MMDS Secs. 3.2-3.4.

Slides adapted from: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

FINDING SIMILAR ITEMS

Slides also adapted from Prof. Srinivasan Parthasarathy @OSU

Task: Finding Similar Documents

- **Goal: Given a large number (N in the millions or billions) of documents, find "near duplicate" pairs**
- **Applications:**
  - Mirror websites, or approximate mirrors → remove duplicates
  - Similar news articles at many news sites → cluster

Task: Finding Similar Documents

- **Goal: Given a large number (N in the millions or billions) of documents, find "near duplicate" pairs**
- **Applications:**
  - Mirror websites, or approximate mirrors → remove duplicates
  - Similar news articles at many news sites → cluster

What are the challenges?

Task: Finding Similar Documents

- **Goal: Given a large number (N in the millions or billions) of documents, find "near duplicate" pairs**
- **Applications:**
  - Mirror websites, or approximate mirrors → remove duplicates
  - Similar news articles at many news sites → cluster
- **Problems:**
  - Many small pieces of one document can appear out of order in another
  - Too many documents to compare all pairs
  - Documents are so large or so many (scale issues)

Two Essential Steps for Similar Docs

1. ***Shingling:*** Convert documents to sets
2. ***Min-Hashing:*** Convert large sets to short signatures, while preserving similarity

Host of follow up applications

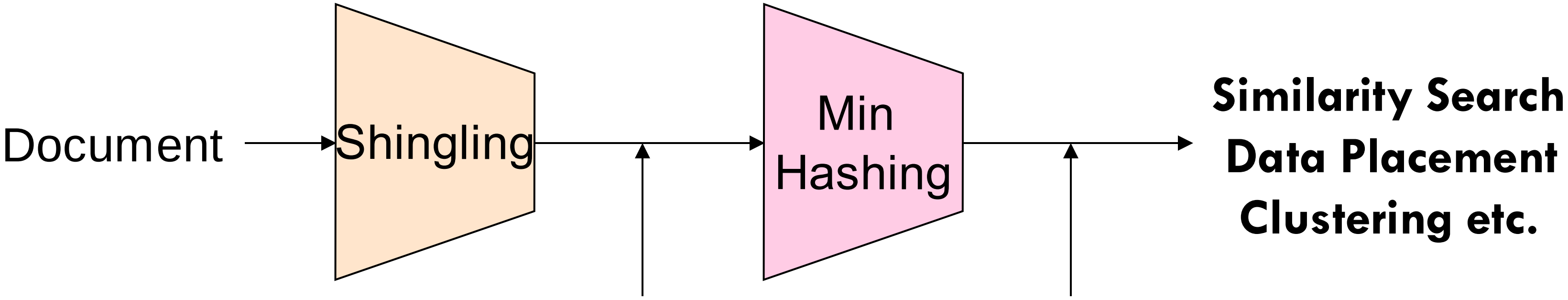
e.g. Similarity Search

Data Placement

Clustering etc.

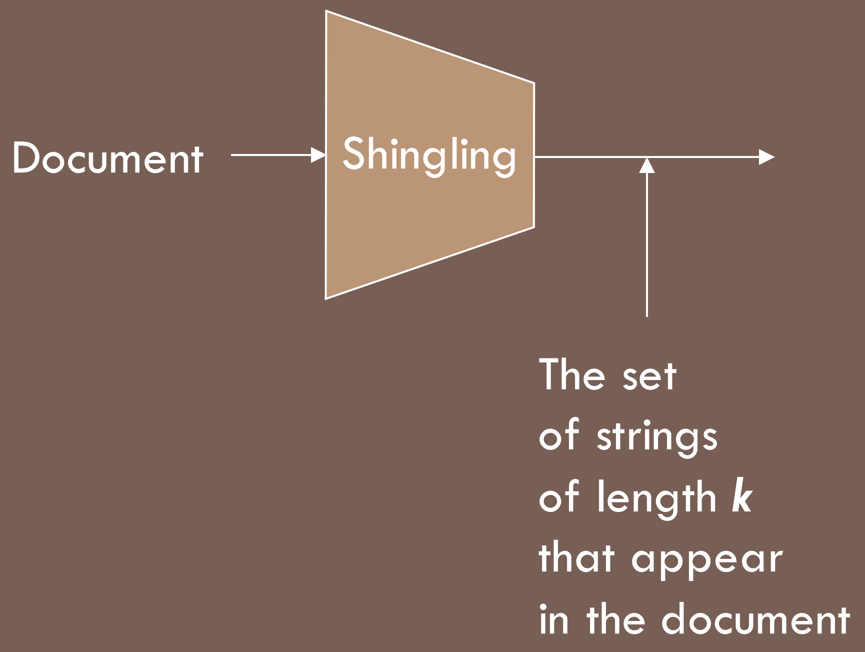
J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

