International Journal of Fuzzy Logic and Intelligent Systems, vol. 12, no. 4, December 2012, pp. 300-307
http://dx.doi.org/10.5391/IJFIS.2012.12.4.300 pISSN 1598-2645 eISSN 2093-744X

Locality-Sensitive Hashing Techniques for Nearest Neighbor Search

Keon Myung Lee

Dept of Computer Science and PT-ERC
Chungbuk National University
Cheongju, Chungbuk 361–763, Korea

Abstract

When the volume of data grows big, some simple tasks could become a significant concern. Nearest neighbor search is such a task which finds from a data set the k nearest data points to queries. Locality-sensitive hashing techniques have been developed for approximate but fast nearest neighbor search. This paper introduces the notion of locality-sensitive hashing and surveys the locality-sensitive hashing techniques. It categories them based on several criteria, presents their characteristics, and compares their performance.

Key Words: locality-sensitive hashing, hashing, nearest neighbor search, similarity search

1. Introduction

Many sectors in business, government, science, and engineering have been collecting data and archiving them as they realize their data as valuable assets. As the volume of data grows big, the simple operations that work well for moderate-sized data set can cause a significant burden. One of such operations is the nearest neighbor search task which searches for k nearest data points with respect to given data point. It just requires to compare the query with all data points one by one. Its time complexity is $O(n)$ where n is the number of data points in the database. A similar but more expensive task is the similar pair identification to find the pairs of close data points from the database, in which all pairwise computations of data points are required to get the close ones. When straightforward techniques is applied to the task, its time complexity becomes $O(n^2)$. It is not rare to have millions or billions number of data records in real applications.

The nearest neighbor search and similar pair identification problems occur in various application domains[20, 41]. Torralba et al.[53] built an image database with about 80 million images, which were collected by querying Web pages with WordNet[35] words and were labeled with query words comparable to the categories of the image contents. Their system enables an image classification service which finds similar images for a query image and labels the query image with reference to the labels for the retrieved similar images. Duplicate search task is to find near duplicate pages from a collection of documents[32, 42]. The task can be applied in detecting duplicate web pages[14, 51], matched newspaper articles syndicated by the same source, and plagiarized documents[48]. To support those tasks, similar pair identification operations need to be carried out in an efficient way. Hays and Efros[12] developed an interesting scene completion service in which when an image is partially occluded by some unpleasant objects, the service makes the occluded objects taken out from the image. To realize this task, the service system retrieves similar images from its image database, then cuts out the occluded parts from the image and blends the corresponding parts of retrieved similar images into the cut-out parts. Baluja and Covell[2] proposed a technique which allows from a snippet of song to retrieve the complete score of the song. To handle this task, a music database is assumed to be constructed which archives acoustic features of songs and their complete score. Their technique enables to retrieve partial matches of acoustic signals from large volume of song database. McFeel and Lanckret[34] also proposed a similarity search technique for music retrieval.

With ever increasing volume of data, much research effort has been exerted to efficiently conduct nearest neighbor search for the last decade. When exactness of search results is not crucial, some approximate approaches can be adopted which work faster by orders of magnitude with some tolerance to inexactness. Locality-sensitive hashing (LSH) techniques are such an approximate approach to nearest neighbor search.

Manuscript received Dec. 1, 2012; revised Dec. 24, 2012; accepted Dec. 24, 2012.
Corresponding author: Keon Myung Lee (kmlee@cbnu.ac.kr)
This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (grant No.2012-0000478).

The objective of this paper is to represent the notion of locality-sensitive hashing, survey the recent LSH-based techniques and conduct meta-analysis on their performance. The paper is organized as follows: Section 2 presents the conventional techniques to give exact results. Section 3 describes the notion of locality-sensitive hashing and how to categorize them. Section 4 surveys the LSH techniques and shows a meta-analysis results for their performance. In final, Section 5 draws the conclusions.

2. The Conventional Approaches to Nearest Neighbor Search

In the literature there have been various indexing data structure developed for efficiently locating data. The Voronoi diagram is a typical data structure for nearest neighbor search which decomposes a metric space, in which boundaries are determined at the equi-distances from adjacent points. The Vorionoi diagram-based methods work well in low-dimensional space, but their performance tends to deteriorate to brute-force search in high-dimensional space.

