OLSH: Occurrence-based Locality Sensitive Hashing

Mohammadhossein Toutiaee
Computer Science Department
The University of Georgia
415 Boyd, 200 D.W. Brooks Drive, Athens, USA
hossein@uga.edu

***Abstract*—Probabilistic data structures are widely used with large amounts of data. Acceptable error or probability of failure can be controlled by statistic inference methods applied in many domains. Locality sensitive hashing (LSH) is an efficient data structure for a nearest neighbor search in high-dimensional data as an alternative to other exact Nearest Neighbor Searches such as R-tree. The basic idea is to provide probabilistic guarantees of solving approximate nearest neighbor searches in rich feature spaces. Using different buckets in a hash table has been proposed when running in a main memory structure (multi probe LSH); however, this method is not optimized for wide datasets such as streaming data. A proposed new approach takes advantage of throwing different bins and enhances LSH by reducing the failure rates of a similarity search.**

***Index Terms*—NN; LSH; MapReduce; Probability; Theory; ”Short Research Paper”**

I. Introduction

The nearest neighbor method can be used in classification and regression problems. Nearest neighbor algorithms search in various parts of a high-dimensional dataset by querying similar data points. The Nearest Neighbor algorithm is being applied in many application domains with simple to complex structure. Pattern recognition, image classification, text mining and information retrieval, marketing analysis and DNA sequencing are just a few examples. This algorithm is quite useful in many applications. Complexity is one of the weak points of nearest neighbor. Searching the exact nearest neighbor point in a space using brute force search would result in O(N) running time, which is inefficient to implement in a large space. Additionally, searching the K^{th} nearest neighbor would take O(NlogK) provided that a priority queue is used as the data structure. In both cases, O(N) is an inevitable part of the algorithm running time, which is not applicable for Large N. However, tree-based data structures enable more efficient pruning of the search space. R-tree[8], K-D-tree[2], SR-tree[11], X-tree[3] and M-tree[5] are data type indexing methods returning the exact query results in O(NlogN) time for constructing the tree. The time complexity of all tree-based structures is between O(logN) and O(N) depending on how well the tree was constructed previously. Although the logarithmic implementation time will be given under certain conditions regarding the distribution of points, the running time is still exponential in a dimension (d). Therefore, the complexity in tree-based methods is heavily reliant on how points are spread in a low-dimension space. In order to prevent exponential running time complexity for the Nearest Neighbor Search in a high-dimension space, Near-Neighbor Search methods are in fact idea for finding approximate nearest-neighbor pairs in certain ways without looking at all pairs. Locality sensitive hashing (LSH) is one of the data structures that has been recently used in exploring various research topics. This paper introduces a new enhanced LSH based on a distributed algorithm showing how efficient this approach is.

II. Related work

Indyk et al. [9] proposed locality sensitive hashing (LSH) based on the idea that a random hash function g exists on space R^{d} such that for any points p and q:

Assume $\varrho = \{g\ :\mathrm{R}^{\text{d}} \longrightarrow \mathrm{Z}^{\text{k}}\}$ is a set of hash functions such as:

$$g(v) = (h_1(v), ..., h_{\text{k}}(v))$$

where the functions h_{i} for i $\in [1, k]$ is a subset of LSH function set $H = \{h\ :\mathrm{R}^{\text{d}} \longrightarrow \mathrm{Z}\}$ called (r_1, cr, p_1, p_2) for any q, v such that:

$$Pr(h(q) = h(v)) \geq p_1, when\|q - v\| \leq r \tag{1}$$

$$Pr(h(q) = h(v)) < p_2, when\|q - v\| > cr \tag{2}$$

where $c > 1$, $p_1 > p_2$, r and cr are the decision and prune boundary, respectively. Intuitively, the pair q and v will more likely be hashed to the same value if their distance is within r, and less likely if their distance is greater than prune value cr.

