Lecture outline

• Nearest-neighbor search in low dimensions
  – kd-trees

• Nearest-neighbor search in high dimensions
  – LSH

• Applications to data mining
Definition

• Given: a set $X$ of $n$ points in $\mathbb{R}^d$
• Nearest neighbor: for any query point $q \in \mathbb{R}^d$
  return the point $x \in X$ minimizing $D(x,q)$

• **Intuition:** Find the point in $X$ that is the closest to $q$
Motivation

• **Learning**: Nearest neighbor rule
• **Databases**: Retrieval
• **Data mining**: Clustering
• Donald Knuth in vol.3 of *The Art of Computer Programming* called it the post-office problem, referring to the application of assigning a resident to the *nearest-post office*
Nearest-neighbor rule
MNIST dataset “2”
Methods for computing NN

• **Linear scan:** $O(nd)$ time

• This is pretty much all what is known for exact algorithms with theoretical guarantees

• In practice:
  – *kd-trees* work “well” in “low-medium” dimensions
2-dimensional kd-trees

- A data structure to support range queries in $\mathbb{R}^2$
  - Not the most efficient solution in theory
  - Everyone uses it in practice

- Preprocessing time: $O(n \log n)$
- Space complexity: $O(n)$
- Query time: $O(n^{1/2} + k)$
2-dimensional kd-trees

- Algorithm:
  - Choose x or y coordinate (alternate)
  - Choose the median of the coordinate; this defines a horizontal or vertical line
  - Recurse on both sides

- We get a binary tree:
  - Size $O(n)$
  - Depth $O(\log n)$
  - Construction time $O(n\log n)$
Construction of kd-trees
Construction of kd-trees
Construction of kd-trees
Construction of kd-trees
Construction of kd-trees
The complete kd-tree
Region of node $v$

Region($v$) : the subtree rooted at $v$ stores the points in black dots
Searching in kd-trees

• Range-searching in 2-d
  – Given a set of $n$ points, build a data structure that for any query rectangle $R$ reports all point in $R$
kd-tree: range queries

- Recursive procedure starting from $v = \text{root}$
- **Search** $(v,R)$
  - If $v$ is a leaf, then report the point stored in $v$ if it lies in $R$
  - Otherwise, if $\text{Reg}(v)$ is contained in $R$, report all points in the $\text{subtree}(v)$
  - Otherwise:
    - If $\text{Reg}(\text{left}(v))$ intersects $R$, then $\text{Search}(\text{left}(v), R)$
    - If $\text{Reg}(\text{right}(v))$ intersects $R$, then $\text{Search}(\text{right}(v), R)$
Query time analysis

- We will show that Search takes at most $O(n^{1/2} + P)$ time, where $P$ is the number of reported points.
  - The total time needed to report all points in all sub-trees is $O(P)$.
  - We just need to bound the number of nodes $v$ such that $\text{region}(v)$ intersects $R$ but is not contained in $R$ (i.e., boundary of $R$ intersects the boundary of $\text{region}(v)$).
  - *gross overestimation*: bound the number of $\text{region}(v)$ which are crossed by any of the 4 horizontal/vertical lines.
Query time (Cont’d)

• **Q(n):** max number of regions in an n-point kd-tree intersecting a (say, vertical) line?

![Diagram showing a vertical line intersecting regions and a tree structure]

- If \( \ell \) intersects region(\( v \)) (due to vertical line splitting), then after two levels it intersects 2 regions (due to 2 vertical splitting lines)
- The number of regions intersecting \( \ell \) is \( Q(n)=2+2Q(n/4) \) \( \rightarrow \) \( Q(n)=(n^{1/2}) \)
d-dimensional kd-trees

- A data structure to support range queries in $\mathbb{R}^d$
- Preprocessing time: $O(n \log n)$
- Space complexity: $O(n)$
- Query time: $O(n^{1-1/d} + k)$
Construction of the $d$-dimensional $kd$-trees

• The construction algorithm is similar as in 2-d

• At the root we split the set of points into two subsets of same size by a hyperplane vertical to $x_1$-axis

