1. Introduction

This supplementary material is organized as follows. Algorithms for query processing for defeatist search with auxiliary information as well as guided prioritized search are presented in section 2. In section 3, we provide proofs of theorems presented in the paper. Additional experiments are presented in section 4.

2. Query processing

In this section, we present algorithms for query processing for defeatist search with auxiliary information, guided prioritized search and the combined approach. These algorithms are presented in Algorithm 2, 3 and 4. For easy reference we provide an RPT construction algorithm with auxiliary information as well.

3. Proof of theorems

Proofs of all theorems presented in submitted manuscript are provided below.

3.1. Proof of theorem 1

Proof. For any \(x_i \in S\), using lemma 1 of (Li & Malik, 2016), we get,

\[
\Pr\left(\left|U^T (q - x_i)\right| \leq \left|U^T (q - x_1)\right| \right) \leq 1 - \frac{2}{\pi} \arccos \left(\frac{\|q - x_i\|_2}{\|q - x_1\|_2}\right).
\]

Noting that for any \(z\), \(\arccos(z) = \frac{\pi}{2} - \arcsin(z)\), and the inequality \(\theta \geq \sin \theta \geq \frac{2\theta}{\pi}\), for \(0 \leq \theta \leq \frac{\pi}{2}\), we get

\[
\Pr\left(\left|U^T (q - x_i)\right| \leq \left|U^T (q - x_1)\right|\right) \leq \frac{\|q - x_i\|_2}{\|q - x_1\|_2}.
\]

Let \(Z_i\) be an indicator variable that takes value 1 if \(\left|U^T (q - x_i)\right| \leq \left|U^T (q - x_1)\right|\), and 0 otherwise. Then \(E(Z_i) \leq \frac{\|q - x_i\|_2}{\|q - x_1\|_2}\). Let \(Z = \sum_{i=1}^{S} Z_i\). Then \(Z\) indicates the number of points in \(S\) whose distance from \(q\) upon projection is smaller than \(\left|U^T (q - x_1)\right|\). Using Markov’s inequality,

\[
\Pr(Z > k) \leq \frac{E(Z)}{k} = \frac{\sum_{i=1}^{S} E(Z_i)}{k} \leq \frac{1}{k} \sum_{i=1}^{S} \frac{\|q - x_i\|_2}{\|q - x_1\|_2}.
\]

3.2. Proof of theorem 2

Proof. Let \(c = kn_0\). Since we are using median split, it is easy to see that exactly \(\left[\frac{S}{2}\right]\) levels from the leaf node level (and excluding leaf node level) will have less than \(c\) points on each side of the median. Note that at the leaf node level, we have at most \(\left\lceil \frac{n}{n_0} \right\rceil\) nodes. Now consider the level just above the leaf node level. Total number of nodes at this level is at most \(\frac{1}{2} \cdot \left\lceil \frac{n}{n_0} \right\rceil\) and on each side of the median we have at most \(n_0\) points. Since \(n_0 < c\), on each side of the median, for each node at this level, will store a matrix of size \(n_0 \times (m + 1)\) matrix \((n_0 \times m\) matrix for \(m\) dimensional representation of \(n_0\) points and additional \(n_0 \times 1\) space for storing index of these \(n_0\) points). If we further go one level up, maximum number of nodes at this level...
**Algorithm 2** Query processing using defeatist search with auxiliary information

**Input:** RP tree constructed using Algorithm 1 from main paper, \( m \) independent random vectors \( \{V_1, \ldots, V_m\} \) sampled uniformly from \( S^{d-1} \), query \( q \), number of candidate neighbors at each node \( c' \).

**Output:** Candidate nearest neighbors

1: Set \( C_q = \emptyset \).
2: Set \( \tilde{q} = (V_1^T q, V_2^T q, \ldots, V_m^T q) \)
3: Set current_node to be root node of the input tree.
4: while current_node \( \neq \) leaf_node do
5:   if \( U^T \tilde{q} < v \) then
6:     \( A = R_{\text{cnn}} \)
7:     ai = air
8:     current_node = current_node.left
9:   else
10:     \( A = L_{\text{cnn}} \)
11:     ai = a\( \tilde{r} \)
12:     current_node = current_node.right
13: end if
14: Sort the rows of \( A \) in increasing order according to their distance from \( \tilde{q} \) and let array \( a \) contains these sorted indices.
15: \( C_q = C_q \cup \{ai(1), ai(2), \ldots, ai(c')\} \)
16: end while
17: Set \( \text{leaf}_q \) to be the indices of points in \( S \) that lie in leaf_node.
18: \( C_q = C_q \cup \text{leaf}_q \)
19: return \( C_q \)

Supplementary material for improved nearest neighbor search using auxiliary information and priority functions