Charikar [4] introduced ”Hyperplane LSH” heavily inspired by Goemans et al. [7], and the method only works for a spherical space. Geomans's method is based on splitting a sphere by a hyperplane randomly. Multi-probe LSH [13] was proposed to maximize the chance of collision between a pair of near data points by binning the space with random buckets. This method merely queries which bucket is more likely to contain relevant data points. A posteriori multi-probe LSH [10] as an extension to Multiprobe takes advantage of the likelihood of each bucket, turning to the probability of containing similar points incorporating a prior knowledge obtained from training data. Both Multiprobe and a posteriori LSH tend to favor a higher false negative for low storage overheads. Voronoi-based locality sensitive hashing [12] partitions the space into a Voronoi diagram by random hash functions; however, backing to Voronoi diagram partitioning space becomes computationally inefficient as the dataset increases. Although LSH Forest [1] attempts to enhance indexing technique in high-dimension data, this technique was only designed for Hamming distance [14], and to use other distance functions is still challenging.

Three approaches addressing drawbacks of the previous work will be presented in this paper. First, the false negative of searching similar points would be enhanced using multi LSH tables. Second, distance functions are not used in this technique. Last, but not least, a distributed algorithm will be introduced to boost the query time.

III. Problem Setup

The objective is to search near-neighbor data points in a given high-dimension space using a global voting function $V(.,.)$ to obtain a similarity between a query q and a point v. Accordingly, the m near-neighbor search will return m desired points that are the most similar objects (data-points) among all the data in a dataset using the global voting function V. Eventually, locality sensitive hashing as a real competitor to the tree-based structures could diminish the search space to a subset of data points binned by some random buckets. The random buckets are constructed by some random lines h in a coordinate space (or random hyperplanes h_d in a spherical space). A "score" for each point can translate each line h (or hyperplane h_d) into a binary index. And at the end, a hash table is created using an h-bit binary vector for each point as a bucket index.

IV. Motivation

First and foremost, the traditional LSH methods mentioned above only depend on some distance function. The shortcoming that exists in applying those methods would result in using very limited distance functions. Therefore, the ultimate purpose of using this new method is to remove any distance functions (including all metric distances) and to search for near neighbors in the space. The method is only based on a number of point occurrences in a set of multi LSH tables. In essence, this method involves finding the most repeated point as an approximate nearest neighbor (or similarly, finding the most repeated points as K near-neighbor) after having counted the global number of occurrences per each point.

LSH methods can be applied fast with an accepted error rate. However, this work shows the improvement in the accuracy of searching near-neighbor by reducing the probability of not finding a near-neighbor with a proof given below. The false negatives could be decreased by constructing more hash tables with random buckets. Multi-tables hash function tends to outperform one table LSH by taking advantage of occurring independent events. Additionally, the running time increases as a result of emitting more hash tables; this would be lessened by proposing a distributed multi-hash table approach using *MapReduce*.

V. OLSH

Occurrence-based LSH is a significantly fast approach for Near Neighbor Search in a high-dimensional dataset. The technique provides an accepted error rate using multiple tables for locality sensitive hashing. In the LSH technique, as the number of buckets increases, the probability of not finding the near neighbor decreases. Accordingly, a smaller error rate would be obtained as a result of using more hashing tables:
Proof: Assume we search bins 1 bit off from query.

$$LSH^{h=1} = \begin{cases} = 1 \text{ - Pr(same bin) - Pr(1 bin off)} \\ = 1 \text{ - Pr(no split) - Pr(1 split)} \\ = 1\text{-}(1\text{-}\delta)^h - h\delta(1-\delta)^{h-1} \\ \delta : \text{Probability of split} \end{cases} \tag{3}$$

$$LSH^{h>1} = \begin{cases} = 1 \text{ - Pr(same bin)}^{h+1} \\ = 1 \text{ - Pr(no split)}^{h+1} \\ = 1\text{-}(1\text{-}\delta)^{h+1} \\ \delta : \text{Probability of split} \end{cases} \tag{4}$$

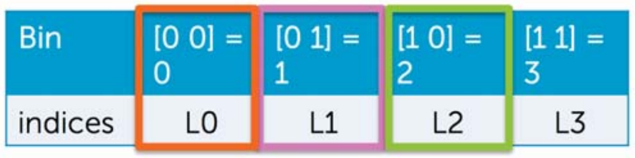
VI. Evaluation

In (Figure 1c), the comparison between multiple hash tables (blue) and one hash table (red) indicates that the probability of not finding the near-neighbor indicated by the red line is lower than the blue line (when $h = 3$). Similarly, in (Figure 1d), the gap between the blue line and the red line becomes more meaningful; therefore, we have a lower chance of not finding the near-neighbor (a higher probability of finding the near-neighbor) when $h = 10$. The higher chance of finding the near-neighbor arises from the probability of falling off of similar points in different bins in an exponential fashion.

