Approximate Nearest Neighbors Search in High Dimensions and Locality-Sensitive Hashing

PAPERS


Overview

- Introduction
- Locality Sensitive Hashing (Aneesh)
- Hash Functions Based on $p$-Stable Distributions (Michael)
Overview

- Introduction
  - Nearest neighbor search problems
  - Higher dimensions
  - Johnson-Lindenstrauss lemma
- Locality Sensitive Hashing (Aneesh)
- Hash Functions Based on $p$-Stable Distributions (Michael)
Problem
Problem Statement

Today’s Talks: **NN-search in high dimensional spaces**

- **Given**
  - Point set \( P = \{p_1, \ldots, p_n\} \)
  - a query point \( q \)

- **Find**
  - \([\varepsilon\text{-approximate}] \) nearest neighbor to \( q \) from \( P \)

- **Goal:**
  - Sublinear query time
  - “Reasonable” preprocessing time & space
  - “Reasonable” growth in \( d \) (exponential not acceptable)
Example Application: Feature spaces

- Vectors $\mathbf{x} \in \mathbb{R}^d$ represent characteristic features of objects
- There are often many features
- Use nearest neighbor rule for classification / recognition
Applications

“Real World” Example: Image Completion
“Real World” Example: Image Completion

- Iteratively fill in pixels with best match (+ multi scale)
- Typically $5 \times 5$ … $9 \times 9$ neighborhoods, i.e.: dimension 25 … 81
- Performance limited by nearest neighbor search
- 3D version: dimension 81 … 729
Higher Dimensions
Higher Dimensions are Weird

Issues with High-Dimensional Spaces:

- $d$-dimensional space:
  $d$ independent neighboring directions to each point

- Volume-distance ratio explodes

$$\text{vol}(r) \in \Theta(r^d)$$
No Grid Tricks

Regular Subdivision Techniques Fail

- Regular $k$-grids contain $k^d$ cells
- The “grid trick” does not work
- Adaptive grids usually also do not help
- Conventional integration becomes infeasible ($\Rightarrow$ MC-approx.)
- Finite element function representation become infeasible
More Weird Effects:

- Dart-throwing anomaly
  - Normal distributions gather prob.-mass in thin shells
  - [Bishop 95]

- Nearest neighbor ~ farthest neighbor
  - For unstructured points (e.g. iid-random)
  - Not true for certain classes of structured data
  - [Beyer et al. 99]
Johnson-Lindenstrauss Lemma
**Johnson-Lindenstrauss Lemma**

**JL-Lemma:** [Dasgupta et al. 99]

- Point set $P$ in $\mathbb{R}^d$, $n := \#P$
- There is $f: \mathbb{R}^d \rightarrow \mathbb{R}^k$, $k \in O(\varepsilon^{-2} \ln n)$
  \[ (k \geq 4(\varepsilon^2/2 - \varepsilon^3/3)^{-1} \ln n) \]
- …that preserves all inter-point distances up to a factor of $(1 + \varepsilon)$

Random orthogonal linear projection works with probability $\geq (1 - 1/n)$
What Does the JL-Lemma Imply?

Pairwise distances in small point set $P$ (sub-exponential in $d$) can be well-preserved in low-dimensional embedding.

What does it not say?

Does not imply that the points themselves are well-represented (just the pairwise distances)
Experiment

[Graph showing dimensionality reduction with distance error and average reconstruction error.]
Difference Vectors

- Normalize (relative error)
- Pole yields bad approximation
- Non-pole area much larger (high dimension)
- Need large number of poles (exponential in $d$)
Overview

• Introduction

• Locality Sensitive Hashing
  • Approximate Nearest Neighbors
  • Big picture
  • LSH on unit hypercube
    - Setup
    - Main idea
    - Analysis
    - Results

• Hash Functions Based on $p$-Stable Distributions
Approximate Nearest Neighbors
ANN: Decision version

Input: $P, q, r$

Output:

- If there is a NN, return yes and output one ANN
- If there is no ANN, return no
- Otherwise, return either
ANN: Decision version

Input: \( P, q, r \)
Output:

- If there is a NN, return yes and output one ANN
- If there is no ANN, return no
- Otherwise, return either
ANN: Decision version

Input: $P, q, r$
Output:
- If there is a NN, return yes and output one ANN
- If there is no ANN, return no
- Otherwise, return either
ANN: Decision version

