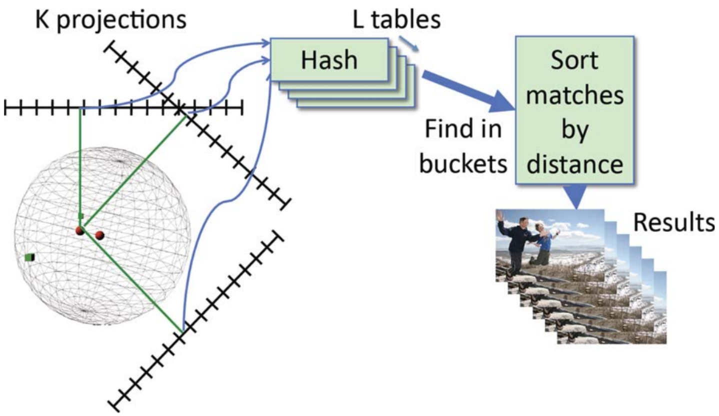
CS60021: Scalable Data Mining

Similarity Search and Hashing

Sourangshu Bhattacharya

MULTI-PROBE LSH

Locality Sensitive Hashing



Picture courtesy Slaney et al.

Given input data, radius r, approx factor c and confident δ

<u>Output:</u> if there is any point at distance $\leq r$ then w.p. $1-\delta$ return one at distance $\leq cr$

<u>Algo:</u> Choose (k, L).

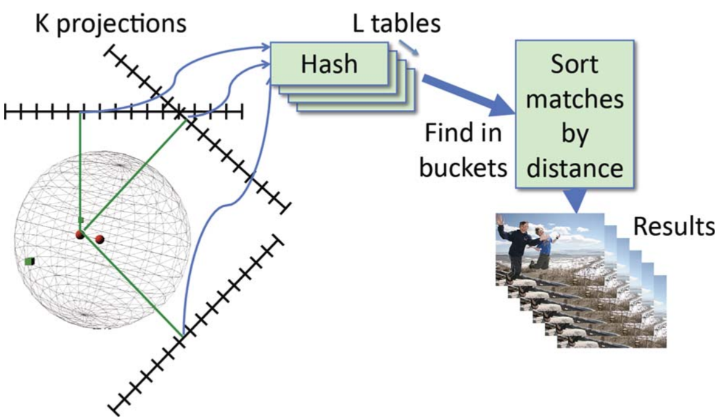
do L times

$\text{iid hash functions}: \{\mathrm{h}_{\mathrm{i}1} \ldots . \mathrm{h}_{\mathrm{ik}}\}$

Create hash table H_i by putting each x in bucket $H_i(x) = \left(h_{i1}(x), \ldots h_{ik}(x)\right)$

Store non-empty buckets in normal hash table

Locality Sensitive Hashing



Picture courtesy Slaney et al.

Given input data, radius r, approx factor c and confident δ

Output: if there is any point at distance $\leq r$ then w.p. $1-\delta$ return one at distance $\leq cr$

Query: Find out all points in buckets $H_1(q) \dots . H_L(q)$ and return ones that are $\leq cr$

Drawbacks

- Trading space with time, strongly super-linear space
  - Even in practice, typically 5-20 times more memory than dataset itself

- Space-time tradeoff mostly practical effective for medium-high dimensions, dense vectors
  - recent advances in ML about dense embeddings

Probing multiple times

- Idea: Can we reduce space while not affecting query time by too much?
  - need to hit buckets that have high probability of the containing the nearest neighbour

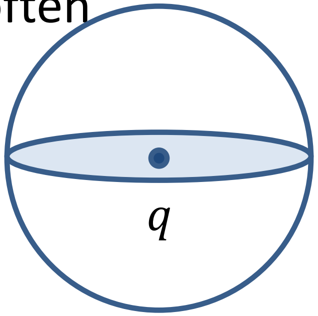
Entropy based LSH

- Assume that we know $R(p, q) =$ distance from query q to nearest neighbour p
  - Buckets are a random partition of the data
  - The success probability of a bucket (i.e. of containing p) depends only on $R(p, q)$
  - Ideally, we can sort the buckets by this probability

