ID2222: Data Mining / Lecture 2

KTH ROYAL INSTITUTE
OF TECHNOLOGY

Finding Similar Items: Locality Sensitive Hashing

Vladimir Vlassov

Acknowledgement: Most 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

Outline

- Applications
- Shingling
- Min-hashing
- LSH: Locality-Sensitive Hashing
- Distance Measures

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 $\boldsymbol{x_1, x_2, \ldots}$;
- 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

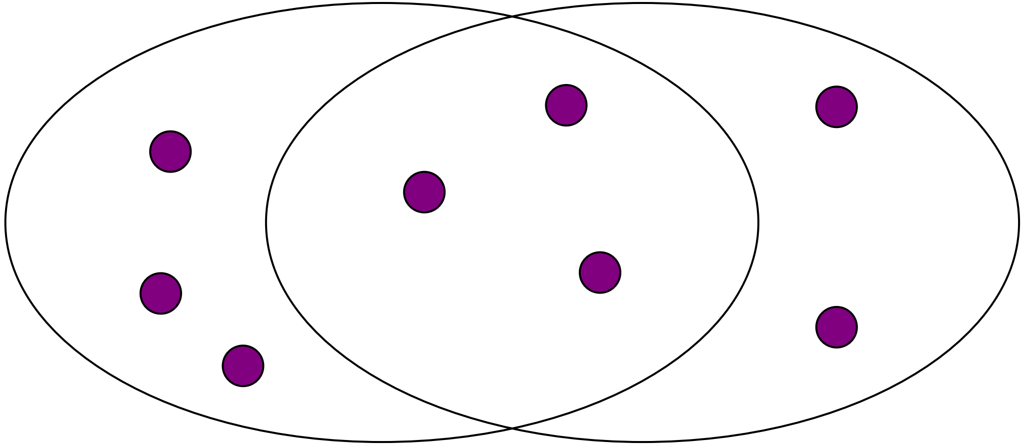
Many data-mining problems can be expressed as finding ***"similar" sets***

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:

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

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

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

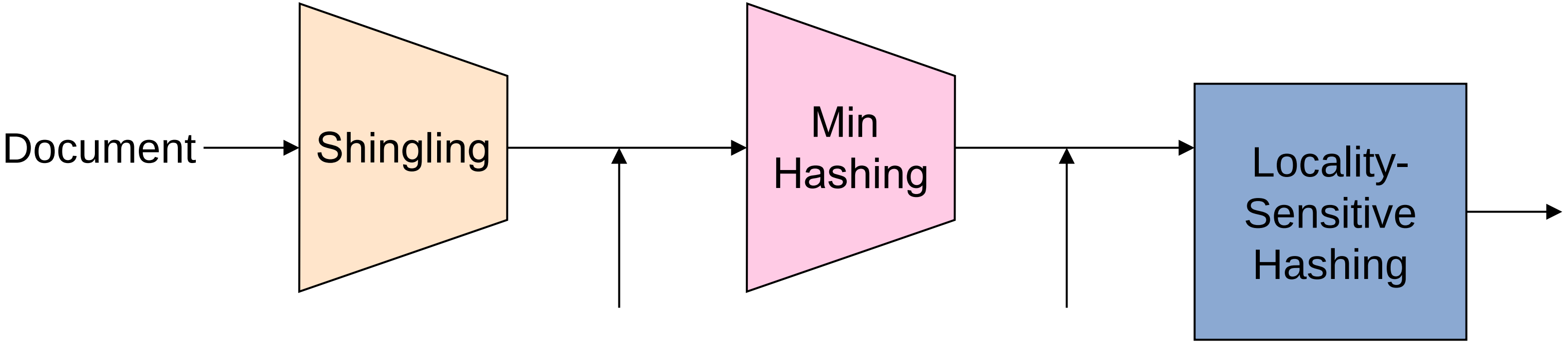
Problems:

- Common text, not common topic.
- Many small pieces of one document may appear out of order in another.
    - Special cases are easy, 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 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

