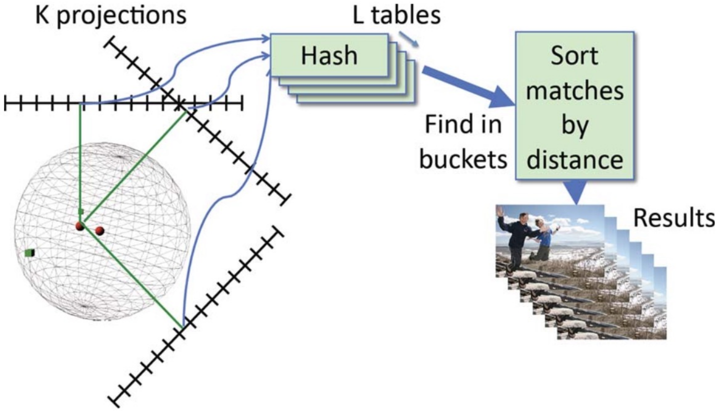
CS60021: Scalable Data Mining

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

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

Algo: Choose (k, L).

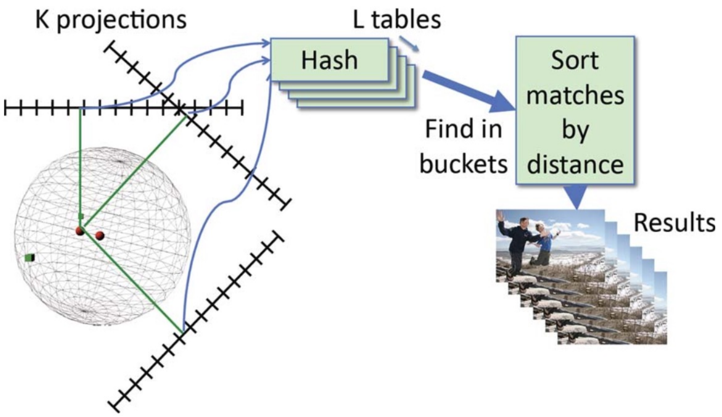
do L times

iid hash functions : $\{\mathrm{h_{i1}} \ldots . \mathrm{h_{ik}}\}$

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 $H_1(q) \ldots . H_L(q)$ 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

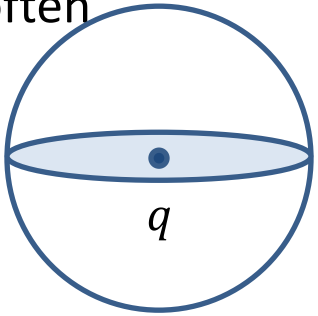
Entropy based LSH

- 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]

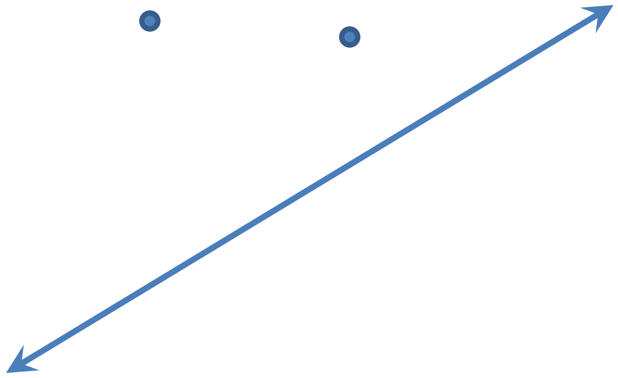
- Elegant way to sample from the success probability distribution
  - Perturb the query point repeatedly and probe
  - Buckets that have high probability should come up often
  - Theoretical guarantee



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. $\Delta = (-1, 0, +1, +1, 0 \ldots . -1)$
  - 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

