INVITED PAPER

Learning to Hash for Indexing Big Data—A Survey

This paper provides readers with a systematic understanding of insights, pros, and cons of the emerging indexing and search methods for Big Data.

By Jun Wang, *Member IEEE*, Wei Liu, *Member IEEE*, Sanjiv Kumar, *Member IEEE*, and Shih-Fu Chang, *Fellow IEEE*

ABSTRACT | The explosive growth in Big Data has attracted much attention in designing efficient indexing and search methods recently. In many critical applications such as large-scale search and pattern matching, finding the nearest neighbors to a query is a fundamental research problem. However, the straightforward solution using exhaustive comparison is infeasible due to the prohibitive computational complexity and memory requirement. In response, approximate nearest neighbor (ANN) search based on hashing techniques has become popular due to its promising performance in both efficiency and accuracy. Prior randomized hashing methods, e.g., locality-sensitive hashing (LSH), explore data-independent hash functions with random projections or permutations. Although having elegant theoretic guarantees on the search quality in certain metric spaces, performance of randomized hashing has been shown insufficient in many real-world applications. As a remedy, new approaches incorporating data-driven learning methods in development of advanced hash functions have emerged. Such learning-to-hash methods exploit information such as data distributions or class labels when optimizing the hash codes or functions. Importantly, the learned hash codes are able to preserve the proximity of neighboring data in the original feature spaces in the hash code spaces. The goal of this paper is to provide readers with systematic understanding of insights, pros, and cons of the emerging techniques. We provide a comprehensive survey of the learning-to-hash framework and representative techniques of various types, including unsupervised, semisupervised, and supervised. In addition, we also summarize recent hashing approaches utilizing the deep learning models. Finally, we discuss the future direction and trends of research in this area.

KEYWORDS | Approximate nearest neighbor (ANN) search; deep learning; learning to hash; semisupervised learning; supervised learning; unsupervised learning

I. Introduction

The advent of Internet has resulted in massive information overloading in the recent decades. Today, the World Wide Web has over 366 million accessible websites, containing more than one trillion webpages.[1] For instance, Twitter receives over 100 million tweets per day, and Yahoo! exchanges over three billion messages per day. Besides the overwhelming textual data, the photo sharing website Flickr has more than five billion images available, where images are still being uploaded at the rate of over 3000 images per minute. Another rich media sharing website YouTube receives more than 100 h of videos uploaded per minute. Due to the dramatic increase in the size of the data, modern information technology infrastructure has to deal with such gigantic databases. In fact, compared to the cost of storage, searching for relevant content in massive databases turns out to be an even more challenging task. In particular, searching for rich media data, such as audio, images, and videos, remains a major challenge since there exist major gaps between available solutions and practical needs in both accuracy and computational costs. Besides the widely used text-based commercial search engines such as Google and Bing, content-based image retrieval (CBIR) has attracted substantial attention in the past decade [1]. Instead of relying on textual keywords-based indexing structures,

[1]The number of webpages is estimated based on the number of indexed links by Google in 2008.

Manuscript received April 08, 2015; revised August 31, 2015; accepted September 16, 2015. Date of current version December 18, 2015.

J. Wang is with the School of Computer Science and Software Engineering, East China Normal University, Shanghai 200062, China (e-mail: wongjun@gmail.com).
W. Liu is with IBM T. J. Watson Research Center, Yorktown Heights, NY 10598 USA (e-mail: wliu@ee.columbia.edu).
S. Kumar is with Google Research, New York, NY 10011 USA (e-mail: sanjivk@google.com).
S.-F. Chang is with the Department of Electrical Engineering and Computer Science, Columbia University, New York, NY 10027 USA (e-mail: sfchang@ee.columbia.edu).

Digital Object Identifier: 10.1109/JPROC.2015.2487976

CBIR requires efficiently indexing media content in order to directly respond to visual queries.

Searching for similar data samples in a given database essentially relates to the fundamental problem of nearest neighbor search [2]. Exhaustively comparing a query point q with each sample in a database $\mathcal{X}$ is infeasible because the linear time complexity $\mathcal{O}(|\mathcal{X}|)$ tends to be expensive in realistic large-scale settings. Besides the scalability issue, most practical large-scale applications also suffer from the curse of dimensionality [3], since data under modern analytics usually contain thousands or even tens of thousands of dimensions, e.g., in documents and images. Therefore, beyond the infeasibility of the computational cost for exhaustive search, the storage constraint originating from loading original data into memory also becomes a critical bottleneck. Note that retrieving a set of approximate nearest neighbors (ANNs) is often sufficient for many practical applications. Hence, a fast and effective indexing method achieving sublinear $(o(|\mathcal{X}|))$, logarithmic $(\mathcal{O}(\log|\mathcal{X}|))$, or even constant $(\mathcal{O}(1))$ query time is desired for ANN search. Tree-based indexing approaches, such as KD tree [4], ball tree [5], metric tree [6], and vantage point tree [7], have been popular during the past several decades. However, tree-based approaches require significant storage costs (sometimes more than the data itself). In addition, the performance of tree-based indexing methods dramatically degrades when handling high-dimensional data [8]. More recently, product quantization techniques have been proposed to encode high-dimensional data vectors via subspace decomposition for efficient ANN search [9], [10].

Unlike the recursive partitioning used by tree-based indexing methods, hashing methods repeatedly partition the entire data set and derive a single hash "bit"[2] from each partitioning. In binary-partitioning-based hashing, input data are mapped to a discrete code space called Hamming space, where each sample is represented by a binary code. Specifically, given N D-dim vectors $\mathbf{X} \in \mathbb{R}^{D\times N}$, the goal of hashing is to derive suitable K-bit binary codes $\mathbf{Y} \in \mathbb{B}^{K\times N}$. To generate $\mathbf{Y}$, K binary hash functions $\{h_k : \mathbb{R}^D \mapsto \mathbb{B}\}_{k=1}^{K}$ are needed. Note that hashing-based ANN search techniques can lead to substantially reduced storage as they usually store only compact binary codes. For instance, 80 million tiny images (32 × 32 pixels, grayscale, double type) cost around 600 GB [11], but can be compressed into 64-b binary codes requiring only about 600 MB. In many cases, hash codes are organized into a hash table for inverse table lookup, as shown in Fig. 1. One advantage of hashing-based indexing is that hash table lookup takes only constant query time. In fact, in many cases, another alternative way of finding the nearest neighbors in the code space by explicitly computing Hamming distance with all the database items can be done very efficiently as well.

[2]Depending on the type of the hash function used, each hash may return either an integer or simply a binary bit. In this survey, we primarily focus on binary hashing techniques as they are used most commonly due to their computational and storage efficiency.

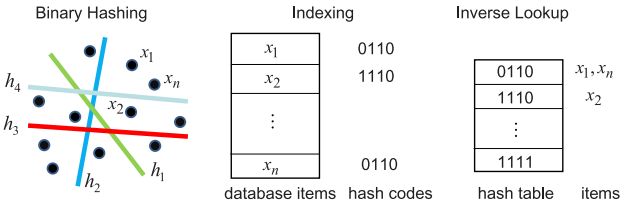

Fig. 1. ***Illustration of linear-projection-based binary hashing, indexing, and hash table construction for fast inverse lookup.***

Hashing methods have been intensively studied and widely used in many different fields, including computer graphics, computational geometry, telecommunication, computer vision, etc., for several decades [12]. Among these methods, the randomized scheme of locality-sensitive hashing (LSH) is one of the most popular choices [13]. A key ingredient in LSH family of techniques is a hash function that, with high probabilities, returns the same bit for the nearby data points in the original metric space. LSH provides interesting asymptotic theoretical properties leading to performance guarantees. However, LSH-based randomized techniques suffer from several crucial drawbacks. First, to achieve desired search precision, LSH often needs to use long hash codes, which reduces the recall. Multiple hash tables are used to alleviate this issue, but it dramatically increases the storage cost as well as the query time. Second, the theoretical guarantees of LSH only apply to certain metrics such as ℓ_p $(p \in (0, 2])$ and Jaccard [14]. However, returning ANNs in such metric spaces may not lead to good search performance when semantic similarity is represented in a complex way instead of a simple distance or similarity metric. This discrepancy between semantic and metric spaces has been recognized in the computer vision and machine learning communities, namely as semantic gap [15].

To tackle the aforementioned issues, many hashing methods have been proposed recently to leverage machine learning techniques to produce more effective hash codes [16]. The goal of learning to hash is to learn data-dependent and task-specific hash functions that yield compact binary codes to achieve good search accuracy [17]. In order to achieve this goal, sophisticated machine learning tools and algorithms have been adapted to the procedure of hash function design, including the boosting algorithm [18], distance metric learning [19], asymmetric binary embedding [20], kernel methods [21], [22], compressed sensing [23], maximum margin learning [24], sequential learning [25], clustering analysis [26], semisupervised learning [14], supervised learning [22], [27], graph learning [28], and so on. For instance, in the specific application of image search, the similarity (or distance) between image pairs is usually not defined via a simple metric. Ideally, one would like to provide

pairs of images that contain "similar" or "dissimilar" images. From such pairwise labeled information, a good hashing mechanism should be able to generate hash codes which preserve the semantic consistency, i.e., semantically similar images should have similar codes. Both the supervised and semisupervised learning paradigms have been explored using such pairwise semantic relationships to learn semantically relevant hash functions [11], [29]–[31]. In this paper, we will survey important representative hashing approaches and also discuss the future research directions.

The remainder of this paper is organized as follows. In Section II, we present necessary background information, prior randomized hashing methods, and the motivations of studying hashing. Section III gives a high-level overview of emerging learning-based hashing methods. In Section IV, we survey several popular methods that fall into the learning-to-hash framework. In addition, Section V describes the recent development of using neural networks to perform deep learning of hash codes. Section VI discusses advanced hashing techniques and large-scale applications of hashing. Several open issues and future directions are described in Section VII.

II. Notations and Background

In this section, we will first present the notations, as summarized in Table 1. Then, we will briefly introduce the conceptual paradigm of hashing-based ANN search. Finally, we will present some background information on hashing methods, including the introduction of two well-known randomized hashing techniques.

A. Notations

Given a sample point $\mathbf{x} \in \mathbb{R}^D$, one can employ a set of hash functions $H = \{h_1, \ldots, h_K\}$ to compute a K-bit binary code $\mathbf{y} = \{y_1, \ldots, y_K\}$ for $\mathbf{x}$ as

$$\mathbf{y} = \{h_1(\mathbf{x}), \ldots, h_2(\mathbf{x}), \ldots, h_K(\mathbf{x})\} \tag{1}$$

where the kth bit is computed as $y_k = h_k(\mathbf{x})$. The hash function performs the mapping as $h_k : \mathbb{R}^D \longrightarrow \mathbb{B}$. Such a binary encoding process can also be viewed as mapping the original data point to a binary valued space, namely Hamming space

$$H : \mathbf{x} \rightarrow \{h_1(\mathbf{x}), \ldots, h_K(\mathbf{x})\}. \tag{2}$$

Given a set of hash functions, we can map all the items in the database $\mathbf{X} = \{\mathbf{x}_n\}_{n=1}^N \in \mathbb{R}^{D \times N}$ to the corresponding binary codes as

$$\mathbf{Y} = H(\mathbf{X}) = \{h_1(\mathbf{X}), h_2(\mathbf{X}), \ldots, h_K(\mathbf{X})\}$$

where the hash codes of the data $\mathbf{X}$ are $\mathbf{Y} \in \mathbb{B}^{K \times N}$.

Table 1 Summary of Notations

Symbol	Definition
N	number of data points
D	dimensionality of data points
K	number of hash bits
i, j	indices of data points
k	index of a hash function
$\mathbf{x}_i \in \mathbb{R}^D, \mathbf{x}_j \in \mathbb{R}^D$	the ith and jth data point
$\mathcal{S}_i, \mathcal{S}_j$	the ith and jth set
$\mathbf{q}_i \in \mathbb{R}^D$	a query point
$\mathbf{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_N] \in \mathbb{R}^{D \times N}$	data matrix with points as columns
$\mathbf{y}_i \in \{1, -1\}^K$, or $\mathbf{y}_i \in \{0, 1\}^K$	hash codes of data points $\mathbf{x}_i$ and $\mathbf{x}_j$
$\mathbf{y}_{k\cdot} \in \mathbb{B}^{N \times 1}$	the k-th hash bit of N data points
$\mathbf{Y} = [\mathbf{y}_1, \cdots, \mathbf{y}_N] \in \mathbb{B}^{K \times N}$	hash codes of data $\mathbf{X}$
θ_{ij}	angle between data points $\mathbf{x}_i$ and $\mathbf{x}_j$
$h_k : \mathbf{R}^D \rightarrow \{1, -1\}$	the k-th hash function
$H = [h_1, \cdots, h_K] : \mathbf{R}^D \rightarrow \{1, -1\}^K$	K hash functions
$J(\mathcal{S}_i, \mathcal{S}_j)$	Jaccard similarity between sets $\mathcal{S}_i$ and $\mathcal{S}_j$
$J(\mathbf{x}_i, \mathbf{x}_j)$	Jaccard similarity between vectors $\mathbf{x}_i$ and $\mathbf{x}_j$
$d_{\mathcal{H}}(\mathbf{y}_i, \mathbf{y}_j)$	Hamming distance between $\mathbf{y}_i$ and $\mathbf{y}_j$
$d_{\mathcal{WH}}(\mathbf{y}_i, \mathbf{y}_j)$	weighted Hamming distance between $\mathbf{y}_i$ and $\mathbf{y}_j$
$(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{M}$	a pair of similar points
$(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{C}$	a pair of dissimilar points
$(\mathbf{q}_i, \mathbf{x}_i^+, \mathbf{x}_i^-)$	a ranking triplet
$S_{ij} = sim(\mathbf{x}_i, \mathbf{x}_j)$	similarity between data points $\mathbf{x}_i$ and $\mathbf{x}_j$
$\mathbf{S} \in \mathbb{R}^{N \times N}$	similarity matrix of data $\mathbf{X}$
$\mathcal{P}_{\mathbf{w}}$	a hyperplane with its normal vector $\mathbf{w}$

After computing the binary codes, one can perform ANN search in Hamming space with significantly reduced computation. Hamming distance between two binary codes $\mathbf{y}_i$ and $\mathbf{y}_j$ is defined as

$$d_{\mathcal{H}}(\mathbf{y}_i, \mathbf{y}_j) = |\mathbf{y}_i - \mathbf{y}_j| = \sum_{k=1}^{K} \left| h_k(\mathbf{x}_i) - h_k(\mathbf{x}_j) \right| \quad (3)$$

where $\mathbf{y}_i = [h_1(\mathbf{x}_i), \ldots, h_k(\mathbf{x}_i), \ldots, h_K(\mathbf{x}_i)]$ and $\mathbf{y}_j = [h_1(\mathbf{x}_j), \ldots, h_k(\mathbf{x}_j), \ldots, h_K(\mathbf{x}_j)]$. Note that the Hamming distance can be calculated in an efficient way as a bitwise logic operation. Thus, even conducting exhaustive search in the Hamming space can be significantly faster than doing the same in the original space. Furthermore, through designing a certain indexing structure, the ANN search with hashing methods can be even more efficient. Below we describe the pipeline of a typical hashing-based ANN search system.

B. Pipeline of Hashing-Based ANN Search

There are three basic steps in ANN search using hashing techniques: designing hash functions, generating hash codes and indexing the database items, and online querying using hash codes. These steps are described in detail below.

1) Designing Hash Functions: There exist a number of ways of designing hash functions. Randomized hashing approaches often use random projections or permutations. The emerging learning-to-hash framework exploits the data distribution and often various levels of supervised information to determine optimal parameters of the hash functions. The supervised information includes pointwise labels, pairwise relationships, and ranking orders. Due to their efficiency, the most commonly used hash functions are of the form of a generalized linear projection

$$h_k(x) = \operatorname{sgn}\left(f\left(\mathbf{w}_k^{\top}\mathbf{x} + b_k\right)\right). \quad (4)$$

Here $f(\cdot)$ is a prespecified function which can be possibly nonlinear. The parameters to be determined are $\{\mathbf{w}_k, b_k\}_{k=1}^{K}$, representing the projection vector $\mathbf{w}_k$ and the corresponding intercept b_k. During the training procedure, the data $\mathbf{X}$, sometimes along with supervised information, are used to estimate these parameters. In addition, different choices of $f(\cdot)$ yield different properties of the hash functions, leading to a wide range of hashing approaches. For example, LSH keeps $f(\cdot)$ to be an identity function, while shift-invariant kernel-based hashing and spectral hashing choose $f(\cdot)$ to be a shifted cosine or sinusoidal function [32], [33].

Note that the hash functions given by (4) generate the codes as $h_k(\mathbf{x}) \in \{-1, 1\}$. One can easily convert them into binary codes from $\{0, 1\}$ as

$$y_k = \frac{1}{2}(1 + h_k(\mathbf{x})). \quad (5)$$

Without loss of generality, in this survey, we will use the term hash codes to refer to either $\{0, 1\}$ or $\{-1, 1\}$ form, which should be clear from the context.

