Nearest Neighbors Search Algorithm for High Dimensional Data

G. Vasanthi, Assistant Professor/IT, Department of Information Technology, Mailam Engineering College, Mailam.
E-mail: vasanthiit@mailamengg.com

Dr.S. Artheeswari, Professor/IT, Department of Information Technology, Mailam Engineering College, Mailam.
E-mail: artheeswariit@mailamengg.com

Dr.S. Kalaivany, Professor/IT, Department of Information Technology, Mailam Engineering College, Mailam.
E-mail: kalaivanyit@mailamengg.com

Abstract--- Machine learning and Computational geometry offer new methods for search, regression, and classification with large amounts of high-dimensional data. The *k*-nearest neighbors (*k*-NN) algorithm is used machine learning method that finds nearest neighbors. Several data structures that solve efficiently the problem of Nearest Neighbour search in higher dimensions. Distinctive image features from scale invariant presents a pattern matching method based on fast Nearest Neighbour search algorithms.[2] The user is not interested to find one closest point. He/She wants to find k closest points. The naïve way to solve the Nearest Neighbour search problem is to use linear search. For the k Nearest Neighbour search, the naïve algorithm consists in sorting the points according to their distance from the query point. For matching high dimensional features, we find two algorithms to be the most efficient: the randomized k-d Tree, the priority search k-means tree. These data structures are used to find the Nearest Neighbour search.[1]

Keywords--- NN (Nearest Neighbour), k-d (k-Dimensional).

I. Introduction

The *k*-nearest neighbor (*k*-NN) algorithm is widely used in many areas such as pattern recognition, machine learning, and data mining. Applications that require frequent k-NN queries from large data sets are for example object recognition, shape recognition and image completion using large databases of image descriptors, and content-based web recommendation systems [5].

1.1 Approximate Nearest Neighbor Search

A metric space M, a data set X = (x1. . . xn) ⊆ M, a query point q ∈ M, and a distance metric m : M2 → R. The k-nearest neighbor (k-NN) search is the task of finding the k closest (w.r.t. m) points to q from the data set X, i.e., find a set K ⊆ X for which it holds that |K| = k and

$m(q, x) \leq m(q, y)$

for all x ∈ K, y ∈ X \ K.

We consider applications where M is the d dimensional Euclidean space R^d, and m is Euclidean distance

$$m(x, y) = \sqrt{\sum_{i=1}^{d}(xi - yi)2}$$

Nearest Neighbor search algorithm can be divided into an offline phase, and an online phase[1]. In offline phase build an index, and it is used for online phase to completing fast nearest neighbor queries. In practical applications, the most critical considerations are the accuracy of the approximation (the definition of the accuracy and the required accuracy level depending on the application), and the query time needed to reach it. As a measure of accuracy, we use recall, which is relevant especially for information retrieval and recommendation settings. It is defined as the proportion of true k-nearest neighbors of the query point returned by the algorithm:

$$\text{Recall} = \frac{|A \cap K|}{k}$$

Where A is a set of k approximate nearest neighbors, and K is the set of true k nearest neighbors.

1.2 Randomized K-d Tree

The Randomized k-d Tree algorithm is an approximate nearest neighbor algorithm. It consists of several Randomized K-d Trees searched in parallel.

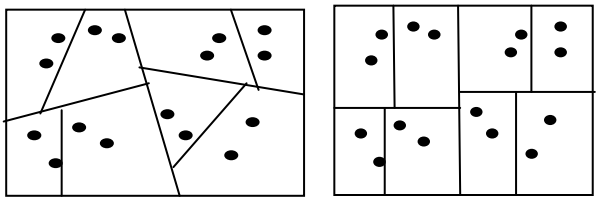
DOI: 10.5373/JARDCS/V12SP8/20202636
ISSN 1943-023X
Received: 19 June 2020/Accepted: 20 July 2020

1.2.1 Three randomized tree structures for exact NN search

Based on a set of data points k-d tree is a partition of R^d into hyper-rectangular cells. In the entire space the root of the tree is a single cell. the cell is split at the median of the data with a coordinated direction, the process is then recursed on the two newly created cells, and continues until all leaf cells contain at most some predetermined number n_o of points. The depth of the tree is at most about $\log(n/n_o)$, where n is the data points[6].