There are some tree-based indexing structures to support nearest neighbor search: kd-tree, R-tree, hierarchical k-means, ball tree, spatial tree, and spill tree. The kd-tree method[3] constructs for a collection of data a binary tree for which every node is a k-dimensional point and the splitting boundaries are axis-parallel. The R-tree method[11] organizes a tree structure by grouping nearby data points and representing them with their minimum bounding rectangles in the next higher level of the tree. R-trees are generally used in 2 or 3 dimensional data like geographical information system. The hierarchical k-means tree method[37] builds a search tree by first applying the k-means algorithm to the whole data set and branching child nodes for partitioned groups, then recursively applying the same procedure to each group of the previous level until a maximum level reaches. Ball tree[40] is a binary tree where each node, called a ball, represents a subspace of the d-dimensional Euclidean space bound by a hyper-sphere, where data live in the d-dimensional space. The radius of a ball node is as large as to contain the child node balls. Spatial tree[54] is a generalization of kd-trees. While a kd-tree partitions the space in an axis-parallel way, a spatial tree recursively partitions a data set by projecting onto an arbitrary direction, splitting at the median of projected positions, and forming two disjoint subgroups. Depending on how to choose directions, they are categorized into maximum variance kd-trees, PCA trees, 2-means trees, and random projection trees. Spill tree[29] is a variant of spatial trees in which the children of a node may contain shared data points in which data are partitioned like spatial trees but subsets are allowed to overlap.

These tree-based indexing methods are effective for low-dimensional data and can provide exact results for queries. Most tree-based methods are known to be poor for high-dimensional, large volume of data. As an alternative to tree-based indexing, hash techniques have been employed to locate data. For huge collection of high dimensional data, hash-based approximate methods have been actively studied for the last decade.[1, 9, 16]

3. The Locality-Sensitive Hashing Approaches

Hashing is a technique to enable to directly locate data points using a special function, called hash function. A hash function is a sort of algorithm to map a large data set, called keys, to a smaller index set of a fixed length. Each index of the index set has its own bucket to which corresponding keys are mapped. When more than one data points fall into a bucket together, they are said to collide. Usually because the size of the index set is much smaller than that of data set, buckets come to have multiple colliding data. In the conventional hashing, colliding data points do not imply any similarity between them.

3.1 Locality-Sensitive Hashing

For efficient near neighbor search, the locality-sensitive hashing(LSH) techniques exploit special hash functions which make buckets contain similar keys (data), yet do not guarantee that all data in a bucket are similar each other. The so-called locality-sensitive hash functions provide high probability for similar data to be in the same bucket, but low probability for dissimilar data to be in the same bucket.

The notion of locality-sensitive hashing was first introduced by Indyk et al.[16], in which the locality-sensitive hash functions are defined as follows: A family of functions $\mathcal{H} = \{h : S \rightarrow U\}$ is an LSH family when for any two points $p, q \in S$, any function h from $\mathcal{H}$, the following conditions hold:

- if $d(p, q) \leq r_1$, then $Pr_{\mathcal{H}}(h(p) = h(q)) \geq P_1$.
- if $d(p, q) \geq r_2$, then $Pr_{\mathcal{H}}(h(p) = h(q)) \leq P_2$.

Here $d(p, q)$ denotes the distance between p and q, $Pr_{\mathcal{H}}()$ indicates the probability, r_1 and r_2 are constants for distances ($r_1 < r_2$), and P_1 and P_2 are constants for probabilities ($P_1 > P_2$). A family $\mathcal{H}$ of functions satisfying the above conditions is called (r_1, r_2, P_1, P_2)-sensitive.

The hash functions by Indyk et al.[16] are defined as follows: First, data points are encoded into binary codes in the Hamming space. To define a hash function $h()$, a fixed number of positions are randomly sampled with replacement from the set of positions, and the function value for a data point is constructed by concatenating the binary values at the selected positions. When the number of selected positions is d, the bucket size for a hash function becomes 2^d. Multiple hash functions $\mathcal{H}$ are constructed in the same manner. Each data point in the database is hashed

by hash functions into buckets. When a query d_q is given, it is hashed by each hash function h into the corresponding bucket. The data points of the buckets to which hash functions map the query become the candidates with which the query is compared to determine whether they are neighbors.

3.2 Categorization of Locality-Sensitive Hashing Approaches

There have been proposed many locality-sensitive hashing techniques for nearest neighbor search. They can be categorized in the various criteria such as types of data under consideration, the number of installed bucket sets, the shape of hash functions, how to use data, what information is available for data, and how to quantize.

