Neurocomputing 166 (2015) 271–281

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Tree-based compact hashing for approximate nearest neighbor search

Guangdong Hou, Runpeng Cui, Zheng Pan, Changshui Zhang*

State Key Lab of Intelligent Technologies and Systems, Tsinghua National Laboratory for Information Science and Technology (TNList), Department of Automation, Tsinghua University, China

ARTICLE INFO

Article history:
Received 5 June 2014
Received in revised form
11 March 2015
Accepted 6 April 2015
Communicated by Deng Cai
Available online 16 April 2015

Keywords:
Hashing
Binary codes
Approximate nearest neighbor search

ABSTRACT

Hashing methods map high-dimensional data onto compact binary codes for efficient retrieval. These methods focus on preserving the data similarity in Hamming distance between the mapped hash codes. In this paper we propose a novel hashing method motivated by maximizing the probability of data with the same hash code being true neighbors, under the constraint of code compactness. This method is data-dependent and generates quite compact hash codes. The key idea is to use a collection of tree-structured hyperplanes to satisfy the compactness constraint, as well as to maximize the lower bound of the objective function. We compare our method with some widely used hashing methods on real datasets of different sizes. The experimental results illustrate the superior performance of our method. The performance of this method is further effectively improved by a multi-table extension.

1. Introduction

With the development of information technology, it is now possible to use huge datasets for challenging machine learning tasks. Therefore, to accelerate search in large-scale and high-dimensional datasets, fast approximate nearest-neighbor (ANN) search becomes more important and necessary.

Hashing method is such an ANN method and demonstrates enormous potential in many applications with large-scale datasets, such as scene recognition [1], pose estimation [2], data fusion [3], image retrieval and classification [4–8], and multi-class object detection [9].

The goal of hashing is to translate high-dimensional data into compact binary codes, so that the hash codes of similar data will have either short Hamming distance or high collision probability. Since Hamming distance between two binary codes can be calculated quite efficiently via XOR operation, alternatively, a lookup table can be easily constructed on binary codes, hashing enables efficient retrieval.

For a high-dimensional input, $\mathbf{x} \in \mathbb{R}^d$, hashing methods map it onto a binary hash code $H(\mathbf{x}) \in \{-1, 1\}^K$, where $K \ll d$ and $H(\mathbf{x}) = [h_1(\mathbf{x}), \ldots, h_K(\mathbf{x})]$. Each hash bit is the output of a hashing function $h_k(\mathbf{x})$. Each possible code value is corresponding to a collection of data, and named a "hash bucket", the data mapped to the same bucket are "collision data".

According to whether the dataset is considered when constructing hashing functions, hashing methods can be briefly categorized into two types: data-independent ones and data-dependent ones.

The data-independent methods construct hashing functions in a random manner. It requires less calculation in construction stage, but is usually not efficient when the code length is short.

For example, LSH (Locality Sensitive Hashing) [10] and E2-LSH [11] use random linear projections as hashing functions. The k-th bit is determined by $\text{sign}(\mathbf{w}_k^T\mathbf{x} + b_k)$, where $\mathbf{w}_k$ and b_k are sampled from a p-stable distribution and a uniform distribution, respectively. It has good asymptotic properties to guarantee the collision probability of similar data, but is usually not quite efficient in practical applications since the bits are usually redundant. Refs. [12,13] expand this approach to its kernel version. SKLSH (Shift-Invariant Kernels Locality-Sensitive Hashing) [14] finds another random way to approximate the original distance with increasing hash bits. Its hashing function is given by $\frac{1}{2}[1 + \text{sign}(\cos(\mathbf{w}_k^T\mathbf{x} + b_k) + t_k)]$, where the parameters $\mathbf{w}_k$, b_k and t_k are sampled independently from their specified probability distributions.

Correspondingly, data-dependent methods create hashing functions by learning from the dataset. The structure of the dataset helps to produce more efficient codes.

