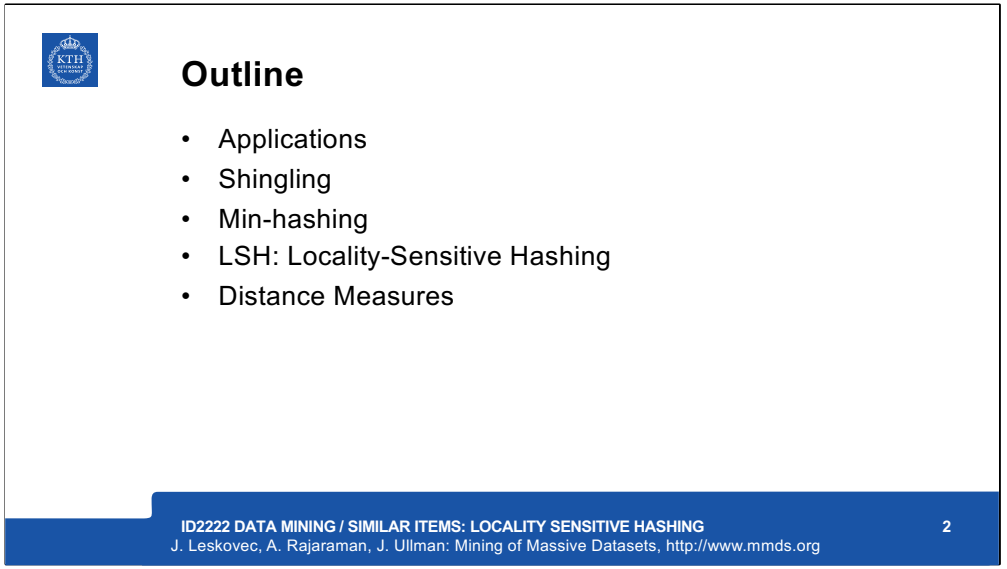
ID2222: Data Mining

KTH ROYAL INSTITUTE OF TECHNOLOGY

Finding Similar Items: Locality Sensitive Hashing

Vladimir Vlassov

Acknowledgement: Some of the slides are adopted from the slide provided at http://www.mmds.org for the book "Mining of Massive Datasets" by Jure Leskovec, Anand Rajaraman, Jeff Ullman, Stanford University



The textbook takes an algorithmic view: data mining is about applying algorithms to data rather than using data to "train" a machine-learning model.

A Fundamental Data-Mining Problem

Examine data for "similar" items

based on some similarity or distance measure, such as Jaccard distance/similarity, Euclidean distances, cosine distance, edit distance, Hamming distance

A Fundamental Data-Mining Problem (cont.)

Find "similar" items.

Given

- high-dimensional data points $\boldsymbol{x_1}$, $\boldsymbol{x_2}$, …;
- a distance function $\boldsymbol{d(x_i, x_j)}$ that quantifies the "distance" between $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$;
- a distance threshold $\boldsymbol{s}$ – the max size of near neighbors

Find all pairs of data points $\boldsymbol{(x_i, x_j)}$ that are within the distance threshold: $\boldsymbol{d(x_i, x_j) \leq s}$, i.e., at most $\boldsymbol{s}$ apart from each other

- Naïve solutions is O(N^2)
- Some optimization allows O(N)

Jaccard Similarity

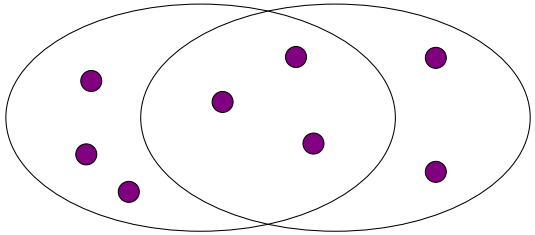
Many data-mining problems can be expressed as finding ***"similar" sets.***

The Jaccard similarity of two sets, C1 and C2, is the fraction of common items, i.e., the fraction of their intersection – size of their intersection divided by the size of their union:

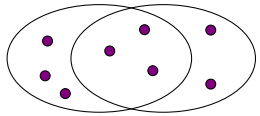
$$sim(C1, C2) = |C1 \cap C2| / |C1 \cup C2|$$

Jaccard distance: $d(C1, C2) = 1 - |C1 \cap C2| / |C1 \cup C2|$

Jaccard Similarity



3 in intersection
8 in union
Jaccard similarity= 3/8
Jaccard distance = 5/8



3 in intersection
8 in union
Jaccard similarity= 3/8
Jaccard distance = 5/8

Similar Documents

Given a body of documents, e.g., the Web, find pairs of ***textually similar documents*** with a lot of text in common (***near duplicate pairs***)

Examples:

- *Mirror sites*, or approximate mirrors
  - quite similar but are rarely identical
  - Don't want to show both in a search result.
- *Plagiarism*, including large quotations.
- *Similar news articles* at many news sites.
  - Articles from the same source, e.g., the Associated Press
  - Cluster articles by "same story."

Collaborative filtering

Find pairs of ***"similar" customers*** with similar tastes – having similar "baskets" with rather high Jaccard similarity (≥ 20%)

Dual: Find pairs of ***"similar" products*** bought by similar sets of customers

Combine similarity-finding with clustering to group mutually-similar products.

- A more powerful notion of customer similarity by asking whether they made purchases within many of the same groups.

Example: NetFlix users with similar tastes in movies for recommendation systems. (*Dual*: movies with similar sets of fans).

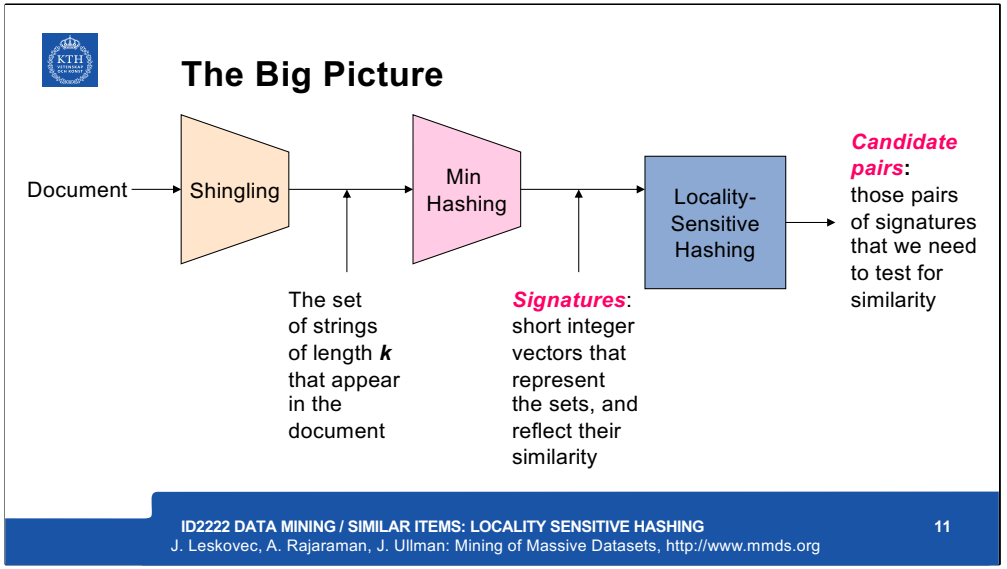
Finding Similar Documents

Problems:

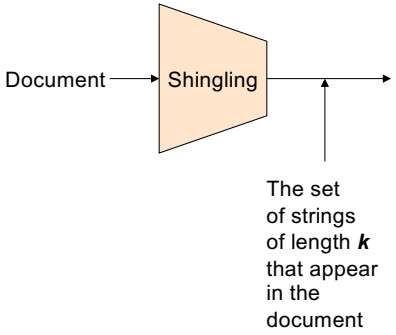
- Common text, not a common topic.
- Many small pieces of one document may appear out of order in another.
  - Special cases are straightforward, e.g., identical documents or one document contained in another.
- *Too many documents* to compare all pairs or *too many pairs*
- Documents are so large or so many that they *cannot fit in the main memory*