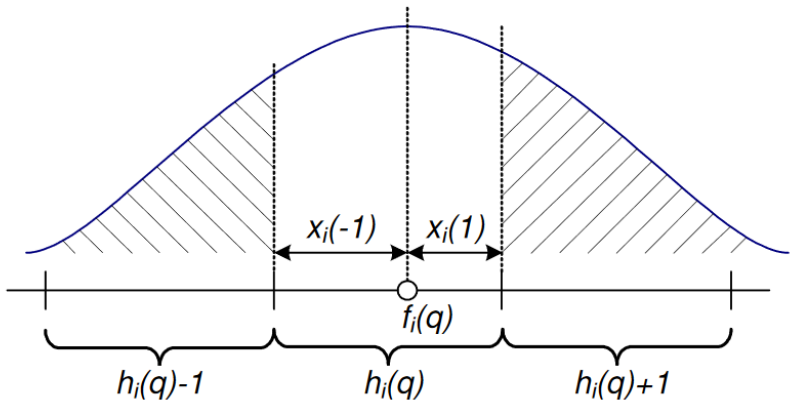


Image from Lv et al.

$f_i(q) = q \cdot v_i + b_i$ 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

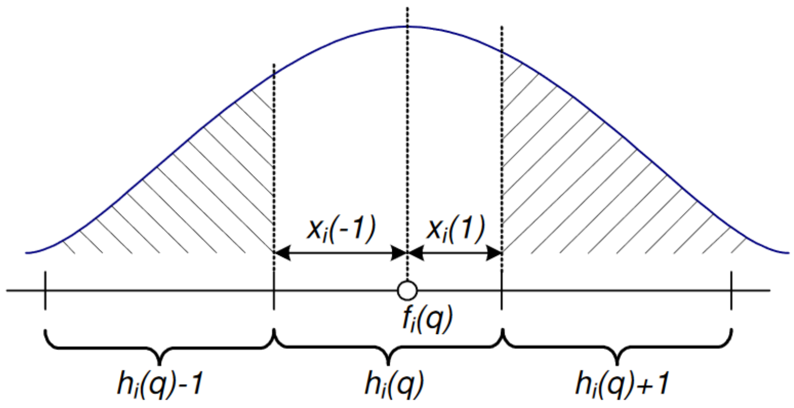


Image from Lv et al.

$x_i(+1)$ *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 $\Delta = (\delta_1 \dots \delta_k)$ then

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

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

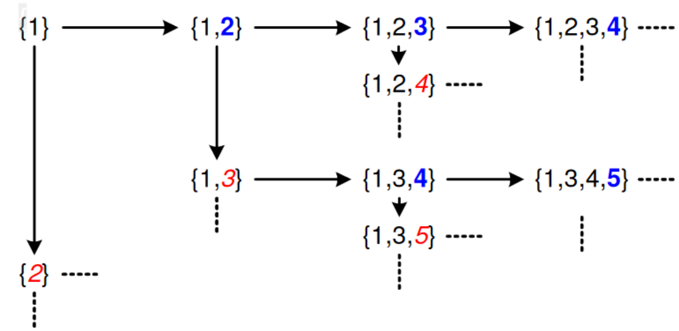
Ex: $\Delta = (+1, 0, -1)$,

Ordering buckets

- Define $score(\Delta) = \sum x_i(\delta_i)^2$
- Lower the score, higher the probability of p being in the bucket
- 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 (call these as $z_1 \leq z_2 \dots \leq z_{2k}$)
- Start with $A = \{1\}$
- Repeatedly do either *shift* or *expand*
  - *shift* replace $\max(A)$ by 1+max(A)
  - *expand* adds 1+max(A) to A