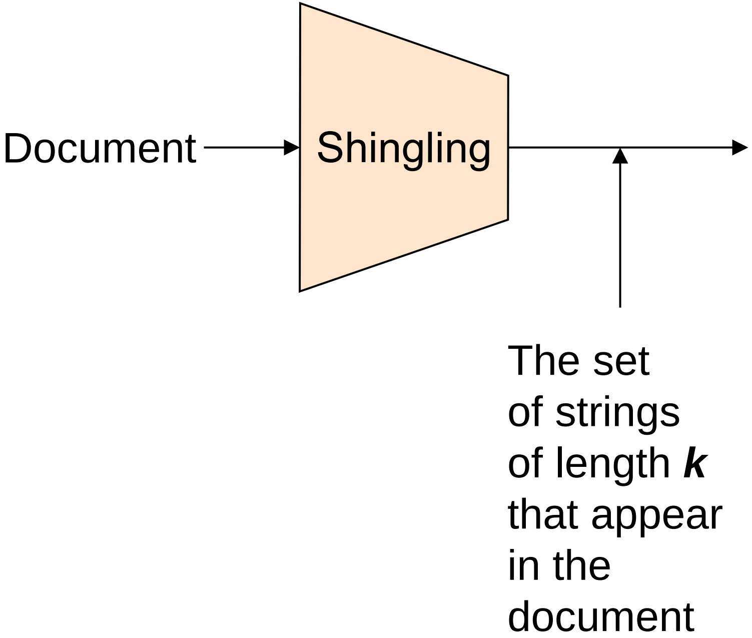


Candidate pairs: those pairs of signatures that we need to test for similarity

The set of strings of length ***k*** that appear in the document

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

Step 1: *Shingling*: Convert documents to sets



Documents as Sets

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 document

- 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}`

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

Similarity Metric for Shingles

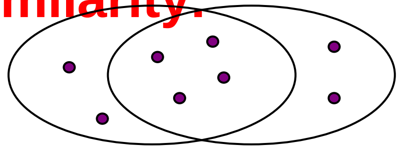
Document D is a set of its k-shingles C=S(D)

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:

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



Working Assumption and Observation

Documents that have lots of shingles in common have similar text, even if the text appears in 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

Example

Find similarity of D_1= "*editorial*" and D_2 = "*factorial*"

k=1

sim({e, d, i, t, o, r, a, l}, {f, a, c, t, o, r, i, l}) = 6/10 = 0.6

k=5

sim({edito, ditor, itori, toria, orial}, {facto, actor, ctori, toria, orial}) = 2 / 8 = 0.25

k=9

sim({editorial}, {factorial}) = 0

Compressing Shingles

To **compress long shingles**, one can **hash** them to (say) 4 bytes to fit in integer

Represent a document by the set of hash values of its *k*-shingles

- For instance, one could construct the set of 9-shingles for a document and then map each of those 9-shingles to a bucket number in the range 0 to $2^{32} - 1$.
  - Each shingle is represented by 4B integer instead of 9B string.

Motivation for Minhash/LSH

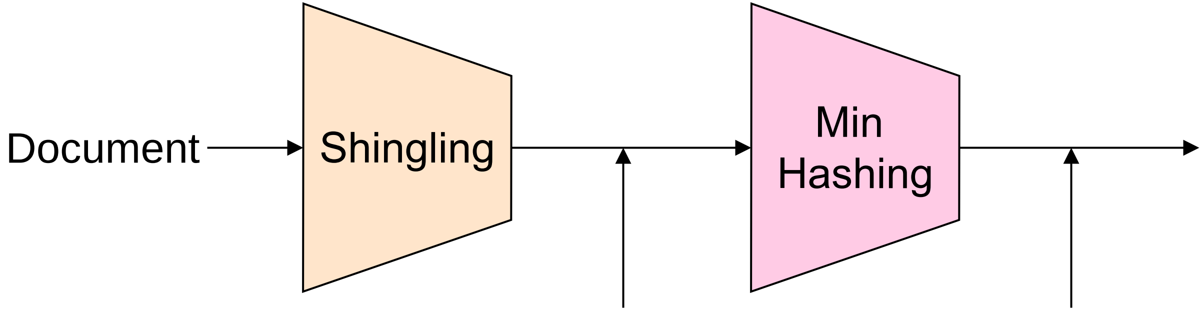
Suppose we need to find near-duplicate documents among $N = 1$ million documents

