Nearest Neighbor Search

Steve Oudot

(steve.oudot@inria.fr)
Nearest-Neighbor problem

pre-processing input: $P$
Nearest-Neighbor problem

pre-processing input: $P$

query input: $q$

goal: find $p \in \text{NN}_P(q)$

\[ d(q, p) = \min_{p' \in P} d(q, p') \]
$\varepsilon$-Nearest-Neighbor problem

pre-processing input: $P, \varepsilon$

query input: $q$

goal: find $p \in \text{NN}_P(q, \varepsilon)$

\[
d(q, p) \leq (1 + \varepsilon) \min_{p' \in P} d(q, p')
\]
Nearest-Neighbor problem

Variants:

• $k$-nearest neighbors: find the $k$ points closest to $q$ in $P$

• $r$-nearest neighbor: find a point $p \in P$ such that $d(q, p) \leq r$

• metrics:
  ▶ $\ell_2$, $\ell_p$, $\ell_\infty$
  ▶ strings: Hamming distance
  ▶ images: optimal transport distances
  ▶ point clouds: (Gromov-)Hausdorff distances
  ▶ proteins: RMSD distances
  ▶ ...
Applications

- clustering, e.g. k-means, mean-shift
- information retrieval in databases
- information theory, e.g. vector quantization
- supervised learning, e.g. NN-classifiers
- ...
Linear scan

Input: \( P = \{p_1, \cdots, p_n\} \subset \mathbb{R}^d, q \in \mathbb{R}^d \)

\( d_{\text{min}} := \infty \) (dist. to nearest neighbor among the pts viewed so far)

for \( i = 1 \) to \( n \) do:

\[ d_{\text{min}} := \min \{d_{\text{min}}, d(q, p_i)\} \]

return \( d_{\text{min}} \) or index \( i \) that achieves \( d_{\text{min}} \)
Linear scan

**Input:** \( P = \{p_1, \cdots, p_n\} \subset \mathbb{R}^d, \ q \in \mathbb{R}^d \)

\( d_{\text{min}} := \infty \) (dist. to nearest neighbor among the pts viewed so far)

**for** \( i = 1 \) to \( n \) **do:**

\[
d_{\text{min}} := \min \{d_{\text{min}}, d(q, p_i)\}
\]

**return** \( d_{\text{min}} \) or index \( i \) that achieves \( d_{\text{min}} \)

**Complexity:**

space: \( O(dn) \) — \( n \) points, \( d \) coordinates each

time: \( O(dn) \) — \( n \) iterations, 1 distance computation each
Strategy and Challenges

Strategy:

▸ preprocess the $n$ point of $P$ in $\mathbb{R}^d$ into some data structure $DS$ for fast nearest-neighbor queries answers

Ideal wish list:

▸ $DS$ should have linear size in $n$ and polynomial size in $d$
▸ a query should take sublinear time in $n$ and polynomial time in $d$
  e.g. binary search trees in $d = 1$: linear size, $O(\log n)$ time

Core difficulties:

▸ *Curse of dimensionality*: hard to outperform linear scan in high $d$
▸ Interpretation: meaningfulness of distances in high $d$ (concentration)
Approaches

- Linear scan
- Voronoi diagrams
- Tree-like data structures
  - quadtrees (split at midpoint in all coordinates)
  - tries / dyadic trees (split at mean, cycle around coordinates)
  - kd-trees (split at median, cycle around coordinates)
  - Random Projection trees (split at median along random coordinates)
  - PCA trees (split at median along 1st eigenvector of covariance matrix)
  - …
- Locality Sensitive Hashing
Voronoi diagrams
Definition

\[ V(p) := \{ q \in \mathbb{R}^d \mid p \in \text{NN}_P(q) \} \]

affine diagram
Definition

\[ V(p) := \{ q \in \mathbb{R}^d \mid p \in \text{NN}_P(q) \} \]

affine diagram

computed/stored via dual

(Delaunay triangulation)
Definition

\[ V(p) := \{ q \in \mathbb{R}^d \mid p \in \text{NN}_P(q) \} \]

affine diagram

computed/stored via dual (Delaunay triangulation)

size:

- worst case: \( \Theta \left( n^{\lceil d/2 \rceil} \right) \)
  Upper Bound Thm [McMullen’70]

- average case (unif. distrib.):
  \( 2^{O(d \log d)} n \)
Usage for NN-search

Localizing by \textbf{walk}:

- start from $p \in P$ random

\textbf{while} $\exists p'$ neighb. of $p$ in Del.

\hspace{1cm} \text{s.t. } d(q, p') < d(q, p):

\hspace{2cm} \text{update } p := p'$
Usage for NN-search

Localizing by walk:

- start from \( p \in P \) random

while \( \exists p' \) neighb. of \( p \) in Del.

\[
\text{s.t. } d(q, p') < d(q, p):
\]

update \( p := p' \)

Prop: Del. neighborhood is complete

walk time:

worst case: \( O(|\text{Del}(P)|) \)

avg. case (2d): \( O(\sqrt{n}) \)
Usage for NN-search

Localizing by hierarchy:

- Voronoi subdivision [Kirk.’83, Meiser’93]:
  
  (2D) $O(n)$ space, $O(\log n)$ time
  
  (dD) $\Theta(n^d)$ space, $O(d^5 \log n)$ time

- Delaunay tree [Mulmuley’91]:
  
  (2D) $O(n \log n)$ space, $O(\log n)$ time

- Delaunay tree + walk [Devillers’02]:
  
  (2D) $O(n \log n)$ space, $O(\log n)$ time
  
  (dD) $O(n^{\lceil \frac{d}{2} \rceil})$ space, $O(n^{\lceil \frac{d-2}{2} \rceil})$ time
Usage for NN-search

Localizing by hierarchy:

- Voronoi subdivision [Kirk.'83, Meiser’93]:
  \[ (2D) \quad O(n) \text{ space}, \quad O(\log n) \text{ time} \]
  \[ (dD) \quad \Theta(n^d) \text{ space}, \quad O(d^5 \log n) \text{ time} \]

- Delaunay tree [Mulmuley’91]:
  \[ (2D) \quad O(n \log n) \text{ space}, \quad O(\log n) \text{ time} \]

- Delaunay tree + walk [Devillers’02]:
  \[ (2D) \quad O(n \log n) \text{ space}, \quad O(\log n) \text{ time} \]
  \[ (dD) \quad O(n^{\left\lceil \frac{d}{2} \right\rceil}) \text{ space}, \quad O(n^{\left\lceil \frac{d-2}{2} \right\rceil}) \text{ time} \]

For small dimensions (2 or 3) only!
k-d trees
Definition

- a binary tree

- each internal node implements a spatial partition induced by a hyperplane $H$, splitting the point cloud into two equal subsets
  - right subtree: all points lying on one side of $H$
  - left subtree: remaining points

