Locality Sensitive Hashing

Lecture 14

October 13, 2022

Near-Neighbor Search

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

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

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

Near-Neighbor Search

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

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

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

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

Near-Neighbor Search

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

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

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

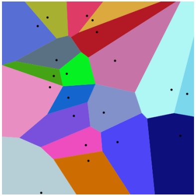
Brute force/linear search: when query y comes check all $x \in \mathcal{P}$

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

NNS in Euclidean Spaces

Collection of n points $\mathcal{P} = \{x_1, \ldots, x_n\}$ in $\mathbb{R}^d$.
$\text{dist}(x, y) = \|x - y\|_2$ is Euclidean distance

- $d = 1$. Sort and do binary search. $O(n)$ space, $O(\log n)$ query time.
- $d = 2$. Voronoi diagram. $O(n)$ space $O(\log n)$ query time.



(Figure from Wikipedia)
- Higher dimensions: Voronoi diagram size grows as $n^{\lfloor d/2 \rfloor}$.

NNS in Euclidean Spaces

Collection of n points $\mathcal{P} = \{x_1, \ldots, x_n\}$ in $\mathbb{R}^d$.
$\mathrm{dist}(x, y) = \|x - y\|_2$ 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 $\mathrm{dist}(x, y) \leq r$ output a point x' such that $\mathrm{dist}(x', y) \leq cr$
- if $\mathrm{dist}(x, y) \geq cr$ 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 $\text{dist}(x, y) \leq r$ 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.

Part I

LSH Framework

LSH Approach for Approximate NNS

[Indyk-Motwani'98]

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

LSH Approach for Approximate NNS

[Indyk-Motwani'98]

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 Example: Hamming Distance

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

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

LSH Example: Hamming Distance

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

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

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

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

- Suppose $\text{dist}(x, y) \le r$ then $\Pr[h(x) = h(y)] \ge (d - r)/d \ge 1 - r/d \simeq e^{-r/d}$
- Suppose $\text{dist}(x, y) \ge cr$ then $\Pr[h(x) = h(y)] \le 1 - cr/d \simeq e^{-cr/d}$

LSH Example: Hamming Distance

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

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

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

Pick a random coordinate: Hash family $= \{h_i \mid i = 1, \ldots, d\}$ 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 $\rho = \frac{\log p_1}{\log p_2} \leq 1/c$

LSH Example: 1-d

n points on line and distance is Euclidean

Question: What is a good LSH?

LSH Example: 1-d

n points on line and distance is Euclidean

Question: What is a good LSH?

Grid line with cr units.

- No two far points will be in same bucket and hence $p_2 = 0$
- But close by points may be in different buckets. So do a random shift of grid to ensure that $p_1 \geq (1 - 1/c)$.

LSH Example: 1-d

n points on line and distance is Euclidean

Question: What is a good LSH?

Grid line with cr units.

- No two far points will be in same bucket and hence $p_2 = 0$
- But close by points may be in different buckets. So do a random shift of grid to ensure that $p_1 \geq (1 - 1/c)$.

Main difficulty is in higher dimensions but above idea will play a role.

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

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 $h_1, h_2, \ldots, h_k$. For each x set

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

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

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

Amplification

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

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

$g(x)$ 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 expected number of far away points that collide with query y is ≤ 1. Then $p_1^k = 1/n^\rho$.

Multiple hash tables

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

Choose k such that $p_2^k \simeq 1/n$ so that expected number of far away points that collide with query y is ≤ 1. $k = \frac{\log n}{\log(1/p_2)}$.

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 $g_1, g_2, \ldots, g_L$

Data Structure

Fix some radius r and consider a (r, cr, p_1, p_2) basic LSH family $\mathcal{H}$ with $\rho = \frac{\log p_1}{\log p_2}$. Given n points $x_1, x_2, \ldots, x_n$ which are the initial set of points.

- Pick $k = \frac{\log n}{\log(1/p_2)}$ and $L = cn^\rho$ for sufficiently large constant c.
- For $i = 1$ to L hash function g_i obtained by picking k independent hash functions from $\mathcal{H}$
- For each $i = 1$ to L create a hash table T_i using hash function g_i. Each x_j is hashed and stored in T_i in location $g_i(x_j)$ via chaining.

Storage: L hash tables. $\Theta(n)$ space for each hash table so total is $nL = n^{1+\rho}$ (ignoring log factors). See next slide for details.

Details

What is the range of each g_i? A k tuple $(h_1(x), h_2(x), \ldots, h_k(x))$. 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 $(h_1(x), h_2(x), \ldots, h_k(x))$. 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. Need to store only index of x in table, not x itself.
- Total storage $O(Ln)$
- Time to hash an item: Lk evaluations of basic LSH functions h_j

Query

Given new point y how to query?

- Hash y using g_i for $1 \leq i \leq L$
- For each $i = 1$ to L do
  - Let S be all points in data that are in same bucket as $g_i(y)$
  - For each $x_j \in S$: if $d(x_j, y) \leq cr$ output x_j
- 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 $i = 1$ to L do
  - Let S be all points in data that are in same bucket as $g_i(y)$
  - For each $x_j \in S$: if $d(x_j, y) \leq cr$ output x_j
- 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

Query time: L hash function evaluations and $\Theta(L)$ distance comparisions.

Analysis

Observations:

- If query outputs a point x then $\text{dist}(x, y) \leq cr$.
- If there is no x such that $\text{dist}(x, y) \leq cr$ then Query correctly fails.

