FAST APPROXIMATE NEAREST NEIGHBORS WITH AUTOMATIC ALGORITHM CONFIGURATION

Marius Muja, David G. Lowe, 2009
Presented by: Gautam Gunjala & Jordan Zhang
Nearest Neighbors?

Given points $P = \{p_1, \ldots, p_n\}$ in a vector space $X$, preprocess such that given a query point $q$ in $X$, we can find the KNN of $q$ in $P$ efficiently.

1-NN example:

Why KNN? Applications

**SIFT**: 128-dimensional feature vector. Find nearest feature vector in DB to match images.

**Visual words**: Cluster local features into “words” representing regions of an image.

**ML**: Classifiers

**Compression**: Best approximations of principal components.
Issues

- Performance of algorithms vary.
  - Dimensionality, internal correlations, dataset size.
- Exact NN not much better than linear search $O(Nd)$ in high dimensions for $N$ scene points, query dimension $d$.
  - Must turn to approximate NN, sometimes returns non-optimal points.
Contributions

- Determine which algorithm to use given a particular dataset structure.
- Improve the hierarchical k-means algorithm to reduce construction time and speed up queries by approximation.
Algorithm 1: KD Tree Search
KD Trees Overview

- “k-dimensional trees”
- Split data in half at each level on the axis of greatest variance
- Fast operations for vectors in small dimensions
- In high dimensions, we get far fewer splits per dimension.
Visualizing KD Tree Search

http://upload.wikimedia.org/wikipedia/commons/9/9c/KDTree-animation.gif
Early KD Trees (Freidman et al.)

Introduces an optimized kd tree algorithm

- $O(N)$ space
- $O(kN \log N)$ tree build time
- $O(\log N)$ search

Early KD Trees (Arya et al.)

Introduces the idea of an $\epsilon$-approximate nearest neighbor

- $\epsilon > 0$, $\delta \leq d \cdot \text{ceil}[1 + 6d/\epsilon]^d$ such that the nearest neighbor can be returned in $O(\delta \log N)$ time

Early KD Trees (Beis & Lowe, 1997)

Achieved a fast approximate nearest neighbor algorithm by fixing the number of leaf nodes examined.

Slightly faster than the $\epsilon$-approximate nearest neighbor method of Arya et al.
Randomized KD Trees

- First implemented by Silpa-Anan & Hartley (2008)
- Faster than traditional KD trees.
- Memory intensive: must build multiple trees for a data set.
Randomized KD Trees (Construction)

- Multiple trees constructed at once
- One of the top D dimensions of greatest variance randomly selected for each tree
- Data split on that dimension
- D = 5 used for testing
- Varying structure allows each tree search to be independent
Randomized KD Trees

- Single priority queue used to search $m$ trees at once; elements ordered by distance from query point
- Fixed number, $n$, of nodes examined (number based on user input search precision)
- On average, $n/m$ nodes examined per tree
- Best candidates returned
Randomized KD Trees Efficiency

- Performance improves up to $10^{1.3} \approx 20$ trees
- 100,000 SIFT features used in test
- Memory overhead increases linearly
Algorithm 2: Hierarchical K-Means
Hierarchical K-Means

1. Precompute: k-means cluster into K clusters. Then k-means cluster each of these recursively to build a tree.

2. Query: Traverse graph with priority queue, limit number of leaf nodes checked.

“Scalable recognition with a vocabulary tree”, Nister, Stewenius, 2006
Hierarchical K-Means


Hierarchical K-Means

Single level K-means is $O(N k l d)^*$ for $N$ data points, $k$ means, $l$ iterations of improvement.
The complexity of K-means tree is then $O(N k l d \log(N))$ by Master theorem.

* “Efficient clustering and matching for object class recognition,” Leibe et al. 2006
Hierarchical K-Means

Hierarchical k-means tree projected into 2-D. Search depth is labeled from red shallowest to green deepest.

“City Scale Location Recognition” Schindler et al. 2007
Hierarchical K-Means

**Improvement:** Order of magnitude speedup by using only 7 iterations in each clustering step. 90% of convergence accuracy.

