Case Study 2: Document Retrieval

Locality-Sensitive Hashing Random Projections for NN Search

Machine Learning for Big Data
CSE547/STAT548, University of Washington
Sham Kakade
April 18, 2017

Announcements:

- HW2 posted
- Project Milestones
  - Start early
  - Lit. review (>= 3 papers read carefully)
  - First rounds of experiments
- Today:
  - Review: ball trees, cover trees
  - Today: locality sensitive hashing

Case Study 2: Document Retrieval

Task Description: Finding Similar Items

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

Where is FAST similarity search important?

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

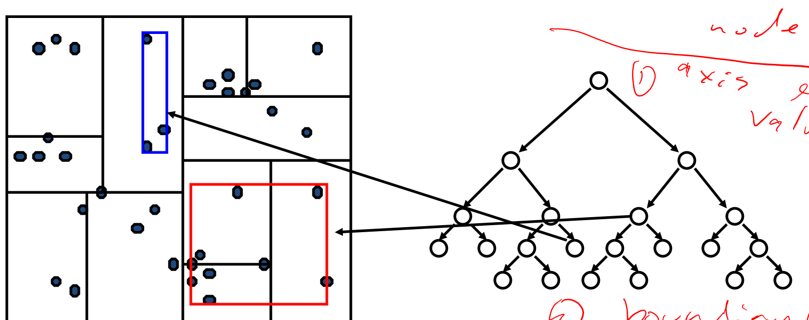
1-Nearest Neighbor

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

KD-Tree Construction

maintain at all nodes

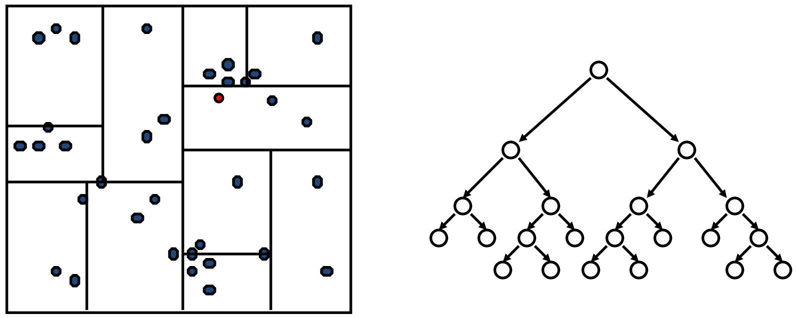
① axis & value



② bounding box

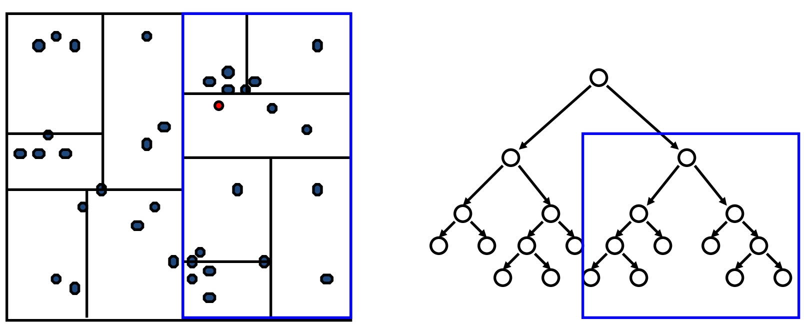
- Keep one additional piece of information at each node:
  - The (tight) bounds of the points at or below this node.

Nearest Neighbor with KD Trees



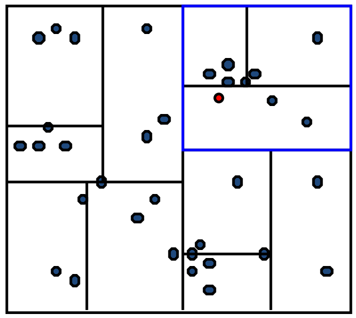
- Traverse the tree looking for the nearest neighbor of the query point.