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

Observations:

- If query outputs a point x then $\text{dist}(x, y) \leq cr$.
- If there is no x such that $\text{dist}(x, y) \leq cr$ then Query correctly fails.

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

Two reasons for failure:

- 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 reason:

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

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

Analysis

Two reasons for failure:

- 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 reason: 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 $1 - 2/10 \geq 4/5$.

Query time:

- Hashing y in L tables with $g_1, g_2, \ldots, g_L$ where each g_i is a k tuple of basic LSH functions. Hence $kL = kn^\rho$.
- Compute $d(y, x)$ for at most $O(L)$ points so total of $O(L)$ distance computations.

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 $[r_{\min}, r_{\max}]$

The above two issues result in an additional multiplicative factor of $O(\log n \log \Delta)$ in the query time where $\Delta = \log(r_{\max})/r_{\min})$

Standard data structure issues

- What if original data changes? Hash tables can be adjusted as new points come in or old ones deleted by periodic recomputation of data structure (cost can be amortized)
- Error analysis is only for one query point. By amplification can handle a polynomial in n queries. Periodic rebuilding of data structure with fresh randomness if too many queries.

Part II

LSH for Hamming Cube

Hamming Distance

n points $x_1, x_2, \ldots, x_n \in \{0,1\}^d$ 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 $= \{h_i \mid i = 1, \ldots, d\}$ 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 $= \{h_i \mid i = 1, \ldots, d\}$ 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 $= \{h_i \mid i = 1, \ldots, d\}$ where $h_i(x) = x_i$

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

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

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

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

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

LSH for Hamming Cube

$\rho = 1/c$

Say $c = 2$ meaning we are setting for a 2-approximate near neighbor.
$k = O(d \log n)$ and $L = O(n^\rho) = O(\sqrt{n})$.

- space is $\tilde{O}(dn + n\sqrt{n})$. We store original points and $L = O(\sqrt{n})$ hash tables.
- query time is $\tilde{O}(kL + Ld) = \tilde{O}(d\sqrt{n})$.

Exact/brute force: $O(nd)$ for storage and $O(nd)$ for query time.
Thus improved query time at expense of increased space.

As c increases (our approximation suffers) we get better bounds.
Spaces is $\tilde{O}(dn + n^{1+1/c})$ and query time is $\tilde{O}(dn^{1/c})$.

LSH for Hamming Cube

Questions:

- Is c-approximation good enough 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

Part III

LSH for Euclidean Distances

LSH for Euclidean Distances

Now $x_1, x_2, \ldots, x_n \in \mathbb{R}^d$ and $\text{dist}(x, y) = \|x - y\|_2$

First do dimensionality reduction (JL) to reduce d (if necessary) to $O(\log n)$ (since we are using c-approximation anyway)

LSH for Euclidean Distances

Now $x_1, x_2, \ldots, x_n \in \mathbb{R}^d$ and $\text{dist}(x, y) = \|x - y\|_2$

First do dimensionality reduction (JL) to reduce d (if necessary) to $O(\log n)$ (since we are using c-approximation anyway)

What is a good basic locality-sensitive hashing scheme? That is, we want a hashing approach that makes nearby points more likely to collide than farther away points.

LSH for Euclidean Distances

Now $x_1, x_2, \ldots, x_n \in \mathbb{R}^d$ and $\text{dist}(x, y) = \|x - y\|_2$

First do dimensionality reduction (JL) to reduce d (if necessary) to $O(\log n)$ (since we are using c-approximation anyway)

What is a good basic locality-sensitive hashing scheme? That is, we want a hashing approach that makes nearby points more likely to collide than farther away points.

Projections onto random lines plus bucketing

LSH for Euclidean Distances

Recall we are interested in (r, cr, p_1, p_2) lsh family for a radius r

Consider hash family with two parameters $\bar{a}, w$ where a is a random unit vector (line) in $\mathbb{R}^d$ and w is a uniform number from $[0, r]$

$$h_{a,w}(x) = \lfloor \frac{x \cdot a + w}{r} \rfloor$$

In other words we consider r length buckets on the line defined by vector a where the origin of the bucketing is via a random shift w

LSH for Euclidean Distances

Recall we are interested in (r, cr, p_1, p_2) lsh family for a radius r

Consider hash family with two parameters $\bar{a}, w$ where a is a random unit vector (line) in $\mathbb{R}^d$ and w is a uniform number from $[0, r]$

$$h_{a,w}(x) = \lfloor \frac{x \cdot a + w}{r} \rfloor$$

In other words we consider r length buckets on the line defined by vector a where the origin of the bucketing is via a random shift w

$\rho < 1/c$ for this scheme though it is quite close to $1/c$.

LSH for Euclidean Distances

Recall we are interested in (r, cr, p_1, p_2) lsh family for a radius r

Consider hash family with two parameters $\bar{a}, w$ where a is a random unit vector (line) in $\mathbb{R}^d$ and w is a uniform number from $[0, r]$

$$h_{a,w}(x) = \lfloor \frac{x \cdot a + w}{r} \rfloor$$

In other words we consider r length buckets on the line defined by vector a where the origin of the bucketing is via a random shift w

$\rho < 1/c$ for this scheme though it is quite close to $1/c$.

Can achieve $\rho = (1 + o(1))\frac{1}{c^2}$ using more advanced schemes and this is close to optimal modulo constant factors.