Entropy based LSH

[Panigrahy' 06]

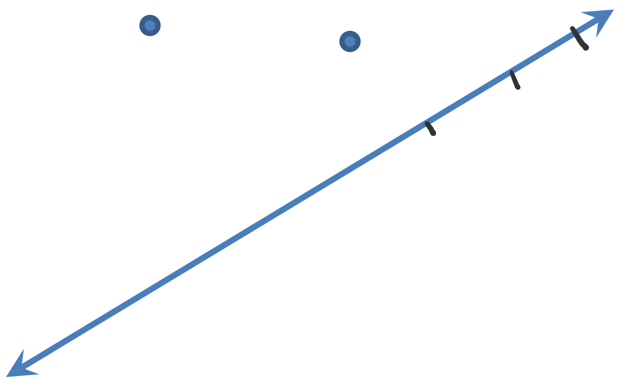
- Elegant way to sample from the success probability distribution
  - Perturb the query point repeatedly and probe
  - Buckets that have high probability should come up often
  - Theoretical guarantee



Multi-probe LSH

- Look at neighbouring buckets!
- Consider LSH for L2

$$h_{v,b}(q) = \left\lfloor \frac{q \cdot v + b}{w} \right\rfloor$$



Multi-probe LSH

- Suppose $k = 3$
- $H_1(q) = (5, 8, 3)$
- We consider buckets that differ in one position, two positions, …

Formalizing

- $\Delta \in \{-1,0,+1\}^k$ be a "perturbation" vector
  - E.g. $\Delta = (-1, 0, +1, +1, 0 \dots . -1)$
  - We get a new hash bucket by doing $H(q) + \Delta$
  - Say Δ has at most S nonzeros
  - Number of possible Δ is:

- Is there a natural way to order these buckets for searching?

Success Probability Estimation

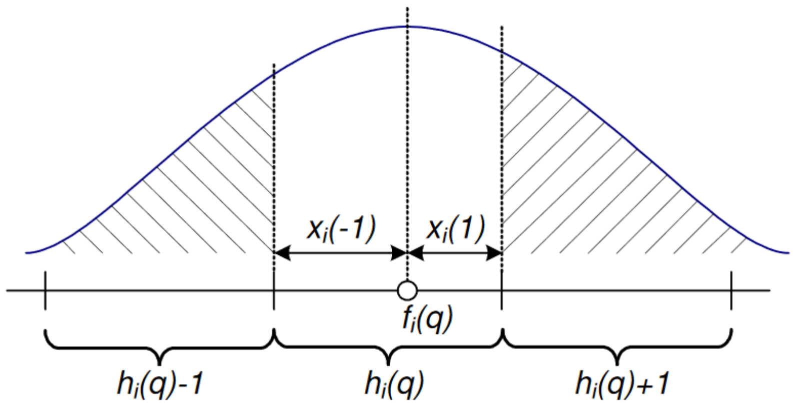


Image from Lv et al.

$f_i(q) = q \cdot v_i + b_i$ be the projection of q

$x_i(+1)$ *and* $x_i(-1)$ be the distance of the projection to the two boundaries

$f_i(q) - f_i(p) \sim N(0, C|p - q|)$ by property of normal distribution

Success Probability Estimation

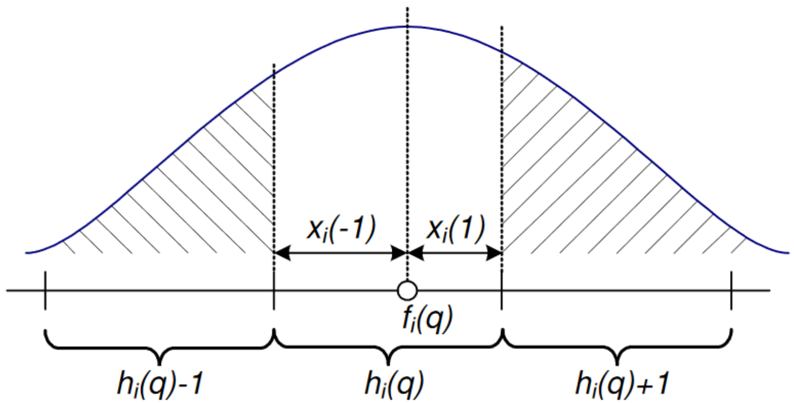


Image from Lv et al.

