Locality-sensitive hashing

From Wikipedia, the free encyclopedia

Locality-sensitive hashing (**LSH**) reduces the dimensionality of high-dimensional data. LSH hashes input items so that similar items map to the same "buckets" with high probability (the number of buckets being much smaller than the universe of possible input items). LSH differs from conventional and cryptographic hash functions because it aims to maximize the probability of a "collision" for similar items.[1] Locality-sensitive hashing has much in common with data clustering and nearest neighbor search.

Contents [hide]

Definition [edit]

An *LSH family*[1][2][3] $\mathcal{F}$ is defined for a metric space $\mathcal{M} = (M, d)$, a threshold $R > 0$ and an approximation factor $c > 1$. This family $\mathcal{F}$ is a family of functions $h : \mathcal{M} \to S$ which map elements from the metric space to a bucket $s \in S$. The LSH family satisfies the following conditions for any two points $p, q \in \mathcal{M}$, using a function $h \in \mathcal{F}$ which is chosen uniformly at random:

- if $d(p,q) \le R$, then $h(p) = h(q)$ (i.e., p and q collide) with probability at least P_1,
- if $d(p,q) \ge cR$, then $h(p) = h(q)$ with probability at most P_2.

A family is interesting when $P_1 > P_2$. Such a family $\mathcal{F}$ is called (R, cR, P_1, P_2)-*sensitive*.

Alternatively[4] it is defined with respect to a universe of items U that have a similarity function $\phi : U \times U \to [0,1]$. An LSH scheme is a family of hash functions H coupled with a probability distribution D over the functions such that a function $h \in H$ chosen according to D satisfies the property that $Pr_{h \in H}[h(a) = h(b)] = \phi(a,b)$ for any $a, b \in U$.

Amplification [edit]

Given a (d_1, d_2, p_1, p_2)-sensitive family $\mathcal{F}$, we can construct new families $\mathcal{G}$ by either the AND-construction or OR-construction of $\mathcal{F}$.[1]

To create an AND-construction, we define a new family $\mathcal{G}$ of hash functions g, where each function g is constructed from k random functions $h_1, \ldots, h_k$ from $\mathcal{F}$. We then say that for a hash function $g \in \mathcal{G}$, $g(x) = g(y)$ if and only if all $h_i(x) = h_i(y)$ for $i = 1, 2, \ldots, k$. Since the members of $\mathcal{F}$ are independently chosen for any $g \in \mathcal{G}$, $\mathcal{G}$ is a (d_1, d_2, p_1^k, p_2^k)-sensitive family.

To create an OR-construction, we define a new family $\mathcal{G}$ of hash functions g, where each function g is constructed from k random functions $h_1, \ldots, h_k$ from $\mathcal{F}$. We then say that for a hash function $g \in \mathcal{G}$, $g(x) = g(y)$ if and only if $h_i(x) = h_i(y)$ for one or more values of i. Since the members of $\mathcal{F}$ are independently chosen for any $g \in \mathcal{G}$, $\mathcal{G}$ is a $(d_1, d_2, 1-(1-p_1)^k, 1-(1-p_2)^k)$-sensitive family.

Applications [edit]

LSH has been applied to several problem domains including[citation needed]

- Near-duplicate detection[5][6]
- Hierarchical clustering[7]
- Genome-wide association study[8]
- Image similarity identification
  - VisualRank
- Gene expression similarity identification[citation needed]
- Audio similarity identification
- Nearest neighbor search
- Audio fingerprint[9]
- Digital video fingerprinting

Methods [edit]

Bit sampling for Hamming distance [edit]

One of the easiest ways to construct an LSH family is by bit sampling.[3] This approach works for the Hamming distance over d-dimensional vectors $\{0,1\}^d$. Here, the family $\mathcal{F}$ of hash functions is simply the family of all the projections of points on one of the d coordinates, i.e., $\mathcal{F} = \{h : \{0,1\}^d \to \{0,1\} \mid h(x) = x_i \text{ for some } i \in \{1, ..., d\}\}$, where x_i is the ith coordinate of x. A random function h from $\mathcal{F}$ simply selects a random bit from the input point. This family has the following parameters: $P_1 = 1 - R/d$, $P_2 = 1 - cR/d$.