2) Indexing Using Hash Tables: With a learned hash function, one can compute the binary codes $\mathbf{Y}$ for all the items in a database. For K hash functions, the codes for the entire database cost only $NK/8$ bytes. Assuming the original data to be stored in double-precision floating-point format, the original storage costs $8ND$ bytes. Since the massive data sets are often associated with thousands of dimensions, the computed hash codes significantly reduce the storage cost by hundreds and even thousands of times.

In practice, the hash codes of the database are organized as an inverse lookup, resulting in a hash table or a hash map. For a set of K binary hash functions, one can have at most 2^K entries in the hash table. Each entry, called a hash bucket, is indexed by a K-bit hash code. In the hash table, one keeps only those buckets that contain at least one database item. Fig. 1 shows an example of using binary hash functions to index the data and construct a hash table. Thus, a hash table can be seen as an inverse-lookup table, which can return all the database items corresponding to a certain code in constant time. This procedure is key to achieving speedup by many hashing-based ANN search techniques. Since most of the buckets from 2^K possible choices are typically empty, creating an inverse lookup can be a very efficient way of even storing the codes if multiple database items end up with the same codes.

3) Online Querying With Hashing: During the querying procedure, the goal is to find the nearest database items to a given query. The query is first converted into a code using the same hash functions that mapped the database items to codes. One way to find nearest neighbors of the query is by computing the Hamming distance between the query code to all the database codes. Note that the Hamming distance can be rapidly computed using logical XOR operation between binary codes as

$$d_{\mathcal{H}}(\mathbf{y}_i, \mathbf{y}_j) = \mathbf{y}_i \oplus \mathbf{y}_j. \quad (6)$$

On modern computer architectures, this is achieved efficiently by running XOR instruction followed by popcount. With the computed Hamming distance between the query and each database item, one can perform

exhaustive scan to extract the approximate nearest neighbors of the query. Although this is much faster than the exhaustive search in the original feature space, the time complexity is still linear. An alternative way of searching for the neighbors is by using the inverse lookup in the hash table and returning the data points within a small Hamming distance r of the query. Specifically, given a query, point $\mathbf{q}$, and its corresponding, hash code $\mathbf{y}_q = H(\mathbf{q})$, we can use the hash lookup table to retrieve all the database points $\tilde{\mathbf{y}}$ whose hash codes fall within the Hamming ball of radius r centered at $\mathbf{y}_q$, i.e., $d_{\mathcal{H}}(\tilde{\mathbf{y}}, H(\mathbf{q})) \leq r$. As shown in Fig. 2, for a K-bit binary code, a total of $\sum_{l=0}^{r} \binom{K}{l}$ possible codes will be within Hamming radius of r. Thus, one needs to search $O(K^r)$ buckets in the hash table. The union of all the items falling into the corresponding hash buckets is returned as the search result. The inverse lookup in a hash table has constant time complexity independent of the database size N. In practice, a small value of r ($r = 1, 2$ is commonly used) is used to avoid the exponential growth in the possible code combinations that need to be searched.

C. Randomized Hashing Methods

Randomized hashing, e.g., LSH family, has been a popular choice due to its simplicity. In addition, it has interesting proximity preserving properties. A binary hash function $h(\cdot)$ from LSH family is chosen such that the probability of two points having the same bit is proportional to their (normalized) similarity, i.e.,

$$P\{h(\mathbf{x}_i) = h(\mathbf{x}_j)\} = \text{sim}(\mathbf{x}_i, \mathbf{x}_j). \tag{7}$$

Here $\text{sim}(\cdot, \cdot)$ represents similarity between a pair of points in the input space, e.g., cosine similarity or Jaccard similarity [34]. In this section, we briefly review two categories of randomized hashing methods, i.e., random-projection-based and random-permutation-based approaches.

1) Random-Projection-Based Hashing: As a representative member of the LSH family, random-projection-based hash (RPH) functions have been widely used in different applications. The key ingredient of RPH is to map nearby points in the original space to the same hash bucket with a high probability. This equivalently preserves the locality in the original space in the Hamming space. Typical examples of RPH functions consist of a random projection $\mathbf{w}$ and a random shift b as

$$h_k(x) = \text{sgn}\left(\mathbf{w}_k^\top \mathbf{x} + b_k\right). \tag{8}$$

The random vector $\mathbf{w}$ is constructed by sampling each component of $\mathbf{w}$ randomly from a standard Gaussian distribution for cosine distance [34].

It is easy to show that the collision probability of two samples $\mathbf{x}_i, \mathbf{x}_j$ falling into the same hash bucket is determined by the angle θ_{ij} between these two sample vectors, as shown in Fig. 3. One can show that

$$\Pr\left[h_k(\mathbf{x}_i) = h_k(\mathbf{x}_j)\right] = 1 - \frac{\theta_{ij}}{\pi} = 1 - \frac{1}{\pi}\cos^{-1}\frac{\mathbf{x}_i^\top \mathbf{x}_j}{\|\mathbf{x}_i\|\|\mathbf{x}_j\|}. \tag{9}$$

The above collision probability gives the asymptotic theoretical guarantees for approximating the cosine similarity defined in the original space. However, long hash codes are required to achieve sufficient discrimination for high precision. This significantly reduces the recall if hash-table-based inverse lookup is used for search. In order to balance the tradeoff of precision and recall, one has to construct multiple hash tables with long hash codes, which increases both storage and computation costs. In particular, with hash codes of length K, it is required to construct a sufficient number of hash tables to ensure the desired performance bound [35]. Given l K-bit tables, the collision probability is given as

$$P\{H(\mathbf{x}_i) = H(\mathbf{x}_j)\} \propto l \cdot \left[1 - \frac{1}{\pi}\cos^{-1}\frac{\mathbf{x}_i^\top \mathbf{x}_j}{\|\mathbf{x}_i\|\|\mathbf{x}_j\|}\right]^K. \tag{10}$$

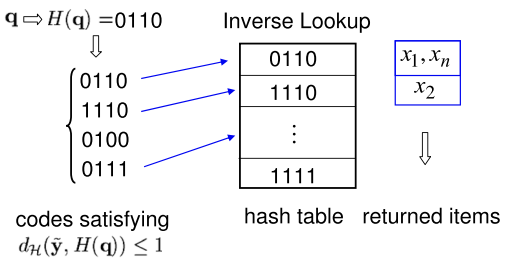

Fig. 2. *Procedure of inverse lookup in hash table, where* q *is the query mapped to a 4-b hash code "0100" and the returned approximate nearest neighbors within Hamming radius 1 are* x_1, x_2, x_n.

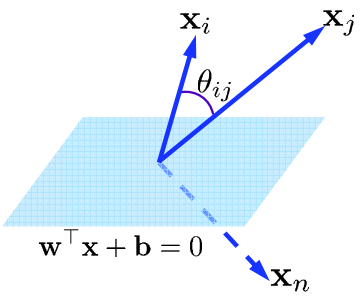

Fig. 3. *Illustration of random-hyperplane-partitioning-based hashing method.*

To balance the search precision and recall, the length of hash codes should be long enough to reduce false collisions (i.e., nonneighbor samples falling into the same bucket). Meanwhile, the number of hash tables l should be sufficiently large to increase the recall. However, this is inefficient due to extra storage cost and longer query time.

To overcome these drawbacks, many practical systems adapt various strategies to reduce the storage overload and to improve the efficiency. For instance, a self-tuning indexing technique, called LSH forest, was proposed in [36], which aims at improving the performance without additional storage and query overhead. In [37] and [38], a technique called MultiProbe LSH was developed to reduce the number of required hash tables through intelligently probing multiple buckets in each hash table. In [39], nonlinear randomized Hadamard transforms were explored to speed up the LSH-based ANN search for Euclidean distance. In [40], BayesLSH was proposed to combine Bayesian inference with LSH in a principled manner, which has probabilistic guarantees on the quality of the search results in terms of accuracy and recall. However, the random-projections-based hash functions ignore the specific properties of a given data set and thus the generated binary codes are data independent, which leads to less effective performance compared to the learning-based methods to be discussed later.

In machine learning and data mining community, recent methods tend to leverage data-dependent and task-specific information to improve the efficiency of random-projection-based hash functions [16]. For example, incorporating kernel learning with LSH can help generalize ANN search from a standard metric space to a wide range of similarity functions [41], [42]. Furthermore, metric learning has been combined with randomized LSH functions to explore a set of pairwise similarity and dissimilarity constraints [19]. Other variants of LSH techniques include superbit LSH [43], boosted LSH [18], as well as nonmetric LSH [44]

2) Random-Permutation-Based Hashing: Another well-known paradigm from the LSH family is min-wise independent permutation hashing (min-hash), which has been widely used for approximating Jaccard similarity between sets or vectors. Jaccard is a popular choice for measuring similarity between documents or images. A typical application is to index documents and then identify near-duplicate samples from a corpus of documents [45], [46]. The Jaccard similarity between two sets $\mathcal{S}_i$ and $\mathcal{S}_j$ is defined as $J(\mathcal{S}_i, \mathcal{S}_j) = (\mathcal{S}_i \cap \mathcal{S}_j)/(\mathcal{S}_i \cup \mathcal{S}_j)$. A collection of sets $\{\mathcal{S}_i\}_{i=1}^N$ can be represented as a characteristic matrix $\mathbf{C} \in \mathbb{B}^{M\times N}$, where M is the cardinality of the universal set $\mathcal{S}_1 \cup \cdots \cup \mathcal{S}_N$. Here the rows of $\mathbf{C}$ represent the elements of the universal set and the columns correspond to the sets. The element $c_{di} = 1$ indicates that the dth element is a member of the ith set, $c_{di} = 0$ otherwise. Assume a random permutation $\pi_k(\cdot)$ that assigns the index of the dth element as $\pi_k(d) \in \{1, \ldots, D\}$. It is easy to see that the random permutation satisfies two properties: $\pi_k(d) \neq \pi_k(l)$ and $\Pr[\pi_k(d) > \pi_k(l)] = 0.5$. A random-permutation-based min-hash signature of a set $\mathcal{S}_i$ is defined as the minimum index of the nonzero element after performing permutation using π_k

$$h_k(\mathcal{S}_i) = \min_{d\in\{1,\ldots,D\}, c_{\pi_k(d)i}=1} \pi_k(d). \tag{11}$$

Note that such a hash function holds a property that the chance of two sets having the same min-hash values is equal to the Jaccard similarity between them [47]

$$Pr\left[h_k(\mathcal{S}_i) = h_k(\mathcal{S}_j)\right] = J(\mathcal{S}_i, \mathcal{S}_j). \tag{12}$$

The definition of the Jaccard similarity can be extended to two vectors $\mathbf{x}_i = \{x_{i1}, \ldots, x_{id}, \ldots, x_{iD}\}$ and $\mathbf{x}_j = \{x_{j1}, \ldots, x_{jd}, \ldots, x_{jD}\}$ as

$$J(\mathbf{x}_i, \mathbf{x}_j) = \frac{\sum_{d=1}^{D} \min(x_{id}, x_{jd})}{\sum_{d=1}^{D} \max(x_{id}, x_{jd})}.$$

Similar min-hash functions can be defined for the above vectors and the property of the collision probability shown in (12) still holds [48]. Compared to the random-projection-based LSH family, the min-hash functions generate nonbinary hash values that can be potentially extended to continuous cases. In practice, the min-hash scheme has shown powerful performance for high-dimensional and sparse vectors like the bag-of-word representation of documents or feature histograms of images. In a large-scale evaluation conducted by Google Inc., the min-hash approach outperforms other competing methods for the application of webpage duplicate detection [49]. In addition, the min-hash scheme is also applied for Google news personalization [50] and near-duplicate image detection [51], [52]. Some recent efforts have been made to further improve the min-hash technique, including b-bit minwise hashing [53], [54], one permutation approach [55], geometric min-Hashing [56], and a fast computing technique for image data [57].

III. Categories of Learning-Based Hashing Methods

Among the three key steps in hashing-based ANN search, design of improved data-dependent hash functions has been the focus in learning-to-hash paradigm. Since the proposal of LSH in [58], many new hashing techniques have been developed. Note that most of the emerging

hashing methods are focused on improving the search performance using a single hash table. The reason is that these techniques expect to learn compact discriminative codes such that searching within a small Hamming ball of the query or even exhaustive scan in Hamming space is both fast and accurate. Hence, in the following, we primarily focus on various techniques and algorithms for designing a single hash table. In particular, we provide different perspectives such as the learning paradigm and hash function characteristics to categorize the hashing approaches developed recently. It is worth mentioning that a few recent studies have shown that exploring the power of multiple hash tables can sometimes generate superior performance. In order to improve precision as well as recall, Xu *et al.* developed multiple complementary hash tables that are sequentially learned using a boosting-style algorithm [31]. Also, in cases when the code length is not very large and the number of database points is large, exhaustive scan in Hamming space can be done much faster by using multitable indexing as shown by Norouzi *et al.* [59].

A. Data Dependent Versus Data Independent

Based on whether design of hash functions requires analysis of a given data set, there are two high-level categories of hashing techniques: data independent and data dependent. As one of the most popular data-independent approaches, random projection has been used widely for designing data-independent hashing techniques such as LSH and SIKH mentioned earlier. LSH is arguably the most popular hashing method and has been applied to a variety of problem domains, including information retrieval and computer vision. In both LSH and SIKH, the projection vector $\mathbf{w}$ and intersect b, as defined in (4), are randomly sampled from certain distributions. Although these methods have strict performance guarantees, they are less efficient since the hash functions are not specifically designed for a certain data set or search task. Based on the random projection scheme, there have been several efforts to improve the performance of the LSH method [37], [39], [41].

Realizing the limitation of data-independent hashing approaches, many recent methods use data and possibly some form of supervision to design more efficient hash functions. Based on the level of supervision, the data-dependent methods can be further categorized into three subgroups, as described below.

B. Unsupervised, Supervised, and Semisupervised

Many emerging hashing techniques are designed by exploiting various machine learning paradigms, ranging from unsupervised and supervised to semisupervised settings. For instance, unsupervised hashing methods attempt to integrate the data properties, such as distributions and manifold structures to design compact hash codes with improved accuracy. Representative unsupervised methods include spectral hashing [32], graph hashing [28], manifold hashing [60], iterative quantization hashing [61], kernalized locality sensitive hashing [21], [41], isotropic hashing [62], angular quantization hashing [63], and spherical hashing [64]. Among these approaches, spectral hashing explores the data distribution and graph hashing utilizes the underlying manifold structure of data captured by a graph representation. In addition, supervised learning paradigms ranging from kernel learning to metric learning to deep learning have been exploited to learn binary codes, and many supervised hashing methods have been proposed recently [19], [22], [65]–[68]. Finally, semisupervised learning paradigm was employed to design hash functions by using both labeled and unlabeled data. For instance, Wang *et al.* proposed a regularized objective to achieve accurate yet balanced hash codes to avoid overfitting [14]. In [69] and [70], the authors proposed to exploit the metric learning and LSH to achieve fast-similarity-based search. Since the labeled data are used for deriving optimal metric distance while the hash function design uses no supervision, the proposed hashing technique can be regarded as a semisupervised approach.

C. Pointwise, Pairwise, Tripletwise, and Listwise

Based on the level of supervision, the supervised or semisupervised hashing methods can be further grouped into several subcategories, including pointwise, pairwise, tripletwise, and listwise approaches. For example, a few existing approaches utilize the instance level semantic attributes or labels to design the hash functions [18], [71], [72]. Additionally, learning methods based on pairwise supervision have been extensively studied, and many hashing techniques have been proposed [14], [19], [22], [65], [66], [69], [70], [73]. As demonstrated in Fig. 4(a), the pair $(\mathbf{x}_2, \mathbf{x}_3)$ contains similar points and the other two pairs $(\mathbf{x}_1, \mathbf{x}_2)$ and $(\mathbf{x}_1, \mathbf{x}_3)$ contain dissimilar points. Such relations are considered in the learning procedure to preserve the pairwise label information in the learned Hamming space. Since the ranking information is not fully utilized, the performance of pairwise-supervision-based methods could be suboptimal for nearest neighbor search. More recently, a triplet ranking that encodes the pairwise proximity comparison among three data points is

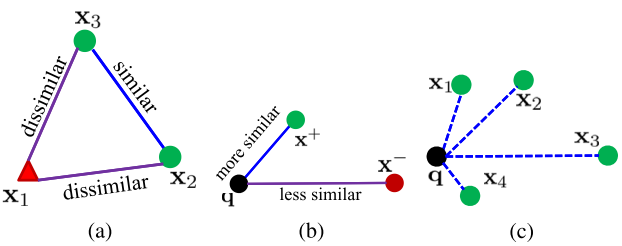

Fig. 4. ***Illustration of different levels of supervised information: (a) pairwise labels; (b) a triplet*** $\mathrm{sim}(\mathrm{q}, \mathrm{x}^+) > \mathrm{sim}(\mathrm{q}, \mathrm{x}^-)$***; and (c) a distance-based rank list*** $(\mathrm{x}_4, \mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3)$ ***to a query point*** q.

exploited to design hash codes [65], [74], [75]. As shown in Fig. 4(b), the point $\mathbf{x}^+$ is more similar to the query point $\mathbf{q}$ than the point $\mathbf{x}^-$. Such a triplet ranking information, i.e., $\text{sim}(\mathbf{q}, \mathbf{x}^+) > \text{sim}(\mathbf{q}, \mathbf{x}^-)$ is expected to be encoded in the learned binary hash codes. Finally, the listwise information indicates the rank order of a set of points with respect to the query point. In Fig. 4(c), for the query point $\mathbf{q}$, the rank list $(\mathbf{x}_4, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ shows the ranking order of their similarities to the query point $\mathbf{q}$, where $\mathbf{x}_4$ is the nearest point and $\mathbf{x}_3$ is the farthest one. By converting rank lists to a triplet tensor matrix, listwise hashing is designed to preserve the ranking in the Hamming space [76].

