Approximate Nearest Neighbor Search
in high dimensional space

Шугаепов Ильнур

VK.com
Performance Advertising Team
ilnur.shug@gmail.com

Higher School of Economics, 2020

- **In session recommendations**

  Given user's u current session $(i_1, i_2, \ldots, i_m)$ in terms of viewd products, goal is to recommend product $i_{m+1} \in I$, where I - large set of items.

- **In session recommendations**

  Given user's u current session $(i_1, i_2, \dots, i_m)$ in terms of viewd products, goal is to recommend product $i_{m+1} \in I$, where I - large set of items.

- **Image retrieval**

  Find images (semantically) similar to image uploaded by user.

- **In session recommendations**

  Given user's u current session $(i_1, i_2, \ldots, i_m)$ in terms of viewd products, goal is to recommend product $i_{m+1} \in I$, where I - large set of items.

- **Image retrieval**

  Find images (semantically) similar to image uploaded by user.

- **Offline evaluation of RecSys**

  Assume we have user's vector representation $\mathbf{u}$, set of user's true relevant items, we would like to compute some $Metric@k$.
  We need to find k items from I with highest similarity score wrt $\mathbf{u}$ (e.g. dot product, cosine similarity).

Motivation
Main Goal

Efficient search of nearest neighbours

Table of Contents

Definition

Given $(\mathcal{X}, \rho)$ *- metric space*[a], $X \subseteq \mathcal{X}$ *- set of* n *points and* $q \in \mathcal{X}$ *- query point, task of Exact k-NN is to find a set* $kNN(q) \subseteq X$ *such that* $|kNN(q)| = k$ *and*

$$\forall x \in kNN(q)\ \forall x' \in X \setminus kNN(q)\ \ \big(\rho(q,x) \leq \rho(q,x')\big)$$

[a] https://en.wikipedia.org/wiki/Metric_space

Definition

Given $(\mathcal{X}, \rho)$ *- metric space*[a], $X \subseteq \mathcal{X}$ *- set of* n *points and* $q \in \mathcal{X}$ *- query point, task of Exact k-NN is to find a set* $kNN(q) \subseteq X$ *such that* $|kNN(q)| = k$ *and*

$$\forall x \in kNN(q)\ \forall x' \in X \setminus kNN(q)\ \ \big(\rho(q,x) \leq \rho(q,x')\big)$$

[a] https://en.wikipedia.org/wiki/Metric_space

Examples of spaces

- $(\mathbb{R}^d, \|\cdot\|_2)$ — Euclidian space
- $\left(\mathbb{R}^d, \arccos \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\| \cdot \|\mathbf{y}\|}\right)$ — $\mathbb{R}^d$ with cosine distance
- ...

Main problems of Exact k-NN

- linear search time complexity of naive solution
- high complexity in high dimensional spaces

Commonly used definitions of the approximate neighbor search

- Search with predefined accuracy

Definition (ε-NN [5])

$$\forall x \in \varepsilon\text{-}NN(q) \ \ (\rho(q,x) \leq (1+\varepsilon)\rho(q,x_k)),$$

where $x_k \in kNN(q)$ - true k-th nearest neighbor of query point q.

Commonly used definitions of the approximate neighbor search

- Search with predefined accuracy

Definition (ε-NN [5])

$$\forall x \in \varepsilon\text{-}NN(q) \ \ (\rho(q,x) \leq (1+\varepsilon)\rho(q,x_k)),$$

where $x_k \in kNN(q)$ - true k-th nearest neighbor of query point q.

- Search with probability guarantee of finding true k closest points

Definition (ANN)

$$\frac{|ANN(q) \cap kNN(q)|}{k} \geq \delta,$$

i.e. we guarantee that fraction is above a certain threshold $\delta \in [0,1]$.

Commonly used definitions of the approximate neighbor search

- Search with predefined accuracy

Definition (ε-NN [5])

$$\forall x \in \varepsilon\text{-}NN(q) \;\; (\rho(q,x) \leq (1+\varepsilon)\rho(q,x_k)) \, ,$$

where $x_k \in kNN(q)$ - true k-th nearest neighbor of query point q.