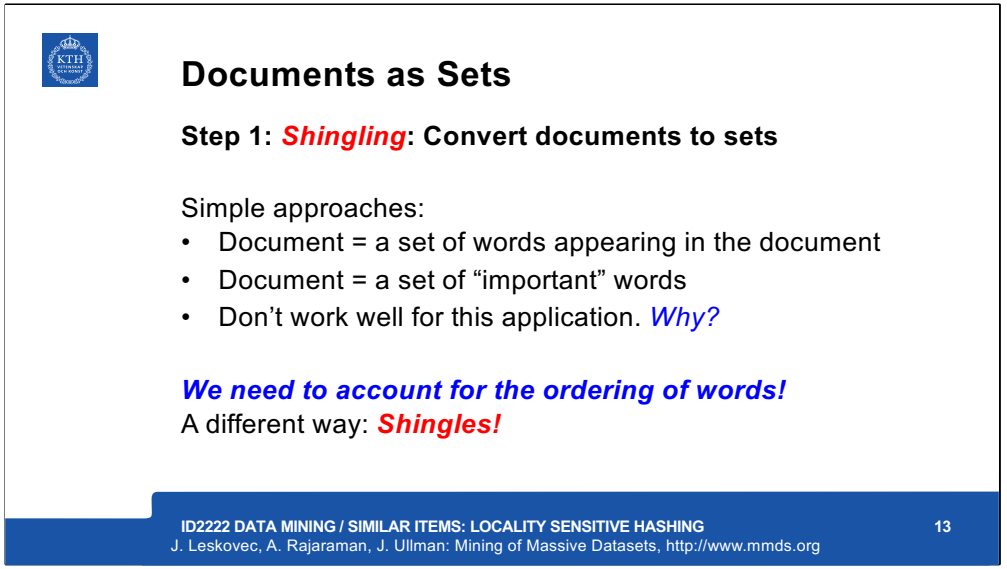
3 Essential Techniques for Similar Documents

1. ***Shingling***: Convert documents to sets.
2. ***Minhashing***: Convert large sets to short signatures while preserving similarity.
3. ***Locality-sensitive hashing***: Focus on pairs of signatures likely to be from similar documents
   - Candidate pairs

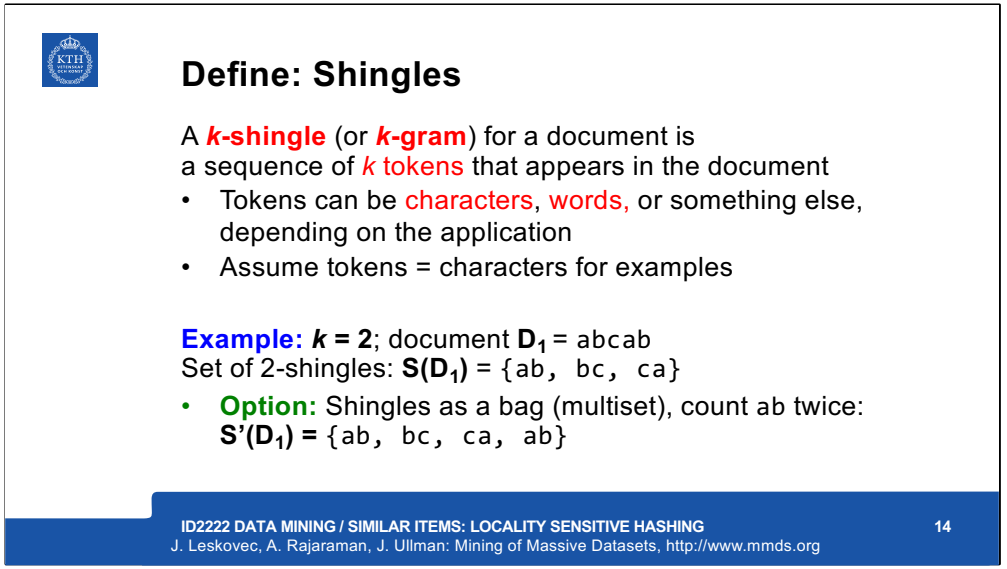

The Big Picture
Document
Shingling
Min Hashing
Locality-Sensitive Hashing
Candidate pairs: those pairs of signatures that we need to test for similarity
The set of strings of length k that appear in the document
Signatures: short integer vectors that represent the sets, and reflect their similarity
ID2222 DATA MINING / SIMILAR ITEMS: LOCALITY SENSITIVE HASHING
11
J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Step 1: *Shingling*: Convert documents to sets

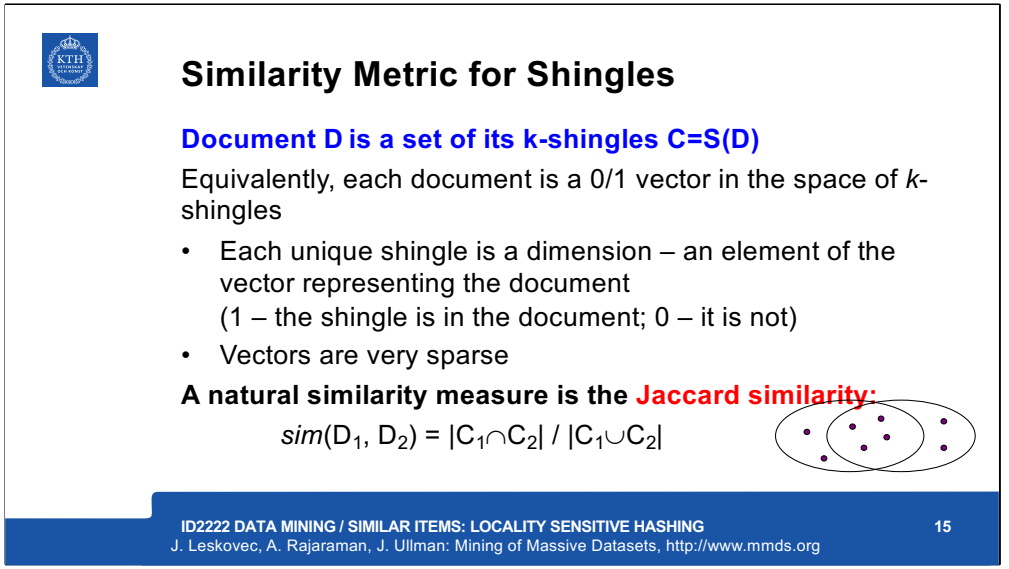


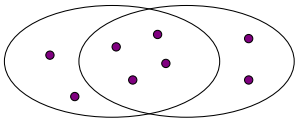


Important words; tf–idf, short for **term frequency-inverse document frequency**, is a numerical statistic intended to reflect how important a word is to a document in a collection or corpus. **Term frequency tf(t,d),** the simplest choice is to use the raw count of a term in a document, i.e., **the number of times that term t occurs in document d**. **The inverse document frequency idf (t, D)** measures how much information the word provides, i.e., if it's common or rare across all documents. It is the logarithmically scaled inverse fraction of the documents that contain the word (obtained **by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient**): **Then tf–idf is calculated as a product tfidf(t,d, D)=tf (t,d) * idf (t, D)**



Several options regarding white spaces: 1. Replace any sequence of one or more white spaces with a single blank; 2. Remove all white spaces.


Similarity Metric for Shingles
Document D is a set of its k-shingles C=S(D)
Equivalently, each document is a 0/1 vector in the space of k-shingles
• Each unique shingle is a dimension – an element of the vector representing the document
(1 – the shingle is in the document; 0 – it is not)
• Vectors are very sparse
A natural similarity measure is the Jaccard similarity:
sim(D1, D2) = |C1∩C2| / |C1∪C2|
ID2222 DATA MINING / SIMILAR ITEMS: LOCALITY SENSITIVE HASHING
15
J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org



Working Assumption and Observation

Documents with lots of shingles in common have similar text, even if the text appears in a different order.

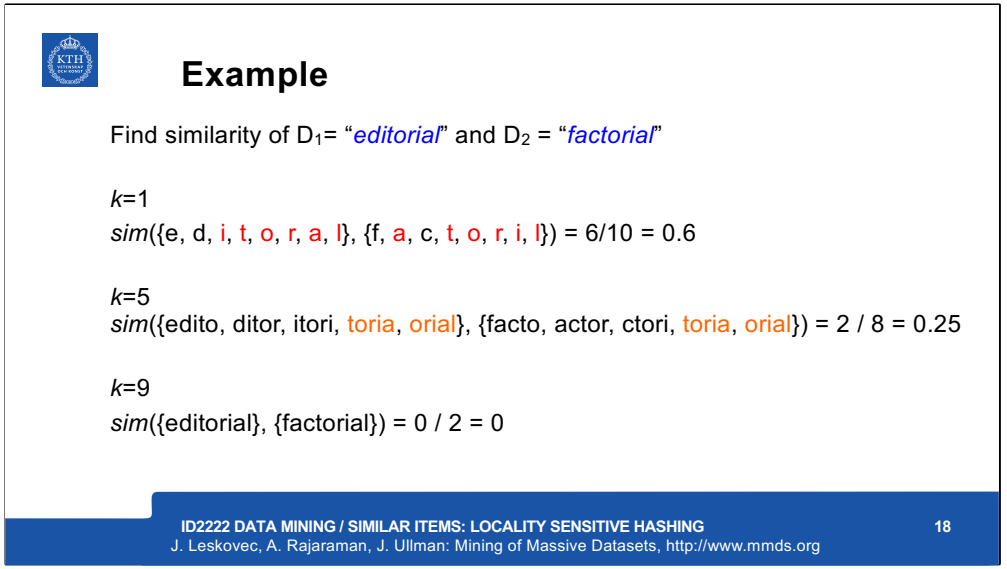
- Changing a word only affects *k*-shingles within distance *k* from the word.
- Reordering paragraphs only affects the 2*k* shingles that cross paragraph boundaries.
- **Example**: *k* = 3, "The dog *which* chased the cat" versus "The dog *that* chased the cat"
  - Only 3-shingles replaced are g_*w*, _*wh*, *whi*, *hic*, *ich*, *ch*_, *h*_c.