- subdivision stops whenever fewer than $n_0$ remain

\[\leadsto \text{size: } O(dn)\]
Definition

- a binary tree

- each internal node implements a spatial partition induced by a hyperplane \( H \), splitting the point cloud into two equal subsets
  - right subtree: all points lying on one side of \( H \)
  - left subtree: remaining points

- subdivision stops whenever fewer than \( n_0 \) remain
  \[ \sim \text{size: } O(dn) \]

kd-tree specifics:

- \( H \) orthogonal to coordinate axis (cycle through coordinates)
- \( H \) goes through the median in the considered coordinate

\( n_0 = 1 \)
\( l_i \): data at internal node

\( p_i \): data at leaf node

\( n_0 = 1 \)
Usage for NN search

**Strategy 1:** defeatist search

\[ d_{min} := \infty \text{ (dist. to pts viewed so far)} \]

search \((node)\): \((node = root \text{ initially})\)

\[ \text{if } node = \text{leaf}: \]
\[ d_{min} := \min\{d_{min}, \min_{p \in node.\text{batch}} d(q, p)\} \]

\[ \text{else:} \]
\[ d_{min} := \min\{d_{min}, d(q, node.\text{point})\} \]

\[ \text{if } q \text{ on "left" side of node.H} \]
\[ \text{recurse on node.left} \]

\[ \text{else (} q \text{ on "right" side of node.H)} \]
\[ \text{recurse on node.right} \]
Usage for NN search

Strategy 1: **defeatist search**

\[ d_{\text{min}} := \infty \] (dist. to pts viewed so far)

search (node): (node = root initially)

\[
\begin{align*}
\text{if } & \text{node = leaf:} \\
& d_{\text{min}} := \min \{ d_{\text{min}}, \min_{p \in \text{node.batch}} d(q, p) \}
\end{align*}
\]

else:

\[
\begin{align*}
& d_{\text{min}} := \min \{ d_{\text{min}}, d(q, \text{node.point}) \}
\end{align*}
\]

\[
\begin{align*}
\text{if } & q \text{ on "left" side of node.H} \\
& \text{recurse on node.left}
\end{align*}
\]

\[
\begin{align*}
\text{else (} q \text{ on "right" side of node.H) } \\
& \text{recurse on node.right}
\end{align*}
\]

Query time: \( O(d(n_0 + \log \frac{n}{n_0})) \)
Usage for NN search

**Strategy 1:** defeatist search

\[ d_{\text{min}} := \infty \text{ (dist. to pts viewed so far)} \]

search \((node)\): \((node = \text{root initially})\)

\[
\text{if } node = \text{leaf}: \quad d_{\text{min}} := \min \{d_{\text{min}}, \min_{p \in \text{node.batch}} d(q, p)\}
\]

\text{else:}

\[
d_{\text{min}} := \min \{d_{\text{min}}, d(q, node.\text{point})\}
\]

\[
\text{if } q \text{ on } "\text{left}" \text{ side of node.}H \\
\quad \text{recurse on node.left}
\]

\[
\text{else } (q \text{ on } "\text{right}" \text{ side of node.}H) \\
\quad \text{recurse on node.right}
\]

Query time: \(O(d(n_0 + \log \frac{n}{n_0}))\)

May fail!

\((n_0 = 1)\)
Example

$l_i$: data at internal node

$p_i$: data at leaf node

(note: left-right labels are arbitrary)
Example

$l_i$: data at internal node

$p_i$: data at leaf node

(note: left-right labels are arbitrary)
Example

$l_i$: data at internal node

$p_i$: data at leaf node

(note: left-right labels are arbitrary)
Example

$l_i$: data at internal node

$p_i$: data at leaf node

(note: left-right labels are arbitrary)
Example

$l_i$: data at internal node

$p_i$: data at leaf node

(note: left-right labels are arbitrary)
Usage for NN search

**Strategy 2:** backtracking search

d\(_{\text{min}} := \infty\) (dist. to pts viewed so far)

search (node): (node = root initially)

\[\text{if } \text{node} = \text{leaf}:\]
\[d_{\text{min}} := \min\{ d_{\text{min}}, \min_{p \in \text{node.batch}} d(q, p)\}\]

\[\text{else:}\]
\[d_{\text{min}} := \min\{ d_{\text{min}}, d(q, \text{node.point})\}\]

\[\text{if } B(q, d_{\text{min}}) \text{ intersects "left" side of node.H}\]
\[\text{recurse on node.left}\]

\[\text{if } B(q, d_{\text{min}}) \text{ intersects "right" side of node.H}\]
\[\text{recurse on node.right}\]
Usage for NN search

Strategy 2: backtracking search

\[ d_{\text{min}} := \infty \] (dist. to pts viewed so far)

search (node): (node = root initially)

\[
\text{if } \text{node} = \text{leaf}: \\
\quad d_{\text{min}} := \min \{ d_{\text{min}}, \min_{p \in \text{node.batch}} d(q, p) \}
\]

\[
\text{else: } \\
\quad d_{\text{min}} := \min \{ d_{\text{min}}, d(q, \text{node.point}) \}
\]

\[
\text{if } B(q, d_{\text{min}}) \text{ intersects "left" side of } \text{node.H} \\
\quad \text{recurse on } \text{node.left}
\]

\[
\text{if } B(q, d_{\text{min}}) \text{ intersects "right" side of } \text{node.H} \\
\quad \text{recurse on } \text{node.right}
\]

Always succeeds

\[
d_{\text{min}} \geq d(q, \text{NN}(q)) \Rightarrow B(q, d_{\text{min}}) \text{ intersects all cells containing } \text{NN}(q) \text{ in subdivision throughout search}
\]
Usage for NN search

**Strategy 2:** backtracking search

\[ d_{\text{min}} := \infty \text{ (dist. to pts viewed so far)} \]

search \((\text{node})\): \((\text{node} = \text{root} \text{ initially})\)

\[
\text{if } \text{node} = \text{leaf}: \\
\quad d_{\text{min}} := \min\{d_{\text{min}}, \min_{p \in \text{node.batch}} d(q, p)\}
\]

\[
\text{else:} \quad d_{\text{min}} := \min\{d_{\text{min}}, d(q, \text{node.point})\}
\]

\[
\text{if } B(q, d_{\text{min}}) \text{ intersects "left" side of node.H} \\
\quad \text{recurse on node.left}
\]

\[
\text{if } B(q, d_{\text{min}}) \text{ intersects "right" side of node.H} \\
\quad \text{recurse on node.right}
\]

Always succeeds

Query time may be up to linear (all cells visited)
Example

\[ l_i \]: data at internal node

\[ p_i \]: data at leaf node

(note: left-right labels are arbitrary)
Example

\[ l_i : \text{data at internal node} \]
\[ p_i : \text{data at leaf node} \]