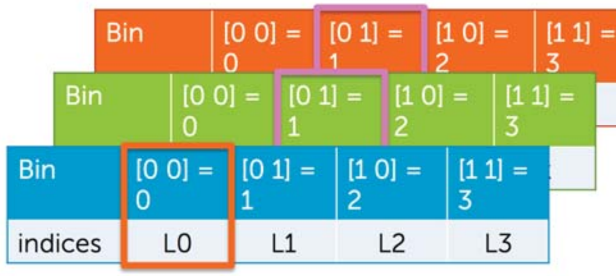
Algorithm 1 MR-MultiLSH-Tables

INPUT: Set of data points
OUTPUT: Top near-neighbor points

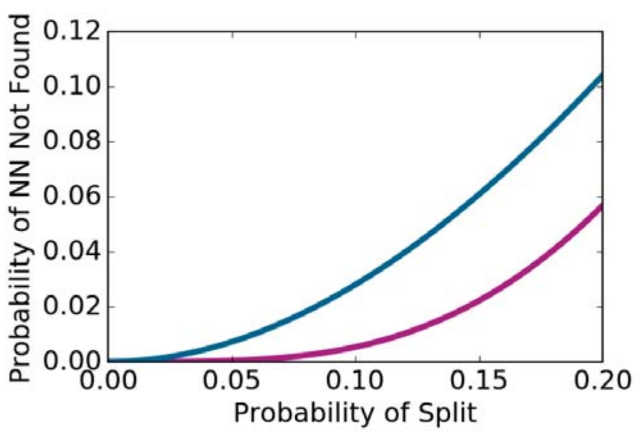
```
1:  Class RandomBins
2:  method RndBin({$\vec{x_1}, ..., \vec{x_n}$}, $h, b$)
3:          for $h = 1$ To $H$
4:  Set $\vec{b_i}, ..., \vec{b_j}$ to be distinct randomly binned inputs from
    $\vec{x_1}, ..., \vec{x_n}$
5:          end for
6:
7:  Class Mapper
8:          method $Map(data(key), value)$
9:                  write $(data(key), value)$
10:
11: Class Reducer
12:         method $ReduceCount(data(key), value)$
13:                 write $None, (sum(values), data(key))$
14:         method $ReduceMax(data(key), value)$
15:                 write $max(values)$
16:
```



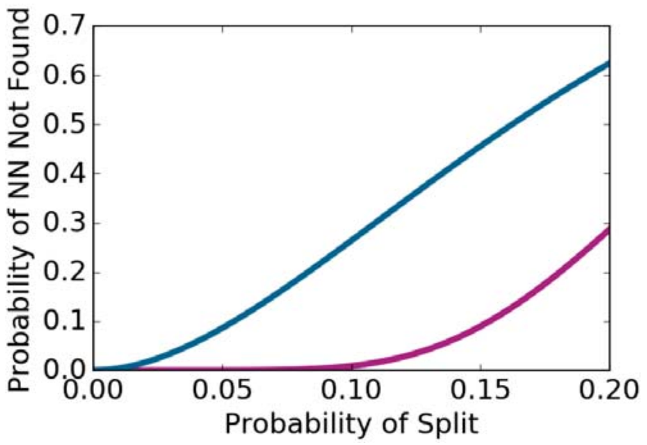
(a) $h = 1$ hash table



(b) $h = 3$ hash tables



(c) $Pr(h = 3)$



(d) $Pr(h = 10)$

Fig. 1: Multi LSH tables would harness the near-neighbor search by reducing false negatives in the result. MapReduce can make the search fast among the space. (a), Binning a hash table with random bins. (b), Multi LSH tables for $h = 3$ tables. (c), Probability of not finding the near-neighbor for one hash table (blue) and for multiple hash tables (red) when the $h = 3$. Obviously the red curve is lower than the blue curve, indicating the lower error. (d), This plot indicates that the higher the number of tables, the lower the error rate. (e) and (d) show the MapReduce schema and the algorithm, respectively [6].