$x_i(+1)$ *and* $x_i(-1)$ be the distance of the projection to the two boundaries

$f_i(q) - f_i(p) \sim N(0, C|p - q|)$ by property of normal distribution

$$\Pr[h_i(p) = h_i(q) + 1] \approx \exp(-C x_i(+1)^2)$$

Ordering buckets

- If $\Delta = (\delta_1 \ldots \delta_k)$ then

$$\Pr[H(p) = H(q) + \Delta] = \Pr \prod [h_i(q) = h_i(q) + \delta_i]$$

$$\approx \prod \exp(-C x_i(\delta_i)^2) = \exp\left(-C \sum x_i(\delta_i)^2\right)$$

Ex: $\Delta = (+1, 0, -1)$,

Ordering buckets

- Define $score(\Delta) = \sum x_i(\delta_i)^2$
- Lower the score, higher the probability of p being in the bucket

Ordering buckets

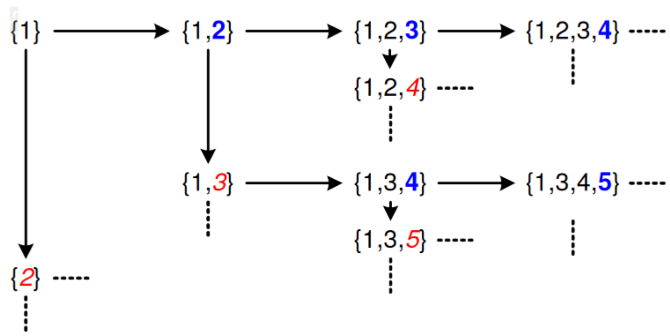
- Define $score(\Delta) = \sum x_i(\delta_i)^2$
- Lower the score, higher the probability of p being in the bucket
- Order the buckets by the score and search them in this order

Query directed ordering

- When a query q arrives
  - Calculate $H(q)$
  - Calculate $\{ x_i(+1)^2, x_i(-1)^2, i = 1 \dots k\}$
  - Sort

Query directed ordering

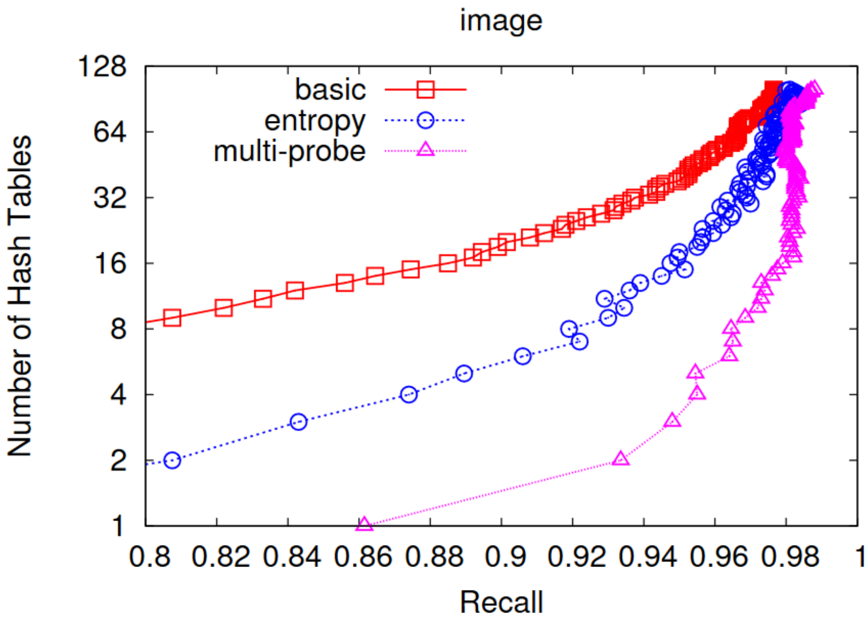
- When a query q arrives
  - Calculate $H(q)$
  - Calculate $\{ x_i(+1)^2, x_i(-1)^2, i = 1 \dots k\}$
  - Sort (call these as $z_1 \leq z_2 \dots \leq z_{2k}$)
- Start with $A = \{1\}$
- Repeatedly do either *shift* or (
  - *shift* replace $\max(A)$ by 1+max(A)
  - *expand* adds 1+max(A) to A