(note: left-right labels are arbitrary)
Example

$l_i$: data at internal node

$p_i$: data at leaf node

(note: left-right labels are arbitrary)
Example

$l_i$: data at internal node

$p_i$: data at leaf node

(note: left-right labels are arbitrary)
Example

\( l_i \): data at internal node

\( p_i \): data at leaf node

(note: left-right labels are arbitrary)
Example

\( l_i \): data at internal node

\( p_i \): data at leaf node

(note: left-right labels are arbitrary)
Example

$l_i$: data at internal node

$p_i$: data at leaf node

(note: left-right labels are arbitrary)
Example

\[ l_i : \text{data at internal node} \]

\[ p_i : \text{data at leaf node} \]

(note: left-right labels are arbitrary)
Example

$l_i$: data at internal node

$p_i$: data at leaf node

(note: left-right labels are arbitrary)
Example

\[ l_i: \text{data at internal node} \]

\[ p_i: \text{data at leaf node} \]

(note: left-right labels are arbitrary)
Example

\[l_i\]: data at internal node

\[p_i\]: data at leaf node

(note: left-right labels are arbitrary)
Example

$l_i$: data at internal node

$p_i$: data at leaf node

(note: left-right labels are arbitrary)
Example

$l_i$: data at internal node

$p_i$: data at leaf node

(note: left-right labels are arbitrary)
Example

$l_i$: data at internal node

$p_i$: data at leaf node

(note: left-right labels are arbitrary)
Example

\[ l_i \]: data at internal node

\[ p_i \]: data at leaf node

(note: left-right labels are arbitrary)
$l_i$: data at internal node

$p_i$: data at leaf node

(note: left-right labels are arbitrary)
Example

- $l_i$: data at internal node
- $p_i$: data at leaf node

(note: left-right labels are arbitrary)
Example

\[ l_i : \text{data at internal node} \]

\[ p_i : \text{data at leaf node} \]

(note: left-right labels are arbitrary)
Example

worst-case input (non-unif. distrib.):
long skinny cells

query time = $\Omega(dn)$
The constant $c_d$ is exponential in $d$. Indeed, when $q$ sits near a vertex of the decomposition, its NN-ball intersects the $2^d$ neighboring cells, which thus have to be inspected. This is fine for the complexity, as point cloud density means basically $n >> 2^d$, so the factor $c_d$ is reasonable in this case.

Example

query time $= O(c_d \log n)$

best-case input (unif. distrib.):

small fat cells

$\downarrow$

query time $= O(c_d \log n)$
The constant $c_d$ is exponential in $d$. Indeed, when $q$ sits near a vertex of the decomposition, its NN-ball intersects the $2^d$ neighboring cells, which thus have to be inspected. This is fine for the complexity, as point cloud density means basically $n \gg 2^d$, so the factor $c_d$ is reasonable in this case.

Example

query time = $O(c_d \log n)$

best-case input (unif. distrib.): small fat cells

⇒ Randomness should help!

(many variants: priority search, early backtracking, random cutting hyperplanes, etc.)
The lack of linearity of the linear scan is probably due to the asymptotic regime starting pretty late (apparently around n = 100000 data points). More refined quantities, e.g., number of executions of loop, may explain the change of regime around n = 100000 data points.

**Benchmarks**

**avg. query time (μs) vs. # data points:** (uniform measure in unit square in 2d)
Benchmarks

**avg. query time \((\mu s)\) vs. # data points:** (uniform measure on unit circle in 2d)
High dimensions

pre-processing input: \( P \subset \mathbb{R}^d \)

query input: \( q \)

goal: find \( p \in \text{NN}_P(q) \)

Curse of Dimensionality:
Every data structure for NN-search has either exponential size or exponential query time (in \( d \)) in the worst case.
Curse of Dimensionality:
Every data structure for NN-search has either exponential size or exponential query time (in $d$) in the worst case.

→ holds both in theory and in practice [Weber et al. ’98] [Arya et al. ’98]
Curse of Dimensionality:
Every data structure for NN-search has either exponential size or exponential query time (in $d$) in the worst case.

pre-processing input: $P \subset \mathbb{R}^d$

query input: $q$

goal: find $p \in \text{NN}_P(q)$

→ holds both in theory and in practice [Weber et al. '98] [Arya et al. '98]

→ underlying phenomenon: concentration of measure

(distances concentrate around mean) [Demartinez '94]
Random Projection/Partition Trees
Exploiting randomness: RP-trees

Random Projection/Partition tree:

at each internal node (corresponding to a cell $C$):

- choose $v \sim \text{unif}(S^{d-1})$ and $\beta \sim \text{unif}([\frac{1}{4}, \frac{3}{4}])$
- let $H = v^\perp + \text{median}_\beta \{(P \cap C) \cdot v\} v$
- partition $P \cap C$ by $H$ (as in kd-tree)

at each leaf node, store at most $n_0$ points
Exploiting randomness: RP-trees

**Prop:** [Dasgupta, Freund’08]
There is a constant $c > 0$ such that, for any cell $C$ in a RP-tree built on $P \in \mathbb{R}^d$, with probability at least $1/2$ (over the choice of $v, \beta$) all the cells lying at least $ck \log k$ levels below $C$ in the tree have at most half the radius of $C$, where $k = \dim_2(P \cap C)$.

**Doubling dimension** of $S \subseteq \mathbb{R}^d$: smallest $k \in \mathbb{N}$ such that, for every Euclidean ball $B$, $B \cap S$ can be covered by $2^k$ Euclidean balls of half radius.

**Radius** of $S \subseteq \mathbb{R}^d$: smallest $r > 0$ such that $\exists x \in C$ with $B(x, r) \supseteq S$. 
### Thm: [Dasgupta, Sinha’13]

Suppose $p_1, \cdots, p_n \overset{iid}{\sim} \mu$ continuous probability measure in $\mathbb{R}^d$ with doubling dimension $k \geq 2$. Then $\exists c_0 > 0$ s.t. for any $q \in \mathbb{R}^d$ and $\delta < 1/e$, with proba. $\geq 1 - 3\delta$ over the choice of the $p_i$’s:

$$\mathbb{P}_{v, \beta} \left[ \text{defeatist search does not return } \text{NN}_P(q) \right] \leq c_0 (k + \ln n_0) \left( \frac{8 \ln 1/\delta}{n_0} \right)^{1/k}$$

**doubling dimension** of $\mu$: smallest $k \in \mathbb{N}$ such that, for every $x \in \mathbb{R}^d$ and every $r > 0$: $\mu(B(x, 2r)) \leq 2^k \mu(B(x, r))$. 
Exploiting randomness: RP-trees