- Search with probability guarantee of finding true k closest points

Definition (ANN)

$$\frac{|ANN(q) \cap kNN(q)|}{k} \geq \delta,$$

i.e. we guarantee that fraction is above a certain threshold $\delta \in [0,1]$.

We will mainly focus on the latter definition

In terms of speed/guaranties

- Speedup vs Recall plot

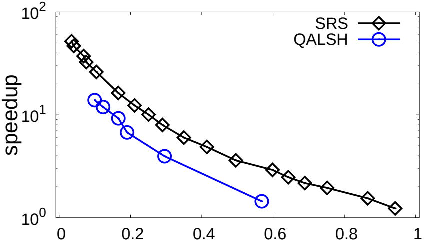


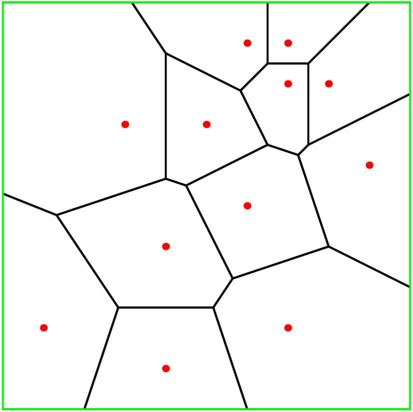
Рис.: Speedup vs Recall

- Recall vs Query time
- Queries per second vs Recall

Other aspects: data structure construction time, footprint size, and scalability

[1] `https://github.com/erikbern/ann-benchmarks`

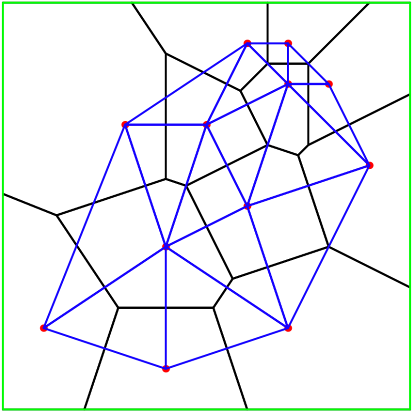
Table of Contents



Consider set of points $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \subseteq \mathbb{R}^2$. With every point $\mathbf{x}_i$ associated region $R(\mathbf{x}_i)$ such that $\forall \mathbf{x} \in R(\mathbf{x}_i)\ \forall j \neq i\ (\|\mathbf{x} - \mathbf{x}_i\|_2 < \|\mathbf{x} - \mathbf{x}_j\|_2)$

[2] `https://en.wikipedia.org/wiki/Voronoi_diagram`

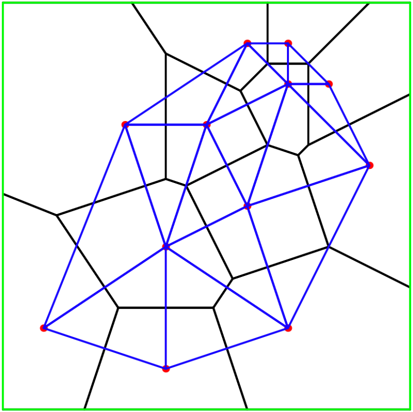
Dual graph for a Voronoi diagram (Delaunay Graph)[3]



Graph $G = \langle V, E \rangle$, where $V = X$ and $(u, v) \in E$ iff $R(u)$ and $R(v)$ are neighbours

[3] https://en.wikipedia.org/wiki/Delaunay_triangulation

Dual graph for a Voronoi diagram (Delaunay Graph)[3]



Graph $G = \langle V, E \rangle$, where $V = X$ and $(u, v) \in E$ iff $R(u)$ and $R(v)$ are neighbours

Algorithm 2.2 Greedy_Search$(q, v_{\text{entry_point}})$

$v_{cur} \leftarrow v_{\text{entry\_point}}$
$\delta_{\min} \leftarrow \rho(q, v_{cur})$; $\quad v_{next} \leftarrow NIL$
for all $u \in N(v_{cur})$ **do**
    $\delta \leftarrow \rho(q, u)$
    if $\delta < \delta_{\min}$ **then**
        $\delta_{\min} \leftarrow \delta$
        $v_{next} \leftarrow u$
if $v_{next} = NIL$ **then**
    return v_{cur}
return Greedy_Search(q, v_{next})