Shingle Size

Caveat: ***k*** should be large enough, or most documents will have most shingles.

In other words, ***k*** should be large enough that the probability of any given shingle appearing in any given document is low.

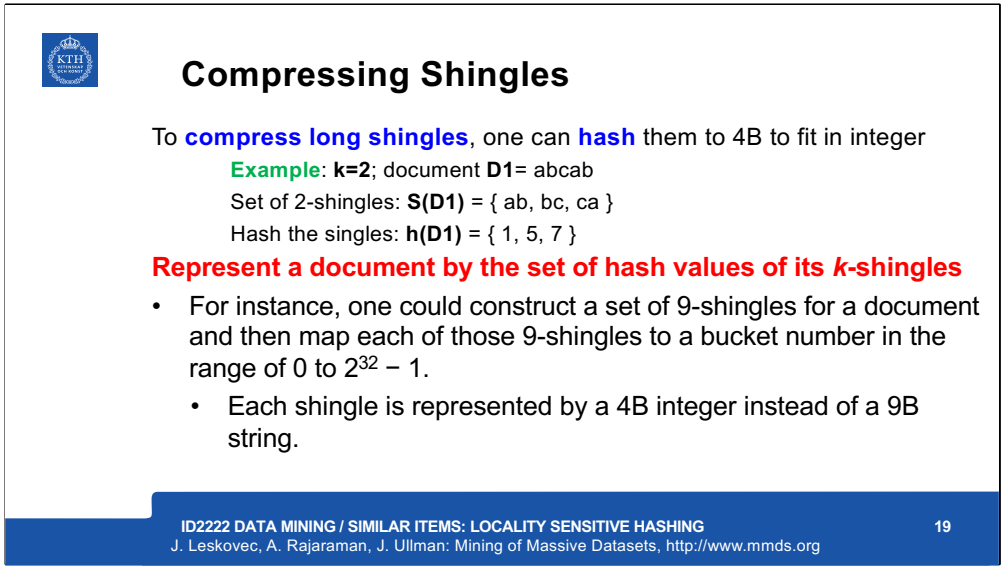
- ***k*** = 5 is OK for short documents, e.g., emails
- ***k*** = 9, 10 is better for long documents



k = 2: SIM({ed, di, it, **to**, **or**, **ri**, **ia**, **al**}, {fa, ac, ct, **to**, **or**, **ri**, **ia**, **al**}) = 5/11 = 0.46

k = 3: SIM({edi, dit, ito, **tor**, **ori**, **ria**, **ial**}, {fac, act, cto, **tor**, **ori**, **ria**, **ial**}) = 4/10 = 0.4

k = 4: SIM({edit, dito, itor, **tori**, **oria**, **rial**}, {fact, acto, ctor, **tori**, **oria**, **rial**}) = 3 / 9 = 0.33



Notice that we can differentiate documents better if we use 9-shingles and hash them down to four bytes than to use 4-shingles, even though the space used to represent a shingle is the same. The reason was touched upon in Section 3.2.2. If we use 4-shingles, most sequences of four bytes are unlikely or impossible to find in typical documents. Thus, the effective number of different shingles is much less than 2^32−1. If, as in Section 3.2.2, we assume only 20 characters are frequent in English text, then the number of different 4-shingles likely to occur is only (20)^4 = 160,000. However, if we use 9-shingles, there are many more than 2^32 likely shingles. When we hash them down to four bytes, we can expect almost any sequence of four bytes to be possible, as was discussed in Section 1.3.2.

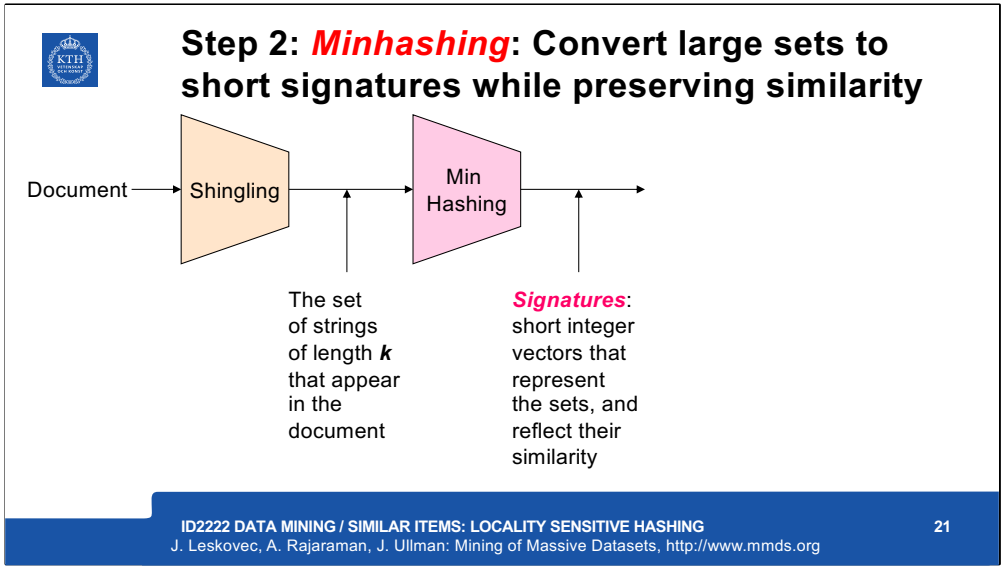
Motivation for Minhash/LSH

Suppose we need to find near-duplicate documents among $N = 1$ million documents.

Naïvely, we would have to compute **pairwise Jaccard similarities** for **every pair of docs – complexity O(N^2)**

- $N(N-1)/2$ **≈ 5*10^{11}** comparisons
- At 10^5 sec/day and 10^6 comparisons/sec, it would take **5 days.**

For $N = 10$ million, it takes more than a year…


Step 2: Minhashing: Convert large sets to short signatures while preserving similarity
Document
Shingling
Min Hashing
The set of strings of length k that appear in the document
Signatures: short integer vectors that represent the sets, and reflect their similarity
ID2222 DATA MINING / SIMILAR ITEMS: LOCALITY SENSITIVE HASHING
21
J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Basic Data Model: Sets

Many similarity problems can be formalized as ***finding subsets of some universal set that have significant intersection***.

Examples include:

1. Documents represented by their sets of shingles (or hashes of those shingles).
2. Similar customers or products.

Remind: Jaccard Similarity of Sets

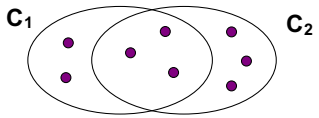
The ***Jaccard similarity*** of the two sets is the size of their intersection divided by the size of their union.

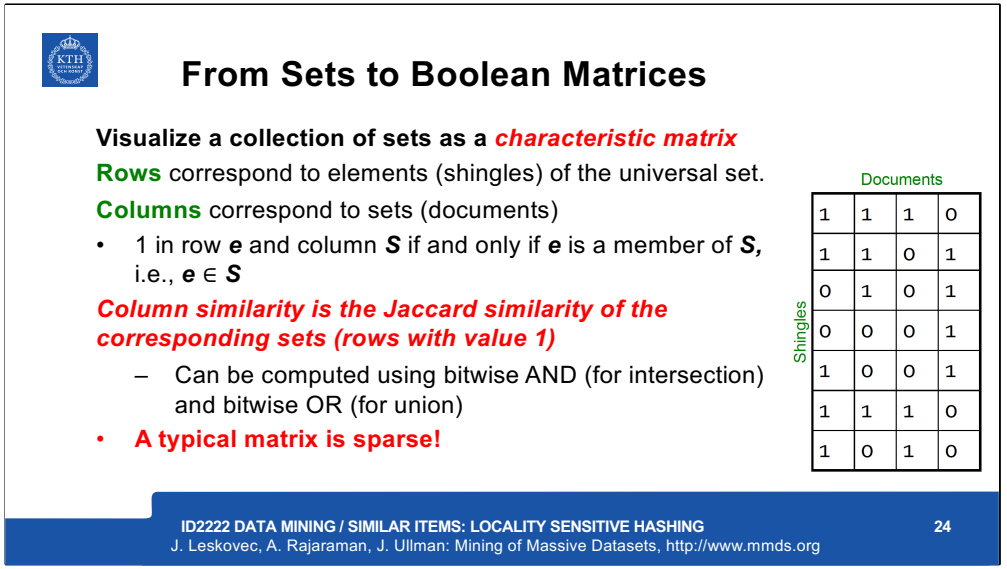
$$Sim(C_1, C_2) = | C_1 \cap C_2 | / | C_1 \cup C_2 |$$