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 secs/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



The set of strings of length ***k*** that appear in the document

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

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

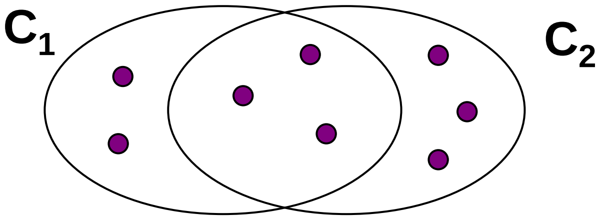
The ***Jaccard similarity*** of two sets is the size of their intersection divided by the size of their union.

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

$$\textbf{\textit{Jaccard} distance: } d(C_1, C_2) = 1 - \textit{Sim}(C_1, C_2)$$

Example

- Size of intersection = 3; size of union = 4,
- Jaccard similarity: $sim(C_1,C_2) = 3/4$
- Distance: $d(C_1,C_2) = 1 - sim(C_1,C_2) = 1/4$



From Sets to Boolean Matrices

Visualise a collection of sets as *characteristic matrix*

Rows correspond to elements (shingles) of the universal set.

Columns correspond to sets (documents)

- 1 in row ***e*** and column ***S*** if and only if ***e*** is a member of ***S*,** i.e. $\boldsymbol{e} \in \boldsymbol{S}$

Column similarity is the Jaccard similarity of the corresponding sets (rows with value 1)

  – Can be computed using bitwise AND and bitwise OR

- **Typical matrix is sparse!**

Example: Jaccard Similarity of Columns (Sets)

	C_1	C_2		
s_1	0	1		*
s_2	1	0		*
s_3	1	1	*	*
s_4	0	0		
s_5	1	1	*	*
s_6	0	1		*

$$\mathrm{Sim}(C_1, C_2) = 2 / 5 = 0{,}4$$

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 = # rows of type a , etc.

Note $Sim(C_1, C_2) = a/(a+b+c)$.

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 $\boldsymbol{h}$ is a hash function that maps the members of a set $\boldsymbol{S}$ to distinct integers.

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

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

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

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

MinHash Signature of a Set

Key idea:

- Take ***k*** (e.g. *k*=100) independent hash functions, e.g., $\boldsymbol{h(x) = (ax+b)\%c}$
- 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 *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, 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.

Minhashing – Example

Hash functions (permutations)

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

Input matrix

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 (indexes of rows with first 1after permutations)

3	1	3	1
2	1	3	1
1	2	1	2

Similarity of Signatures

For two sets S_1 and S_2, the probability

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

is the fraction of the minHash functions in which they agree

- 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***.

This probability is *the Jacard similarity* of the two corresponding 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.

Similarity of Signatures – Example

Hash functions (permutations)

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

Input matrix

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 (indexes of rows with first 1after permutations)

2	1	2	1
2	1	3	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	0	0

Scalability Issue with Permutations

- Suppose 1 billion rows (shingles).
- Hard to pick a random permutation of 1…billion.
- Also, representing a random permutation requires 1 billion entries.
- And accessing rows in permuted order may lead to thrashing.