3.2.1 The Types of Data Attributes

Data can be categorized into numeric data, set-valued data data, categorical data, and mixed data. Numeric data consist of all continuous or ordinal attributes, and hence the distance between data can be easy defined, e.g., Euclidean distance. Set-valued data allow to have multiple values at a time. Such a typical example is textual data. Categorical data have categorical values like student, book, car. The distances for categorical values are not so easily defined. Many business data happen in mixed data form which contains multiple attributes of which types can be either numeric, set-valued, or categorical. Most LSH methods have been developed for numeric data and they assume that data are somehow embedded into the Euclidean space. For set-valued data like textual data, minhash-based method[5, 6] can be used for quickly estimating how similar two sets are, e.g., Jaccard distance. In case of textual documents, it first transforms them into bags of words, generates permutations used to select words from the bags, constructs a signature for each document with the selected words, and uses as hash codes some bands of the constructed signatures. It is proved that the minhash-based method approximates Jaccord distance. Lee et al.[26] proposed an LSH method for data sets with only categorical attributes, even though it is hard to define distance between categorical values. The method first partitions the categorical values into disjoint clusters using similarity information obtained by data-driven similarity measures[4]. Then it defines for each attribute a mapping component which maps attribute values into their cluster ID, and constructs a hash function by combining the mapping components.

3.2.2 The Number of Bucket Sets

In locality-sensitive hashing, multiple hash functions are used at a time. In some methods[5, 16, 26, 33, 58], each hash function has its own bucket set. When there are n hash functions and each hash function has m buckets. Then there are nm buckets in total. A data point is mapped by each hash function to a bucket, and hence it appears n times in the bucket stes, i.e., once at each bucket set. When a query is given, the candidates to be compared are the union of data points from the buckets to which the query is mapped by the hash functions. As we increase the number of hash functions and the number of buckets for each hash function, the probability to find the nearest neighbors increases, but the storage overhead also increases.

In the other methods, the hash functions all together participate in generating a single hash code for a data point. Therefore, there is only one bucket set in which a data point falls into a bucket. When a query is given, its hash code determines which bucket to be searched for nearest neighbors. Usually the neighborhood relationship among buckets are available in the methods. Hence, neighboring buckets are further examined for nearest neighbors. The hash codes are here binary codes, the buckets are labeled by the binary hash codes, and Hamming distances between buckets reflect the similarity of data from the buckets. We call these methods the binary code LSH techniques.

3.2.3 The Shape of Hash Functions

Some hash functions are expressed in explicit form on the data space, whereas some methods like Restricted Boltzmann Machine-based method[45] is not because its behaviors are probabilistic. Some hash functions[9, 28, 56] are expressed by hyperplanes on the data space. To get nonlinear decision boundaries, some methods[13, 22, 30] define hyperplanes on the feature space by using kernel trick which allows the dot product operations of feature space data in the data space. There are methods[43, 57] to use analytic periodic functions like sine function. There is a method[15] to use hyperspheres instead of hyperplanes.

3.2.4 How to Use Data

Depending on whether the data are used in constructing hashing functions, the techniques are divided into data-independent methods and data-dependent methods. The data-independent methods[9, 43] do not take care of data distribution. In the data-dependent methods, the data distribution affects the formation of hash functions, where some objective functions are defined to characterize the quality of hash codes for given data set and their parameters are determined by optimization techniques. Hence these methods are sometimes called machine learning-based methods because hash functions are learned from data.

3.2.5 What Information is Available for Data

When hash functions are learned from data, i.e., data-dependent methods, the LSH methods can be grouped into three categories: unsupervised, supervised, and semi-supervised methods. In unsupervised methods[22, 29, 43,

45, 57], it is assumed that no labels are available for data concerning which pairs are similar or dissimilar. The supervised methods[2, 36, 47, 49, 55, 58] use only labeled data in learning hash functions. The semi-supervised methods[17, 56] use both labeled data and unlabeled data.

3.2.6 How to Quantize

In binary code LSH techniques, hash codes are expressed in binary code. Depending on how many bits a hash function produces, the techniques can be categorized into single-bit quantization (SQ) and multi-bits quantization (MQ) methods. Most methods take the SQ approach in which a hash function produces a single bit that indicates on which side of the corresponding projection plane a data point lies. A few techniques[18, 19, 29] allow a hash function to produce multiple bits. The MQ methods include hierarchical quantization and Manhattan quantization[19].

4. The Locality-Sensitive Hashing Techniques

Most Locality-Sensitive Hashing Techniques belong to the binary code LSH techniques, in which each hash function produces bit value(s) and the hash codes are constructed by concatenating the bit outputs of all hash functions. The following shows some LSH techniques, not exhaustive. This section briefly presents their characteristics.