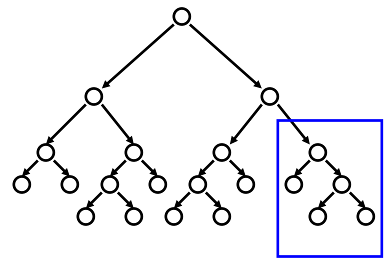
Nearest Neighbor with KD Trees



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

Nearest Neighbor with KD Trees

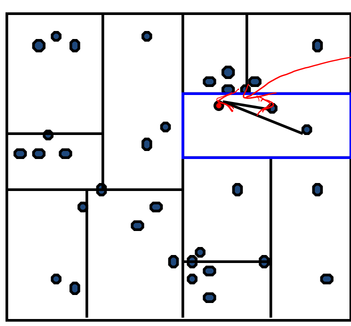


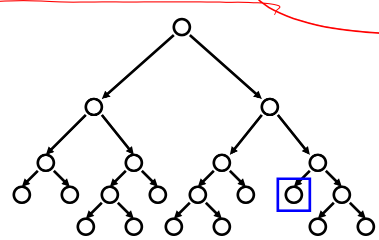


- Examine nearby points first:
  - Explore branch of tree closest to the query point first.

Nearest Neighbor with KD Trees

guess at NN
along with a distance



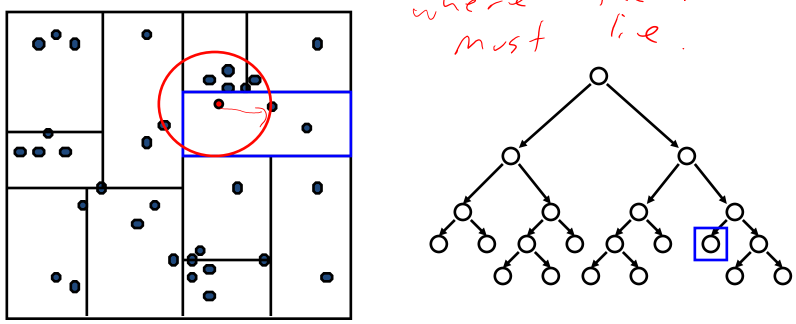


does the NN have to be in this box??

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

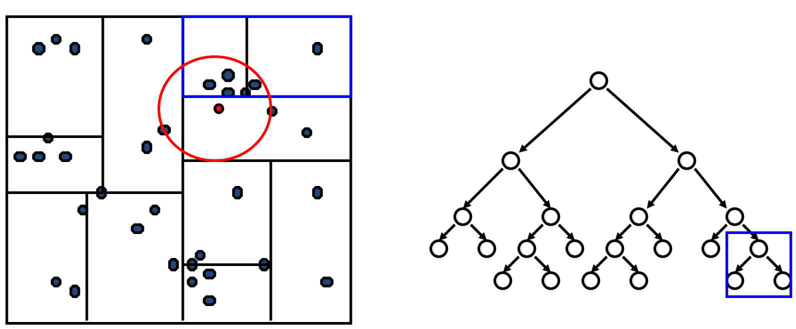
Nearest Neighbor with KD Trees

guaranteed region where the NN must lie.



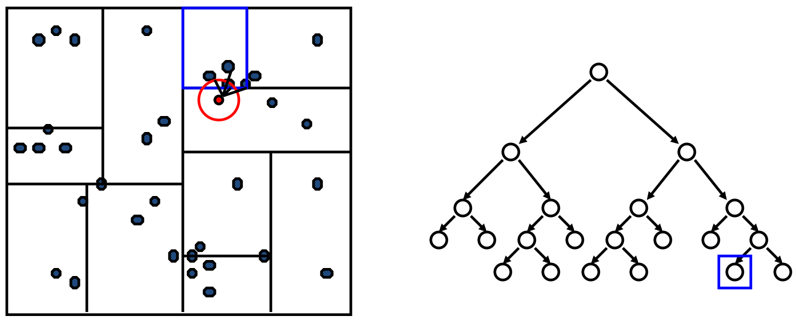
- When we reach a leaf node:
  - Compute the distance to each point in the node.

Nearest Neighbor with KD Trees



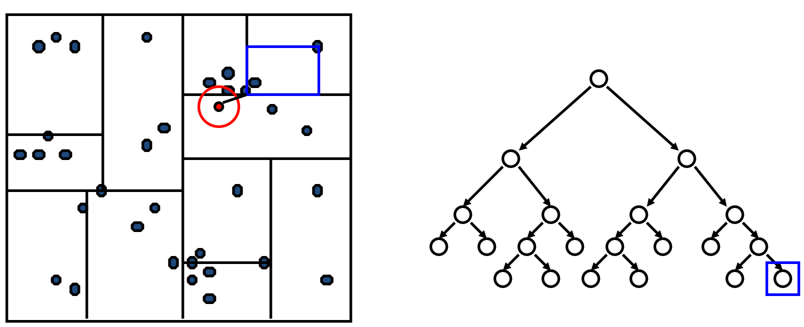
- Then backtrack and try the other branch at each node visited

