Nearest neighbor search in high dimensional spaces

Ilya Fedorov

March 2020

Content

- Why do we need to optimize KNN?
- Data structures
- Complexity analysis
- Difficulties in high dimensional spaces
- Approximate NN search: Locality Sensitive Hashing
- Approximate NN search: FlyHash
- Approximate NN search: Hierarchical Navigable Small World

Why do we need to optimize KNN?

- Very large datasets
- Very frequent queries
- Duplicates search
- It seems that brute force algorithm doens't use information gained from calculating previous distances: if $d(a, b)$ is big and $d(a, c)$ is small does it mean that $d(c, b)$ is big?

Data structures

- KD-Tree
- Ball-tree
- Ball*-tree
- R-Tree
- etc...

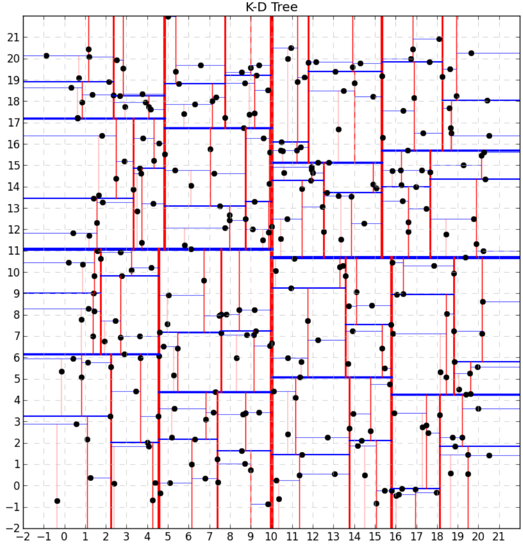
algorithm : *{'auto', 'ball_tree', 'kd_tree', 'brute'}, optional*

Algorithm used to compute the nearest neighbors:

- 'ball_tree' will use `BallTree`
- 'kd_tree' will use `KDTree`
- 'brute' will use a brute-force search.
- 'auto' will attempt to decide the most appropriate algorithm based on the values passed to `fit` method.

Note: fitting on sparse input will override the setting of this parameter, using brute force.

Data structures

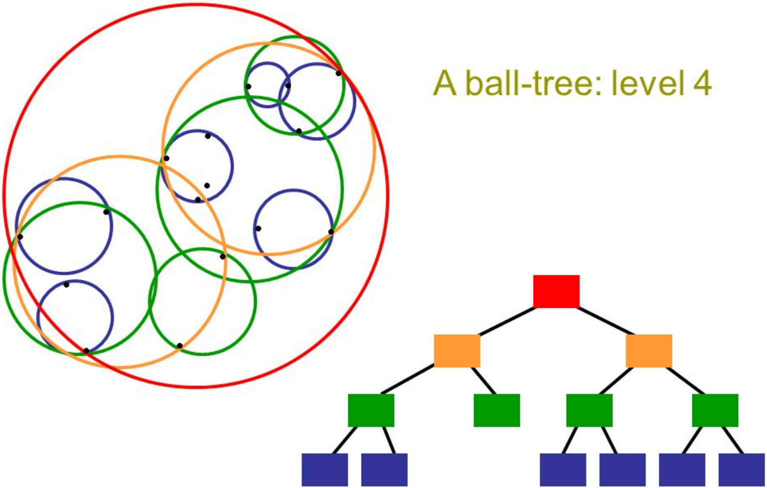


Data structures

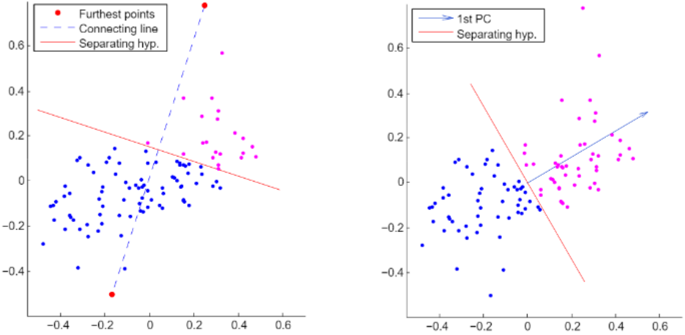
- Tree construction - by $build\_node(\{(x_1, y_1), ...(x_N, y_N)\})$:

```
build_node(Ω):  # omega-objects in the node
   if |Ω| < n_min:
      node.objects = Ω
   else:
      find feature with maximal spread in Ω:
         x^i = argmax_{x^i} σ(x^i)
      find median Ω: μ = median{x^i} # yields balanced tree
      node.feature = i
      node.threshold = μ
      node.left child =
            build_node({x_k ∈ Ω : x_k^i < μ})
      node.right child =
            build_node({x_k ∈ Ω : x_k^i ≥ μ})
   return node
```