ITQ (Iterative Quantization) [15] minimizes the quantization loss when using sign functions in hashing. It takes orthogonal basis, *e.g.*, basis from PCA, as the initialization of linear hashing functions. Then an EM mechanism is used to seek a rotation matrix for the basis, which helps to minimize the quantization loss.

SH (Spectral Hashing) [16] tries to find hash codes whose Hamming distances are consistent with the original Gaussian kernel distances in some degree. Constraints on the codes compactness are concerned in this method. With more assumptions on the data

* Corresponding author.
E-mail address: zcs@mail.tsinghua.edu.cn (C. Zhang).

http://dx.doi.org/10.1016/j.neucom.2015.04.012
0925-2312/

distribution, they convert the problem into an eigenvalue decomposition problem and obtain hashing functions with a simple form.

Semantic hashing [17] uses restricted Boltzmann machines to learn the hash codes. CGHash [18] learns the hashing functions on the basis of proximity comparison information. SSH (Semi-Supervised Hashing) [19,20] uses pairwise similarity and dissimilarity information to improve the learning of linear hashing functions. Ref. [21] learns the hashing functions sequentially, thus the deviation caused by the pervious hashing functions can be compensated by the subsequent ones. CH (Complementary Hashing) [22,6] maps data into multiple hash tables to balance the precision and recall in different ways. They illustrate the benefit of multiple hash tables.

What makes a good hash code? First, it should be efficient for responding the query, *i.e.*, finding the approximate nearest neighbors; second, it should be compact, which makes more efficient retrieval possible and reduces storage space; finally, it should be easy to encode new samples. These standards make the hashing methods challenging.

In this paper, we propose a novel hashing method and its multi-table extension. This method generates quite compact hash codes by evenly splitting the dataset. Our contributions can be summarized as follows:

(1) A compact hashing method. It aims to maximize the probability that collision data are true neighbors. Compared with the widely used hashing methods, our method generates more compact binary codes, and gives superior retrieval performance in our experiments.

(2) A multiple tables paradigm. The compact hashing method can be extended into the multi-table version, which achieves a better precision–recall performance than a single table at the price of a slight loss in code compactness.

(3) The characteristics of several hashing methods are compared. The experiments and theoretical analysis are given to reveal that on what kind of dataset the proposed method can be more suitable.

The rest of the paper is organized as follows: the proposed method is detailed in Section 2, including the motivation and methodology. The different behaviors with other methods are also discussed. Section 3 describes the multiple tables version of the method. The experimental results are shown and analyzed in Section 4.

2. Tree-based compact hashing

In this section, we introduce our method to learn compact and efficient hash codes. We start with the constraints on code compactness. In our method, we take such constraints as hard constraints and use a collection of tree-structured functions to meet them. Besides, instead of trying to preserve the original data distance, we focus on making the data with the same hash code closer.

2.1. Objective function

An ideal hash code should be as compact as possible for more efficient retrieval and more storage space saving. Many hashing methods formulate the objective function under the code compactness constraint.

To make hash codes compact, the bits generated by one hashing function on the whole dataset are expected to maximize information entropy reduction of the data, and be uncorrelated with those of other hashing functions. For linear projection-based hashing function $h_k(\mathbf{x}) = \text{sign}(\mathbf{w}_k^T\mathbf{x})$, the optimization on compactness can be expressed by the formulation (1) [15,19]:

$$\begin{aligned} \max_{\mathbf{w}_k} \quad & \sum_{k=1}^{K} \text{var}(\text{sign}(X\mathbf{w}_k)) \\ \text{s.t.} \quad & \frac{1}{N}B^TB = I, \end{aligned} \tag{1}$$

where $B \in \{-1,1\}^{N\times K}$ denotes the hash codes for a dataset X with N data points. The (i,k)-th element of B is given by $h_k(\mathbf{x}_i)$. It is noted that the maximum variance for (1) means exactly half of the dataset have $h_k(\mathbf{x}_i) = 1$, and the other half are -1 s [15,19]. It implies that $B^T\mathbf{1} = \mathbf{0}$, which is used as another compactness constraint in [16].