Nearest Neighbor with KD Trees



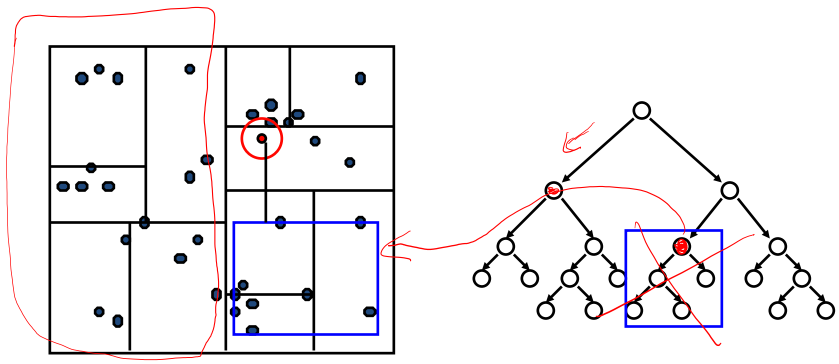
- Each time a new closest node is found, update the distance bound

Nearest Neighbor with KD Trees



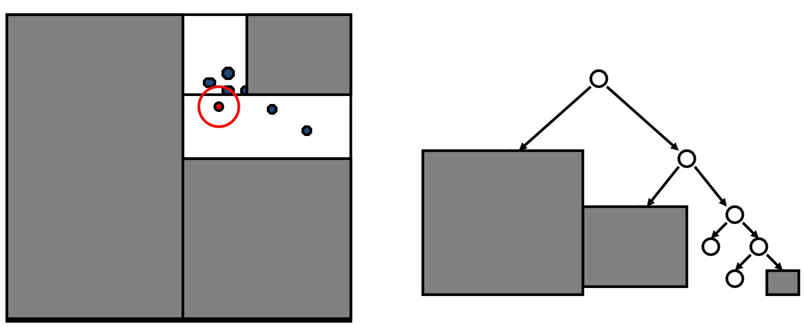
- Using the distance bound and bounding box of each node:
  - Prune parts of the tree that could NOT include the nearest neighbor

Nearest Neighbor with KD Trees



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

Nearest Neighbor with KD Trees



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

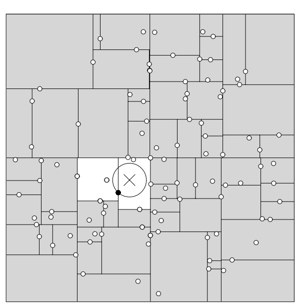
Complexity

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

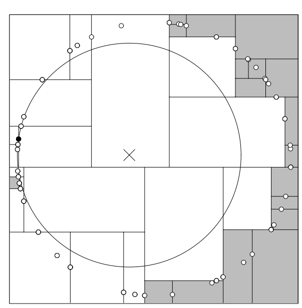
    $O(\log N) \longleftrightarrow O(N)$
- Under some assumptions on distribution of points, we get O(log*N*) but exponential in *d* (see citations in reading)

Complexity

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



$O(\log N)$



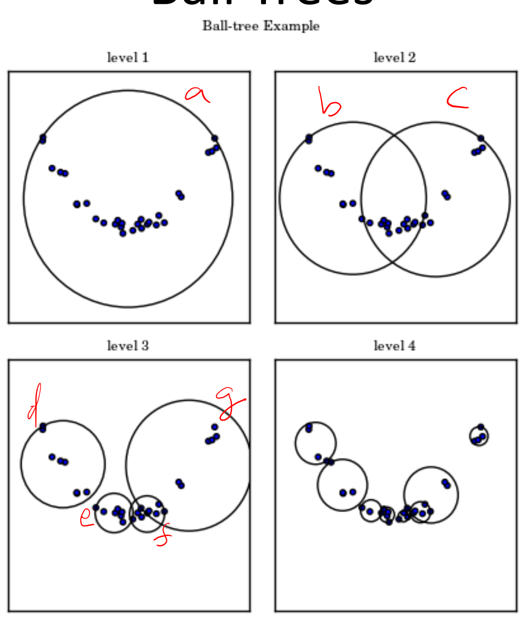
$O(N)$

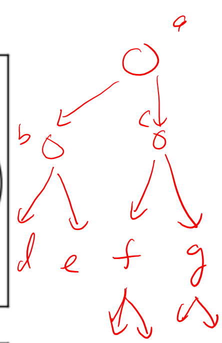
What about NNs searches in high dimensions?