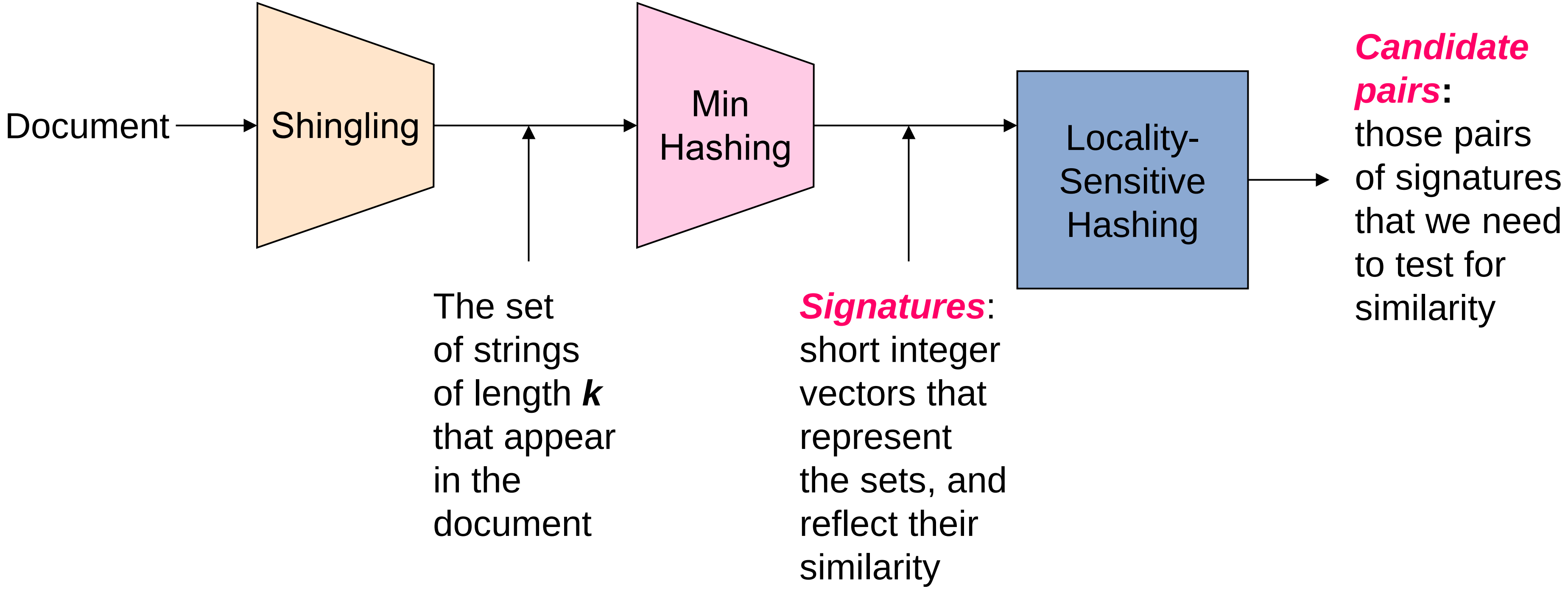
  - Better yet to use several has functions, say, 100, to “generate” 100 “permutations” of rows and compute minimal hash value for elements (rows with 1) of each column corresponding to a set

Implementation

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


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

Step 3: Locality-Sensitive Hashing



Scalability Issue

Assume we have build 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 microsecond/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 t, a fraction < 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 t of the rows.
  - I.e., $M(i, c) = M(i, d)$ for at least fraction t values of i.

