Nearest Neighbor Search Data Structure

Sanjoy Dasgupta
Department of Computer Science
University of California, San Diego
dasgupta@cs.ucsd.edu

Zhen Zhai
Department of Computer Science
University of California, San Diego
zzhai@ucsd.edu

Eugene Che
Department of Computer Science
University of California, San Diego
eche@ucsd.edu

Abstract

Nearest-neighbor(NN) search is boardly used within all different fields of study to gain information on new data from training data set. For NN search, the more complex the training data set is the more accurate the result will be. However, doing NN search on complex and large training data sets is time consuming. Therefore, improving the speed and acuracy of nearest-neighbor search becomes essential. We look at different data structures for NN search and compare the results on varied data sets. We focus on data structures including KD tree and PCA tree. We also look at KD spill tree, KD virtual spill tree, and PCA spill tree to explore about the technique of spilling and virtual spilling. We conclude that spilling and virtual spilling improve the performance of the data structures.

1 Introduction

Gaining information of an input query by looking for the most similar match from the training set is referred as nearest neighbor(NN) search. We are given a data set $\mathbf{S}$ that contains a set of points $\{s_1, s_2, s_3, ..., s_n\}$ and an input query q, and we are looking for the nearest neighbor s_i of q in $\mathbf{S}$ such that s_i can provide valuable information about input query q.

A lot of machine learning algorithms spend a big amount of time on nearest neighbor search. The reason that NN search is time consuming is that the training set could very well be represented by high dimentional vectors. In fact, the accuracy of the NN search result is proportional with the complexity of the training set. Therefore, speeding up NN search can lead to tremoudous improvment to many applications of not only machine learning but also data analysis, bio-informatics, signal processing, etc.

There has been a large number of research devoting to speed up the performance of NN search. In this paper, we evaluate two most promising data structures, KD tree and PCA tree. We conclude that PCA tree has better performance than KD tree.

We also experiment the technique of spilling by looking at KD spill tree vs. KD tree and PCA spill tree vs. PCA tree. We observe that spilling gives a better result.

We finally experiment the technique of virtual spilling by looking at KD virtual spill tree. We compare it with KD spill tree and conclude that virtual spilling improve the performance of spilling.

We apply these data structures on five different data sets from different area of study including computer vision, human relationship... We focus on the accuracy of NN search using these data structures and make sure it generally apply to all different kinds of data sets. In conclusion, we observe the following: PCA tree has a better performance comparing to KD tree, spilling improve the performance, and virtual spilling improve the accuracy slightly more than spilling.

2 KD tree vs. PCA tree

We focus our study on binary space partitioning(BSP) trees, one of the most promising data structure for NN search. We first look at k-dimentional(KD) trees. Then we study principle component analysis(PCA) tree, in which we preprocess the data using PCA(Pearson, 1901) before partitioning the space.

2.1 KD trees

KD tree is a BSP tree that partition data points into a k dimentional binary tree(Bentley, 1975). Starting from the root of the tree, a cooridnate with biggest variance is chosen. The data points **S** from root are split at the median of the coordinate into two children of the root. Recursively apply the process on the children until the leaf of the tree contain at most $\mathbf{n}_0$ data points.

Given KD tree T and query q, simply trace q down the tree and find the leaf l it falls in. Return all the possible nearest neighbors within leaf l. Then do distance calculation for each of the data point in leaf and return the point that lies closest to query q.

Pseudocode:

function BuildTree(S)
  If $|S| < n_0$:
    return S
  Choose the coordinate c with max variance
  Split S into L and R:
    $m = median\{x_c : x \in S\}$
    $L = \{x \in S : x_c \leq m\}$
    $R = \{x \in S : x_c > m\}$
  $L = BuildTree(L)$
  $R = BuildTree(R)$
  return (c, m, L, R)

function SearchTree(T)
  If T is leaf
    return $NN(q, T)$
  If $q[c] < m$
    return $SearchTree(L)$
  Else
    return $SearchTree(R)$

function NN(q, S)
  brute-force search closest data d of q
  return d

The problem with KD tree is that the true nearest neighbor n of query q could be in a leaf that is different from the leaf q falls in. This could happen easily when q and n are both located close to the boundary of the tree (Figure 1). Therefore, the error rate could be very high.

