

## 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

©Sham Kakade 2017

1

## Announcements:

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

©Kakade 2017

## 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

©Sham Kakade 2017

3

### Where is FAST similarity search important?

- web search
- image search
- sky maps / location
- shazam / song identification
- physics simulations
  - robotics

4

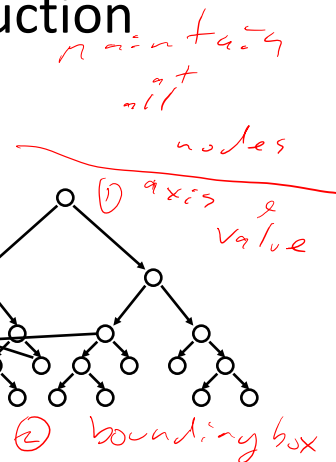
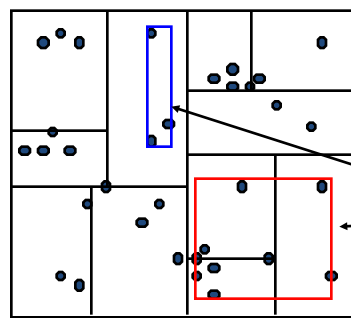
# 1-Nearest Neighbor

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

©Sham Kakade 2017

5

# KD-Tree Construction

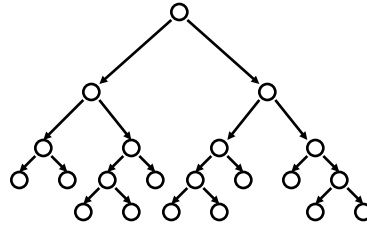
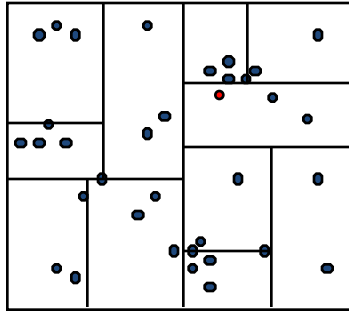


- Keep one additional piece of information at each node:
  - The (tight) bounds of the points at or below this node.

©Sham Kakade 2017

6

## Nearest Neighbor with KD Trees

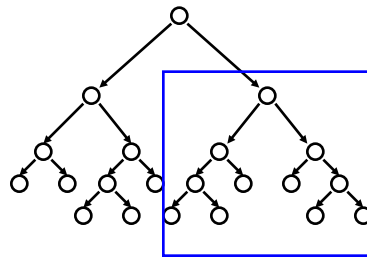
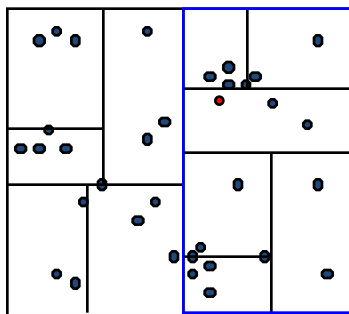


- Traverse the tree looking for the nearest neighbor of the query point.

©Sham Kakade 2017

7

## Nearest Neighbor with KD Trees

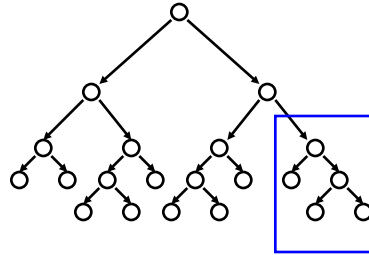
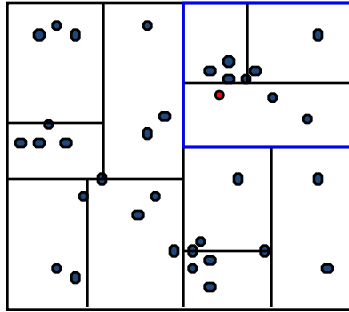


- Examine nearby points first:
  - Explore branch of tree closest to the query point first.

