LSH

June 1, 2019

Table of Contents

1 Introduction

2 Review of LSH Search Scheme
- Entropy-based search
- Adaptative LSH
- LSH Forest
- Multi-Probe LSH
- Dynamic Collision Counting

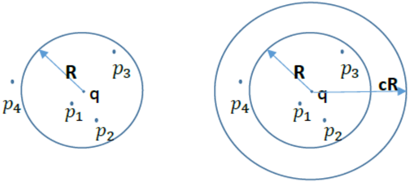
3 Implementation and Results Analysis

Locality Sensitive Hashing

LSH Families

A family $\mathcal{H}$ of functions from a domain S to a range U is called (r, ϵ, p_1, p_2)-sensitive, with $r, \epsilon > 0$, $p_1 > p_2 > 0$, if for any $p, q \in S$, the following conditions hold:

- if $D(p, q) \leq r$, then $Pr_{\mathcal{H}}[h(p) = h(q)] \geq p_1$
- if $D(p, 1) > r(1 + \epsilon)$ then $Pr_{\mathcal{H}}[h(p) = h(q)] \leq p_2$



Properties of Good LSH

- **Accuracy**: $\frac{\#\ of\ True\ Near\ Neightbors}{\#\ of\ Retrieved\ Candidates}$ should be as large as possible.
- **Efficient Queries**: $\#\ of\ Retrived\ Candidates$ should be as small as possible.
- **Efficient Maintenance**: A single scan to build tables.
- **Domain Independence**: Work well on any data domain.
- **Minimum Storage**: Storage consummption should be as little as possible.

Table of Contents

1 Introduction

2 Review of LSH Search Scheme
- Entropy-based search
- Adaptative LSH
- LSH Forest
- Multi-Probe LSH
- Dynamic Collision Counting

3 Implementation and Results Analysis

Entropy-based search

- Use one or a few hash table and hash several randomly chosen points in the neighborhood to reduce time and space complexity
- **Constraction of hash table**: pick $k = \frac{\log n}{\log(1/g)}$ random hash functions $h_1, h_2, ..., h_k$. For each point p in the database compute $H(p) = (h_1(p), h_2(p), ..., h_k(p))$ and store p in a table at location $H(p)$. *polylogn* is used to construct hash tables.
- **Search**: Given q and r, pick $O(n^\rho)$ random points v from $B(q, r)$, where $\rho = \frac{M}{\log(1/g)}$, and search in the buckets $H(v)$.

Adaptative LSH

- Based on the idea that the relevance of the hash function used in LSH are the lower the better
- D_8 lattice is the set of points of Z^8 whose sum is even, e.g. $(1, 1, 1, 1, 1, 1, 1, 1) \in D_8$
- $E_8 = D_8 \cup (D_8 + \frac{1}{2})$
- $h_i(x) = E_8(\frac{x_{i,8} - b_i}{w})$

Basic Idea of LSH Forest

- $B+$ tree is alwayse accurate
- Variance number of hash functions for different queries
- Efficient implementation for main memory and Disk

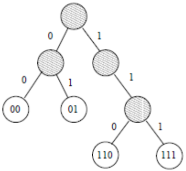


Figure 1: A prefix tree on the set of LSH labels

Algorithm for Query

ALGORITHM DESCEND(q, x_i, s)
    ▷ Args: query q at level x_i on node s
    if (s is leaf) {
        Return (s, x_i)
    } **else** {
        $y = x_i + 1$
        Evaluate $g_i(q, y)$
        t = Child node from branch labeled $h_y(q)$
        $(p, z) = \text{DESCEND}(q, y, t)$
        Return (p, z)
    }

ALGORITHM SYNCHASCEND$(x[1, \dots, l], s[1, \dots, l])$
    ▷ Args: x_i values and corresponding leaf nodes s_i for each T_i
    $x = max_i^l\{x[i]\}$
    $P = \phi$
    while ($x > 0$ and ($|P| < cl$ or $|distinct(P)| < m$)) {
        for ($i = 1$; $i \leq l$; $i{+}{+}$)
            if ($x[i] == x$) {
                $P = P \cup Descendants(s[i])$
                $s[i] = Parent(s[i])$
                $x[i] = x[i] - 1$
            }
        $x = x - 1$
    }
    Return P

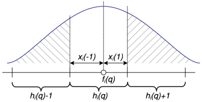
The Idea of Multi-Probe LSH [1]

Trade time for space:
Reduce the number of hash table while achieving similar performance by probing a sequence of buckets in one hash table.