The Big Picture



The set of strings of length ***k*** that appear in the doc-ument

Signatures: short integer vectors that represent the sets, and reflect their similarity



SHINGLING

Step 1: ***Shingling:*** Convert documents to sets

Documents as High-Dim Data

- **Step 1: *Shingling:* Convert documents to sets**

- **Simple approaches:**
  - Document = set of words appearing in document
  - Document = set of "important" words
  - Don't work well for this application. Why?

- **Need to account for ordering of words!**
- A different way: **Shingles!**

Define: Shingles

- A *k*-shingle (or *k*-gram) for a document is a sequence of *k* tokens that appears in the doc
  - Tokens can be characters, words or something else, depending on the application
  - Assume tokens = characters for examples

Define: Shingles

- A *k*-shingle (or *k*-gram) for a document is a sequence of *k* tokens that appears in the doc
  - Tokens can be characters, words or something else, depending on the application
  - Assume tokens = characters for examples

- **Example: k=2**; document $\mathbf{D_1}$ = abcab
  Set of 2-shingles: $\mathbf{S(D_1)}$ = {ab, bc, ca}

Define: Shingles

- A *k*-shingle (or *k*-gram) for a document is a sequence of *k* tokens that appears in the doc
  - Tokens can be characters, words or something else, depending on the application
  - Assume tokens = characters for examples

- **Example:** **k=2**; document $\mathbf{D_1}$ = abcab
  Set of 2-shingles: $\mathbf{S(D_1)}$ = {ab, bc, ca}

  - **Another option:** Shingles as a **bag** (multiset), count ab twice: $\mathbf{S'(D_1)}$ **=** {ab, bc, ca, ab}

Shingles: How to treat white-space chars?

Example 3.4 : If we use $k = 9$, but eliminate whitespace altogether, then we would see some lexical similarity in the sentences "`The plane was ready for touch down`". and "`The quarterback scored a touchdown`". However, if we retain the blanks, then the first has shingles `touch dow` and `ouch down`, while the second has `touchdown`. If we eliminated the blanks, then both would have `touchdown`. □

It makes sense to replace any sequence of one or more white-space characters (blank, tab, newline, etc.) by a single blank.

This way distinguishes shingles that cover two or more words from those that do not.

How to choose K?

- **Documents that have lots of shingles in common have similar text, even if the text appears in different order**

- **Caveat:** You must pick ***k*** large enough, or most documents will have most shingles
  - ***k*** = 5 is OK for short documents
  - ***k*** = 10 is better for long documents

Compressing Shingles

- To **compress long shingles**, we can **hash** them to (say) 4 bytes
  - Like a Code Book
  - If #shingles manageable → Simple dictionary suffices

e.g., 9-shingle => bucket number [0, 2^32 - 1]

(using 4 bytes instead of 9)

Compressing Shingles

- To **compress long shingles**, we can **hash** them to (say) 4 bytes
  - Like a Code Book
  - If #shingles manageable → Simple dictionary suffices

- **Doc represented by the set of hash/dict. values of its *k*-shingles**
  - **Idea:** Two documents could appear to have shingles in common, when the hash-values were shared

Compressing Shingles

- To **compress long shingles**, we can **hash** them to (say) 4 bytes
  - Like a Code Book
  - If #shingles manageable → Simple dictionary suffices

- **Doc represented by the set of hash/dict. values of its *k*-shingles**

- **Example:** **k=2**; document $\mathbf{D_1}$ = abcab
  Set of 2-shingles: $\mathbf{S(D_1)}$ = {ab, bc, ca}
  Hash the singles: $\mathbf{h(D_1)}$ = {1, 5, 7}

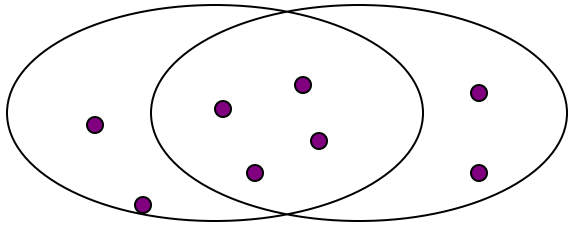
Similarity Metric for Shingles

- **Document D_1 is a set of its k-shingles $C_1 = S(D_1)$**

- Equivalently, each document is a 0/1 vector in the space of *k*-shingles
  - Each unique shingle is a dimension
  - Vectors are very sparse

