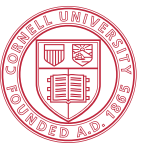
CS5112: Algorithms and Data Structures for Applications

Lecture 18: Locality-sensitive hashing

Ramin Zabih

Some content from: Piotr Indyk; Wikipedia/Google image search;
J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org



Today

- Prelim discussion
- Locality sensitive hashing

Prelim discussion

- Final will be a lot like the prelim
  - But including material starting today
  - Review session in class before exam
- Regrades are done on the entire exam
  - Your score can go up or down
  - Clearly you should request one if we made an obvious mistake
  - Regrades on gradescope, open at 2PM, close a week later
- Staff available after lecture, informally

Prelim stats

- Overall people did quite well
- 14 students got a perfect score
- Hardest questions:
  - 1.4 (collision handling data structures): 54% got this right
  - 4.5 (last 4 had c): 45%
  - Dynamic programming question: 72%

Curse of dimensionality

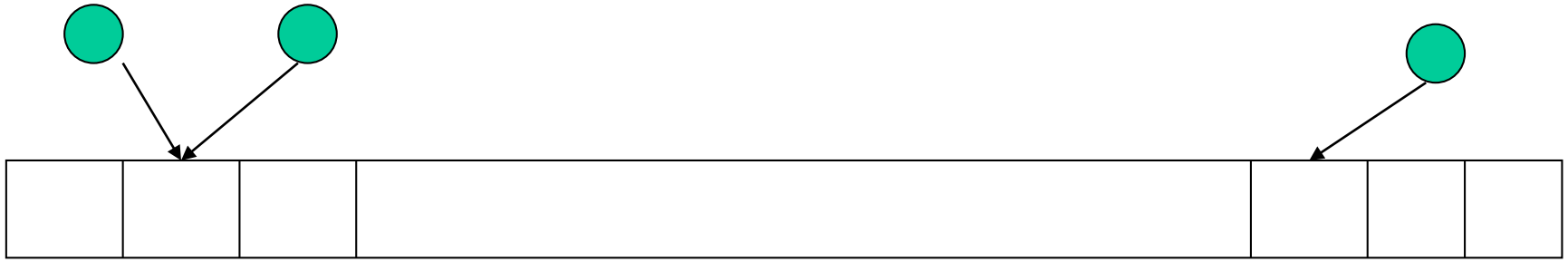
- High dimensions are not intuitive, and data becomes sparse
  - Volume goes up so fast that an exponential amount of data needed
- “Most of the volume is in the corners” theorem
  - Think of a 3D sphere inscribed in a box
- Particularly problematic for NN search
- Quad trees require exponential space and time
- k-d trees requires exponential time

Locality sensitive hashing

- We can use hashing to solve ‘near neighbors’
  - And thus nearest neighbors, if we really need this
- Ensuring collisions is the key idea!
- Make nearby points hash to the same bucket
  - In a probabilistic manner
- By giving up on exactness we can overcome the curse of dimensionality
  - This is an extremely important technique in practice

Basic idea of LSH

- A family of hash functions is locality sensitive if, when we pick a random element h of that family, for any 2 points p, q
  - $P[h(p) = h(q)]$ is 'high' if p is 'close' to q
  - $P[h(p) = h(q)]$ is 'low' if p is 'far' from q



- Cool idea, assuming such hash functions exist! (they do)

How to use LSH for NN

- Pick l different hash functions at random, $h_1, \ldots, h_l$
- Put each data point p into the l buckets $h_1(p), \ldots, h_l(p)$
- Now for a query point q we look in the corresponding buckets $h_1(q), \ldots, h_l(q)$
  - Return the closest point to q
- How does the running time scale up with the dimension d?
  - Nothing is exponential in d, which is awesome!
  - Can prove worst case is slightly better than $O(nd)$

Do such functions exist ?

- Generally LSH functions use random projections
- Main example: consider the hypercube, i.e.,
  - Points from $\{0,1\}^d$
  - Hamming distance $D(p,q)$ = # positions on which p and q differ
- Define hash function h by choosing a set S of k random coordinates, and setting