©Sham Kakade 2017

8

## Nearest Neighbor with KD Trees

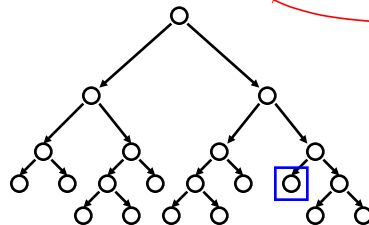
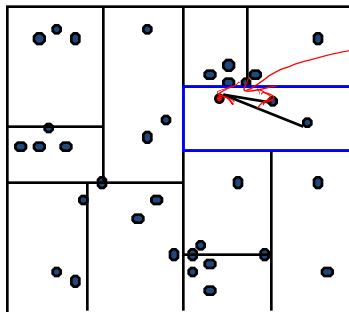


- Examine nearby points first:
  - Explore branch of tree closest to the query point first.

©Sham Kakade 2017

9

## Nearest Neighbor with KD Trees



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

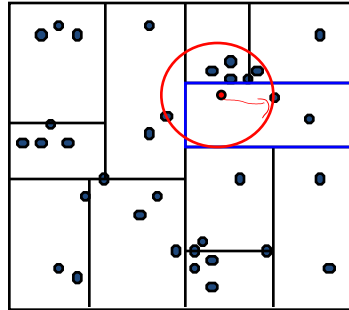
©Sham Kakade 2017

10

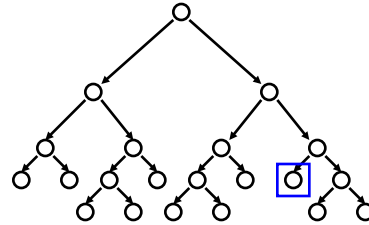
*guess at NN  
along with distance*

*Does the NN  
have to be in  
this box??*

## Nearest Neighbor with KD Trees



*guaranteed region  
where the NN  
must lie.*

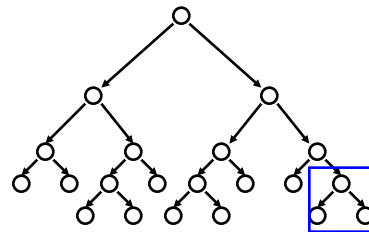
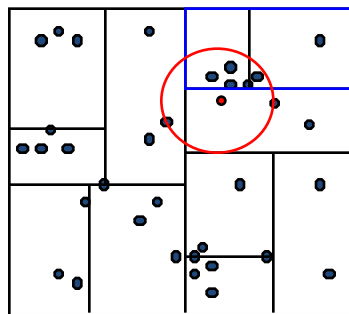


- When we reach a leaf node:
  - Compute the distance to each point in the node.

©Sham Kakade 2017

11

## Nearest Neighbor with KD Trees

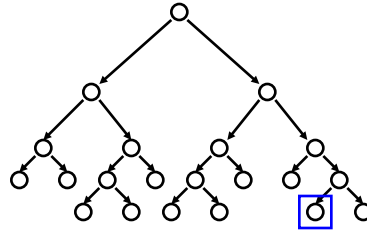
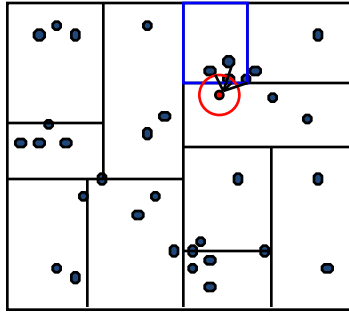


- Then backtrack and try the other branch at each node visited

©Sham Kakade 2017

12

## Nearest Neighbor with KD Trees

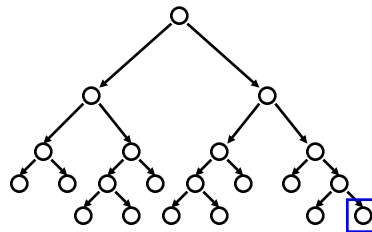
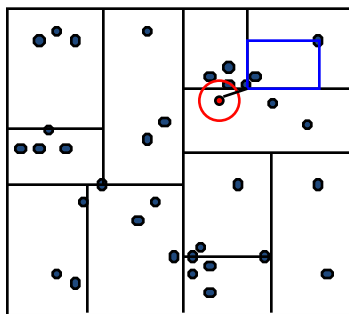