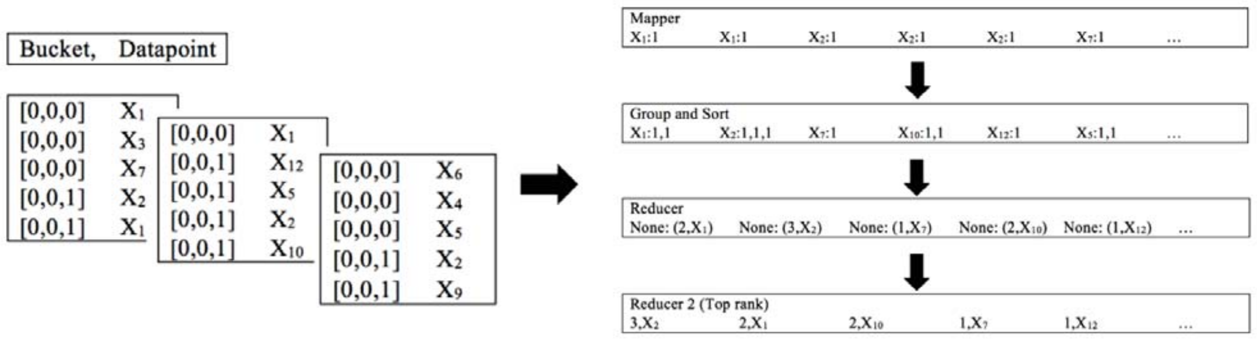


Fig. 2: : This diagram is showing the MapReduce schema and the algorithm.

VII. Conclusion

The expectation of finding Near Neighbors in a high-dimension data would increase exponentially fast using multiple hash tables. OLSH, as a new technique, takes that advantage and reduces its time complexity using the MapReduce approach. Moreover, the limitation of using some distance functions in old techniques is eliminated because in OLSH the near points are only detected in an ON/OFF mode.

References

[1] M. Bawa, T. Condie, and P. Ganesan. Lsh forest: Self-tuning indexes for similarity search. In *Fourteenth International World Wide Web Conference (WWW 2005)*, 2005.

[2] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, Sept. 1975.

[3] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The x-tree: An index structure for high-dimensional data. In *Proceedings of the 22th International Conference on Very Large Data Bases*, VLDB '96, pages 28–39, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[4] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 380–388, New York, NY, USA, 2002. ACM.

[5] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, VLDB '97, pages 426–435, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[6] E. Fox and C. Guestrin. Clustering and retreival. In *Machine Learning Specialization, Coursera*, 2016.

[7] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, Nov. 1995.

[8] A. Guttman. R-trees: A dynamic index structure for spatial searching. *SIGMOD Rec.*, 14(2):47–57, June 1984.

[9] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 604–613, New York, NY, USA, 1998. ACM.

[10] A. Joly and O. Buisson. A posteriori multi-probe locality sensitive hashing. In *Proceedings of the 16th ACM International Conference on Multimedia*, MM '08, pages 209–218, New York, NY, USA, 2008. ACM.

[11] N. Katayama and S. Satoh. The sr-tree: An index structure for high-dimensional nearest neighbor queries. *SIGMOD Rec.*, 26(2):369–380, June 1997.

[12] T. L. Loi, J. P. Heo, J. Lee, and S. e. Yoon. Vlsh: Voronoi-based locality sensitive hashing. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5345–5352, Nov 2013.

[13] 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 '07, pages 950–961. VLDB Endowment, 2007.

[14] M. Norouzi, D. J. Fleet, and R. R. Salakhutdinov. Hamming distance metric learning. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1061–1069. Curran Associates, Inc., 2012.