Neurocomputing 141 (2014) 117–123

Contents lists available at ScienceDirect

Neurocomputing

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

Linear spectral hashing

Zalán Bodó*, Lehel Csató

Babeş–Bolyai University, Faculty of Mathematics and Computer Science, Kogălniceanu 1, 400084 Cluj-Napoca, Romania

ARTICLE INFO

Article history:
Received 27 June 2013
Received in revised form
11 October 2013
Accepted 8 November 2013
Available online 8 April 2014

Keywords:
Nearest-neighbor search
Spectral hashing
Spectral clustering

ABSTRACT

Spectral hashing assigns binary hash keys to data points. This is accomplished via thresholding the eigenvectors of the graph Laplacian and obtaining binary codewords. While calculation for inputs in the training set is straightforward, an intriguing and difficult problem is how to compute the hash codewords for previously unseen data. For specific problems we propose linear scalar products as similarity measures and analyze the performance of the algorithm. We implement the linear algorithm and provide an inductive – generative – formula that leads to a codeword generation method similar to random hyperplane-based *locality-sensitive hashing* for a new data point.

1. Introduction

In recent years several algorithms were proposed for fast approximate nearest-neighbor search, providing sub-linear search times for a query. Introduced by Cover and Hart [1], the *k*-nearest neighbor algorithm is one of the most successful and most investigated methods in the machine learning community. However, for large datasets it has a serious drawback: it must go through all the *training* data to find the nearest neighbors. This implies a linear time complexity for a single point. The method of *k*-nearest neighbors can be used in machine learning to label an unknown point based on the majority label among the neighbors. Considering the nearest neighbors of a point, however, is not used only in machine learning; the properties of the neighbors can be beneficially utilized to infer supposedly valid facts about the point in question. Without pretension to completeness, we can mention important applications in information retrieval, computer vision, coding theory, recommendation systems, computational geometry, etc. [2] Therefore, speeding up the search for nearest neighbors we consider a task of great importance.

One of the first approaches to reduce the number of comparisons in finding the nearest-neighbors of a given point was to store points in a special data structure, a k-d tree [3]. K-d trees are binary space-partitioning trees, in which every node is a k-dimensional point, used to hierarchically decompose the space by hyperplanes along orthogonal dimensions. The main disadvantage of using such data structures lies in its sensibility to high-dimensional points, in which case a nearest-neighbor search evaluates most of the points in the tree.

Learned binary embeddings for large datasets, where nearest-neighbors of a given point needed to be found, are efficient and powerful tools for indexing these sets. These embeddings are designed to approximately preserve similarities in the embedding Hamming space. The beneficial properties of the codewords make possible to use Hamming distance computations for nearest-neighbor search, by which the searching process can be substantially sped up: to compute the Hamming distance of two vectors, a XOR operation has to be performed between the vectors and the resulting set bits have to be counted.

We differentiate between two parts of fast nearest-neighbor search with binary embeddings. The first part consists of generating the binary codes, and the second part is the actual searching process. The simplest method for determining the nearest-neighbors is linear scan: calculate the Hamming distance to every codeword from the database. Although it is a brute-force method, it is practical even for very large datasets. Semantic hashing [4] maps binary codes to memory addresses, where at each address a pointer is stored to a list of points that share the respective codeword. Querying the neighbors of a point is done by calculating the hash codeword, flipping some of the bits to obtain the desired Hamming ball around the query, and taking the points with these codewords. Another approach is locality-sensitive hashing (LSH) [5], a randomized framework for generating hash codewords and searching for nearest-neighbors. It creates a hash table where similar points will likely to be put in the same bucket or nearby buckets having a small Hamming distance. Linear scan and semantic hashing can be used with any codeword generation method, however, in semantic hashing the codeword length is limited by the size of the physical memory. LSH provides an elegant way to control the size of the Hamming ball in the original

* Corresponding author.
E-mail addresses: zbodo@cs.ubbcluj.ro (Z. Bodó), lehel.csato@cs.ubbcluj.ro (L. Csató).

http://dx.doi.org/10.1016/j.neucom.2013.11.039
0925-2312/

feature space, but choosing an appropriate LSH function limits its flexibility [6].

One can distinguish between unsupervised, supervised and semi-supervised codeword generation methods, based on the information they use to obtain the embedding [6]. Unsupervised methods use only the information carried by the points themselves. Additional information can be given to supervised methods in the form of label information as in a supervised machine learning setting, as well as giving the neighborhood list for a subset of points, or as paired constraints – defining the points which ought or ought not cluster together. Finally, semi-supervised methods exploit the supervised information, besides which other clustering or regularization approaches are used.