• At the children of the root, the partition is based on the second coordinate: $x_2$-coordinate

• At depth $d$, we start all over again by partitioning on the first coordinate

• The recursion stops until there is only one point left, which is stored as a leaf
Locality-sensitive hashing (LSH)

- **Idea**: Construct hash functions $h: \mathbb{R}^d \rightarrow U$ such that for any pair of points $p, q$:
  - If $D(p, q) \leq r$, then $\Pr[h(p) = h(q)]$ is high
  - If $D(p, q) \geq cr$, then $\Pr[h(p) = h(q)]$ is small

- Then, we can solve the “approximate NN” problem by hashing

- LSH is a general framework; for a given $D$ we need to find the right $h$
Approximate Nearest Neighbor

- Given a set of points $X$ in $\mathbb{R}^d$ and query point $q \in \mathbb{R}^d$
- Approximate $r$-Nearest Neighbor search returns:
  - Returns $p \in P$, $D(p,q) \leq r$
  - Returns NO if there is no $p' \in X$, $D(p',q) \leq cr$
Locality-Sensitive Hashing (LSH)

• A family $H$ of functions $h: \mathbb{R}^d \rightarrow \mathbb{U}$ is called $(P_1, P_2, r, cr)$-sensitive if for any $p, q$:
  – if $D(p, q) \leq r$, then $\Pr[h(p) = h(q)] \geq P_1$
  – If $D(p, q) \geq cr$, then $\Pr[h(p) = h(q)] \leq P_2$

• $P_1 > P_2$

• Example: Hamming distance
  – LSH functions: $h(p) = p_i$, i.e., the $i$-th bit of $p$
  – Probabilities: $\Pr[h(p) = h(q)] = 1 - D(p, q)/d$
Algorithm -- preprocessing

- $g(p) = <h_1(p), h_2(p), \ldots, h_k(p)>

- Preprocessing
  - Select $g_1, g_2, \ldots, g_L$
  - For all $p \in X$ hash $p$ to buckets $g_1(p), \ldots, g_L(p)$
  - Since the number of possible buckets might be large we only maintain the non empty ones

- Running time?
Algorithm -- query

• Query $q$:
  – Retrieve the points from buckets $g_1(q), g_2(q), \ldots, g_L(q)$ and let points retrieved be $x_1, \ldots, x_L$
    • If $D(x_i, q) \leq r$ report it
    • Otherwise report that there does not exist such a NN
  – Answer the query based on the retrieved points
  – Time $O(dL)$
Applications of LSH in data mining

• Numerous....
Applications

• Find pages with similar sets of words (for clustering or classification)

• Find users in Netflix data that watch similar movies

• Find movies with similar sets of users

• Find images of related things
How would you do it?

• Finding very similar items might be computationally demanding task

• We can relax our requirement to finding *somewhat similar* items
Running example: comparing documents

• Documents have common text, but no common topic

• Easy special cases:
  – Identical documents
  – Fully contained documents (letter by letter)

• General case:
  – Many small pieces of one document appear out of order in another. What do we do then?
Finding similar documents

• Given a collection of documents, find pairs of documents that have lots of text in common
  – Identify mirror sites or web pages
  – Plagiarism
  – Similar news articles
Key steps

• **Shingling**: convert documents (news articles, emails, etc) to sets

• **LSH**: convert large sets to *small signatures*, while preserving the similarity

• Compare the signatures instead of the actual documents
Shingles

- A **k-shingle** (or **k-gram**) is a sequence of **k** characters that appears in a document.

- If doc = abcab and k=3, then 2-singles: \{ab, bc, ca\}

- Represent a document by a set of **k**-shingles.
Assumption

• Documents that have similar sets of k-shingles are similar: same text appears in the two documents; the position of the text does not matter

• What should be the value of k?
  – What would large or small k mean?
Data model: sets

• Data points are represented as sets (i.e., sets of shingles)

• Similar data points have large intersections in their sets
  – Think of documents and shingles
  – Customers and products
  – Users and movies
Similarity measures for sets

• Now we have a set representation of the data

• Jaccard coefficient

• $A, B$ sets (subsets of some, large, universe $U$)

$$sim(A, B) = \frac{|A \cap B|}{|A \cup B|}$$
Find similar objects using the Jaccard similarity

• Naïve method?