D. Linear Versus Nonlinear

Based on the form of function $f(\cdot)$ in (4), hash functions can also be categorized in two groups: linear and nonlinear. Due to their computational efficiency, linear functions tend to be more popular, which include random-projection-based LSH methods. The learning-based methods derive optimal projections by optimizing different types of objectives. For instance, PCA hashing performs principal component analysis on the data to derive large variance projections [63], [77], [78], as shown in Fig. 5(a). In the same league, supervised methods have used linear discriminant analysis to design more discriminative hash codes [79], [80]. Semisupervised hashing methods estimate the projections that have minimum empirical loss on pairwise labels while partitioning the unlabeled data in a balanced way [14]. Techniques that use variance of the projections as the underlying objective also tend to use orthogonality constraints for computational ease. However, these constraints lead to a significant drawback since the variance for most real-world data decays rapidly with most of the variance contained only in top few directions. Thus, in order to generate more bits in the code, one is forced to use progressively low-variance directions due to orthogonality constraints. The binary codes derived from these low-variance projections tend to have significantly lower performance. Two types of solutions based on relaxation of the orthogonality constraints or random/learned rotation of the data have been proposed in the literature to address these issues [14], [61]. Isotropic hashing is proposed to derive projections with equal variances and is shown to be superior to anisotropic-variances-based projections [62]. Instead of performing one-shot learning, sequential projection learning derives correlated projections with the goal of correcting errors from previous hash bits [25]. Finally, to reduce the computational complexity of full projection, circulant binary embedding was recently proposed to significantly speed up the encoding process using the circulant convolution [81].

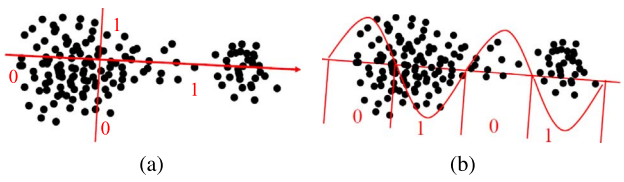

Fig. 5. *Comparison of hash bits generated using (a) PCA hashing and (b) spectral hashing.*

Despite its simplicity, linear hashing often suffers from insufficient discriminative power. Thus, nonlinear methods have been developed to override such limitations. For instance, spectral hashing first extracts the principal projections of the data, and then partitions the projected data by a sinusoidal function (nonlinear) with a specific angular frequency. Essentially, it prefers to partition projections with large spread and small spatial frequency such that the large variance projections can be reused. As illustrated in Fig. 5(b), the fist principal component can be reused in spectral hashing to divide the data into four parts while being encoded with only one bit. In addition, shift-invariant kernel-based hashing chooses $f(\cdot)$ to be a shifted cosine function and samples the projection vector in the same way as standard LSH does [33]. Another category of nonlinear hashing techniques employs kernel functions [21], [22], [28], [82]. Anchor graph hashing proposed by Liu *et al.* [28] uses a kernel function to measure similarity of each points with a set of anchors resulting in nonlinear hashing. Kernerlized LSH uses a sparse set of data points to compute a kernel matrix and preform random projection in the kernel space to compute binary codes [21]. Based on similar representation of kernel metric, Kulis and Darrell propose learning of hash functions by explicitly minimizing the reconstruction error in the kernel space and Hamming space [27]. Liu *et al.* apply kernel representation but optimize the hash functions by exploring the equivalence between optimizing the code inner products and the Hamming distances to achieve scale invariance [22].

E. Single-Shot Learning Versus Multiple-Shot Learning

For learning-based hashing methods, one first formulates an objective function reflecting desired characteristics of the hash codes. In a single-shot learning paradigm, the optimal solution is derived by optimizing the objective function in a single shot. In such a learning-to-hash framework, the K hash functions are learned simultaneously. In contrast, the multiple-shot learning procedure considers a global objective, but optimizes a hash function considering the bias generated by the previous hash functions. Such a procedure sequentially trains hash functions one bit at a time [25], [83], [84]. The multiple-shot hash function learning is often used in supervised or semisupervised settings since the given label information can be used to assess the quality of the hash functions learned in previous steps. For instance, the sequential-projection-based hashing aims to incorporate

the bit correlations by iteratively updating the pairwise label matrix, where higher weights are imposed on point pairs violated by the previous hash functions [25]. In the complementary projection learning approach [84], Jin *et al.* present a sequential learning procedure to obtain a series of hash functions that cross the sparse data region, as well as generate balanced hash buckets. Column generation hashing learns the best hash function during each iteration and updates the weights of hash functions accordingly. Other interesting learning ideas include two-step learning methods which treat hash bit learning and hash function learning separately [85], [86].

F. Nonweighted Versus Weighted Hashing

Given the Hamming embedding defined in (2), traditional-hashing-based indexing schemes map the original data into a nonweighted Hamming space, where each bit contributes equally. Given such a mapping, the Hamming distance is calculated by counting the number of different bits. However, it is easy to observe that different bits often behave differently [14], [32]. In general, for linear-projection-based hashing methods, the binary code generated from large variance projection tends to perform better due to its superior discriminative power. Hence, to improve discrimination among hash codes, techniques were designed to learn a weighted hamming embedding as

$$H : \mathcal{X} \rightarrow \{\alpha_1 h_1(\mathbf{x}), \ldots, \alpha_K h_K(\mathbf{x})\}. \tag{13}$$

Hence, the conventional hamming distance is replaced by a weighted version as

$$d_{\mathcal{WH}} = \sum_{k=1}^{K} \alpha_k \left| h_k(\mathbf{x}_i) - h_k(\mathbf{x}_j) \right|. \tag{14}$$

One of the representative approaches is boosted similarity sensitive coding (BSSC) [18]. By learning the hash functions and the corresponding weights $\{\alpha_1, \ldots, \alpha_k\}$ jointly, the objective is to lower the collision probability of nonneighbor pair $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{C}$ while improving the collision probability of neighboring pair $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{M}$. If one treats each hash function as a decision stump, the straightforward way of learning the weights is to directly apply adaptive boosting algorithm [87], as described in [18]. In [88], a boosting-style method called BoostMAP is proposed to map data points to weighted binary vectors that can leverage both metric and semantic similarity measures. Other weighted hashing methods include designing specific bit-level weighting schemes to improve the search accuracy [74], [89]–[92]. In addition, a recent work about designing a unified bit selection framework can be regarded as a special case of weighted hashing approach, where the weights of hash bits are binary [93]. Another effective hash code ranking method is the query-sensitive hashing, which explores the raw feature of the query sample and learns query-specific weights of hash bits to achieve accurate ϵ-nearest neighbor search [94].

IV. Methodology Review and Analysis

In this section, we will focus on a review of several representative hashing methods that explore various machine learning techniques to design data-specific indexing schemes. The techniques consist of unsupervised, semisupervised, as well as supervised approaches, including spectral hashing, anchor graph hashing, angular quantization, binary-reconstructive-embedding-based hashing, metric-learning-based hashing, semisupervised hashing, column generation hashing, and ranking supervised hashing. Table 2 summarizes the surveyed hashing techniques, as well as their technical merits.

Table 2 Summary of the Surveyed Hashing Techniques

Method	Hash Function/Objective Function	Parameters	Learning Paradigm	Supervision
Spectral Hashing	$\mathrm{sgn}(\cos(\alpha \mathbf{w}^\top \mathbf{x}))$	$\mathbf{w}, \alpha$	unsupervised	NA
Anchor Graph Hashing	$\mathrm{sgn}(\mathbf{w}^\top \mathbf{x})$	$\mathbf{w}$	unsupervised	NA
Angular Quantization	$(\mathbf{b}, \mathbf{w}) = \arg\max \sum_i \frac{\mathbf{b}_i^\top}{\|\mathbf{b}_i\|_2} \mathbf{w}^\top \mathbf{x}_i$	$\mathbf{b}$, $\mathbf{w}$	unsupervised	NA
Binary Reconstructive Embedding	$\mathrm{sgn}(\mathbf{w}^\top K(\mathbf{x}))$	$\mathbf{w}$	unsupervised/supervised	pairwise distance
Kernel-Based Supervised Hashing	$\mathrm{sgn}(\mathbf{w}^\top K(\mathbf{x}))$	$\mathbf{w}$	supervised	pairwise similarity
Metric Learning Hashing	$\mathrm{sgn}(\mathbf{w}^\top \mathbf{G}^\top \mathbf{x})$	$\mathbf{G}, \mathbf{w}$	supervised	pairwise similarity
Semi-Supervised Hashing	$\mathrm{sgn}(\mathbf{w}^\top \mathbf{x})$	$\mathbf{w}$	semi-supervised	pairwise similarity
Column generation hashing	$\mathrm{sgn}(\mathbf{w}^\top \mathbf{x} + b)$	$\mathbf{w}$	supervised	triplet
Listwise hashing	$\mathrm{sgn}(\mathbf{w}^\top \mathbf{x} + b)$	$\mathbf{w}$	supervised	ranking list
Circulant binary embedding	$\mathrm{sgn}(circ(\mathbf{r}) \cdot \mathbf{x})$	$\mathbf{r}$	unsupervised/supervised	pairwise similarity

Note that this section mainly focuses on describing the intuition and formulation of each method, as well as discussing their pros and cons. The performance of each individual method highly depends on practical settings, including learning parameters and data set itself. In general, the nonlinear and supervised techniques tend to generate better performance than linear and unsupervised methods, while being more computationally costly [14], [19], [21], [22], [27].

A. Spectral Hashing

In the formulation of spectral hashing, the desired properties include keeping neighbors in input space as neighbors in the hamming space and requiring the codes to be balanced and uncorrelated [32]. Hence, the objective of spectral hashing is formulated as

$$\begin{aligned} \min \sum_{ij} \frac{1}{2} A_{ij} \|\mathbf{y}_i - \mathbf{y}_j\|^2 &= \frac{1}{2} \mathrm{tr}(\mathbf{Y}^\top \mathbf{L} \mathbf{Y}) \\ \text{subject to:} \quad & \mathbf{Y} \in \{-1, 1\}^{N \times K} \\ & \mathbf{1}^\top \mathbf{y}_{k\cdot} = 0, \qquad k = 1, \ldots, K \\ & \mathbf{Y}^\top \mathbf{Y} = n \mathbf{I}_{K \times K} \end{aligned} \tag{15}$$

where $\mathbf{A} = \{A_{ij}\}_{i,j=1}^{N}$ is a pairwise similarity matrix and the Laplacian matrix is calculated as $\mathbf{L} = \mathrm{diag}(\mathbf{A}\mathbf{1}) - \mathbf{A}$. The constraint $\mathbf{1}^\top \mathbf{y}_k = 0$ ensures that the hash bit $\mathbf{y}_k$ reaches a balanced partitioning of the data and the constraint $\mathbf{Y}^\top \mathbf{Y} = n \mathbf{I}_{K \times K}$ imposes orthogonality between hash bits to minimize the redundancy.

The direct solution for the above optimization is nontrivial for even a single bit since it is essentially a balanced graph partition problem, which is NP hard. The orthogonality constraints for K-bit balanced partitioning make the above problem even harder. Motivated by the well-known spectral graph analysis [95], Fowlkes *et al.* suggest to minimize the cost function with relaxed constraints. In particular, with the assumption of uniform data distribution, the spectral solution can be efficiently computed using 1-D Laplacian eigenfunctions [32]. The final solution for spectral hashing equals to apply a sinusoidal function with precomputed angular frequency to partition data along PCA directions. Note that the projections are computed using data but learned in an unsupervised manner. As most of the orthogonal-projection-based hashing methods, spectral hashing suffers from the low-quality binary coding using low-variance projections. Hence, a "kernel trick" is used to alleviate the degraded performance when using long hash bits [96]. Moreover, the assumption of uniform data distribution usually hardly holds for real-world data.

B. Anchor Graph Hashing

Following the similar objective as spectral hashing, anchor graph hashing was designed to solve the problem from a different perspective without the assumption of uniform distribution [28]. Note that the critical bottleneck for solving (15) is the cost of building a pairwise similarity graph $\mathbf{A}$, the computation of associated graph Laplacian, as well as solving the corresponding eigensystem, which at least has a quadratic complexity. The key idea is to use a small set of $M (M \ll N)$ anchor points to approximate the graph structure represented by the matrix $\mathbf{A}$ such that the similarity between any pair of points can be approximated using point-to-anchor similarities [97]. In particular, the truncated point-to-anchor similarity $\mathbf{Z} \in \mathbb{R}^{N \times M}$ gives the similarities between N database points to the M anchor points. Thus, the approximated similarity matrix $\hat{\mathbf{A}}$ can be calculated as $\hat{\mathbf{A}} = \mathbf{Z}\mathbf{\Lambda}\mathbf{Z}^\top$, where $\mathbf{\Lambda} = \mathrm{diag}(\mathbf{Z}\mathbf{1})$ is the degree matrix of the anchor graph $\mathbf{Z}$. Based on such an approximation, instead of solving the eigensystem of the matrix $\hat{\mathbf{A}} = \mathbf{Z}\mathbf{\Lambda}\mathbf{Z}^\top$, one can alternatively solve a much smaller eigensystem with an $M \times M$ matrix $\mathbf{\Lambda}^{1/2}\mathbf{Z}^\top\mathbf{Z}\mathbf{\Lambda}^{1/2}$. The final binary codes can be obtained through calculating the sign function over a spectral embedding as

$$\mathbf{Y} = \mathrm{sgn}\left(\mathbf{Z}\mathbf{\Lambda}^{\frac{1}{2}}\mathbf{V}\mathbf{\Sigma}^{\frac{1}{2}}\right). \tag{16}$$

Here we have the matrices $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_k, \ldots, \mathbf{v}_K] \in \mathbb{R}^{M \times K}$ and $\mathbf{\Sigma} = \mathrm{diag}(\sigma_1, \ldots, \sigma_k, \ldots, \sigma_K) \in \mathbb{R}^{K \times K}$, where $\{\mathbf{v}_k, \sigma_k\}$ are the eigenvector–eigenvalue pairs [28]. Fig. 6 shows the two-bit partitioning on a synthetic data with nonlinear structure using different hashing methods, including spectral hashing, exact graph hashing, and anchor graph hashing. Note that since spectral hashing computes two smoothest pseudograph Laplacian eigenfunctions instead of performing real spectral embedding, it cannot handle such type of nonlinear data structures. The exact graph hashing method first constructs an exact neighborhood graph, e.g., k-NN graph, and then performs partitioning with spectral techniques to solve the optimization problem in (15). The anchor graph hashing archives a good separation (by the first bit) of the nonlinear manifold and balancing partitioning, and even performs better than the exact graph hashing, which loses the property of balancing partitioning for the second bit. The anchor graph hashing approach was recently further improved by leveraging a discrete optimization technique to directly solve binary hash codes without any relaxation [98].