Locality-sensitive hashing with hyperplanes [7] is perhaps the most popular unsupervised hashing method, generating binary codewords by taking random vectors as normal vectors of separating hyperplanes. The hash function is the sign of the dot product between the data point and the random vectors. In [8] it was extended to support arbitrary kernels by using a clever method to generate random vectors in the feature space. Spectral hashing [9] picks codewords by searching minimum-distance binary hash sequences between similar points, similarity being defined by an appropriate proximity matrix. The optimization leads to a clustering problem, from which it becomes intriguing and difficult to generalize to new data points. In [9] this is solved using the eigenfunctions of the weighted Laplace–Beltrami operators, assuming a multidimensional uniform distribution. Shift-invariant kernel-based binary codes [10] use the random projections of [11] giving a low-dimensional similarity-preserving representation of points, where similarity is defined by the kernel. These low-dimensional vectors are then used to obtain binary codes for hashing. The method is limited to use shift-invariant (e.g. Gaussian) kernels. Self-taught hashing [12] uses a two-step approach for codeword generation. The first step consists of the unsupervised learning of binary codes using a similar objective function to spectral hashing. To generalize the obtained mappings, in the second step it considers the set of training examples together with their labels, obtained in the previous step, and trains a support vector machine for each bit of the codeword. We also mention Laplacian co-hashing [13], a hashing method for document retrieval. It considers the documents and terms a bipartite graph, where a term is connected to a document if appears in it, and finds binary codes that best preserves the similarities encoded by the term–document matrix. It differs from existing methods by simultaneously minimizing Hamming distances between document and term codewords.

Parameter-sensitive hashing [14] is an extension of LSH which trains classifiers for the individual regression tasks using label information. In [15] the codeword generation method learns a parametrization of the Mahalanobis distance, based on the supervised information given, and simultaneously encodes the learned information into randomized hash functions. This approach uses the hash function family proposed by [7] with generalized dot products. The embedding proposed in [4] uses multiple layers of restricted Boltzmann machines to assign efficient binary codes to documents. The learning process consists of two phases: an unsupervised pre-training and a supervised fine-tuning of the weights using label information.

The recently proposed semi-supervised approaches from works [16,17] minimize the empirical error on the labeled training data and an information theoretic regularizer over both labeled and unlabeled sets.

Getting supervised information is often a costly process. Spectral hashing is a successful unsupervised codeword generation method that *learns* short binary codewords by minimizing the codeword distances between similar points. Assuming a multidimensional uniform distribution is a very limiting factor, however, spectral hashing performs surprisingly well in practice on a large variety of data. Furthermore, in the original formulation of the method the Gaussian kernel is used, which is not well-suited for specific problems.

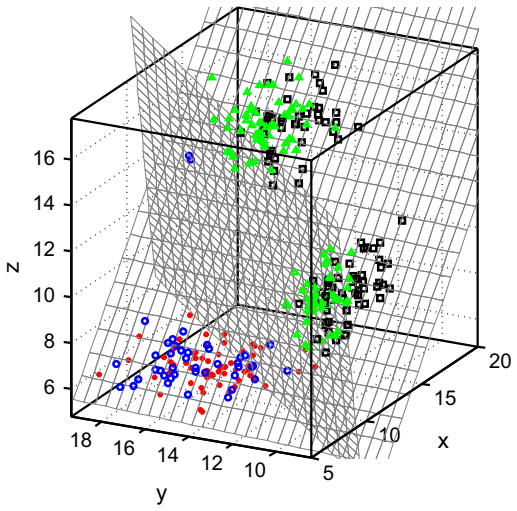

Fig. 1. Points sampled from three multivariate Gaussians and their hashing to two dimensions performed using linear spectral hashing.

In this paper we propose another method for computing the hash codewords for previously unseen points using the linear kernel. The codeword is obtained by computing the dot product of the new vector with a set of precomputed vectors and thresholding the resulting values at zero. The vectors output by the method are normal vectors of maximum-margin separating hyperplanes. Fig. 1 illustrates our method, hashing three-dimensional points to two dimensions.

The structure of the paper is as follows: Section 2 presents spectral clustering in general. It presents normalized spectral clustering and its generalization using maximum-margin separating hyperplanes. After presenting spectral hashing in Section 3, the maximum-margin formulation is used in Section 4 to derive a new algorithm for computing the codewords of new data points. Sections 5 and 6 describe our experiments and discuss the results, respectively.

2. Spectral clustering and max-margin hyperplanes

2.1. Spectral clustering

In *spectral clustering* [18] – one of the most successful clustering algorithms – partitioning is done using the minimum cut in the neighborhood graph, where minimum cut is defined as removing edges such that the graph is partitioned into disconnected subgraphs with minimal sum of edges, $\sum_{i\in A, j\in \bar{A}} W_{ij}$, where A and $\bar{A}$ denotes the clusters, V is the entire vertex set, $A\cup\bar{A}=V$, and W_{ij} is the proximity of the i-th and j-th examples.

The sum of edges alone as an objective function favors unbalanced and small clusters. To overcome this problem, normalized cut was introduced [19], where instead of considering only the sum of weights, the ratio of the sum of weights between partitions and the sum of weights to all the nodes from the partitions is taken, that is

$$\frac{\sum_{i\in A, j\in \bar{A}} W_{ij}}{\sum_{i\in A, v\in V} W_{iv}} + \frac{\sum_{i\in A, j\in \bar{A}} W_{ij}}{\sum_{j\in \bar{A}, v\in V} W_{jv}} \tag{1}$$

The above formulation of spectral clustering considers only binary partitioning, but the problem can be extended to support multiple