- Each time a new closest node is found, update the distance bound

©Sham Kakade 2017

13

## Nearest Neighbor with KD Trees

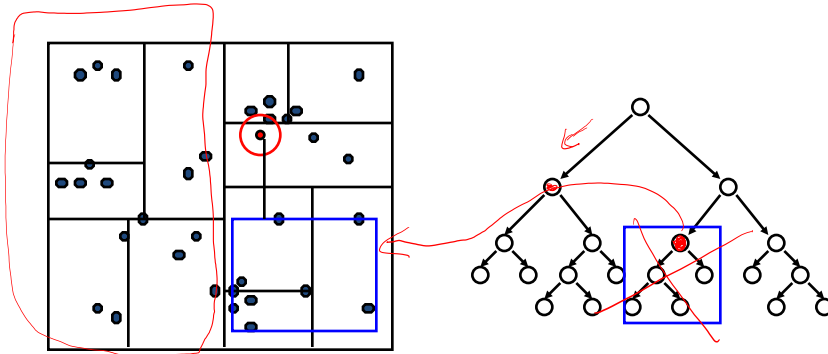


- Using the distance bound and bounding box of each node:
  - Prune parts of the tree that could NOT include the nearest neighbor

©Sham Kakade 2017

14

## Nearest Neighbor with KD Trees

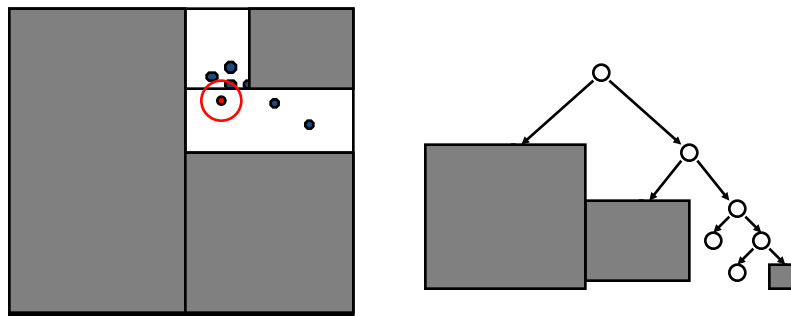


- Using the distance bound and bounding box of each node:
  - Prune parts of the tree that could NOT include the nearest neighbor

©Sham Kakade 2017

15

## 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

©Sham Kakade 2017

16



## Complexity

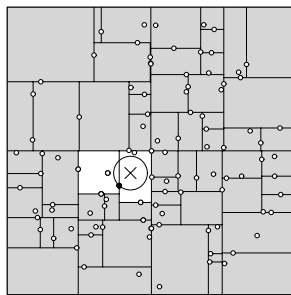
- For (nearly) balanced, binary trees...
- Construction
  - Size:  $O(N)$
  - Depth:  $O(\log N)$  (if can balance)
  - Median + send points left right:
  - Construction time:  $O(N \log N)$
- 1-NN query
  - Traverse down tree to starting point:  $O(\log N)$  (leaf node)
  - Maximum backtrack and traverse:  $O(N)$
  - Complexity range:  $O(\log N) \leftrightarrow O(N)$
- Under some assumptions on distribution of points, we get  $O(\log N)$  but exponential in  $d$  (see citations in reading)