Min-wise independent permutations [edit]

Main article: MinHash

Suppose U is composed of subsets of some ground set of enumerable items S and the similarity function of interest is the Jaccard index J. If π is a permutation on the indices of S, for $A \subseteq S$ let $h(A) = \min_{a \in A}\{\pi(a)\}$. Each possible choice of π defines a single hash function h mapping input sets to elements of S.

Define the function family H to be the set of all such functions and let D be the uniform distribution. Given two sets $A, B \subseteq S$ the event that $h(A) = h(B)$ corresponds exactly to the event that the minimizer of π over $A \bigcup B$ lies inside $A \bigcap B$. As h was chosen uniformly at random, $Pr[h(A) = h(B)] = J(A, B)$ and (H, D) define an LSH scheme for the Jaccard index.

Because the symmetric group on n elements has size n!, choosing a truly random permutation from the full symmetric group is infeasible for even moderately sized n. Because of this fact, there has been significant work on finding a family of permutations that is "min-wise independent" - a permutation family for which each element of the domain has equal probability of being the minimum under a randomly chosen π. It has been established that a min-wise independent family of permutations is at least of size $lcm(1, 2, ..., n) \geq e^{n-o(n)}$.[10] and that this bound is tight.[11]

Because min-wise independent families are too big for practical applications, two variant notions of min-wise independence are introduced: restricted min-wise independent permutations families, and approximate min-wise independent families. Restricted min-wise independence is the min-wise independence property restricted to certain sets of cardinality at most k.[12] Approximate min-wise independence differs from the property by at most a fixed ϵ.[13]

Open source methods [edit]

Nilsimsa Hash [edit]

Main article: Nilsimsa Hash

Nilsimsa is an anti-spam focused locality-sensitive hashing algorithm.[14] The goal of Nilsimsa is to generate a hash digest of an email message such that the digests of two similar messages are similar to each other. The paper suggests that the Nilsimsa satisfies three requirements:

1. The digest identifying each message should not vary significantly for changes that can be produced

automatically.
2. The encoding must be robust against intentional attacks.
3. The encoding should support an extremely low risk of false positives.

TLSH [edit]

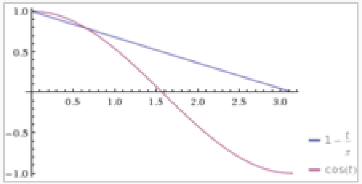
TLSH is locality-sensitive hashing algorithm designed for a range of security and digital forensic applications.[15] The goal of TLSH is to generate a hash digest of document such that if two digests have a low distance between them, then it is likely that the messages are similar to each other.

Testing performed in the paper demonstrates that on a range of file types identified the Nilsimsa hash as having a significantly higher false positive rate when compared to other similarity digest schemes such as TLSH, Ssdeep and Sdhash.

An implementations of TLSH is available as open-source software.[16]

Random projection [edit]

The random projection method of LSH[4] (termed arccos by Andoni and Indyk [17]) is designed to approximate the cosine distance between vectors. The basic idea of this technique is to choose a random hyperplane (defined by a normal unit vector r) at the outset and use the hyperplane to hash input vectors.



For small angles (not too close to orthogonal), $1-\frac{\theta}{\pi}$ is a pretty good approximation to $\cos(\theta)$.

Given an input vector v and a hyperplane defined by r, we let $h(v) = sgn(v \cdot r)$. That is, $h(v) = \pm 1$ depending on which side of the hyperplane v lies.

Each possible choice of r defines a single function. Let H be the set of all such functions and let D be the uniform distribution once again. It is not difficult to prove that, for two vectors u, v, $Pr[h(u) = h(v)] = 1 - \frac{\theta(u,v)}{\pi}$, where $\theta(u,v)$ is the angle between u and v. $1 - \frac{\theta(u,v)}{\pi}$ is closely related to $\cos(\theta(u,v))$.