**Thm:** [Dasgupta, Sinha’13]

Suppose \( p_1, \ldots, p_n \overset{iid}{\sim} \mu \) continuous probability measure in \( \mathbb{R}^d \) with doubling dimension \( k \geq 2 \). Then \( \exists c_0 > 0 \) s.t. for any \( q \in \mathbb{R}^d \) and \( \delta < 1/e \), with proba. \( \geq 1 - 3\delta \) over the choice of the \( p_i \)’s:

\[
\mathbb{P}_{v, \beta} \left[ \text{defeatist search does not return } \text{NN}_P(q) \right] \leq c_0 (k + \ln n_0) \left( \frac{8 \ln 1/\delta}{n_0} \right)^{1/k}
\]

**doubling dimension** of \( \mu \): smallest \( k \in \mathbb{N} \) such that, for every \( x \in \mathbb{R}^d \) and every \( r > 0 \):

\[
\mu(B(x, 2r)) \leq 2^k \mu(B(x, r)).
\]

→ take \( n_0 \propto (k \ln k)^k \ln 1/\delta \) to make \( \mathbb{P}_{v, \beta} \left[ \cdots \right] \) an arbitrarily small constant

→ query time: \( O(d((k \ln k)^k + \log n)) \) sensitive to intrinsic dimension \( k \) requires to know \( k \)
Exploiting randomness: RP-trees

**Thm:** [Dasgupta, Sinha’13]
Suppose \( p_1, \ldots, p_n \overset{iid}{\sim} \mu \) continuous probability measure in \( \mathbb{R}^d \) with doubling dimension \( k \geq 2 \). Then \( \exists c_0 > 0 \) s.t. for any \( q \in \mathbb{R}^d \) and \( \delta < 1/e \), with proba. \( \geq 1 - 3\delta \) over the choice of the \( p_i \)'s:

\[
P_{v,\beta} \text{[defeatist search does not return } \text{NN}_P(q) \text{]} \leq c_0 (k + \ln n_0) \left( \frac{8 \ln 1/\delta}{n_0} \right)^{1/k}
\]

Variant: **spill-trees** (overlapping splits)

\( \beta \quad 1 - \beta \) (RP-tree)

\( 1/2 + \alpha \quad 1/2 + \alpha \) (spill-tree)
Exploiting randomness: RP-trees

**Thm:** [Dasgupta, Sinha’13]
Suppose $p_1, \ldots, p_n \sim \mu$ continuous probability measure in $\mathbb{R}^d$ with doubling dimension $k \geq 2$. Then $\exists c_0 > 0$ s.t. for any $q \in \mathbb{R}^d$ and $\delta < 1/e$, with proba. $\geq 1 - 3\delta$ over the choice of the $p_i$’s:

$$\mathbb{P}_{v,\beta} [\text{defeatist search does not return } \text{NN}_P(q)] \leq c_0(k + \ln n_0) \left( \frac{8 \ln 1/\delta}{n_0} \right)^{1/k}$$

**Variant:** spill-trees (overlapping splits)

Similar behavior

(RP-tree)

(spill-tree)
recall that RP-trees only have a certain probability of success (Monte-Carlo) → Las Vegas version: build and search multiple trees in parallel, keeping the best result (compare distances to query point) (note: number of trees depends on target precision)

Benchmarking

contenders
effect of size on winners

RP-trees vs. other methods on data sets of 100k, 1M and 31M features
Random kd-trees (RP-trees, spill-trees) are fast, scalable and reliable on data with low-dimensional intrinsic structure.
Locality-Sensitive Hashing
Back to the NN problem

Pre-processing input: \( P \)

Query input: \( q \)

Goal: find \( p \in \text{NN}_P(q) \)

**Curse of Dimensionality**: every DS for NN-search has either exponential size or exponential query time (in \( d \)) in the worst case.

→ holds in theory and in practice for exact NN queries [Weber et al. '98]
Back to the $\varepsilon$-NN problem

pre-processing input: $P, \varepsilon$

query input: $q$

goal: find $p \in \text{NN}_P(q, \varepsilon)$

Curse of Dimensionality: every DS for NN-search has either exponential size or exponential query time (in $d$) in the worst case.

→ holds in theory and in practice for exact NN queries [Weber et al. ’98]

→ still holds for approximate queries in decision tree model [Arya et al. ’98]
Back to the $\varepsilon$-NN problem

pre-processing input: $P$, $\varepsilon$

query input: $q$

goal: find $p \in \text{NN}_P(q, \varepsilon)$

**Curse of Dimensionality**: every DS for NN-search has either exponential size or exponential query time (in $d$) in the worst case.

$$(1 + \varepsilon)d(q, P)$$

$\text{NN}_P(q, \varepsilon)$

$\text{NN}_P(q)$

d$(q, P)$

$\text{NN}_P(q, \varepsilon)$

→ holds in theory and in practice for exact NN queries [Weber et al. ’98]

→ still holds for approximate queries in decision tree model [Arya et al. ’98]

→ no longer true in Real-RAM model thanks to LSH [Indyk, Motwani ’98]
Locality-Sensitive Hashing

Comparing elements via hashing:

\[\text{hashCode} : X \rightarrow \mathbb{Z}\]

\[x = y \Rightarrow \text{hashCode}(x) = \text{hashCode}(y)\]

\[x \neq y \Rightarrow \text{hashCode}(x) \neq \text{hashCode}(y)\]

(no collisions hypothesis)
Locality-Sensitive Hashing

Comparing elements via hashing:

\[ \text{hashCode} : X \rightarrow \mathbb{Z} \]

\[ x = y \Rightarrow \text{hashCode}(x) = \text{hashCode}(y) \]

\[ x \neq y \Rightarrow \text{hashCode}(x) \neq \text{hashCode}(y) \]

(no collisions hypothesis)

Metric case \((X, d)\): given \(r > 0\),

\[ d(x, y) \leq r \Rightarrow \text{hashCode}(x) = \text{hashCode}(y) \]

\[ d(x, y) > r \Rightarrow \text{hashCode}(x) \neq \text{hashCode}(y) \]
Locality-Sensitive Hashing

Comparing elements via hashing:

$$\text{hashCode} : X \rightarrow \mathbb{Z}$$

$$x = y \Rightarrow \text{hashCode}(x) = \text{hashCode}(y)$$

$$x \neq y \Rightarrow \text{hashCode}(x) \neq \text{hashCode}(y)$$

(no collisions hypothesis)