- LSH Hashing for Binary Codes (LSH) [1, 9]
- Boost Similarity Sensitive Hashing (BoostSSC) [47]
- Restricted Boltzmann Machine Hashing (RBM) [45]
- Spectral Hashing (SH) [57]
- Forgiving Hashing (FH) [2]
- Kernelized Locality-Sensitive Hashing (KLSH) [22]
- Shift-Invariant Kernel Hashing (SIKH) [43]
- Binary Reconstructive Embedding Hashing (BRE) [23]
- Optimized Kernel Hashing (OKH) [13]
- Self-Taught Hashing (STH) [59]
- Semi-Supervised Hashing (SSH) [56]
- Unsupervised Sequential Projection Learning for Hashing (USPLH) [55]
- Label-regularized max-margin partition (LAMP) [36]
- LHS for Learned Metrics(LHS-LM) [24]
- Laplacian co-Hashing (LCH) [60]
- Anchor Graph Hashing (AGH) [29]
- Minimum Loss Hashing (MLH) [38]
- Semi-Supervised SimHash (S^3H) [17]
- Kernel-based Supervised Hashing (KSH) [30]
- PCA-based Hashing (PCAH) [56]
- LDA Hashing (LDAHash) [49]
- Iterative Quantization (ITQ)[10]
- Isotropic Hashing (IsoHash) [18]
- Manhattan Hashing (MH) [19]
- Spherical Hashing (SHD) [15]
- Density Sensitive Hashing (DSH) [28]
- Multi-Valued Hashing (MVH) [50]
- Complementary Hashing (CH) [58]

4.1 The Characteristics of the Binary Code LSH Techniques

LSH Hashing for Binary Codes[1, 7] uses random projection vectors to define hyperplanes. Each hyperplane plays the role of a hash function which produces a single bit, 1 for one side of the hyperplane and 0 for the other side.

Boost Similarity Sensitive Hashing method[47] is a supervised method which first samples a subset of data and determines similar and dissimilar pairs from them, then treats similar pairs as positive examples, dissimilar pairs as negative examples, after that learns weak classifiers to separate positives from negatives with AdaBoost[46], a machine learning algorithm.

Restricted Boltzmann Machine Hashing (RBM) is a Markov random field which has connections only between visible layer and hidden layer, and their nodes are stochastic binary units. RBM can be learned so as to maximize the probability to reconstruct original input at visible layer from activation of hidden layer using the contrastive divergence sampling-based update rule. The RBM-based method[45] uses a stacked RBM with multiple layers of which upper layers gradually have smaller number of nodes. The output of the topmost layer becomes the binary hash codes for the data fed into the bottommost layer as the input.

Spectral Hashing[57] is related to the spectral decomposition of graphs, uses the property that the uniform distribution with Gaussian distance has an analytic form of eigenfunctions for Laplace-Betrami operator approximating eigenvectors of Graph Laplacian[31]. It first finds the principled axes of the data set and adjusts the data along the axes. Then it chooses the k smallest single dimension eigenfunctions with which it produces binary codes for data by thresholding the eigenfunction values for data at zero level.

Forgiving Hashing[2] learns multiple strong classifiers using AdaBoost and builds multiple hash tables by combining some of the strong classifiers. When a query is given, the data points in the hashed buckets, one bucket from each hash table, become the candidates to be compared for similarity.

Kernelized Locality-Sensitive Hashing[22] defines hash functions in a feature space using the kernel trick. It randomly selects a small number of samples from the data set and constructs a hash function $h(x)$ with linear combination of kernel value of x with the selected points.

Shift-Invariant Kernel Hashing[43] is a distribution free encoding method based on random projections, such that

the expected Hamming distance between the binary codes of two data points is related to the value of a shift-invariant kernel, like Gaussian kernel or Laplacian kernel, between the data points. It uses the random Fourier features and thresholds the values to get binary codes for data.

Binary Reconstructive Embedding Hashing[23] defines hash functions as a linear combination of kernel functions over a selected sample set. The weights of the hash functions are determined so as to minimize the difference between the metric distances, in the original data space, and the reconstructed distances, in binary codes, for the sample set. They are updated iteratively using a coordinate descent method, an optimization technique for parameters.

Optimized Kernel Hashing[13] finds kernel hash functions by explicitly representing and learning them with spectral decomposition[31]. This method allows the similarities between data points to be flexibly defined, and hence there are no constraints such as symmetry and positiveness, on the similarities.

Self-Taught Hashing[59] is two staged hashing technique. It first maps data points into binary codes using graph Laplacian spectral decomposition by constructing the k-nearest neighbor graph, computing eigenvectors for graph Laplacian, and thresholding the eigenvectors at a level and then taking the binary codes as hash codes. The next stage is to train an SVM classifier for each bit of the hash code using the data labeled with hash codes to handle the out-of-sample extension. The SVMs are used to assign hash codes for new data points.