©Sham Kakade 2017

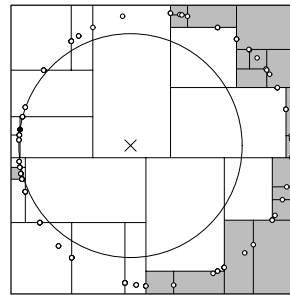
17

## Complexity

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



$O(\log N)$



$O(N)$

©Sham Kakade 2017

18

## 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**

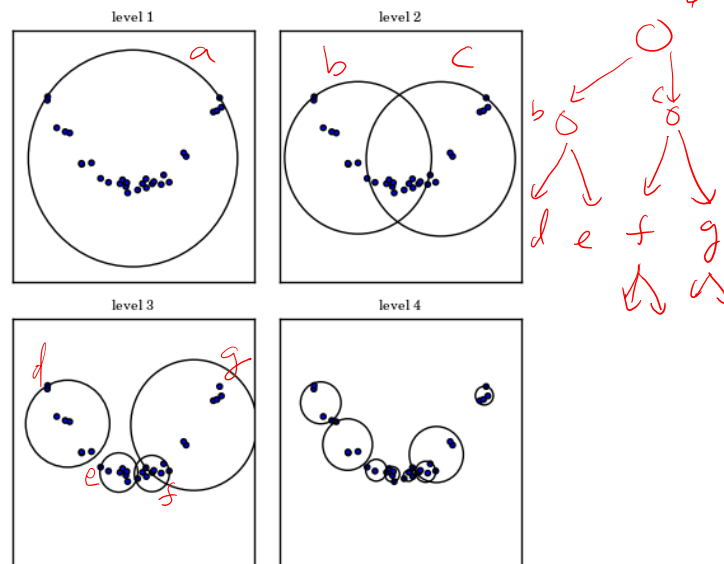
□ New ideas: ball trees and cover trees

©Sham Kakade 2017

19

## Ball Trees

Ball-tree Example



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

©Sham Kakade 2017

21

## 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 triangle inequality
    - (just like in KD-trees)
- How good is it?
  - Guarantees: *in high dimensions??*
  - Practice: *worst case complexity is bad. one of best for exact NN-searches*

©Sham Kakade 2017

22

## 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: *explicit the structure in the data*

©Sham Kakade 2017

23

## Intrinsic Dimension

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

$$\frac{\text{Vol}(\text{Ball}_{2R})}{\text{Vol}(\text{Ball}_R)} = 2^d$$

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

□ This is the “doubling” dimension:

©Sham Kakade 2017

24

## Cover trees: data structure

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

©Sham Kakade 2017

25

## Cover Tree Complexity

- Construction
  - Size:
  - Construction time:
- 1-NN query:
  - Check all paths with triangle.
  - Maximum time complexity:
  
- Under assumptions that “doubling dimension” is  $D$ .
- Provable method for datastructure construction.

©Sham Kakade 2017

26

## 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...  $N \gg 2^d$ ... Typically useless.
  - Ball Trees and Cover Trees more effective here!

©Sham Kakade 2017

27

## 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  $d$

©Sham Kakade 2017

28

## 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

©Sham Kakade 2017

29

## Intuition (?): NN in 1D and Sorting

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

or How do we sort?

- Pre-processing time:

$O(N)$

|||||

- Query time:

$O(1)$

sorting

$O(N \log N)$

$O(\log N)$

©Sham Kakade 2017

30

## 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:
    - Look for neighbors that fall in same bucket as x:
- But, by design...

©Sham Kakade 2017

31

## What to hash?

- Before: we were hashing 'words'/strings
- Remember, we can think of hash functions abstractly:
- Idea of LSH: try to hash similar items into same buckets and different items into different buckets

©Sham Kakade 2017

32



## 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  $r > 0$ ,  $\alpha > 1$ ,  $1 > P_1, P_2 > 0$ :
  - $d(x, x') \leq r$ , then  $\Pr_H(h(x)=h(x'))$  is high, with  $\text{prob} > P_1$
  - $d(x, x') > \alpha r$ , then  $\Pr_H(h(x)=h(x'))$  is low, with  $\text{prob} < P_2$
  - (in between, not sure about probability)

©Sham Kakade 2017

33

## LSH: basic paradigm

- Step 0: pick a 'simple' way to construct LSH functions
- Step 1: (amplification) make another hash function by repeating this construction
- Step 2: the output of this function specifies the index to a bucket.
- Step 3: use multiple hash tables. for recall, search for similar items in the same buckets.