C. Angular-Quantization-Based Hashing

Since similarity is often measured by the cosine of the angle between pairs of samples, angular quantization is thus proposed to map nonnegative feature vectors onto a vertex of the binary hypercube with the smallest angle [63]. In such a

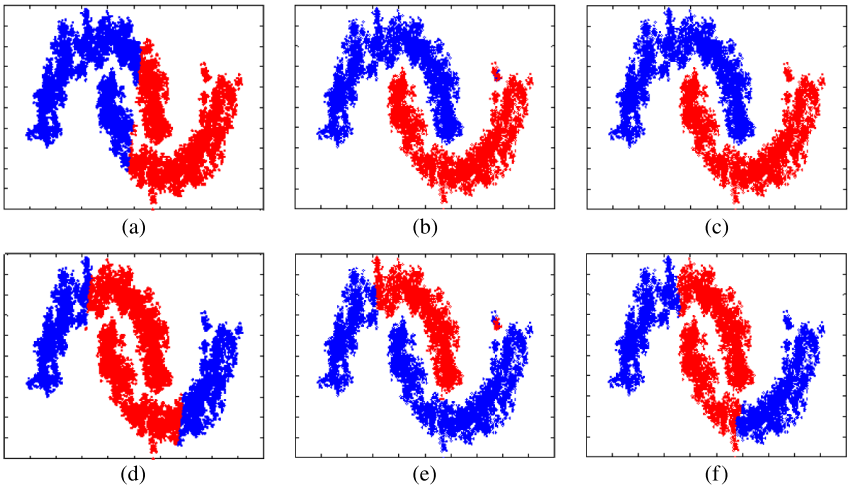

Fig. 6. *Comparison of partitioning a two-moon data by the first two hash bits using different methods: (a) the first bit using spectral hashing; (b) the first bit using exact graph hashing; (c) the first bit using anchor graph hashing; (d) the second bit using spectral hashing; (e) the second bit using exact graph hashing; and (f) the second bit using anchor graph hashing.*

setting, the vertices of the hypercube are treated as quantization landmarks that grow exponentially with the data dimensionality D. As shown in Fig. 7, the nearest binary vertex $\mathbf{b}$ in a hypercube to the data point $\mathbf{x}$ is given by

$$\mathbf{b}^* = \arg\max_{\mathbf{b}} \frac{\mathbf{b}^\top \mathbf{x}}{\|\mathbf{b}\|_2} \quad \text{subject to:} \quad \mathbf{b} \in \{0,1\}^K. \tag{17}$$

Although it is an integer programming problem, its global maximum can be found with a complexity of $\mathcal{O}(D \log D)$. The optimal binary vertices will be used as the binary hash codes for data points as $\mathbf{y} = \mathbf{b}^*$. Based on this angular quantization framework, a data-dependent extension is designed to learn a rotation matrix $\mathbf{R} \in \mathbb{R}^{D\times D}$ to align the projected data $\mathbf{R}^\top \mathbf{x}$ to the binary vertices without changing the similarity between point pairs. The objective is formulated as follows:

$$\left(\mathbf{b}_i^*, \mathbf{R}^*\right) = \arg\max_{\mathbf{b}_i, \mathbf{R}} \sum_i \frac{\mathbf{b}_i^\top}{\|\mathbf{b}_i\|_2} \mathbf{R}^\top \mathbf{x}_i \quad \text{subject to:} \quad \mathbf{b} \in \{0,1\}^K \quad \mathbf{R}^\top \mathbf{R} = \mathbf{I}_{D\times D}. \tag{18}$$

Note that the above formulation still generates a D-bit binary code for each data point, while compact codes are often desired in many real-world applications [14]. To generate a $K =$ bit code, a projection matrix $\mathbf{S} \in \mathbb{R}^{D\times K}$ with orthogonal columns can be used to replace the rotation matrix $\mathbf{R}$ in the above objective with additional normalization, as discussed in [63]. Finally, the optimal binary codes and the projection/rotation matrix are learned using an alternating optimization scheme.

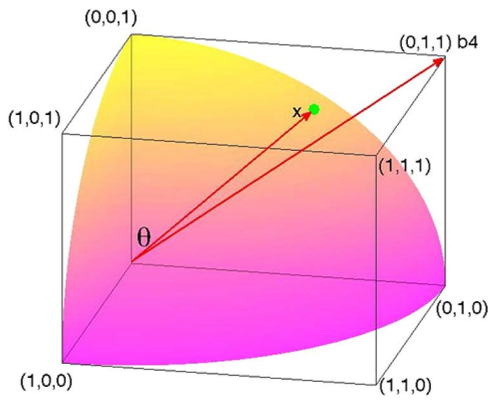

Fig. 7. *Illustration of angular-quantization-based hashing method [63]. The binary code of a data point* x *is assigned as the nearest binary vertex in the hypercube, which is* $\mathrm{b}_4 = [0\ 1\ 1]^\top$ *in the illustrated example [63].*

D. Binary Reconstructive Embedding

Instead of using data-independent random projections as in LSH or principal components as in SH, Kulis and Darrell [27] proposed data-dependent and bit-correlated hash functions as

$$h_k(\mathbf{x}) = \operatorname{sgn}\left(\sum_{q=1}^{s} \mathbf{W}_{kq} \kappa(\mathbf{x}_{kq}, \mathbf{x})\right). \tag{19}$$

The sample set $\{\mathbf{x}_{kq}\}$, $q = 1, \ldots, s$, is the training data for learning hash function h_k and $\kappa(\cdot)$ is a kernel function, and $\mathbf{W}$ is a weight matrix.

Based on the above formulation, a method called binary reconstructive embedding (BRE) was designed to minimize a cost function measuring the difference between the metric and reconstructed distance in hamming space. The Euclidean metric $d_{\mathcal{M}}$ and the binary reconstruction distance $d_{\mathcal{R}}$ are defined as

$$d_{\mathcal{M}}(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{2}\|\mathbf{x}_i - \mathbf{x}_j\|^2$$
$$d_{\mathcal{R}}(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{K}\sum_{k=1}^{K}\big(h_k(\mathbf{x}_i) - h_k(\mathbf{x}_j)\big)^2. \quad (20)$$

The objective is to minimize the following reconstruction error to derive the optimal $\mathbf{W}$:

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} \sum_{(\mathbf{x}_i, \mathbf{x}_j)\in\mathcal{N}} \big[d_{\mathcal{M}}(\mathbf{x}_i, \mathbf{x}_j) - d_{\mathcal{R}}(\mathbf{x}_i, \mathbf{x}_j)\big]^2 \quad (21)$$

where the set of sample pairs $\mathcal{N}$ is the training data. Optimizing the above objective function is difficult due to the nondifferentiability of $\mathrm{sgn}(\cdot)$ function. Instead, a coordinate-descent algorithm was applied to iteratively update the hash functions to a local optimum. This hashing method can be easily extended to a supervised scenario by setting pairs with the same labels to have zero distance and pairs with different labels to have a large distance. However, since the binary reconstruction distance $d_{\mathcal{R}}$ is bounded in $[0, 1]$ while the metric distance $d_{\mathcal{M}}$ has no upper bound, the minimization problem in (21) is only meaningful when input data are appropriately normalized. In practice, the original data point $\mathbf{x}$ is often mapped to a hypersphere with unit length so that $0 \leqslant d_{\mathcal{M}} \leqslant 1$. This normalization removes the scale of data points, which is often not negligible for practical applications of nearest neighbor search. In addition, Hamming-distance-based objective is hard to optimize due to its nonconvex and nonsmooth properties.

Therefore, Liu *et al.* proposed to take advantage of the equivalence between code inner products and Hamming distances to design kernel-based hash functions and learn them under a supervised manner [22]. The objective of the proposed approach, namely kernel-based supervised hashing (KSH), is to ensure the inner products of hash codes consistent with the supervised pairwise similarities. The strategy of optimizing hash code inner products in KSH rather than Hamming distances like what is done in BRE pays off nicely and leads to major performance gains in similarity-based retrieval tasks, which has been consistently confirmed through extensive experiments reported in [22] and recent studies [99].

E. Metric-Learning-Based Hashing

The key idea for metric-learning-based hashing method is to learn a parameterized Mahalanobis metric using pairwise label information. Such learned metrics are then employed to the standard-random-projection-based hash functions [19]. The goal is to preserve the pairwise relationship in the binary code space, where similar data pairs are more likely to collide in the same hash buck and dissimilar pairs are less likely to share the same hash codes, as illustrated in Fig. 8.

The parameterized inner product is defined as

$$\mathrm{sim}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\top \mathbf{M} \mathbf{x}_j$$

where $\mathbf{M}$ is a positive–definite $d \times d$ matrix to be learned from the labeled data. Note that this similarity measure corresponds to the parameterized squared Mahalanobis distance $d_{\mathbf{M}}$. Assume that $\mathbf{M}$ can be factorized as $\mathbf{M} = \mathbf{G}^\top \mathbf{G}$. Then, the parameterized squared Mahalanobis distance can be written as

$$d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i - \mathbf{x}_j)^\top \mathbf{M} (\mathbf{x}_i - \mathbf{x}_j)$$
$$= (\mathbf{G}\mathbf{x}_i - \mathbf{G}\mathbf{x}_i)^\top (\mathbf{G}\mathbf{x}_i - \mathbf{G}\mathbf{x}_i). \quad (22)$$

Based on the above equation, the distance $d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j)$ can be interpreted as the Euclidian distance between the projected data points $\mathbf{G}\mathbf{x}_i$ and $\mathbf{G}\mathbf{x}_j$. Note that the matrix $\mathbf{M}$ can be learned through various metric learning method such as information-theoretic metric learning [100]. To accommodate the learned distance metric, the randomized hash function is given as

$$h_k(\mathbf{x}) = \mathrm{sgn}(\mathbf{w}_k \mathbf{G}^\top \mathbf{x}). \quad (23)$$

It is easy to see that the above hash function generates the hash codes which preserve the parameterized similarity

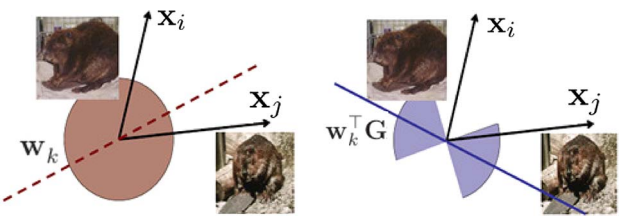

Fig. 8. *Illustration of the hashing-method-based on metric learning. The left shows the partitioning using standard LSH method and the right shows the partitioning of the metric-learning-based LSH method (modified from the original figure in [19]).*

measure in the Hamming space. Fig. 8 demonstrates the difference between standard-random-projection-based LSH and the metric-learning-based LSH, where it is easy to see that the learned metric help assign the same hash bit to the similar sample pairs. Accordingly, the collision probability is given as

$$\Pr[h_k(\mathbf{x}_i) = h_k(\mathbf{x}_i)] = 1 - \frac{1}{\pi}\cos^{-1}\frac{\mathbf{x}_i^\top \mathbf{G}^\top \mathbf{G}\mathbf{x}_j}{\|\mathbf{G}\mathbf{x}_i\|\|\mathbf{G}\mathbf{x}_j\|}. \quad (24)$$

Realizing that the pairwise constraints often come to be available incrementally, Jain *et al.* exploit an efficient online LSH with gradually learned distance metrics [70].

F. Semisupervised Hashing

Supervised hashing techniques have been shown to be superior to unsupervised approaches since they leverage the supervision information to design task-specific hash codes. However, for a typical setting of large-scale problem, the human annotation process is often costly and the labels can be noisy and sparse, which could easily lead to overfitting.

Considering a small set of pairswise labels and a large amount of unlabled data, semisupervised hashing aims in designing hash functions with minimum empirical loss while maintaining maximum entropy over the entire data set. Specifically, assume the pairwise labels are given as two type of sets $\mathcal{M}$ and $\mathcal{C}$. A pair of data point $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{M}$ indicates that $\mathbf{x}_i$ and $\mathbf{x}_j$ are similar and $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{C}$ means that $\mathbf{x}_i$ and $\mathbf{x}_j$. Hence, the empirical accuracy on the labeled data for a family of hash functions $\mathbf{H} = [h_1, \ldots, h_K]$ is given as

$$J(\mathbf{H}) = \sum_k \left[\sum_{(\mathbf{x}_i,\mathbf{x}_j)\in\mathcal{M}} h_k(\mathbf{x}_i)h_k(\mathbf{x}_j) - \sum_{(\mathbf{x}_i,\mathbf{x}_j)\in\mathcal{C}} h_k(\mathbf{x}_i)h_k(\mathbf{x}_j) \right]. \quad (25)$$

We define a matrix $\mathbf{S} \in \mathbb{R}^{l\times l}$ incorporating the pairwise labeled information from $\mathbf{X}_l$ as

$$S_{ij} = \begin{cases} 1 & : \ (\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{M} \\ -1 & : \ (\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{C} \\ 0 & : \ \text{otherwise.} \end{cases} \quad (26)$$

The above empirical accuracy can be written in a compact matrix form after dropping off the $\mathrm{sgn}(\cdot)$ function

$$J(\mathbf{H}) = \frac{1}{2}\mathbf{tr}\left(\mathbf{W}^\top \mathbf{X}_l \mathbf{S}\ \mathbf{W}^\top \mathbf{X}_l^\top\right). \quad (27)$$

However, only considering the empirical accuracy during the design of the hash function can lead to undesired results. As illustrated in Fig. 9(b), although such a hash bit partitions the data with zero error over the pairwise labeled data, it results in imbalanced separation of the unlabeled data, thus being less informative. Therefore, an information-theoretic regularization is suggested to maximize the entropy of each hash bit. After relaxation, the final objective is formed as

$$\mathbf{W}^* = \arg\max_{\mathbf{W}} \frac{1}{2}\mathbf{tr}\left(\mathbf{W}^\top \mathbf{X}_l \mathbf{S} \mathbf{X}_l^\top \mathbf{W}\right) + \frac{\eta}{2}\mathbf{tr}(\mathbf{W}^\top \mathbf{X}\mathbf{X}^\top \mathbf{W}) \quad (28)$$

where the first part represents the empirical accuracy and the second component encourages partitioning along large variance projections. The coefficient η weighs the contribution from these two components. The above objective can be solved using various optimization strategies, resulting in orthogonal or correlated binary codes, as described in [14] and [25]. Fig. 9 illustrates the comparison of one-bit linear partition using different learning paradigms, where the semisupervised method tends to produce balanced yet accurate data separation. Finally, Xu *et al.* employ similar semisupervised formulation to sequentially learn multiple complementary hash tables to further improve the performance [31].

G. Column Generation Hashing

Beyond pairwise relationship, complex supervision like ranking triplets and ranking lists has been exploited to learn hash functions with the property of ranking preserving. In many real applications such as image retrieval and recommendation system, it is often easier to receive the relative comparison instead of instance-wise or pairwise labels. For a general claim, such relative comparison information is given in a triplet form. Formally, a set of triplets are represented as

$$\mathcal{E} = \left\{ \left(\mathbf{q}_i, \mathbf{x}_i^+, \mathbf{x}_i^-\right) | \mathrm{sim}\left(\mathbf{q}_i, \mathbf{x}_i^+\right) > \mathrm{sim}\left(\mathbf{q}_i, \mathbf{x}_i^-\right) \right\}$$

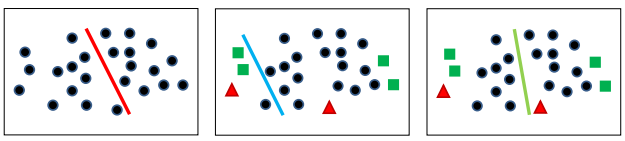

Fig. 9. *Illustration of one-bit partitioning of different linear-projection-based hashing methods: (a) unsupervised hashing; (b) supervised hashing; and (c) semisupervised hashing. The similar point pairs are indicated in the green rectangle shape, and the dissimilar point pairs are with a red triangle shape.*

where the function $(\cdot)$ could be an unknown similarity measure. Hence, the triplet $(\mathbf{q}_i, \mathbf{x}_i^+, \mathbf{x}_i^-)$ indicates that the sample point $\mathbf{x}_i^+$ is more semantically similar or closer to a query point $\mathbf{q}$ than the point $\mathbf{x}_i^-$, as demonstrated in Fig. 4(b).

As one of the representative methods falling into this category, column generation hashing explores the large-margin framework to leverage such type of proximity comparison information to design weighted hash functions [74]. In particular, the relative comparison information $\text{sim}(\mathbf{q}_i, \mathbf{x}_i^+) > \text{sim}(\mathbf{q}_i, \mathbf{x}_i^-)$ will be preserved in a weighted Hamming space as $d_{\mathcal{WH}}(\mathbf{q}_i, \mathbf{x}_i^+) < d_{\mathcal{WH}}(\mathbf{q}_i, \mathbf{x}_i^-)$, where $d_{\mathcal{WH}}$ is the weighted Hamming distance as defined in (14). To impose a large margin, the constraint $d_{\mathcal{WH}}(\mathbf{q}_i, \mathbf{x}_i^+) < d_{\mathcal{WH}}(\mathbf{q}_i, \mathbf{x}_i^-)$ should be satisfied as well as possible. Thus, a typical large-margin objective with ℓ_1-norm regularization can be formulated as

$$\begin{aligned} &\arg\min_{\mathbf{w},\xi} \sum_{i=1}^{|\mathcal{E}|} \xi_i + C\|\mathbf{w}\|_1 \\ &\text{subject to:} \quad \mathbf{w} \succeq \mathbf{0}, \boldsymbol{\xi} \succeq \mathbf{0}; \\ &\qquad\qquad d_{\mathcal{WH}}(\mathbf{q}_i, \mathbf{x}_i^-) - d_{\mathcal{WH}}(\mathbf{q}_i, \mathbf{x}_i^-) \geqslant 1 - \xi_i, \ \forall i \end{aligned} \tag{29}$$

where $\mathbf{w}$ is the random projection for computing the hash codes. To solve the above optimization problem, the authors proposed using the column generation technique to learn the hash function and the associated bit weights iteratively. For each iteration, the best hash function is generated and the weight vector is updated accordingly. In addition, different loss functions with other regularization terms such as ℓ_∞ are also suggested as alternatives in the above formulation.