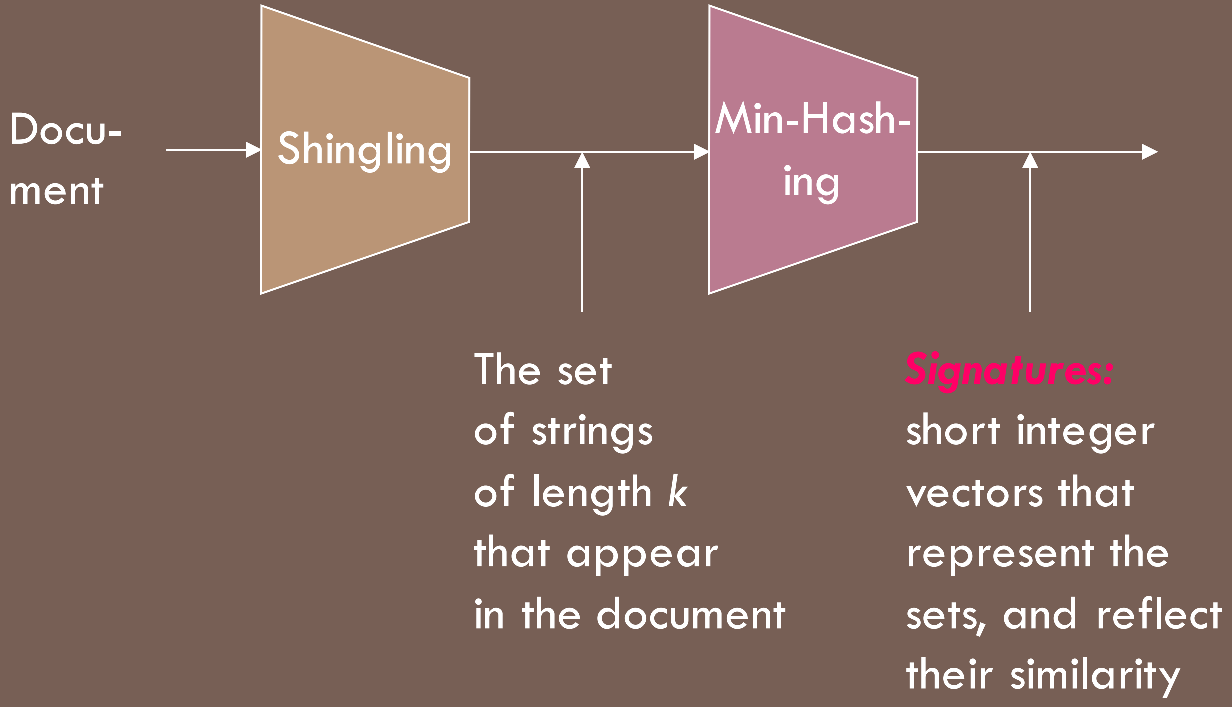
- **A natural similarity measure is the Jaccard similarity:**

$$sim(D_1, D_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$$



Motivation for Minhash/LSH

- **Suppose we need to find similar documents among $N = 1$ million documents**
- Naïvely, we would have to compute **pairwise Jaccard similarities** for **every pair of docs**
  - $N(N-1)/2 \approx 5*10^{11}$ comparisons
  - At 10^5 secs/day and 10^6 comparisons/sec, it would take **5 days**
- For $N = 10$ million, it takes more than a year…

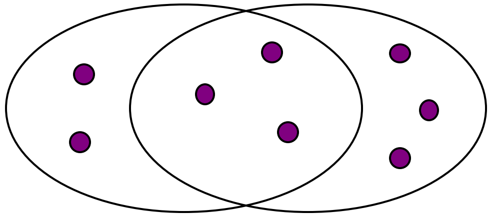


MINHASHING

Step 2: ***Minhashing:*** Convert **large variable length sets** to **short <u>fixed-length</u> signatures**, while **<u>preserving similarity</u>**

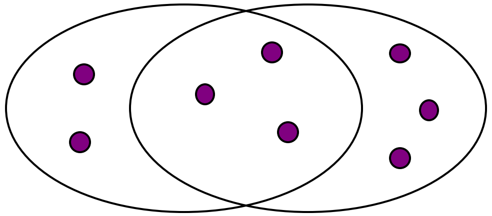
Encoding Sets as Bit Vectors

- Many similarity problems can be formalized as **finding subsets that have significant intersection**



