Using multiple tables for even greater efficiency in NN search

If I throw down 2 lines...

<table>
<thead>
<tr>
<th>Bin indices</th>
<th>L0</th>
<th>L1</th>
<th>L2</th>
<th>L3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bin index:</td>
<td>[0 0] = 0</td>
<td>[0 1] = 1</td>
<td>[1 0] = 2</td>
<td>[1 1] = 3</td>
</tr>
</tbody>
</table>

For simplicity, assume we search bins 1 bit off from query

Let $\delta$ be the probability of a line falling between points $\theta$ apart

Search 3 bins and do not find NN with probability $\delta^2$
What if I repeat the 2-line binning?

<table>
<thead>
<tr>
<th>Bin</th>
<th>Line 1</th>
<th>Line 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0 0]</td>
<td>[0 0]</td>
<td>[0 1]</td>
</tr>
<tr>
<td>[0 1]</td>
<td>[0 1]</td>
<td>[0 2]</td>
</tr>
<tr>
<td>[1 0]</td>
<td>[1 0]</td>
<td>[1 1]</td>
</tr>
<tr>
<td>[1 1]</td>
<td>[1 1]</td>
<td>[1 2]</td>
</tr>
</tbody>
</table>

indices: L0 L1 L2 L3

Now, search only query bin per table

Still searching 3 bins, but what is chance of not finding NN?
What if I repeat the 2-line binning?

<table>
<thead>
<tr>
<th>Bin</th>
<th>[0 0] = 0</th>
<th>[0 1] = 1</th>
<th>[1 0] = 2</th>
<th>[1 1] = 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>indices</td>
<td>L0</td>
<td>L1</td>
<td>L2</td>
<td>L3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bin</th>
<th>[0 0] = 0</th>
<th>[0 1] = 1</th>
<th>[1 0] = 2</th>
<th>[1 1] = 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>indices</td>
<td>L0</td>
<td>L1</td>
<td>L2</td>
<td>L3</td>
</tr>
</tbody>
</table>

Probability of splitting neighboring points many times

<table>
<thead>
<tr>
<th>Bin</th>
<th>[0 0] = 0</th>
<th>[0 1] = 1</th>
<th>[1 0] = 2</th>
<th>[1 1] = 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>indices</td>
<td>L0</td>
<td>L1</td>
<td>L2</td>
<td>L3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bin</th>
<th>[0 0] = 0</th>
<th>[0 1] = 1</th>
<th>[1 0] = 2</th>
<th>[1 1] = 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>indices</td>
<td>L0</td>
<td>L1</td>
<td>L2</td>
<td>L3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bin</th>
<th>[0 0] = 0</th>
<th>[0 1] = 1</th>
<th>[1 0] = 2</th>
<th>[1 1] = 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>indices</td>
<td>L0</td>
<td>L1</td>
<td>L2</td>
<td>L3</td>
</tr>
</tbody>
</table>

What is chance that query pt and NN are split in all tables?

Probability NN is in different bin:

\[
\text{Prob} = 1 - \Pr(\text{same bin}) = 1 - (1 - \delta)^2 \\
= 2\delta - \delta^2
\]

\[
1 - \delta = \text{prob. that 1 line does not split query + NN} \\
(1-\delta)^2 = \text{prob. that 2 lines don't split pts}
\]
Probability of splitting neighboring points many times

<table>
<thead>
<tr>
<th>Bin</th>
<th>[0 0]</th>
<th>[0 1]</th>
<th>[1 0]</th>
<th>[1 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>indices</td>
<td>L0</td>
<td>L1</td>
<td>L2</td>
<td>L3</td>
</tr>
</tbody>
</table>

Probability NN is in different bin in all 3 tables:

$\text{Prob} = (2\delta - \delta^2)^3$
Comparing probabilities

If I throw down $h$ lines...

<table>
<thead>
<tr>
<th>Bin</th>
<th>[0 0 0]</th>
<th>[0 0 1]</th>
<th>[0 1 0]</th>
<th>[0 1 1]</th>
<th>[1 0 0]</th>
<th>[1 0 1]</th>
<th>[1 1 0]</th>
<th>[1 1 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>L0</td>
<td>0</td>
<td>L1</td>
<td>L2</td>
<td>L3</td>
<td>L4</td>
<td>L5</td>
<td>L6</td>
<td>L7</td>
</tr>
</tbody>
</table>

Still assume we search bins 1 bit off from query

Prob. of being > 1 bit away

$= 1 - \Pr(\text{same bin}) - \Pr(1 \text{ bin away})$

$= 1 - \Pr(\text{no split lines}) - \Pr(1 \text{ split line})$

$= 1 - (1 - \delta)^h - h\delta(1 - \delta)^{h - 1}$
If I throw down $h$ lines...

