Case Study 2: Document Retrieval

Parallelization in ML.

Clustering Documents

Machine Learning for Big Data
CSE547/STAT548, University of Washington
Sham Kakade
April 20, 2017

Announcements:

add more lectures

- HW2 posted
- Project Milestones

- Shameless plug for my talk
  - Talk: Accelerating Stochastic Gradient Descent
  - Next Tue at 1:30 in CSE 303
  - It's a very promising directions....

- Today:
  - Review: locality sensitive hashing
  - Today: clustering and map-reduce

& parallelization

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

 3

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)

 4

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:

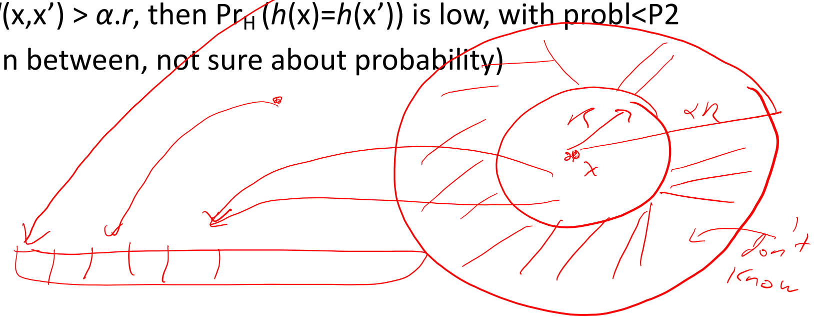
$$h : \mathcal{X} \rightarrow \{1, \ldots m\}$$

'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:
  - $d(x,x') \le r$, 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)

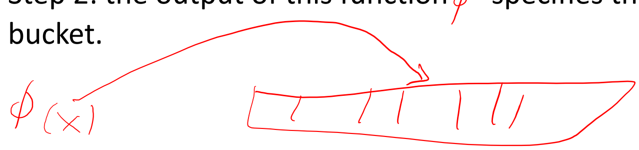


LSH: basic paradigm

$h \sim \mathcal{H}$

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

  $\phi(x) = (h_1(x), \ldots, h_k(x))$
- 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. $\phi^{(1)} \ldots \phi^{(L)}$

  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:

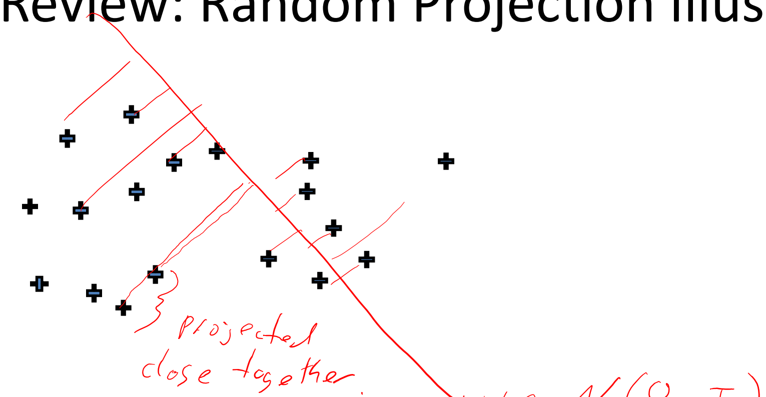
$\phi^{(1)} = (h_1^{(1)}(x), h_2(x), \ldots h_k^{(1)}(x))$

$\vdots$

$\phi^{(L)} = (h_1^{(L)}(x) \ldots h_k^{(L)}(x))$

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

Review: Random Projection Illustration



projected close together

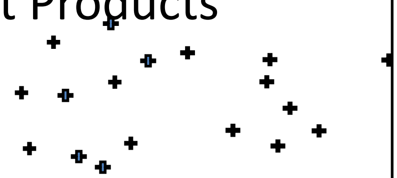
$v \sim \mathcal{N}(0, I)$

- 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

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

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

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

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

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

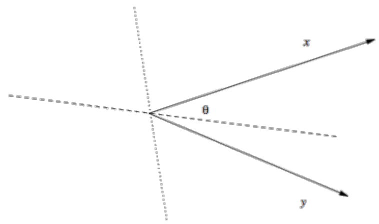
$$m = \mathcal{O}\left(\frac{\log N}{\epsilon^2}\right)$$