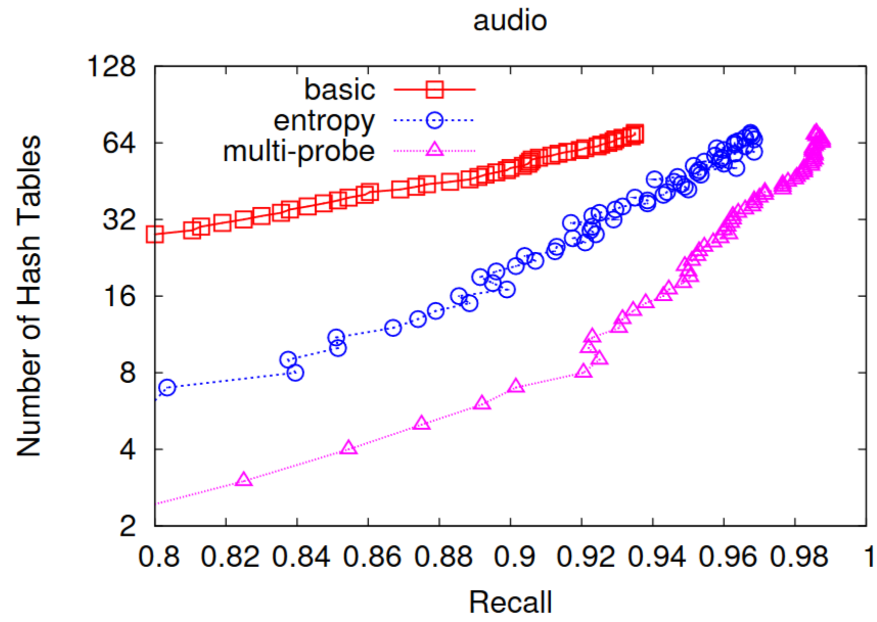


Multiprobe LSH

- Using a min-heap at query time we can use the shift and expand operations to explore all buckets in order
  - Can optimize further
- In practice, will stop after a budget

Experiments





Summary

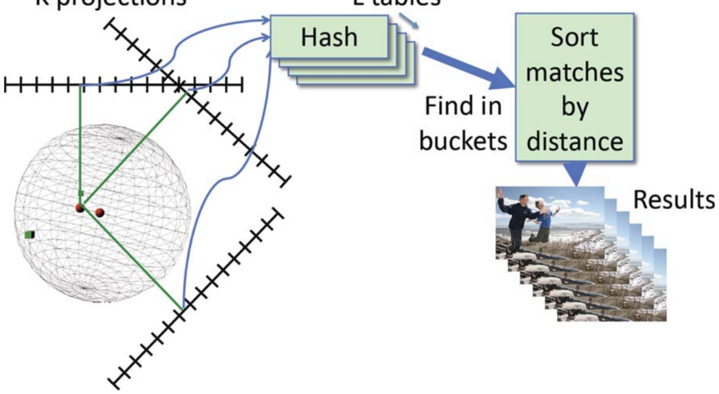
- While LSH is a powerful technique, there are few areas of concern, memory usage among them
- Entropy and Multi-probe LSH are elegant solutions that are useful in practice
  - Shown to be useful in practice, reduce space usage by a factor
  - also form part of the state-of-art LSH system
- Intuition based on idea of probing multiple buckets in a query-dependent manner

References:

- Primary references for this lecture
    - Multi-Probe LSH: Efficient Indexing for High Dimensional Similarity Search. By Qin Lv, William Josephson, Zhe Wang, Moses Charikar, Kai Li, VLDB 2007
    - R. Panigrahy. Entropy based nearest neighbor search in high dimensions. In Proc. of ACM-SIAM Symposium on Discrete Algorithms(SODA), 2006.