Semi-Supervised Hashing[55, 56] uses both labeled data pairs and unlabeled data to determine projection vectors in a way to minimize the empirical error on the labeled data while maximizing the entropy of generated hash bits over the unlabeled data. It provides an efficient algorithm to solve optimization problem by eigendecomposing a KxK matrix, where K is the number of hash functions.

Unsupervised Sequential Projection Learning for Hashing[55] learns hash functions sequentially as it generates pseudo-labels at each iteration. The subsequent hash functions are determined so that they correct the errors made by previous hash functions. The pseudo-labels are assigned to the pairs of close data points, residing on the opposite sides of the hyperplane, and those of far data points on the same side of the hyperplane. Once pseudo-labels are given, the method works in the same way with the semi-supervised hashing[56].

Label-regularized Max-Margin Partition[36] generates hash functions in supervised setting, where a small portion of sample pairs are manually labeled to be either similar or dissimilar. As the hash functions, it searches max margin hyperplanes in which it selects randomly some data as support vectors for the objective function.

LHS for Learned Metrics[24] first finds a learned metric which makes similarity and dissimilarity constraints hold, and generates randomized hash functions using the learned metric.

Laplacian co-Hashing[60] is used for nearest neighbor search of textual documents. It hashes both documents and terms simultaneously according to their semantic similarities, and directly optimizes the Hamming distance. It regards the document collection as an undirected bipartite graph.

Anchor Graph Hashing[29] uses a small set, called anchor set, of the data set to approximate the adjacency matrix of the whole data set. It efficiently performs spectral clustering to get hash codes because it enables to compute the graph Laplacian eigenvectors of the original data set from the graph Laplacian eigenvectors of the anchar graph for the anchor set. The hash functions compute hash codes for data based on Nyström theorem.

Minimum Loss Hashing[38] is based on structured prediction with latent variables and a hinge-like loss function. It is efficient to train for large datasets, scales well to large code lengths, learns weights for the hyperplanes that minimize empirical loss over training pairs.

Semi-Supervised SimHash[17] learns the optimal feature weights from prior knowledge to relocate the data set such that similar data have similar hash codes. It uses both labeled data and unlabeled data.

Kernel-based Supervised Hashing [30] uses a limited amount of information, i.e., similar and dissimilar data pairs. It sequentially trains the hash functions one bit at a time. It gets the hash functions coefficients by spectral relaxation or by introducing smooth surrogate function instead of sign function and then applying gradient method to determine the hash function coefficients

PCA-based Hashing[56] defines hash functions by principal axes which are obtained using principal component analysis (PCA) on the data. It uses the top m eigenvectors which are orthogonal each other.

LDA Hashing[49] is a supervised method to use both positive pairs of similar data and negative pairs of dissimilar data. It finds the projection mappings which minimize the expectation of the Hamming distances on the positive pairs while maximizing it on the set of negative pairs. It uses an LDA(Linear Discrimiant Analysis)-like method or covariance difference method to determine projection mappings, i.e., hash functions.

Iterative Quantization[10] first quantizes the PCA results to get some binary codes, and then learns an orthogonal rotation matrix based on the resulting binary codes so that the quantization error of mapping the data to the vertices of binary hypercube is minimized.

Isotropic Hashing[18] is a hashing technique in which a hash function generates different number of bits to guarantee the equal variances. Most other binary code hashing techniques assign the same number of bits to each hash function.

Manhattan Hashing[19] encodes each projected dimension with multiple bits of natural binary code. Instead of

Hamming distance, the Manhattan distance between points in the hash code space is calculated for nearest neighbor search. It uses an existing technique to choose projection planes.

Spherical Hashing[15] uses hyperspheres to partition the space and spherical Hamming distance tailored to the hypersphere based binary coding scheme. It provides an efficient iterative optimization process to achieve balanced partitioning of data points for each hash function and independence between hashing functions.

Density Sensitive Hashing[28] produces hash functions which best agree with the distribution of the data set. It first applies k-means algorithm to the data set to generate small groups, and determines pairs of adjacent groups. Then, it finds the median planes to separate the adjacent groups and evaluates them with their entropy. As the hash functions, it chooses top-ranked median planes by high entropy scores.

Multi-Valued Hashing[50] uses the multiple hash tables based on projection-based LSH, and assigns the L_1 distance to neighboring buckets on the same projection vector. It computes the distance sum of assigned L_1 distances and selects the neighboring buckets to be examined by the L_1 distance sum.

Complementary Hashing[58] employs multiple complementary hash tables, learned sequentially in a boosting manner. In the hashing technique, for a given query, missed true neighbors are likely to be found in the active bucket of the next hash table.

