Case Study 2: Document Retrieval

Task Description:
Finding Similar Documents

Machine Learning for Big Data
CSE547/STAT548, University of Washington
Emily Fox
January 21st, 2014

Task 1: Find Similar Documents

- To begin...
  - Input: Query article  ❌
  - Output: Set of k similar articles
k-Nearest Neighbor

- **Articles**: \( X = \{ x^1, \ldots, x^N \}, \ x^i \in \mathbb{R}^d \)
- **Query**: \( x \in \mathbb{R}^d \)
- **k-NN**
  - **Goal**: Find \( k \) articles in \( X \) closest to \( x \)
  - **Formulation**:
    \[
    X^{\text{NN}} = \{ x^{1\text{NN}}, \ldots, x^{k\text{NN}} \} \subseteq X
    \]
    \[
    \text{s.t. } \forall \ x^i \in X \setminus X^{\text{NN}} \rightarrow \ "\text{not in nearest neighbors}\"
    \]
    \[
    d(x^i, x) \geq \max_{x^{j\text{NN}}} d(x^{j\text{NN}}, x)
    \]

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)
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

- Keep one additional piece of information at each node:
  - The (tight) bounds of the points at or below this node.
Traverse the tree looking for the nearest neighbor of the query point.

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

distance to closest NN so far
does NN have to be in this box (blue)? NO
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

No article in this box could be the NN.
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:
  - 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

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.

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., 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 -> 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 $d$

Acknowledgment

- This lecture contains some material from Andrew Moore’s excellent collection of ML tutorials:
  - http://www.cs.cmu.edu/~awm/tutorials
- In particular, see:
  - http://grist.caltech.edu/sc4devo/.../files/sc4devo_scalable_datamining.ppt
Locality-Sensitive Hashing
Random Projections for NN Search

Using Hashing to Find Neighbors

- KD-trees are cool, but…
  - Non-trivial to implement efficiently
  - Problems with high-dimensional data
- Approximate neighbor finding…
  - Don’t find exact neighbor, but that’s OK for many apps, especially with Big Data
- What if we could use hash functions:
  - Hash elements into buckets:
    - Look for neighbors that fall in same bucket as \( x \):
  - But, by design…
Locality Sensitive Hashing (LSH)

- A LSH function $h$ satisfies (for example), for some similarity function $d$, for $r > 0$, $\alpha > 1$:  
  - $d(x,x') \leq r$, then $P(h(x)=h(x'))$ is high  
  - $d(x,x') > \alpha r$, then $P(h(x)=h(x'))$ is low  
  - (in between, not sure about probability)

Random Projection Illustration

- Pick a random vector $v$:  
  - Independent Gaussian coordinates  

- Preserves separability for most vectors  
  - Gets better with more random vectors
Multiple Random Projections: Approximating Dot Products

- Pick \( m \) random vectors \( v^{(i)} \):
  - Independent Gaussian coordinates

- Approximate dot products:
  - Cheaper, e.g., learn in smaller \( m \) dimensional space

- Only need logarithmic number of dimensions!
  - \( N \) data points, approximate dot-product within \( \epsilon > 0 \):
    \[
    m = O \left( \frac{\log N}{\epsilon^2} \right)
    \]

- But all sparsity is lost

LSH Example: Sparser Random Projection for Dot Products

- Pick random vectors \( v^{(i)} \)
- Simple 0/1 projection: \( h_i(x) = \)
- Now, each vector is approximated by a bit-vector
- Dot-product approximation:
LSH for Approximate Neighbor Finding

- Very similar elements fall in exactly same bin:
- And, nearby bins are also nearby:
- Simple neighbor finding with LSH:
  - For bins $b$ of increasing hamming distance to $h(x)$:
    - Look for neighbors of $x$ in bin $b$
  - Stop when run out of time
- Pick $m$ such that $N/2^m$ is “smallish”

Hash Kernels: Even Sparse LSH for Learning

- Two big problems with random projections:
  - Data is sparse, but random projection can be a lot less sparse
  - You have to sample $m$ huge random projection vectors
    - And, we still have the problem with new dimensions, e.g., new words
- **Hash Kernels**: Very simple, but powerful idea: combine sketching for learning with random projections
- Pick 2 hash functions:
  - $h$: Just like in Min-Count hashing
  - $\xi$: Sign hash function
    - Removes the bias found in Min-Count hashing (see homework)
- Define a “kernel”, a projection $\phi$ for $x$:
Hash Kernels, Random Projections and Sparsity

\[ \phi_i(x) = \sum_{j : h(j) = i} \xi(j)x_j \]

- Hash Kernel as a random projection:
- Random projection vector for coordinate \( i \) of \( \phi \):
- Implicitly define projection by \( h \) and \( \xi \), so no need to compute apriori and automatically deal with new dimensions
- Sparsity of \( \phi \), if \( x \) has \( s \) non-zero coordinates:

What you need to know

- Locality-Sensitive Hashing (LSH): nearby points hash to the same or nearby bins
- LSH use random projections
  - Only \( O(\log N/\epsilon^2) \) vectors needed
  - But vectors and results are not sparse
- Use LSH for nearest neighbors by mapping elements into bins
  - Bin index is defined by bit vector from LSH
  - Find nearest neighbors by going through bins
- Hash kernels:
  - Sparse representation for feature vectors
  - Very simple, use two hash function
    - Can even use one hash function, and take least significant bit to define \( \xi \)
  - Quickly generate projection \( \phi(x) \)
  - Learn in projected space