Approximate Nearest Neighbor Search in High Dimensional Space

Junhao Gan

ITEE
University of Queensland

Nearest Neighbor Search

Let P be a set of n d-dimensional points in $\mathbb{R}^d$. Denote the Euclidean distance between two points $p, q \in \mathbb{R}^d$ by $\|p, q\|$.

Recall that:

> Given a query point q, a nearest neighbor (NN) query returns all the points $p \in P$ such that $\|p, q\| \leq \|p', q\|$ for $\forall p' \in P$.

In this class, the dimensionality d cannot be regarded as a constant. The dependence on d in all the complexities must be made explicit.

The Curse of Dimensionality

Many efficient nearest neighbor algorithms are known for the case when the dimensionality d is "low". However, for all the existing solutions, either the space or query time is exponential in the dimensionality d.

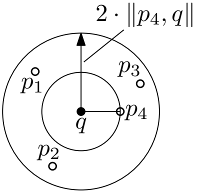
This phenomenon is called the curse of dimensionality.

One approach to deflate the curse is to trade precision for efficiency: specifically, how to achieve polynomial (in both d and n) space and query cost by accepting slightly worse neighbor points.

c-Approximate Nearest Neighbor Search

For $c > 1$, a c-approximate nearest neighbor (c-ANN) query specifies a point q. If p^* is the NN of q, the query returns an arbitrary point $p \in P$ such that $\|p, q\| \leq c \cdot \|p^*, q\|$.

- p_4 is the NN of q.
- $p_1, \ldots, p_4$ are all 2-ANNs of q.
- Any of $p_1, \ldots, p_4$ is a legal answer to the 2-ANN query w.r.t. q.



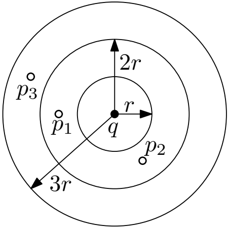
(r, c)-Near Neighbor Search

Given a point q, define $B(q, r)$ as the set of the points in P whose distances to q are at most r.

For $c > 1$, the result of an (r, c)-near neighbor query with a point q is defined as follows:

- If there exists a point in $B(q, r)$, the result must be a point in $B(q, c \cdot r)$.
- Otherwise, the result is either empty or a point in $B(q, c \cdot r)$.

- For the $(r, 2)$-near neighbor query with q, the result can be either empty or any one of p_1 and p_2.
- The result must be one of p_1, p_2 and p_3 for the $(2r, \frac{3}{2})$-near neighbor query with q.



Reduction from 4-ANN to $(r, 2)$-Near Neighbor Search

Next we show how to answer a 4-ANN query by solving a sequence of $(r, 2)$-near neighbor queries with different r values.

Remark. Our technique can be extended to reduce a $((1+\epsilon) \cdot c)$-ANN query to a sequence of (r, c)-near neighbor queries, for any value of $c > 1$ and an arbitrary constant $\epsilon > 0$.

For simplicity, let us make a mild assumption:

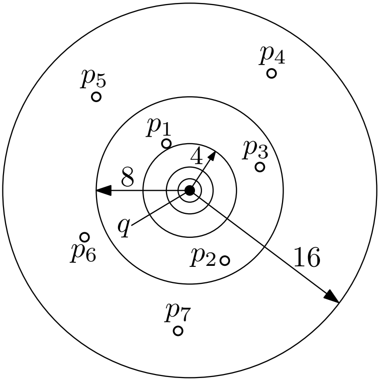
- All the point coordinates are in an integer domain of range $[1, M]$. In other words, the data space is $[1, M]^d$.

Thus, the distance between any two **distinct** points in the data space is in $[1, d_{max}]$, where $d_{max} = \sqrt{d} \cdot M$.

Reduction from 4-ANN to $(r, 2)$-Near Neighbor Search

In the figure, the radii of the circles are $1, 2, 4, 8$ and 16, respectively. Namely, the radius grows by a factor of 2.

We perform $(2^i, 2)$-near neighbor queries in ascending order of i, until a query returns a non-empty result.



Reduction from 4-ANN to $(r, 2)$-Near Neighbor Search

The 4-ANN Query Algorithm

Set $r = 1$. Repeat the following steps:

- Perform an $(r, 2)$-near neighbor query with q. If a point p is returned from the query, then return p as a 4-ANN of q.
- Otherwise, set $r = 2 \cdot r$.

Clearly, there can be at most $\lceil \log_2 d_{max} \rceil$ iterations.

Lemma: The query algorithm correctly returns a 4-ANN of a query point q.

Proof. Let p^* be the NN of q, p the point returned by the algorithm, and r^* the value of r when the algorithm terminates.