***Jaccard* distance: $d(C_1, C_2) = 1 - Sim(C_1, C_2)$**

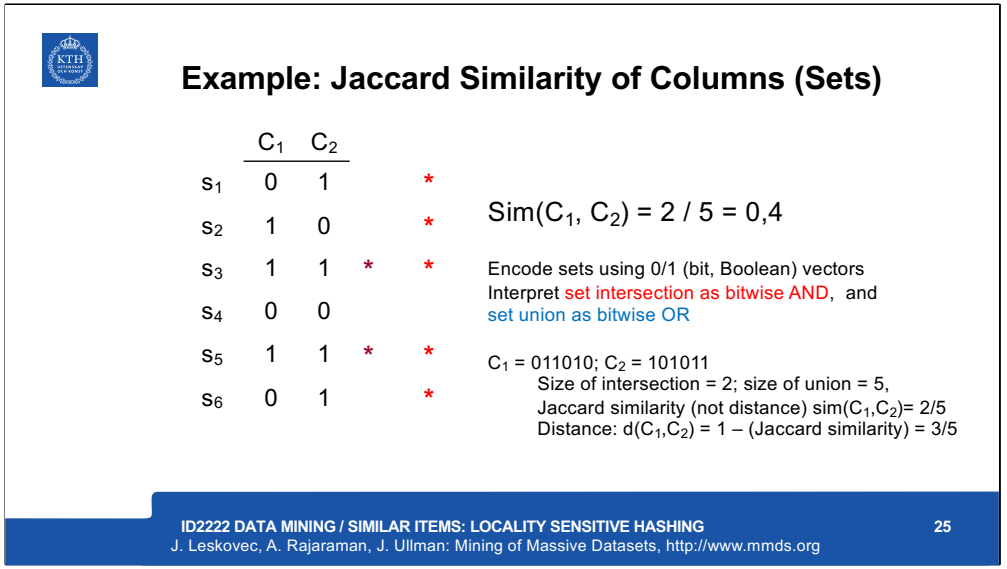
Example

- Size of intersection = 3; size of union = 8,
- Jaccard similarity: $sim(C_1,C_2) = 3/8$
- Distance: $d(C_1,C_2) = 1 – sim(C_1,C_2) = 5/8$





Remark: A large sparse matrix is better represented by the list of places with non-zero values.



Remark: A large sparse matrix is better represented by the list of places with non-zero values.

Four Types of Rows

Given columns C_1 and C_2, rows may be classified as:

C_1	C_2		
1	1	*a*	*1 in both columns*
1	0	*b*	*columns are different*
0	1	*c*	
0	0	*d*	*0 in both columns*

Denote, *A* = the number of rows of type *a*;
B = the number of rows of type *b*, etc.

Sim $(C_1, C_2) = A/(A+B+C)$.

Outline: Finding Similar Columns

So far:

- Documents → Sets of shingles
- Represent sets as Boolean column-vectors in a matrix

Next goal: Find similar columns while computing *small signatures*

- **Similarity of columns ≈ similarity of signatures**

Outline: Finding Similar Columns (cont)

1. Compute ***signatures of columns*** = small summaries of columns.
2. Examine ***pairs of signatures*** to find similar signatures.
   - 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)

MinHashing

Let ***h*** be a hash function that maps the members of a set ***S*** to distinct integers.

MinHash $h_{min}(S)$ is the member $e \in S$ with the minimum value of $h(e)$

If for two sets S_1 and S_2, $h_{min}(S_1) = h_{min}(S_2) = e$, then $e \in S_1 \cap S_2$

The probability of this to be true is $|S_1 \cap S_2|/|S_1 \cup S_2|$, i.e., the ***Jaccard similarity***:

$$Pr[\, h_{min}(S_1) = h_{min}(S_2) \,] = Sim(S_1, S_2)$$

MinHash Signature of a Set

Key idea:

- Take ***k*** (e.g., k = 100) independent hash functions, e.g., $h(x) = (ax + b) \% c$
- Apply the functions to the elements (the row numbers with value 1) to compute a vector of ***k*** minHash values for a set S.

The resulting column-vector of k minHash values is a signature of the set S

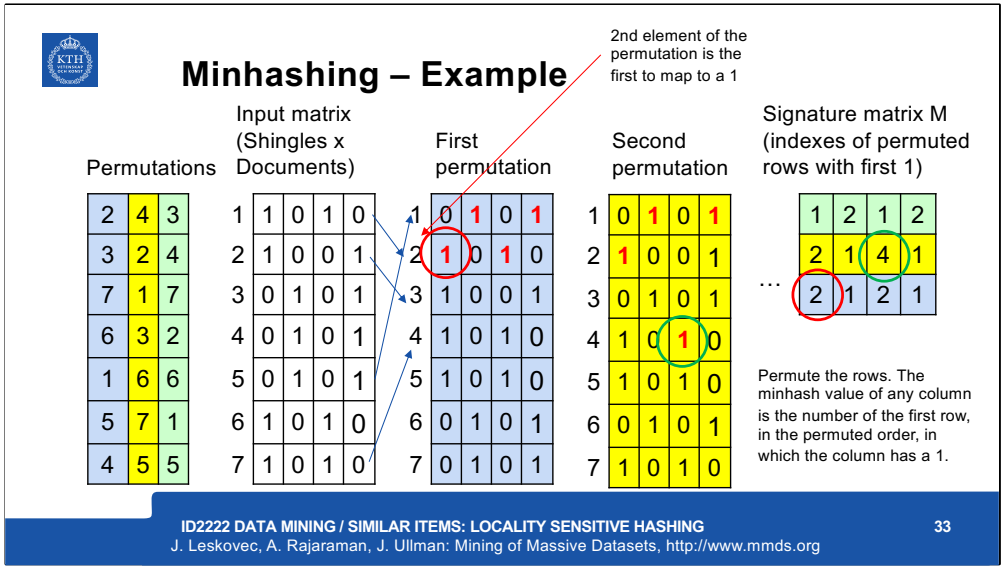
MinHashing using Permutations

- Permute the rows.
- Define the ***minhash function*** for this permutation, $h(C)$ = the number of the first (in the permuted order) row in which column C has 1.
- Apply, to all columns, several (e.g., 100) randomly chosen permutations to create a ***signature*** for each column.
- Result is a ***signature matrix***: columns = sets (documents), rows = minhash values, in order for that column.

Using several permutations is equivalent to using several hash functions of the type h(x) = (ax + b)%c

Signature Matrix

Thus, we can form from characteristic matrix **M** a ***signature matrix***, in which the *i*-th column of **M** is replaced by the minhash signature for (the set of) the *i*-th column.



In the above example's first (blue) permutation, the 5-th row of the input matrix becomes 1-st in the permuted order; the 1-st input row becomes the 2-nd in the permuted order; the 2-nd input row because the 3-rd in the permuted order, etc.

Similarity of Signatures

The ***similarity of the two signatures*** is the fraction of the minHash functions in which they agree

$$\Pr[\, h_{min}(S_1) = h_{min}(S_2) \,]$$

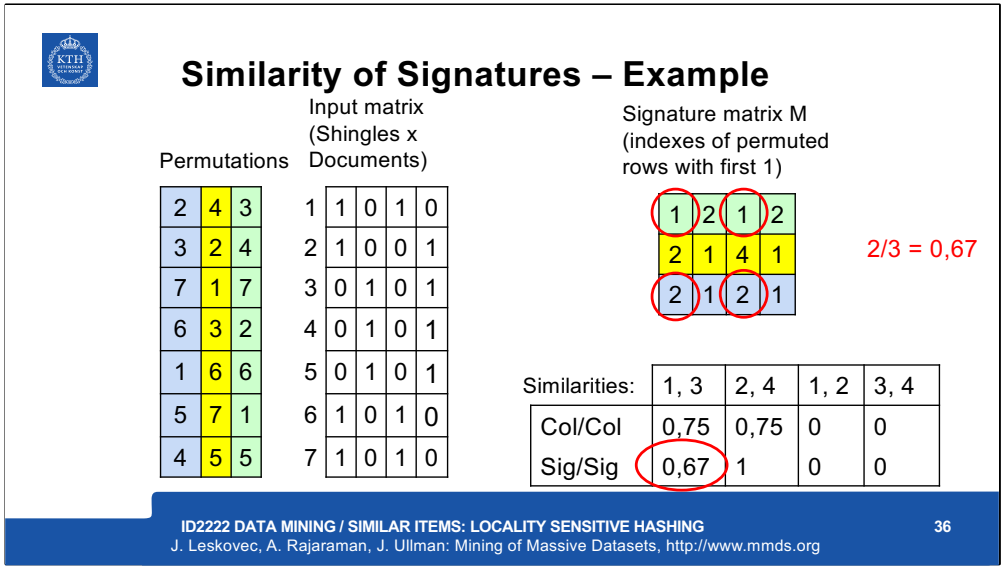
- i.e., the number of rows in the signature matrix with the same values in S_1 and S_2 columns divided by the total number of rows in the signature (k).