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

 19

Ball Trees





20

Ball Tree Construction

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

Ball Tree Search

- Given point x, how do find its nearest neighbor quickly?
- Approach:
  - Start: follow a greedy path through the tree
  - Backtrack and prune: rule out other paths based on the triange inequality
    - (just like in KD-trees)
- How good is it?
  - Guarantees: in high dimensions?? → worst case complexity is bad.
  - Practice: one of best for exact NN-searches

Cover trees

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

 23

Intrinsic Dimension

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

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

- Can we relax this idea to get at the "intrinsic" dimension?

$$\forall R,p \quad \frac{\#\text{ points in } Ball_{2R}(p)}{\#\text{ points in } Ball_R(p)} \le 2^{\tilde{d}}$$

  - This is the "doubling" dimension: $\tilde{d}$

 24

Cover trees: data structure

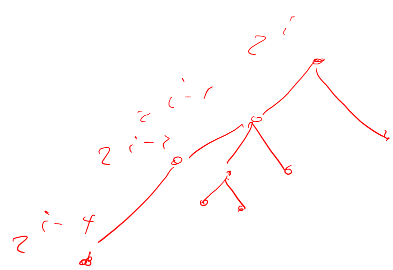
- Ball Trees: each node had associated
  - Center:
  - (tight) Radius:
  - Points:

many children

- Cover trees:
  - Center: a point in the dataset
  - (tight) Radius: 2^i
  - Points: keep points in the ball.

Cover trees construction

- Construct tree by inserting one point at time
- Follow a point down. Does it "fit" in each ball?
  - Yes: Insert.
  - No:



Cover Tree Complexity

- Construction $2^{\tilde{D}} N$
  - Size:
  - Construction time: $2^{\tilde{D}} N$
- 1-NN query:
  - Check all paths with triangle.
  - Maximum time complexity: $2^{\tilde{D}} \log N$
- Under assumptions that "doubling dimension" is $\tilde{D}$.
- Provable method for datastructure construction.

Wrapping Up – Important Points

kd-trees

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

Nearest Neighbor Search

- Distance metric and data representation are crucial to answer returned

For both…

- High dimensional spaces are hard!
  - Number of kd-tree searches can be exponential in dimension
    - Rule of thumb… $N >> 2^d$… Typically useless.
  - Ball Trees and Cover Trees more effective here!

What you need to know

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

Case Study 2: Document Retrieval

Locality-Sensitive Hashing Random Projections for NN Search

Machine Learning for Big Data
CSE547/STAT548, University of Washington
Sham Kakade
April 18, 2017

Intuition (?): NN in 1D and Sorting

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

or How do we sort ?

approx.

- Pre-processing time:

O(N)

bins

sorting

O(N log N)

- Query time:

O(1)

O(log N)

Using Hashing to Find Neighbors

- KD-trees are cool, but…
  - Non-trivial to implement efficiently
  - Problems with high-dimensional data
- Approximate neighbor finding…
  - Don't find exact neighbor, but that's OK for many apps, especially with Big Data
- What if we could use hash functions:
  - Hash elements into buckets:

h('...immigration policy...')

h('...election...')

h('...NBA...')

want "similar" points to fall into same bucket

  - Look for neighbors that fall in same bucket as x:

find the hash(query point) bucket and check in bucket

- But, by design…

What to hash?

- Before: we were hashing 'words'/strings
- Remember, we can think of hash functions abstractly:

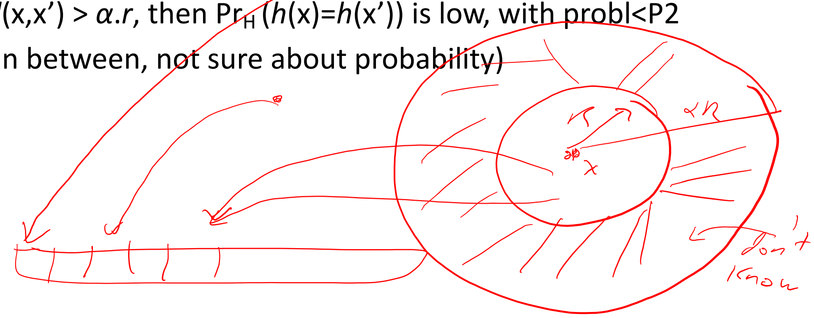
$h : \mathcal{X} \rightarrow \{1, \ldots m\}$