[3] https://en.wikipedia.org/wiki/Delaunay_triangulation

Properties of Greedy Search in Delaunay Graph

- *GS will always find true nearest neighbour to query point*
- *Result could be generilized to other metric spaces*
- *GS could be linear in the worst case*

Properties of Greedy Search in Delaunay Graph

- *GS will always find true nearest neighbour to query point*
- *Result could be generilized to other metric spaces*
- *GS could be linear in the worst case*

Lets assume that along with existing edges G also contains *long range* edges conecting far away nodes

Properties of Greedy Search in Delaunay Graph

- *GS will always find true nearest neighbour to query point*
- *Result could be generilized to other metric spaces*
- *GS could be linear in the worst case*

Lets assume that along with existing edges G also contains *long range* edges conecting far away nodes

Definition (Navigatable Small World)

Graphs with $\mathcal{O}(\log^{\alpha} n)$ scalability ($\alpha \geq 1$) of the GS algorithm are called navigable small world graphs

- Build graph G that approximates Delaunay Graph and has NSW property
- Greedy search for ANN (with ability to choose search accuracy)
- GS time complexity $\mathcal{O}(\log^2 n)$ (avg. node degree multiplied by avg. path length)

[4] `https://github.com/nmslib/nmslib`

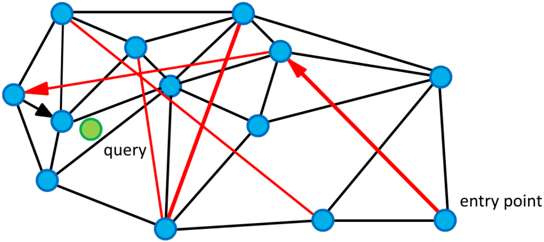


Рис.: Vertices are the data in metric space, black edges are the approximation of the Delaunay graph, and red edges are long range links for logarithmic scaling. Arrows show a sample path of the greedy algorithm from the entry point to the query (shown green)

Algorithm 2.3 k-NNSearch(q, m, k)

$tempRes \leftarrow \emptyset; candidates \leftarrow \emptyset; visited \leftarrow \emptyset; result \leftarrow \emptyset$ ▷ empty TreeSets
for $i = 1, \dots, m$ **do**
    put random entry point in candidates
    $tempRes \leftarrow \emptyset$
    loop
        get element c closest from candidates to q
        remove c from candidates
        if c is further than k-th element from result **then**
            break
        for all $e \in N(c)$ **do**
            if $e \notin visited$ **then**
                add e to visited, candidates, tempRes
    add objects from tempRes to result
return best k elements from result

Algorithm 2.4 Insert(x, f, w)

$N_x \leftarrow k$-NNSearch(x, w, f)
for all $u \in N_x$ **do**
  $N(x) \leftarrow N(x) \cup \{u\}$
  $N(u) \leftarrow N(u) \cup \{x\}$

Emperical study showed that graph G constructed with this inserting procedure is a good Delaunay approximation and has NSW property

m

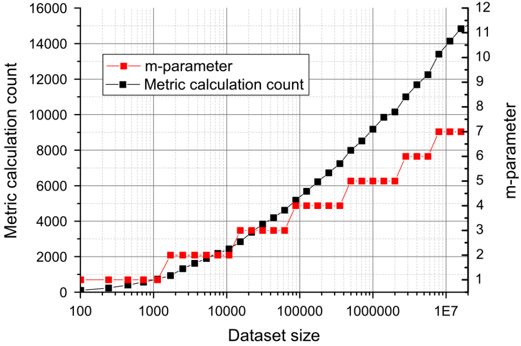


Рис.: Distance calculations and the value of m to get a 0.999 recall versus the size of the dataset for $d = 20$. The metric calculation count has $C \log^2 n$ complexity scaling.

w for optimal recall changes slowly (logarithmically) with the dataset size

f $3d$ where d - dimensionality of space (good value for high recall)

Hierarchical Navigable Small World[5] [10]