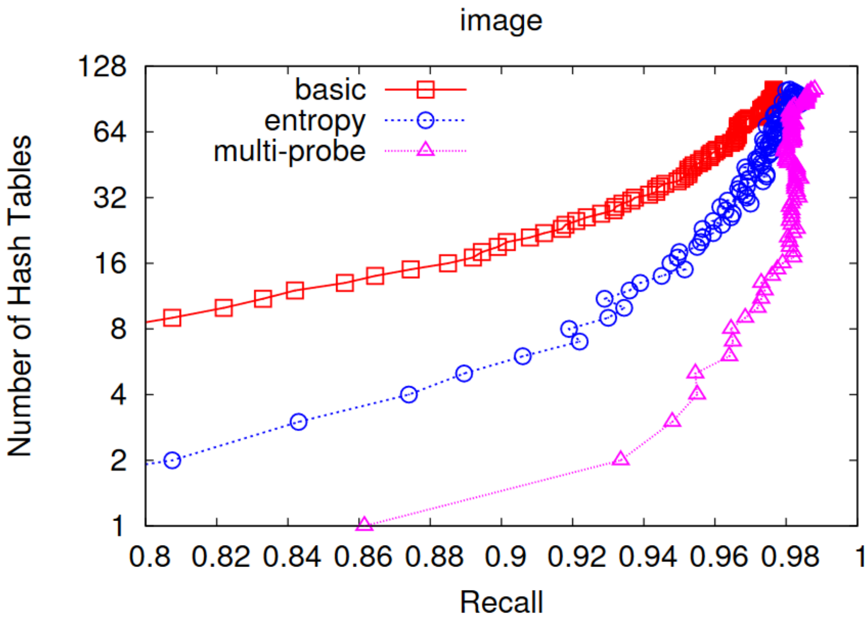


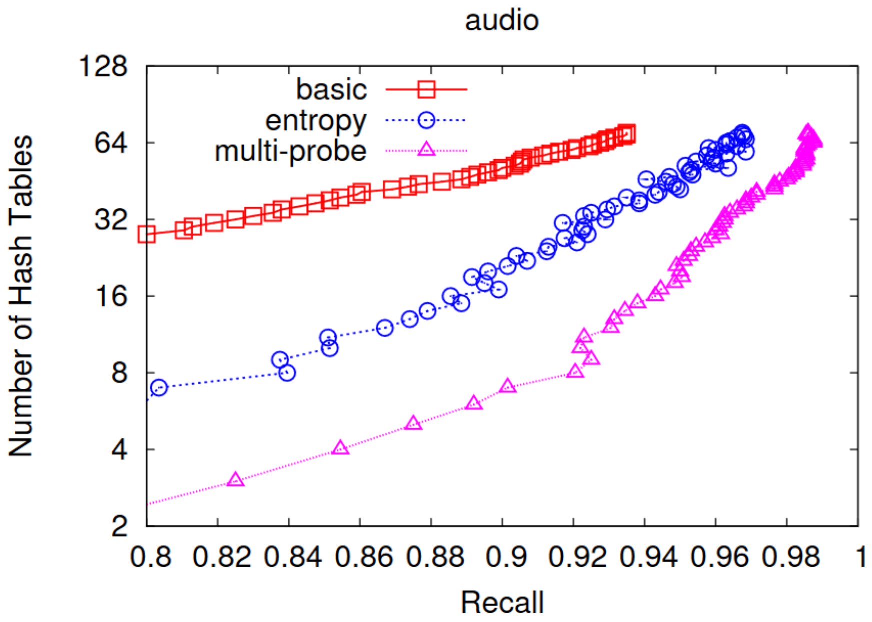
Multiprobe LSH

- 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





Implementation Notes

- FALCONN:
  - https://github.com/FALCONN-LIB/FALCONN/wiki/How-to-Use-FALCONN
  - Original authors: Andoni et al.
  - Implements LSH for cosine similarity.
  - Set #bits, #tables, #probes
  - Set the LSH family – crosspolytope.
  - Build index and query

Implementation Notes

- FAISS:
  - https://github.com/facebookresearch/faiss/wiki/
  - L2 Distance based search.
  - Many indexes implemented – Flat, IVF, IndexBinaryHash.
  - Another key idea is Product quantization:
    - Find k-centroids (e.g. using k-means clustering) – expensive
    - Encode data as a binary vector by first splitting the vector dimensions and then encoding each dimension as sign of dot product with all the centroids.
    - Multi-probe can be used to reduce memory requirement by reducing k.
  - Not discussed here: Graph-based HNSW is also popular.

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.