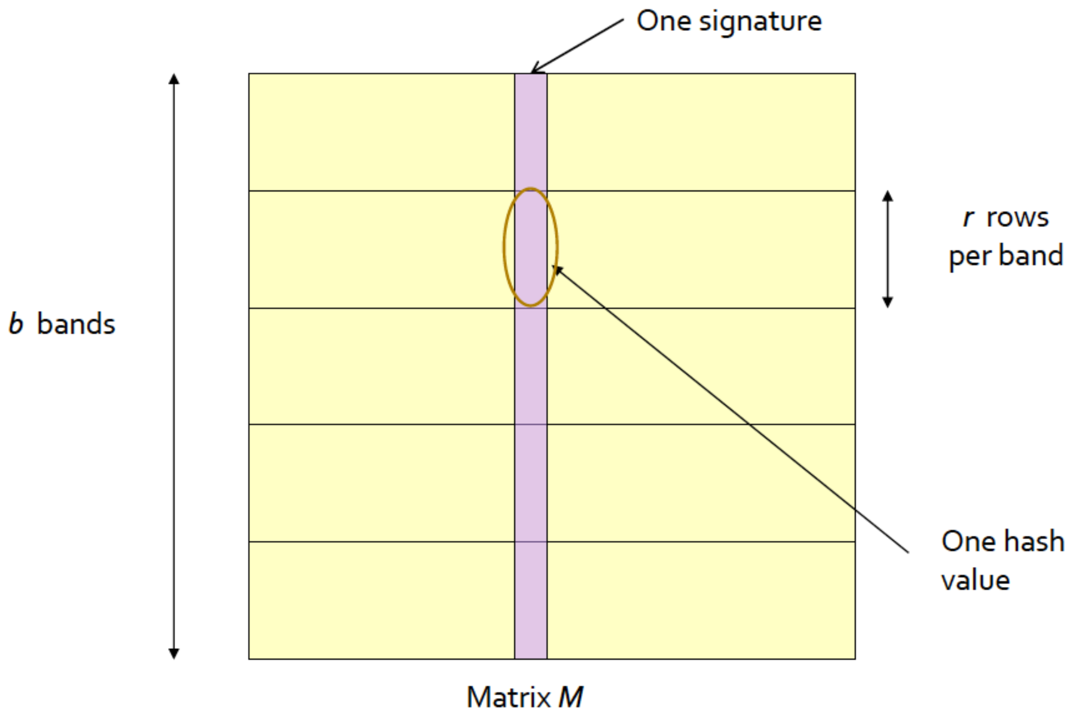
LSH for MinHash Signatures

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.

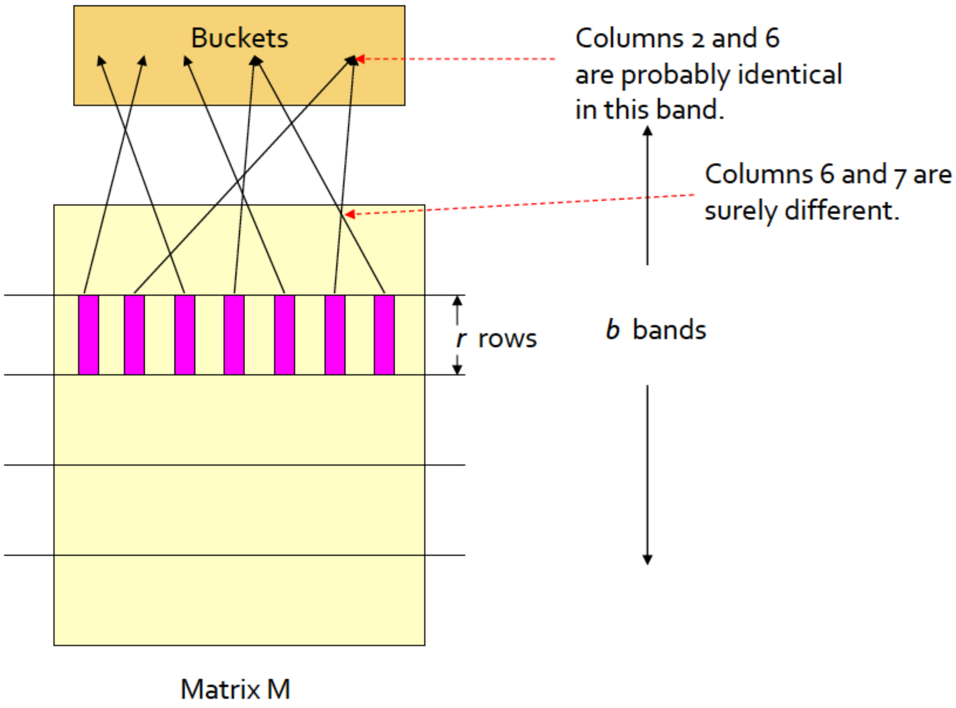
Partition into Bands



Partition into Bands (cont)

- Divide matrix *M* into *b* bands of *r* rows.
- For each band, hash its portion of each column to a hash table with *k* buckets.
  - Make *k* as large as possible.
- *Candidate* column pairs are those that hash to the same bucket for ≥ 1 band.
- Tune *b* and *r* to catch most similar pairs, but few non-similar pairs.

Hash Function for One Bucket



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.”

Example: Bands

- Suppose 100,000 columns.
- Signatures of 100 integers.
- Therefore, signatures take 40Mb.
    - They fit easily into 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 raws/band.