• Problems with the naïve method?
  – There are too many objects
  – Each object consists of too many sets
Speeding up the naïve method

- Represent every object by a signature (summary of the object)
- Examine pairs of signatures rather than pairs of objects
- Find all similar pairs of signatures
- **Check point:** check that objects with similar signatures are actually similar
Still problems

• Comparing large number of signatures with each other may take too much time (although it takes less space)

• The method can produce pairs of objects that might not be similar (false positives). The check point needs to be enforced
Creating signatures

• For object \( x \), signature of \( x \) (sign(\( x \))) is much smaller (in space) than \( x \)

• For objects \( x, y \) it should hold that \( \text{sim}(x,y) \) is almost the same as \( \text{sim}(\text{sing}(x),\text{sign}(y)) \)
Intuition behind Jaccard similarity

• Consider two objects: \( x, y \)

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

• \( a \): # of rows of form same as \( a \)
• \( \text{sim}(x,y) = \frac{a}{a+b+c} \)
A type of signatures -- minhashes

• Randomly **permute** the rows

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

• **h(x):** first row (in permuted data) in which column **x** has an **1**

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

• Use several (e.g., 100) independent hash functions to design a signature
“Surprising” property

• The probability (over all permutations of rows) that $h(x)=h(y)$ is the same as $\text{sim}(x,y)$

• Both of them are $a/(a+b+c)$

• So?
  – The similarity of signatures is the fraction of the hash functions on which they agree
Minhash algorithm

• Pick $k$ (e.g., 100) permutations of the rows

• Think of $\text{sign}(x)$ as a new vector

• Let $\text{sign}(x)[i]$: in the $i$-th permutation, the index of the first row that has 1 for object $x$
Example of minhash signatures

• Input matrix

<table>
<thead>
<tr>
<th></th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>X4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>X4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

1 2 1 2
Example of minhash signatures

- **Input matrix**

<table>
<thead>
<tr>
<th></th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

2 1 3 1
Example of minhash signatures

• Input matrix
Example of minhash signatures

- Input matrix

<table>
<thead>
<tr>
<th></th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
&\approx \\
&\begin{array}{cccc}
    x1 & x2 & x3 & x4 \\
    1 & 2 & 1 & 2 \\
    2 & 1 & 3 & 1 \\
    3 & 1 & 3 & 1 \\
\end{array}
\end{align*}
\]

- Actual signs

<table>
<thead>
<tr>
<th></th>
<th>actual</th>
<th>signs</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x1,x2)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(x1,x3)</td>
<td>0.75</td>
<td>2/3</td>
</tr>
<tr>
<td>(x1,x4)</td>
<td>1/7</td>
<td>0</td>
</tr>
<tr>
<td>(x2,x3)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(x2,x4)</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>(x3,x4)</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Is it now feasible?

• Assume a billion rows
• Hard to pick a random permutation of 1...billion
• Even representing a random permutation requires 1 billion entries!!!
• How about accessing rows in permuted order?
• ☹️
Being more practical

• Approximating row permutations: pick $k=100$ hash functions $(h_1,\ldots,h_k)$

for each row $r$
  for each column $c$
    if $c$ has 1 in row $r$
      for each hash function $h_i$
        if $h_i(r)$ is a smaller value than $M(i,c)$ then
          $M(i,c) = h_i(r)$;

$M(i,c)$ will become the smallest value of $h_i(r)$ for which column $c$ has 1 in row $r$; i.e., $h_i(r)$ gives order of rows for $i$-th permutation.
Example of minhash signatures

- Input matrix

<table>
<thead>
<tr>
<th></th>
<th>x1</th>
<th>x2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ h(r) = r + 1 \mod 5 \]
\[ g(r) = 2r + 1 \mod 5 \]