Scheme	**Query**
Basic	$g(q) = (h_1(q), h_2(q), ..., h_M(q))$
Multi-Probe	$g(q)+\Delta^{(i)}, i=1, 2, ..., T, \ \Delta^{(i)}=(\delta_1^{(i)}, \delta_2^{(i)}, ..., \delta_M^{(i)})$

Probing Sequence

- Step-Wise:
  Firstly search all 1-step perturbations, then 2-step ones, and so on. The total number of all n-step buckets is $L \times \binom{M}{n} \times 2^n$.
- Query Based:
  $h(q) = \lfloor \frac{a \cdot q + b}{W} \rfloor$, $f(q) = a \cdot q + b$,
  $f(p) - f(q)$ follows Gaussian distribution.
  $P(h(p) = h(q) + \Delta) \approx C \exp(\sum_{i=1}^{T} x_i(\delta_i)^2)$



Algorithm:

- Sort all $x_i(-1), x_i(+1), i{=}1, 2..., M$
- Find T smallest valid subset

Optimized Probing Sequence Construction

The perturbations sequence generated every query time.
Actually a certain sequence can be precomputed.

The idea is use $\mathbb{E}[z_j^2]$ to replace z_j. z_j is the j-th value in the sorted $x_i(\delta)$ sequence. It can be proved that
$E[z_j]=\frac{j}{2(M+1)}W$, $E[z_j^2]=\frac{j(j+1)}{4(M+1)(M+2)}W^2$

Using Dynamic Compound Hash

Traditional LSH scheme use L static compound hash functions to reduce false positives. But it also reduce recall. Many hash tables will be used to improve recall.

The collision counting LSH [2] uses only a set of single hash function. Only data points with large enough collision numbers will be considered as cR-NN candidates.

In case of no data points returned, it uses virtual reranking to equivalently search a neighbor with radius $1, c, c^2, \ldots$.

Table of Contents

1. Introduction
2. Review of LSH Search Scheme
   - Entropy-based search
   - Adaptative LSH
   - LSH Forest
   - Multi-Probe LSH
   - Dynamic Collision Counting
3. Implementation and Results Analysis

Implementation

We choose several representative LSH schemes, implement them and compare their performance.

- Basic LSH
- LSH Forest
- Multi-ProbeLSH
- Bayesian LSH

Dataset. We use MNIST[1] to evaluate the algorithms. We choose 50 dimensions of the original 28×28 images with largest variances as [2] does. Ground truth are got by linear scan in the training set with Euclidean distance.

[1] http://yann.lecun.com/exdb/mnist/

Experiment Settings

Algorithm	Basic	LSHForest	MultiProbe	Bayesian
#compounds	8	25	8	
w	8	1	8	
T	-	-	16	-

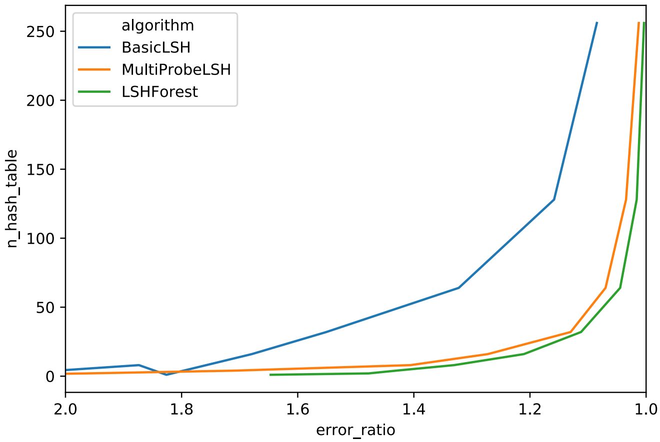
Error Ratio (1/2)

Error ratio indicates the accuracy of LSH's results and the ground truth. Error ratio is close to 1 means this LSH scheme successfully find the nearest neighbors.

$$\text{Error Ratio} = \frac{1}{N}\frac{1}{K}\sum_{n=1}^{N}\sum_{i=1}^{K}\frac{d(q, p_{lsh}^{(i)})}{d(q, p_{label}^{(i)})} \tag{1}$$

We use $K = 20$.