On one hand, since r^* is the **smallest** value of r such that a point in P is returned, we have $\frac{r^*}{2} < \|p^*, q\|$. Because otherwise, a point would have been returned when $r = \frac{r^*}{2}$, which contradicts with the definition of r^*. Thus, $r^* < 2 \cdot \|p^*, q\|$.

On the other hand, as p is returned from an $(r^*, 2)$-near neighbor query, $\|p, q\| \leq 2 \cdot r^*$.

Combining the above two inequalities, $\|p, q\| < 4 \cdot \|p^*, q\|$. Therefore, p is a 4-ANN of q.

$\square$

Next we will focus on how to answer $(r, 2)$-near neighbor queries. In particular, we will consider only $r = 1$ (this does not lose generality; why?).

We will learn a new technique called locality sensitive hashing (LSH).

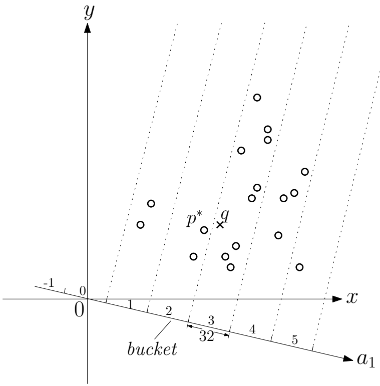
Basic Idea

First, pick a random line ℓ_1 passing through the origin. Then, chop the line into intervals of width 32. Associate each interval with a unique ID.

Let $h_1 : \mathbb{R}^d \to \mathbb{N}$ be the hash function that projects $\forall p \in \mathbb{R}^d$ into the interval with ID $h_1(p)$ of ℓ_1. As a result, each interval essentially is a hash bucket.

Observe that by h_1, "nearby" points are more likely to be hashed into the same bucket than those "far apart" points.

A hash function with such "locality preserving" property is called locality sensitive.



(p_1, p_2)-Sensitive Family

For $p_1 > p_2$, a function family $\mathcal{H} = \{h : \mathbb{R}^d \to U\}$ is called (p_1, p_2)-sensitive if for $\forall h \in \mathcal{H}$ and any two points $u, v \in \mathbb{R}^d$, we have:

- if $\|u, v\| \leq 1$, then the probability $Pr[h(u) = h(v)] \geq p_1$,
- if $\|u, v\| > 2$, then the probability $Pr[h(u) = h(v)] \leq p_2$.

There exists a (p_1, p_2)-sensitive family such that $\rho = \frac{\log 1/p_1}{\log 1/p_2} \leq 0.5$.

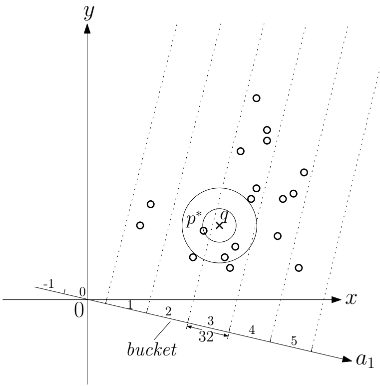
For a query point q, the points in $B(q, 1)$ are hashed into the bucket $h(q)$ with a relatively high probability. While those points that are not in $B(q, 2)$ are hashed into $h(q)$ with a smaller probability.

Intuitively, the points in the bucket $h(q)$ are more likely in $B(q, 2)$.

False Positive

For a query point q, the points u in the bucket $h(q)$ with $\|u, q\| > 2$ are called false positives.

Unfortunately, the expected number of false positives can be as large as $p_2 \cdot n$. This seriously affects the query time.



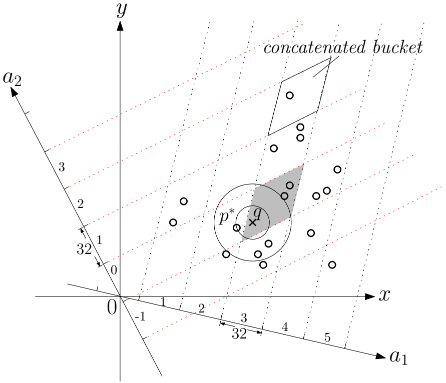
We remedy this issue by "concatenating" multiple hash functions in $\mathcal{H}$ together.

Concatenating Hash Functions

Continuing the previous example, let us generate another hash function h_2 in the same way as h_1.

Consider a hash function $g : \mathbb{R}^d \rightarrow \mathbb{N}^2$ defined by concatenating h_1 and h_2, i.e., $g(u) = (h_1(u), h_2(u))$. Each $g(u)$ corresponds to a (concatenated) bucket. $g(u) = g(v)$ **if and only if** $h_1(u) = h_1(v)$ and $h_2(u) = h_2(v)$.