LEARNING TO HASH

Locality Sensitive Hashing



Picture courtesy Slaney et al.

Given input data, radius r, approx factor c and confident δ

<u>Output:</u> if there is any point at distance $\leq r$ then w.p. $1-\delta$ return one at distance $\leq cr$

<u>Algo:</u> Choose (k, L).

do L times

iid hash functions : $\{\mathrm{h}_{\mathrm{i}1} \ldots . \mathrm{h}_{\mathrm{ik}}\}$

Create hash table H_i by putting each x in bucket $H_i(x) = \left(h_{i1}(x), \ldots h_{ik}(x)\right)$

Store non-empty buckets in normal hash table

Issues

- Parameters k, L need to be tuned for each domain
- Random directions are meant to create a random partitioning of the dataset
- While useful to guard against “worst case datasets”, we do not exploit the dataset structure

Hashing as binary codes

- Assume points are in Euclidean space

- How can we get binary vectors so that Hamming distance approximates Euclidean distance

Properties of a binary code

- Should be easily computable
- Should preserve distances approximately
- Should have small number of bits
  - the bits should be independent and unbiased

Optimization

- W_{ij} = similarity between i and j
  - Say $W_{ij} = \exp\left(-\frac{|x_i - x_j|^2}{s}\right)$

- y_i = codeword for point i
- $|y_i - y_j|^2$ also equals Hamming(i, j)

Learning codes

- Average hamming distance = $\sum_{ij} W_{ij} \left| y_i - y_j \right|^2$

- $y_i \in \{-1, +1\}^k$
- Each bit should be unbiased: $\sum_i y_i = 0$

- Bits should be uncorrelated $\sum_i y_i y_i^t = I$

Casting as optimization problem

[Waiss et al.]

- Can we solve : minimize $\sum_{ij} W_{ij} \left| y_i - y_j \right|^2$
- subject to
  - $y_i \in \{-1, +1\}^k$
  - $\sum_i y_i = 0$
  - $\sum_i y_i y_i^t = I$

Hardness

- Unfortunately, no!, even for single bit

- Graph partitioning problem: For graph G partition V(G) into two sets A and B such that $|A| = |B|$ and $minimize \sum_{i \in A, j \in B} W_{ij}$

Spectral Relaxation

- $Y = n \times k$ code matrix
- Diagonal $D,\ D_{ii} = \sum_j W_{ij}$
- minimize $\sum_{ij} W_{ij} \left| y_i - y_j \right|^2$ = $trace(Y^t(D\ - W)Y)$
  - $Y^t \cdot 1 = 0$
  - $Y^t\, Y = I$
  - Drop the constraint that Y are in $\{-1, +1\}$

Spectral codes

- The above problem is solved by $Y =$ smallest – k eigenvectors of $D - W$
  - After dropping the one with value 0

- To get codes,
  - We could threshold eigenvectors, but then hard to extend it for query

Eigenvectors

- Assume that the data is coming from some distribution in R^d
  - But estimating this distribution is hard also
  - We could try to interpolate the eigenvectors to query points, under above assumptions, but is computationally expensive (Nystrom extension)

Eigenvectors

- Assume that the data is coming from some distribution in R^d
  - But estimating this distribution is hard also
  - We could try to interpolate the eigenvectors to query points, under above assumptions, but is computationally expensive (Nystrom extension)