Metric case $$(X, d)$$: given $$r > 0$$,

$$d(x, y) \leq r \Rightarrow \text{hashCode}(x) = \text{hashCode}(y)$$

$$d(x, y) > r \Rightarrow \text{hashCode}(x) \neq \text{hashCode}(y)$$

too good to be true → allow for some slack
Locality-Sensitive Hashing

Def: Given \( r_1 < r_2, \ p_1 > p_2 \) and \( U \subset \mathbb{N} \), a family \( \mathcal{F} \) of hash functions \( f : (X, d) \to U \) is \((r_1, r_2, p_1, p_2)\)-sensitive if \( \forall x, y \in X \),

- \( d(x, y) \leq r_1 \Rightarrow P[f(x) = f(y)] \geq p_1 \)
- \( d(x, y) \geq r_2 \Rightarrow P[f(x) = f(y)] \leq p_2 \)

(probability is over a random choice of function according to a given probability distribution over \( \mathcal{F} \))
Locality-Sensitive Hashing

**Def:** Given $r_1 < r_2$, $p_1 > p_2$ and $\mathcal{U} \subset \mathbb{N}$, a family $\mathcal{F}$ of hash functions $f : (X, d) \rightarrow \mathcal{U}$ is \textbf{$(r_1, r_2, p_1, p_2)$-sensitive} if $\forall x, y \in X$,

1. $d(x, y) \leq r_1 \Rightarrow P[f(x) = f(y)] \geq p_1$
2. $d(x, y) \geq r_2 \Rightarrow P[f(x) = f(y)] \leq p_2$

(probability is over a random choice of function according to a given probability distribution over $\mathcal{F}$)

**Example 1:** $(X, d) = (\{0, 1\}^d, d_H)$

→ take $\mathcal{F} = \{f_i\}_{i=1}^d$ where $f_i(b_1 \cdots b_d) = b_i$ | unif. proba. on $\mathcal{F}$

→ $\mathcal{F}$ is $(r, r(1 + \varepsilon), 1 - \frac{r}{d}, 1 - \frac{r(1 + \varepsilon)}{d})$-sensitive for all $r \geq 1$ and $\varepsilon \geq 0$. 


Locality-Sensitive Hashing

**Def:** Given $r_1 < r_2$, $p_1 > p_2$ and $\mathcal{U} \subset \mathbb{N}$, a family $\mathcal{F}$ of hash functions $f : (X, d) \rightarrow \mathcal{U}$ is $(r_1, r_2, p_1, p_2)$-**sensitive** if $\forall x, y \in X$,

- $d(x, y) \leq r_1 \Rightarrow \mathbb{P}[f(x) = f(y)] \geq p_1$
- $d(x, y) \geq r_2 \Rightarrow \mathbb{P}[f(x) = f(y)] \leq p_2$

(probability is over a random choice of function according to a given probability distribution over $\mathcal{F}$)

**Example 1:** $(X, d) = (\{0, 1\}^d, d_H)$

$\rightarrow$ take $\mathcal{F} = \{f_i\}_{i=1}^d$ where $f_i(b_1 \cdots b_d) = b_i$ | unif. proba. on $\mathcal{F}$

$\rightarrow$ $\mathcal{F}$ is $(r, r(1 + \varepsilon), 1 - \frac{r}{d}, 1 - \frac{r(1+\varepsilon)}{d})$-sensitive for all $r \geq 1$ and $\varepsilon \geq 0$.

**proof:** $\forall x, y$, $\mathbb{P}_f[f(x) = f(y)] = \frac{d - d_{H}(x, y)}{d} = 1 - \frac{d_{H}(x, y)}{d}$
Locality-Sensitive Hashing

**Def:** Given $r_1 < r_2$, $p_1 > p_2$ and $\mathcal{U} \subseteq \mathbb{N}$, a family $\mathcal{F}$ of hash functions $f : (X, d) \rightarrow \mathcal{U}$ is $(r_1, r_2, p_1, p_2)$-sensitive if $\forall x, y \in X$,

- $d(x, y) \leq r_1 \Rightarrow \mathbb{P}[f(x) = f(y)] \geq p_1$
- $d(x, y) \geq r_2 \Rightarrow \mathbb{P}[f(x) = f(y)] \leq p_2$

(probability is over a random choice of function according to a given probability distribution over $\mathcal{F}$)

**Example 2:** $(X, d) = (\mathbb{R}^d, \| \cdot \|_2)$

→ take $\mathcal{F} = \{ f_{\mathbf{v}, b} \}_{\mathbf{v} \in \mathbb{R}^d, b \in [0, r]}$ where $f_{\mathbf{v}, b}(x) = \left\lfloor \frac{x \cdot \mathbf{v} + b}{r} \right\rfloor$

→ choose $\mathbf{v} = (v_1, \cdots, v_d)$ with $v_i \sim \mathcal{N}(0, 1)$, and $b$ uniformly in $[0, r]$

→ $\mathcal{F}$ is $(r, r(1 + \varepsilon), p_1, p_2)$ sensitive for $p_1 = g(1)$ and $p_2 = g(1 + \varepsilon)$, where $g(\kappa) = 1 - 2\operatorname{cdf}(-\frac{r}{\kappa}) - \frac{2}{\sqrt{2\pi \frac{r}{\kappa}}}(1 - e^{-\frac{r^2}{2\kappa^2}})$

\[ \begin{align*}
\text{cumulative density func. of normal distrib.}
\end{align*} \]
Locality-Sensitive Hashing

**Def:** Given \( r_1 < r_2, \ p_1 > p_2 \) and \( U \subset \mathbb{N} \), a family \( \mathcal{F} \) of hash functions \( f : (X, d) \rightarrow U \) is \((r_1, r_2, p_1, p_2)\)-sensitive if \( \forall x, y \in X \),

\[
\begin{align*}
\bullet & \ d(x, y) \leq r_1 \Rightarrow \mathbb{P}[f(x) = f(y)] \geq p_1 \\
\bullet & \ d(x, y) \geq r_2 \Rightarrow \mathbb{P}[f(x) = f(y)] \leq p_2
\end{align*}
\]

(probability is over a random choice of function according to a given probability distribution over \( \mathcal{F} \))

**Lemma** [Johnson, Lindenstrauss 84]:

For any dimensions \( 0 < k < d \) there is a probability distribution \( \mu \) over the projections \( \mathbb{R}^d \rightarrow \mathbb{R}^k \) such that, given any set \( P \) of \( n \) points in \( \mathbb{R}^d \) and any \( \varepsilon \in (0, 1) \) with \( k > 10 \ln n/\varepsilon^2 \), a projection \( \pi : \mathbb{R}^d \rightarrow \mathbb{R}^k \) sampled at random from \( \mu \) satisfies w.h.p.

\[
\forall p, q \in P, \ (1 - \varepsilon)\|p - q\| \leq \|\pi(p) - \pi(q)\| \leq (1 + \varepsilon)\|p - q\|
\]
Locality-Sensitive Hashing