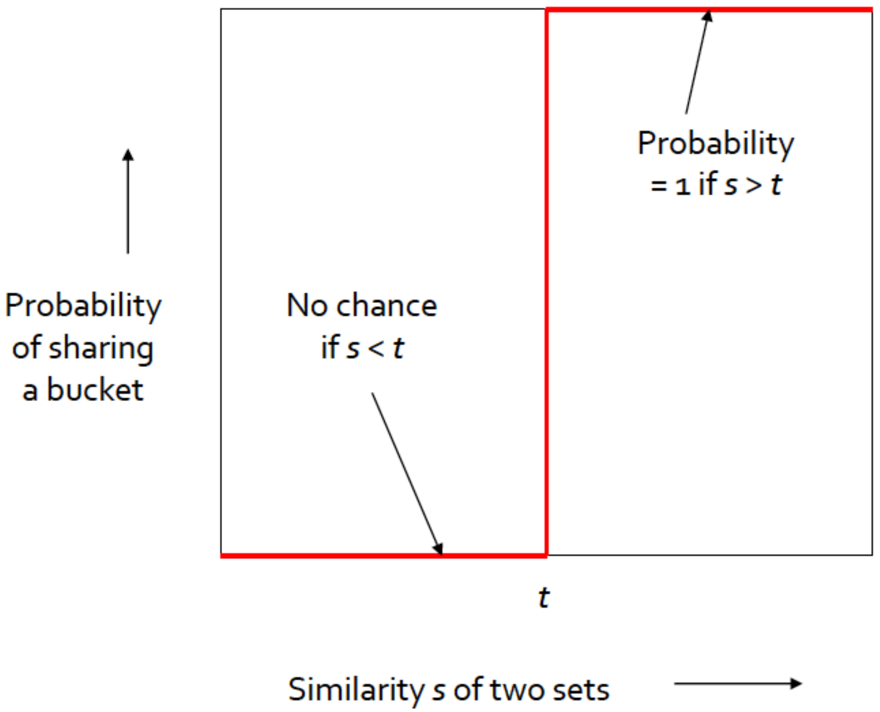
Suppose C_1 and C_2 are 80% Similar

- Probability C_1, C_2 identical in one particular band: $(0.8)^5 = 0{,}328$.
- Probability C_1, C_2 are *not* similar in any of the 20 bands: $(1-0.328)^{20} = 0{,}00035$.
  - i.e., about 1/3000th of the 80%-similar underlying sets are *false negatives*.

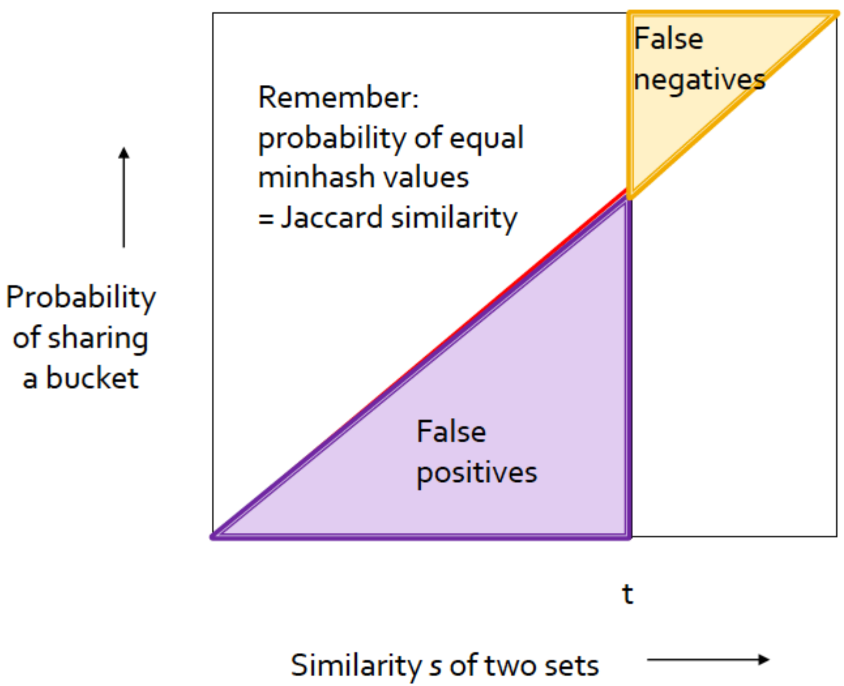
Suppose C_1 and C_2 Only 40% Similar

- Probability C_1, C_2 identical in any one particular band: $(0.4)^5 = 0{,}01$.
- Probability C_1, C_2 identical in ≥ 1 of 20 bands: $\leq 20 * 0.01 = 0{,}2$.
- But *false positives* much lower for similarities << 40%.