In this instance hashing produces only a single bit. Two vectors' bits match with probability proportional to the cosine of the angle between them.

Stable distributions [edit]

The hash function [18] $h_{\mathbf{a},b}(\boldsymbol{\upsilon}) : \mathcal{R}^d \to \mathcal{N}$ maps a *d* dimensional vector $\boldsymbol{\upsilon}$ onto a set of integers. Each hash function in the family is indexed by a choice of random $\mathbf{a}$ and b where $\mathbf{a}$ is a *d* dimensional vector with entries chosen independently from a stable distribution and b is a real number chosen uniformly from the range [0,r]. For a fixed $\mathbf{a}, b$ the hash function $h_{\mathbf{a},b}$ is given by $h_{\mathbf{a},b}(\boldsymbol{\upsilon}) = \left\lfloor \frac{\mathbf{a} \cdot \boldsymbol{\upsilon} + b}{r} \right\rfloor$.

Other construction methods for hash functions have been proposed to better fit the data. [19] In particular k-means hash functions are better in practice than projection-based hash functions, but without any theoretical guarantee.

LSH algorithm for nearest neighbor search [edit]

One of the main applications of LSH is to provide a method for efficient approximate nearest neighbor search algorithms. Consider an LSH family $\mathcal{F}$. The algorithm has two main parameters: the width parameter k and the number of hash tables L.

In the first step, we define a new family $\mathcal{G}$ of hash functions g, where each function g is obtained by concatenating k functions $h_1, ..., h_k$ from $\mathcal{F}$, i.e., $g(p) = [h_1(p), ..., h_k(p)]$. In other words, a random hash function g is obtained by concatenating k randomly chosen hash functions from $\mathcal{F}$. The algorithm then constructs L hash tables, each corresponding to a different randomly chosen hash function g.

In the preprocessing step we hash all n points from the data set S into each of the L hash tables. Given that the resulting hash tables have only n non-zero entries, one can reduce the amount of memory used per each hash table to $O(n)$ using standard hash functions.

Given a query point q, the algorithm iterates over the L hash functions g. For each g considered, it retrieves the data points that are hashed into the same bucket as q. The process is stopped as soon as a point within distance cR from q is found.

Given the parameters k and L, the algorithm has the following performance guarantees:

- preprocessing time: $O(nLkt)$, where t is the time to evaluate a function $h \in \mathcal{F}$ on an input point p;
- space: $O(nL)$, plus the space for storing data points;
- query time: $O(L(kt + dnP_2^k))$;
- the algorithm succeeds in finding a point within distance cR from q (if there exists a point within distance R) with probability at least $1 - (1 - P_1^k)^L$;

For a fixed approximation ratio $c = 1 + \epsilon$ and probabilities P_1 and P_2, one can set $k = \frac{\log n}{\log 1/P_2}$ and $L = n^\rho$, where $\rho = \frac{\log P_1}{\log P_2}$. Then one obtains the following performance guarantees:

- preprocessing time: $O(n^{1+\rho}kt)$;
- space: $O(n^{1+\rho})$, plus the space for storing data points;
- query time: $O(n^\rho(kt + d))$;

See also [edit]

- Bloom Filter
- Curse of dimensionality
- Feature hashing
- Fourier-related transforms
- Multilinear subspace learning
- Principal component analysis
- Random indexing[20]
- Rolling hash
- Singular value decomposition
- Sparse distributed memory
- Wavelet compression

References [edit]