1.2.2 Randomized Trees for NN Search

K-d Tree Randomized K-d Tree



Algorithm 1

1: function GROW TREES(X, T, l, a)
2: n ← X.nrows
3: let trees[1 . . . T] be a new array
4: for t in 1, . . . , T do
5: let R be a new d × l matrix
6: for level in 1, . . . , l do
7: for i in 1, . . . , d do
8: generate z from Bernoulli(a)
9: if z = 1 then
10: generate R[i, level] from N(0, 1)
11: else
12: R[i, level] ← 0
13: P ← XR
14: trees[t].root ← GROW TREE(X, [1 . . . n], 0, P)
15: trees[t].random matrix ← R
16: return trees

1.3 Priority Search K-Means Tree

Priority Search K-Means Tree measures the effect changing the parameters on the precision and the performance of Priority K-Means Tree[5].

Assume there are *n* objects p_i for ($1 \le i \le n$) in a training dataset and each object p_i has *m* dimensions. We present a new algorithm *k*M*k* NN (*k*-Means for *k*-Nearest Neighbors) that searches for exact *k* nearest neighbors to a query object *q*, shown as follows:

Algorithm 2

The *k*M*k*NN algorithm

The BUILDUP stage

Input: *n* training objects p_i for ($1 \le i \le n$); each with *m* dimensions.

Output: *kc* cluster centers c_j for ($1 \le j \le kc$) with the assignment of each object p_i to its nearest cluster.

1. Set kc=s√n , where $s > 0$.
2. Classify the *n* objects into *kc* clusters using a *k*-means clustering algorithm.
3. Calculate and record the distance d_{ij} from each object p_i to its nearest cluster center c_j.
4. For each cluster center c_j, sort the distances d_{ij} in descending order for all the training objects associated to c_j.

DOI: 10.5373/JARDCS/V12SP8/20202636
ISSN 1943-023X

Received: 19 June 2020/Accepted: 20 July 2020

The SEARCHING stage

Input: *kc* cluster centers c_j for ($1 \leq j \leq kc$) with the assignments of each object p_i to its nearest center c_j; a query object q; the number of nearest neighbors k.

Output: A set of k nearest training objects to q.

1. Initialize the set of k nearest objects of q by using a maximum distance as the k distances. Set the maximum distance in the set of k current nearest objects to $d_{\max}$.
2. Calculate the distances $\|q - c_j\|$ for all cluster centers c_j that ($1 \leq j \leq kc$) and sort the distances in ascending order.
3. For each cluster c_j from the nearest to the farthest to q, do the step 4.
4. For each object p_i in the cluster c_j from the farthest to the nearest to the cluster center c_j:

if $d_{\max} \leq \|q - c_j\| - \|p_i - c_j\|$, go to step 3 to visit the next cluster;

Otherwise, calculate the distance $\|q - p_i\|$ from q to p_i and if $d_{\max} > \|q - p_i\|$, remove the object with distance $d_{\max}$ from the set of k current nearest objects and insert p_i with the distance $\|q - p_i\|$; update $d_{\max}$

II. General Comparison of the Algorithm

The general comparison of the search and construction algorithms for each data structure as follows. The K-d construction of the k-d Tree is obviously the fastest. However, the performance of construction of the Randomized K-d Tree depends on the number of trees, in this case it is ten. Increasing this number will affect negatively the construction time performance but it will improve the search performance. Concerning the search algorithms performance, the K-d Tree and the linear search have almost the same performance because of the dimensionality of the data. The increase of the dimensionality of the data the performance of the K-d Tree decreases. However, the Randomized K-d Tree has the best performance in term of accuracy and speed, and then it comes the Priority K-Means Tree. Furthermore, increasing epsilon from 0.3 to 0.7 increases the precision from 87% to 99% for the Randomized K-tree, and increases the precision from 70 to 80% for the Priority K-Means Tree. In addition, increasing the K parameter for the Priority K-Means tree from 4 to 16 with epsilon fixed to 0.7 increases the precision from 80 % to 99%.