As shown in the figure, the number of false positives for q in the bucket $g(q) = (3, 0)$ (i.e., the gray region) has been significantly reduced.



Concatenating Hash Functions

For an integer k, we define a function family $\mathcal{G} = \{g : \mathbb{R}^d \to U^k\}$, where each $g(u) = (h_1(u), h_2(u), \cdots, h_k(u))$ consists of k hash functions chosen independently and uniformly from an (p_1, p_2)-sensitive family $\mathcal{H}$.

For any two points $u, v \in \mathbb{R}^d$, $g(u) = g(v)$ **if and only if** $h_i(u) = h_i(v)$ for all $i = 1, \cdots, k$. Thus, $Pr[g(u) = g(v)] = \prod_{i=1}^{k} Pr[h_i(u) = h_i(v)]$. Hence:

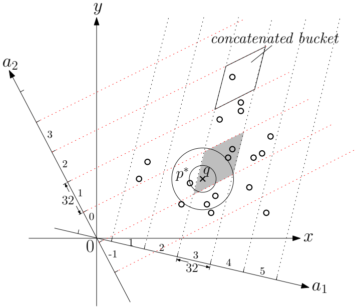
- if $\|u, v\| \leq 1$, then $Pr[g(u) = g(v)] \geq p_1^k$,
- if $\|u, v\| > 2$, then $Pr[g(u) = g(v)] \leq p_2^k$.

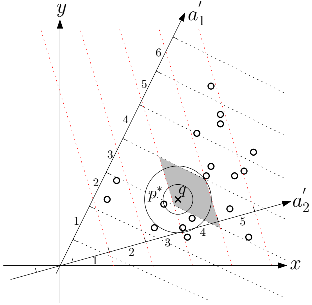
Therefore, the function family $\mathcal{G}$ is (p_1^k, p_2^k)-sensitive.

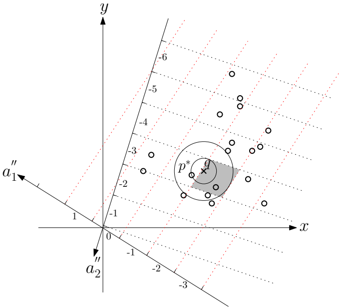
Remark. By a hash function $g \in \mathcal{G}$, the expected number of false positives is reduced to $p_2^k \cdot n$. However, in the meanwhile, the probability for a point in $B(q, 1)$ being hashed into $g(q)$ also decreases to as small as p_1^k.

The Repeating Trick

To increase the probability for a near neighbor being hashed into the same bucket of q, we repeatedly use different hash functions from $\mathcal{G}$ to construct different hash tables.







The LSH Technique

For an integer L, the LSH constructs L hash tables for P as follows:

- Independently and uniformly choose L functions $g_1, g_2, \cdots, g_L$ from the (p_1^k, p_2^k)-sensitive function family $\mathcal{G}$.
- For each g_i, construct a hash table for P by hashing each point $u \in P$ into bucket $g_i(u)$.

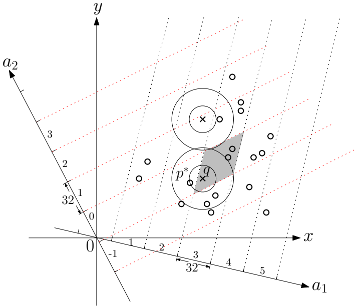
The $(1, 2)$-Near Neighbor Query Algorithm

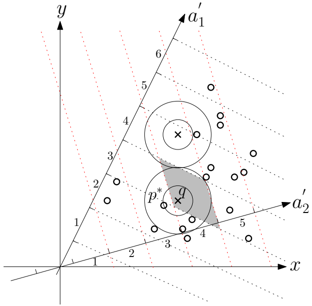
For a query point q, inspect the L hash buckets $g_1(q), \cdots, g_L(q)$ by checking each point u therein:

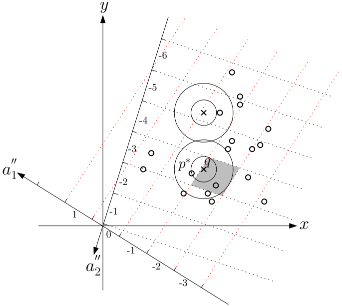
- If $\|u, q\| \leq 2$, then return u.
- Otherwise, if so far in total $3 \cdot L$ or all the points in the L buckets have been checked, then terminate and return nothing.

