Fast Nearest Neighbor Search in High Dimensions by Multiple Random Projection Trees

Teemu Roos
PhD, Associate Professor
Department of Computer Science
University of Helsinki
**k-Nearest Neighbor Search**

Given a set of $n$ data points ($d$-dimensional vectors), $X \in \mathbb{R}^{d \times n}$, and a query point, $q$, retrieve the $k$ data points nearest to $q$.

Naive solution: evaluate $d(x, q)$ for all $x \in X$. Complexity $\mathcal{O}(dn)$. 
**k-Nearest Neighbor Search**

Given a set of \( n \) data points (\( d \)-dimensional vectors), \( X \in \mathbb{R}^{d \times n} \), and a query point, \( q \), retrieve the \( k \) data points nearest to \( q \).

Naive solution: evaluate \( d(x, q) \) for all \( x \in X \). Complexity \( \mathcal{O}(dn) \).

Speed up strategy: Narrow down the set of candidates to a small subset, \( S \), and evaluate \( d(x, q) \) for all \( x \in S \).

When the number of queries on the same data set is large, we can first construct an index that provides the subset \( S \).

Approximate nearest neighbors: Subset \( S \) may not contain all \( k \) nearest neighbors.
Example applications: Clustering

RP–PCA Class labels

RP–PCA Clustering result

(bacterial genome data, \( n = 3105, d = 380601 \))
Example applications: Visualization

ISOMAP, t-SNE, …
Example applications: Information retrieval

The Kvasir application uses RP-trees to find suggestions of similar content on the Internet.
http://www.cs.helsinki.fi/u/lxwang/kvasir/

semifinalist in Cambridge University Business Plan Competition!
Random Projection Trees

error: NN not found
Random Projection Trees

buildRPTree($X, n_0$):
  if $|X| \leq n_0$:
    return leaf node $X$
  draw $r \in \mathbb{R}^d$ uniformly at random s.t. $\|r\|_2 = 1$
  project data to $rX$
  splitpoint $\leftarrow$ median of $\{rX_i\}$
  left $\leftarrow$ buildRPTree($\{X_i \in X: rX_i < \text{splitpoint}\}, n_0$)
  right $\leftarrow$ buildRPTree($\{X_i \in X: rX_i \geq \text{splitpoint}\}, n_0$)
  return $(r, \text{splitpoint, left, right})$

queryRPTree($node, q$):
  if $node$ is a leaf node:
    use brute force to find neighbors of $q$ in $node$
  else:
    if $rq < \text{splitpoint}$:
      queryRPTree($node$.$left, q$)
    else:
      queryRPTree($node$.$right, q$)
Random Projection Trees

By splitting at the median until the number of points is less than $n_0$, the depth of the tree becomes $\lceil \log_2 (n/n_0) \rceil$.

The time complexity of the tree building stage is $O(dn \log_2 (n/n_0))$.

Time complexity of the query stage is $O(d \log_2 (n/n_0) + dn_0)$ which is much faster than brute force $O(dn)$ if $n_0 \ll n$.

However, the probability of not finding a nearest neighbor increases as $n_0$ is decreased.
A similar index structure based on random projections can be constructed using **Locality-Sensitive Hashing (LSH)**.

**Idea (RP-based LSH):**
1. Project all data points onto $K$ random vectors.
2. Split them in half (e.g. at the median) independently wrt. each projection.
3. Encode splits as a $K$-bit hash vector.
4. Given a query $q$, search for neighbors that are mapped to the same “bucket” (hash vector).

**Consequence of choosing split points independently:**
No way to guarantee that buckets are even roughly of even size.
Multiple Random Projection Trees: Idea

Any RP tree may fail to find a given $k$-nearest neighbor.

Solution: Try $T$ times with different random projections.

Time complexity:
- tree building $O(Tdn \log_2 (n/n_0))$
- query $O(Td \log_2 (n/n_0) + Tdn_0)$

**KEY IDEA OF THIS TALK (KITT):**
with $n_0 = N/T$, for some fixed $N$, the query complexity is $O(Td \log_2 (Tn/N) + dN)$

For a moderate number of trees and large enough $N$, we have $T \log_2 (Tn/N) < N$, and the $O(dN)$ term dominates.
Performance of the MRPT algorithm

The accuracy increases when the number of trees built $T$ is increased while maximum leaf size is $n_0$ proportionately decreased, so that the size of the final search set $S$ stays approximately constant!