Still assume we **search bins 1 bit off** from query

Prob. of being > 1 bit away

$$= 1 - \text{Pr(same bin)} - \text{Pr(1 bin away)}$$

$$= 1 - \text{Pr(no split lines)} - \text{Pr(1 split line)}$$

$$= 1 - (1 - \delta)^h - h\delta(1 - \delta)^{h - 1}$$

Search $h + 1$ bins and do not find NN with probability $1 - (1 - \delta)^h - h\delta(1 - \delta)^{h - 1}$

---

### Probability of splitting neighboring points many times

```
<table>
<thead>
<tr>
<th>Bin</th>
<th>[0 0 0]</th>
<th>[0 0 1]</th>
<th>[0 1 0]</th>
<th>[0 1 1]</th>
<th>[1 0 0]</th>
<th>[1 0 1]</th>
<th>[1 1 0]</th>
<th>[1 1 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>indices</td>
<td>L0</td>
<td>L1</td>
<td>L2</td>
<td>L3</td>
<td>L4</td>
<td>L5</td>
<td>L6</td>
<td>L7</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Bin</th>
<th>[0 0 0]</th>
<th>[0 0 1]</th>
<th>[0 1 0]</th>
<th>[0 1 1]</th>
<th>[1 0 0]</th>
<th>[1 0 1]</th>
<th>[1 1 0]</th>
<th>[1 1 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>indices</td>
<td>L0</td>
<td>L1</td>
<td>L2</td>
<td>L3</td>
<td>L4</td>
<td>L5</td>
<td>L6</td>
<td>L7</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Bin</th>
<th>[0 0 0]</th>
<th>[0 0 1]</th>
<th>[0 1 0]</th>
<th>[0 1 1]</th>
<th>[1 0 0]</th>
<th>[1 0 1]</th>
<th>[1 1 0]</th>
<th>[1 1 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>indices</td>
<td>L0</td>
<td>L1</td>
<td>L2</td>
<td>L3</td>
<td>L4</td>
<td>L5</td>
<td>L6</td>
<td>L7</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Bin</th>
<th>[0 0 0]</th>
<th>[0 0 1]</th>
<th>[0 1 0]</th>
<th>[0 1 1]</th>
<th>[1 0 0]</th>
<th>[1 0 1]</th>
<th>[1 1 0]</th>
<th>[1 1 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>indices</td>
<td>L0</td>
<td>L1</td>
<td>L2</td>
<td>L3</td>
<td>L4</td>
<td>L5</td>
<td>L6</td>
<td>L7</td>
</tr>
</tbody>
</table>
```

---

Probability NN is in different bin in all $h + 1$ tables

$$= (1 - \text{Pr(same bin)})^{h + 1}$$

$$= (1 - \text{Pr(no split line)})^{h + 1}$$

$$= (1 - (1 - \delta)^h)^{h + 1}$$
Comparing approaches for h-bit tables

<table>
<thead>
<tr>
<th>Bin</th>
<th>[0 0] = 0</th>
<th>[0 1] = 1</th>
<th>[1 0] = 2</th>
<th>[1 1] = 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>indices</td>
<td>L0</td>
<td>L1</td>
<td>L2</td>
<td>L3</td>
</tr>
</tbody>
</table>

# bins searched  

<table>
<thead>
<tr>
<th>prob. of no NN</th>
</tr>
</thead>
</table>
| h+1  

\[
1 - (1 - \delta)^h - h\delta(1 - \delta)^{h-1}
\]

Comparing probabilities

<table>
<thead>
<tr>
<th>h = 3</th>
</tr>
</thead>
</table>

One hash table

\[
1 - (1 - \delta)^h - h\delta(1 - \delta)^{h-1}
\]

<table>
<thead>
<tr>
<th>h = 10</th>
</tr>
</thead>
</table>

Multiple hash table

\[
(1 - (1 - \delta)^h)^{h+1}
\]
Fix #bits and increase depth

Typically higher probability of finding NN than searching m bins in 1 table
Summary of LSH approaches

<table>
<thead>
<tr>
<th>Bin</th>
<th>[0 0] = 0</th>
<th>[0 1] = 1</th>
<th>[1 0] = 2</th>
<th>[1 1] = 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>indices</td>
<td>L0</td>
<td>L1</td>
<td>L2</td>
<td>L3</td>
</tr>
</tbody>
</table>

Cost of binning points is **lower**, but likely need to search **more bins** per query.

<table>
<thead>
<tr>
<th>Bin</th>
<th>[0 0] = 0</th>
<th>[0 1] = 1</th>
<th>[1 0] = 2</th>
<th>[1 1] = 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>indices</td>
<td>L0</td>
<td>L1</td>
<td>L2</td>
<td>L3</td>
</tr>
</tbody>
</table>

Cost of binning points is **higher**, but likely need to search **fewer bins** per query.

KD-trees

**OPTIONAL**
KD-trees

Structured organization of documents
  - Recursively partitions points into axis aligned boxes.
  
Enables more efficient pruning of search space

Works “well” in “low-medium” dimensions
  - We’ll get back to this...

---

KD-tree construction

Start with a list of d-dimensional points.

