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

$$X = \begin{bmatrix} w_{c_1} \\ \vdots \\ w_{c_d} \end{bmatrix} \in \mathbb{R}^d$$

Bag of words model:
- Ignore word order.

© Sham Kakade 2017
Where is FAST similarity search important?

- Image search
- Image search
- Sky maps identification
- Species/ song identification
- Physics simulators
- Robotics
1-Nearest Neighbor

- Articles
  \[ \mathcal{X} = \{X^1, \ldots, X^n\} \]

- Query:
  \[ \cdot \]

- 1-NN
  - Goal:
    \[ \text{find } \mathbf{x} \in \mathcal{X} \text{ closest to } \mathbf{x} \text{ query} \]
  - Formulation:
    \[ \mathbf{x}_{NN} \in \mathbf{x} \in \mathcal{X} \setminus \mathbf{x} \]

Issues with Search Techniques

- Naïve approach:
  - **Brute force search**
    - Given a query point \( \mathbf{x} \)
    - Scan through each point \( \mathbf{x}^i \)
    - \( O(N) \) 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 (?): $\text{NN in 1D and Sorting}$

- How do we do 1-NN searches in 1 dim?
- Pre-processing time:
  \[ O(N) \]
  \[ O(N \log N) \]
- Query time:
  \[ O(1) \]
  \[ O(\log N) \]
KD-Trees

- Smarter approach: *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.

- *kd-trees* work “well” in “low-medium” dimensions
  - We’ll get back to this...

KD-Tree Construction

- 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 $x_{d_j} \leq V$.

Consider each group separately and possibly split again (along same/different dimension).
- Stopping criterion to be discussed...
Consider each group separately and possibly split again (along same/different dimension).

- Stopping criterion to be discussed...

- 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?
    - widest \((\text{some variance measure})\)
  - Which value do we split at?
    - median of content of range
  - When do we stop?
    - Stop when each box has \(s \leq m\) points.
Many heuristics...

- Median heuristic
- Center-of-range heuristic

Nearest Neighbor with KD Trees

- Traverse the tree looking for the nearest neighbor of the query point.
Examinate 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.
Nearest Neighbor with KD Trees

- 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
Nearest Neighbor with KD Trees

- 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: \( O(N) \)
    - Depth: \( O(\log N) \) (if can balance)
    - Median + send points left right:
    - Construction time: \( O(N \log N) \)
  - 1-NN query
    - Traverse down to starting point:
    - Maximum backtrack and traverse:
    - Complexity range: \( O(N \log N) \) to \( O(N) \)

- 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:
  \[ O(N^2) \]

- kd-trees:
  \[ O(N^2) \Rightarrow O(N \log N) \]
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: $O(k \log N)$
Approximate K-NN with KD Trees

- **Before**: Prune when distance to bounding box > \( r \)
- **Now**: Prune when distance to bounding box > \( r/\alpha \)
- 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 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
    - 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

- What about exact NNs in general metric spaces?

- Same Idea: utilize triangle inequality of metric (so allow for arbitrary metric)

- What does the dimension even mean?

- cover-tree idea: exploit the structure in the data
Intrinsic Dimension

- How does the volume grow, from radius $R$ to $2R$?
  \[
  \frac{\text{Vol}(B_{2R})}{\text{Vol}(B_R)} = 2^d
  \]

- Can we relax this idea to get at the “intrinsic” dimension?

  □ This is the “doubling” dimension:

Cover trees: data structure

- Ball Trees: each node had associated
  □ Center:
  □ (tight) Radius:
  □ Points:

- Cover trees:
  □ Center:
  □ (tight) Radius:
  □ Points:
Cover Tree Complexity

- Construction
  - Size:
  - Construction time:
- 1-NN query
  - Traverse down tree to starting point:
  - Maximum backtrack and traverse:
- Under assumptions that doubling dimension is D.

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 $d$