Data structures



Data structures



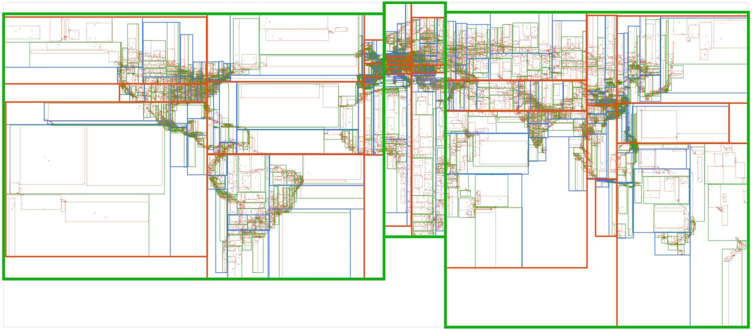
(a) Ball-tree (b) Ball*-tree

Fig. 2: Comparison of the splitting algorithms in ball-tree and ball*-tree

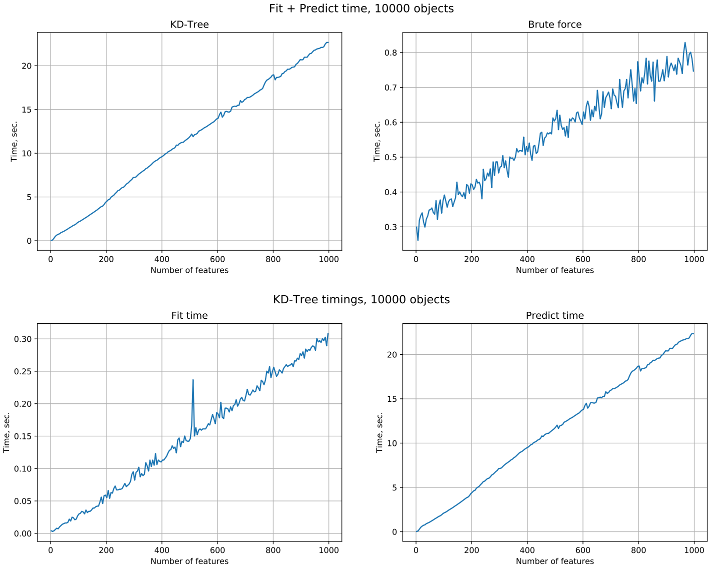
Dolatshah, Mohamad Hadian, Ali Minaei, Behrouz. (2015). Ball*-tree: Efficient spatial indexing for constrained nearest-neighbor search in metric spaces.

Data structures

R-tree

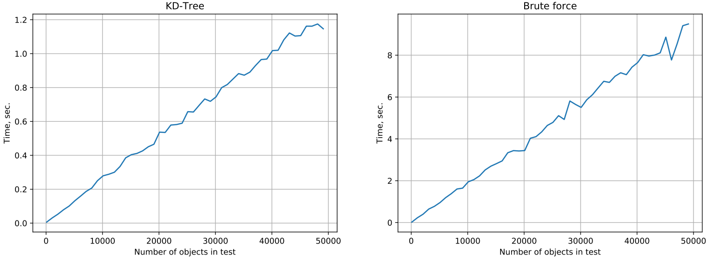


Complexity analysis

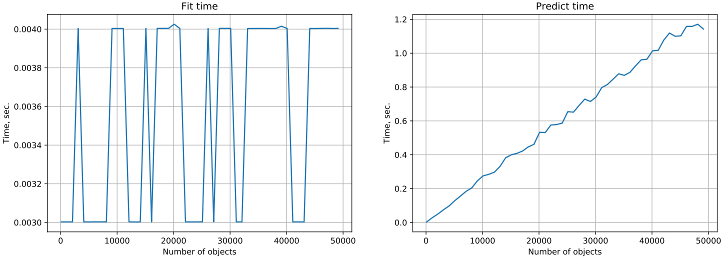


Complexity analysis

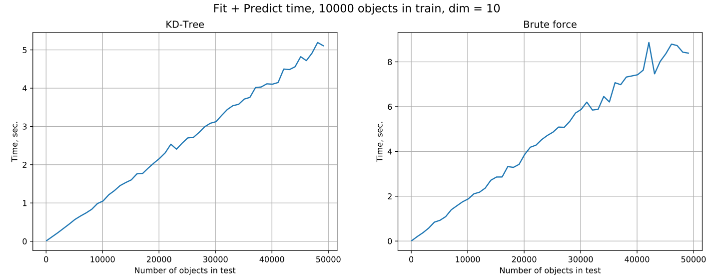
Fit + Predict time, 10000 objects in train, dim = 5