Encoding Sets as Bit Vectors

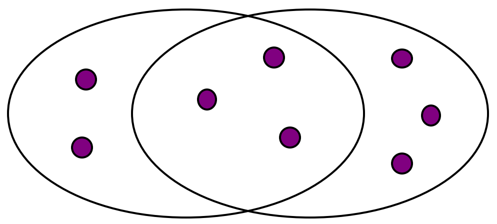
- Many similarity problems can be formalized as **finding subsets that have significant intersection**



- **Encode sets using 0/1 (bit, boolean) vectors**
  - One dimension per element in the universal set
- Interpret set intersection as bitwise **AND**, and set union as bitwise **OR**

Encoding Sets as Bit Vectors

- Many similarity problems can be formalized as **finding subsets that have significant intersection**



- **Encode sets using 0/1 (bit, boolean) vectors**
  - One dimension per element in the universal set
- Interpret set intersection as bitwise **AND**, and set union as bitwise **OR**
- **Example:** $C_1 = 10111$; $C_2 = 10011$
  - Size of intersection = **3**; size of union = **4,**
  - **Jaccard similarity** (not distance) = **3/4**
  - **Distance:** $d(C_1,C_2) = 1 - \text{(Jaccard similarity)} = 1/4$

From Sets to Boolean Matrices

Note: Transposed Document Matrix

- **Rows** = elements (shingles)
- **Columns** = sets (documents)
  - 1 in row **e** and column **s** if and only if **e** is a valid shingle of document represented by s
  - Column similarity is the Jaccard similarity of the corresponding sets (rows with value *1*)
  - **Typical matrix is sparse!**

Documents (columns) × Shingles (rows)

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

Outline: Finding Similar Columns

- **So far:**
  - A documents → a set of shingles
  - Represent a set as a boolean vector in a matrix

Documents

Shingles

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

Outline: Finding Similar Columns

- **So far:**
  - A documents → a set of shingles
  - Represent a set as a boolean vector in a matrix
- **Next goal: Find similar columns while computing small signatures**
  - **Similarity of columns == similarity of signatures**

Documents

Shingles

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

Outline: Finding Similar Columns

- **Next Goal: Find similar columns, Small signatures**

- **Naïve approach:**
  - **1) Signatures of columns:** small summaries of columns

Outline: Finding Similar Columns

- **Next Goal: Find similar columns, Small signatures**
- **Naïve approach:**
  - **1) Signatures of columns:** small summaries of columns
  - **2) Examine pairs of signatures** to find similar columns
    - **Essential:** Similarities of signatures and columns are related
  - **3) Optional:** Check that columns with similar signatures are really similar

Outline: Finding Similar Columns

- **Next Goal: Find similar columns, Small signatures**

- **Naïve approach:**
  - **1) Signatures of columns:** small summaries of columns
  - **2) Examine pairs of signatures** to find similar columns
    - **Essential:** Similarities of signatures and columns are related
  - **3) Optional:** Check that columns with similar signatures are really similar

- **Warnings:**
  - Comparing all pairs may take too much time: **Job for LSH**
    - These methods can produce false negatives, and even false positives (if the optional check is not made)

Hashing Columns (Signatures) : LSH principle

- **Key idea:** "hash" each column **C** to a small ***signature*** ***h(C)***, such that:
  - (1) ***h(C)*** is small enough that the signature fits in RAM
  - (2) $sim(C_1, C_2)$ is the same as the "similarity" of signatures $h(C_1)$ and $h(C_2)$

Hashing Columns (Signatures) : LSH principle

- **Key idea:** "hash" each column **C** to a small ***signature*** ***h(C)***, such that:
  - (1) *h(C)* is small enough that the signature fits in RAM
  - (2) $sim(C_1, C_2)$ is the same as the "similarity" of signatures $h(C_1)$ and $h(C_2)$

- **Goal: Find a hash function *h(·)* such that:**
  - If $sim(C_1,C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$
  - If $sim(C_1,C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$

Hashing Columns (Signatures) : LSH principle

- **Key idea:** "hash" each column **C** to a small ***signature*** ***h(C)***, such that:
  - (1) *h(C)* is small enough that the signature fits in RAM
  - (2) *sim*(C_1, C_2) is the same as the "similarity" of signatures $h(C_1)$ and $h(C_2)$

- **Goal: Find a hash function *h(·)* such that:**
  - If *sim*(C_1,C_2) is high, then with high prob. $h(C_1) = h(C_2)$
  - If *sim*(C_1,C_2) is low, then with high prob. $h(C_1) \neq h(C_2)$

- Hash docs into buckets. Expect that "most" pairs of near duplicate docs hash into the same bucket!