'keys' values

- Idea of LSH: try to has similar items into same buckets and different items into different buckets

Locality Sensitive Hashing (LSH)

- Suppose we have a set of functions H and a distribution over these functions.
- A LSH family H satisfies (for example), for some similarity function d, for r>0, α>1, 1>P1,P2>0:
  – $d(x,x') \leq r$, then $Pr_H(h(x)=h(x'))$ is high, with prob>P1
  – $d(x,x') > \alpha.r$, then $Pr_H(h(x)=h(x'))$ is low, with probl<P2
  – (in between, not sure about probability)

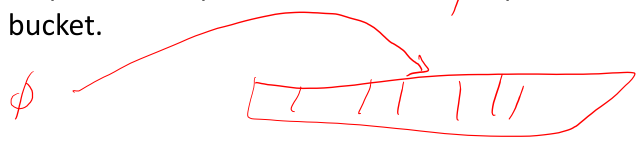


LSH: basic paradigm

$h \sim \mathcal{H}$

- Step 0: pick a 'simple' way to construct LSH functions
- Step 1: (amplification) make another hash function by repeating this construction

  $\phi(x) = (h_1(x), \dots, h_k(x))$
- Step 2: the output of this function ϕ specifies the index to a bucket.



- Step 3: use multiple hash tables. for recall, search for similar items in the same buckets. $\phi^{(1)} \dots \phi^{(L)}$

  have L hash tables.

Example: hashing binary strings

$x \in \{0,1\}^d$

- Suppose x and x' are binary strings
- Hamming distance metric $|x-x'|_1$
- What is a simple family of hash function?

  $h^{(i)}(x) = x_i$
- Suppose $|x-x'|$ are R close, what is P1?

  $P_1 = 1 - R/d$
- Suppose $|x-x'|>$ ~~cR~~ αR, what is P2?

  $P_2 = 1 - \alpha R/d$

Amplification

each ϕ gives us one hash table

- Improving P1 and P2
- Now the hash function is:

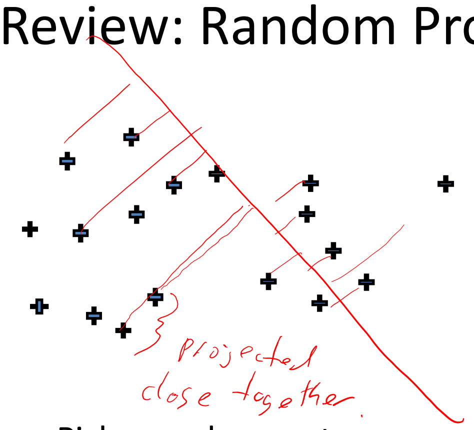
$$\phi^{(1)} = \left(h_1^{(1)}(x),\ h_2(x), \dots h_K^{(1)}(x)\right)$$

$$\vdots$$

$$\phi^{(L)} = \left(h_1^{(L)}(x) \dots h_K^{(L)}(x)\right)$$

- The choice ~~m~~ K is a parameter.

Review: Random Projection Illustration



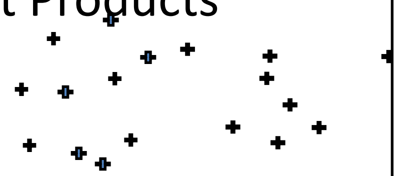
$v \sim N(0, I)$

- Pick a random vector *v*:
  - Independent Gaussian coordinates

$y(x) = v \cdot x$

- Preserves separability for most vectors
  - Gets better with more random vectors

Multiple Random Projections: Approximating Dot Products



- Pick m random vectors v(i):
  - Independent Gaussian coordinates

$v_1, \dots v_m \quad v_i \sim \mathcal{N}(0, I)$

- Approximate dot products:
  - Cheaper, e.g., learn in smaller *m* dimensional space

$\phi(x) = (v_1 \cdot x, \dots v_m \cdot x)$

$E[(v_i \cdot x)(v_i \cdot x')]$

$= x \cdot x'$

- Only need logarithmic number of dimensions!
  - N data points, approximate dot-product within ε>0:

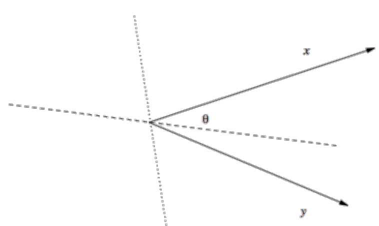
$$m = \mathcal{O}\left(\frac{\log N}{\epsilon^2}\right)$$

$|x \cdot x'|$

$= |\phi(x) \cdot \phi(x')| \pm \varepsilon$

- But all sparsity is lost

LSH Example function: Sparser Random Projection for Dot Products



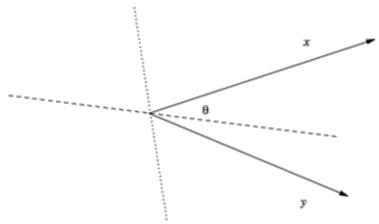
$\vec{v}_i \sim \mathcal{N}(0, I)$

- Pick random vector *v*
- Simple 0/1 projection: h(x) = $\text{sgn}(\vec{v}_i \cdot \vec{x})$
- Now, each vector is approximated by a single bit

$\phi(x) = (h_1(x), \dots h_k(x))$

- ~~This is an LSH function, though with poor α and P2~~

LSH Example continued: Amplification with multiple projections



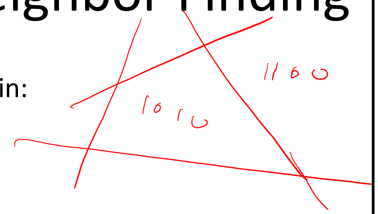
- Pick random vectors $v^{(i)}$
- Simple 0/1 projection: $\phi_i(x)$ =

- Now, each vector is approximated by a bit-vector

- Dot-product approximation:

LSH for Approximate Neighbor Finding

- Very similar elements fall in exactly same bin:



- And, nearby bins are also nearby:

- Simple neighbor finding with LSH:
  - For bins *b* of increasing hamming distance to $\phi(x)$:
    - Look for neighbors of x in bin *b*

  - Stop when run out of time

- Pick m such that $N/2^m$ is "smallish" + use multiple tables

LSH: using multiple tables

ϕ_1

ϕ_2

$\vdots$

ϕ_L $\quad h_1^{(L)}(x) \mid \cdots \mid h_k^{(L)}(x)$

NN complexities

	Query time	Space used	Preprocessing time
Vornoi	$O(2^d \log n)$	$O(n^{d/2})$	$O(n^{d/2})$
Kd-tree	$O(2^d \log n)$	$O(n)$	$O(n \log n)$
LSH	$O(n^\rho \log n)$	$O(n^{1+\rho})$	$O(n^{1+\rho} \log n)$

Hash Kernels: Even Sparser LSH for Learning

- Two big problems with random projections:
  - Data is sparse, but random projection can be a lot less sparse
  - You have to sample m huge random projection vectors
    - And, we still have the problem with new dimensions, e.g., new words
- **Hash Kernels**: Very simple, but powerful idea: combine sketching for learning with random projections
- Pick 2 hash functions:
  - *h* : Just like in Count-Min hashing
  - ξ : Sign hash function
    - Removes the bias found in Count-Min hashing (see homework)
- Define a "kernel", a projection ϕ for x:

Hash Kernels, Random Projections and Sparsity

$$\phi_i(\mathbf{x}) = \sum_{j:h(j)=i} \xi(j)\mathbf{x}_j$$

- Hash Kernel as a random projection:
- What is the random projection vector for coordinate *i* of ϕ_i:
- Implicitly define projection by *h* and ξ, so no need to compute apriori and automatically deals with new dimensions
- Sparsity of ϕ, if x has *s* non-zero coordinates:

What you need to know

- Locality-Sensitive Hashing (LSH): nearby points hash to the same or nearby bins
- LSH uses random projections
  - Only $O(\log N/\varepsilon^2)$ vectors needed
  - But vectors and results are not sparse
- Use LSH for nearest neighbors by mapping elements into bins
  - Bin index is defined by bit vector from LSH
  - Find nearest neighbors by going through bins
- Hash kernels:
  - Sparse representation for feature vectors
  - Very simple, use two hash functions
    - Can even use one hash function, and take least significant bit to define ξ
  - Quickly generate projection $\phi(x)$
  - Learn in projected space