4.2 Benchmark Data Sets

In the performance study, several benchmark data sets have been used. As the labeled data sets, there are Caltech101, MNIST, and CIFAR. Caltech-101[27] is a database of images with 101 distinct cateogries and one background category. Each category contains tens of images of about 300x200 pixels in dimension. MNIST[25] is a database of handwritten digits which consist of a training set of 60,000 examples, and a test of 10,000 examples.There are 7,000 images of 28x28 pixels for each digit in 0 to 9. CIFAR-10 and CIFAR-100 datasets[21] are labeled subset of the 80 milion Tiny images database[52]. CIFAR-10 consists of 32x32 pixels images in 10 classes and is divided into 5 training sets and one test set, each with 10,000 images. CIFAR-100 is just like CIFAR-10, except it has 100 classes each of which has 600 images.

Some unlabeled datasets have been used in performance evaluation of LSH techniques. LabelMe database[44] is an image database with about 187,000 images, about 62,000 annotated images, and about 659,000 labeled objects. Tiny image database[52] contains about 80 million images of 32x32 pixels which are subsampled.

Small images are sometimes regarded as data points by matrix-vector transformation. Large images are reduced into lower dimensional data by feature extraction methods like GIST[39] and SIFT[8]. GIST is an abstract representation of a scene to capture global features and SIFT is a widely used technique to extract local features on a scene.

The performance is evaluated in terms of precision, recall, and execution time. Precision is the ratio of the number of retrieved relevant points to the number of all retrieved points, and recall is the ratio of the number of retrieved relevant points to the number of relevant points.

4.3 Performance Comparison of Binary Code LSH Techniques

To see the relative performance of the LSH techniques, a simple meta-analysis was conducted in terms of precision and recall. Their performances vary depending on the test data sets, the number of bits used to encode the data. The developers compared their own method with some of existing methods, under the different experiment settings. In the comparisons, the developers' claims are recognized as is. The relative superiority among the LSH techniques has been analyzed and we obtained the summarized results shown in Figure 1. The superiority of performance is expressed by the relative positions on the X axis of the plot, where the right-hand side ones are superior to the-hand side ones when there is a path between them. For a pair of techniques which does not have a path going straingt from one technique to the other, there is no information about superiority between them, e.g., there is no straight going path between STH and OKL and thus it does not say anything about which one is better. The performance evaluation results conducted by the same research group are shown by linking the bars for the corresponding techniques.

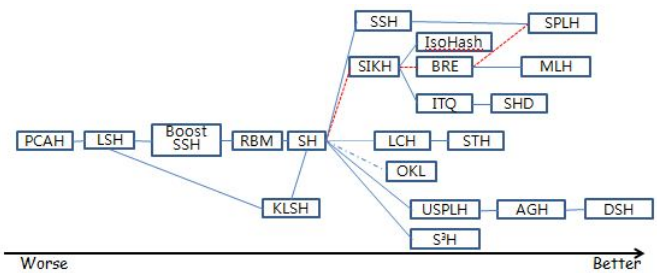

Figure 1: Performance Comparison of LSH Techniques

5. Conclusions

The locality-sensitive hashing techniques are very useful in nearest neighbor search and similar pair identifcation tasks for large volume of data. For the last decade, many LSH techniques have been developed and applied to many real applications including near-duplicate detection, image similarity identification, gene expression similarity identification, and audio/video similarity identification. They are very effective in handling high dimensional data like im-

ages, audio files, textual documents, and video files. Most techniques have been developed under the assumption that data are embedded in Euclidean space despite the data do not always live in the space. Further studies remains for LSH techniques of non-Euclidean spaces. Semantic information on objects might give different similarity between objects depending on the context. Semantic-based LSH studies are also an issue.

References

[1] A. Andoni and P. Indyk, "Near-optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions," *Comm. ACM*, vol.51, no.1, pp.117–122, 2008.

[2] S. Baluja and M. Covell, "Learning Forgiving Hash Functions: Algorithms and Large Scale Tests," *Proc. 20th Int. Joint Conf. on Artifical intelligence*, pp. 2663–2669, 2007.

[3] J. L. Bentley, "Multidimensional Binary Search Trees used for Associative Searching," *Commun. Ass. Comput. Mach.*, vol. 19, pp. 509-517, 1975.

[4] S. Boriah, V. Chandola, and V. Kumar, "Similarity Measures for Categorical Data: A Comparative Evaluation," *Proc. of the 8th SIAM Int. Conf. on Data Mining,* pp.243–254, 2008.

[5] A. Z. Broder, "On the Resemblance and Containment of Documents," *Proc. Compression and Complexity of Sequence*, pp. 21–29, 1997.