2.2 PCA tree

PCA tree is also a BSP tree which partition data after a process of PCA on the data points. From the root of the tree, we do PCA and find the first principle component **v**. We then reduce the dimension of the data by mapping all the data points onto **v** which gives us a one dimensional data set s. The original data points from root are partition using the corresponding data in s and split at the median of s.
Similar to KD tree, searching NN of a query q can be done by tracing q down the tree.

Pseudocode:

function BuildTree(S)
  If $|S| < n_0$
    return S
  Run PCA and find the first principal component $\mathbf{v} \in \mathbb{R}^d$
  $m =$ median $\{x \cdot c : x \in S\}$
  Split S into L and R:
    $L = \{x : x \cdot c \leq m\}$

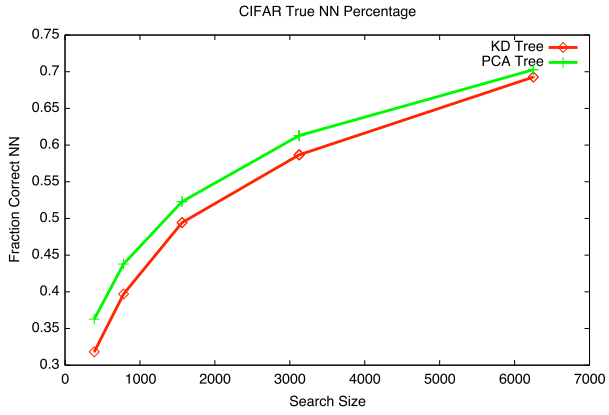
$R = \{x : x \cdot c > m\}$
$L = BuildTree(L, n)$
$R = BuildTree(R, n)$
return (c, m, L, R)

function SearchTree(T)
If T is leaf
return $NN(q, T)$
$q_c = q \cdot c$
If $q_c < m$
return $SearchTree(L)$
Else
return $SearchTree(R)$

PCA tree has the similar problem with KD tree that q could falls into different leaf. However, because of the PCA process, the probability of q falling into different leaf is smaller. (EXPLAIN HERE)

2.3 Experimental results

We examine the performance of KD tree and PCA tree with five different data sets. (WRITE ABOUT EXPERIMENT DATA SET HERE)



3 Spilling or not

We explore the technique of spilling on both KD tree and PCA tree. Experiment supports that spilling improve the accuracy of NN search.

3.1 Spilling

Spilling is a technique used when building a tree with spill factor α. Starting from the root of the tree, a cooridnate c is chosen. The data points **S** from root are split at the median of the coordinate into two children of the root. Some data points which fall into the range of the spilling factor α are placed in both of the children. Recursively apply the process on the children until the leaf of the tree contain at most **n** data points. The search function is same as previous stated.

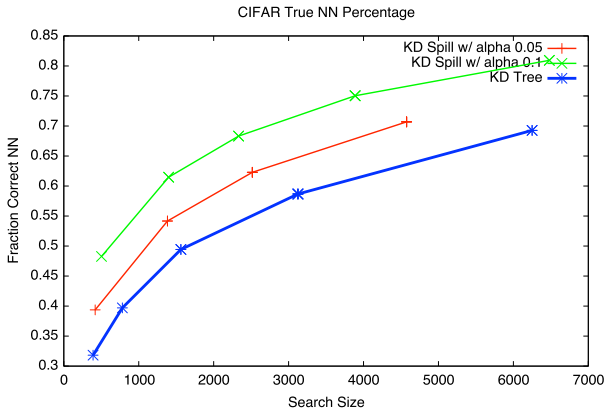
Pseudocode of KD Spill Tree:
function BuildTree(S)
If $|S| < n_0$

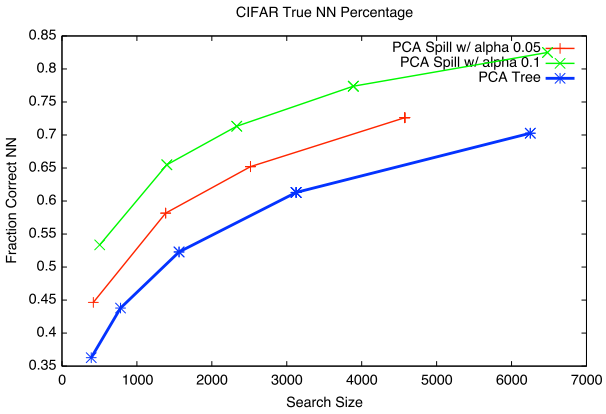
return S
Choose the coordinate c with max variance
Split S into L and R:
$m = median\{x_c : x \in S\}$
$L = \{x \in S : x_c \leq m\}$
$R = \{x \in S : x_c > m\}$
Find spill range (r_1, r_2):
$r_1 = (0.5 - \alpha) quantile\{x_1 : x \in S\}$
$r_2 = (0.5 + \alpha) quantile\{x_2 : x \in S\}$
If $\{d \in S : r_1 < d < r_2\}$:
$L.add(d)$
$R.add(d)$
$LeftTree = BuildTree(L)$
$RightTree = BuildTree(R)$
return (c, m)