H. Ranking Supervised Hashing

Different from other methods that explore the triplet relationship [65], [74], [75], the ranking supervised hashing method attempts to preserve the ranking order of a set of database points corresponding to the query point [76]. Assume that the training data set $\mathcal{X} = \{\mathbf{x}_n\}$ has N points with $\mathbf{x}_n \in \mathbb{R}^D$. In addition, a query set is given as $\mathcal{Q} = \{\mathbf{q}_m\}$, and $\mathbf{q}_m \in \mathbb{R}^D$, $m = 1, \ldots, M$. For any specific query point $\mathbf{q}_m$, we can derive a ranking list over $\mathcal{X}$, which can be written as a vector as $r(\mathbf{q}_m, \mathcal{X}) = (r_1^m, \ldots, r_n^m, \ldots, r_N^m)$. Each element r_n^m falls into the integer range $[1, N]$ and no two elements share the same value for the exact ranking case. If $r_i^m < r_j^m$ $(i, j = 1, \ldots, N)$, it indicates that sample $\mathbf{x}_i$ has higher rank than $\mathbf{x}_j$, which means $\mathbf{x}_i$ is more relevant or similar to $\mathbf{q}_m$ than $\mathbf{x}_j$. To represent such a discrete ranking list, a ranking triplet matrix $\mathbf{S} \in \mathbb{R}^{N \times N}$ is defined as

$$S(\mathbf{q}_m; \mathbf{x}_i, \mathbf{x}_j) = \begin{cases} 1 & : \ r_i^q < r_j^q \\ -1 & : \ r_i^q > r_j^q \\ 0 & : \ r_i^q = r_j^q. \end{cases} \tag{30}$$

Hence, for a set of query points $\mathcal{Q} = \{\mathbf{q}_m\}$, we can derive a triplet tensor, i.e., a set of triplet matrices

$$\mathbf{S} = \{\mathbf{S}_{(\mathbf{q}_m)}\} \in \mathbb{R}^{M \times N \times N}.$$

In particular, the element of the triplet tensor is defined as $\mathbf{S}_{mij} = \mathbf{S}_{(\mathbf{q}_m)}(i, j) = S(\mathbf{q}_m; \mathbf{x}_i, \mathbf{x}_j)$, $m = 1, \ldots, M$, $i, j = 1, \ldots, N$. The objective is to preserve the ranking lists in the mapped Hamming space. In other words, if $S(\mathbf{q}_m; \mathbf{x}_i, \mathbf{x}_j) = 1$, we tend to ensure $d_{\mathcal{H}}(\mathbf{q}_m, \mathbf{x}_i) < d_{\mathcal{H}}(\mathbf{q}_m, \mathbf{x}_i)$, otherwise $d_{\mathcal{H}}(\mathbf{q}_m, \mathbf{x}_i) > d_{\mathcal{H}}(\mathbf{q}_m, \mathbf{x}_i)$. Assume the hash code has the value as $\{-1, 1\}$, such ranking order is equivalent to the similarity measurement using the inner products of the binary codes, i.e.,

$$d_{\mathcal{H}}(\mathbf{q}_m, \mathbf{x}_i) < d_{\mathcal{H}}(\mathbf{q}_m, \mathbf{x}_i) \Leftrightarrow H(\mathbf{q}_m)^\top H(\mathbf{x}_i) > H(\mathbf{q}_m)^\top H(\mathbf{x}_j).$$

Then, the empirical loss function $L_{\mathcal{H}}$ over the ranking list can be represented as

$$L_{\mathcal{H}} = -\sum_m \sum_{i,j} H(\mathbf{q}_m)^\top \big[H(\mathbf{x}_i) - H(\mathbf{x}_j)\big] S_{mij}.$$

Assume that we utilize linear hash functions, then the final objective is formed as the following constrained quadratic problem:

$$\begin{aligned} &\mathbf{W}^* = \arg\max_{\mathbf{W}} L_H = \arg\max_{\mathbf{W}} \text{tr}(\mathbf{W}\mathbf{W}^\top \mathbf{B}) \\ &\text{subject to: } \mathbf{W}^\top \mathbf{W} = \mathbf{I} \end{aligned} \tag{31}$$

where the constant matrix $\mathbf{B}$ is computed as $\mathbf{B} = \sum_m \mathbf{p}_m \mathbf{q}_m^\top$ with $\mathbf{p}_m = \sum_{i,j} [\mathbf{x}_i - \mathbf{x}_j] S_{mij}$. The orthogonality constraint is utilized to minimize the redundancy between different hash bits. Intuitively, the above formulation is to preserve the ranking list in the Hamming space, as shown in the conceptual diagram in Fig. 10. The augmented Lagrangian multiplier method was introduced to derive feasible solutions for the above constrained problem, as discussed in [76].

I. Circulant Binary Embedding

Realizing that most of the current hashing techniques rely on linear projections, which could suffer from very high computational and storage costs for high-dimensional data, circulant binary embedding was recently developed to handle such a challenge using the circulant projection [81]. Briefly, given a vector $\mathbf{r} = \{r_0, \ldots, r_{d-1}\}$, we can generate its corresponding circulant matrix $\mathbf{R} = \text{circ}(\mathbf{r})$

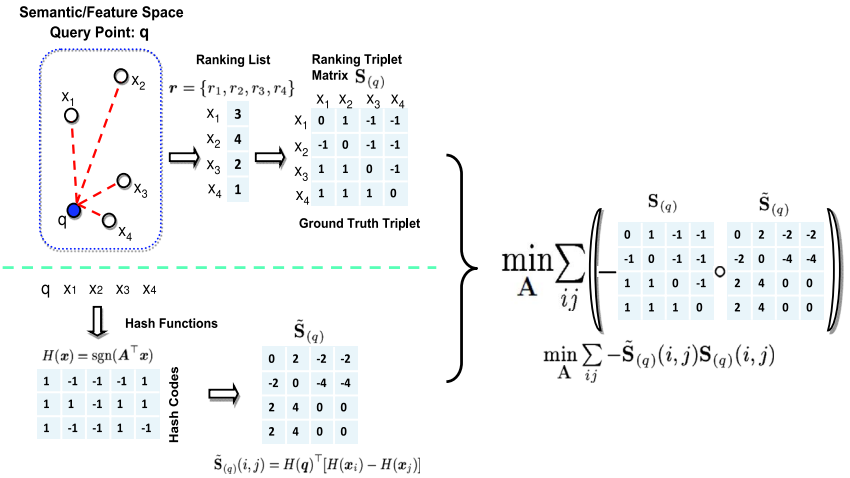

Fig. 10. *Conceptual diagram of the rank supervised hashing method. The top left component demonstrates the procedure of deriving ground truth ranking list* r *using the semantic relevance or feature similarity/distance, and then converting it to a triplet matrix* $\mathbf{S}_{(q)}$ *for a given query* q*. The bottom left component describes the estimation of relaxed ranking triplet matrix* $\tilde{\mathbf{S}}_{(q)}$ *from the binary hash codes. The right component shows the objective of minimizing the inconsistency between the two ranking triplet matrices.*

[101]. Therefore, the binary embedding with the circulant projection is defined as

$$h(\mathbf{x}) = \text{sgn}(\mathbf{R}\mathbf{x}) = \text{sgn}(\text{circ}(\mathbf{r}) \cdot \mathbf{x}). \tag{32}$$

Since the circulant projection $\text{circ}(\mathbf{r})\mathbf{x}$ is equivalent to circular convolution $\mathbf{r} \circledast \mathbf{x}$, the computation of linear projection can be eventually realized using fast Fourier transform as

$$\text{circ}(\mathbf{r})\mathbf{x} = \mathbf{r} \circledast \mathbf{x} = \mathcal{F}^{-1}(\mathcal{F}(\mathbf{r}) \circ \mathcal{F}(\mathbf{x})). \tag{33}$$

Thus, the time complexity is reduced from d^2 to $d \log d$. Finally, one could randomly select the circulant vector $\mathbf{r}$ or design specific ones using supervised learning methods.

V. Deep Learning for Hashing

During the past decade (since around 2006), deep learning [102], also known as deep neural networks, has drawn increasing attention and research efforts in a variety of artificial intelligence areas including speech recognition, computer vision, machine learning, text mining, etc. Since one main purpose of deep learning is to learn robust and powerful feature representations for complex data, it is very natural to leverage deep learning for exploring compact hash codes which can be regarded as binary representations of data. In this section, we briefly introduce several recently proposed hashing methods that employ deep learning. In Table 3, we compare eight deep-learning-based hashing methods in terms of four key characteristics that can be used to differentiate the approaches.

The earliest work in deep-learning-based hashing may be semantic hashing [103]. This method builds a deep generative model to discover hidden binary units (i.e., latent topic features) which can model input text data (i.e., word-count vectors). Such a deep model is made as a stack of restricted Boltzmann machines (RBMs) [104]. After learning a multilayer RBM through pretraining and fine tuning on a collection of documents, the hash code of any document is acquired by simply thresholding the output of the deepest layer. Such hash codes provided by the deep RBM were shown to preserve semantically similar relationships among input documents into the code space, in which each hash code (or hash key) is used as a memory address to locate corresponding documents. In this way, semantically similar documents are mapped to adjacent memory addresses, thereby enabling efficient search via hash table lookup. To enhance the performance of deep RBMs, a supervised version was proposed in [66], which borrows the idea of nonlinear neighborhood component analysis (NCA) embedding [105]. The supervised information stems from given neighbor/nonneighbor relationships between training examples. Then, the objective function of NCA is optimized on top of a deep RBM, making the deep RBM yield discriminative hash codes. Note that supervised deep RBMs can be applied to broad data domains other than text data. In [66], supervised deep RBMs using a Gaussian distribution to model visible units in the first layer were successfully applied to handle massive image data.

A recent work named sparse similarity-preserving hashing [99] tried to address the low recall issue pertaining to relatively long hash codes, which affect

Table 3 Characteristics of Eight Recently Proposed Deep-Learning-Based Hashing Methods

Deep Learning based Hashing Methods	Data Domain	Learning Paradigm	Learning Features?	Hierarchy of Deep Neural Networks
Semantic Hashing [103]	text	unsupervised	no	4
Restricted Boltzmann Machine [66]	text and image	supervised	no	4 and 5
Tailored Feed-Forward Neural Network [99]	text and image	supervised	no	6
Deep Hashing [106]	image	unsupervised	no	3
Supervised Deep Hashing [106]	image	supervised	no	3
Convolutional Neural Network Hashing [107]	image	supervised	yes	5
Deep Semantic Ranking Hashing [108]	image	supervised	yes	8
Deep Neural Network Hashing [109]	image	supervised	yes	10

most of previous hashing techniques. The idea is enforcing sparsity into the hash codes to be learned from training examples with pairwise supervised information, that is, similar and dissimilar pairs of examples (also known as side information in the machine learning literature). The relaxed hash functions, actually nonlinear embedding functions, are learned by training a tailored feedforward neural network. Within this architecture, two ISTA-type networks [110] that share the same set of parameters and conduct fast approximations of sparse coding are coupled in the training phase. Since each output of the neural network is continuous albeit sparse, a hyperbolic tangent function is applied to the output followed by a thresholding operation, leading to the final binary hash code. In [99], an extension to hashing multimodal data, e.g., web images with textual tags, was also presented.

Another work named deep hashing [106] developed a deep neural network to learn a multiple hierarchical nonlinear transformation which maps original images to compact binary hash codes and hence supports large-scale image retrieval with the learned binary image representation. The deep hashing model is established under three constraints which are imposed on the top layer of the deep neural network: 1) the reconstruction error between an original real-valued image feature vector and the resulting binary code is minimized; 2) each bit of binary codes has a balance; and 3) all bits are independent from each other. Similar constraints have been adopted in prior unsupervised hashing or binary coding methods such as iterative quantization (ITQ) [111]. A supervised version called supervised deep hashing[3] was also presented in [106], where a discriminative term incorporating pairwise supervised information is added to the objective function of the deep hashing model. Liong *et al.* [106] showed the superiority of the supervised deep hashing model over its unsupervised counterpart. Both of them produce hash codes through thresholding the output of the top layer in the neural network, where all activation functions are hyperbolic tangent functions.

It is worthwhile to point out that the above methods, including sparse similarity-preserving hashing, deep hashing, and supervised deep hashing, did not include a pretraining stage during the training of the deep neural networks. Instead, the hash codes are learned from scratch using a set of training data. However, the absence of pretraining may make the generated hash codes less effective. Specifically, the sparse similarity-preserving hashing method is found to be inferior to the state-of-the-art supervised hashing method, i.e., kernel-based supervised hashing (KSH) [22], in terms of search accuracy on some image data sets [99]; the deep hashing method and its supervised version are slightly better than ITQ and its supervised version CCA + ITQ, respectively [106], [111]. Note that KSH, ITQ, and CCA + ITQ exploit relatively shallow learning frameworks.

Almost all existing hashing techniques including the aforementioned ones relying on deep neural networks take a vector of handcrafted visual features extracted from an image as input. Therefore, the quality of produced hash codes heavily depends on the quality of handcrafted features. To remove this barrier, a recent method called convolutional neural network hashing [107] was developed to integrate image feature learning and hash value learning into a joint learning model. Given pairwise supervised information, this model consists of a stage of learning approximate hash codes and a stage of training a deep convolutional neural network (CNN) [112] that outputs continuous hash values. Such hash values can be generated by activation functions like sigmoid, hyperbolic tangent or softmax, and then quantized into binary hash codes through appropriate thresholding. Thanks to the power of CNNs, the joint model is capable of simultaneously learning image features and hash values, directly working on raw image pixels. The deployed CNN is composed of three convolution-pooling layers that involve rectified linear activation, max pooling, and local contrast normalization, a standard fully connected layer, and an output layer with softmax activation functions.

[3]It is essentially semisupervised as abundant unlabeled examples are used for training the deep neural network.

Also based on CNNs, a latest method called as deep semantic ranking hashing [108] was presented to learn hash values such that multilevel semantic similarities among multilabeled images are preserved. Like the convolutional neural network hashing method, this method takes image pixels as input and trains a deep CNN, by which image feature representations and hash values are jointly learned. The deployed CNN consists of five convolution-pooling layers, two fully connected layers, and a hash layer (i.e., output layer). The key hash layer is connected to both fully connected layers and in the function expression as

$$h(\mathbf{x}) = 2\sigma\left(\mathbf{w}^\top [f_1(\mathbf{x}); f_2(\mathbf{x})]\right) - 1$$

in which $\mathbf{x}$ represents an input image, $h(\mathbf{x})$ represents the vector of hash values for image $\mathbf{x}$, $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$, respectively, denote the feature representations from the outputs of the first and second fully connected layers, $\mathbf{w}$ is the weight vector, and $\sigma()$ is the logistic function. The deep semantic ranking hashing method leverages listwise supervised information to train the CNN, which stems from a collection of image triplets that encode the multilevel similarities, i.e., the first image in each triplet is more similar to the second one than the third one. The hash code of image $\mathbf{x}$ is finally obtained by thresholding the output $h(\mathbf{x})$ of the hash layer at zero.

The above convolutional neural network hashing method [107] requires separately learning approximate hash codes to guide the subsequent learning of image representation and finer hash values. The latest method called deep neural network hashing [109] goes beyond, in which the image representation and hash values are learned in one stage so that representation learning and hash learning are tightly coupled to benefit each other. Similar to the deep semantic ranking hashing method [108], the deep neural network hashing method incorporates listwise supervised information to train a deep CNN, giving rise to a currently deepest architecture for supervised hashing. The pipeline of the deep hashing architecture includes three building blocks: 1) a triplet of images (the first image is more similar to the second one than the third one) which are fed to the CNN, and upon which a triplet ranking loss is designed to characterize the listwise supervised information; 2) a shared subnetwork with a stack of eight convolution layers to generate the intermediate image features; and 3) a divide-and-encode module to divide the intermediate image features into multiple channels, each of which is encoded into a single hash bit. Within the divide-and-encode module, there are one fully connected layer and one hash layer. The former uses sigmoid activation, while the latter uses a piecewise thresholding scheme to produce a nearly discrete hash values. Eventually, the hash code of any image is yielded by thresholding the output of the hash layer at 0.5. In [109], the deep neural network hashing method was shown to surpass the convolutional neural network hashing method as well as several shallow-learning-based supervised hashing methods in terms of image search accuracy.

Last, a few observations are worth mentioning about deep-learning-based hashing methods introduced in this section.

1) The majority of these methods did not report the time of hash code generation. In real-world search scenarios, the speed for generating hashes should be substantially fast. There might be concern about the hashing speed of those deep neural-network-driven approaches, especially the approaches involving image feature learning, which may take much longer time to hash an image compared to shallow-learning-driven approaches like ITQ and KSH.
2) Instead of employing deep neural networks to seek hash codes, another interesting problem is to design a proper hashing technique to accelerate deep neural network training or save memory space. The latest work [113] presented a hashing trick named HashedNets, which shrinks the storage costs of neural networks significantly while mostly preserving the generalization performance in image classification tasks.