Query Examples

Theoretically speaking, we do need to construct a sufficiently large number of hash tables to ensure correctness. However, in most cases, about 10 hash tables are enough to answer queries. In this example, we only need three.







Correctness

For a fixed query point q, consider the following two events:

- E_1: If there exists a point $u \in B(q, 1)$, then $g_i(u) = g_i(q)$ for some $i \in \{1, 2, \cdots, L\}$.
- E_2: The total number of false positives in the L buckets $g_1(q), g_2(q), \cdots, g_L(q)$ is less than $3 \cdot L$.

Lemma: When both E_1 and E_2 hold at the same time, the query algorithm correctly answers an $(1, 2)$-near neighbor query with q.

Correctness

Proof. Let $|g_i(q)|$ be the number points in the bucket $g_i(q)$. Observe that the query algorithm examines at most $\min\{\sum_i |g_i(q)|, 3 \cdot L\}$ points.

When $\sum_i |g_i(q)| < 3 \cdot L$, by the fact that E_1 holds, if there exists $u \in B(q, 1)$, then u is in at least one of the L buckets. Thus, u must have been checked. Hence, a point in $B(q, 2)$ must be returned. On the other hand, if $B(q, 1) = \emptyset$, then either reporting a point in $B(q, 2)$ or not is correct.

When the algorithm has checked $3 \cdot L$ points, since E_2 holds, there must be at least one point in $B(q, 2)$. Hence, one such point will be returned.

$\square$

Next, we show that:

By setting the values of k and L carefully, both the two events E_1 and E_2 hold at the same time with at least constant probability.

In other words, the query algorithm correctly answers an $(1, 2)$-near neighbor query with q with at least constant probability.

Before we jump into the technical details, let us first get an idea of the basic direction to set k and L.

On one hand, as the expected number of false positives in $g_i(q)$ is $p_2^k \cdot n$, its total expected number over all the L buckets is $L \cdot p_2^k \cdot n$. If we can make this total expectation $\leq L$, then its actual value is not likely to be much larger than L. As a result, $L \cdot p_2^k \cdot n \leq L \Rightarrow k \geq \log_{1/p_2} n$.

On the other hand, since $Pr[g_i(u) = g_i(q)] \geq p_1^k$ for a point $u \in B(q, 1)$, the probability of $g_i(u) \neq g_i(q)$ for all the L buckets is $\leq (1 - p_1^k)^L$. We will show that this probability is no more than a constant when $L \geq 1/p_1^k$. As a result, the probability of at least one $g_i(u) = g_i(q)$ among all the L buckets is $\geq 1 - (1 - p_1^k)^L$ which is greater than a constant.

Thus, we set $k = \lceil \log_{1/p_2} n \rceil$ and $L = \lceil \frac{\sqrt{n}}{p_1} \rceil \geq \lceil \frac{n^\rho}{p_1} \rceil \geq \lceil \frac{1}{p_1^k} \rceil$ for $\rho = \frac{\log 1/p_1}{\log 1/p_2} \leq 0.5$.

In what follows, we will prove that both $Pr[E_1]$ and $Pr[E_2]$ are greater than a constant under the above values of k and L.

Preliminary 1: Markov's Inequality

For a nonnegative random integer variable X and $t > 0$, we have:

$$Pr[X \geq t] \leq \frac{E[x]}{t}.$$

Proof.

$$\begin{aligned} E[X] &= \sum_{x} x \cdot Pr[X = x] \\ &\geq \sum_{x \geq t} x \cdot Pr[X = x] \\ &\geq t \sum_{x \geq t} Pr[X = x] \\ &= t \cdot Pr[X \geq t] \end{aligned}$$

□

Preliminary 2:

For $x \geq 1$, $(1 - \frac{1}{x})^x \leq \frac{1}{e}$ holds.

Proof. By the well-known inequality $1 + y \leq e^y$ for $|y| \leq 1$, we have:

$$(1 - \frac{1}{x})^x \leq e^{-\frac{1}{x} \cdot x} = \frac{1}{e}$$

for $x \geq 1$.

$\square$

Preliminary 3: Union Bound

For two events A and B, we have:

$$Pr[A \cup B] = Pr[A] + Pr[B] - Pr[A \cap B] \leq Pr[A] + Pr[B].$$

The event

- E_1: If there exists a point $u \in B(q,r)$, then $g_i(u) = g_i(q)$ for some $i \in \{1, 2, \cdots, L\}$.

holds with at least probability of $1 - \frac{1}{e}$, for $k = \lceil \log_{1/p_2} n \rceil$ and $L = \lceil \frac{\sqrt{n}}{p_1} \rceil$.

