lecture 5 - Similarity Hashing

Hash Function

\( h: x \rightarrow \{1, \ldots, N\} \) (Typically 8, sometimes 2)

- Useful as random number generator replacement (instead of rand(), rand(), and \( x \lt x \lt \text{hash}(1), \text{hash}(2) \),)
- Alternatively use seed(\( x \)), rand() (But this is very expensive)

Useful things to do with a hash function

1) random value \( \frac{h(x)}{N} \in [0, 1] \)
2) Perform hash check parity \( h(\text{hash}(x)) \)
   is \( [0, 1] \) or \( \{0, 1\} \) with prob. \( \frac{1}{2} \)
3) Basic request \( \Pr \{ h(x) = c \} = \frac{1}{N} \)
   \( \text{histo indep } \Pr \{ h(a) = c, \ldots, h(a_n) = c \} = \frac{1}{N^n} \)
   for \( i \neq j \) distinct
4) For simplicity (unrealistic) use ideal hash \( h(x) \)

Permutations for hashes (the Fastest hash)

\( (x, y) \rightarrow (y \times \text{xor} \ h(y)) \)
multiply and hash

Useful - Back off Pseudorandom Permute Generate

\( \rightarrow x \rightarrow \text{Crypt}(x) \) is 1:1

Locality Sensitive Hashing

Map \( x \rightarrow h(x) \) such that similarity between \( (x, x') \) is preserved

\( \frac{\text{d}(h(x), h(x'))}{D(x, x')} \leq 1 \) \( \epsilon \in \mathbb{E} \)

Not possible for all \( D(x, x') \) but available e.g. for Euclidean distance

Example (for \( L_2 \): \( H \times \text{for } \text{Pyramid} \))

(His is the Inonyk & Runion Example)

Similarly Estimation Techniques for

Random Algorithms (11, Chapter 102)

Thm: \( \frac{1}{2n} \sum_{i=1}^{n} \| h(x_i) - h(x'_i) \|^2 \)

\( = \arccos \langle x, x' \rangle \)

for \( \| x \| = \| x' \| = 1 \)
Lecture 5 - Similarity Hashing

Estimator for inner product:

\[ \langle x, x' \rangle = \frac{1}{2m} \| h(x) - h(x') \| \]

This is much cheaper:

- modern CPUs can do 64-256 bit
- Hamming distance waps in 1 clock cycle (1ns)
- only need to store 1 float \( \langle x, x' \rangle \), 64bit hash regardless of the dimensionality of the data

**Theorem:**

\[ \Pr \left\{ \left| \langle x, x' \rangle - \cos \frac{\| h(x) - h(x') \|}{2m} \right| > \varepsilon \right\} \leq 2 \exp \left( -\frac{m \varepsilon^2}{\pi^2} \right) \]

**Proof:** use the Poincaré inequality (like Hölder's)

\[ \rightarrow \text{ replace } 1 \text{ hash } 1 \text{ hash } \]

\[ \rightarrow \text{ argmax doesn't change by any hash } \frac{h}{2m} \]

\[ \rightarrow \text{ cos doesn't change w.r.t } \]

\[ \frac{1}{m} \text{ w.r.t } \frac{\| h(x) - h(x') \|}{2m} \neq \langle x, x' \rangle \]

\[ \rightarrow \text{ but by inverting the condition we can get it } \]

**Definition:** LHSH scheme with dist:

\[ \text{Sim} (x, x') = \Pr_{h \in S} \{ h(x) = h(x') \} \]

**Lemma:** \( \text{Sim} (x, x') \) is a kernel

**Proof:** \( S (h(x), h(x')) \) is a kernel

**Lemma:** \( d(x, x') = 1 - \text{Sim} (x, x') \)

satisfies the triangle inequality

**Proof:** for any given \( h \) we have

\[ \left[ 1 - S (h(x), h(x')) \right] + \left[ 1 - S (h(x'), h(x)) \right] \]

\[ - \left[ 1 - S (h(x), h(x')) \right] \]

\[ = 1 + S (h(x), h(x')) - S (h(x), h(x')) S (h(x'), h(x)) \]

\[ = 0 \]

Taking expectation over \( h \)

**Example:** Sim Hash (as a claim)

\[ x \rightarrow \text{Sim} (x, x') \]

**Example:** Minwise Hash (Broder)

\[ \text{hash set } \frac{1}{|S|} \text{ min}(h(S)) \rightarrow h(S) \]

\[ = \text{ selects a random chain for } S \]

\[ \Pr \{ h(S) = h(S') \} = \frac{|S \cap S'|}{|S||S'|} \]
lecture 5 - Similarity Hashing

Proof:

- Smallest hash in $S \subseteq S'$ is with probability $\frac{|S \cap S'|}{|S'|}$ in $S \subseteq S'$
- Probability $h(x') = h(x)$ only if the element is in both sets

Detour: Shingles (Broder et al.)

- Instead of picking single element, pick $k$ smallest entities
- Not quite correct but good enough

Better: Conditional Random Sampling (Li et al.) (use the hash ID, too)

Application: Duplicate detection on images

1) $(h_i(x), \text{decide}) \rightarrow$ sort by $h_i(x)$
2) quit (decide, decide) pairs to find collision candidates
3) count (decide, decide) to confirm links

$O(d \cdot k)$ for large clusters

Back to hash functions

- Can we get a binary $h(x)$?

Definition $\text{sim}^+(x, x') := Pr_{h_i} \{ \text{dist}(h(x), h(x')) = \text{dist}(h(x), h(x')) \}$

$= \frac{1}{2} + \frac{1}{2} \text{dist}(h(x), h(x'))$ for different hashes

Corollary: we can embed $\text{sim}^+(x, x')$ on cube

Application: Fast ER sampling (Ahmed et al.)

The problem: in clusters we have to eval

$p(y|x) \approx p(x \in \Theta) \cdot p(y|\Theta) = \exp(\langle c(x), \Theta \rangle - g(\Theta)) \cdot p(y|\Theta)$

Computing $\langle c(x), \Theta \rangle$ is tractable:

$O(d \cdot k)$ for $10^4 - 10^5$ clusters
CPU cost is \( \sim 10^7 \) FLOPS for 1 draw per sampling round. This is fatal.

Variant A) approximate by SIM hash and use Chernoff bound for rejection sampling.

\[ E = O\left(\frac{1}{\sqrt{n}}\right) \]

This is terrible since \( \exp(-E) \)

Variant B) Metropolis-Hastings scheme

have \( p(x) \) to sample, but can only afford \( q(x'|x) \)

\[ = \text{accept with prob } \min \left( 1 \frac{p(x')q(x|x')}{p(x)q(x'|x)} \right) \]

\[ = \text{special case } q(x'|x) = q(x') \text{ to obtain } \min \left( 1 \frac{p(x')q(x)}{p(x)q(x')} \right) \]

\[ \text{so use approx } \frac{1}{3}h(x) + 1h(x) - h(\Theta)k \text{ as proposal } \]

3 FLOPS \( \Rightarrow \) \( 3 \times 10^5 \) FLOPS per proposal

\[ \text{good acceptance ratio } \Rightarrow \text{fast} \]