NN - Search:

Given: set P of n points in $\mathbb{R}^d$ (query point q from set Q in $\mathbb{R}^d$)

Goal: find the nearest-neighbor p of q in P

Background:

Tree Based Methods: (small dimensionality)

Stochastic Methods: (approximate) → LSH

Filtering Based Methods: (exact)

Approximate Nearest Neighbors : input $\{P, q, r, \epsilon\}$

output: if some points exist with $dist(p,q) \leq r$, output any such ANN (YES)

if no points within $dist(p,q) \leq (1+\epsilon)r$, return NO

otherwise return any points within $dist(p,q) \leq (1+\epsilon)r$

Local Sensitive Hashing: A set of functions is locally-sensitive if for a random hash function h in the set, for any pair of points p, q

$Pr[h(p) = h(q)]$ is "high" if p close to q

$Pr[h(p) = h(q)]$ is "low" if p far from q

preprocessing: hash data using several LSH functions so pr of collision for closer objects ↑

Querying: hash query point and retrieve elements in buckets containing query point

give $ pull between # hashes $ # tables

LSH: Hamming Distance

$\{0,1\}^d$

$ham(p,q)$ = # positions in which they differ

define hash function by choosing set S of K random dimensions

and setting $h(p)$ = projection of p on S

$Pr[h(p) = h(q)] = (1 - ham(p,q)/d)^K$

ex: $d = 12$, $k = 3$, $p = \underline{1}10\,11\underline{0}0\,100\,10$ (positions 1, 5, 6)

$S = \{1,5,6\}$ → bucket $h(p) = 110$

L-p Norms: probability Distribution D is p-stable if

→ for any $v \in R$ and random variables $X_1, \dots X_n \sim D$, $\sum_i v_i X_i$ has same distribution as $[\sum_i v_i^p]^{1/p} X_i$ where $X \sim D$

Gaussian - normal distribution is 2-stable

Hash functions for L-p norm:

choose random v_i w/ entries randomly drawn from p-stable distribution

compute $h_i(x) = \left\lfloor \frac{\langle v_i, x \rangle + b}{w} \right\rfloor$ where w is bucket width & b is uniformly random in $[0, w]$

LSH: Jaccard coefficient (consider binary vectors p, q)

$$Jaccard(p,q) = \frac{|P \cap Q|}{|P \cup Q|}$$

-> disjoint sets have $J(A,B) = 0$ & $J(A,A) = 1$

LSH: Cosine Similarity

-> choose random unit vector r

-> $h_r(p) = 1$ if $\langle r, p \rangle > 0$, 0 otherwise

-> build sketches/signatures by concatenating hash functions

Index Join (straight-forward exact solution)

$$\cos(d_1, d_2) = \frac{\langle d_i, d_j \rangle}{\|d_i\|_2 \times \|d_j\|_2}$$

-> main idea: leverage data sparsity

-> $sim(d_2, d_3) = d_{2,1} \times d_{3,1} + d_{2,2} + d_{3,2} \dots d_{2,n} + d_{3,n}$ (n features)

method: normalize object vectors

construct inverted index from objects

for each obj:

-> compare only w/ objects containing features in common

-> select neighbors

Advantages:

-> skipping some comparisons

-> small memory footprint

Inverted Index:

f_1 f_2 f_3 ... features

documents: d_1, d_2, d_3 => features: f_1, f_2, f_3, ⋮

Inverted index (so you know which obj to compare to each other)

d_1 d_2 d_3 ... documents

Accumulator (A): data structure tracking similarities while being computed

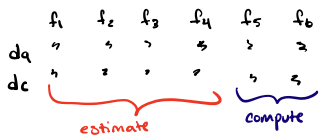
Sort the remaining results in Accumulator

select top k results and/or $sim(p,q) \geq \epsilon$

L2AP: (Fast Cosine Sim Search w/ Prefix L-2 Norm Bounds)

main idea: leverage similarity estimates

$$\langle d_q, d_c \rangle = \langle d_q^{\leq p}, d_c^{\leq p} \rangle + \langle d_q^{>p}, d_c^{>p} \rangle$$



filter objects not in final graph based on similarity estimates

L2AP: Algorithm

1. Accumulate Similarity using inverted index
2. For each un-pruned obj, finish similarity computation using forward index

L2AP leads to greatly improved runtime

-> filters most objects without computing their similarity

-> outperforms all exact & most approximate baselines

---

key concepts:

-> more likely to be neighbors if we have common high weight feature

-> you are my neighbor's neighbor

Step 1: choose initial candidates

-> sort vectors & inverted lists in non-increasing weight order

-> traverse two lists at a time & gather $M \geq k$ candidates with higher dot product

$C_{M=3} = [d_5, d_1, d_4]$

-> compute similarities w/ candidates & keep k NN

Step 2:

-> Find potential better results (γ times)

-> visit neighbors in non-increasing similarity order

-> consider d_s, a neighbor of d_r, as a candidate if

- have collected less than M candidates
- $sim(d_s, d_r) \geq sim(d_r, d_q)$

-> compute similarities w candidates & update both neighborhoods of d_s & d_q

---

L2Knng:

-> filtering Framework for k-NNG construction

-> build initial approximate graph $\hat{G}$

- provides threshold for filtering

-> improve $\hat{G}$ until exact

In practice:

higher # candidates = higher recall, but slower

Step 2 crucial to high recall

higher # step 2 iterations = higher recall