For two sets, S_1 and S_2, ***the similarity of two signatures is an estimate of the Jaccard similarity*** of the two sets.

$$\Pr[\, h_{min}(S_1) = h_{min}(S_2) \,] = Sim(S_1, S_2)$$

Similarity of Signatures (cont)

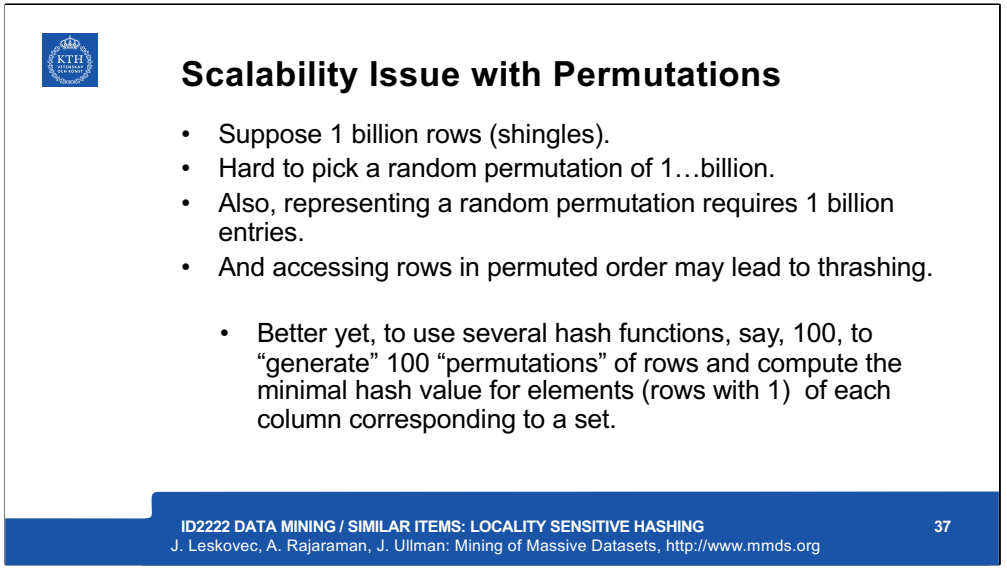
- The ***similarity of signatures*** is the fraction of the minhash functions (rows) in which they agree.
- Thus, the expected similarity of two signatures equals the Jaccard similarity of the columns or sets that the signatures represent.
  - And the longer the signatures, the smaller will be the expected error.



In the above example:

The number of rows in which the 1-st signature agrees with the 3-rd signature is 2, and the total number of rows in the signature matrix is 3; therefore, the similarity of the 1-st and the 3-rd signatures is 2 / 3 = 0,67.

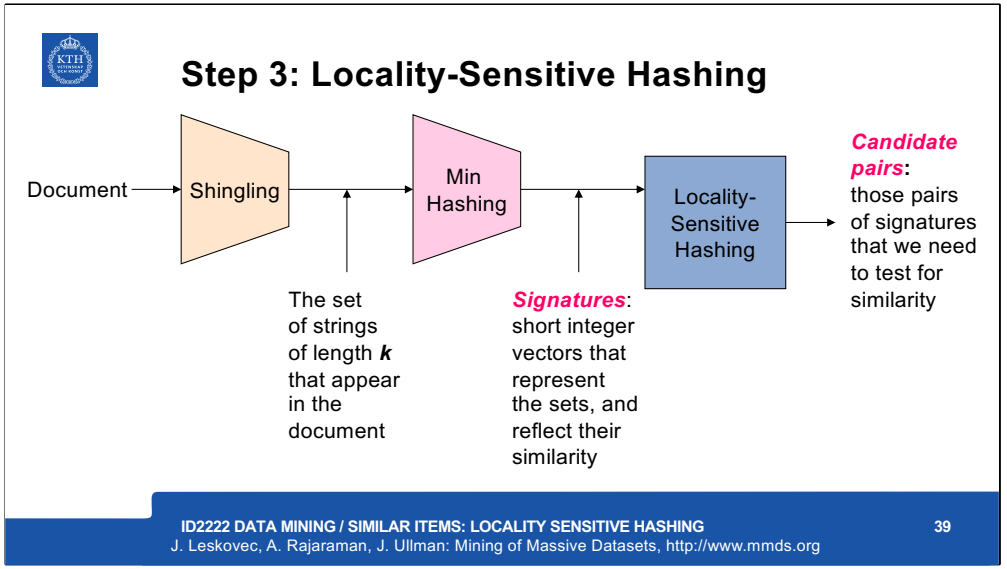
The size of the intersection of the 1-st shingle set with the 3-rd shingle set is 3 (the number of rows with 1 in both columns in the input matrix), whereas the union of the two shingle sets is 4 (the number of rows with 1 in at least one of the two columns); therefore, the Jaccard similarity of 1-st and 3-rd shingle sets (1-st and 3-rd columns) is 3 / 4 = 0,75.



4B integer: 2^32 is about 4,29 billion possible combinations

Implementation

```
for each row r do begin
  for each hash function h_i do
     compute h_i(r);
  for each column c
    if c has 1 in row r
      for each hash function h_i do
        if h_i(r) is smaller than M(i, c) then
             M(i, c) := h_i(r);
end;
```


Step 3: Locality-Sensitive Hashing
Document
Shingling
Min Hashing
Locality-Sensitive Hashing
Candidate pairs: those pairs of signatures that we need to test for similarity
The set of strings of length k that appear in the document
Signatures: short integer vectors that represent the sets, and reflect their similarity
ID2222 DATA MINING / SIMILAR ITEMS: LOCALITY SENSITIVE HASHING
39
J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Scalability Issue

Assume we have built a signature matrix.

Need to compare signatures (columns) in pairs for similarity – $O(N^2)$

Checking all pairs is hard

Example: 10^6 columns implies (10^6 by 2) $\frac{n!}{k!\,(n-k)!}$

=~ $5*10^{11}$ column-comparisons.

At 1 μs/comparison: 6 days.

LSH: Locality-Sensitive Hashing

- General idea: Generate from the collection of all elements (signatures in our example) a small list of ***candidate pairs***: pairs of elements whose similarity must be evaluated.

- For signature matrices: Hash columns to many buckets and make elements of the same bucket candidate pairs.

Candidate Generation from Minhash Signatures

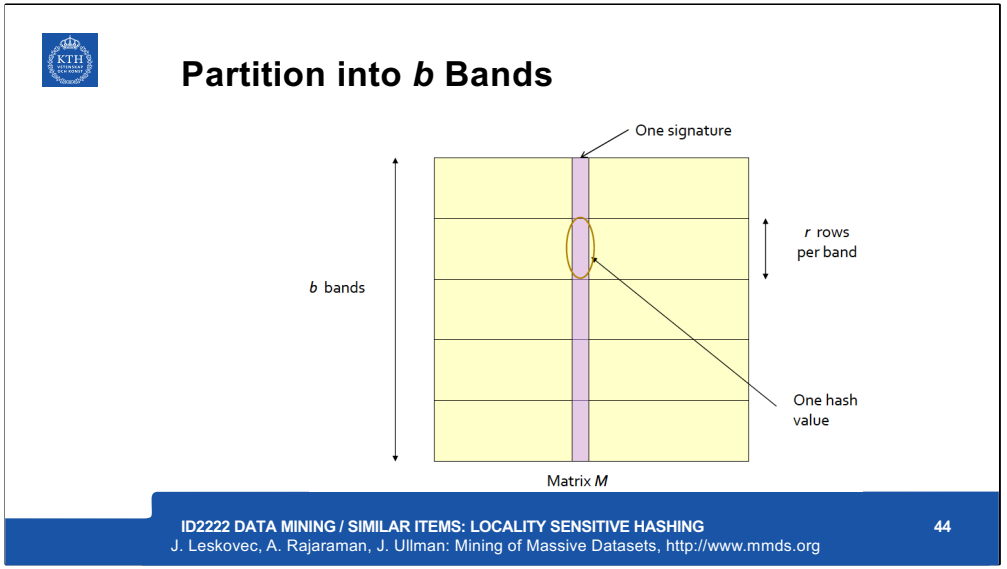
- Pick a similarity threshold s ($0 < s < 1$).
- We want a pair of columns c and d of the signature matrix M to be a ***candidate pair*** if and only if their signatures agree in at least fraction s of the rows.
  - I.e., $M(i, c) = M(i, d)$ for at least fraction s values of i.