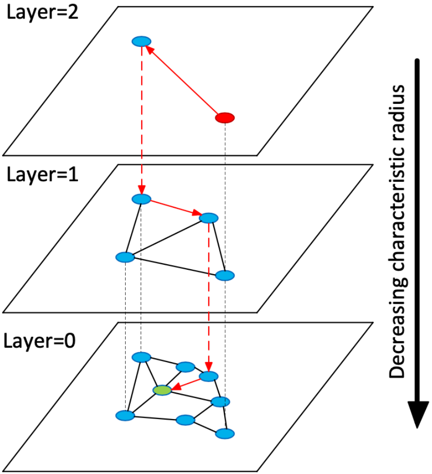


Рис.: Illustration of the Hierarchical NSW idea. The search starts from an element from the top layer (shown red). Red arrows show direction of the greedy algorithm from the entry point to the query (shown green)

[5] https://github.com/nmslib/nmslib

Table of Contents

Given $(\mathcal{X}, \rho)$ - metric space, set of object $X \subseteq \mathcal{X}$ and a hash function $h: \mathcal{X} \to [m]$.

LSH property

Similar items have larger probability to be mapped to the same bucket than dissimilar items.

Given $(\mathcal{X}, \rho)$ - metric space, set of object $X \subseteq \mathcal{X}$ and a hash function $h\colon \mathcal{X} \to [m]$.

LSH property

Similar items have larger probability to be mapped to the same bucket than dissimilar items.

To find $ANN(q)$ we need to exemine bucket of object q

Definition

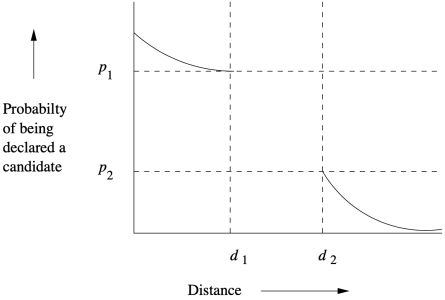
A family $\mathcal{H}$ of hash functions is called (d_1, d_2, p_1, p_2)-sensitive ($d_1 < d_2$) if for any $x, y \in \mathcal{X}$

- *if* $\rho(x, y) \leq d_1 \quad \Rightarrow \quad \mathbb{P}\{h(x) = h(y)\} \geq p_1$
- *if* $\rho(x, y) \geq d_2 \quad \Rightarrow \quad \mathbb{P}\{h(x) = h(y)\} \leq p_2$

Definition

A family $\mathcal{H}$ of hash functions is called (d_1, d_2, p_1, p_2)-sensitive ($d_1 < d_2$) if for any $x, y \in \mathcal{X}$

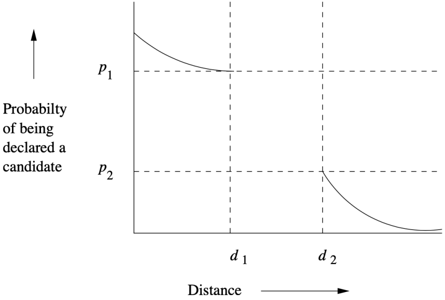
- *if* $\rho(x, y) \leq d_1 \quad \Rightarrow \quad \mathbb{P}\{h(x) = h(y)\} \geq p_1$
- *if* $\rho(x, y) \geq d_2 \quad \Rightarrow \quad \mathbb{P}\{h(x) = h(y)\} \leq p_2$



Definition

A family $\mathcal{H}$ of hash functions is called (d_1, d_2, p_1, p_2)-sensitive ($d_1 < d_2$) if for any $x, y \in \mathcal{X}$

- *if* $\rho(x, y) \leq d_1 \quad \Rightarrow \quad \mathbb{P}\{h(x) = h(y)\} \geq p_1$
- *if* $\rho(x, y) \geq d_2 \quad \Rightarrow \quad \mathbb{P}\{h(x) = h(y)\} \leq p_2$



It is possible to drive p_1 and p_2 apart while keeping d_1 and d_2 fixed

- Given family $\mathcal{H}$ we can construct family $\mathcal{H}'$ in a following way
  1. Select $h_1, h_2, \ldots, h_r \sim_{\text{i.i.d}} \mathcal{H}$,
  2. New hash function $h(x) = [h_1(x), h_2(x), \ldots, h_r(x)] \in \mathcal{H}'$
  3. $\forall x, y \in \mathcal{X}$

$$h(x) = h(y) \quad \Leftrightarrow \quad \forall i = \overline{1, r}\ (h_i(x) = h_i(y))$$