©Sham Kakade 2017

34

## Example: hashing binary strings

- Suppose  $x$  and  $x'$  are binary strings
- Hamming distance metric  $|x-x'|$
- What is a simple family of hash function?
- Suppose  $|x-x'|$  are  $R$  close, what is  $P1$ ?
- Suppose  $|x-x'| > cR$ , what is  $P2$ ?

©Sham Kakade 2017

35

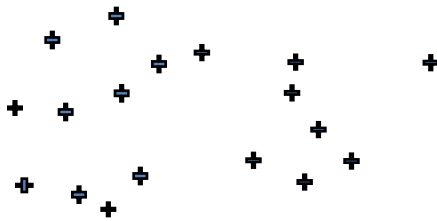
## Amplification

- Improving  $P1$  and  $P2$
- Now the hash function is:
- The choice  $m$  is a parameter.

©Sham Kakade 2017

36

## Review: Random Projection Illustration



- Pick a random vector  $v$ :
  - Independent Gaussian coordinates
- Preserves separability for most vectors
  - Gets better with more random vectors

©Sham Kakade 2017

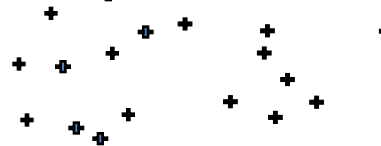
37

## Multiple Random Projections: Approximating Dot Products

- Pick  $m$  random vectors  $v(i)$ :
  - Independent Gaussian coordinates
- Approximate dot products:
  - Cheaper, e.g., learn in smaller  $m$  dimensional space
- Only need logarithmic number of dimensions!
  - $N$  data points, approximate dot-product within  $\epsilon > 0$ :

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

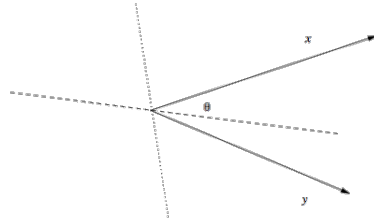
- But all sparsity is lost



©Sham Kakade 2017

38

## LSH Example function: Sparser Random Projection for Dot Products

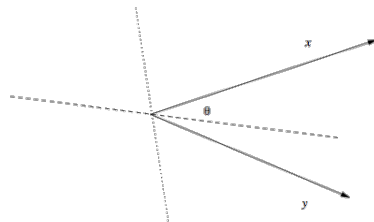


- Pick random vector  $v$
- Simple 0/1 projection:  $h(x) =$
- Now, each vector is approximated by a single bit
- This is an LSH function, though with poor  $\alpha$  and  $P_2$

©Sham Kakade 2017

39

## 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:

©Sham Kakade 2017

40

## 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

©Sham Kakade 2017

41

## LSH: using multiple tables

©Sham Kakade 2017

42

## NN complexities

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

©Sham Kakade 2017

43

## 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
  - $\xi$ : Sign hash function
    - Removes the bias found in Count-Min hashing (see homework)
- Define a “kernel”, a projection  $\phi$  for  $x$ :

©Sham Kakade 2017

44

## 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  $\phi$ :
- Implicitly define projection by  $h$  and  $\xi$ , so no need to compute apriori and automatically deals with new dimensions
- Sparsity of  $\phi$ , if  $x$  has  $s$  non-zero coordinates:

©Sham Kakade 2017

45

## What you need to know

- **Locality-Sensitive Hashing (LSH)**: nearby points hash to the same or nearby bins
- LSH uses **random projections**
  - Only  $O(\log N/\epsilon^2)$  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  $\xi$
  - Quickly generate projection  $\phi(x)$
  - **Learn in projected space**

©Sham Kakade 2017

46