These constraints force the codes to be compact. From the view of information theory, the codes are compact means having a large entropy with the same number of bits, and the codes generated under these constraints have the maximum entropy, as we stated in Proposition 1.

Proposition 1. *For a dataset $X \in \mathbb{R}^{N\times d}$ and a scheme H with K hashing functions, if the generated hash codes $B \in \{-1,1\}^{N\times K}$ satisfy the compactness constraints $B^T\mathbf{1} = \mathbf{0}$ and $\frac{1}{N}B^TB = I$, then the codes have the maximum entropy.*

Proof. Let s denote a bucket, $X_s = \{\mathbf{x}|H(\mathbf{x}) = s\}$. So $p_s = P(H(\mathbf{x}) = s) = \frac{|X_s|}{N}$. Since $B^T\mathbf{1} = \mathbf{0}$ and $\frac{1}{N}B^TB = I$, for any s we have

$$p_s = \prod_{k=1}^{K} P(h_k(\mathbf{x}) = s(k)) = \prod_{k=1}^{K} \frac{1}{2} = 2^{-K}.$$

Consider the function $f(x) = -x\log_2 x$, $x \in (0,1)$. So $-f(x)$ is convex. For the hashing scheme H, we have

$$\begin{aligned} \text{Entropy}(H) &= -\sum_{s=0}^{2^K-1} p_s \log_2 p_s \le -2^K\left(\frac{\sum_{s=0}^{2^K-1} p_s}{2^K}\right)\log_2\left(\frac{\sum_{s=0}^{2^K-1} p_s}{2^K}\right) \\ &= \log_2 2^K = K. \end{aligned}$$

If and only if $p_s = 2^{-K}, s = 0,\ldots,2^K-1$, the codes have the maximum entropy. □

Unfortunately, the maximum variance solution of (1) may not be reachable for some forms of hashing functions, such as the above linear projection-based hashing functions, and it is also intractable to solve since this problem is non-convexity and non-differentiable. Thus in previous works, it is usually relaxed into (2) [15,19]:

$$\begin{aligned} \max_{W} \quad & \frac{1}{N}\text{tr}(W^TX^TXW) \\ \text{s.t.} \quad & W^TW = I, \end{aligned} \tag{2}$$

where $W \in \mathbb{R}^{d\times K}$ and the k-th column of W is a projection vector $\mathbf{w}_k$ which determines the corresponding hashing function by $h_k(\mathbf{x}) = \text{sign}(\mathbf{w}_k^T\mathbf{x})$. This relaxation makes the problem tractable. It replaces the bit-orthogonality with the orthogonality of the basis of the original data space. If the data have a uniform distribution in every direction given by $\mathbf{w}_k$, the two are equivalent. However, in most cases, this assumption does not hold and the relaxation may lead to a quite different solution from that of the original problem.

To address this problem, we try to optimize our objective function under the original compactness constraints [16]. We use a new optimizing objective and formulate our objective function as (3):

$$\begin{aligned} \max_{H} \quad & P(\|\mathbf{x}_i - \mathbf{x}_j\| < R | H(\mathbf{x}_i) = H(\mathbf{x}_j)) \\ \text{s.t.} \quad & B^T\mathbf{1} = \mathbf{0} \\ & \frac{1}{N}B^TB = I. \end{aligned} \tag{3}$$

This optimization objective is the probability of collision data being within a specified distance R, which can be the distance threshold to define two neighbors. It is meaningful for the hash retrieval, especially when using a lookup table, since only the collision data (or those within quite small Hamming radius) are checked in this case.

For the most compact K-bits hash codes, *i.e.*, the codes satisfy the constraints in (3), the total number of buckets is 2^K. They provide a