- Given family $\mathcal{H}$ we can construct family $\mathcal{H}'$ in a following way
  1. Select $h_1, h_2, \dots, h_r \sim_{\text{i.i.d}} \mathcal{H}$,
  2. New hash function $h(x) = [h_1(x), h_2(x), \dots, h_r(x)] \in \mathcal{H}'$
  3. $\forall x, y \in \mathcal{X}$
$$h(x) = h(y) \quad \Leftrightarrow \quad \forall i = \overline{1, r}\ (h_i(x) = h_i(y))$$
- Since the members of $\mathcal{H}$ are independently chosen to make a member of $\mathcal{H}'$, we can assert that $\mathcal{H}'$ is a $(d_1, d_2, (p_1)^r, (p_2)^r)$-sensitive family

- Given family $\mathcal{H}$ we can construct family $\mathcal{H}'$ in a following way
  1. Select $h_1, h_2, \dots, h_r \sim_{\text{i.i.d}} \mathcal{H}$,
  2. New hash function $h(x) = [h_1(x), h_2(x), \dots, h_r(x)] \in \mathcal{H}'$
  3. $\forall x, y \in \mathcal{X}$
$$h(x) = h(y) \quad \Leftrightarrow \quad \forall i = \overline{1, r}\ (h_i(x) = h_i(y))$$

- Since the members of $\mathcal{H}$ are independently chosen to make a member of $\mathcal{H}'$, we can assert that $\mathcal{H}'$ is a $(d_1, d_2, (p_1)^r, (p_2)^r)$-sensitive family

Remark

AND-construction lowers all probabilities, but if we choose $\mathcal{H}$ and r judiciously, we can make the small probability p_2 get very close to 0, while the higher probability p_1 stays significantly away from 0.

- Given family $\mathcal{H}$ we can construct family $\mathcal{H}'$ in a following way
  1. Select $h_1, h_2, \ldots, h_b \sim_{\text{i.i.d}} \mathcal{H}$,
  2. New hash function $h(x) = [h_1(x), h_2(x), \ldots, h_b(x)] \in \mathcal{H}'$
  3. $\forall x, y \in \mathcal{X}$
$$h(x) = h(y) \quad \Leftrightarrow \quad \exists i \; (h_i(x) = h_i(y))$$

- Given family $\mathcal{H}$ we can construct family $\mathcal{H}'$ in a following way
  1. Select $h_1, h_2, \dots, h_b \sim_{\text{i.i.d}} \mathcal{H}$,
  2. New hash function $h(x) = [h_1(x), h_2(x), \dots, h_b(x)] \in \mathcal{H}'$
  3. $\forall x, y \in \mathcal{X}$
  $$h(x) = h(y) \quad \Leftrightarrow \quad \exists i \; (h_i(x) = h_i(y))$$

- OR-construction turns (d_1, d_2, p_1, p_2)-sensitive family $\mathcal{H}$ into $(d_1, d_2, 1-(1-p_1)^b, 1-(1-p_2)^b)$-sensitive family $\mathcal{H}'$

- Given family $\mathcal{H}$ we can construct family $\mathcal{H}'$ in a following way
  1. Select $h_1, h_2, \ldots, h_b \sim_{\text{i.i.d}} \mathcal{H}$,
  2. New hash function $h(x) = [h_1(x), h_2(x), \ldots, h_b(x)] \in \mathcal{H}'$
  3. $\forall x, y \in \mathcal{X}$
$$h(x) = h(y) \quad \Leftrightarrow \quad \exists i \ (h_i(x) = h_i(y))$$

- OR-construction turns (d_1, d_2, p_1, p_2)-sensitive family $\mathcal{H}$ into $(d_1, d_2, 1-(1-p_1)^b, 1-(1-p_2)^b)$-sensitive family $\mathcal{H}'$

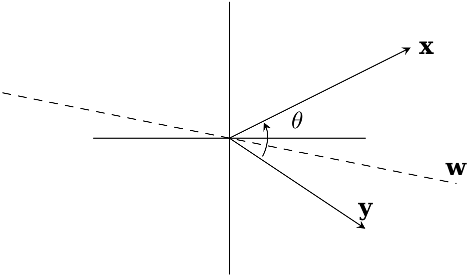
Remark

OR-construction makes all probabilities rise, but by choosing $\mathcal{H}$ and b judiciously, we can make the larger probability approach 1 while the smaller probability remains bounded away from 1.

