

Locality Sensitive Hashing

Lecture 14

March 7, 2019

Near-Neighbor Search

Collection of n points $\mathcal{P} = \{x_1, \dots, x_n\}$ in a metric space.

NNS: preprocess \mathcal{P} to answer near-neighbor queries: given query point y output $\arg \min_{x \in \mathcal{P}} \text{dist}(x, y)$

c -approximate NNS: given query y , output x such that $\text{dist}(x, y) \leq c \min_{z \in \mathcal{P}} \text{dist}(z, y)$. Here $c > 1$.

Near-Neighbor Search

Collection of n points $\mathcal{P} = \{x_1, \dots, x_n\}$ in a metric space.

NNS: preprocess \mathcal{P} to answer near-neighbor queries: given query point y output $\arg \min_{x \in \mathcal{P}} \text{dist}(x, y)$

c -approximate NNS: given query y , output x such that $\text{dist}(x, y) \leq c \min_{z \in \mathcal{P}} \text{dist}(z, y)$. Here $c > 1$.

Brute force/linear search: when query y comes check all $x \in \mathcal{P}$

Near-Neighbor Search

Collection of n points $\mathcal{P} = \{x_1, \dots, x_n\}$ in a metric space.

NNS: preprocess \mathcal{P} to answer near-neighbor queries: given query point y output $\arg \min_{x \in \mathcal{P}} \text{dist}(x, y)$

c -approximate NNS: given query y , output x such that $\text{dist}(x, y) \leq c \min_{z \in \mathcal{P}} \text{dist}(z, y)$. Here $c > 1$.

Brute force/linear search: when query y comes check all $x \in \mathcal{P}$

Beating brute force is hard if one wants near-linear space!

NNS in Euclidean Spaces

Collection of n points $\mathcal{P} = \{x_1, \dots, x_n\}$ in \mathbb{R}^d .

$\text{dist}(x, y) = \|x - y\|_2$ is Euclidean distance

Assume n and d are large.

- Linear search with no data structures: $\Theta(nd)$ time, storage is $\Theta(nd)$
- Exact NNS: either query time or space or both are exponential in dimension d
- $(1 + \epsilon)$ -approximate NNS for dimensionality reduction: reduce d to $O(\frac{1}{\epsilon^2} \log n)$ using JL but exponential in d is still impractical
- Even for approximate NNS, beating nd query time while keeping storage close to $O(nd)$ is non-trivial!

Approximate NNS

Focus on c -approximate NNS for some small $c > 1$

Simplified problem: given query point y and fixed radius $r > 0$, distinguish between the following two scenarios:

- if there is a point $x \in \mathcal{P}$ such that $\text{dist}(x, y) \leq r$ output a point x' such that $\text{dist}(x', y) \leq cr$
- if $\text{dist}(x, y) \geq cr$ for all $x \in \mathcal{P}$ then recognize this and fail

Algorithm allowed to make a mistake in intermediate case

Approximate NNS

Focus on c -approximate NNS for some small $c > 1$

Simplified problem: given query point y and fixed radius $r > 0$, distinguish between the following two scenarios:

- if there is a point $x \in \mathcal{P}$ such that $\text{dist}(x, y) \leq r$ output a point x' such that $\text{dist}(x', y) \leq cr$
- if $\text{dist}(x, y) \geq cr$ for all $x \in \mathcal{P}$ then recognize this and fail

Algorithm allowed to make a mistake in intermediate case

Can use binary search and above procedure to obtain c -approximate NNS.

LSH Approach for Approximate NNS

[Indyk-Motwani]

Initially developed for NNSearch in high-dimensional Euclidean space and then generalized to other similarity/distance measures.

Use **locality-sensitive hashing** to solve simplified decision problem

Definition

A family of hash functions is (r, cr, p_1, p_2) -LSH with $p_1 > p_2$ and $c > 1$ if h drawn randomly from the family satisfies the following:

- $\Pr[h(x) = h(y)] \geq p_1$ when $\text{dist}(x, y) \leq r$
- $\Pr[h(x) = h(y)] \leq p_2$ when $\text{dist}(x, y) \geq cr$

Key parameter: the gap between p_1 and p_2 measured as $\rho = \frac{\log p_1}{\log p_2}$

LSH Approach for Approximate NNS

Use **locality-sensitive hashing** to solve simplified decision problem

Definition

A family of hash functions is (r, cr, p_1, p_2) -LSH with $p_1 > p_2$ and $c > 1$ if h drawn randomly from the family satisfies the following:

- $\Pr[h(x) = h(y)] \geq p_1$ when $\text{dist}(x, y) \leq r$
- $\Pr[h(x) = h(y)] \leq p_2$ when $\text{dist}(x, y) \geq cr$

Key parameter: the gap between p_1 and p_2 measured as $\rho = \frac{\log p_1}{\log p_2}$ usually small.