General ANN

\[ \uparrow \quad \text{PLEB} \quad \downarrow \]

Decision version + Binary search
### ANN: previous results

<table>
<thead>
<tr>
<th>Method</th>
<th>Query time</th>
<th>Space used</th>
<th>Preprocessing time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vornoi</td>
<td>$O(2^d \log n)$</td>
<td>$O(n^{d/2})$</td>
<td>$O(n^{d/2})$</td>
</tr>
<tr>
<td>Kd-tree</td>
<td>$O(2^d \log n)$</td>
<td>$O(n)$</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>LSH</td>
<td>$O(n^\rho \log n)$</td>
<td>$O(n^{1+\rho})$</td>
<td>$O(n^{1+\rho} \log n)$</td>
</tr>
</tbody>
</table>
LSH: Big picture
Locality Sensitive Hashing

• Remember: solving decision ANN
• Input:
  • No. of points: $n$
  • Number of dimensions: $d$
  • Point set: $P$
  • Query point: $q$
• Family of hash functions:
  • Close points to same buckets
  • Faraway points to different buckets
• Choose a random function and hash $P$
• Only store non-empty buckets
• Hash $q$ in the table

• Test every point in $q$’s bucket for ANN

• Problem:
  • $q$’s bucket may be empty
**LSH: Big Picture**

- **Solution:**
  - Use a number of hash tables!
  - We are done if any ANN is found
LSH: Big Picture

• Problem:
  • Poor resolution too many candidates!
  • Stop after reaching a limit, small probability
• Want to find a hash function:

If $u \in B(q, r)$ then $\Pr[h(u) = h(q)] \geq \alpha$

If $u \notin B(q, R)$ then $\Pr[h(u) = h(q)] \leq \beta$

$r < R, \quad \alpha \gg \beta$

• $h$ is randomly picked from a family
• Choose

$$R = r(1 + \varepsilon)$$
LSH on unit Hypercube
Setup: unit hypercube

- Points lie on hypercube: $H^d = \{0,1\}^d$
- Every point is a binary string
- Hamming distance ($r$):
  - Number of different coordinates

$u : 0010111101$
$v : 0110001000$
Setup: unit hypercube

- Points lie on hypercube: $H^d = \{0,1\}^d$
- Every point is a binary string
- Hamming distance ($r$):
  - Number of different coordinates

\[
\begin{align*}
\mathbf{u} : & \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1 \\
\mathbf{v} : & \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0
\end{align*}
\]
Main idea
• Define family $F$:

Given: Hypercube $H^d$, point $b = (b_1, \ldots, b_d)$

$h \in F : \left\{ h_i(b) = b_i \left| b = (b_1, \ldots, b_d) \in H^d, \text{ for } i = 1, \ldots, d \right. \right\}$

$\alpha = 1 - \frac{r}{d}$, \hspace{0.5cm} $\beta = 1 - \frac{r(1+\varepsilon)}{d}$

• Intuition: compare a random coordinate
• Called: $(r, r(1+\varepsilon), \alpha, \beta)$-sensitive family
Hash functions for hypercube

- Define family $G$:

$$\text{Given } : b \in H^d, F$$

$$g \in G :$$

$$\begin{cases} g : \{0,1\}^d \rightarrow \{0,1\}^k \bigg| g(b) = \left( h^1(b), \ldots, h^k(b) \right), \text{ for } h^i \in F \end{cases}$$

$$\alpha' = \left( 1 - \frac{r}{d} \right)^k = \alpha^k, \quad \beta' = \left( 1 - \frac{r(1+\varepsilon)}{d} \right)^k = \beta^k$$

- Intuition: Compare $k$ random coordinates
- Choose $k$ later – logarithmic in $n$ \quad J-L lemma
Constructing hash tables

• Choose $g_1, \ldots, g_\tau$ uniformly at random from $G$
  • Constructing $\tau$ hash tables, hash $P$
  • Will choose $\tau$ later

\[
\begin{array}{c}
g_1 \\
g_2 \\
g_\tau
\end{array}
\]
LSH: ANN algorithm

- Hash \( q \) into each \( g_1, \ldots, g_\tau \)
  - Check colliding nodes for ANN
  - Stop if more than \( 4\tau \) collisions, return fail
Details...
Choosing parameters