- There are different kinds of LSH families for different distances or similarities
- We can cascade AND- and OR-constructions in any order to make the low probability close to 0 and the high probability close to 1

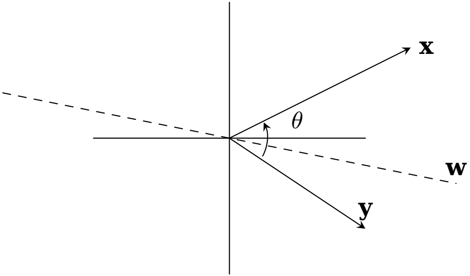
- Consider $(\mathbb{R}^n, \theta)$, where $\theta(\mathbf{x}, \mathbf{y}) = \arccos \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\| \cdot \|\mathbf{y}\|}$ - space with cosine distance

- Consider $(\mathbb{R}^n, \theta)$, where $\theta(\mathbf{x}, \mathbf{y}) = \arccos \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\| \cdot \|\mathbf{y}\|}$ - space with cosine distance
- $\mathbf{w}$ - random hyperplane through the origin



$h_{\mathbf{w}}(\mathbf{x}) = \mathrm{sign}(\langle \mathbf{w}, \mathbf{x} \rangle)$, i.e. $\mathbf{x}$ and $\mathbf{y}$ are in the same bucket iff they are on the same side wrt hyperplane $\mathbf{w}$

- Consider $(\mathbb{R}^n, \theta)$, where $\theta(\mathbf{x}, \mathbf{y}) = \arccos \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\| \cdot \|\mathbf{y}\|}$ - space with cosine distance
- $\mathbf{w}$ - random hyperplane through the origin



$h_{\mathbf{w}}(\mathbf{x}) = \mathrm{sign}(\langle \mathbf{w}, \mathbf{x} \rangle)$, i.e. $\mathbf{x}$ and $\mathbf{y}$ are in the same bucket iff they are on the same side wrt hyperplane $\mathbf{w}$

- Set of such random hyperplanes defines $(d_1, d_2, 1 - \frac{d_1}{180}, 1 - \frac{d_2}{180})$ - sensitive family $\mathcal{H}$

LSH families examples

Min-hash [6]

- Consider $(2^{\mathcal{U}}, J)$, where $\mathcal{U}$ - finite set of m elements and $J(S,S') = 1 - \frac{|S \cap S'|}{|S \cup S'|}$ - set of sets with Jaccard distance
- $\pi : [m] \to [m]$ - random permutation
- $h_\pi(S) = \min\limits_{x \in S} \pi(x)$ - *minhash* function
- It is easily shown that $\mathbb{P}\{h(S) = h(S')\} = 1 - J(S,S')$

Resume

- The LSH scheme has very nice theoretic properties
- However, as the hash functions are data-independent, the practical performance is not as good as expected in certain applications

Table of Contents

Given $(\mathcal{X}, \rho)$ - input space, set of object $X \subseteq \mathcal{X}$ and a hash function $h\colon \mathcal{X} \to V$ with $\rho'\colon V \times V \to \mathbb{R}$ hash-distance function.

Property

Distance computation using the hash values is efficient and well approximates the original distance ρ

Given $(\mathcal{X}, \rho)$ - input space, set of object $X \subseteq \mathcal{X}$ and a hash function $h\colon \mathcal{X} \to V$ with $\rho'\colon V \times V \to \mathbb{R}$ hash-distance function.

Property

Distance computation using the hash values is efficient and well approximates the original distance ρ

This strategy exploits two advantages of hash codes:

- distance using hash codes can be efficiently computed and the cost is much smaller than that of the computation in the input space
- the size of the hash codes is much smaller than the input features and hence can be loaded into memory

Definition

Learning to hash task: learning a hash function, $y = h_\theta(\mathbf{x})$, such that the NNS in the V space is efficient and the result is an effective approximation of the true NNS result in the $\mathcal{X}$ space.