Min-Hashing

- **Goal: Find a hash function $h(\cdot)$ such that:**
  - if $sim(C_1,C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$
  - if $sim(C_1,C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$

- **Clearly, the hash function depends on the similarity metric:**
  - Not all similarity metrics have a suitable hash function

Min-Hashing

- **Goal: Find a hash function $h(\cdot)$ such that:**
  - if $sim(C_1,C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$
  - if $sim(C_1,C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$

- **Clearly, the hash function depends on the similarity metric:**
  - Not all similarity metrics have a suitable hash function

- **There is a suitable hash function for the Jaccard similarity:** It is called **Min-Hashing**

Min-Hashing

- Imagine the rows of the boolean matrix permuted under **random permutation** π

Min-Hashing

- Imagine the rows of the boolean matrix permuted under **random permutation** π

- Define a **"hash" function** $h_\pi(C)$ = the index of the **first** (in the permuted order π) row in which column **C** has value **1**:

$$h_\pi(C) = \min_\pi \pi(C)$$

Min-Hashing

- Imagine the rows of the boolean matrix permuted under **random permutation** π

- Define a **"hash" function** $h_\pi(C)$ = the index of the **first** (in the permuted order π) row in which column **C** has value **1**:

$$h_\pi(C) = \min_\pi \pi(C)$$

- Use several (e.g., 100) independent hash functions (that is, permutations) to create a signature of a column

Min-Hashing

- Imagine the rows of the boolean matrix permuted under **random permutation π**

- Define a **"hash" function $h_\pi(C)$** = the index of the **first** (in the permuted order π) row in which column **C** has value **1**:

$$h_\pi(C) = \min_\pi \pi(C)$$

- Use several (e.g., 100) independent hash functions (that is, permutations) to create a signature of a column

Zoo example (shingle size k=1)

Universe → { dog, cat, lion, tiger, mouse}

π_1 → [cat, mouse, lion, dog, tiger]

π_2 → [lion, cat, mouse, dog, tiger]

A = { mouse, lion }

Zoo example (shingle size k=1)

Universe $\longrightarrow$ { dog, cat, lion, tiger, mouse}

π_1 $\longrightarrow$ [cat, mouse, lion, dog, tiger]

π_2 $\longrightarrow$ [lion, cat, mouse, dog, tiger]

A = { mouse, lion }

$mh_1(A) = \min($ π_1{mouse, lion }) = mouse

$mh_2(A) = \min($ π_2{ mouse, lion }) = lion

Min-Hashing Example

Permutation π

2
3
7
6
1
5
4

Input matrix (Shingles x Documents)

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*

2	1	2	1

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Min-Hashing Example

2nd element of the permutation is the first to map to a 1

Permutation π

2
3
7
6
1
5
4

Input matrix (Shingles x Documents)

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*

2	1	2	1

Min-Hashing Example

Permutation π

2	4
3	2
7	1
6	3
1	6
5	7
4	5

Input matrix (Shingles x Documents)

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*

2	1	2	1
2	1	4	1

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Min-Hashing Example

2nd element of the permutation is the first to map to a 1

Permutation π

2	4
3	2
7	1
6	3
1	6
5	7
4	5

Input matrix (Shingles x Documents)

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*

2	1	2	1
2	1	4	1

4th element of the permutation is the first to map to a 1

Min-Hashing Example

Permutation π

2	4	3
3	2	4
7	1	7
6	3	2
1	6	6
5	7	1
4	5	5

Input matrix (Shingles x Documents)

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*

2	1	2	1
2	1	4	1
1	2	1	2

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Min-Hashing Example

Note: Another (equivalent) way is to store row indexes or raw shingles (e.g. mouse, lion):

1	5	1	5
2	3	1	3
6	4	6	4

2nd element of the permutation is the first to map to a 1

Permutation π

2	4	3
3	2	4
7	1	7
6	3	2
1	6	6
5	7	1
4	5	5

Input matrix (Shingles x Documents)

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*

2	1	2	1
2	1	4	1
1	2	1	2

4th element of the permutation is the first to map to a 1

Min-Hash Signatures

- **Pick K=100 random permutations of the rows**
- Think of ***sig*(C)** as a column vector
- ***sig*(C)[i] =** according to the *i*-th permutation, the index of the first row that has a 1 in column C

$$sig(C)[i] = \min(\pi_i(C))$$

- **Note:** The sketch (signature) of document C is small **~100 bytes!**
- **We achieved our goal! We "compressed" long bit vectors into short signatures**

