Pattern Recognition Letters 55 (2015) 42–50

Contents lists available at ScienceDirect

Pattern Recognition Letters

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

An efficient tree structure for indexing feature vectors ☆

The-Anh Pham [a,b,*], Sabine Barrat [a], Mathieu Delalandre [a], Jean-Yves Ramel [a]

[a] Laboratory of Computer Science, Francois Rabelais University, Tours 37200, France
[b] Hong Duc University, Thanh Hoa City, Viet Nam

ARTICLE INFO

Article history:
Received 4 December 2013
Available online 11 October 2014

Keywords:
Exact nearest neighbour search
Approximate nearest neighbour search
Feature indexing
Randomized KD-trees
Randomized clustering trees

ABSTRACT

This paper addresses the problem of feature indexing in feature vector space. A linked-node m-ary tree (LM-tree) structure is presented to quickly produce the queries for an approximate and exact nearest neighbour search. Three main contributions are made, which can be attributed to the proposed LM-tree. First, a new polar-space-based method of data decomposition is presented to construct the LM-tree. Second, a novel pruning rule is proposed to efficiently narrow down the search space. Finally, a bandwidth search method is introduced to explore the nodes of the LM-tree. Extensive experiments were performed to study the behaviour of the proposed indexing algorithm. These experimental results showed that the proposed algorithm provides a significant improvement in the search performance for the tasks of both exact and approximate nearest neighbour (ENN/ANN) searches.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

Fast nearest neighbour search is of central importance for many computer vision systems, such as systems for object matching, object recognition, and image retrieval. Although a large number of indexing algorithms have been proposed in the literature, few of them (*e.g.*, LHS-based schemes of Lv et al. [1], randomized KD-trees of Silpa-Anan and Hartley [2], randomized K-medoids clustering trees of Muja and Lowe [3], and the hierarchical K-means tree of Muja and Lowe [4]) have been well validated with extensive experiments to show satisfactory performance on specific benchmarks. In this work, we restrict the indexing algorithms to feature vector space rather than metric space. Generally, these indexing algorithms are categorized into space-partitioning, clustering, hashing and hybrid approaches. We will discuss hereafter the most representative methods.

Space-partitioning approach: Friedman et al. [5] introduced KD-tree, whose basic idea is to iteratively partition the data X into two roughly equal-sized subsets, by using a hyperplane that is perpendicular to a split axis in a D-dimensional vector space. Searching for a nearest neighbour of a given query point q is accomplished by using a branch-and-bound technique. Several variations of the KD-tree have been investigated to address the ANN search. *Best-Bin-First* (BBF) search or priority search of Beis and Lowe [6] is a typical improvement on the KD-tree. Its basic idea is twofold. First, it limits the maximum number of data points to be searched. Second, it visits the nodes in the order of increasing distances to the query. The use of priority search was further improved in [2]. In this paper, the author constructed multiple *randomized* KD-trees (*RKD*-trees), each of which is built by selecting, randomly, at each node, a split axis from a few dimensions having the highest variance. A slight difference in the *RKD*-trees was also investigated in this work, where the data are initially aligned to the principal axes. Hence, the obtained indexing scheme is called principal component KD-trees (*PKD*-trees). Experimental results show significantly outstanding performance compared to the use of a single KD-tree. A last noticeable improvement in the KD-tree for the ENN search is the principal axis tree (PAT-tree) of [7]. In the PAT-tree, the split axis is chosen as the principal axis, which has the highest variance in the underlying data. Therefore, the obtained regions are treated as hyper-polygons rather than as hyper-rectangles, as in the KD-tree. Consequently, this approach complicates the process of computing the lower bound of the distance from the query to a given node.