Definition

Learning to hash task: learning a hash function, $y = h_\theta(\mathbf{x})$, such that the NNS in the V space is efficient and the result is an effective approximation of the true NNS result in the $\mathcal{X}$ space.

Main components:

- Hash function (h_θ)
  - linear projection,
  - kernels,
  - neural nets

Definition

Learning to hash task: learning a hash function, $y = h_\theta(\mathbf{x})$, such that the NNS in the V space is efficient and the result is an effective approximation of the true NNS result in the $\mathcal{X}$ space.

Main components:

- Hash function (h_θ)
  - linear projection,
  - kernels,
  - neural nets
- Similarity measure (ρ')
  - Hamming distance
  - Euclidean distance

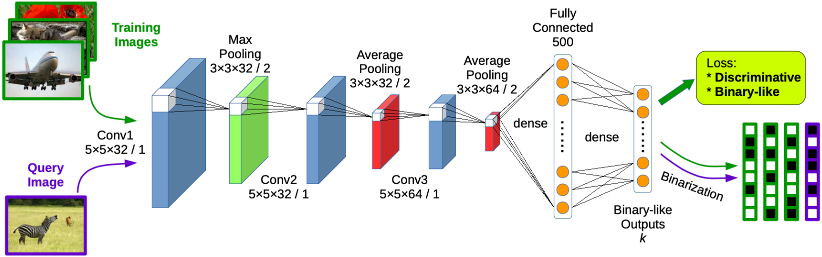
Definition

Learning to hash task: learning a hash function, $y = h_\theta(\mathbf{x})$, such that the NNS in the V space is efficient and the result is an effective approximation of the true NNS result in the $\mathcal{X}$ space.

Main components:

- Hash function (h_θ)
  - linear projection,
  - kernels,
  - neural nets
- Similarity measure (ρ')
  - Hamming distance
  - Euclidean distance
- Optimization criterion

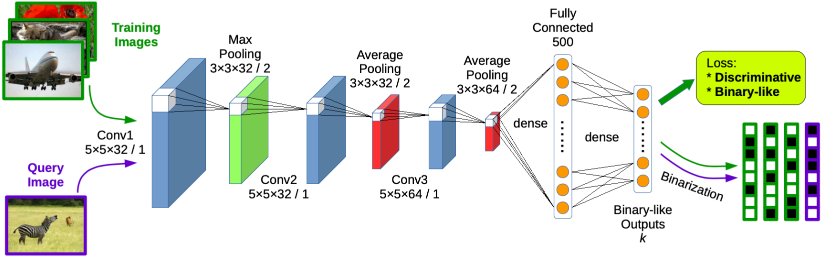
Hash function $h \colon \Omega \to \{+1, -1\}^k$, where Ω - RGB space:



Training data:

- Set of image pairs along with label: $\{(I_{i,1}, I_{i,2}, y_i)\}_{i=1}^{N}$, where $y_i = \mathbb{1}\{I_{i,1}$ not similar to $I_{i,2}\}$

Hash function $h \colon \Omega \to \{+1, -1\}^k$, where Ω - RGB space:



Training data:

- Set of image pairs along with label: $\{(I_{i,1}, I_{i,2}, y_i)\}_{i=1}^N$, where $y_i = \mathbb{1}\{I_{i,1}$ not similar to $I_{i,2}\}$

Similarity measure: $D_h(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^k \mathbb{1}\{\mathbf{x}_i \neq \mathbf{y}_i\}$ - Hamming distance

Main idea: codes of similar images should be as close as possible, while the codes of dissimilar images being far away

Main idea: codes of similar images should be as close as possible, while the codes of dissimilar images being far away

- Loss wrt the pair of images $I_1, I_2 \in \Omega$

$$L(\mathbf{b}_1, \mathbf{b}_2, y) = \frac{1}{2}(1-y)D_h(\mathbf{b}_1, \mathbf{b}_2) + \frac{1}{2}y \ \max\{m - D_h(\mathbf{b}_1, \mathbf{b}_2), 0\},$$

  where $\mathbf{b}_i = h(I_i) \in \{+1, -1\}^k$ - network output and $m > 0$ - margin threshold parameter.

