Data Mining
Learning from Large Data Sets

Lecture 2 – Nearest neighbor search

263-5200-00L
Andreas Krause
Announcement

- Homework 1 out by tomorrow
Approximate retrieval
- Given a query, find “most similar” item in a large data set
- *Applications*: Google Goggles, Shazam, ...

**Supervised learning** (Classification, Regression)
- Learn a concept (function mapping queries to labels)
- *Applications*: Spam filtering, predicting price changes, ...

**Unsupervised learning** (Clustering, dimension reduction)
- Identify clusters, “common patterns”; anomaly detection
- *Applications*: Recommender systems, fraud detection, ...

**Interactive data mining**
- Learning through experimentation / from limited feedback
- *Applications*: Online advertising, opt. UI, learning rankings, ...
Today:

Fast nearest neighbor search in high dimensions
Multimedia retrieval

Google.com

shazam.com
Image completion

[Hays and Efros, SIGGRAPH 2007]
Nearest-neighbor search
A function
\[ d : S \times S \rightarrow \mathbb{R} \]

is called a **distance function (metric)** if it is

- **Nonnegative:** \( \forall s, t \in S : d(s, t) \geq 0 \)
- **Discerning:** \( d(s, t) = 0 \iff s = t \)
- **Symmetric:** \( \forall s, t : d(s, t) = d(t, s) \)
- **Triangle inequality:**
\[
\forall s, t, r : d(s, t) + d(t, r) \geq d(s, r)
\]
Representing objects as vectors

- Bag of words for documents
- Feature vectors for images (SIFT, GIST, PHOG, etc.)
- ... 
- Allows to use the same distances / same algorithms for different object types
Examples: Distance of vectors in $\mathbb{R}^D$

- **Euclidean distance**
  
  $$d_2(x, x') = \sqrt{\sum_{i=1}^{D} (x_i - x'_i)^2}$$

- **Manhattan distance**
  
  $$d_1(x, x') = \sum_{i=1}^{D} |x_i - x'_i|$$

- **$\ell^p$ distances:**

  $$d_p(x, x') = \left( \sum_{i=1}^{D} |x_i - x'_i|^p \right)^{1/p}$$

  
  $p = \infty$, $d_\infty(x, x') = \max_i |x_i - x'_i|$
Cosine distance

\[ d(x, x') = \arccos \frac{x^T x'}{\|x\|_2 \|x'\|_2} \]
Edit distance

**Edit distance**: How many inserts and deletes are necessary to transform one string to another?

Example:
- \( d(\text{"The quick brown fox"}, \text{"The quikc brwn fox"}) \approx 3 \)
- \( d(\text{"GATTACA"}, \text{"ATACAT"}) \)

- Allows various extensions (mutations; reversal; ...)
- Can compute in polynomial time, but expensive for large texts
- \( \Rightarrow \) We will focus on vector representation
Many real-world problems are high-dimensional

- Text on the web
  - Billions of documents, millions of terms
  - In Bag Of Words representation, each term is a dimension..
- Scene completion, image classification, ...
  - Large # of image features
- Scientific data
  - Large number of measurements
- Product recommendations and advertising
  - Millions of customers, millions of products
  - Traces of behavior (websites visited, searches, ...)
Curse of dimensionality

- Suppose we would like to find neighbors of maximum distance at most .1 in $[0,1]^D$
- Suppose we have $N$ data points sampled uniformly at random from $[0,1]^D$
Theorem [Beyer et al. ‘99] Fix $\varepsilon > 0$ and $N$. Under fairly weak assumptions on the distribution of the data

$$\lim_{D \to \infty} P[d_{\text{max}}(N, D) \leq (1 + \varepsilon)d_{\text{min}}(N, D)] = 1$$
The Blessing of Large Data
10 nearest neighbors from a collection of 20,000 images

Hays and Efros, SIGGRAPH 2007
10 nearest neighbors from a collection of 2 million images

Hays and Efros, SIGGRAPH 2007
Application: Find similar documents

- Find “near-duplicates” among a large collection of documents
  - Find clusters in a document collection (blog articles)
  - Spam detection
  - Detect plagiarism
  - ...