Proof. Since for a point $u \in B(q,1)$, we have $Pr[g_i(u) = g_i(q)] \geq p_1^k$ for $\forall i = 1, \ldots, L$. Thus, $Pr[\bigwedge_{i=1}^{L} g_i(u) \neq g_i(q)] \leq (1 - p_1^k)^L$.

As $k = \lceil \log_{1/p_2} n \rceil$, we have $p_1^k \geq \frac{p_1}{n^\rho} \geq \frac{p_1}{\sqrt{n}} \geq \frac{1}{L}$. Thus,

$$Pr[\bigwedge_{i=1}^{L} g_i(u) \neq g_i(q)] \leq (1 - p_1^k)^L \leq (1 - \frac{1}{L})^L \leq \frac{1}{e}.$$

Therefore, $Pr[E_1] = 1 - Pr[\bigwedge_{i=1}^{L} g_i(u) \neq g_i(q)] \geq 1 - \frac{1}{e}$.

□

The event

- E_2: The total number of false positives in the L buckets $g_1(q), g_2(q), \ldots, g_L(q)$ is less than $3 \cdot L$.

holds with at least probability of $\frac{2}{3}$, for $k = \lceil \log_{1/p_2} n \rceil$ and $L = \lceil \frac{\sqrt{n}}{p_1} \rceil$.

Proof. The expected number of false positive in $g_i(q)$ is at most $p_2^k \cdot n \leq 1$. Denote by X the random variable of the total number of false positives over all $g_i(q)$'s. Thus, $E[X] \leq L$.

By Markov's inequality, we have $Pr[X \geq 3 \cdot L] \leq \frac{E[X]}{3 \cdot L} \leq \frac{1}{3}$. Therefore, $Pr[E_2] = 1 - Pr[X \geq 3 \cdot L] \geq \frac{2}{3}$.

□

Finally, by the Union Bound, $Pr[\bar{E}_1 \cup \bar{E}_2] \leq Pr[\bar{E}_1] + Pr[\bar{E}_2] \leq \frac{1}{e} + \frac{1}{3}$. Hence, $Pr[E_1 \cap E_2] \geq 1 - \frac{1}{e} - \frac{1}{3} = \frac{2}{3} - \frac{1}{e}$.

Therefore,

> There exists a (p_1, p_2)-sensitive family such that by setting $k = \lceil \log_{1/p_2} n \rceil$ and $L = \lceil \frac{\sqrt{n}}{p_1} \rceil$, the LSH correctly answers an $(1, 2)$-near neighbor query with probability at least $\frac{2}{3} - \frac{1}{e}$.

Query Time

For a query point q, the time for computing $g_1(q), \cdots, g_L(q)$ is $O(d \cdot k \cdot L)$, and the time for checking at most $3 \cdot L$ points is $O(d \cdot L)$. Thus, the total query time is bounded by $O(d \cdot k \cdot L) = O(d \cdot \sqrt{n} \cdot \log n)$.

Space

The space consumption consists of two parts: (i) the space $O(d \cdot n)$ for storing P, and (ii) the space $O(n \cdot L) = O(n^{1.5})$ for the L hash tables. Hence, the total space consumption is $O(d \cdot n + n^{1.5})$.

Remark. The value $L = \lceil \frac{\sqrt{n}}{p_1} \rceil$ is only valid for $\rho = \frac{\log 1/p_1}{\log 1/p_2} \leq 0.5$ for some specific (p_1, p_2)-sensitive families. In fact, for any such family this bound does not always hold, in which case, we can only bound $L = \lceil \frac{n^\rho}{p_1} \rceil$.

Nevertheless, all our previous analysis applies to any (p_1, p_2)-sensitive family $\mathcal{H}$ (and hence, $\mathcal{G}$) by using $L = \lceil \frac{n^\rho}{p_1} \rceil$. In other words, both query time and space consumption essentially depend on the value of ρ.

Different families $\mathcal{H}$ have various ρ values, and hence would result in different performance. The smaller value of ρ the better performance can be achieved.

A (p_1, p_2)-Sensitive Family

A well-known (p_1, p_2)-sensitive family $\mathcal{H} = \{h : \mathbb{R}^d \to \mathbb{N}\}$ with $\rho \leq 0.5$ for the Euclidean distance has the following form:

$$h(u) = \lfloor \frac{\vec{a} \cdot \vec{u} + b}{w} \rfloor,$$

where:

- $\vec{a}$ is a d-dimensional vector, whose each coordinate is chosen independently from the standard Gaussian Distribution $N(0, 1)$;
- w is an appropriate integer (e.g., $w = 32$); and
- b is a real value uniformly drawn from the range $[0, w)$.