**Def:** Given $r_1 < r_2$, $p_1 > p_2$ and $U \subset \mathbb{N}$, a family $\mathcal{F}$ of hash functions $f : (X, d) \to U$ is $(r_1, r_2, p_1, p_2)$-**sensitive** if $\forall x, y \in X$,

- $d(x, y) \leq r_1 \Rightarrow \mathbb{P}[f(x) = f(y)] \geq p_1$
- $d(x, y) \geq r_2 \Rightarrow \mathbb{P}[f(x) = f(y)] \leq p_2$

(probability is over a random choice of function according to a given probability distribution over $\mathcal{F}$)

→ **General idea:**

- choose $k$-dimensional vector of random functions $(f_1, \cdots, f_k) \in \mathcal{F}^k$
- pre-process $P$ by hashing its points into the corresponding hash table
- given $q \in X$, hash $q$ and choose collision with smallest distance
Locality-Sensitive Hashing

**Def:** Given \( r_1 < r_2, p_1 > p_2 \) and \( U \subset \mathbb{N} \), a family \( \mathcal{F} \) of hash functions \( f : (X, d) \to U \) is \((r_1, r_2, p_1, p_2)\)-sensitive if \( \forall x, y \in X \),

- \( d(x, y) \leq r_1 \Rightarrow P[f(x) = f(y)] \geq p_1 
- d(x, y) \geq r_2 \Rightarrow P[f(x) = f(y)] \leq p_2 

(probability is over a random choice of function according to a given probability distribution over \( \mathcal{F} \))

→ **General idea:**
- choose \( k \)-dimensional vector of random functions \( (f_1, \cdots, f_k) \in \mathcal{F}^k \)
- pre-process \( P \) by hashing its points into the corresponding hash table
- given \( q \in X \), hash \( q \) and choose collision with smallest distance

→ **Technical detail:** family works only for fixed \( r_1, r_2 \)

\( \rightarrow \) fix \( r_1 = r \) and \( r_2 = r(1 + \varepsilon) \), and solve \((r, \varepsilon)\)-NN query
The \((r, \varepsilon)-NN\) problem (PLEB)

**Goal:** pre-process \(P\) such that, for any query point \(q\),
- if \(d(q, P) \leq r\) then answer YES and return some \(p \in \text{NN}_P(q, r, \varepsilon)\),
- if \(d(q, P) > r(1 + \varepsilon)\) then answer NO,
- else \((r < d(q, P) \leq r(1 + \varepsilon))\) give any of the above answers.
The \((r, \varepsilon)\)-NN problem (PLEB)

**Goal:** pre-process \(P\) such that, for any query point \(q\),
- if \(d(q, P) \leq r\) then answer YES and return some \(p \in \text{NN}_P(q, r, \varepsilon)\),
- if \(d(q, P) > r(1 + \varepsilon)\) then answer NO,
- else \((r < d(q, P) \leq r(1 + \varepsilon))\) give any of the above answers.

**Step 1:** boost the sensitivity of the hash family

\[ \mathcal{G} = \{ g = (f_1, \cdots, f_k) \in \mathcal{F}^k \mid f_1, \cdots, f_k \text{ chosen randomly in } \mathcal{F} \} \]
The $(r, \varepsilon)$-NN problem (PLEB)

**Goal:** pre-process $P$ such that, for any query point $q$,
- if $d(q, P) \leq r$ then answer YES and return some $p \in \text{NN}_P(q, r, \varepsilon)$,
- if $d(q, P) > r(1 + \varepsilon)$ then answer NO,
- else ($r < d(q, P) \leq r(1 + \varepsilon)$) give any of the above answers.

**Step 1:** boost the sensitivity of the hash family

$$G = \{g = (f_1, \cdots, f_k) \in F^k | \text{ } f_1, \cdots, f_k \text{ chosen randomly in } F\}$$

$$\rightarrow \forall x, y, d(x, y) \leq r \Rightarrow \mathbb{P}[g(x) = g(y)] \geq p^k_1 \text{ (coords. are independent)}$$

$$d(x, y) > r(1 + \varepsilon) \Rightarrow \mathbb{P}[g(x) = g(y)] \leq p^k_2$$
The \((r, \varepsilon)\)-NN problem (PLEB)

**Goal:** pre-process \(P\) such that, for any query point \(q\),
- if \(d(q, P) \leq r\) then answer YES and return some \(p \in \text{NN}_P(q, r, \varepsilon)\),
- if \(d(q, P) > r(1 + \varepsilon)\) then answer NO,
- else \((r < d(q, P) \leq r(1 + \varepsilon))\) give any of the above answers.

**Step 2:** pre-process the data points

- choose \(\tau\) random functions \(g_1, \cdots, g_\tau\) from boosted hash family \(\mathcal{G}\),
- initialise \(\tau\) hash tables \(H_1, \cdots, H_\tau\)
- \(\forall i = 1, \cdots, \tau\), hash every point \(p \in P\) into \(H_i\) using \(g_i(p)\) as the key
- use chaining to store the points in the same entry of \(H_i\)
The $(r, \varepsilon)$-NN problem (PLEB)

**Goal:** pre-process $P$ such that, for any query point $q$,
- if $d(q, P) \leq r$ then answer YES and return some $p \in \text{NN}_P(q, r, \varepsilon)$,
- if $d(q, P) > r(1 + \varepsilon)$ then answer NO,
- else ($r < d(q, P) \leq r(1 + \varepsilon)$) give any of the above answers.

**Step 2:** pre-process the data points
The $(r, \varepsilon)$-NN problem (PLEB)

**Goal:** pre-process $P$ such that, for any query point $q$,

- if $d(q, P) \leq r$ then answer YES and return some $p \in \text{NN}_P(q, r, \varepsilon)$,
- if $d(q, P) > r(1 + \varepsilon)$ then answer NO,
- else ($r < d(q, P) \leq r(1 + \varepsilon)$) give any of the above answers.

**Step 2:** pre-process the data points

```
H_1  H_2  ...  H_\tau
```

(3) (2) (1) (7) (4) (3) (8) (2) (6) (3)

(1) (2) (3) (4) (1) (1) (3) (4) (2)

(5) (1) (1) (9) (3)

(6)

(use chaining)
The \((r, \varepsilon)-\text{NN problem (PLEB)}\)

**Goal:** pre-process \(P\) such that, for any query point \(q\),
- if \(d(q, P) \leq r\) then answer YES and return some \(p \in \text{NN}_P(q, r, \varepsilon)\),
- if \(d(q, P) > r(1 + \varepsilon)\) then answer NO,
- else \((r < d(q, P) \leq r(1 + \varepsilon))\) give any of the above answers.