$h(p)$ = projection of p on S

Example

- Take

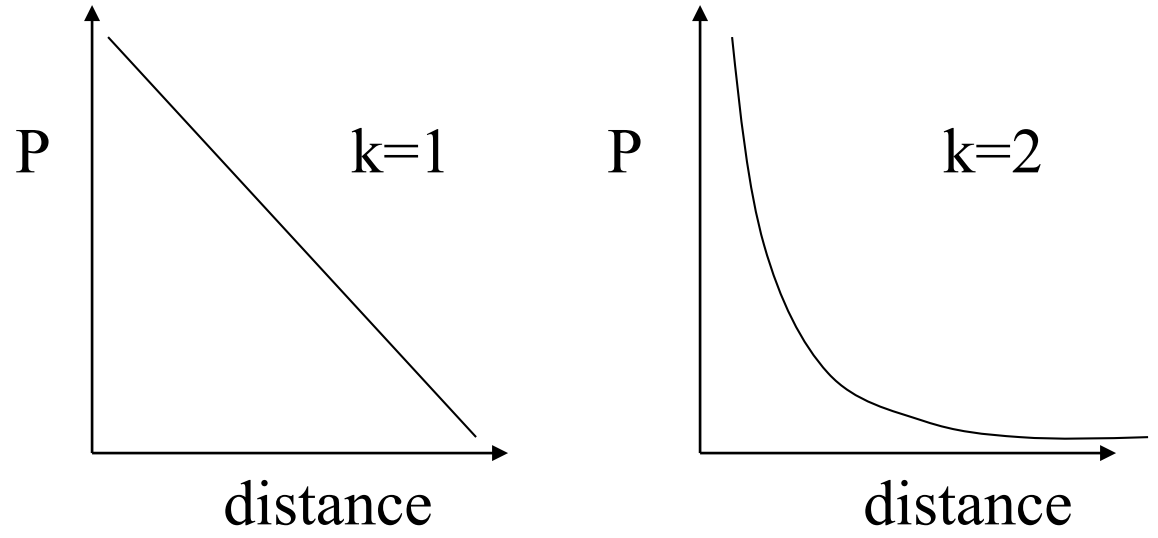
$$d = 10,$$
$$p = 0101110010,$$
$$k = 2,$$
$$S = \{2,5\}$$

- Then $h(p) = 11$

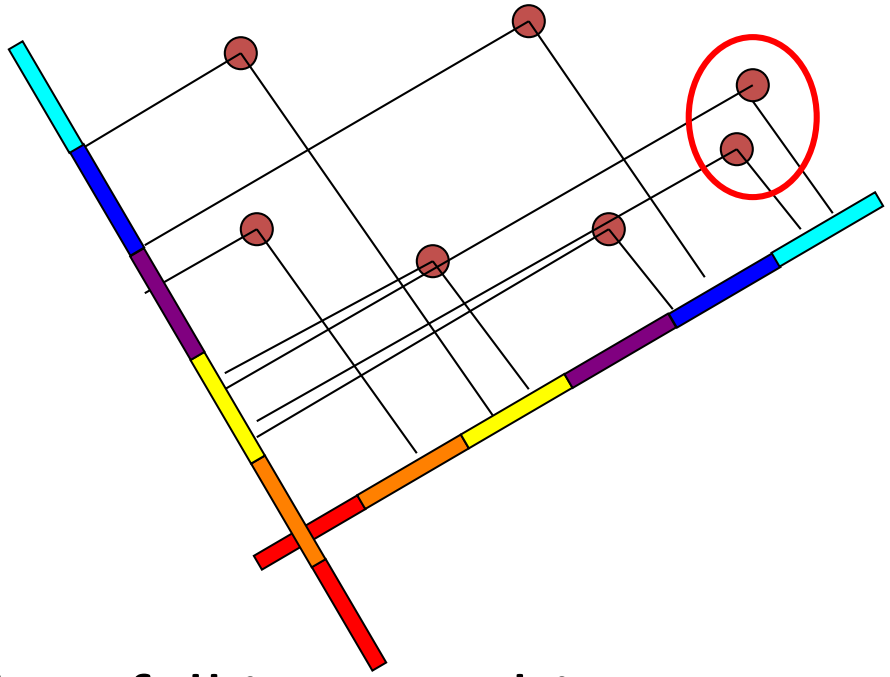
These hash functions are locality-sensitive

$$\mathrm{P}[h(p) = h(q)] = (1 - \frac{D(p,q)}{d})^k$$

- We can vary the probability by changing k



Random Projections + Binning



- Nearby points will often fall in same bin
- Faraway points will rarely fall in same bin

Other important LSH families

- We looked at Hamming distance via random subset projection
- There is a large literature on what distance functions support LSH, including lots of important cases and impossibility proofs
- Two particularly nice examples:
  - Jacard similarity via MinHash
  - Angle similarity via SimHash

Jacard similarity via min hash

- Given a universe of items we want to compare two subsets
  - Need a hash function which maps to same bucket if they have a lot of overlap
- Hash functions are random re-ordering of the universe
  - Hash is the minimum value of a subset under a permutation
- When there is a lot of overlap, the minimum value tends to be the same, and so the two subsets map to the same bucket

Angle similarity via SimHash

- Angle similarity via projection onto random vector
  - VERY important for machine learning, etc.
- Pick a random unit vector r, and check if the two inputs x, y are on the same side of the half-space it defines
- Compute the dot products $\langle x, r \rangle, \langle y, r \rangle$
  - Do they have the same sign?