VI. Advanced Methods and Related Applications

In this section, we further extend the survey scope to cover a few more advanced hashing methods that are developed for specific settings and applications, such as point-to-hyperplane hashing, subspace hashing, and multimodality hashing.

A. Hyperplane Hashing

Distinct from the previously surveyed conventional hashing techniques all of which address the problem of fast point-to-point nearest neighbor search [see Fig. 11(a)], a new scenario "point-to-hyperplane" hashing emerges to tackle fast point-to-hyperplane nearest neighbor search [see Fig. 11(b)], where the query is a hyperplane instead of a data point. Such a new scenario requires hashing the hyperplane query to near database points, which is difficult to accomplish because point-to-hyperplane distances are quite different from routine point-to-point distances in terms of the computation mechanism. Despite the bulk of research on point-to-point hashing, this special hashing paradigm is rarely touched. For convenience, we call point-to-hyperplane hashing as hyperplane hashing.

Hyperplane hashing is actually fairly important for many machine learning applications such as large-scale active learning with SVMs [114]. In SVM-based active learning [115], the well-proven sample selection strategy is

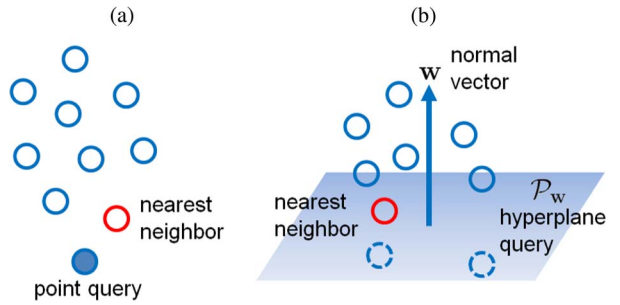

Fig. 11. *Two distinct nearest neighbor search problems. (a) Point-to-point search; the blue solid circle represents a point query, and the red circle represents the found nearest neighbor point. (b) Point-to-hyperplane search; the blue plane denotes a hyperplane query $\mathcal{P}_w$ with w being its normal vector, and the red circle denotes the found nearest neighbor point.*

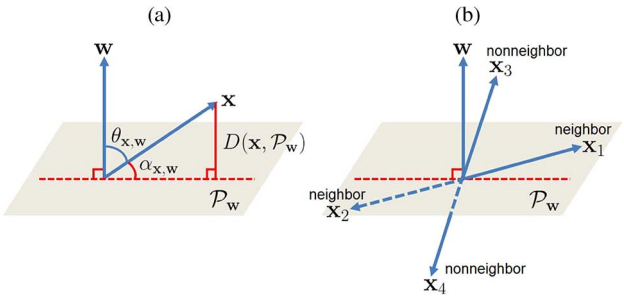

Fig. 12. *Hyperplane hashing problem. (a) Point-to-hyperplane distance $D(x, \mathcal{P}_w)$ and point-to-hyperplane angle $\alpha_{x,w}$. (b) Neighbors (x_1, x_2) and nonneighbors (x_3, x_4) of the hyperplane query $\mathcal{P}_w$, and the ideal neighbors are the points $\perp w$.*

to search in the unlabeled sample pool to identify the sample closest to the current hyperplane decision boundary, thus providing the most useful information for improving the learning model. When making such active learning scalable to gigantic databases, exhaustive search for the point nearest to the hyperplane is not efficient for the online sample selection requirement. Hence, novel hashing methods that can principally handle hyperplane queries are called for.

We demonstrate the geometric relationship between a data point $\mathbf{x}$ and a hyperplane $\mathcal{P}_\mathbf{w}$ with the vector normal as $\mathbf{w}$ in Fig. 12(a). Given a hyperplane query $\mathcal{P}_\mathbf{w}$ and a set of points $\mathcal{X}$, the target nearest neighbor is

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathcal{X}} D(\mathbf{x}, \mathcal{P}_\mathbf{w})$$

where $D(\mathbf{x}, \mathcal{P}_\mathbf{w}) = |\mathbf{w}^\top \mathbf{x}|/\|\mathbf{w}\|$ is the point-to-hyperplane distance. The existing hyperplane hashing methods [116], [117] all attempt to minimize a slightly modified "distance" $|\mathbf{w}^\top \mathbf{x}|/\|\mathbf{w}\|\|\mathbf{x}\|$, i.e., the sine of the point-to-hyperplane angle $\alpha_{\mathbf{x},\mathbf{w}} = |\theta_{\mathbf{x},\mathbf{w}} - (\pi/2)|$. Note that $\theta_{\mathbf{x},\mathbf{w}} \in [0, \pi]$ is the angle between $\mathbf{x}$ and $\mathbf{w}$. The angle measure $\alpha_{\mathbf{x},\mathbf{w}} \in [0, \pi/2]$ between a database point and a hyperplane query turns out to be reflected into the design of hash functions.

As shown in Fig. 12(b), the goal of hyperplane hashing is to hash a hyperplane query $\mathcal{P}_\mathbf{w}$ and the desired neighbors (e.g., $\mathbf{x}_1, \mathbf{x}_2$) with narrow $\alpha_{\mathbf{x},\mathbf{w}}$ into the same or nearby hash buckets, meanwhile avoiding to return the undesired nonneighbors (e.g., $\mathbf{x}_3, \mathbf{x}_4$) with wide $\alpha_{\mathbf{x},\mathbf{w}}$. Because $\alpha_{\mathbf{x},\mathbf{w}} = |\theta_{\mathbf{x},\mathbf{w}} - (\pi/2)|$, the point-to-hyperplane search problem can be equivalently transformed to a specific point-to-point search problem where the query is the hyperplane normal $\mathbf{w}$ and the desired nearest neighbor to the raw query $\mathcal{P}_\mathbf{w}$ is the one whose angle $\theta_{\mathbf{x},\mathbf{w}}$ from $\mathbf{w}$ is closest to $\pi/2$, i.e., most closely perpendicular to $\mathbf{w}$ (we write "perpendicular to $\mathbf{w}$" as $\perp \mathbf{w}$ for brevity). This is very different from traditional point-to-point nearest neighbor search which returns the most similar point to the query point. In the following, several existing hyperplane hashing methods will be briefly discussed.

Jain *et al.* [116] devised two different families of randomized hash functions to attack the hyperplane hashing problem. The first one is angle-hyperplane hash (AH-Hash) $\mathcal{A}$, of which one instance function is

$$h^\mathcal{A}(\mathbf{z}) = \begin{cases} [\mathrm{sgn}(\mathbf{u}^\top \mathbf{z}), \mathrm{sgn}(\mathbf{v}^\top \mathbf{z})], & \mathbf{z} \text{ is a database point} \\ [\mathrm{sgn}(\mathbf{u}^\top \mathbf{z}), \mathrm{sgn}(-\mathbf{v}^\top \mathbf{z})], & \mathbf{z} \text{ is a hyperplane normal} \end{cases} \tag{34}$$

where $\mathbf{z} \in \mathbb{R}^d$ represents an input vector, and $\mathbf{u}$ and $\mathbf{v}$ are both drawn independently from a standard d-variate Gaussian, i.e., $\mathbf{u}, \mathbf{v} \sim \mathcal{N}(0, I_{d\times d})$. Note that $h^\mathcal{A}$ is a two-bit hash function which leads to the probability of collision for a hyperplane normal $\mathbf{w}$ and a database point $\mathbf{x}$

$$\mathbf{Pr}\left[h^\mathcal{A}(\mathbf{w}) = h^\mathcal{A}(\mathbf{x})\right] = \frac{1}{4} - \frac{\alpha_{\mathbf{x},\mathbf{w}}^2}{\pi^2}. \tag{35}$$

This probability monotonically decreases as the point-to-hyperplane angle $\alpha_{\mathbf{x},\mathbf{w}}$ increases, ensuring angle-sensitive hashing.

The second family proposed by Jain *et al.* is embedding-hyperplane hash (EH-Hash) function family $\mathcal{E}$ of which one instance is

$$h^\mathcal{E}(\mathbf{z}) = \begin{cases} \mathrm{sgn}\left(\mathbf{U}^\top \mathbf{V}(\mathbf{z}\mathbf{z}^\top)\right), & \mathbf{z} \text{ is a database point} \\ \mathrm{sgn}\left(-\mathbf{U}^\top \mathbf{V}(\mathbf{z}\mathbf{z}^\top)\right), & \mathbf{z} \text{ is a hyperplane normal} \end{cases} \tag{36}$$

where $\mathbf{V}(A)$ returns the vectorial concatenation of matrix A, and $\mathbf{U} \sim \mathcal{N}(0, I_{d^2 \times d^2})$. The EH hash function $h^{\mathcal{E}}$ yields hash bits on an embedded space $\mathbb{R}^{d^2}$ resulting from vectorizing rank-one matrices $\mathbf{z}\mathbf{z}^\top$ and $-\mathbf{z}\mathbf{z}^\top$. Compared with $h^{\mathcal{A}}, h^{\mathcal{E}}$ gives a higher probability of collision

$$\mathbf{Pr}\left[h^{\mathcal{E}}(\mathbf{w}) = h^{\mathcal{E}}(\mathbf{x})\right] = \frac{\cos^{-1}\sin^2(\alpha_{\mathbf{x},\mathbf{w}})}{\pi} \tag{37}$$

which also bears the angle-sensitive hashing property. However, it is much more expensive to compute than AH-Hash.

More recently, Liu *et al.* [117] designed a randomized function family with bilinear bilinear-hyperplane hash (BH-Hash) as

$$\mathcal{B} = \left\{h^{\mathcal{B}}(\mathbf{z}) = \mathrm{sgn}(\mathbf{u}^\top \mathbf{z}\mathbf{z}^\top \mathbf{v}),\ \text{i.i.d. } \mathbf{u}, \mathbf{v} \sim \mathcal{N}(0, I_{d\times d})\right\}. \tag{38}$$

As a core finding, Liu *et al.* proved in [117] that the probability of collision for a hyperplane query $\mathcal{P}_{\mathbf{w}}$ and a database point $\mathbf{x}$ under $h^{\mathcal{B}}$ is

$$\mathbf{Pr}\left[h^{\mathcal{B}}(\mathcal{P}_{\mathbf{w}}) = h^{\mathcal{B}}(\mathbf{x})\right] = \frac{1}{2} - \frac{2\alpha^2_{\mathbf{x},\mathbf{w}}}{\pi^2}. \tag{39}$$

Specifically, $h^{\mathcal{B}}(\mathcal{P}_{\mathbf{w}})$ is prescribed to be $-h^{\mathcal{B}}(\mathbf{w})$. Equation (39) endows $h^{\mathcal{B}}$ with the angle-sensitive hashing property. It is important to find that the collision probability given by the BH hash function $h^{\mathcal{B}}$ is always twice the collision probability by the AH hash function $h^{\mathcal{A}}$, and also greater than the collision probability by the EH hash function $h^{\mathcal{E}}$. As illustrated in Fig. 13, for any fixed r, BH-Hash accomplishes the highest probability of collision, which indicates that the BH-Hash has a better angle-sensitive property.

In terms of the formulation, the bilinear hash function $h^{\mathcal{B}}$ is correlated yet differently from the linear hash functions $h^{\mathcal{A}}$ and $h^{\mathcal{E}}$. 1) $h^{\mathcal{B}}$ produces a single hash bit which is the product of the two hash bits produced by $h^{\mathcal{A}}$. 2) $h^{\mathcal{B}}$ may be a rank-one special case of $h^{\mathcal{E}}$ in algebra if we write $\mathbf{u}^\top \mathbf{z}\mathbf{z}^\top \mathbf{v} = \mathrm{tr}(\mathbf{z}\mathbf{z}^\top \mathbf{v}\mathbf{u}^\top)$ and $\mathbf{U}^\top \mathbf{V}(\mathbf{z}\mathbf{z}^\top) = \mathrm{tr}(\mathbf{z}\mathbf{z}^\top \mathbf{U})$. 3) $h^{\mathcal{B}}$ appears in a universal form, while both $h^{\mathcal{A}}$ and $h^{\mathcal{E}}$ treat a query and a database item in a distinct manner. The computation time of $h^{\mathcal{B}}$ is $\Theta(2d)$ which is the same as that of $h^{\mathcal{A}}$ and one order of magnitude faster than $\Theta(2d^2)$ of $h^{\mathcal{E}}$. Liu *et al.* further improved the performance of $h^{\mathcal{B}}$ through learning the bilinear projection directions $\mathbf{u}, \mathbf{v}$ in $h^{\mathcal{B}}$ from the data. Gong *et al.* extended the bilinear formulation to the conventional point-to-point hashing scheme through designing compact binary codes for high-dimensional visual descriptors [118].

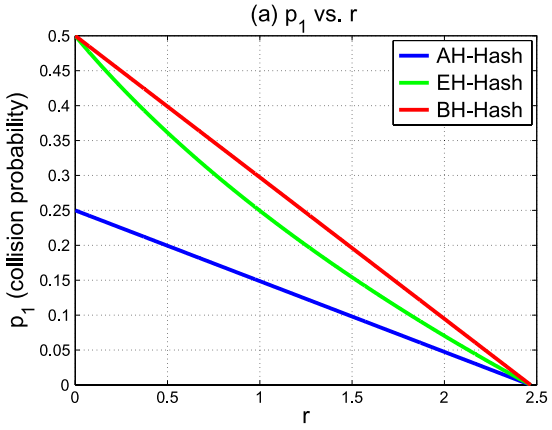

Fig. 13. *Comparison of the collision probabilities of the three randomized hyperplane hashing schemes using p_1 (probability of collision) versus r (squared point-to-hyperplane angle).*

B. Subspace Hashing

Beyond the aforementioned conventional hashing which tackles searching in a database of vectors, subspace hashing [119], which has been rarely explored in the literature, attempts to efficiently search through a large database of subspaces. Subspace representation is very common in many computer vision, pattern recognition, and statistical learning problems, such as subspace representations of image patches, image sets, video clips, etc. For example, face images of the same subject with fixed poses but different illuminations are often assumed to reside near linear subspaces. A common use scenario is to use a single face image to find the subspace (and the corresponding subject ID) closest to the query image [120]. Given a query in the form of vector or subspace, searching for a nearest subspace in a subspace database is frequently encountered in a variety of practical applications including example-based image synthesis, scene classification, speaker recognition, face recognition, and motion-based action recognition [120].

However, hashing and searching for subspaces are both different from the schemes used in traditional vector hashing and the latest hyperplane hashing. Basri *et al.* [119] presented a general framework to the problem of approximate nearest subspace (ANS) search, which uniformly deals with the cases that query is a vector or subspace, query and database elements are subspaces of fixed dimension, query and database elements are subspaces of different dimension, and database elements are subspaces of varying dimension. The critical technique exploited by [119] has two steps: 1) a simple mapping that maps both query and database elements to "points" in a new vector space; and 2) doing approximate nearest

neighbor search using conventional vector hashing algorithms in the new space. Consequently, the main contribution of [119] is reducing the difficult subspace hashing problem to a regular vector hashing task. Basri *et al.* [119] used LSH for the vector hashing task. While simple, the hashing technique (mapping + LSH) of [119] perhaps suffers from the high dimensionality of the constructed new vector space.

More recently, Wang *et al.* [120] exclusively addressed the point-to-subspace query where query is a vector and database items are subspaces of arbitrary dimension. Wang *et al.* [120] proposed a rigorously faster hashing technique than that of [119]. Their hash function can hash D-dimensional vectors (D is the ambient dimension of the query) or $D \times r$-dimensional subspaces (r is arbitrary) in a linear time complexity $O(D)$, which is computationally more efficient than the hash functions devised in [119]. Wang *et al.* [120] further proved the search time under the $O(D)$ hashes to be sublinear in the database size.

Based on the nice finding of [120], we would like to achieve faster hashing for the subspace-to-subspace query by means of crafted novel hash functions to handle subspaces in varying dimension. Both theoretical and practical explorations in this direction will be beneficial to the hashing area.

C. Multimodality Hashing

Note that the majority of the hash learning methods are designed for constructing the Hamming embedding for a single modality or representation. Some recent advanced methods are proposed to design the hash functions for more complex settings, such as that the data are represented by multimodal features or the data are formed in a heterogeneous way [121]. Hashing methods of such type are closely related to the applications in social network, whether multimodality and heterogeneity are often observed. Below we survey several representative methods that are proposed recently.

Realizing that data items like webpage can be described from multiple information sources, composing hashing was recently proposed to design hashing scheme using several information sources [122]. Besides the intuitive way of concatenating multiple features to derive hash functions, Zhang *et al.* [122] also presented an iterative weighting scheme and formulated convex combination of multiple features. The objective is to ensure the consistency between the semantic similarity and the Hamming similarity of the data. Finally, a joint optimization strategy is employed to learn the importance of individual type of features and the hash functions. Coregularized hashing was proposed to investigate the hashing learning across multiparity data in a supervised setting, where similar and dissimilar pairs of intramodality points are given as supervision information [123]. One such typical setting is to index images and the text jointly to preserve the semantic relations between the image and the text. Zhen and Yeung [123] formulate their objective as a boosted coregularization framework with the cost component as a weighted sum of the intramodality and intermodality loss. The learning process of the hash functions is performed via a boosting procedure so that the bias introduced by previous hash function can be sequentially minimized. Dual-view hashing attempts to derive a hidden common Hamming embedding of data from two views, while maintaining the predictability of the binary codes [124]. A probabilistic model called multimodal latent binary embedding was recently presented to derive binary latent factors in a common Hamming space for indexing multimodal data [125]. Other closely related hashing methods include the design of multiple feature hashing for near-duplicate detection [126], submodular hashing for video indexing [127], and probabilistic attributed hashing for integrating low-level features and semantic attributes [128].