Two-level hashing scheme:

- Amplify basic locality sensitive hash family to create better family by repetition
- Use several copies of amplified hash functions

Amplification

Fix some r . Pick k independent hash functions h_1, h_2, \dots, h_k . For each x set

$$g(x) = h_1(x)h_2(x) \dots h_k(x)$$

$g(x)$ is now the larger hash function

- If $\text{dist}(x, y) \leq r$: $\Pr[g(x) = g(y)] \geq p_1^k$
- If $\text{dist}(x, y) \geq cr$: $\Pr[g(x) = g(y)] \leq p_2^k$

Amplification

Fix some r . Pick k independent hash functions h_1, h_2, \dots, h_k . For each x set

$$g(x) = h_1(x)h_2(x) \dots h_k(x)$$

$g(x)$ is now the larger hash function

- If $\text{dist}(x, y) \leq r$: $\Pr[g(x) = g(y)] \geq p_1^k$
- If $\text{dist}(x, y) \geq cr$: $\Pr[g(x) = g(y)] \leq p_2^k$

Choose k such that $p_2^k \simeq 1/n$ so that far away points don't collide with query y . Then $p_1^k = 1/n^\rho$.

Multiple hash tables

- If $\text{dist}(x, y) \leq r$: $\Pr[g(x) = g(y)] \geq p_1^k$
- If $\text{dist}(x, y) \geq cr$: $\Pr[g(x) = g(y)] \leq p_2^k$

Choose k such that $p_2^k \simeq 1/n$ so that far away points don't collide with query y . $k = \frac{\log n}{\log(1/p_2)}$. Then $p_1^k = 1/n^\rho$ which is also small.

To make good point collide with y choose $L \simeq n^\rho$ hash functions

g_1, g_2, \dots, g_L

- $L \simeq n^\rho$ hash tables
- Storage: $n^{1+\rho}$ (ignoring log factors)
- Query time: kn^ρ (ignoring log factors)

Details

What is the range of each g_i ? A k tuple $(h_1(x), h_2(x), \dots, h_k(x))$. Hence depends on range of the h 's.

Details

What is the range of each g_i ? A k tuple $(h_1(x), h_2(x), \dots, h_k(x))$. Hence depends on range of the h 's.

We leave the range implicit. Say range of g_i is $[m^k]$ where range of each h is $[m]$. We only store non-empty buckets of each g_i and there can be at most n of them. For each g_i can use another hash function ℓ_i that maps m^k to $[n]$.

Details

What is the range of each g_i ? A k tuple $(h_1(x), h_2(x), \dots, h_k(x))$. Hence depends on range of the h 's.

We leave the range implicit. Say range of g_i is $[m^k]$ where range of each h is $[m]$. We only store non-empty buckets of each g_i and there can be at most n of them. For each g_i can use another hash function ℓ_i that maps m^k to $[n]$.

So what is actually stored?

- L hash tables one for each g_i using chaining
- Each item x in database is hashed and stored in each of the L tables.
- Total storage $O(Ln)$
- Time to hash an item: Lk evaluations of basic hash functions h_j

Query

Given new point y how to query?

- Hash y using g_i for $1 \leq i \leq L$
- For each i check all items in bucket of $g_i(y)$ and compute all their distances and output first item x such that $\text{dist}(x, y) \leq cr$.
- If no item found report FAIL

Query

Given new point y how to query?

- Hash y using g_i for $1 \leq i \leq L$
- For each i check all items in bucket of $g_i(y)$ and compute all their distances and output first item x such that $\text{dist}(x, y) \leq cr$.
- If no item found report FAIL

What if too many items collide with y ? How do we bound query time?

Fix: Stop search after comparing with $\Theta(L)$ items and report failure

Analysis

Query correctly fails if no item x such that $\text{dist}(x, y) \leq cr$

If query outputs a point x then $\text{dist}(x, y) \leq cr$

Main issue: What is the probability that there be a good point x^* such that $\text{dist}(x, y) \leq r$ and algorithm fails?

Analysis

Query correctly fails if no item x such that $\text{dist}(x, y) \leq cr$

If query outputs a point x then $\text{dist}(x, y) \leq cr$

Main issue: What is the probability that there be a good point x^* such that $\text{dist}(x, y) \leq r$ and algorithm fails?