Error Ratio (2/2)

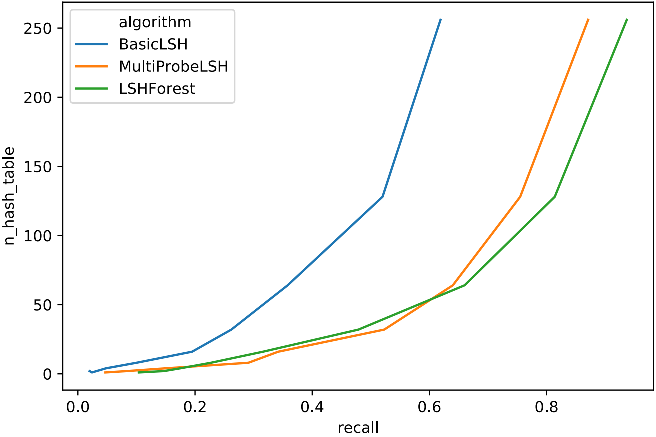


Recall (1/2)

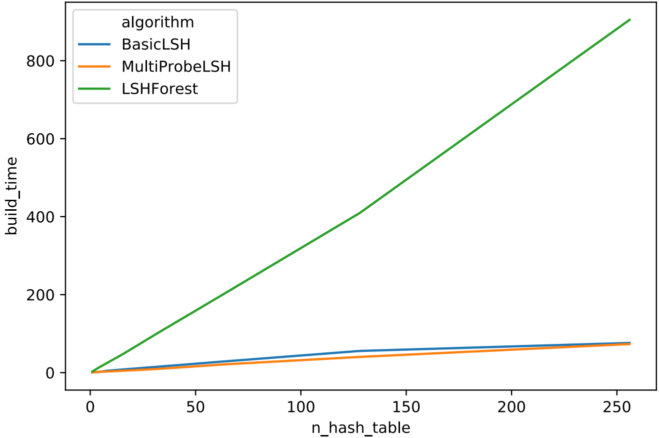
Recall shows how many nearest neighbors are found by LSH.

$$\text{Recall} = \frac{|A_{lsh} \cap A_{label}|}{|A_{label}|} \tag{2}$$

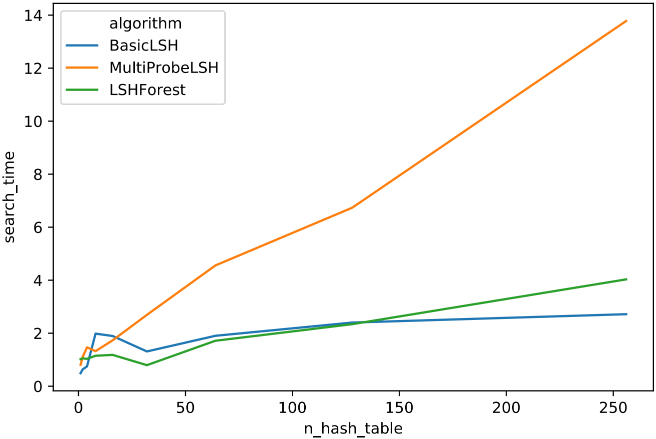
Recall (2/2)


algorithm
BasicLSH
MultiProbeLSH
LSHForest
n_hash_table
250
200
150
100
50
0
0.0
0.2
0.4
0.6
0.8
recall

Time Usage (1/2)

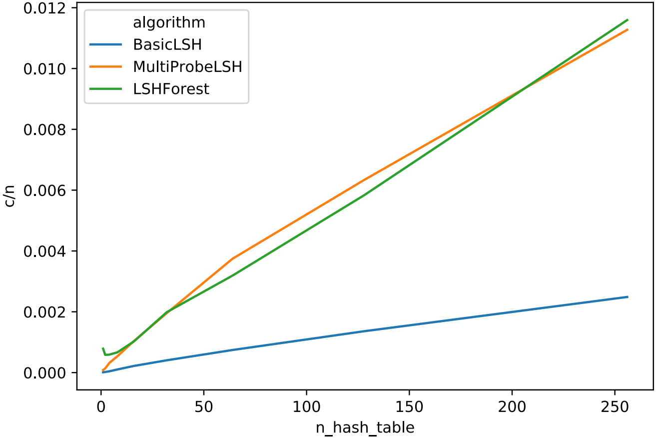


Time Usage (2/2)



Candidate Number

$$c/n = \frac{\#candidates}{\#train\ samples}$$



Observations

- LSHForest and MultiProbeLSH return more candidates with the same number of hash tables. Therefore they have better error ratio and recall than BasicLSH
- MultiProbeLSH takes most time in searching because it has to determine the probe sequence at every query.
- LSHForest suffers from long build time, especially when there are many hash tables.

Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe lsh: efficient indexing for high-dimensional similarity search," in *Proceedings of the 33rd international conference on Very large data bases*. VLDB Endowment, 2007, pp. 950–961.

J. Gan, J. Feng, Q. Fang, and W. Ng, "Locality-sensitive hashing scheme based on dynamic collision counting," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 2012, pp. 541–552.

Thank You