III. Conclusion

In conclusion, Nearest Neighbour Search is one of the most important problems in many fields. Solving this problem in low dimensions can be done efficiently using K-d Tree.[7] However, increasing the dimensionality of this data renders this data structure ineffective. Then the use of different data structure such as Randomized K-d Tree and Priority K-Means Tree becomes essential. But these data structures are affected by many parameters; therefore this project aims to discover the effects of these parameters and to provide the following guidelines to a better use of the data structures. A common observation for the two data structures is increasing epsilon always yields better results this means that we are almost very close to the correct answer by a factor of two in the worst case.

For the Randomized K-d Tree,

- Increasing the number of trees will lead to a better performance. However, after a certain threshold, increasing the number of trees becomes invective this is due to the increase in the use of memory [2].
- Increasing the number of visited leaves will lead to a better precision without a serious effect on the performance.

For the Priority K-Means Tree,

- Increasing the branching factor K affects slightly the performance.
- Increasing the Branching factor improves the precision.
- Increasing the number of collecting points L leads to a significant improvement in the precision with no significant effects on the precision.
- The number of iteration for the K-Means algorithm has no notable effect.

DOI: 10.5373/JARDCS/V12SP8/20202636
ISSN 1943-023X

Received: 19 June 2020/Accepted: 20 July 2020

References

[1] Muja, M., & Lowe, D.G. (2014). Scalable nearest neighbor algorithms for high dimensional data. *IEEE transactions on pattern analysis and machine intelligence*, 36(11), 2227-2240.

[2] Silpa-Anan, C., & Hartley, R. (2008, June). Optimised KD-trees for fast image descriptor matching. *In 2008 IEEE Conference on Computer Vision and Pattern Recognition* (pp. 1-8). IEEE.

[3] T. Liu, A. Moore, A. Gray, K. Yang, "An investigation of practical approximate nearest neighbor algorithms," *presented at the Advances in Neural Information Processing Systems, Vancouver, BC, Canada*, 2004.

[4] Jegou, H., Douze, M., & Schmid, C. (2010). Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1), 117-128.

[5] Wang, J., Wang, J., Zeng, G., Tu, Z., Gan, R., & Li, S. (2012, June). Scalable k-nn graph construction for visual descriptors. *In 2012 IEEE Conference on Computer Vision and Pattern Recognition* (pp. 1106-1113). IEEE.

[6] Muja, M. (2013). *Scalable nearest neighbour methods for high dimensional data* (Doctoral dissertation, University of British Columbia).

[7] Aly, M., Munich, M., & Perona, P. (2011, August). Distributed kd-trees for retrieval from very large image collections. In Proceedings of the British machine vision conference (BMVC) (Vol. 17).

[8] S. Artheeswari, "Enhancing the Efficiency of Artificial Bee Colony (ABC) Algorithm Using Genetic Operations", Fronteiras: *Journal of Social, Technological and Environmental Science* V.6, N.2, p. 396-401.

[9] S. Artheeswari, "Securing the data using ABC Algorithm and Secure Multi-Party Computation Protocol in Cloud. *International Journal of Scientific Research and Review*, Volume 7, Issue 3.

[10] M. Ramalingam and R.M.S. Parvathi, "Policy-Based Semantic Access Control Framework for Fine-Grained Access in Semantic Web Services". *European Journal of Scientific Research*, Vol.74, No.1, pp. 154-163, 2012

[11] M. Ramalingam and R.M.S. Parvathi. "Secure Semantic Aware Middleware: a Security Based Semantic Access Control for Web Services". *International Review on Computers and Software*, Vol. 8, No.9, pp. 2136-2141, September 2013.

DOI: 10.5373/JARDCS/V12SP8/20202636
ISSN 1943-023X

Received: 19 June 2020/Accepted: 20 July 2020