Two reasons

- x^* does not collide with y
- too many bad points (more than $10L$ collide with y and cause query algorithm to stop and fail without discovering x^*)

Analysis

Main issue: What is the probability that there be a good point x^* such that $\text{dist}(x, y) \leq r$ and algorithm fails?

Two reasons

- x^* does not collide with y
- too many bad points (more than $10L$ collide with y and cause query algorithm to stop and fail without discovering x^*)

First issue:

$$\Pr[g_i(x^*) = g_i(y)] = p_1^k \geq 1/n^\rho$$

If $L > 10n^\rho$ then $\Pr[g_i(x^*) \neq g_i(y) \forall i] \leq 1/10$.

Analysis

Main issue: What is the probability that there be a good point x^* such that $\text{dist}(x, y) \leq r$ and algorithm fails?

Two reasons

- x^* does not collide with y
- too many bad points (more than $10L$ collide with y and cause query algorithm to stop and fail without discovering x^*)

Second issue: let x be a bad point, that is $\text{dist}(x, y) > cr$

$\Pr[g_i(x) = g_i(y)] = p_2^k \leq 1/n$ by choice of k

Hence expected number of bad points that collide with y in any table is ≤ 1 . Hence expected number of bad points that collide with y in all tables is at most L . By Markov, probability of more than $10L$ colliding with y is at most $1/10$

Analysis

Hence query for y succeeds with probability $1 - 2/10 \geq 4/5$.

Amplify success probability to $1 - (1/5)^t$ by constructing t copies

Data structure only for one radius r . Need separate data structure for geometrically increasing values of r in some range $[r_{\min}, r_{\max}]$

Part I

LSH for Hamming Cube

Hamming Distance

n points $x_1, x_2, \dots, x_n \in \{0, 1\}^d$ for some large d

$\text{dist}(x, y)$ is the number of coordinates in which x, y differ

Recall that minhash and simhash reduce to Hamming distance estimation

Closely related to more general ℓ_1 distance (ideas carry over)

Question: What is a good (r, cr, p_1, p_2) -LSH? What is ρ ?

LSH for Hamming Cube

Question: What is a good (r, cr, p_1, p_2) -LSH? What is ρ ?

Pick a random coordinate. Hash family = $\{h_i \mid i = 1, \dots, d\}$
where $h_i(x) = x_i$

LSH for Hamming Cube

Question: What is a good (r, cr, p_1, p_2) -LSH? What is ρ ?

Pick a random coordinate. Hash family = $\{h_i \mid i = 1, \dots, d\}$
where $h_i(x) = x_i$

Suppose $\text{dist}(x, y) \leq r$ then

$$\Pr[h(x) = h(y)] \geq (d - r)/d \geq 1 - r/d \simeq e^{-r/d}$$

LSH for Hamming Cube

Question: What is a good (r, cr, p_1, p_2) -LSH? What is ρ ?

Pick a random coordinate. Hash family = $\{h_i \mid i = 1, \dots, d\}$
where $h_i(x) = x_i$

Suppose $\text{dist}(x, y) \leq r$ then

$$\Pr[h(x) = h(y)] \geq (d - r)/d \geq 1 - r/d \simeq e^{-r/d}$$

Suppose $\text{dist}(x, y) \geq cr$ then

$$\Pr[h(x) = h(y)] \leq 1 - cr/d \simeq e^{-cr/d}$$

Therefore $\rho = \frac{\log p_1}{\log p_2} < 1/c$

LSH for Hamming Cube

$$\rho = 1/c$$

Say $c = 2$ meaning we are setting for a 2-approximate near neighbor

- query time is $\tilde{O}(d\sqrt{n})$
- space is $\tilde{O}(dn + n\sqrt{n})$

while exact/brute force requires $O(nd)$ and $O(nd)$. Thus improved query time at expense of increased space.

LSH for Hamming Cube

$$\rho = 1/c$$

Say $c = 2$ meaning we are setting for a 2-approximate near neighbor

- query time is $\tilde{O}(d\sqrt{n})$
- space is $\tilde{O}(dn + n\sqrt{n})$

while exact/brute force requires $O(nd)$ and $O(nd)$. Thus improved query time at expense of increased space.

Questions:

- Is c -approximation good in “high”-dimensions?
- Isn't space a big bottleneck?

Practice: use heuristic choices to settle for reasonable performance. LSH allows for a high-level non-trivial tradeoff between approximation and query time which is not apriori obvious