**Algorithm 3** Query processing using prioritized guided search

**Input:** RP tree constructed using Algorithm 1 from main paper, query \( q \), number of iterations \( t \)

**Output:** Candidate nearest neighbors

1: Set \( C_q = \emptyset \).
2: Set \( P \) to be an empty priority queue.
3: Set current_node to be root node of the input tree.
4: if \( t > 0 \) then
5:   while current_node \( \neq \) leaf_node do
6:     if \( U^T \tilde{q} < v \) then
7:       current_node = current_node.left
8:     else
9:       current_node = current_node.right
10: end if
11: Compute priority_value
12: \( P.\text{insert}(\text{current_node.priority_value}) \).
13: end while
14: Set \( \text{leaf}_q \) to be the indices of points in \( S \) that lie in leaf_node.
15: \( C_q = C_q \cup \text{leaf}_q \)
16: Set \( t = t - 1 \)
17: Set current_node = \( P.\text{extract_max} \)
18: end if
19: return \( C_q \)

Space required to store this auxiliary information is,

\[
2(m + 1)\left(\frac{\log(\frac{m}{n_0}) - \left(\left\lfloor \frac{k}{2}\right\rfloor + 1\right)}{2^t}\right)
\]

\[
= 2(m + 1)c\left(\frac{2\log(\frac{m}{n_0}) - \left(\left\lfloor \frac{k}{2}\right\rfloor + 1\right)}{2^t - 2}\right)
\]

\[
\leq 2(m + 1)c\left(\frac{1}{2^{t/2}} + \frac{n_0}{2^k}\right)
\]

Summing these two terms, additional space requirement is,

\[
(m + 1)n\left(\left\lfloor \frac{k}{2}\right\rfloor + \frac{2k}{2^{t/2}}\right) \leq 6(m + 1)n
\]

where the last inequality follows from the fact that \( \left\lfloor \frac{k}{2}\right\rfloor + \frac{2k}{2^{t/2}} \leq 6 \) for all \( k \leq 10 \). In addition, we also need to store \( m \) random projection directions for the entire tree requiring extra \( md \) space. Therefore, total additional space requirement for storing auxiliary information is at most \( 6(m + 1)n + md \leq (m + 1)(6n + d) \).

Now, note that we want to choose number of projection directions \( m \) in such a way that for all auxiliary data points
Algorithm 4 Query processing using combined approach with auxiliary information

Input: RP tree constructed using Algorithm 1, $m$ independent random vectors $\{V_1, \ldots, V_m\}$ sampled uniformly from $S^{d-1}$, query $q$, number of iterations $t$, number of candidate neighbors at each node $c'$.

Output: Candidate nearest neighbors

1. Set $C_q = \emptyset$, set $B$ to be an empty binary search tree
2. Set $P$ to be an empty priority queue, set count $= 0$
3. Set $\tilde{q} = (V_1', q, V_2', q, \ldots, V_m', q)$
4. Set current node to be root node of the input tree.
5. if $t > 0$ then
6. while current node $\neq$ leaf node do
7. if $U^T q < v$ then
8. $A = R_{\text{can}}, a_1 = a_{\text{root}}$
9. current node = current node.left
else
10. $A = L_{\text{can}}, a_1 = a_{\text{root}}$
11. current node = current node.right
12. end if
13. Compute priority value, set count $= count + 1$
14. Sort the rows of $A$ in increasing order according to their distance from $\tilde{q}$ and let array $a$ contains these sorted indices.
15. Insert $\{a_i(a(1)), \ldots, a_i(a(c'))\}$ into $B$ with value count
16. Set struct = (count, current node)
17. $P$.insert(struct, priority value).
18. end while
19. Set leaf node to be the indices of points in $S$ that lie in leaf node.
20. Set $C_q = C_q \cup \text{leaf node}, t = t - 1$
21. Set current node = ($P$.extract_max, current node)
22. Delete from $B$ candidate set with value ($P$.extract_max).count
23. end if
24. return $C_q \cup \{\text{all candidate set from } B\}$

Total number of nodes is,

$$2^{k-1} = 1 \cdot \frac{1}{2} \cdot 2 \log \frac{2}{1+\epsilon} (\frac{n}{n_0})$$

Due to median split, total number of internal nodes will be exactly one less than the number of leaf nodes, moreover, we need to store a random projection direction at each of these internal nodes. Therefore, for fixed $n_0$, space complexity of spill tree is $O \left( \frac{1}{1-\log(1+2\epsilon)} \right)$.

4. Additional experiments

4.1. 1-NN search experiments

In the following we present experimental evaluations for Experiment 3 of the submitted manuscript. Here, we presents for 1-NN search results in Figure 1 and 2. We note that we observe similar trend that we observed for 10-NN search in the submitted manuscript.

References

Figure 1. 1-NN accuracy for all six datasets. The x-axis represents the ratio of # of retrieved points to the total number of instances. The markers from left to right corresponding to 2, 5, 10, 15 and 20 iterations (for combined and Multi-Combined method) / trees (for normal RPT).

Figure 2. Rank error (1-NN) for all six datasets. The x-axis represents the ratio of # of retrieved points to the total number of instances. The markers from left to right corresponding to 2, 5, 10, 15 and 20 iterations (for combined and Multi-Combined method) / trees (for normal RPT).