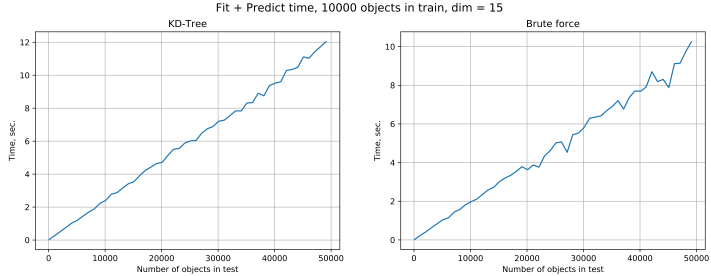


KD-Tree timings, 10000 objects in train, dim = 5



Complexity analysis





Difficulties in high dimensional spaces

Why does it happen?

Weber, Roger et al. “A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces.” VLDB (1998).

Difficulties in high dimensional spaces

Observation 1 (Number of partitions)
The most simple partitioning scheme splits the data space in each dimension into two halves. With d dimensions, there are 2^d partitions. With $d \leq 10$ and N on the order of 10^6, such a partitioning makes sense. However, if d is larger, say $d = 100$, there are around 10^{30} partitions for only 10^6 points—the overwhelming majority of the partitions are empty.

Difficulties in high dimensional spaces

Observation 2 (Data space is sparsely populated) Consider a hyper-cube range query with length l in all dimensions as depicted in Figure 1(a). The probability that a point lies within that range query is given by:

$$P^d[s] = s^d \tag{1}$$

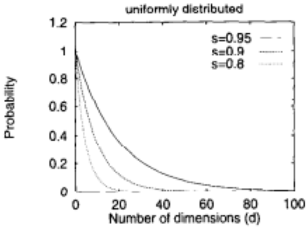


Figure 2: The probability function $P^d[s]$.

Difficulties in high dimensional spaces

Observation 3 (Spherical range queries)
The largest spherical query that fits entirely within the data space is the query $sp^d(Q, 0.5)$, where Q is the centroid of the data space (see Figure 1(b)). The probability that an arbitrary point R lies within this sphere is given by the spheres volume:[1]

$$P[R \in sp^d(Q, \frac{1}{2})] = \frac{Vol(sp^d(Q, \frac{1}{2}))}{Vol(\Omega)} = \frac{\sqrt{\pi^d} \cdot (\frac{1}{2})^d}{\Gamma(\frac{d}{2}+1)} \quad (2)$$

If d is even, then this probability simplifies to

$$P[R \in sp^d(Q, \frac{1}{2})] = \frac{\sqrt{\pi^d} \cdot (\frac{1}{2})^d}{(\frac{d}{2})!} \quad (3)$$

Observation 4 (Exponentially growing DB size)
Given equation (2), we can determine the size a data set would have to have such that, on average, at least one point falls into the sphere $sp^d(Q, 0.5)$ (for even d):

$$N(d) = \frac{(\frac{d}{2})!}{\sqrt{\pi^d} \cdot (\frac{1}{2})^d} \quad (4)$$

d	$P[R \in sp^d(Q, 0.5)]$	$N(d)$
2	0.785	1.273
4	0.308	3.242
10	0.002	401.5
20	$2.461 \cdot 10^{-8}$	40'631'627
40	$3.278 \cdot 10^{-21}$	$3.050 \cdot 10^{20}$
100	$1.868 \cdot 10^{-70}$	$5.353 \cdot 10^{69}$

Table 2: Probability that a point lies within the largest range query inside Ω, and the expected database size.

Difficulties in high dimensional spaces

Conclusion 1 (Performance) *For any clustering and partitioning method there is a dimensionality $\hat{d}$ beyond which a simple sequential scan performs better. Because equation (17) establishes a crude estimation, in practice this threshold $\hat{d}$ will be well below 610.*

Conclusion 2 (Complexity) *The complexity of any clustering and partitioning methods tends towards $O(N)$ as dimensionality increases.*

Conclusion 3 (Degeneration) *For every partitioning and clustering method there is a dimensionality $\bar{d}$ such that, on average, all blocks are accessed if the number of dimensions exceeds $\bar{d}$.*