• Choose $k$ and $\tau$ to ensure constant probability of:
  • Finding an ANN if there is a NN
  • Few collisions ($< 4\tau$) when there is no ANN

Define: $\rho = \frac{\ln 1/\alpha}{\ln 1/\beta}$

Choose: $k = \frac{\ln n}{\ln 1/\beta}$, $\tau = 2n^\rho$
Analysis of LSH

- Probability of finding an ANN if there is a NN
  - Consider a point \( p \in B(q, r) \) and a hash function \( g_i \in G \)

\[
\Pr[g_i(p) = g_i(q)] \geq \alpha^k \\
= \alpha^{\ln n / \ln 1/\beta} \\
= n^{-\rho}
\]
Analysis of LSH

• Probability of finding an ANN if there is a NN
  • Consider a point \( p \in B(q, r) \) and a hash function \( g_i \in G \)

\[
\Pr[g_i \text{ hashes } p \text{ and } q \text{ to different locations}] \leq 1 - n^{-\rho}
\]
\[
\Pr[p \text{ and } q \text{ collide at least once in } \tau \text{ tables}] \geq 1 - (1 - n^{-\rho})^\tau
\]
\[
\geq 1 - 1/e^2
\]
\[
> \frac{4}{5}
\]
Analysis of LSH

• Probability of collision if there is no ANN
  • Consider a point \( p \not\in B(q, r(1+\varepsilon)) \) and a hash function \( g \in G \)

\[
\Pr[g(p) = g(q)] \leq \beta^k = \exp\left(\ln(\beta) \cdot \frac{\ln n}{\ln 1/\beta}\right) = \frac{1}{n}
\]
Analysis of LSH

- Probability of collision if there is no ANN
  
  Consider a point $p \notin B(q, r(1+\epsilon))$ and a hash function $g \in G$

  \[
  E[\text{collisions with } q \text{ in a table}] \leq 1
  \]

  \[
  E[\text{collisions with } q \text{ in } \tau \text{ tables}] \leq \tau
  \]

  \[
  \Pr[\geq 4\tau \text{ collisions}] \leq \frac{\tau}{4\tau} = \frac{1}{4}
  \]

  \[
  \Pr[< 4\tau \text{ collisions}] \geq \frac{3}{4}
  \]
Results
Complexity of LSH

• **Given:** $(r, r(1+\varepsilon), \alpha, \beta)$-sensitive family for Hypercube
  
  • Can answer Decision-ANN with:
    
    $$O\left( d n + n^{1+\rho} \right)$$ space

    $$O\left( d n^\rho \right)$$ query time

• **Show:**

  $$\rho = \frac{\ln 1/\alpha}{\ln 1/\beta} = \frac{\ln \left( 1 - \frac{r}{d} \right)}{\ln \left( 1 - \frac{(1+\varepsilon)r}{d} \right)} \leq \frac{1}{1+\varepsilon}$$
Complexity of LSH

• **Given:** \((r, r(1+\varepsilon), \alpha, \beta)\)-sensitive family for Hypercube
  • Can answer Decision-ANN with:

\[
O\left(dn + n^{1+1/(1+\varepsilon)}\right)_{\text{space}}
\]

\[
O\left(dn^{1/(1+\varepsilon)}\right)_{\text{query time}}
\]
• Can amplify success probability
  • Build $O(\log n)$ structures
  • Can answer Decision-ANN with:

\[
O\left( dn + n^{1+1/(1+\varepsilon)} \log n \right) \text{space}
\]

\[
O\left( dn^{1/(1+\varepsilon)} \log n \right) \text{query time}
\]
Complexity of LSH

• Can answer ANN on the Hypercube:
  • Build $O\left(\varepsilon^{-1} \log n\right)$ structures with $r_i = (1 + \varepsilon)^i$

\[
O\left(dn + n^{1+1/(1+\varepsilon)} \varepsilon^{-1} \log^2 n\right) \text{ space}
\]

\[
O\left(dn^{1/(1+\varepsilon)} \log\left(\varepsilon^{-1} \log n\right)\right) \text{ query time}
\]
• Randomized Monte-Carlo algorithm for ANN

• First truly sub-linear query time for ANN

• Need to examine only logarithmic number of coordinates

• Can be extended to any metric space if we can find a hash function for it!