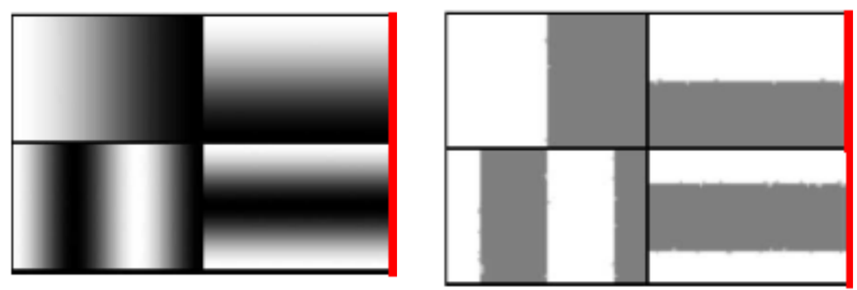
- Assume data distribution is product of uniform distributions
  - Use PCA to find the axes

Eigenfunctions

- Take limit of eigenvectors as $n \to \infty$, and consider the "normalized" similarity matrix (Laplacian)
- Analytical form of Eigenfunctions exists for certain distributions (uniform, Gaussian)
- For uniform

$$\Phi_k(x) = \sin\left(\frac{\pi}{2} + \frac{k\pi}{b-a}x\right)$$

$$\lambda_k = 1 - e^{-\frac{\epsilon^2}{2}\left|\frac{k\pi}{b-a}\right|^2}$$

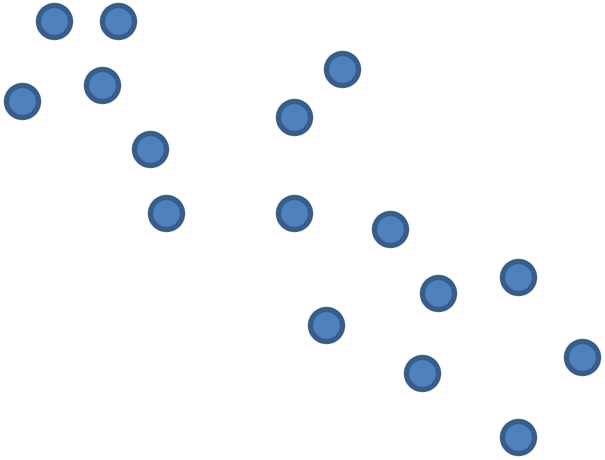


- Constant time calculation for any new point

[Image from Waiss et al]