Clustering approach: The clustering-based indexing methods differ from the space-partitioning-based methods mainly in the step of tree construction. Instead of dividing the data by using a hyper-plane, these methods employ a clustering method (*e.g.*, K-means by Fukunaga and Narendra [8], K-medoids by Muja and Lowe [3]) to iteratively partition the underlying data into sub-clusters. The partitioning process is then repeated until the size of all of the sub-clusters falls below a threshold. Muja and Lowe [4] extended the work in Fukunaga and Narendra [8] by incorporating the use of priority queue to the hierarchical clustering tree. In their work, an ANN search proceeds by

☆ This paper has been recommended for acceptance by G. Sanniti di Baja.

* Corresponding author at: Hong Duc University, Thanh Hoa City, Viet Nam. Tel.: +84 912 721 200.
E-mail address: phamtheanh@hdu.edu.vn (T. Pham).

http://dx.doi.org/10.1016/j.patrec.2014.08.006
0167-8655/© 2014 Elsevier B.V. All rights reserved.

traversing down the tree and always chooses the node whose cluster centre is closest to the query. Each time that a node is selected for further exploration, the other sibling nodes are inserted into a priority queue that contains a sequence of nodes that are stored in increasing order of their distances to the query. This process continues when reaching a leaf node and is followed by a sequence search for the points contained in that node. Backtracking is then invoked, starting from the top node in the priority queue. During the search process, the algorithm maintains adding new candidate nodes to the priority queue. Experimental results show that the proposed algorithm gives better results than two other state-of-the-art methods. The hierarchical clustering tree in [4] has been further extended in [3] to build up multiple hierarchical clustering trees. The ANN search proceeds in parallel among the hierarchical clustering trees. Experimental results show that a significant improvement in the search performance is achieved and that the proposed algorithm can scale well to large-scale datasets.

Hashing approach: Locality-sensitive hashing (LHS) of Indyk and Motwani [9] has been known as one of the most popular hashing-based methods. The key idea is to project the data into a large number of random lines in such a way that the likelihood of hashing two data points into the same bucket is proportional to their similarity degree. Given a query, a proximity search is proceeded by first projecting the query using the LSH functions. The obtained indices are then used to access the appropriate buckets followed by a sequence search for the data points contained in the buckets. Given a sufficiently large number of hash tables, the LSH can perform an ANN search in sub-linear time complexity. Kulis and Grauman [10] extended the LSH to the case in which the similarity function is an arbitrary kernel function κ: $D(p,q) = \kappa(p,q) = \phi(p)^{\mathrm{T}}\phi(q)$, where $\phi(x)$ is an unknown embedding function. The main drawback of these two studies is the use of a very large memory space to construct the hash tables. To address this issue, Panigrahy [11] introduced an entropy-based LSH indexing technique. Its basic idea is quite interesting: given a query q and a parameter r of the distance from q to its nearest neighbour, the *synthetic* neighbours of q within a distance r are randomly generated. These synthetic neighbours are then hashed by using the LSH functions. The obtained hash keys are used to access the bucket candidates, where it is expected that the true nearest neighbour of q could be present. A detailed analysis of the entropy-based LSH algorithm was reported by Lv et al. [1]; this paper showed that the entropy-based LSH algorithm does not give a noticeable search improvement compared to the original LSH method. Lv et al. [1] proposed another approach, which is known as *multi-probe* LSH, to reduce the utilized hash tables. Its basic idea is to search multiple buckets, which probably contain the potential nearest neighbours of the query. The rationale is easily seen: if a data point p is close to another data point q but they are not hashed into the same bucket, then there is a high chance that they are hashed into two "close" buckets. Experimental results show a significant improvement in terms of the space efficiency, compared to the original LSH scheme. Recently, Aiger et al. [12] introduced a variation of the LSH scheme based on random grids. A dataset X, which consists of n D-dimensional vectors, is randomly rotated and shifted up to $e^{D/c}$ times where c is an approximate factor of search precision (*e.g.*, $c = 2$ in their experiments). For each rotation/translation, the corresponding dataset is partitioned using an uniform grid of cells where cell size $w = c/\sqrt{D}$ and the points contained in each cell are hashed into the same bucket. Consequently, space overhead is a big concern of this method (*i.e.*, $O(De^{D/c}n)$). For instance, if 128-dimensional SIFT features are used, the proposed method consumes a huge memory space of $O(128e^{64}n)$.