- What does “near-duplicates” mean?
Near-duplicates

Naïve approach:
- Represent documents as “bag of words”
- Apply nearest-neighbor search on resulting vectors

Doesn’t work too well in this setting.
To keep track of word order, extract **k-shingles** (aka **k-grams**)

Document represented as “bag of k-shingles”

Example: \( a \ b \ c \ a \ b \)

2 shingles = \( \{ a \ b, b c, c a \} \)
Shingling implementation

- How large should one choose $k$?
  - Long enough s.t. the don’t occur “by chance”
  - Short enough so that one expects “similar” documents to share some $k$-shingles

- Storing shingles
  - Want to save space by compressing
  - Often, simply hashing works well (e.g., hash 10-shingle to 4 bytes)
Comparing shingled documents

- Documents are now represented as **sets of shingles**
- Want to compare two sets
- E.g.: $A = \{1,3,7\}$; $B = \{2,3,4,7\}$

\[
\begin{align*}
    \text{Overlap} & \quad |A \cap B| = 2 \\
    \text{Total #} & \quad |A \cup B| = 5
\end{align*}
\]
Jaccard distance

- Jaccard similarity:

\[
\text{Sim}(A, B) = \frac{|A \cap B|}{|A \cup B|} \in [0, 1]
\]

- Jaccard distance:

\[
d(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}
\]
Example

\[ \sin(A, B) = \frac{3}{\pi} \]

\[ \alpha(A, B) = \frac{5}{\pi} \]
Near-duplicate detection

- Want to find documents that have similar sets of k-shingles

- Naïve approach:
  - For i=1:N
    - For j=1:N
      - Compute d(i,j)
      - If d(i,j) < ε then declare near-duplicate

- Infeasible even for moderately large N 😞

- Can we do better??
Warm-up

- Given a large collection of documents, determine whether there exist **exact** duplicates?

  - Compute hash code / checksum (e.g., MD5) for all documents
  - Check whether the same checksum appears twice
  - Both can be easily parallelized
**Locality sensitive hashing**

- **Idea:** Create hash function that maps “similar” items to same bucket

- **Key problem:** Is it possible to construct such hash functions??
  - Depends on the distance function
  - Possible for Jaccard distance!! 😊
  - Some other distance functions work as well
### Shingle Matrix

<table>
<thead>
<tr>
<th>Shingles</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documents</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Min-hashing

Simple hash function, constructed in the following way:

- Use random permutation $\pi$ to reorder the rows of the matrix
  - Must use same permutation for all columns $C$!!
- $h(C) =$ minimum row number in which permuted column contains a 1

$h(C) = h_{\pi}(C) = \min_{i:C(i)=1} \pi(i)$
Min-hashing example

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

Input matrix

$\Rightarrow [2, 1, 2, 1]$
# Min-hashing example

**Input matrix**

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

\( h \)

\[
\begin{array}{c}
2 \\
1 \\
2 \\
1 \\
\end{array}
\]
Min-hashing property

- Want that similar documents (columns) have same value of hash function (with high probability)

- Turns out it holds that

\[
\Pr[h(C_1) = h(C_2)] = \text{Sim}(C_1, C_2)
\]
Proof

\[ \begin{align*}
C_1 & \quad C_2 \\
0 & \quad 0 \\
0 & \quad 0 \\
0 & \quad 1 \\
1 & \quad 1 \\
1 & \quad 0 \\
1 & \quad 0 \\
1 & \quad 0
\end{align*} \]

\[ \begin{align*}
\text{4 cases} & \quad \#0\text{cc} \\
1 & \quad 1 & \quad a \\
1 & \quad 0 & \quad b \\
0 & \quad 1 & \quad c \\
0 & \quad 0 & \quad d
\end{align*} \]

\[ \sin(C_1, C_2) = \frac{a}{a + b + c} \]
Proof

Theorem: rows

in $TT$-order

Step upon row that contains at least one 1

What's the prob. that row is of type $\mid 1 \mid$

$$P(\ " \ ) = \frac{a}{a+b+c}$$

$\square$
Suppose we would like to find all duplicates with more than 90% similarity

Apply min-hash function to all documents, and look for candidate pairs (documents hashed to same bucket)

- How many 90%-duplicates will we find? \( \approx 90\% \)
- How many 90%-duplicates will we miss? \( \approx 10\% \)
- How can we reduce the number of misses?
Reducing the “misses”

- Apply multiple *independently random* hash functions
- Consider candidate pair of near duplicates if at least one of the functions hashes to same bucket
- What’s the probability of a “miss” with k functions?

\[
P(\text{"miss"}) = d(c_1, c_2)^k \\
= (1 - s)^k \quad s = \text{Sim}(c_1, c_2)
\]
Thus, using multiple independent hash functions can exponentially reduce probability of misses!
Min-hash signatures

Input matrix

<table>
<thead>
<tr>
<th>1</th>
<th>4</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

1 0 1 0
1 0 0 1
0 1 0 1
0 1 0 1
0 1 0 1
1 0 1 0
1 0 1 0

Signature matrix \( M \)