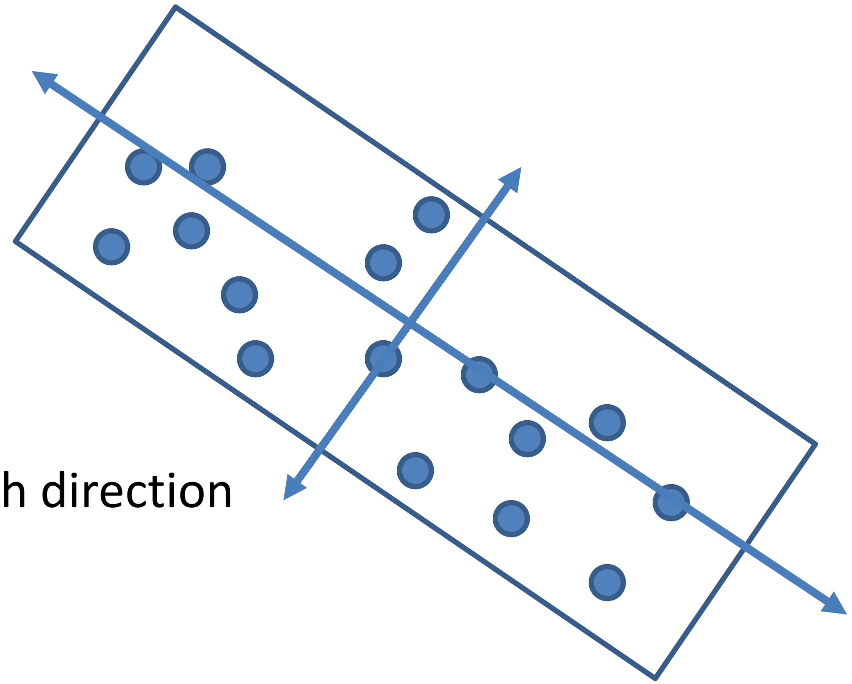
Algorithm

Input: Data $\{x_i\}$, target dimensionality k



Algorithm

Create top k PCA of $D - W$



Gives us top k axes

Find the $[a_i, b_i]$ for each axes

and create $\phi_1(x) \ldots \phi_k(x)$ for each direction

Algorithm

Create top k PCA of $D - W$

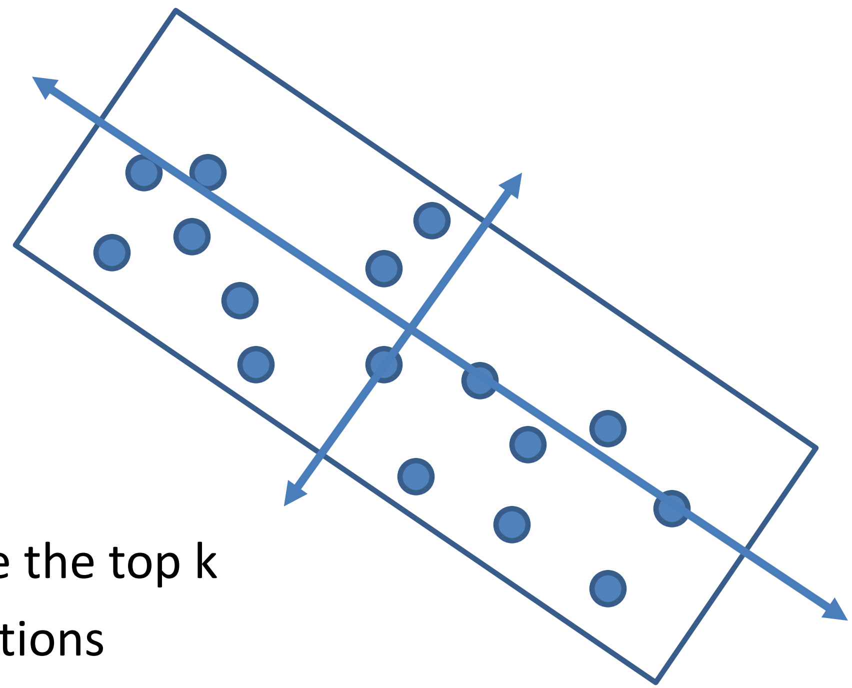
Gives us top k axes

Find the $[a_i, b_i]$ for each axes

and create $\phi_{i1}(x) \ldots \phi_{ik}(x)$

and $\lambda_{i1} \ldots \lambda_{ik}$ for each direction

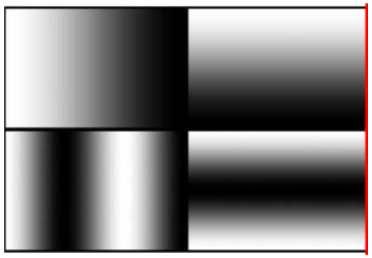
Total dk eigenvalues→ sort and take the top k eigenvalues and corresponding functions