D. Applications With Hashing

Indexing massive multimedia data, such as images and video, are the natural applications for learning-based hashing. Especially, due to the well-known semantic gap, supervised and semisupervised hashing methods have been extensively studied for image search and retrieval [29], [41], [69], [129]–[131], and mobile product search [132]. Other closely related computer vision applications include image patch matching [133], image classification [118], face recognition [134], [135], pose estimation [71], object tracking [136], and duplicate detection [51], [52], [126], [137], [138]. In addition, this emerging hash learning framework can be exploited for some general machine learning and data mining tasks, including cross-modality data fusion [139], large-scale optimization [140], large-scale classification and regression [141], collaborative filtering [142], and recommendation [143]. For indexing video sequences, a straightforward method is to independently compute binary codes for each key frames and use a set of hash code to represent video index. More recently, Ye *et al.* proposed a structure learning framework to derive a video hashing technique that incorporates both temporal and spatial structure information [144]. In addition, advanced hashing methods are also developed for document search and retrieval. For instance, Wang *et al.* proposed to leverage both tag information and semantic topic modeling to achieve more accurate hash codes [145]. Li *et al.* designed a two-stage unsupervised hashing framework for fast document retrieval [146].

Hashing techniques have also been applied to the active learning framework to cope with big data applications. Without performing exhaustive test on all the data points, hyperplane hashing can help significantly speed up the interactive training sample selection procedure [114], [117], [116]. In addition, a two-stage hashing scheme is developed to achieve fast query pair selection for large-scale active learning to rank [147].

VII. Open Issues and Future Directions

Despite the tremendous progress in developing a large array of hashing techniques, several major issues remain open. First, unlike the locality sensitive hashing family, most of the learning-based hashing techniques lack the theoretical guarantees on the quality of returned neighbors. Although several recent techniques have presented theoretical analysis of the collision probability, they are mostly based on randomized hash functions [19], [116], [117]. Hence, it is highly desired to further investigate such theoretical properties. Second, compact hash codes have been mostly studied for large-scale retrieval problems. Due to their compact form, the hash codes also have great potential in many other large-scale data modeling tasks such as efficient nonlinear kernel SVM classifiers [148] and rapid kernel approximation [149]. A bigger question is: Instead of using the original data, can one directly use compact codes to do generic unsupervised or supervised learning without affecting the accuracy? To achieve this, theoretically sound practical methods need to be devised. This will make efficient large-scale learning possible with limited resources, for instance, on mobile devices. Third, most of the current hashing technicals are designed for given feature representations that tend to suffer from the semantic gap. One of the possible future directions is to integrate representation learning with binary code learning using advanced learning schemes such as deep neural network. Finally, since heterogeneity has been an important characteristics of the big data applications, one of the future trends will be to design efficient hashing approaches that can leverage heterogeneous features and multimodal data to improve the overall indexing quality. Along those lines, developing new hashing techniques for composite distance measures, i.e., those based on combinations of different distances acting on different types of features, will be of great interest.

REFERENCES

[1] R. Datta, D. Joshi, J. Li, and J. Z. Wang, "Image retrieval: Ideas, influences, trends of the new age," *ACM Comput. Surv.*, vol. 40, no. 2, pp. 1–60, 2008.

[2] G. Shakhnarovich, T. Darrell, and P. Indyk, *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. Cambridge, MA, USA: MIT Press, 2006.

[3] R. Bellman, *Dynamic Programming*, Princeton, NJ, USA: Princeton Univ. Press, 1957.

[4] J. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[5] S. Omohundro, "Efficient algorithms with neural network behavior," *Complex Syst.*, vol. 1, no. 2, pp. 273–347, 1987.

[6] J. Uhlmann, "Satisfying general proximity/similarity queries with metric trees," *Inf. Process. Lett.*, vol. 40, no. 4, pp. 175–179, 1991.

[7] P. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in *Proc. 4th Annu. ACM-SIAM Symp. Discrete Algorithms*, 1993, pp. 311–321.

[8] P. Indyk, "Nearest-neighbor searching in high dimensions," in *Handbook of Discrete and Computational Geometry*, J. E. Goodman and J. O'Rourke, Eds. Boca Raton, FL, USA: CRC Press, 2004.

[9] H. Jegou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 117–128, Jan. 2011.

[10] T. Ge, K. He, Q. Ke, and J. Sun, "Optimized product quantization for approximate nearest neighbor search," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recognit.*, Portland, OR, USA, 2013, pp. 2946–2953.

[11] A. Torralba, R. Fergus, and W. Freeman, "80 million tiny images: A large data set for nonparametric object and scene recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 11, pp. 1958–1970, Nov. 2008.

[12] D. Knuth, *Art of Computer Programming, Volume 3: Sorting and Searching*. Reading, MA, USA: Addison-Wesley, 1997.

[13] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proc. 25th Int. Conf. Very Large Data Bases*, 1999, pp. 518–529.

[14] J. Wang, S. Kumar, and S.-F. Chang, "Semi-supervised hashing for large scale search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 12, pp. 2393–2406, Dec. 2012.

[15] A. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain, "Content-based image retrieval at the end of the early years," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 12, pp. 1349–1380, Dec. 2000.

[16] L. Cayton and S. Dasgupta, "A learning framework for nearest neighbor search," in *Advances in Neural Information Processing Systems 20*, J. Platt, D. Koller, Y. Singer, and S. Roweis, Eds. Cambridge, MA, USA: MIT Press, 2008, pp. 233–240.

[17] J. He, S.-F. Chang, R. Radhakrishnan, and C. Bauer, "Compact hashing with joint optimization of search accuracy and time," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Colorado Springs, CO, USA, 2011, pp. 753–760.

[18] G. Shakhnarovich, "Learning task-specific similarity," Ph.D. dissertation, Dept. Electr. Eng. Comput. Sci., Massachusetts Inst. Technol., Cambridge, MA, USA, 2005.

[19] B. Kulis, P. Jain, and K. Grauman, "Fast similarity search for learned metrics," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 12, pp. 2143–2157, Dec. 2009.

[20] A. Gordo and F. Perronnin:, "Asymmetric distances for binary embeddings," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Colorado Springs, CO, USA, 2011, pp. 729–736.

[21] B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 6, pp. 1092–1104, Jun. 2011.

[22] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang, "Supervised hashing with kernels," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recognit.*, Providence, RI, USA, 2012, pp. 2074–2081.

[23] Y. Lin, R. Jin, D. Cai, S. Yan, and X. Li, "Compressed hashing," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recognit.*, Portland, OR, USA, 2013, pp. 446–451.

[24] A. Joly and O. Buisson, "Random maximum margin hashing," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Colorado Springs, CO, USA, 2011, pp. 873–880.

[25] J. Wang, S. Kumar, and S.-F. Chang, "Sequential projection learning for hashing with compact codes," in *Proc. 27th Int. Conf. Mach. Learn.*, 2010, pp. 1127–1134.

[26] K. He, F. Wen, and J. Sun, "K-means hashing: An affinity-preserving quantization method for learning binary compact codes," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recognit.*, Portland, OR, USA, 2013, pp. 2938–2945.

[27] B. Kulis and T. Darrell, "Learning to hash with binary reconstructive embeddings," in *Advances in Neural Information Processing Systems 20*, Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, Eds. Cambridge, MA, USA: MIT Press, 2009, pp. 1042–1050.

[28] W. Liu, J. Wang, S. Kumar, and S.-F. Chang, "Hashing with graphs," in *Proc. Int. Conf. Mach. Learn.*, Bellevue, WA, USA, 2011, pp. 1–8.

[29] J. Wang, S. Kumar, and S.-F. Chang, "Semi-supervised hashing for scalable image retrieval," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, San Francisco, CA, USA, 2010, pp. 3424–3431.

[30] M. Norouzi and D. Fleet, "Minimal loss hashing for compact binary codes," in *Proc. 27th Int. Conf. Mach. Learn.*, 2011, pp. 353–360.

[31] H. Xu *et al.*, "Complementary hashing for approximate nearest neighbor search," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2011, pp. 1631–1638.

[32] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Advances in Neural Information Processing Systems 21*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds. Cambridge, MA, USA: MIT Press, 2008, pp. 1753–1760.

[33] M. Raginsky and S. Lazebnik, "Locality-sensitive binary codes from shift-invariant kernels," in *Advances in Neural Information Processing Systems 22*, Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, Eds. Cambridge, MA, USA: MIT Press, 2009, pp. 1509–1517.

[34] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proc. 34th Annu. ACM Symp. Theory Comput.*, 2002, pp. 380–388.

[35] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proc. 20th Annu. Symp. Comput. Geometry*, 2004, pp. 253–262.

[36] M. Bawa, T. Condie, and P. Ganesan, "LSH forest: Self-tuning indexes for similarity search," in *Proc. 14th Int. Conf. World Wide Web*, Chiba, Japan, 2005, pp. 651–660.

[37] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe LSH: Efficient indexing for high-dimensional similarity search," in *Proc. 33rd Int. Conf. Very Large Data Bases*, 2007, pp. 950–961.

[38] W. Dong, Z. Wang, W. Josephson, M. Charikar, and K. Li, "Modeling LSH for performance tuning," in *Proc. 17th ACM Conf. Inf. Knowl. Manage.*, 2008, pp. 669–678.

[39] A. Dasgupta, R. Kumar, and T. Sarlos, "Fast locality-sensitive hashing," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Disc. Data Mining*, 2011, pp. 1073–1081.

[40] V. Satuluri and S. Parthasarathy, "Bayesian locality sensitive hashing for fast similarity search," *Proc. VLDB Endowment*, vol. 5, no. 5, pp. 430–441, 2012.

[41] B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing for scalable image search," in *Proc. IEEE Int. Conf. Comput. Vis.*, Kyoto, Japan, 2009, pp. 2130–2137.

[42] Y. Mu, J. Shen, and S. Yan, "Weakly-supervised hashing in kernel space," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, San Francisco, CA, USA, 2010, pp. 3344–3351.

[43] J. Ji, J. Li, S. Yan, B. Zhang, and Q. Tian, "Super-bit locality-sensitive hashing," in *Advances in Neural Information Processing Systems 25*. Cambridge, MA, USA: MIT Press, 2012, pp. 108–116.

[44] Y. Mu and S. Yan, "Non-metric locality-sensitive hashing," in *Proc. 24th AAAI Conf. Artif. Intell.*, 2010, pp. 539–544.

[45] A. Broder, "On the resemblance and containment of documents," in *Proc. Compression Complexity Sequences*, 1997, pp. 21–29.

[46] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations," in *Proc. 13th Annu. ACM Symp. Theory Comput.*, 1998, pp. 327–336.

[47] A. Rajaraman and J. D. Ullman, *Mining of Massive Datasets*. Cambridge, U.K.: Cambridge Univ. Press, 2012.

[48] S. Ioffe, "Improved consistent sampling, weighted Minhash and L1 sketching," in *Proc. IEEE Int. Conf. Data Mining*, 2010, pp. 246–255.

[49] M. Henzinger, "Finding near-duplicate web pages: A large-scale evaluation of algorithms," in *Proc. 29th Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2006, pp. 284–291.

[50] A. S. Das, M. Datar, A. Garg, and S. Rajaram, "Google news personalization: Scalable online collaborative filtering," in *Proc. 16th Int. Conf. World Wide Web*, 2007, pp. 271–280.

[51] O. Chum, J. Philbin, and A. Zisserman, "Near duplicate image detection: Min-hash and TF-IDF weighting," in *Proc. British Mach. Vis. Conf.*, 2008, vol. 810, pp. 812–815.

[52] D. C. Lee, Q. Ke, and M. Isard, "Partition Min-hash for partial duplicate image discovery," in *Proc. Eur. Conf. Comput. Vis.*, Heraklion, Greece, 2012, pp. 648–662.

[53] P. Li and C. König, "B-bit minwise hashing," in *Proc. 19th Int. Conf. World Wide Web*, 2010, pp. 671–680.

[54] P. Li, A. Konig, and W. Gui, "B-bit minwise hashing for estimating three-way similarities," in *Advances in Neural Information Processing Systems 23*, J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, Eds. Cambridge, MA, USA: MIT Press, 2010, pp. 1387–1395.

[55] P. Li, A. Owen, and C.-H. Zhang, "One permutation hashing," in *Advances in Neural Information Processing Systems 25*, P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds. Cambridge, MA, USA: MIT Press, 2012, pp. 3122–3130.

[56] O. Chum, M. Perdoch, and J. Matas, "Geometric Min-hashing: Finding a (thick) needle in a haystack," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, Miami, FL, USA, 2009, pp. 17–24.

[57] O. Chum and J. Matas, "Fast computation of min-Hash signatures for image collections," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recognit.*, Providence, RI, USA, 2012, pp. 3077–3084.

[58] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proc. 30th ACM Symp. Theory Comput.*, 1998, pp. 604–613.

[59] M. Norouzi, A. Punjani, and D. J. Fleet, "Fast search in hamming space with multi-index hashing," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Providence, RI, USA, 2012, pp. 3108–3115.

[60] F. Shen, C. Shen, Q. Shi, A. van den Hengel, and Z. Tang, "Inductive hashing on manifolds," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recognit.*, Portland, OR, USA, 2013, pp. 1562–1569.

[61] Y. Gong and S. Lazebnik, "Iterative quantization: A procrustean approach to learning binary codes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Colorado Springs, CO, USA, 2011, pp. 817–824.

[62] W. Kong and W.-J. Li, "Isotropic hashing," in *Advances in Neural Information Processing Systems 25*. Cambridge, MA, USA: MIT Press, 2012, pp. 1655–1663.

[63] Y. Gong, S. Kumar, V. Verma, and S. Lazebnik, "Angular quantization based binary codes for fast similarity search," in *Advances in Neural Information Processing Systems 25*. Cambridge, MA, USA: MIT Press, 2012, pp. 1205–1213.

[64] J.-P. Heo, Y. Lee, J. He, S.-F. Chang, and S.-E. Yoon, "Spherical hashing," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Providence, RI, USA, 2012, pp. 2957–2964.

[65] M. Norouzi, D. Fleet, and R. Salakhutdinov, "Hamming distance metric learning," in *Advances in Neural Information Processing Systems*. Cambridge, MA, USA: MIT Press, 2012, pp. 1070–1078.

[66] A. Torralba, R. Fergus, and Y. Weiss, "Small codes and large image databases for recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Anchorage, Alaska, USA, 2008, DOI: 10.1109/CVPR.2008.4587633.

[67] L. Fan, "Supervise binary hash code learning with Jensen Shannon divergence," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2013, pp. 2616–2623.

[68] F. Shen, C. Shen, W. Liu, and H. T. Shen, "Supervised discrete hashing," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recognit.*, Boston, MA, USA, 2015, pp. 37–45.

[69] P. Jain, B. Kulis, and K. Grauman, "Fast image search for learned metrics," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Anchorage, AK, USA, 2008, DOI: 10.1109/CVPR.2008.4587841.

[70] P. Jain, B. Kulis, I. S. Dhillon, and K. Grauman, "Online metric learning and fast similarity search," in *Advances in Neural Information Processing Systems*. Cambridge, MA, USA: MIT Press, 2008, pp. 761–768.

[71] G. Shakhnarovich, P. Viola, and T. Darrell, "Fast pose estimation with parameter-sensitive hashing," in *Proc. IEEE Int. Conf. Comput. Vis.*, Nice, France, 2003, pp. 750–757.

[72] M. Rastegari, A. Farhadi, and D. A. Forsyth, "Attribute discovery via predictable discriminative binary codes," in *Proc. Eur. Conf. Comput. Vis.*, Florence, Italy, 2012, pp. 876–889.

[73] M. Ou, P. Cui, F. Wang, J. Wang, and W. Zhu, "Non-transitive hashing with latent similarity components," in *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Disc. Data Mining*, 2015, pp. 895–904.

[74] X. Li, G. Lin, C. Shen, A. V. den Hengel, and A. Dick, "Learning hash functions using column generation," in *Proc. 30th Int. Conf. Mach. Learn.*, 2013, pp. 142–150.

[75] J. Wang, J. Wang, N. Yu, and S. Li, "Order preserving hashing for approximate nearest neighbor search," in *Proc. 21st ACM Int. Conf. Multimedia*, 2013, pp. 133–142.

[76] J. Wang, W. Liu, A. X. Sun, and Y.-G. Jiang, "Learning hash codes with listwise supervision," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2013, pp. 3032–3039.