Figure 3: The influence that the number of k-means iterations has on the search time efficiency of the k-means tree (a) and on the tree construction time (b) (100K SIFT features dataset)
Hierarchical K-Means

Improvement: Best Bin First by distance of cluster center.
1. Greedily traverse once to a leaf node. Add unexplored branches along path to a priority queue.
2. Restart another greedy traversal from the cluster branch closest to query (can be in the middle of the tree)

Stops after a given number of leaf nodes (points) are checked.
Data Dimensionality

(a) Random vectors
(b) Image patches
Data Dimensionality

Queries on Trevi Fountain patch data set with different patch sizes. Demonstrates that even for high dimensional data \((64\times64 = 4096\) dimensions in a patch) that similarity in a small number of dimensions provides strong enough evidence to determine overall patch similarity.
Choosing an Algorithm

Data features:

- Correlations between dimensions.
- Size of data set.

Algorithm features:

- Number of kd-trees.
- Branching factor of k-means.
- Number of iterations in k-means clustering step.
Choosing an Algorithm

\[ \text{cost} = \frac{s + w_b b}{(s + w_b b)_{\text{opt}}} + w_m m \]

s -- search time
b -- tree build time
\(w_b\) -- build time weight
\(m = m_t / m_d\) -- ratio of tree memory to raw data
\(w_m\) -- tree memory weight
Choosing an Algorithm

\[ cost = \frac{s + wb b}{(s + wb b)_{opt}} + w_{mm} \]

\((s + w_{b} b)_{opt}\) -- optimal time cost if memory doesn’t matter.
Computed during following optimization process.
Choosing an Algorithm

For a given data set, compute cost for:

- \{1,4,8,16,32\} kd trees
- \{16,32,64,128,256\} k-means branching factor.
- \{1,5,10,15\} k-means clustering iterations

May check 1/10 of the data set and still be close to optimum.
Choosing an Algorithm

Optimize using simplex locally after finding the best option from the previous step.

Expensive to optimize, but results for each type of dataset can simply be stored.
Results
Results
## Results

<table>
<thead>
<tr>
<th>Pr. (%)</th>
<th>( w_b )</th>
<th>( w_m )</th>
<th>Algorithm</th>
<th>Dist. Error</th>
<th>Search Speedup</th>
<th>Memory Used</th>
<th>Build Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>60%</td>
<td>0</td>
<td>0</td>
<td>k-means, 16, 15</td>
<td>0.096</td>
<td>181.10</td>
<td>0.51</td>
<td>0.58</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>k-means, 32, 10</td>
<td>0.058</td>
<td>180.9</td>
<td>0.37</td>
<td>0.56</td>
</tr>
<tr>
<td></td>
<td>0.01</td>
<td>0</td>
<td>k-means, 16, 5</td>
<td>0.077</td>
<td>163.25</td>
<td>0.50</td>
<td>0.26</td>
</tr>
<tr>
<td></td>
<td>0.01</td>
<td>1</td>
<td>kd-tree, 4</td>
<td>0.041</td>
<td>109.50</td>
<td>0.26</td>
<td>0.12</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>kd-tree, 1</td>
<td>0.044</td>
<td>56.87</td>
<td>0.07</td>
<td>0.03</td>
</tr>
<tr>
<td>*</td>
<td>( \infty )</td>
<td></td>
<td>kd-tree, 1</td>
<td>0.044</td>
<td>56.87</td>
<td>0.07</td>
<td>0.03</td>
</tr>
<tr>
<td>90%</td>
<td>0</td>
<td>0</td>
<td>k-means, 128, 10</td>
<td>0.008</td>
<td>31.67</td>
<td>0.18</td>
<td>1.82</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>k-means, 128, 15</td>
<td>0.007</td>
<td>30.53</td>
<td>0.18</td>
<td>2.32</td>
</tr>
<tr>
<td></td>
<td>0.01</td>
<td>0</td>
<td>k-means, 32, 5</td>
<td>0.011</td>
<td>29.47</td>
<td>0.36</td>
<td>0.35</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>k-means, 16, 1</td>
<td>0.016</td>
<td>21.59</td>
<td>0.48</td>
<td>0.10</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>kd-tree, 1</td>
<td>0.005</td>
<td>5.05</td>
<td>0.07</td>
<td>0.03</td>
</tr>
<tr>
<td>*</td>
<td>( \infty )</td>
<td></td>
<td>kd-tree, 1</td>
<td>0.005</td>
<td>5.05</td>
<td>0.07</td>
<td>0.03</td>
</tr>
</tbody>
</table>
In Conclusion

- Improved hierarchical k-means tree search algorithm
- Identification of two best nearest neighbor search algorithms for any data set
- Automatic choice of algorithm and optimal parameters
- Algorithms contained in public domain library