$|x - x'|$

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

- But all sparsity is lost

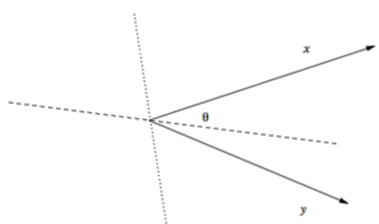
LSH Example function: Sparser Random Projection for Dot Products



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

- Pick random vector *v*
- Simple 0/1 projection: h(x) = sgn $(\vec{v}_i \cdot \vec{x})$
- Now, each vector is approximated by a single bit
  $\phi(x) = (h_1(x), \ldots h_k(x))$
- ~~This is an LSH function, though with poor α and P2~~

LSH Example continued: Amplification with multiple projections

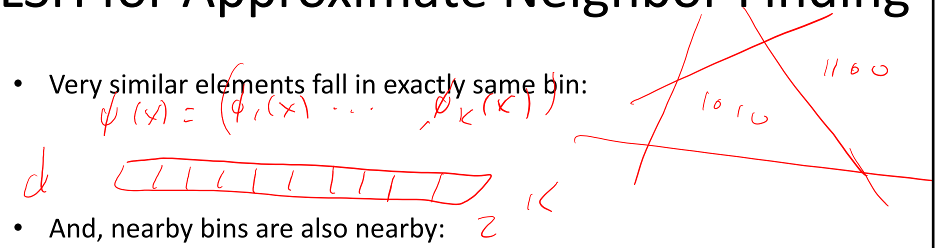


- Pick random vectors $v^{(i)}$
- Simple 0/1 projection: $\phi_i(x)$ =
- Now, each vector is approximated by a bit-vector
- Dot-product approximation:

LSH for Approximate Neighbor Finding

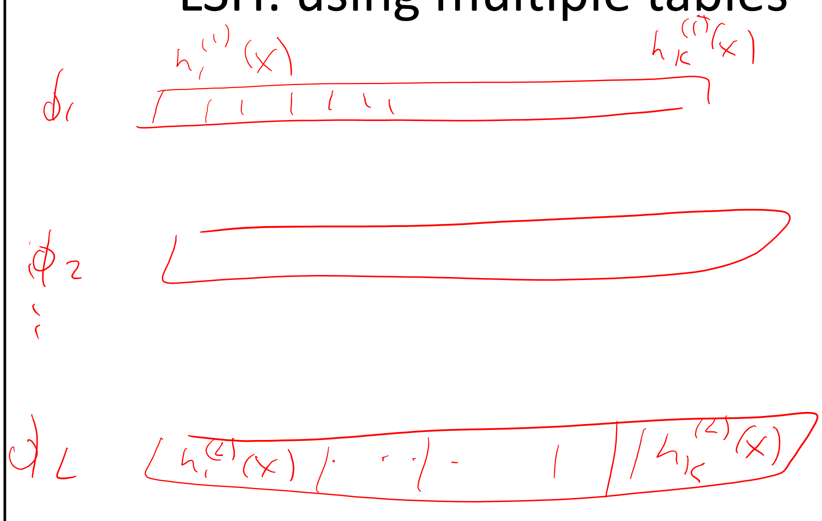
- Very similar elements fall in exactly same bin:

$\phi(x) = (\phi_1(x) \ldots \phi_K(x))$



- And, nearby bins are also nearby:
- Simple neighbor finding with LSH:
  - For bins *b* of increasing hamming distance to ϕ(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



$\|\cdot\|_2$ $\rho \approx \frac{1}{\alpha^2}$

use $|\cdot|_1$ $\rho \approx 1/\alpha$

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	$O(n^{\rho} \log n)$	$O(n^{1+\rho})$	$O(n^{1+\rho} \log n)$
Cover Tree	$O(2^{\hat{d}} \log n)$	$O(\hat{n})$	$O(2^{\hat{d}} n \log n)$

$\hat{d}$ is "intrinsic" dim.

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