We try to increase the accuracy by examing through the data which lie close to the boundary. The distance from the boundary is determined by the spill factor α. The bigger the α the more the data we need to examine. To maintain the efficiency of KD tree algorithm, we keep the value of α small and we compare performance of different values of α.

3.2 Experimental results

In the following experiments, we apply two different values of α: 0.05 and 0.1. We compare the performance of KD tree vs. KD spill tree and PCA tree vs. PCA spill tree to evaluate the improvement of spilling.





4 Virtual spill tree vs. spill tree

We now look at a revised spilling technique, virtual spilling. We compare the performance of KD virtual spill tree with KD spill tree.

4.1 Virtual spill tree

Virtual spilling is a technique used in the searching process of the tree. Given KD tree T and query q, trace q down the tree and find the leaf(ves) l it falls in. If q fall into the range of the spill factor α of a node, trace both of the children of the node. Return all the possible nearest neighbors nn within the leaf(ves) l. Then do distance calculation for each of the data point in each leaf and return the point that lies closest to query q among all leaves.

Pseudocode:

function SearchTree(T)

If T is leaf

return $NN(q, T)$

Find spill range (r_1, r_2):

$r_1 = (0.5 - \alpha) quantile\{x_1 : x \in S\}$

$r_2 = (0.5 + \alpha) quantile\{x_2 : x \in S\}$

If $r_1 < q[c] < r_2$

$nn.add(SearchTree(L))$

$nn.add(SearchTree(R))$

Else if $q[c] < m$

$nn.add(SearchTree(L))$

Else if $q[c] > m$

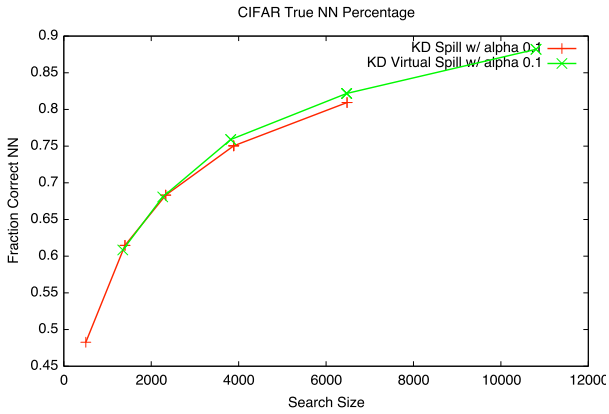
$nn.add(SearchTree(R))$

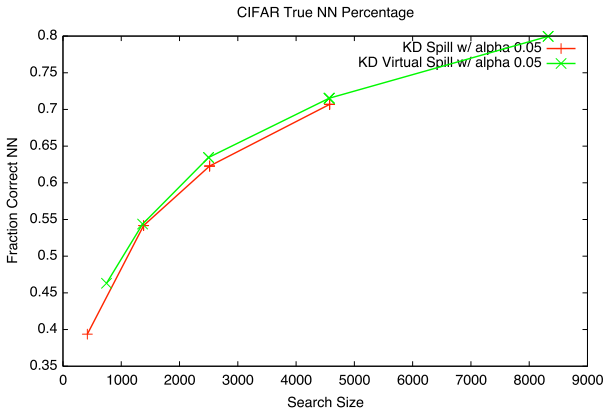
return $NN(q, nn)$

Virtual spill tree is also aim at increasing the accuracy by examing datas lie near boundary. However, unlike spill tree, where α determine the range of examination, virtual spill tree examine the adjacent leaves if the query falls into the spilling range which is determined by α.

4.2 Experimental results

Similar to the experiment on spill tree, we focus on α being 0.05 and 0.1. We see that the accuracy is increased by using the virtual spilling technique on KD tree.





5 Conclusion