Key Fact

For two sets A, B, and a min-hash function $mh_i()$:

$$Pr[mh_i(A) = mh_i(B)] = Sim(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Unbiased estimator for Sim using *K* hashes (notation policy – this is a different K from size of shingle)

$$\hat{Sim}(A, B) = \frac{1}{k} \sum_{i=1:k} I[mh_i(A) = mh_i(B)]$$

Min-Hashing Example

Permutation π

2	4	3
3	2	4
7	1	7
6	3	2
1	6	6
5	7	1
4	5	5

Input matrix (Shingles x Documents)

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*

2	1	2	1
2	1	4	1
1	2	1	2

Similarities:

	1-3	2-4	1-2	3-4
Col/Col	0.75	0.75	0	0
Sig/Sig	0.67	1.00	0	0

The Min-Hash Property

- **Choose a random permutation π**
- **Claim:** $\Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
- **Why?**

0	0
0	0
1	**1**
0	0
0	1
1	0

One of the two cols had to have 1 at position ***y***

The Min-Hash Property

- **Choose a random permutation π**
- **<u>Claim:</u>** $\Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
- **Why?**
  - Let **X** be a doc (set of shingles), $y \in X$ is a shingle

0	0
0	0
1	**1**
0	0
0	1
1	0

One of the two cols had to have 1 at position ***y***

The Min-Hash Property

- **Choose a random permutation π**
- **Claim:** $\Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
- **Why?**
  - Let **X** be a doc (set of shingles), $y \in X$ is a shingle
  - **Then:** $\Pr[\pi(y) = \min(\pi(X))] = 1/|X|$
    - It is equally likely that any $y \in X$ is mapped to the ***min*** element

0	0
0	0
1	**1**
0	0
0	1
1	0

One of the two cols had to have 1 at position ***y***

The Min-Hash Property

- **Choose a random permutation π**
- **Claim:** $\Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
- **Why?**
  - Let **X** be a doc (set of shingles), $y \in X$ is a shingle
  - **Then:** $\Pr[\pi(y) = \min(\pi(X))] = 1/|X|$
    - It is equally likely that any $y \in X$ is mapped to the *min* element
  - Let y be s.t. $\pi(y) = \min(\pi(C_1 \cup C_2))$
  - **Then either:** $\pi(y) = \min(\pi(C_1))$ if $y \in C_1$, **or**
    $\pi(y) = \min(\pi(C_2))$ if $y \in C_2$

0	0
0	0
1	**1**
0	0
0	1
1	0

One of the two cols had to have 1 at position ***y***

The Min-Hash Property

- **Choose a random permutation π**
- **<u>Claim:</u>** $\Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
- **Why?**
  - Let **X** be a doc (set of shingles), $y \in X$ is a shingle
  - **Then:** $\Pr[\pi(y) = \min(\pi(X))] = 1/|X|$
    - It is equally likely that any $y \in X$ is mapped to the *min* element

  - Let y be s.t. $\pi(y) = \min(\pi(C_1 \cup C_2))$
  - **Then either:** $\pi(y) = \min(\pi(C_1))$ if $y \in C_1$, **or**

    $\pi(y) = \min(\pi(C_2))$ if $y \in C_2$

One of the two cols had to have 1 at position ***y***

  - So the prob. that **both** are true is the prob. $y \in C_1 \cap C_2$
  - $\Pr[\min(\pi(C_1))=\min(\pi(C_2))]=|C_1 \cap C_2|/|C_1 \cup C_2| = sim(C_1, C_2)$

0	0
0	0
1	**1**
0	0
0	1
1	0

The Min-Hash Property (Take 2: simpler proof)

- **Choose a random permutation π**
- **<u>Claim:</u>** **$\Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$**
- **Why?**
  - Given a set X, the probability that any one element is the min-hash under π is $1/|X|$ ← (0)
    - It is equally likely that any ***y*** $\in$ ***X*** is mapped to the ***min*** element
  - Given a set X, the probability that one of any **k** elements is the min-hash under π is **k**$/|X|$ ← (1)
  - For $C_1 \cup C_2$, the probability that any element is the min-hash under π is $1/|C_1 \cup C_2|$ (from 0) ← (2)
  - For any C_1 and C_2, the probability of choosing the same min-hash under π is $|C_1 \cap C_2|/|C_1 \cup C_2|$ ← from (1) and (2)

Similarity for Signatures

- We know: $\Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
- Now generalize to multiple hash functions
- **The *similarity of two signatures* is the fraction of the hash functions in which they agree**
- **Note:** Because of the Min-Hash property, the similarity of columns is the same as the expected similarity of their signatures