<table>
<thead>
<tr>
<th>2</th>
<th>1</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Similarities:

<table>
<thead>
<tr>
<th>1-3</th>
<th>2-4</th>
<th>1-2</th>
<th>3-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.75</td>
<td>0.75</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.67</td>
<td>1.00</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Implementing min-hashing

- Difficult to randomly permute a data set with a billion rows
- Even representing a permutation of size $10^9$ is expensive
- Accessing rows in permuted order is infeasible (requires random access)
Approximate min-hashing

- Directly represent permutation \( \pi \) through hash function \( h \! \):
  \[
  \pi(i) = h(i) = a_i + b \mod m
  \]

- Could happen that \( h(i) = h(j) \) for \( i \neq j \), but this is rare for good \( h \)

**Note:** Will use same notation for \( h(r) \) and \( h(C) \)

\[
 h(C) = \min_{i: C(i) = 1} h(i)
\]

- Suppose \( h(r) < h(s) \). Then row \( r \) appears before \( s \) in \( \pi \)
- Why is this useful?
  - Can store \( h \) very efficiently
  - Allows to process data matrix row-wise..
Example

<table>
<thead>
<tr>
<th>Row</th>
<th>C1</th>
<th>C2</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>

$h(x) = x \mod 5$

$h(1)=1, h(2)=2, h(3)=3, h(4)=4, h(5)=0$

$g(x) = 2x+1 \mod 5$

$g(1)=3, g(2)=0, g(3)=2, g(4)=4, g(5)=1$

\[
M = \begin{bmatrix}
1 & 0 \\
2 & 0
\end{bmatrix}
\]
False positives

- Increasing number of hash tables reduces false negative rate 😊
- Also increases false positive rate 😞
False positives

Ideally want:

\[ P(\text{Hit}) \]

\[ \text{Sim}(C_1, C_2) \]
Ingenious trick

- Signature matrix compactly represents similarity between documents
  - Jaccard distance $\sim$ $l_1$-distance of columns
  - Similar documents have similar signatures
- Naïve approach: Compare any pair of columns to see if their similar
  - Compact representation $\Rightarrow$ faster
  - Still $N^2$ comparisons 😞
- Will see how to hash columns s.t. with high probability
  - return similar pairs ($d(C1,C2) < \epsilon$)
  - do not return dissimilar pairs ($d(C1,C2) > \epsilon$)
Partitioning the signature matrix

$\text{Signature Matrix } M$

$b$ bands

$r$ rows per band

One signature
Hashing bands of M
Hashing the signature matrix

- Signature matrix $M$ partitioned into $b$ bands of $r$ rows.
- One hash table per band, independent hash functions
- For each band, hash its portion of each column to its hash table
  - For purpose of analysis, let’s assume there’s no “false collisions”
    - Doesn’t affect correctness of algorithm
- **Candidate pairs** are columns that hash to the same bucket for at least one band.
- Why is this useful?
Analysis of partitioning

- Suppose columns M1 and M2 have similarity \( s \)

\[
M_i = [B_{i,1}, \ldots, B_{i,b}]
\]

For fixed band \( j \), what's the prob.
that \( B_{1,j} \) and \( B_{2,j} \) collide?

\[
\begin{align*}
P(h(B_{1,j}) = h(B_{2,j})) &= s^r \\
\end{align*}
\]

\[
\begin{align*}
P(h(B_{1,j}) \neq h(B_{2,j})) &= 1 - s^r \\
& \text{"no collision in j\textsuperscript{th} band"}
\end{align*}
\]

\[
\begin{align*}
P( \text{no collision in any band}) &= (1 - s^r)^b \\
P( \text{collision in some band}) &= 1 - (1 - s^r)^b
\end{align*}
\]
One hash function

\[ r = 1, \quad b = 1 \]

\[ \text{Similarity} \]

\[ P(\text{hash hit}) \]
100 hash functions

\[ r = 10 \]
\[ b = 10 \]
100 hash functions

![Graph showing 100 hash functions with similarity on the x-axis and values on the y-axis. The graph includes curves for different values of r and b.]
1000 hash functions

Similarity

0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9

r=1 r=2 r=5 r=10 r=20 r=50
b=1000 b=200 b=100 b=50 b=20
10000 hash functions
Implementation details

- Tune $r$ and $b$ to achieve desired similarity threshold
- Typically favor
  - few false negatives
  - more false positives
- Do pairwise comparisons of all resulting candidate pairs (in main memory), to eliminate false positives
- Typically also compare the actual documents (needs another pass through the data)
Several slides adapted from the material accompanying the textbook (Anand Rajaraman, Stanford)