[6] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise Independent Permutations," *ACM Symposium on Theory of Computing*, pp. 327–336, 1998.

[7] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive Hashing Scheme based on p-stable Distribution," *Symp. on Computational Geometry*, pp. 253–262, 2004.

[8] D. G. Lowe, "Object recognition from local scale-invariant features," *Proc. of the Int.l Conf. on Computer Vision*, vol.2. pp.1150–1157, 1999.

[9] A. Gionis, P. Indyk, and R. Motwani, "Similarity Search in High Dimensions via Hashing," *Proc. of VLDB*, 1999.

[10] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin, "Iterative quantization: A Procrustean approach to learning binary codes for large-scale image retrieval," *IEEE Trans. Pattern Anal. Mach. Intell.*, 2012.

[11] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *Proc. of SIGMOD'84*, 1984.

[12] J. Hays and A. A. Efros, "Scene Completion Using Millions of Photographs," *Proc. of SIGGRAPH*, 2007.

[13] J. He, W. Liu, and S.-F. Chang, "Scalable Similarity Search with Optimized Kernel Hashing," *Proc. of IEEE Int. Conf. on Knowledge Discovery and Data Mining*, pp.1129–1138 2010.

[14] H. Henzinger, "Finding Nearest-Duplicate Web Pages: a Large-Scale Evaluation of Algorithms," *Proc. of SIGIR*, pp. 284–291, 2006.

[15] J.-P. Heo, Y. Lee, J. He, S.-F. Chang, and S.-E. Yoon, "Spectral Hashing," *Proc. of CVPR*, 2012.

[16] P. Indyk and R. Motwani, "Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality," *Proc. of STOC*, 1998.

[17] Q. Jiang and M. Sun, "Semi-supervised Simhash for Efficient Document Similarity Search," *Proc. The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies,* pp.93–101, 2011.

[18] W. Kong and W.-J. Li, "Isotropic Hashing," *Proc. of NIPS2012*, 2012.

[19] W. Kong, W.-J. Li, and M. Guo, "Manhattan hashing for large-scale image retrieval," *Proc. of SIGIR*, 2012.

[20] Y. Koren, "Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model," *KDD*, 2008.

[21] A. Krizhevsky, V. Nair, and G. Hinton, The CIFAR-10 and CIFAR-100 Databases, http://www.cs.toronto.edu/ kriz/cifar.html.

[22] B. Kulis and K. Grauman, "Kernelized Locality-sensitive Hashing," *Proc. of 12th Int. Conf. on Computer Vision*, 2009.

[23] B. Kulis and T. Barrell, "Learning to Hash with Binary Reconstructive Embeddings," *Tech. Rep.*, UC Berkeley, 2009.

[24] B. Kulis, P. Jain, and K. Grauman, "Fast Similarity Search for Learned Metrics," *IEEE TPAMI*, vol.31, no. 12, 2009.

[25] Y. LeCun and C. Cortes, MNIST Database, http://yann. lecun.com/exdb/mnist/.

[26] K. M. Lee and K.M. Lee, "A Locality Sensitive Hashing Technique for Categorical Data," *Applied Mech. And Mat.*, 2013(to appear).

[27] F.-F. Li, M. Andreetto, and M. A. Ranzato, Caltech 101 Database, http://www.vision.caltech.edu/Image_Datasets/Caltech101/.

[28] Y. Lin, D. Cai, "Density Sensitive Hashing," *ArXiv e-prints arXiv:1205.2930*, 2012.

[29] T.Liu, A. W. Moore, A. Gray, and K. Yang, "An Investigation of Practical Approximate Nearest Neighbor Algorithms," *Proc. of NIPS*, pp.825–832. 2005.

[30] W. Liu, J. Wang, S. Kumar, and S.-F. Chang, "Hashing with Graphs," *Proc. of Int. Conf. on Machine Learning*, 2011.

[31] U. von Luxburg, "A Tutorial on Spectral Clustering," *Stat. Comput.*, vol.17, pp. 395–416, 2007.

[32] U. Manber, "Finding Similar Files in a Large File System," *Proc. USENIX Conference*, pp. 1–10, 1994.

[33] Y. Matsushita and T. Wada, "Principal Component Hashing: An Accelerated Approximate Nearest Neighbor Search," *Proc. of PSIVT*, 2009.

[34] B. McFee and G. Lanckriet, "Large-Scale Music Similarity Search With Spatial Trees," *Proc. of ISMIR*, 2011.

[35] G. A. Miller, R. Beckwith, C. D. Fellbaum, D. Gross, and K. Miller, "WordNet: An Online Lexical Database," *Int. J. Lexicograph*, vol.3, no.4, pp. 235–244, 1990.