Min-Hashing Example

Permutation π

2	4	3
3	2	4
7	1	7
6	3	2
1	6	6
5	7	1
4	5	5

Input matrix (Shingles x Documents)

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*

2	1	2	1
2	1	4	1
1	2	1	2

Similarities:

	1-3	2-4	1-2	3-4
Col/Col	0.75	0.75	0	0
Sig/Sig	0.67	1.00	0	0

Min-Hash Signatures

- **Pick K=100 random permutations of the rows**
- Think of ***sig*(C)** as a column vector
- ***sig*(C)[i] =** according to the *i*-th permutation, the index of the first row that has a 1 in column C

$$sig(C)[i] = \min(\pi_i(C))$$

- **Note:** The sketch (signature) of document C is small **~100 bytes!**
- **We achieved our goal! We "compressed" long bit vectors into short signatures**

Implementation Trick

- **Permuting rows even once is prohibitive**
- **Approximate Linear Permutation Hashing**
- Pick K independent hash functions (use a, b below)
  - Apply the idea on ***each column (document)*** for each hash function and get minhash signature

How to pick a random hash function h(x)?

Universal hashing:

$h_{a,b}(x)=((a\cdot x+b) \bmod p) \bmod N$
where:
a,b … random integers
p … prime number (p > N)

Summary: 3 Steps

- **Shingling:** Convert documents to sets
  - We used hashing to assign each shingle an ID
- **Min-Hashing:** Convert large sets to short signatures, while preserving similarity
  - We used **similarity preserving hashing** to generate signatures with property $\Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
  - We used hashing to get around generating random permutations

Backup slides

Outline: Finding Similar Columns

- **So far:**
  - Documents → Sets of shingles
  - Represent sets as boolean vectors in a matrix
- **Next goal: Find similar columns while computing small signatures**
  - **Similarity of columns == similarity of signatures**

Outline: Finding Similar Columns

- **Next Goal: Find similar columns, Small signatures**

- **Naïve approach:**
  - **1) Signatures of columns:** small summaries of columns
  - **2) Examine pairs of signatures** to find similar columns
    - **Essential:** Similarities of signatures and columns are related
  - **3) Optional:** Check that columns with similar signatures are really similar

- **Warnings:**
  - Comparing all pairs may take too much time: **Job for LSH**
    - These methods can produce false negatives, and even false positives (if the optional check is not made)

Hashing Columns (Signatures) : LSH principle

- **Key idea:** "hash" each column **C** to a small ***signature*** ***h(C)***, such that:
  - (1) ***h(C)*** is small enough that the signature fits in RAM
  - (2) $sim(C_1, C_2)$ is the same as the "similarity" of signatures $h(C_1)$ and $h(C_2)$

Hashing Columns (Signatures) : LSH principle

- **Key idea:** "hash" each column **C** to a small ***signature*** ***h(C)***, such that:
  - (1) ***h(C)*** is small enough that the signature fits in RAM
  - (2) ***sim***(C_1, C_2) is the same as the "similarity" of signatures $h(C_1)$ and $h(C_2)$

- **Goal: Find a hash function *h(·)* such that:**
  - If ***sim***(C_1,C_2) is high, then with high prob. $h(C_1) = h(C_2)$
  - If ***sim***(C_1,C_2) is low, then with high prob. $h(C_1) \neq h(C_2)$

- **Hash docs into buckets. Expect that "most" pairs of near duplicate docs hash into the same bucket!**

Min-Hashing

- **Goal: Find a hash function $h(\cdot)$ such that:**
  - if $sim(C_1,C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$
  - if $sim(C_1,C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$
- **Clearly, the hash function depends on the similarity metric:**
  - Not all similarity metrics have a suitable hash function
- **There is a suitable hash function for the Jaccard similarity:** It is called **Min-Hashing**

Min-Hashing

- Imagine the rows of the boolean matrix permuted under **random permutation π**
- Define a **"hash" function $h_\pi(C)$** = the index of the **first** (in the permuted order π) row in which column **C** has value **1**:

$$h_\pi(C) = \min_\pi \pi(C)$$

- Use several (e.g., 100) independent hash functions (that is, permutations) to create a signature of a column

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Zoo example (shingle size k=1)

Universe $\longrightarrow$ { dog, cat, lion, tiger, mouse}

π_1 $\longrightarrow$ [cat, mouse, lion, dog, tiger]

π_2 $\longrightarrow$ [lion, cat, mouse, dog, tiger]