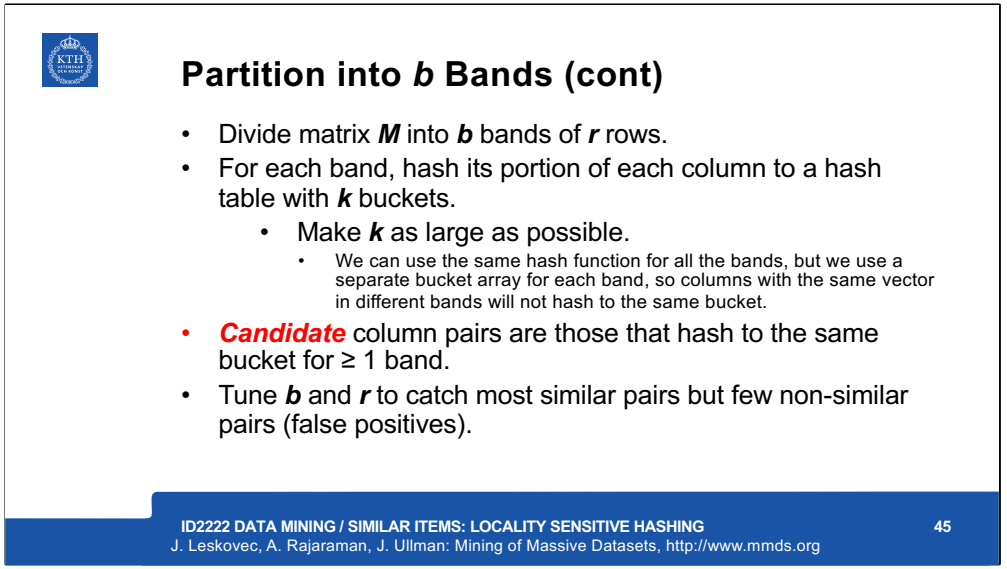
LSH for MinHash Signatures

Big idea: hash columns of signature matrix *M* several times.

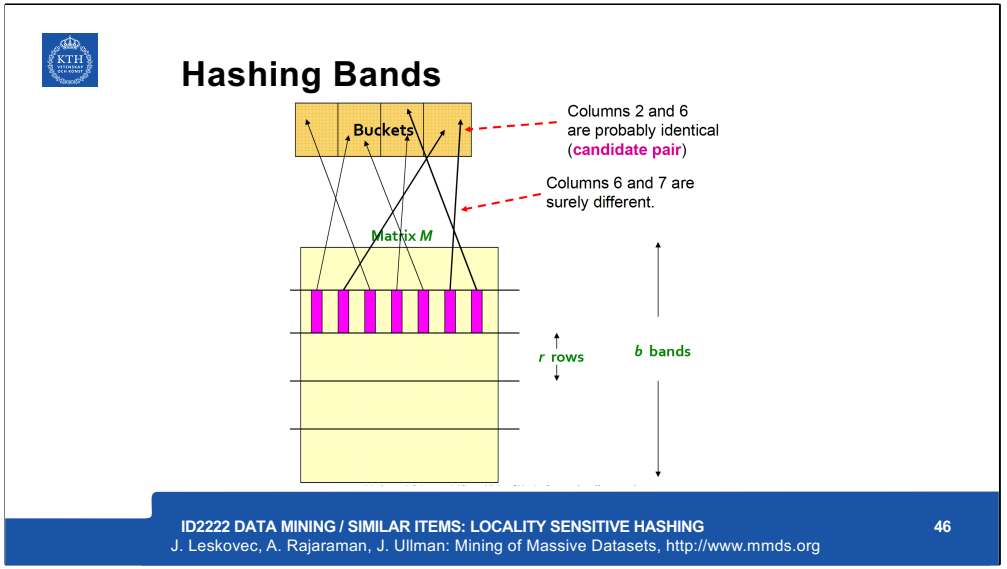
Arrange that (only) similar columns are likely to hash to the same bucket.

Candidate pairs are those that hash ***at least once*** to the ***same bucket***.


Partition into b Bands
One signature
b bands
r rows
per band
One hash
value
Matrix M
ID2222 DATA MINING / SIMILAR ITEMS: LOCALITY SENSITIVE HASHING
44
J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org



If we have minhash signatures for the items, an effective way to choose the hashings is to divide the signature matrix into b bands consisting of r rows each. There is a hash function for each band that takes vectors of r integers (the portion of one column within that band) and hashes them to many buckets. We can use the same hash function for all the bands, but we use a separate bucket array for each band, so columns with the same vector in different bands will not hash to the same bucket.

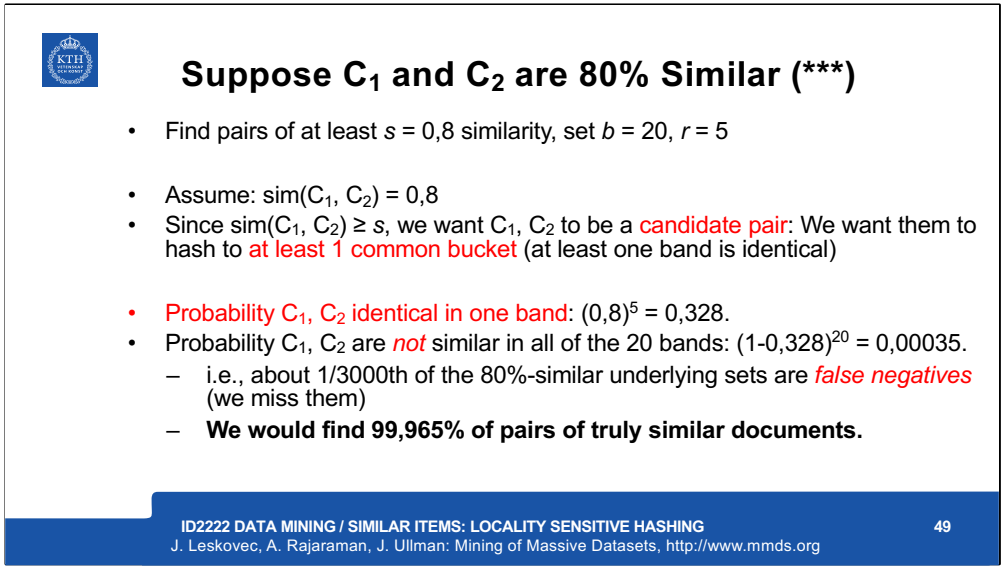

Hashing Bands
Buckets
Columns 2 and 6
are probably identical
(candidate pair)
Columns 6 and 7 are
surely different.
Matrix M
r rows
b bands
ID2222 DATA MINING / SIMILAR ITEMS: LOCALITY SENSITIVE HASHING
46
J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Simplifying Assumption

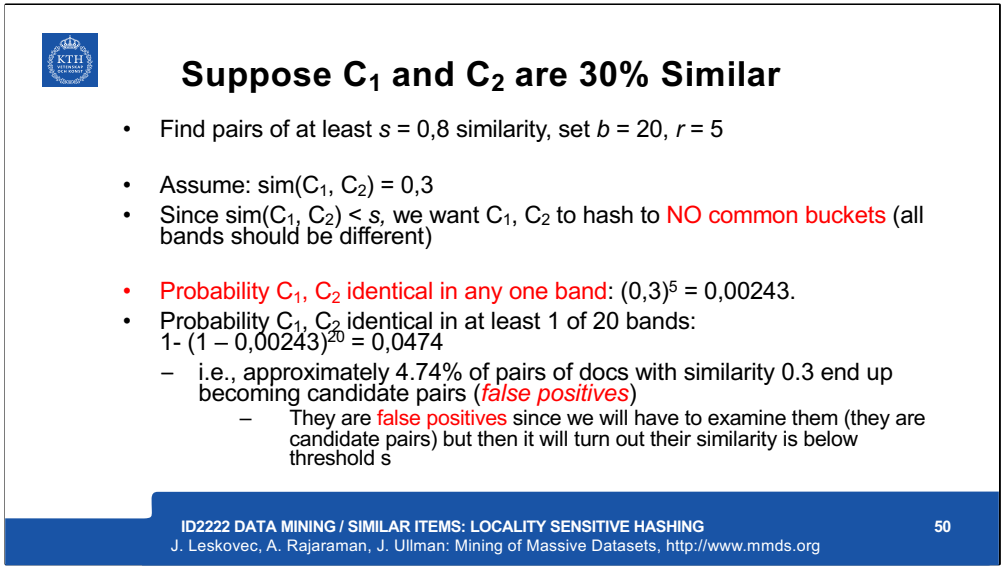
- There are enough buckets that columns are unlikely to hash to the same bucket unless they are ***identical*** in a particular band.
- ***Hereafter, we assume that "same bucket" means "identical in that band."***
  - The assumption is needed only to simplify the analysis, not for the correctness of the algorithm.