• Easy to update dynamically

• Can reduce ANN in $\mathbb{R}^d$ to ANN on hypercube
Overview

- Introduction
- Locality Sensitive Hashing
- Hash Functions Based on $p$-Stable Distributions
  - The basic idea
  - The details (more formal)
  - Analysis, experimental results
Idea:

• Hash function is a projection to a line of random orientation
• One composite hash function is a random grid
• Hashing buckets are grid cells
• Multiple grids are used for prob. amplification
• Jitter grid offset randomly (check only one cell)
• Double hashing: Do not store empty grid cells
LSH by Random Projections

Basic Idea:
Questions:

• What distribution should be used for the projection vectors?
• What is a good bucket size?
• Local sensitivity:
  • How many lines per grid?
  • How many hash grids overall?
  • Depends on sensitivity (as explained before)
• How efficient is this scheme?
The Details
Distribution for the Projection Vectors:

- Need to make the projection process formally accessible
- Mathematical tool: $p$-stable distributions
$p$-Stable Distributions:

A prob. distribution $D$ is called $p$-stable $⇔$

- For any $v_1, \ldots, v_n ∈ \mathbb{R}$
- And i.i.d. random variables $X_1, \ldots, X_n ∼ D$

$$\sum_{i} v_i X_i \text{ has the same distribution as } \left[\sum_{i} |v_i|^{1/p}\right]^{1/p} X$$

where $X ∼ D$
Gaussian Normal Distributions are 2-stable
Other distributions:

- Cauchy distribution \( \frac{1}{\pi(1 + x^2)} \) is **1-stable**
  (must have infinite variance
c so that the central limit theorem is not violated)

- Distributions exists for \( p \in (0,2] \)
- No closed form, but can be sampled
- Sampling sufficient for LSH-algorithm
Projection

Projection Algorithm:

- Chose $p$ according to metric of the space $l_p$
- Compute vector with entries according to a $p$-stable distribution
  [for example: Gaussian noise entries]
- Each vector $v_i$ yields a hash function $h_i$
- Compute: $h_i(x) = \left\lfloor \frac{\langle v_i, x \rangle + b}{r} \right\rfloor$ where $r$ is a random value $\in [0…r]$ and $b$ is the bucket size
Locality Sensitive Hash Functions

\( H = \{ h: S \rightarrow U \} \) is \((r_1, r_2, \alpha, \beta)\)-sensitive :

\[
\begin{align*}
\nu \in B(q, r_1) & \implies \Pr(\text{collision}(p, q)) \geq \alpha \\
\nu \not\in B(q, r_2) & \implies \Pr(\text{collision}(p, q)) \leq \beta 
\end{align*}
\]

Performance

\[
\rho = \frac{\ln \alpha}{\ln \beta} \quad (O(dn + n^{1+\rho}) \text{ space, } O(dn^\rho) \text{ query time})
\]
Locality Sensitivity

Computing the Locality “Sensitivity”

Distance \( c = \|v_1 - v_2\|_p \)

\( c \)X-distributed, \( X \) from \( p \)-stable distr.

\[
Pr(\text{collision}) = \int_0^r \frac{1}{c} f_p \left( \frac{t}{c} \right) \left( 1 - \frac{t}{r} \right) dt
\]

The constructed family of hash functions is \((r_1, r_2, \alpha, \beta)\)-sensitive for

\[
\alpha = p(1), \quad \beta = p(c), \quad r_2/r_1 = c
\]
Numerical Computation

Numerical result: \( \rho \sim 1/c = 1/(1+\varepsilon) \)

\[
\rho = \frac{\ln \alpha}{\ln \beta}, \quad O(dn + n^{1+\rho}) \text{ space, } O(dn^\rho) \text{ query time}
\]

[Datar et al. 04]
Width Parameter $r$

- Intuitively: In the range of ball radius
- Num. result: not too small (too large increases $k$)
- Practice: factor 4 (E2LSH manual)
Experimental Results
Comparison with ANN (Mount, Arya, kD/BBD-trees)

MNIST handwritten digits, $60000 \times 28^2$ pix ($d=784$)
Remarks:

• ANN with $c = 10$ is comparably fast and 65% correct, but there are no guarantees [Indyk]

• LSH needs more memory: 1.2GB vs. 360MB [Indyk]

• Empirically, LSH shows linear performance when forced to use linear memory [Goldstein et al. 05]

• Benchmark searches only for points in the data set, LSH is much slower for negative results [Goldstein et al. 05, report ~1.5 ord. of mag.]