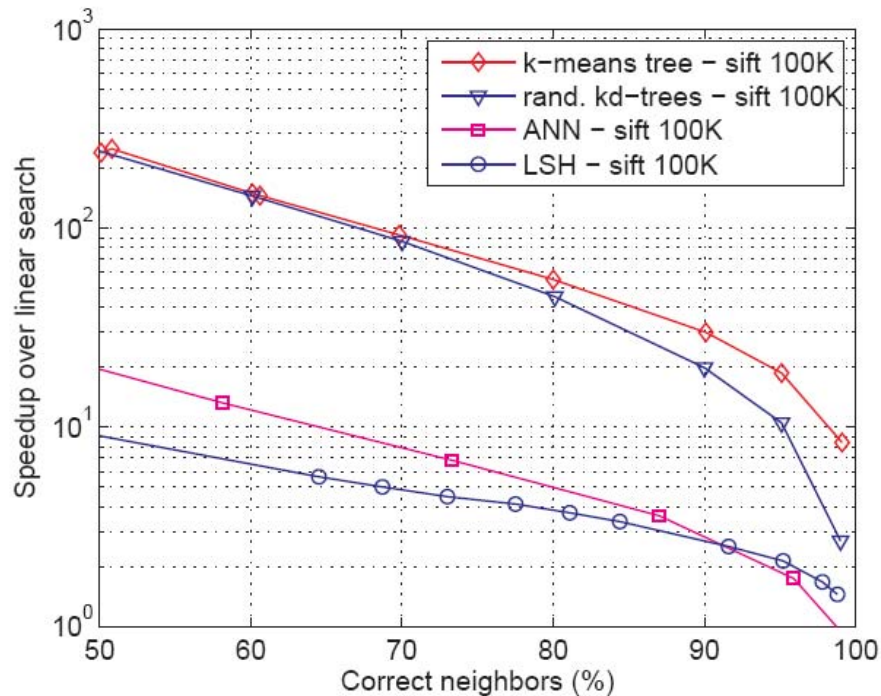
N. Kumar et al. [12]

Experiments - Exact Search



N. Kumar et al. [12]

Experiments - Approximate Search



$d = 128$

Muja&Lowe[10]

References

1. J. L. Bentley. Multidimensional binary search trees used for associative searching. *Comm. ACM*, 18(9), 1975.
2. Freidman, J. H., Bentley, J. L., and Finkel, R. A. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3:209–226, 1977.
3. S. M. Omohundro. Five balltree construction algorithms. Technical report, Int. Computer Science Inst., 1989.
4. Sproull, R.F.: Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica* 6(4) (1991) 579–589
5. P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA*, 1993.
6. Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R., and Wu, A. Y. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of the ACM*, 45:891–923, 1998.
7. T. Liu, A. Moore, A. Gray, and K. Yang. An investigation of practical approximate nearest neighbor algorithms. In *NIPS* 2004, 12 2004.
8. S. Dasgupta and Y. Freund. Random projection trees and low-dimensional manifolds. *UCSD Technical Report CS2007-0890*, 2007.
9. C. Silpa-Anan and R. Hartley. Optimised KD-trees for fast image descriptor matching. In *CVPR*, 2008.
10. M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP*, 2009.
11. Y. Freund, S. Dasgupta, M. Kabra, and N. Verma. Learning the structure of manifolds using random projections. *NIPS*, 2007.
12. N. Kumar, L. Zhang, and S. Nayar. What is a good nearest neighbors algorithm for finding similar patches in images? In *ECCV*, 2008.