Main idea: codes of similar images should be as close as possible, while the codes of dissimilar images being far away

- Loss wrt the pair of images $I_1, I_2 \in \Omega$

$$L(\mathbf{b}_1, \mathbf{b}_2, y) = \frac{1}{2}(1-y)D_h(\mathbf{b}_1, \mathbf{b}_2) + \frac{1}{2}y \ \max\{m - D_h(\mathbf{b}_1, \mathbf{b}_2), 0\},$$

  where $\mathbf{b}_i = h(I_i) \in \{+1, -1\}^k$ - network output and $m > 0$ - margin threshold parameter.
- Optimization criterion

$$\mathcal{L} = \sum_{i=1}^{N} L(\mathbf{b}_{i,1}, \mathbf{b}_{i,2}, y_i) \ \rightarrow \ \min$$

Main idea: codes of similar images should be as close as possible, while the codes of dissimilar images being far away

- Loss wrt the pair of images $I_1, I_2 \in \Omega$

$$L(\mathbf{b}_1, \mathbf{b}_2, y) = \frac{1}{2}(1-y)D_h(\mathbf{b}_1, \mathbf{b}_2) + \frac{1}{2}y\ \max\{m - D_h(\mathbf{b}_1, \mathbf{b}_2), 0\},$$

  where $\mathbf{b}_i = h(I_i) \in \{+1, -1\}^k$ - network output and $m > 0$ - margin threshold parameter.
- Optimization criterion

$$\mathcal{L} = \sum_{i=1}^{N} L(\mathbf{b}_{i,1}, \mathbf{b}_{i,2}, y_i) \ \rightarrow\ \min$$

it is infeasible to directly optimize $\mathcal{L}$

Relaxed loss function

$$
\begin{aligned}
L(\mathbf{b}_1, \mathbf{b}_2, y) = &\frac{1}{2}(1-y)\|\mathbf{b}_1 - \mathbf{b}_2\|_2^2 \\
&+ \frac{1}{2}y\ \max\{m - \|\mathbf{b}_1 - \mathbf{b}_2\|_2^2, 0\} \\
&+ \underbrace{\alpha(\||\mathbf{b}_1| - \mathbf{1}\|_1 + \||\mathbf{b}_2| - \mathbf{1}\|_1)}_{\text{regularizer to replace the binary constraints}}
\end{aligned}
$$

Table of Contents

Annoy[6] [2, 7]

Very simple idea. Go directly to `https://bit.ly/2YsuEb1` to see details

[6] `https://github.com/spotify/annoy`

FAISS[7] [4]

[7] `https://github.com/facebookresearch/faiss`

Table of Contents

Resume

1. There are many different solutions to ANN problem which can be categorized into
   - Neighbourhood-based
   - Hashing-based (LSH, Learning to Hash)
   - Space Partitioning-based
2. Commonly used benchmark[8] for ANN methods evaluation

[8] `https://github.com/erikbern/ann-benchmarks`

Recommendations

1. When there are sufficient computing resources HNSW is the best choice for ANNS on high dimensional data
2. Annoy is also a very good choice
3. If there are avaliable GPU then FAISS is a good choice

References I

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *2006 47th annual IEEE symposium on foundations of computer science (FOCS'06)*, pages 459–468. IEEE, 2006.

[2] E. Bernhardsson. Annoy. `https://bit.ly/2YsuEb1`, 2015.

[3] N. Bhatia et al. Survey of nearest neighbor techniques. *arXiv preprint arXiv:1007.0085*, 2010.

[4] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 2019.

[5] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM Journal on Computing*, 30(2):457–474, 2000.

[6] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of massive data sets*. Cambridge university press, 2020.

[7] W. Li, Y. Zhang, Y. Sun, W. Wang, M. Li, W. Zhang, and X. Lin. Approximate nearest neighbor search on high dimensional data-experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[8] H. Liu, R. Wang, S. Shan, and X. Chen. Deep supervised hashing for fast image retrieval. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2064–2072, 2016.

[9] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 45:61–68, 2014.

[10] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[11] J. Wang, H. T. Shen, J. Song, and J. Ji. Hashing for similarity search: A survey. *arXiv preprint arXiv:1408.2927*, 2014.

[12] J. Wang, T. Zhang, N. Sebe, H. T. Shen, et al. A survey on learning to hash. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):769–790, 2017.