[36] Y. Mu, J. Shen, and S. Yan, "Weakly-Supervised Hashing in Kernel Space," *Proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp.3344–3351, 2010.

[37] D. Nister and H. Stewenius, "Scalable Recognition with a Vocabulary Tree," *Proc. CVPR* , vol. 5, 2006.

[38] M. Norouzi and D. J. Fleet, "Minimal Loss Hashing for Compact Binary Codes," *Proc. of ICML*, 2011.

[39] A. Oliva, A. Torralba, "Modeling the Shape of the Scene: A Holistic Representation of the Spatial Envelope," *Int. J. of Computer Vision*, vol.42,no.3, pp.145–175, 1989.

[40] S. Omohundro, "Five balltree construction algorithms," *Technical Report, ICSI*, 1989.

[41] S. Pandey, A. Broder, and F. Chierichetti, "Nearest-Neighbor Caching for Content-Match Applications," *Proc. of WWW Conf.*, 2009.

[42] M. Potthast and B. Stein, "New Issues in Near-Duplicate Detection," *Data Analysis, Machine Learning and Applications*, pp. 601–609, Springer, 2008.

[43] M. Raginsky, and S. Lazebnik, "Locality-sensitive binary codes from shift-invariant kernels," *Proc. of NIPS*, 2009.

[44] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman, LabelMe, http://labelme.csail.mit.edu/.

[45] R. R. Salakhutdinov and G.E. Hinton, "Semantic hashing," *Proc. of Int.l J. of Approximate Reasoning*, vol.50, no.7, 2009.

[46] R. E. Schapire, "The Boosting Approach to Machine Learning : An Overview," *Nonlinear Estimation and Classification, Springer*, 2003.

[47] G. Shakhnarovich, P. Viola, and T. Darrell, "Fast Pose Estimation with Parameter Sensitive Hashing," *Proc. ICCV*, 2003.

[48] B. Stein, S. M. Eissen, and M. Potthas, "Strategies for retrieving plagiarized documents," *SIGIR*, 2007.

[49] C. Strecha, A. M. Bronstein, M. M. Bronstein, and P. Fua," LDAHash: Improved Matching with Smaller Descriptors," *IEEE TPAMI*, vol34, no.1, 2012.

[50] M. Tata, T. Muto, M. Iwamura, and K. Kise, "Extension of Approximate Nearest Neighbor Search Based on Multi-Valued Expression on Closeness to General Distributions," DEIM Forum, 2010(in Japanese).

[51] M. Theodbald, J. Siddhaarth, and A. Paepcke, "SpotSigs: robust and efficient near duplicate detection in large web collections," *Proc. ACM SIGIR*, Singapore, pp.563-570, 2008.

[52] A. Torralba, R. Fergus, and Y. Weiss, "Small Codes and Large Image Databases for Recognition," *Proc. of CVPR*, pp.1–8, 2008.

[53] A. Torralba, R. Fergus, and W. T. Freeman, 80 Million Tiny Images: A Large Data Set for Nonparametric Object and Scene Recognition," *IEEE PAMI*, vol.30, no.11, 2008.

[54] J. K. Uhlmann, "Satisfying general proximity/similarity queries with metric trees," *Information Processing Letters,*, vol.4, pp.175–179, 1991.

[55] J. Wang, S. Kumar, and S.-F. Chang, "Sequential Projection Learning for Hashing with Compact Codes," *Proc. of Int. Conf. on Machine Learning*, 2010.

[56] J. Wang, S. Kumar, and S.-F. Chang, "Semi-Supervised Hashing for Large Scale Search," *IEEE PAMI*, vol.34, no.12, 2012.

[57] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," *Proc. of Neural Information Processing Systems*, pp.1753–1760, 2008.

[58] H. Xu, J. Wang, Z. Li, G. Zeng, S. Le, and N. Yu, "Complementary Hashing for Approximate Nearest Neighbor Search," *Proc. of IEEE Int. Conf. on Computer Vision*, 2011.

[59] D. Zhang, J. Wang, D. Cai, and J. Lu, "Self-taught hashing for fast similarity search," *Proc. SIGIR*, pp.18–25, 2010.

[60] D. Zhang, J. Wang, D. Cai, and J. Lu, "Laplacian Co-hashing of Terms and Documents," *Proc. ECIR2010, LNCS*, vol.5993, pp.577–580, 2010.

Keon Myung Lee

Professor of Chungbuk National University, Korea
Research Area: machine learning, data mining, big data processing, intelligent services
E-mail : kmlee@cbnu.ac.kr