Multiple Random Projection Trees: Idea

An illustration of KITT: $n_0 = N/T$

$N = 400$

One tree approach

$T = 1, n_0 = 400$

$T n_0 = 1 \times 400 = 400$

Multiple tree approach

$T = 2, n_0 = 200$

$T n_0 = 2 \times 200 = 400$
Multiple Random Projection Trees: Idea
Multiple Random Projection Trees: Idea
Multiple Random Projection Trees: Results

So far we have only argued that the MRPT approach (with KITT) doesn’t significantly affect the query time.

But there is more…

**KEY OBSERVATION OF THIS TALK (KOTT):**
The MRPT approach (with KITT) leads to significantly *better accuracy* in finding the $k$-nearest neighbors.
Effect of the number of trees when the total search set size, $N = T n_0$, is fixed (0–10% of the data). (Wikipedia data, $n = 500,000$, $d = 1000$)
Multiple Random Projection Trees: Results

Better recall/time tradeoff than existing state-of-the-art.

News, $k = 10$, $n = 262144$, $d = 1000$
Almost opposite of RP-LSH:
- with MRPT, large $T$ and small $n_0$: sufficient that $x$ is in any of many small subsets $S$.
- with LSH, each bit corresponds a subset $|S| = n/2$: necessary that $x$ is on the same side of all splits.

Flexibility of MRPT wrt. choosing $n_0$ and $T$ (with $n_0 \propto 1/T$) allows optimization of accuracy vs speed.
So far we have:

• argued that the MRPT approach (with KITT) doesn’t significantly affect the query time compared to a single tree.

• shown that the MRPT approach (with KITT) leads to KOTT: *significantly better accuracy* in finding the $k$-nearest neighbors than a single tree (or virtual spill tree).

But, again, there is more…
Multiple Random Projection Trees: Implementation

Speed-up and compression:

1. MRPT can be easily and efficiently parallelized (different trees on different servers) giving almost linear speed-up except for small problems with response times in ms.

2. The projections can be computed separately and element-by-element, hence requiring minimal space complexity.

3. The index has a negligible memory footprint (sparsity, storing random seeds instead of projection directions).
Next Steps

What we don’t know yet:

1. Deeper theoretical understanding:
   What a priori performance guarantees can be given?

\[
\Pr_U(y \cdot U \text{ falls (strictly) between } q \cdot U \text{ and } x \cdot U) = \frac{1}{\pi} \arcsin \left(\frac{\|q - x\|}{\|q - y\|} \sqrt{1 - \left(\frac{(q - x) \cdot (y - x)}{\|q - x\| \cdot \|y - x\|}\right)^2}\right).
\]

(Dasgupta & Sinha, 2015)
Next Steps

For one tree and $n$ data points, the probability of failure can be characterized in terms of the potential function:

$$\Phi(q, \{x_1, \ldots, x_n\}) = \frac{1}{n} \sum_{i=2}^{n} \frac{\|q - x(1)\|}{\|q - x(i)\|}.$$
Next Steps

Ignoring constants:

1 tree:

\[ P_{\text{fail}}^{n_0} \propto \left( \frac{1}{n_0} \right)^{1/d_0} \]

(1)

2 trees:

\[ \left( P_{\text{fail}}^{n_0/2} \right)^2 \propto \left( \frac{2}{n_0} \right)^{2/d_0} \]

\[ < \left( \frac{1}{n_0} \right)^{1/d_0} \]

if \( n_0 > 4 \)

(Dasgupta & Sinha, 2015)
Next Steps

What we don’t know yet:

1. Deeper theoretical understanding: What a priori performance guarantees can be given?

2. Automatic parameter tuning.


4. Is it possible to speed up exact $k$-NN using MRPT?

5. Is random really the best we can do?
The end

Our article about the MRPT algorithm has been submitted to 2015 IEEE International Conference on Big Data. An optimized version of the MRPT algorithm for approximate nearest neighbor search will be later released as an R package. Thanks for your attention!

L Wang
Univ Cambridge

J Kangasharju
Univ Helsinki

S Tasoulis
COIN -> Liverpool JMU

J Corander
COIN

+ students T. Pitkänen, V. Hyvönen, E. Jääsaari


An optimized C++ implementation (with R bindings) for parallel MRPT will be released.