1. ^ a b c Rajaraman, A.; Ullman, J. (2010). "Mining of Massive Datasets, Ch. 3.".
2. ^ Gionis, A.; Indyk, P.; Motwani, R. (1999). , "Similarity Search in High Dimensions via Hashing". *Proceedings of the 25th Very Large Database (VLDB) Conference*.
3. ^ a b Indyk, Piotr.; Motwani, Rajeev. (1998). , "Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality.". *Proceedings of 30th Symposium on Theory of Computing*.
4. ^ a b Charikar, Moses S. (2002). "Similarity Estimation Techniques from Rounding Algorithms". *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*. pp. 380–388. doi:10.1145/509907.509965.
5. ^ Gurmeet Singh, Manku; Jain, Arvind; Das Sarma, Anish (2007), "Detecting near-duplicates for web crawling", *Proceedings of the 16th international conference on World Wide Web* (ACM).
6. ^ Das, Abhinandan S. et al. (2007), "Google news personalization: scalable online collaborative filtering", *Proceedings of the 16th international conference on World Wide Web* (ACM), doi:10.1145/1242572.1242610 .
7. ^ Koga, Hisashi, Tetsuo Ishibashi, and Toshinori Watanabe (2007), "Fast agglomerative hierarchical clustering algorithm using Locality-Sensitive Hashing", *Knowledge and Information Systems* **12** (1): 25–53.
8. ^ Brinza, Dumitru et al. (2010), "RAPID detection of gene–gene interactions in genome-wide association studies", *Bioinformatics* **26** (22): 2856–2862
9. ^ *dejavu - Audio fingerprinting and recognition in Python*
10. ^ Broder, A.Z.; Charikar, M.; Frieze, A.M.; Mitzenmacher, M. (1998). "Min-wise independent permutations". *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*. pp. 327–336. doi:10.1145/276698.276781. Retrieved 2007-11-14.
11. ^ Takei, Y.; Itoh, T.; Shinozaki, T. "An optimal construction of exactly min-wise independent permutations". *Technical Report COMP98-62, IEICE, 1998.*
12. ^ Matoušek, J.; Stojakovic, M. (2002). "On Restricted Min-Wise Independence of Permutations". *Preprint*. Retrieved 2007-11-14.
13. ^ Saks, M.; Srinivasan, A.; Zhou, S.; Zuckerman, D. (2000). "Low discrepancy sets yield approximate min-wise independent permutation families". *Information Processing Letters* **73** (1-2): 29–32. doi:10.1016/S0020-0190(99)00163-5. Retrieved 2007-11-14.
14. ^ Damiani et. al (2004). "An Open Digest-based Technique for Spam Detection" (PDF). Retrieved 2013-09-01.
15. ^ Oliver et. al (2013). "TLSH - A Locality Sensitive Hash". *4th Cybercrime and Trustworthy Computing*

Workshop. Retrieved 2015-04-06.
16. ^ "https://github.com/trendmicro/tlsh". Retrieved 2014-04-10.
17. ^ Alexandr Andoni; Indyk, P. (2008). "Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions". *Communications of the ACM* **51** (1): 117–122. doi:10.1145/1327452.1327494.
18. ^ Datar, M.; Immorlica, N.; Indyk, P.; Mirrokni, V.S. (2004). "Locality-Sensitive Hashing Scheme Based on p-Stable Distributions". *Proceedings of the Symposium on Computational Geometry*.
19. ^ Pauleve, L.; Jegou, H.; Amsaleg, L. (2010). "Locality sensitive hashing: A comparison of hash function types and querying mechanisms". *Pattern recognition Letters*.
20. ^ Gorman, James, and James R. Curran. "Scaling distributional similarity to large corpora." Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics. Association for Computational Linguistics, 2006.

Further reading [edit]

- Samet, H. (2006) *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann. ISBN 0-12-369446-9

External links [edit]

- Alex Andoni's LSH homepage
- LSHKIT: A C++ Locality Sensitive Hashing Library
- A Python Locality Sensitive Hashing library that optionally supports persistence via redis
- Caltech Large Scale Image Search Toolbox: a Matlab toolbox implementing several LSH hash functions, in addition to Kd-Trees, Hierarchical K-Means, and Inverted File search algorithms.
- Slash: A C++ LSH library, implementing Spherical LSH by Terasawa, K., Tanaka, Y
- LSHBOX: An Open Source C++ Toolbox of Locality-Sensitive Hashing for Large Scale Image Retrieval, Also Support Python and MATLAB.
- SRS: A C++ Implementation of An In-memory, Space-efficient Approximate Nearest Neighbor Query Processing Algorithm based on p-stable Random Projection

Categories: Search algorithms | Classification algorithms | Dimension reduction | Hashing | Probabilistic data structures