Example: Bands

- Suppose 100,000 columns.
- Signatures of 100 integers.
- Therefore, signatures take 40Mb.
  - They fit easily into the main memory.
- Want all 80%-similar pairs of documents.
- 5,000,000,000 pairs of signatures can take a while to compare.
- Choose 20 bands of 5 rows/band.



Probability of missing an 80%-similar candidate pair is very small


Suppose C_1 and C_2 are 30% Similar
• Find pairs of at least s = 0,8 similarity, set b = 20, r = 5
• Assume: sim(C_1, C_2) = 0,3
• Since sim(C_1, C_2) < s, we want C_1, C_2 to hash to NO common buckets (all bands should be different)
• Probability C_1, C_2 identical in any one band: $(0,3)^5$ = 0,00243.
• Probability C_1, C_2 identical in at least 1 of 20 bands: $1- (1 – 0,00243)^{20}$ = 0,0474
– i.e., approximately 4.74% of pairs of docs with similarity 0.3 end up becoming candidate pairs (*false positives*)
– They are false positives since we will have to examine them (they are candidate pairs) but then it will turn out their similarity is below threshold s
ID2222 DATA MINING / SIMILAR ITEMS: LOCALITY SENSITIVE HASHING
50
J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

If sim(C1

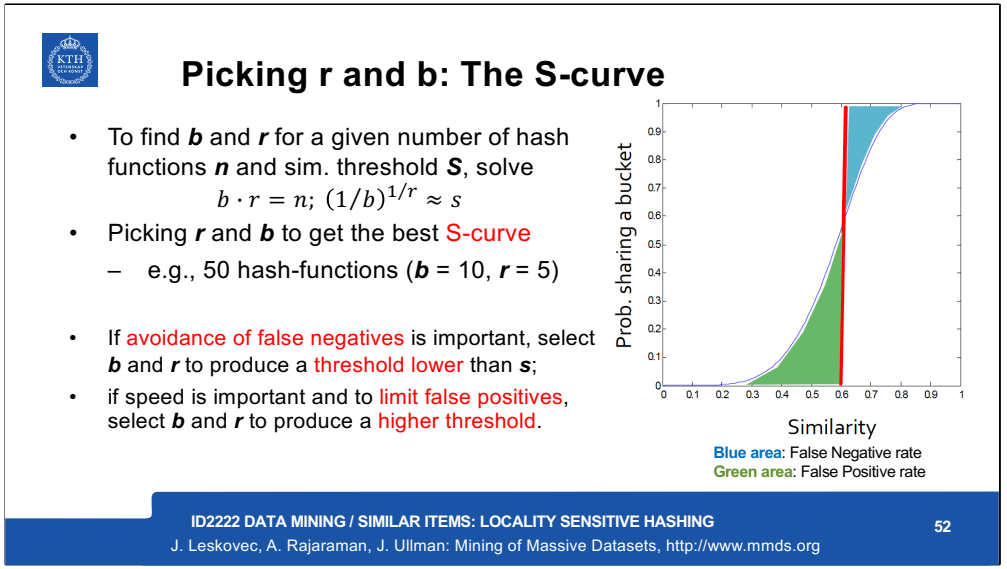
LSH Involves a Tradeoff

Pick:

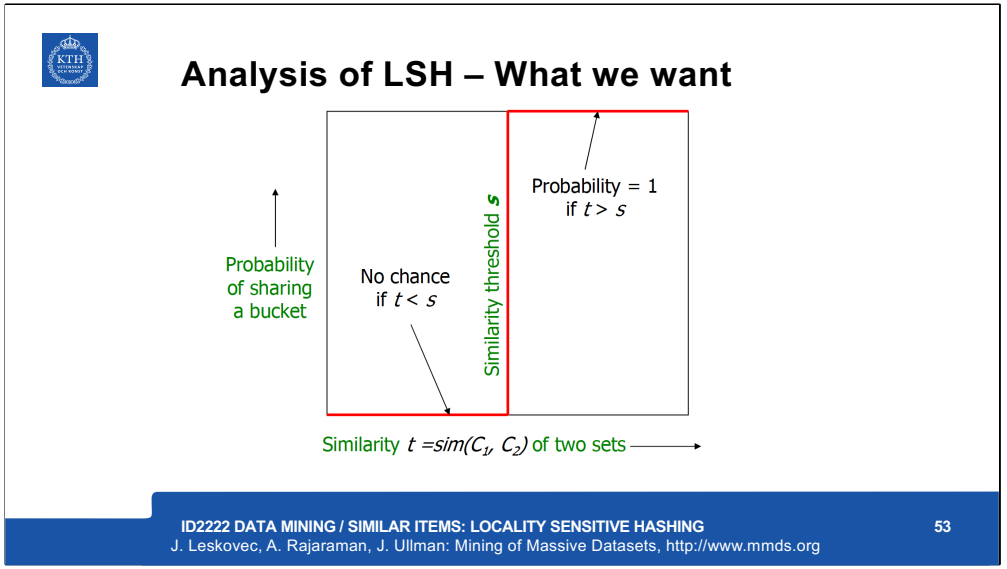
- The number of Min-Hashes (signature length)
- The number of bands ***b***, and
- The number of rows ***r*** per band

to balance false positives/negatives

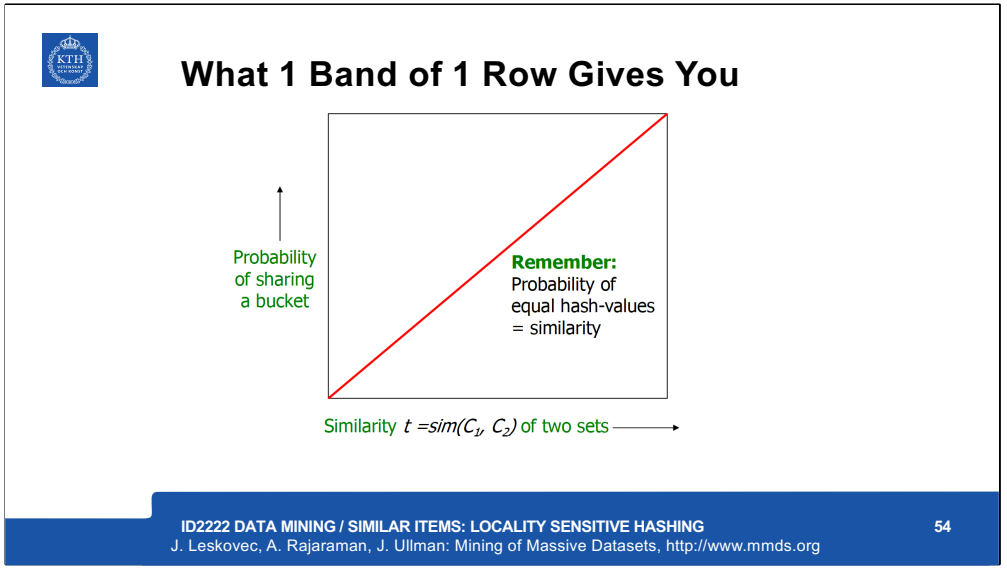
- Example: If we had only 15 bands of 5 rows, the number of false positives would go down, but the number of false negatives (missed candidate pairs) would go up



It may not be obvious, but regardless of the chosen constants b and r, this function has the form of an S-curve, as suggested in Fig. 3.8. The threshold, that is, the value of similarity s at which the probability of becoming a candidate is 1/2, is a function of b and r. Choose a threshold t that defines how similar documents have to be in order for them to be regarded as a desired “similar pair.” Pick a number of bands b and a number of rows r such that br = n and the threshold t is approximately (1/b)^1/r. If avoiding false negatives is important, you may select b and r to produce a threshold lower than t; if speed is important and you wish to limit false positives, select b and r to have a higher threshold.



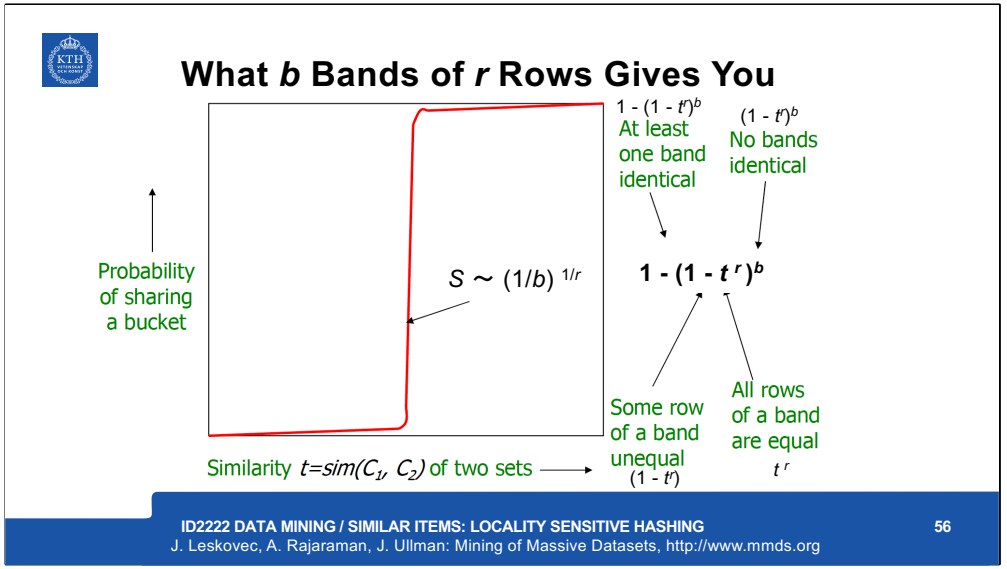
The function looks like a step function that we need to have (see slide before the previous one).