A = { mouse, lion }

$mh_1(A) = \min(\pi_1\{mouse, lion\}) = mouse$

$mh_2(A) = \min(\pi_2\{mouse, lion\}) = lion$

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Key Fact

For two sets A, B, and a min-hash function $mh_i()$:

$$Pr[mh_i(A) = mh_i(B)] = Sim(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Unbiased estimator for Sim using *K* hashes (notation policy – this is a different K from size of shingle)

$$\hat{Sim}(A, B) = \frac{1}{k} \sum_{i=1:k} I[mh_i(A) = mh_i(B)]$$

Min-Hashing Example

Note: Another (equivalent) way is to store row indexes or raw shingles (e.g. mouse, lion):

1	5	1	5
2	3	1	3
6	4	6	4

2nd element of the permutation is the first to map to a 1

Permutation π

2	4	3
3	2	4
7	1	7
6	3	2
1	6	6
5	7	1
4	5	5

Input matrix (Shingles x Documents)

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*

2	1	2	1
2	1	4	1
1	2	1	2

4th element of the permutation is the first to map to a 1

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

The Min-Hash Property

0	0
0	0
1	**1**
0	0
0	1
1	0

- **Choose a random permutation π**
- **Claim:** $\Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
- **Why?**
  - Let **X** be a doc (set of shingles), $y \in X$ is a shingle
  - **Then:** $\Pr[\pi(y) = \min(\pi(X))] = 1/|X|$
    - It is equally likely that any $y \in X$ is mapped to the ***min*** element
  - Let y be s.t. $\pi(y) = \min(\pi(C_1 \cup C_2))$
  - **Then either:** $\pi(y) = \min(\pi(C_1))$ if $y \in C_1$, **or**

    $\pi(y) = \min(\pi(C_2))$ if $y \in C_2$
  - So the prob. that **both** are true is the prob. $y \in C_1 \cap C_2$
  - $\Pr[\min(\pi(C_1))=\min(\pi(C_2))]=|C_1 \cap C_2|/|C_1 \cup C_2| = sim(C_1, C_2)$

One of the two cols had to have 1 at position ***y***

The Min-Hash Property (Take 2: simpler proof)

- **Choose a random permutation π**
- **<u>Claim:</u>** **$Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$**
- **Why?**
  - Given a set X, the probability that any one element is the min-hash under π is $1/|X|$ ← (0)
    - It is equally likely that any $y \in X$ is mapped to the ***min*** element
  - Given a set X, the probability that one of any **k** elements is the min-hash under π is $\mathbf{k}/|X|$ ← (1)
  - For $C_1 \cup C_2$, the probability that any element is the min-hash under π is $1/|C_1 \cup C_2|$ (from 0) ← (2)
  - For any C_1 and C_2, the probability of choosing the same min-hash under π is $|C_1 \cap C_2|/|C_1 \cup C_2|$ ← from (1) and (2)

Similarity for Signatures

- We know: $\Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
- Now generalize to multiple hash functions
- **The *similarity of two signatures* is the fraction of the hash functions in which they agree**
- **Note:** Because of the Min-Hash property, the similarity of columns is the same as the expected similarity of their signatures

Min-Hashing Example

Permutation π

2	4	3
3	2	4
7	1	7
6	3	2
1	6	6
5	7	1
4	5	5

Input matrix (Shingles x Documents)

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*

2	1	2	1
2	1	4	1
1	2	1	2

Similarities:

	1-3	2-4	1-2	3-4
Col/Col	0.75	0.75	0	0
Sig/Sig	0.67	1.00	0	0

Min-Hash Signatures

- **Pick K=100 random permutations of the rows**
- Think of ***sig*(C)** as a column vector
- ***sig*(C)[i] =** according to the *i*-th permutation, the index of the first row that has a 1 in column C

$$sig(C)[i] = \min(\pi_i(C))$$

- **Note:** The sketch (signature) of document C is small **~100 bytes!**

- **We achieved our goal! We "compressed" long bit vectors into short signatures**

Implementation Trick

- Permuting rows even once is prohibitive
- Approximate Linear Permutation Hashing
- Pick K independent hash functions (use a, b below)
  - Apply the idea on ***each column (document)*** for each hash function and get minhash signature

How to pick a random hash function h(x)?

Universal hashing:

$h_{a,b}(x)=((a\cdot x+b) \bmod p) \bmod N$

where:

a,b … random integers

p … prime number (p > N)

Summary: 3 Steps

- **Shingling:** Convert documents to sets
  - We used hashing to assign each shingle an ID
- **Min-Hashing:** Convert large sets to short signatures, while preserving similarity
  - We used **similarity preserving hashing** to generate signatures with property $\mathbf{Pr}[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
  - We used hashing to get around generating random permutations

Backup slides