**Step 3:** hash the query point using the \(g_i\)
The \((r, \varepsilon)\)-NN problem (PLEB)

**Goal:** pre-process \(P\) such that, for any query point \(q\),
- if \(d(q, P) \leq r\) then answer YES and return some \(p \in \text{NN}_P(q, r, \varepsilon)\),
- if \(d(q, P) > r(1 + \varepsilon)\) then answer NO,
- else \((r < d(q, P) \leq r(1 + \varepsilon))\) give any of the above answers.

**Step 3:** hash the query point using the \(g_i\)

if there are more than \(2\tau\) collisions then return NO

else let \(p_1, \cdots, p_l\) be the collisions \((l \leq 2\tau)\):

\[\text{if some } p_j \text{ is such that } d(p_j, q) \leq r(1 + \varepsilon) \text{ then return YES and } p_j\]

else return NO
The \((r, \varepsilon)\)-NN problem (PLEB)

**Goal:** pre-process \(P\) such that, for any query point \(q\),
- if \(d(q, P) \leq r\) then answer YES and return some \(p \in \text{NN}_P(q, r, \varepsilon)\),
- if \(d(q, P) > r(1 + \varepsilon)\) then answer NO,
- else \((r < d(q, P) \leq r(1 + \varepsilon))\) give any of the above answers.

**Analysis in a nutshell:**
- test \(d(p_j, q) \leq r(1 + \varepsilon) \Rightarrow \) answer is NO whenever \(d(q, P) > r(1 + \varepsilon)\)
The \((r, \varepsilon)\)-NN problem (PLEB)

**Goal:** pre-process \(P\) such that, for any query point \(q\),
- if \(d(q, P) \leq r\) then answer YES and return some \(p \in \text{NN}_P(q, r, \varepsilon)\),
- if \(d(q, P) > r(1 + \varepsilon)\) then answer NO,
- else \((r < d(q, P) \leq r(1 + \varepsilon))\) give any of the above answers.

**Analysis in a nutshell:**
- test \(d(p_j, q) \leq r(1 + \varepsilon) \Rightarrow \) answer is NO whenever \(d(q, P) > r(1 + \varepsilon)\)
- if \(\exists p \in P\) s.t. \(d(p, q) \leq r\) then:
  - for any fixed \(i \in \{1, \cdots, \tau\}\), \(\mathbb{P}[p \text{ collides with } q \text{ in } H_i] \geq p_1^k\)
  \[\Rightarrow \mathbb{P}[p \text{ collides with } q \text{ in } H_i \text{ for at least one } i] \geq 1 - (1 - p_1^k)^\tau\]
The \((r, \varepsilon)\)-NN problem (PLEB)

**Goal:** pre-process \(P\) such that, for any query point \(q\),
- if \(d(q, P) \leq r\) then answer YES and return some \(p \in \text{NN}_P(q, r, \varepsilon)\),
- if \(d(q, P) > r(1 + \varepsilon)\) then answer NO,
- else \((r < d(q, P) \leq r(1 + \varepsilon))\) give any of the above answers.

**Analysis in a nutshell:**
- test \(d(p_j, q) \leq r(1 + \varepsilon) \Rightarrow\) answer is NO whenever \(d(q, P) > r(1 + \varepsilon)\)
- if \(\exists p \in P\) s.t. \(d(p, q) \leq r\) then:
  - for any fixed \(i \in \{1, \cdots, \tau\}\), \(\mathbb{P}[p\text{ collides with }q\text{ in }H_i] \geq p_1^k\)
    \(\Rightarrow \mathbb{P}[p\text{ collides with }q\text{ in }H_i\text{ for at least one }i] \geq 1 - (1 - p_1^k)\tau\)

  - \(\mathbb{P}[\text{total number of collisions} > 2\tau] \leq \frac{np_2^k}{2}\) (Markov’s inequality)
The \((r, \varepsilon)\)-NN problem (PLEB)

**Goal:** pre-process \(P\) such that, for any query point \(q\),
- if \(d(q, P) \leq r\) then answer YES and return some \(p \in \text{NN}_P(q, r, \varepsilon)\),
- if \(d(q, P) > r(1 + \varepsilon)\) then answer NO,
- else \(r < d(q, P) \leq r(1 + \varepsilon)\) give any of the above answers.

**Analysis in a nutshell:**
- test \(d(p_j, q) \leq r(1 + \varepsilon) \Rightarrow \text{answer is NO whenever } d(q, P) > r(1 + \varepsilon)\)
- if \(\exists p \in P\) s.t. \(d(p, q) \leq r\) then:
  - for any fixed \(i \in \{1, \cdots, \tau\}\), \(\mathbb{P}[p\text{ collides with } q\text{ in } H_i] \geq p_1^k\)
  \[\Rightarrow \mathbb{P}[p\text{ collides with } q\text{ in } H_i\text{ for at least one } i] \geq 1 - (1 - p_1^k)^\tau\]
- \(\mathbb{P}[\text{total number of collisions} > 2\tau] \leq \frac{np_2^k}{2}\) (Markov’s inequality)
  \[\Rightarrow \mathbb{P}[\text{answer is YES}] \geq 1 - (1 - p_1^k)^\tau - \frac{np_2^k}{2}\) (union bound)
The \((r, \varepsilon)-\text{NN}\) problem (\textsc{PLEB})

**Goal:** pre-process \(P\) such that, for any query point \(q\),
- if \(d(q, P) \leq r\) then answer YES and return some \(p \in \text{NN}_P(q, r, \varepsilon)\),
- if \(d(q, P) > r(1 + \varepsilon)\) then answer NO,
- else \((r < d(q, P) \leq r(1 + \varepsilon))\) give any of the above answers.

**Analysis in a nutshell:**
- test \(d(p_j, q) \leq r(1 + \varepsilon) \Rightarrow\) answer is NO whenever \(d(q, P) > r(1 + \varepsilon)\)

▶ in Hamming cube: let \(k := \log_{1/p_2} n \leq \frac{d}{1+\varepsilon} \ln n\) and \(\tau := n^{\ln p_1/\ln p_2} \leq n^{1/(1+\varepsilon)}\):

\[
\begin{align*}
\Pr[\text{success}] &\geq \frac{1}{2} - \frac{1}{e} \\
\text{query time in } O(\tau (k + d)) &\subseteq O \left(\frac{d}{1+\varepsilon} n^{1/(1+\varepsilon)} \ln n\right) \\
\Rightarrow \quad \mathbb{P}[\text{answer is YES}] &\geq 1 - \left(1 - p_1^k\right)^\tau - \frac{n p_2^k}{2} \text{ (union bound)}
\end{align*}
\]
From \((r, \varepsilon)\)-NN to \(\varepsilon\)-NN

- Hamming cube \((X, d) = (\{0, 1\}^d, d_H)\)

**Observation:** inter-point distances lie within \(\{0, 1, 2, \cdots, d\}\)
From $(r, \varepsilon)$-NN to $\varepsilon$-NN