If two documents have similarity < s, say p1, then the probability to share the backet is at least p1; if the probability of <p1 and they are placed in the same bucket than they are false positive.

b Bands Each of *r* Rows

- Columns C_1 and C_2 have similarity ***t***
- Pick any band (***r*** rows)
  - The probability that **all rows in the band equal** = t^r
  - The probability that **some row in the band unequal** = $1 - t^r$
- Probability that **no band identical** = $(1 - t^r)^b$
- Probability that **at least 1 band identical** = $1 - (1 - t^r)^b$



It may not be obvious, but regardless of the chosen constants b and r, this function has the form of an S-curve, as suggested in Fig. 3.8. **The threshold, that is, the value of similarity s at which the probability of becoming a candidate is 1/2, is a function of b and r**. The threshold is roughly where the rise is the steepest. For large b and r, we find that pairs with similarity above the threshold likely become candidates. In contrast, those below the threshold are unlikely to become candidates – precisely the situation we want. An approximation to the threshold is (1/b) 1/r. For example, if b = 16 and r = 4, then the threshold is approximately at s = 1/2 since the 4th root of 1/16 is 1/2.

Example: $b = 20$; $r = 5$

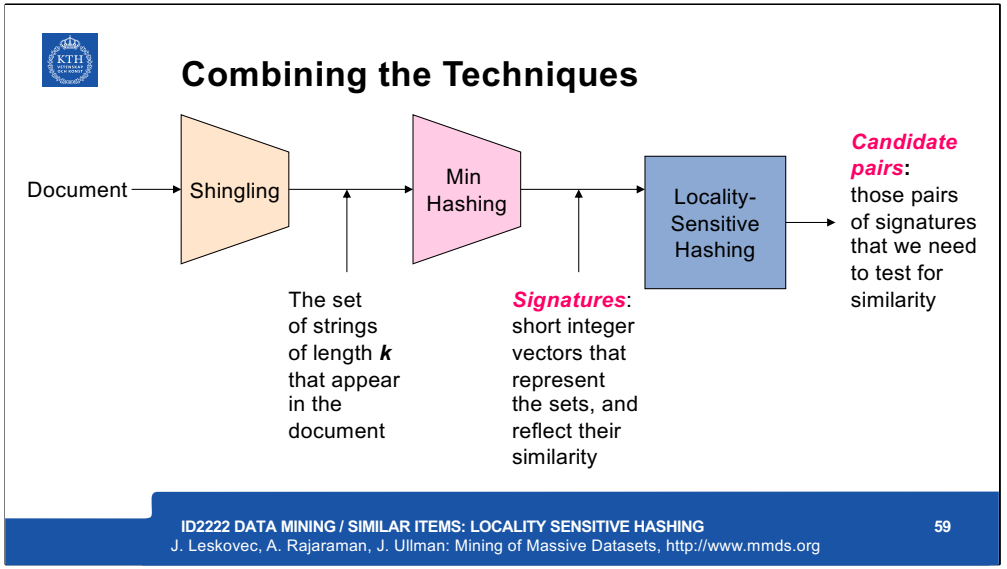
Probability that at least 1 band is identical:

t	$1 - (1 - t^r)^b$
0,2	0,006
0,3	0,047
0,4	0,186
0,5	0,470
0,6	0,802
0,7	0,975
0,8	0,9996

$s \approx (1/b)^{1/r} = 0{,}549$

LSH Summary

- Tune ***r*** and ***b*** to get almost all pairs with similar signatures but eliminate most pairs that do not have similar signatures.
- Check that candidate pairs really do have similar signatures.
- Optional: In another pass through data, check that the remaining candidate pairs really represent similar sets.


Combining the Techniques
Document
Shingling
Min Hashing
Locality-Sensitive Hashing
Candidate pairs: those pairs of signatures that we need to test for similarity
The set of strings of length k that appear in the document
Signatures: short integer vectors that represent the sets, and reflect their similarity
ID2222 DATA MINING / SIMILAR ITEMS: LOCALITY SENSITIVE HASHING
59
J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Combining the Techniques

1. ***Shingling:*** Pick ***k*** and construct the set of k –shingles from each document. Optionally, hash the k -shingles to shorter bucket numbers.
2. Sort the document-shingle pairs to order them by shingle.
3. ***MinHashing***: Pick a length ***n*** for the minhash signatures. Compute the ***minhash signatures for all the documents*** (see the algorithm in Section 3.3.5).

Combining the Techniques (cont)

4. ***LSH***: Choose a similarity threshold ***s*** to classify a pair of sets (signatures) as a desired "***similar pair***."
   Pick the number of bands ***b*** and the number of rows ***r*** such that ***b*r = n***, and the threshold ***s*** ≃ $(1/b)^{1/r}$.
   Tradeoff: To avoid *false negatives,* select ***b*** and ***r*** to produce a threshold lower than ***s***; if speed is important and you wish to limit *false positives*, select ***b*** and ***r*** to produce a higher threshold.
5. Find *candidate pairs* by applying LSH (Section 3.4.1).

Combining the Techniques (cont)

6. Check each candidate pair's signatures if the fraction of components in which they agree is at least ***s***, i.e., if they are similar at least ***s***
7. Optionally, if the signatures are sufficiently similar, check if the corresponding documents are truly similar.

Distance Measures

Generalized LSH is based on some kind of "***distance***" between points.

Similar points are "***close***."

Axioms of a Distance Measure

d is a distance measure if it is a function from pairs of points to real numbers such that:

1. $d(x,y) \geq 0$.
2. $d(x,y) = 0$ iff $x = y$.
3. $d(x,y) = d(y,x)$.
4. $d(x,y) \leq d(x,z) + d(z,y)$ (*triangle inequality*).

Euclidean Distances

In *n*-dimensional Euclidean space, points are *vectors of n real numbers*.

The conventional distance L_2-norm

$$d([x_1, x_2, \ldots, x_n],\ [y_1, y_2, \ldots, y_n]) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Jaccard Distance

$$d(x, y) = 1 - \text{SIM}(x, y)$$

The Jaccard distance is 1 minus the ratio of the sizes of the intersection and union of sets x and y.

Cosine Distance

- Used in spaces with dimensions, where points may be thought of as directions.
- The cosine distance between two points is ***the angle*** that the vectors to those points make.

To calculate the cosine distance:

1. compute the cosine of the angle
2. apply the arc-cosine function to translate to an angle in the 0-180 degree range.

Edit Distance

Edit distance between two strings x and y is the smallest ***number of insertions and deletions*** of single characters that will convert x to y.

The edit distance d(x, y) = length(x) + length(y) – 2* length(LCS)

- Here LCS is the longest common subsequence of x and y;
- to find LCS, delete characters from x and y until they are identical.

Hamming Distance

The Hamming distance between two vectors in a vector space is the number of components in which they differ.

Distance Measures

Generalized LSH is based on some "***distance***" between points.

Similar points are "***close***."

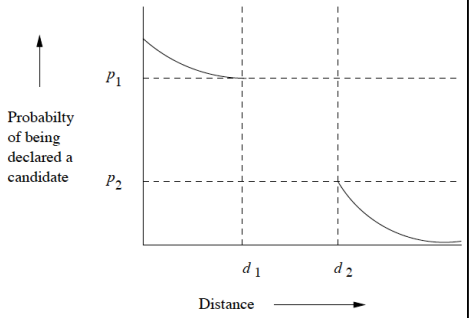
Locality-Sensitive Functions – General Definition

Generalized LSH is based on some "***distance***" between points. Similar points are "***close***."

Let $d_1 < d_2$ be two distances according to some distance measure d.

A family $\boldsymbol{F}$ of hash functions is said to be ***(d_1, d_2, p_1, p_2)-sensitive*** if for every $\boldsymbol{f}$ in $\boldsymbol{F}$:

1. If $d(x, y) \leq d_1$, then the probability that $f(x) = f(y)$ is at least p_1.
2. If $d(x, y) \geq d_2$, then the probability that $f(x) = f(y)$ is at most p_2.



ID2222 Data Mining / Similar Items: Locality Sensitive Hashing

Lecture starts 10:15