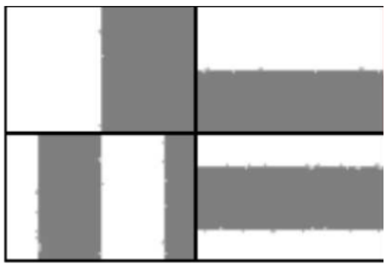


Algorithm

Threshold chosen

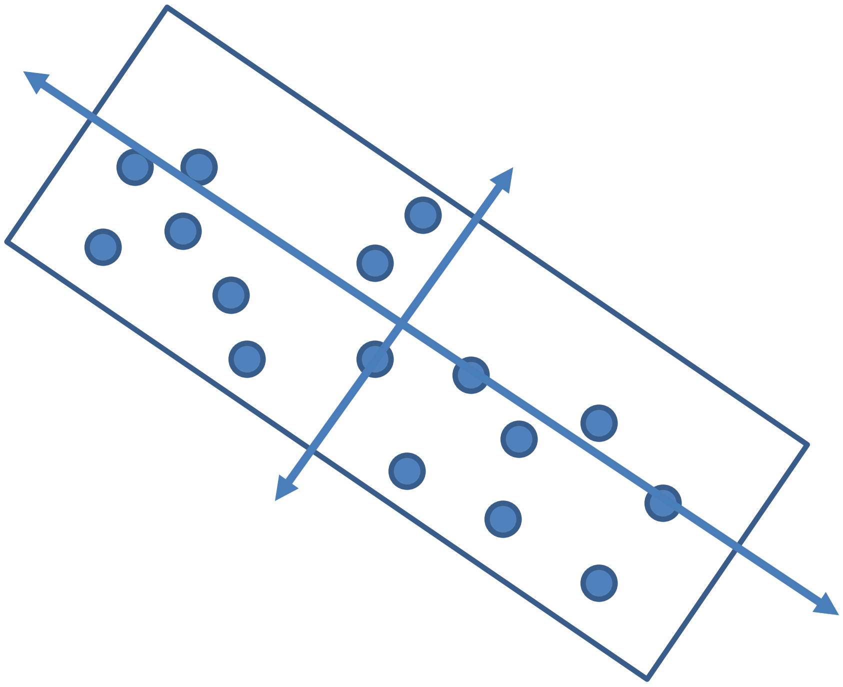
Eigenfunctions





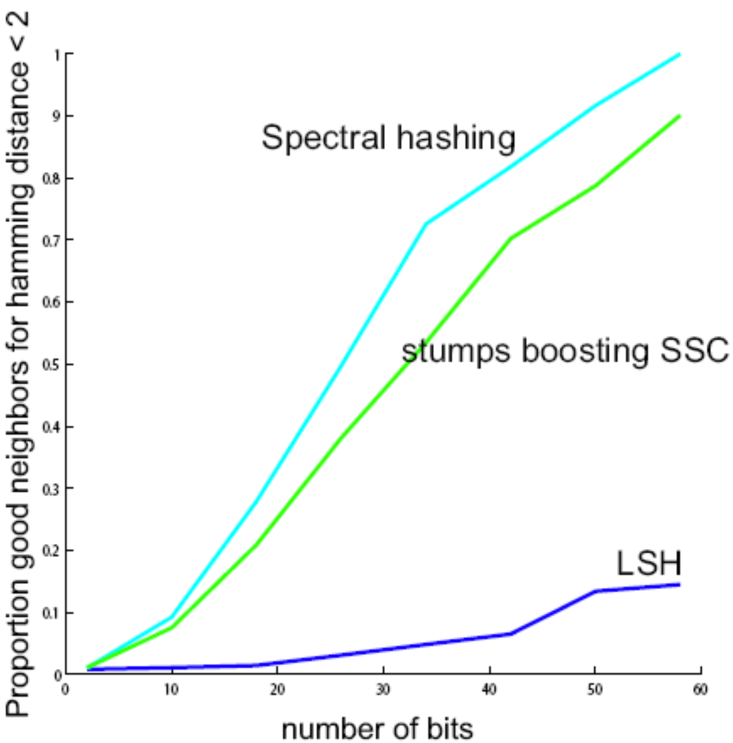
Empirical observation:
 bit codes

seem robust to the uniform assumption



Results

- Shown to have better properties than naïve LSH on large datasets



[Image from Waiss et al]

Summary

- Large literature on learning the hash codes rather than use random projection
  - Liu, Wei, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. "**Supervised hashing with kernels.**" *IEEE CVPR 2012.*
  - Muja, Marius, and David G. Lowe. "**Scalable nearest neighbor algorithms for high dimensional data**." *IEEE TPAMI (2014): 2227-2240*.
  - Wang, Jingdong, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. "**Hashing for similarity search: A survey**." *arXiv preprint arXiv:1408.2927 (2014).*

- Unfortunately, theoretical guarantees are not available for such data-dependent version
  - time to calculate projections might also be higher.

References:

- Primary references for this lecture
  - Spectral Hashing**,** Yair Weiss, Antonio Torralba and Rob Fergus. [*NIPS*], 2008