[77] H. Jégou, M. Douze, C. Schmid, and P. Pérez, "Aggregating local descriptors into a compact image representation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2010, pp. 3304–3311.

[78] J. He, S. Kumar, and S.-F. Chang, "On the difficulty of nearest neighbor search," in *Proc. 29th Int. Conf. Mach. Learn.*, 2012, pp. 1127–1134.

[79] C. Strecha, A. M. Bronstein, M. M. Bronstein, and P. Fua, "LDAHash: Improved matching with smaller descriptors," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 1, pp. 66–78, Jan. 2012.

[80] T. Trzcinski and V. Lepetit, "Efficient discriminative projections for compact binary descriptors," in *Proc. Eur. Conf. Comput. Vis.*, Florence, Italy, 2012, pp. 228–242.

[81] F. Yu, S. Kumar, Y. Gong, and S.-F. Chang, "Circulant binary embedding," in *Proc. Int. Conf. Mach. Learn.*, Beijing, China, 2014, pp. 946–954.

[82] J. He, W. Liu, and S.-F. Chang, "Scalable similarity search with optimized kernel hashing," in *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Disc. Data Mining*, 2010, pp. 1129–1138.

[83] R.-S. Lin, D. A. Ross, and J. Yagnik, "Spec hashing: Similarity preserving algorithm for entropy-based coding," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, San Francisco, CA, USA, 2010, pp. 848–854.

[84] Z. Jin *et al.*, "Complementary projection hashing," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2013, pp. 257–264.

[85] G. Lin, C. Shen, A. V. den Hengel, and D. Suter, "A general two-step approach to learning-based hashing," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2013, pp. 2552–2559.

[86] D. Zhang, J. Wang, D. Cai, and J. Lu, "Self-taught hashing for fast similarity search," in *Proc. 33rd Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2010, pp. 18–25.

[87] Y. Freund and R. Schapire, "A decision-theoretic generalization of on-line

learning and an application to boosting," in *Computational Learning Theory*, vol. 904, Berlin, Germany: Springer-Verlag, 1995, pp. 23–37.

[88] V. Athitsos, J. Alon, S. Sclaroff, and G. Kollios, "Boostmap: An embedding method for efficient nearest neighbor retrieval," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 1, pp. 89–104, Jan. 2008.

[89] Y.-G. Jiang, J. Wang, X. Xue, and S.-F. Chang, "Query-adaptive image search with hash codes," *IEEE Trans. Multimedia*, vol. 15, no. 2, pp. 442–453, Feb. 2013.

[90] Y.-G. Jiang, J. Wang, and S.-F. Chang, "Lost in binarization: Query-adaptive ranking for similar image search with compact codes," in *Proc. ACM Int. Conf. Multimedia Retrieval*, 2011, DOI: 10.1145/1991996.1992012.

[91] Q. Wang, D. Zhang, and L. Si, "Weighted hashing for fast large scale similarity search," in *Proc. 22nd ACM Conf. Inf. Knowl. Manage.*, 2013, pp. 1185–1188.

[92] L. Zhang, Y. Zhang, X. Gu, and Q. Tian, "Binary code ranking with weighted hamming distance," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recognit.*, Portland, OR, USA, 2013, pp. 1586–159.

[93] X. Liu, J. He, B. Lang, and S.-F. Chang, "Hash bit selection: A unified solution for selection problems in hashing," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recognit.*, Portland, OR, USA, 2013, pp. 1570–1577.

[94] X. Zhang, L. Zhang, and H.-Y. Shum, "Qsrank: Query-sensitive hash code ranking for efficientϵ-neighbor search," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Providence, RI, USA, 2012, pp. 2058–2065.

[95] C. Fowlkes, S. Belongie, F. Chung, and J. Malik, "Spectral grouping using the Nyström method," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 2, pp. 214–225, Feb. 2004.

[96] Y. Weiss, R. Fergus, and A. Torralba, "Multidimensional spectral hashing," in *Proc. Eur. Conf. Comput. Vis.*, Florence, Italy, 2012, pp. 340–353.

[97] W. Liu, J. He, and S.-F. Chang, "Large graph construction for scalable semi-supervised learning," in *Proc. 27th Int. Conf. Mach. Learn.*, 2010, pp. 679–686.

[98] W. Liu, C. Mu, S. Kumar, and S.-F. Chang, "Discrete graph hashing," in *Advances in Neural Information Processing Systems*. Cambridge, MA, USA: MIT Press, 2014, pp. 3419–3427.

[99] J. Masci, A. Bronstein, M. Bronstein, P. Sprechmann, and G. Sapiro, "Sparse similarity-preserving hashing," in *Proc. Int. Conf. Learn. Represent.*, 2014, pp. 1–13.

[100] J. V. Davis, B. Kulis, P. Jain, S. Sra, and I. S. Dhillon, "Information-theoretic metric learning," in *Proc. 24th Int. Conf. Mach. Learn.*, 2007, pp. 209–216.

[101] R. M. Gray, *Toeplitz and Circulant Matrices: A Review*. New York, NY, USA: Now Publishers, 2006.

[102] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015.

[103] R. Salakhutdinov and G. Hinton, "Semantic hashing," *Int. J. Approx. Reason.*, vol. 50, no. 7, pp. 969–978, 2009.

[104] G. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[105] R. Salakhutdinov and G. Hinton, "Learning a nonlinear embedding by preserving class neighbourhood structure," *in Proc. Int. Conf. Artif. Intell. Stat.*, 2007, pp. 412–419.

[106] V. E. Liong, J. Lu, G. Wang, P. Moulin, and J. Zhou, "Deep hashing for compact binary codes learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 2475–2483.

[107] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan, "Supervised hashing for image retrieval via image representation learning," in *Proc. AAAI Conf. Artif. Intell.*, 2014, pp. 2156–2162.

[108] F. Zhao, Y. Huang, L. Wang, and T. Tan, "Deep semantic ranking based hashing for multi-label image retrieval," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1556–1564.

[109] H. Lai, Y. Pan, Y. Liu, and S. Yan, "Simultaneous feature learning and hash coding with deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 3270–3278.

[110] K. Gregor and Y. LeCun, "Learning fast approximations of sparse coding," in *Proc. Int. Conf. Mach. Learn.*, 2010, pp. 399–406.

[111] 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.*, vol. 35, no. 12, pp. 2916–2929, Dec. 2013.

[112] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems 25*. Cambridge, MA, USA: MIT Press, 2012, pp. 1106–1114.

[113] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 2285–2294.

[114] S. Vijayanarasimhan, P. Jain, and K. Grauman, "Hashing hyperplane queries to near points with applications to large-scale active learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 2, pp. 276–288, Feb. 2013.

[115] S. Tong and D. Koller, "Support vector machine active learning with applications to text classification," *J. Mach. Learn. Res.*, vol. 2, pp. 45–66, 2001.

[116] P. Jain, S. Vijayanarasimhan, and K. Grauman, "Hashing hyperplane queries to near points with applications to large-scale active learning," in *Advances in Neural Information Processing Systems 23*, J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, Eds. Cambridge, MA, USA: MIT Press, 2010, pp. 928–936.

[117] W. Liu, J. Wang, Y. Mu, S. Kumar, and S.-F. Chang, "Compact hyperplane hashing with bilinear functions," in *Proc. 29th Int. Conf. Mach. Learn.*, 2012, pp. 17–24.

[118] Y. Gong, S. Kumar, H. Rowley, and S. Lazebnik, "Learning binary codes for high-dimensional data using bilinear projections," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recognit.*, Portland, OR, USA, 2013, pp. 484–491.

[119] R. Basri, T. Hassner, and L. Zelnik-Manor, "Approximate nearest subspace search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 2, pp. 266–278, Feb. 2011.

[120] X. Wang, S. Atev, J. Wright, and G. Lerman, "Fast subspace search via grassmannian based hashing," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2013, pp. 2776–2783.

[121] S. Kim, Y. Kang, and S. Choi, "Sequential spectral learning to hash with multiple representations," in *Proc. Eur. Conf. Comput. Vis.*, Florence, Italy, 2012, pp. 538–551.

[122] D. Zhang, F. Wang, and L. Si, "Composite hashing with multiple information sources," in *Proc. 34th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2011, pp. 225–234.

[123] Y. Zhen and D.-Y. Yeung, "Co-regularized hashing for multimodal data," in *Advances in Neural Information Processing Systems 25*, P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds. Cambridge, MA, USA: MIT Press, 2012, pp. 1385–1393.

[124] M. Rastegari, J. Choi, S. Fakhraei, D. Hal, and L. Davis, "Predictable dual-view hashing," in *Proc. 30th Int. Conf. Mach. Learn.*, 2013, pp. 1328–1336.

[125] Y. Zhen and D.-Y. Yeung, "A probabilistic model for multimodal hash function learning," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Disc. Data Mining*, 2012, pp. 940–948.

[126] J. Song, Y. Yang, Z. Huang, H. T. Shen, and R. Hong, "Multiple feature hashing for real-time large scale near-duplicate video retrieval," in *Proc. 19th ACM Int. Conf. Multimedia*, 2011, pp. 423–432.

[127] L. Cao, Z. Li, Y. Mu, and S.-F. Chang, "Submodular video hashing: A unified framework towards video pooling and indexing," in *Proc. 20th ACM Int. Conf. Multimedia*, 2012, pp. 299–308.

[128] M. Ou, P. Cui, J. Wang, F. Wang, and W. Zhu, "Probabilistic attributed hashing," in *Proc. 29th AAAI Conf. Artif. Intell.*, 2015, pp. 2894–2900.

[129] K. Grauman and R. Fergus, "Learning binary hash codes for large-scale image search," in *Machine Learning for Computer Vision*. New York, NY, USA: Springer-Verlag, 2013, pp. 49–87.

[130] K. Grauman, "Efficiently searching for similar images," *Commun. ACM*, vol. 53, no. 6, pp. 84–94, 2010.

[131] W. Kong, W.-J. Li, and M. Guo, "Manhattan hashing for large-scale image retrieval," in *Proc. 35th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2012, pp. 45–54.

[132] J. He *et al.*, "Mobile product search with bag of hash bits and boundary reranking," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Providence, RI, USA, 2012, pp. 3005–3012.

[133] S. Korman and S. Avidan, "Coherency sensitive hashing," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2011, pp. 1607–1614.

[134] Q. Shi, H. Li, and C. Shen, "Rapid face recognition using hashing," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, San Francisco, CA, USA, 2010, pp. 2753–2760.

[135] Q. Dai, J. Li, J. Wang, Y. Chen, and Y.-G. Jiang, "Optimal Bayesian hashing for efficient face recognition," in *Proc. 24th Int. Joint Conf. Artif. Intell.*, 2015, pp. 3430–3437.

[136] X. Li, C. Shen, A. Dick, and A. van den Hengel, "Learning compact binary codes for visual tracking," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recognit.*, Portland, OR, USA, 2013, pp. 2419–2426.

[137] J. Yuan, G. Gravier, S. Campion, X. Liu, and H. Jgou, "Efficient mining of repetitions in large-scale TV streams with product quantization hashing," in *Proc. Eur. Conf. Comput. Vis.*, Florence, Italy, 2012, pp. 271–280.

[138] G. S. Manku, A. Jain, and A. Das Sarma, "Detecting near-duplicates for web

crawling," in *Proc. 16th Int. Conf. World Wide Web*, 2007, pp. 141–150.

[139] M. M. Bronstein, A. M. Bronstein, F. Michel, and N. Paragios, "Data fusion through cross-modality metric learning using similarity-sensitive hashing," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, San Francisco, CA, USA, 2010, pp. 3594–3601.

[140] Y. Mu, J. Wright, and S.-F. Chang, "Accelerated large scale optimization by concomitant hashing," in *Proc. Eur. Conf. Comput. Vis.*, Florence, Italy, 2012, pp. 414–427.

[141] P. Li, A. Shrivastava, J. L. Moore, and A. C. Knig, "Hashing algorithms for large-scale learning," in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, Eds. Cambridge, MA, USA: MIT Press, 2011, pp. 2672–2680.

[142] K. Zhou and H. Zha, "Learning binary codes for collaborative filtering," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Disc. Data Minings*, 2012, pp. 498–506.

[143] M. Ou *et al.*, "Comparing apples to oranges: A scalable solution with heterogeneous hashing," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Disc. Data Mining*, 2013, pp. 230–238.

[144] G. Ye, D. Liu, J. Wang, and S.-F. Chang, "Large-scale video hashing via structure learning," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2013, pp. 2272–2279.

[145] Q. Wang, D. Zhang, and L. Si, "Semantic hashing using tags and topic modeling," in *Proc. 36th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2013, pp. 213–222.

[146] H. Li, W. Liu, and H. Ji, "Two-stage hashing for fast document retrieval," in *Proc. Annu. Meeting Assoc. Comput. Linguistics*, Baltimore, MD, USA, 2014, pp. 495–501.

[147] B. Qian *et al.*, "Fast pairwise query selection for large-scale active learning to rank," in *Proc. IEEE Int. Conf. Data Mining*, 2013, pp. 607–616.

[148] Y. Mu, G. Hua, W. Fan, and S.-F. Chang, "Hash-SVM: Scalable kernel machines for large-scale visual classification," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recognit.*, Columbus, OH, USA, 2014, pp. 446–451.

[149] Q. Shi *et al.*, "Hash kernels for structured data," *J. Mach. Learn. Res.*, vol. 10, pp. 2615–2637, 2009.

ABOUT THE AUTHORS

Jun Wang (Member, IEEE) received the Ph.D. degree in electrical engineering from Columbia University, New York, NY, USA, in 2011.

Currently, he is a Professor at the School of Computer Science and Software Engineering, East China Normal University, Shanghai, China and an adjunct faculty member of Columbia University. He is also affiliated with the Institute of Data Science and Technology, Alibaba Group, Seattle, WA, USA. From 2010 to 2014, he was a Research Staff Member at IBM T. J. Watson Research Center, Yorktown Heights, NY, USA. His research interests include machine learning, data mining, mobile intelligence, and computer vision.

Dr. Wang has been the recipient of several awards, including the award of the "Thousand Talents Plan" in 2014, the Outstanding Technical Achievement Award from IBM Corporation in 2013, and the Jury Thesis Award from Columbia University in 2011.

Wei Liu (Member, IEEE) received the M.Phil. and Ph.D. degrees in electrical engineering from Columbia University, New York, NY, USA, in 2012.

He has been a Research Scientist at IBM T. J. Watson Research Center, Yorktown Heights, NY, USA, since 2012. He holds adjunct faculty positions at the Rensselaer Polytechnic Institute (RPI), Troy, NY, USA and Stevens Institute of Technology, Hoboken, NJ, USA. He has published more than 80 peer-reviewed journal and conference papers. His research interests include machine learning, data mining, information retrieval, computer vision, pattern recognition, and image processing. His current research work is geared to large-scale machine learning, Big Data analytics, multimedia search engine, mobile computing, and parallel and distributed computing.

Dr. Liu is the recipient of the 2011–2012 Facebook Fellowship and the 2013 Jury Award for best thesis of the Department of Electrical Engineering, Columbia University.

Sanjiv Kumar (Member, IEEE) received the Ph.D. degree from the School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, in 2005.

He joined Google Research, New York, NY, USA, in 2005, where he is currently a lead researcher working on theory and applications in Big Data. His recent research interests have included binary embeddings for large-scale machine learning, nearest neighbor search in massive data sets, low-rank decompositions of huge matrices, and high-throughput online clustering. He has also been an adjunct faculty member at Columbia University, New York, NY, USA, where he developed and taught a new course on large-scale machine learning in 2010. In the past, he held various research positions at the National Robotics Engineering Consortium, Pittsburgh, PA, USA; and the Department of Surgery, National University of Singapore, Singapore, in medical and industrial robotics.

Shih-Fu Chang (Fellow, IEEE) received the Ph.D. degree in electrical engineering and computer sciences from the University of California at Berkeley, Berkeley, CA, USA, in 1993.

In his current capacity as Senior Executive Vice Dean of Columbia Engineering School, Columbia University, New York, NY, USA, he plays a key role in strategic planning, research initiatives, and faculty development. He is a leading researcher in multimedia information retrieval, computer vision, signal processing, and machine learning. His work set trends in areas such as content-based image search, video recognition, image authentication, hashing for large image database, and novel application of visual search in brain–machine interface and mobile systems. Impact of his work can be seen in more than 300 peer-reviewed publications, best paper awards, 25 issued patents, and technologies licensed to many companies.

Dr. Chang has been recognized with the IEEE Signal Processing Society Technical Achievement Award, the ACM Multimedia SIG Technical Achievement Award, the IEEE Kiyo Tomiyasu Award, the ONY YIA award, the IBM Faculty Award, and the Great Teacher Award from the Society of Columbia Graduates. He served as the Editor-in-Chief of the IEEE Signal Processing Magazine during 2006–2008. He is a Fellow of the American Association for the Advancement of Science.