Case Study 2: Document Retrieval

Task Description:
Finding Similar Items

Announcements:

• HW2 posted
• Project Milestones
  – Start early
  – Lit. review (>= 3 papers read carefully)
  – First rounds of experiments

• Today:
  – Review: Sim search, k-NNs, KD-trees
  – Today: KD-trees (cont.), ball trees, cover trees
Task 1: Find Similar Documents

- To begin...
  - **Input:** Query article
  - **Output:** Set of $k$ similar articles

Document Representation

- Bag of words model

\[ X = \begin{bmatrix} wc_1 \\ \vdots \\ wc_d \end{bmatrix} \in \mathbb{R}^d \]

"Bag of words": count of words; ignore word order.
Image Search...

1-Nearest Neighbor

- Articles
- Query:
- 1-NN
  - Goal:
    \[ \text{find } x \in \mathbb{R}^d \text{ "closest" to } x. \]
  - Formulation:
    \[ x_{\text{NN}} = \arg\min_{x' \in \mathbb{R}^d} d(x', x) \]
Issues with Search Techniques

- Naïve approach:
  - **Brute force search**
    - Given a query point $x'$
    - Scan through each point $x^i$
    - $O(N)$ distance computations per 1-NN query!
    - $O(N \log k)$ per $k$-NN query!

- What if $N$ is huge???
  (and many queries)

Think about Web Search/Image Search

- How big is $N$?

- How fast do we desire to do recall?
Intuition (?): NN in 1D and Sorting

- How do we do 1-NN searches in 1 dim?

  - How do we sort?

  - Pre-processing time:
    \[ O(n) \]
  - Query time:
    \[ O(1) \]

Pre-processing time:

- Query time:

KD-Trees

- Smarter approach: \textit{kd-trees}
  - Structured organization of documents
    - Recursively partitions points into axis aligned boxes.
  - Enables more efficient pruning of search space
    - Examine nearby points first.
    - Ignore any points that are further than the nearest point found so far.

\textit{kd-trees} work “well” in “low-medium” dimensions

- We’ll get back to this…
- Start with a list of $d$-dimensional points.

Split the points into 2 groups by:
- Choosing dimension $d_j$ and value $V$ (methods to be discussed...)
- Separating the points into $x_{d_j} > V$ and $x_{d_j} \leq V$. 

<table>
<thead>
<tr>
<th>Pt</th>
<th>X</th>
<th>Y</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>
Consider each group separately and possibly split again (along same/different dimension).

- Stopping criterion to be discussed...
KD-Tree Construction

- Continue splitting points in each set
  - creates a binary tree structure
- Each leaf node contains a list of points

KD-Tree Construction

- Keep one additional piece of information at each node:
  - The (tight) bounds of the points at or below this node.
KD-Tree Construction

- Use heuristics to make splitting decisions:
  - Which dimension do we split along?
  - Which value do we split at?
  - When do we stop?

Many heuristics...

- median heuristic
- center-of-range heuristic
Traverse the tree looking for the nearest neighbor of the query point.

- Examine nearby points first:
  - Explore branch of tree closest to the query point first.
Examine nearby points first:
- Explore branch of tree closest to the query point first.

When we reach a leaf node:
- Compute the distance to each point in the node.
When we reach a leaf node:
- Compute the distance to each point in the node.

Then backtrack and try the other branch at each node visited.
Each time a new closest node is found, update the distance bound.

Using the distance bound and bounding box of each node:
- Prune parts of the tree that could NOT include the nearest neighbor.
Using the distance bound and bounding box of each node:

- Prune parts of the tree that could NOT include the nearest neighbor
Complexity

- For (nearly) balanced, binary trees...
- Construction
  - Size:
  - Depth:
  - Median + send points left right:
  - Construction time:
- 1-NN query
  - Traverse down tree to starting point:
  - Maximum backtrack and traverse:
  - Complexity range:

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

- Ask for nearest neighbor to each document
- Brute force 1-NN:
- $kd$-trees:

Inspections vs. $N$ and $d$
K-NN with KD Trees

- Exactly the same algorithm, but maintain distance as distance to furthest of current $k$ nearest neighbors
- Complexity is:

Approximate K-NN with KD Trees

- **Before**: Prune when distance to bounding box >
- **Now**: Prune when distance to bounding box >
- Will prune more than allowed, but can guarantee that if we return a neighbor at distance $r$, then there is no neighbor closer than $r/\alpha$.
- In practice this bound is loose...Can be closer to optimal.
- Saves lots of search time at little cost in quality of nearest neighbor.
What about NNs searches in high dimensions?

- KD-trees:
  - What is going wrong?
  - Can this be easily fixed?

- What do have to utilize?
  - Utilize triangle inequality of metric
  - New ideas: ball trees and cover trees

**Ball Trees**

Ball-tree Example

(level 1) (level 2)

(level 3) (level 4)
Ball Tree Construction

- **Node:**
  - Every node defines a ball (hypersphere), containing
    - a subset of the points (to be searched)
    - A center
    - A (tight) radius of the points

- **Construction:**
  - Root: start with a ball which contains all the data
  - Take a ball and make two children (nodes) as follows:
    - Make two spheres, assign each point (in the parent sphere) to its closer sphere
    - Try to make the two spheres in a “reasonable” manner

Ball Tree Search

- Given point \( x \), how do find its nearest neighbor quickly?
- **Approach:**
  - Start: follow a greedy path through the tree
  - Backtrack and prune: rule out other paths based on the triangle inequality
    - (just like in KD-trees)
- **How good is it?**
  - **Guarantees:**
  - **Practice:**
Cover trees (+ ball trees)

- What about exact NNs searches in high dimensions?
- Idea: utilize triangle inequality of metric (so allow for arbitrary metric)
- cover-tree guarantees:

Cover trees: what does the triangle inequality imply?
Cover trees: data structure

Wrapping Up – Important Points

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

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

For both...
- High dimensional spaces are hard!
  - Number of kd-tree searches can be exponential in dimension
    - Rule of thumb... $N \gg 2^d$... Typically useless.
  - Distances are sensitive to irrelevant features
    - Most dimensions are just noise $\Rightarrow$ Everything equidistant (i.e., everything is far away)
    - Need technique to learn what features are important for your task
What you need to know

- Document retrieval task
  - Document representation (bag of words)
  - tf-idf
- Nearest neighbor search
  - Formulation
  - Different distance metrics and sensitivity to choice
  - Challenges with large $N$
- kd-trees for nearest neighbor search
  - Construction of tree
  - NN search algorithm using tree
  - Complexity of construction and query
  - Challenges with large $d$