Hybrid approach: Recent interests in ANN search have been moved towards product quantization (PQ) which can be considered as a hybrid fashion of space-partitioning and clustering approaches. The crucial benefit of using PQ for ANN search is the capability of indexing extremely large-scale and high-dimensional datasets. Jégou et al. [13] introduced a PQ-based method whose basic idea is to uniformly decompose the original data space into distinct sub-spaces and then to create separately an optimized sub-codebook for the data in each subspace using K-means algorithm. Non exhaustive search is proceeded by employing an inverted file structure. However, as PQ employs an axis-aligned grid for space decomposition, many centroids are created without the support of data. This damages the search performance because it was widely agreed that better fitting to the underlying data is crucial for achieving good search speed and search accuracy. Ge et al. [14] extended PQ by introducing optimized product quantization (OPQ). The main spirit of OPQ is to design a quantizer that solves the quantization optimization problem in both aspects of space decomposition and sub-codebook construction. Doing so, OPQ allows the split grid to be rotated by arbitrary orientation to make better fitting of the underlying distribution. Although OPQ significantly outperforms PQ in search performance, such an alignment is less-efficient to the cases of multi-model distribution as discussed by Kalantidis and Avrithis [15]. To address this issue, Kalantidis and Avrithis [15] proposed optimizing locally the PQ per centroid for quantizing the residual distribution. However, learning locally an optimized PQ is not an efficient process if a non-parametric optimization fashion is used. Alternatively, a parametric learning technique can be applied that requires an assumption of some prior knowledge about the underlying distribution (*e.g.*, typically a Gaussian distribution). Consequently, search performance would be degraded if the assumption is not satisfied.

In this work, we conduct a detailed investigation of our previous work in [16] for indexing the feature vectors. Specifically, we have carried out the following extensions:

- We provide a deeper and wider review of related work by including recent progress for ANN search (*e.g.*, product quantization approaches).
- The spirit of the proposed approach is thoroughly presented with deeper analysis and illustration.
- Additional experiments (*e.g.*, more datasets) and an application to image retrieval are included for better evaluation of the proposed approach.
- A thorough study of parameter impact is also investigated coupling with the addition of an automatic parameter tuning algorithm to make the proposed indexing scheme well-adapted to a specific dataset and search precision.

For the remainder of this paper, Section 2 presents the proposed approach. Section 3 dedicates to performance evaluation. Section 4 concludes the paper and defines some future work.

2. The LM-tree indexing algorithm

2.1. Construction of the LM-tree

For a better presentation of our approach, we use the notation $\mathbf{p}$ component of $\mathbf{p}$ $(1 \leq i \leq D)$. We also denote $p = (p_{i_1}, p_{i_2})$ as a point in a $2D$ space. It is assumed that the Euclidean distance is employed in this work unless specifying otherwise. We adopted here the conclusion made in [2] about the use of PCA for aligning the data before constructing the LM-tree. This approach enables us to partition the data via the narrowest directions. In particular, the dataset X is translated to its centroid following a step of data rotation to make the coordinate axes aligned with the principal axes. Note that no dimension reduction is performed in this step. In fact, PCA analysis is used only to align the data. Next, the LM-tree is constructed by recursively partitioning the dataset X into m roughly equal-sized subsets as follows:

- Sort the axes in decreasing order of variance, and choose randomly two axes, i_1 and i_2, from the first L highest variance axes ($L < D$).

Download English Version:

https://daneshyari.com/en/article/536284

Download Persian Version:

https://daneshyari.com/article/536284

Daneshyari.com