- Hamming cube $(X, d) = (\{0, 1\}^d, d_H)$

**Observation:** inter-point distances lie within $\{0, 1, 2, \cdots, d\}$

→ solve case $d_H(q, P) = 0$ independently (use lexicographical sorting)
→ take geometric sequence $r_0 = 1, r_1 = 1 + \varepsilon, \cdots, r_j = (1 + \varepsilon)^j, \cdots$
→ for $j = 0$ to $\lceil \log_{1+\varepsilon} d \rceil = O\left(\frac{1}{\varepsilon} \log d\right)$, solve $(r_j, r_{j+1})$-NN query
→ let $j_l$ be the lowest $j$ s.t. the answer to $(r_j, r_{j+1})$-MM query is YES
→ return the output point of the $(r_{j_l}, r_{j_l+1})$-NN query
→ if no YES answer, return output of case $d_H(q, P) = 0$
From $(r, \varepsilon)$-NN to $\varepsilon$-NN

- Hamming cube $(X, d) = (\{0, 1\}^d, d_H)$

**Observation:** inter-point distances lie within $\{0, 1, 2, \cdots, d\}$

$\rightarrow$ query time in $O\left(\frac{d \log d}{\varepsilon(1+\varepsilon)} \cdot n^{1/1+\varepsilon} \log n\right)$

(becomes $O\left(\frac{d^2}{1+\varepsilon} \cdot n^{1/1+\varepsilon} \log n\right)$ if arithmetic sequence is used)
From \((r, \varepsilon)-\text{NN}\) to \(\varepsilon-\text{NN}\)

- **Hamming cube** \((X, d) = (\{0, 1\}^d, d_H)\)

**Observation:** inter-point distances lie within \(\{0, 1, 2, \cdots, d\}\)

\[\rightarrow \text{query time in } O\left( \frac{d \log d}{\varepsilon(1+\varepsilon)} n^{1/1+\varepsilon} \log n \right)\]

(becomes \(O\left( \frac{d^2}{1+\varepsilon} n^{1/1+\varepsilon} \log n \right)\) if arithmetic sequence is used)

**Observation:** deterministically, \(r_{j_l} \geq d_H(q, P)/(1 + \varepsilon)\)

\[\Rightarrow \text{output} \in NNP(q, \varepsilon(2 + \varepsilon)) \text{ iff LSH data structure works for } j = j_l\]

\[\Rightarrow \mathbb{P}[\text{success}] \geq \frac{1}{2} - \frac{1}{e}\]
From \((r, \varepsilon)\)-NN to \(\varepsilon\)-NN

- General case: use **hierarchical clustering tree** [Har-Peled’01]
  - consider geometric sequences of scales as before
  - cluster data points in order to bound the lengths of the sequences
From \((r, \varepsilon)-\text{NN}\) to \(\varepsilon-\text{NN}\)

- General case: use **hierarchical clustering tree** [Har-Peled'01]
  - consider geometric sequences of scales as before
  - cluster data points in order to bound the lengths of the sequences
From \((r, \varepsilon)\)-NN to \(\varepsilon\)-NN

- General case: use hierarchical clustering tree [Har-Peled’01]
  - consider geometric sequences of scales as before
  - cluster data points in order to bound the lengths of the sequences
From \((r, \varepsilon)\)-NN to \(\varepsilon\)-NN

- General case: use **hierarchical clustering tree** [Har-Peled'01]
  - consider geometric sequences of scales as before
  - cluster data points in order to bound the lengths of the sequences

Assign \(P_v \subseteq P\) and \([r_v, R_v]\) to each node \(v\)
From \((r, \varepsilon)\)-NN to \(\varepsilon\)-NN

- General case: use **hierarchical clustering tree** [Har-Peled'01]
  - consider geometric sequences of scales as before
  - cluster data points in order to bound the lengths of the sequences

Assign \(P_v \subseteq P\) and \([r_v, R_v]\) to each node \(v\)

\(\varepsilon\)-NN query:
  - traverse down the tree along one root-leaf path
From \((r, \varepsilon)\)-NN to \(\varepsilon\)-NN

- General case: use **hierarchical clustering tree** [Har-Peled'01]
  - consider geometric sequences of scales as before
  - cluster data points in order to bound the lengths of the sequences

Assign \(P_v \subseteq P\) and \([r_v, R_v]\) to each node \(v\)

\(\varepsilon\)-NN query:
- traverse down the tree along one root-leaf path
- at each visited node \(v\), perform \((r_v, \varepsilon)\)-NN and \((R_v, \varepsilon)\)-NN queries
  \(\rightarrow\) decide if \(d(q, P) \in [r_v, R_v]\) or not
From $(r, \varepsilon)$-NN to $\varepsilon$-NN

- General case: use **hierarchical clustering tree** [Har-Peled'01]
  - consider geometric sequences of scales as before
  - cluster data points in order to bound the lengths of the sequences

Assign $P_v \subseteq P$ and $[r_v, R_v]$ to each node $v$

$\varepsilon$-NN query:
- traverse down the tree along one root-leaf path
- at each visited node $v$, perform $(r_v, \varepsilon)$-NN and $(R_v, \varepsilon)$-NN queries
  → decide if $d(q, P) \in [r_v, R_v]$ or not
  → if so, locate $d(q, P)$ in $[r_v, R_v]$
  → if not, recurse into one child only
From \((r, \varepsilon)\)-NN to \(\varepsilon\)-NN

- General case: use **hierarchical clustering tree** [Har-Peled'01]
  - consider geometric sequences of scales as before
  - cluster data points in order to bound the lengths of the sequences

Assign \(P_v \subseteq P\) and \([r_v, R_v]\) to each node \(v\)

\(\varepsilon\)-NN query:
- traverse down the tree along one root-leaf path
- at each visited node \(v\), perform
  \((r_v, \varepsilon)\)-NN and \((R_v, \varepsilon)\)-NN queries
  \(\rightarrow\) decide if \(d(q, P) \in [r_v, R_v]\) or not
  \(\rightarrow\) if so, locate \(d(q, P)\) in \([r_v, R_v]\)
  \(\rightarrow\) if not, recurse into one child only

\(O\left(\frac{1}{\varepsilon} \log \frac{n}{\varepsilon}\right)\) \((r, \varepsilon)\)-NN queries per \(\varepsilon\)-NN query
Take-Home Messages

• (Approximate) NN search requires an exponential amount of resources (space/time) in the algebraic comparison tree model [Arya et al. 98].

• Using random hashing allows to beat the curse of dimensionality.

• The price to pay is that algorithms become almost linear in practice, a trade-off must be found.

• The complexity of the exact NN search problem is not fully understood.

→ what about reverse NN search? [Cheong et al. 09], [Arthur, O. 10], ...