<table>
<thead>
<tr>
<th>Pt</th>
<th>x[1]</th>
<th>x[2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>1.00</td>
<td>4.31</td>
</tr>
<tr>
<td>3</td>
<td>0.13</td>
<td>2.85</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
**KD-tree construction**

Split points into 2 groups

<table>
<thead>
<tr>
<th>Pt</th>
<th>x[1]</th>
<th>x[2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>1.00</td>
<td>4.31</td>
</tr>
<tr>
<td>3</td>
<td>0.13</td>
<td>2.85</td>
</tr>
</tbody>
</table>

Recurse on each group separately

<table>
<thead>
<tr>
<th>Pt</th>
<th>x[1]</th>
<th>x[2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>1.00</td>
<td>4.31</td>
</tr>
<tr>
<td>3</td>
<td>0.13</td>
<td>2.85</td>
</tr>
</tbody>
</table>

©2018 Emily Fox
KD-tree construction

Recurse on each group separately

- Split dim 1
- Split value 2
- Split dim 2
- Split value 2

Continue splitting points at each set
- Creates a binary tree structure

Each leaf node contains a list of points
KD-tree construction

Keep one additional piece of info at each node:
- The (tight) bounds of points at or below node

KD-tree construction choices

Use heuristics to make splitting decisions:
- Which dimension do we split along?
  - widest (or alternate)
- Which value do we split at?
  - median (or center point of box, ignoring data in box)
- When do we stop?
  - fewer than m pts left
  - box hits minimum width
Many heuristics...

- median heuristic
- center-of-range heuristic

NN search with KD-trees

**OPTIONAL**
Nearest neighbor with KD-trees

Traverse tree looking for nearest neighbor to query point

1. Start by exploring leaf node containing query point
1. Start by exploring leaf node containing query point
Nearest neighbor with KD-trees

1. Start by exploring leaf node containing query point
2. Compute distance to each other point at leaf node

Does nearest neighbor have to live at leaf node containing query point?
Nearest neighbor with KD-trees

1. Start by exploring leaf node containing query point
2. Compute distance to each other point at leaf node
3. Backtrack and try other branch at each node visited

Update distance bound when new nearest neighbor is found
Nearest neighbor with KD-trees

Use distance bound and bounding box of each node to prune parts of tree that cannot include nearest neighbor.
Nearest neighbor with KD-trees

Use distance bound and bounding box of each node to prune parts of tree that cannot include nearest neighbor

Complexity

For (nearly) balanced, binary trees...

- Construction
  - Size: $2N-1$ nodes if 1 datapt at each leaf $\Rightarrow O(N)$
  - Depth: $O(\log N)$
  - Median + send points left right: $O(N)$ at every level of the tree
  - Construction time: $O(N \log N)$

- 1-NN query
  - Traverse down tree to starting point: $O(\log N)$
  - Maximum backtrack and traverse: $O(N)$ in worst case
  - Complexity range: $O(\log N)$ to $O(N)$

Under some assumptions on distribution of points, we get $O(\log N)$ but exponential in $d$
Complexity

Complexity for N queries

• Ask for nearest neighbor to each doc
  \[ N \text{ queries} \]

• Brute force 1-NN:
  \[ O(N^2) \]

• kd-trees:
  \[ O(N\log N) \rightarrow O(N^2) \]
  → potentially very large savings for large \( N \)!
Inspections vs. N and d

- **log(N) trend**
- **exp(d) trend**

k-NN with KD-trees

- Distance to 2nd nearest neighbor in 2-NN example
- Exactly same algorithm, but maintain distance to furthest of current k nearest neighbors
Approximate k-NN search

Approximate k-NN with KD-trees

Before: Prune when distance to bounding box > r
Now: Prune when distance to bounding box > r/α

Prunes more than allowed, but can guarantee that if we return a neighbor at distance r, then there is no neighbor closer than r/α

Bound loose...In practice, often closer to optimal.

Saves lots of search time at little cost in quality of NN!
Closing remarks on KD-trees

Tons of variants of KD-trees
- On construction of trees (heuristics for splitting, stopping, representing branches...)
- Other representational data structures for fast NN search (e.g., ball trees,...)

Nearest Neighbor Search
- Distance metric and data representation crucial to answer returned

For both, high-dim spaces are hard!
- Number of KD-tree searches can be exponential in dimension
  - Rule of thumb... \( N \gg 2^d \)... Typically useless for large \( d \).
- Distances sensitive to irrelevant features
  - Most dimensions are just noise \( \Rightarrow \) everything is far away
  - Need technique to learn which features are important to given task

Motivating alternative approaches to approximate NN search

- KD-trees are cool, but...
  - Non-trivial to implement efficiently
  - Problems with high-dimensional data
KD-trees in high dimensions

- Unlikely to have any data points close to query point
- Once “nearby” point is found, the search radius is likely to intersect many hypercubes in at least one dim
- Not many nodes can be pruned
- Can show under some conditions that you visit at least $2^d$ nodes

Acknowledgements

Thanks to Andrew Moore (http://www.cs.cmu.edu/~awm/) for the KD-trees slide outline