Announcements

•

Course outline

• Probabilistic algorithms and average case analysis
• Sublinear space algorithms for streaming
• Geometry and searching
  • Nearest neighbor problems
  • Low dimensional searching
  • Higher dimensions
  • Locally sensitive hashing
  • Document similarity
  • Linear programming

High dimensional searching

• Many data sets are high dimensional
  • High dimension can mean a mathematical space, such as $\mathbb{R}^d$, or a structure, such as bag-of-words representation of documents
• Canonical problem:
  • Given a new datum $x$, find the closest element $y$ in the dataset
  • Lots of things need to be defined, like “closest”
  • Think of the data set as being very large, so we would like a mechanism that avoids having to do comparisons with all elements

Tentative outline

• Concepts
• Coding theory

Concepts

• Product space
  • Mathematically – Cartesian Product
• Euclidean space, $\mathbb{R}^d$
  • Other spaces, $\mathbb{Z}^d$
• Metric
  • Distance measure, $d(x,y)$, $d : A \times A \rightarrow [0, \infty)$
  • Properties
    • $d(x,y) = 0$ if and only if $x = y$
    • $d(x,y) = d(y,x)$
    • $d(x,z) \leq d(x,y) + d(y,z)$
    • $d(x,y) \geq 0$
Closest points and approximate closest points

• Set of points S
• Given query point y, find a point in S closest to y
• Approximate closest point
  • Suppose the closest point distance from y to a point in S is r
  • Find a point in S that has distance (1+ε)r from y

Intuition and where it breaks down

• My pictures are in R^2
• I can imagine what happens in R^3
• Higher dimensions are much, much harder
• Imagine an N dimensional sphere

<table>
<thead>
<tr>
<th>Dim</th>
<th>Lower</th>
<th>Upper</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>8</td>
<td>240</td>
<td>240</td>
</tr>
<tr>
<td>12</td>
<td>1960</td>
<td>19656</td>
</tr>
<tr>
<td>16</td>
<td>196560</td>
<td>196560</td>
</tr>
<tr>
<td>20</td>
<td>17400</td>
<td>36764</td>
</tr>
<tr>
<td>24</td>
<td>4320</td>
<td>3183</td>
</tr>
</tbody>
</table>

Warm up – Coding theory

• Problem – sending data across a noisy channel

<table>
<thead>
<tr>
<th>Codeword</th>
<th>Binary Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>00000000</td>
</tr>
<tr>
<td>001</td>
<td>00100101</td>
</tr>
<tr>
<td>010</td>
<td>01001001</td>
</tr>
<tr>
<td>011</td>
<td>01101101</td>
</tr>
<tr>
<td>100</td>
<td>10010010</td>
</tr>
<tr>
<td>101</td>
<td>10110110</td>
</tr>
<tr>
<td>110</td>
<td>11011010</td>
</tr>
<tr>
<td>111</td>
<td>11111111</td>
</tr>
</tbody>
</table>

Idea one – parity bit

• Add a parity bit or check sum
  • Message x_1, x_2, ..., x_k
  • Let y = x_1 \oplus x_2 \oplus ... \oplus x_k (exclusive OR)
  • Send message x_1, x_2, ..., x_k, y
  • Resulting message has even parity
  • If a block is received with odd parity, at least one bit was flipped
  • Single error detection

Idea two - redundancy

• Make three copies of each code word
• One error correcting
• Every code word is a distance at least 3
• But this is a very dumb code

Block codes

• Coding theory / Information theory started in 1940s at Bell Labs
  • Clause Shannon, Richard Hamming

<table>
<thead>
<tr>
<th>(n,k,d)q</th>
<th>Alphabet size q (omit for 2), block length n, message length k, distance d</th>
</tr>
</thead>
<tbody>
<tr>
<td>k/n gives the rate</td>
<td></td>
</tr>
<tr>
<td>(n,k,d)_2 code can detect d-1 errors and correct \lceil (d-1)/2 \rceil errors</td>
<td></td>
</tr>
</tbody>
</table>
Hamming(7,4) code

- Linear code with 3 parity bits
- Basis vectors
  - $[1,1,0,0,0,0,0]$
  - $[1,0,1,1,0,0,0]$
  - $[0,1,0,1,0,1,0]$
  - $[1,1,0,1,0,0,1]$
- Encoding / Decoding / Error correction are linear algebra operations over $\mathbb{Z}_2$

Golay Code: $G_{24}$ $[24,12,8]_2$ and $G_{23}$ $[23,12,7]_2$

- Closely related codes, 12 dimensional subspaces of $\mathbb{Z}_2^{24}$ and $\mathbb{Z}_2^{23}$ respectively
- $G_{24}$ is used because it is 3 bytes
- $G_{23}$ is a perfect code. Spheres of radius 3 around the code words partition the vector space
- Imagine a 23 dimensional sphere of radius three centered at $z$, find the codeword in the sphere

Low dimensional problems

- $S = \{x_1, x_2, \ldots, x_n\}$
- Given an value $y$, find the closest point in $S$ to $y$.
  - $\min_{i} d(y, x_i)$

How do we solve this in $\mathbb{R}^1$

Issues

- Static versus dynamic data structures
- Average case versus worst case
- Numerical precision of coordinates

And now for $\mathbb{R}^2$
What is the distance function?

- Standard Euclidean distance – $L^2$ Norm
  \[ \left\| (x, y) \right\|_2 = \sqrt{x^2 + y^2} \]
- $L^p$ Norm
  \[ \left\| (x, y) \right\|_p = \sqrt[p]{x^p + y^p} \]
- $L^1$ Norm
  \[ \left\| (x, y) \right\|_1 = |x + y| \]
- $L^\infty$ Norm
  \[ \left\| (x, y) \right\|_\infty = \max(x, y) \]

Data structures for 2-d nearest neighbor

- Unlike 1-d we do not have a linear order on the points
- Multiple options are available (and variants exist)
  - Quad trees
  - $k$-d trees
  - Voronoi diagram

Quad Tree

- Start with a bounding square
- Each level divides a square into four quadrants
- Search explores cells which may contain nearest neighbor
  - Track best-so-far distance to prune sub trees in recursive tree traversal
- Depth is determined by closest pair distance

Voronoi diagram

- For each point $x$, Voronoi region is the set of points (in $R^2$) where $x$ is the nearest neighbor in $S$
- Between each pair of points we can look at the separating half spaces
- A point’s Voronoi region is the intersection of half spaces (and convex)
- The number of segments is $O(N)$

Building the Voronoi diagram

- Lots of algorithms exist
- It can be done in $O(n \log n)$ time
- Programming is a challenge
  - Lots of special cases
  - Careful numerical programming
  - Hard to debug
- Most practical algorithm is probably to insert points in random order into an existing diagram

Search in a Voronoi diagram

- Need to overlay a search structure on top of the diagram
- Can use a sequence of separating segments
- Binary space partition trees can be used
- In theory, this can be done in $O(\log n)$ query time
What about 3 dimensions?

- Quad trees generalize to oct-trees in 3d, with 8 children instead of 4.
- Unfortunately, the 3-d voronoi tessellation (honeycomb) can have size $n^2$.
- Proof: divide the points into two sets $A$ and $B$, and put $A$ and $B$ on separate arcs. This can be done so that each point $a_i$ in $A$ shares a face with each $b_j$ in $B$. 