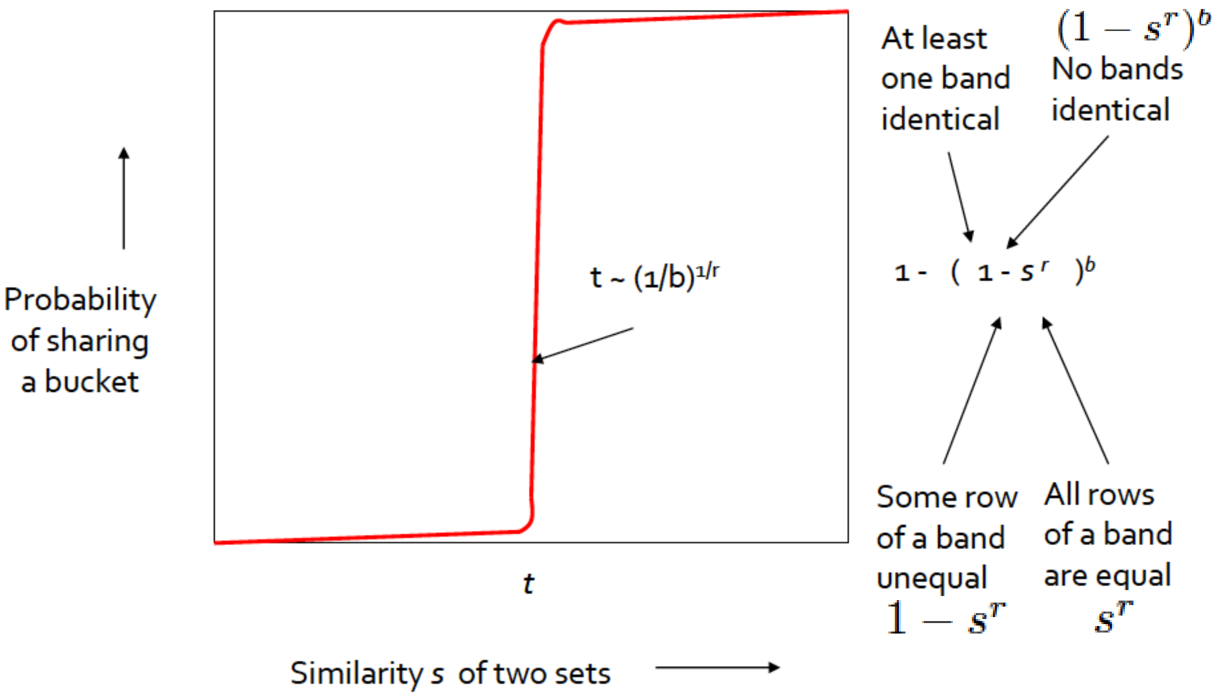
Analysis of LSH – What we want



One Band of One Raw

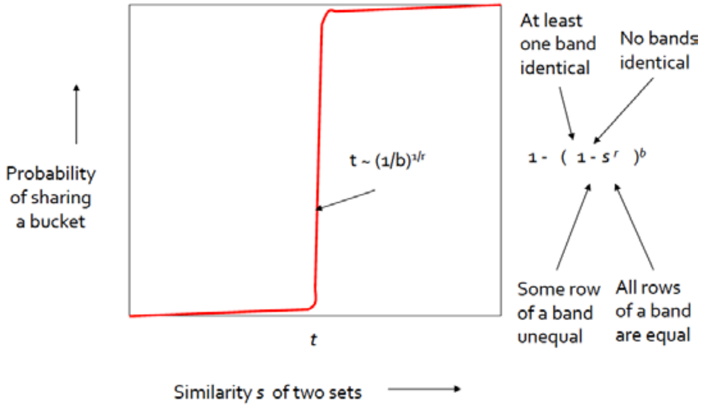


b Bands of *r* Raws



Example: b = 20; r = 5

s	$1 - (1- s^r)^b$
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

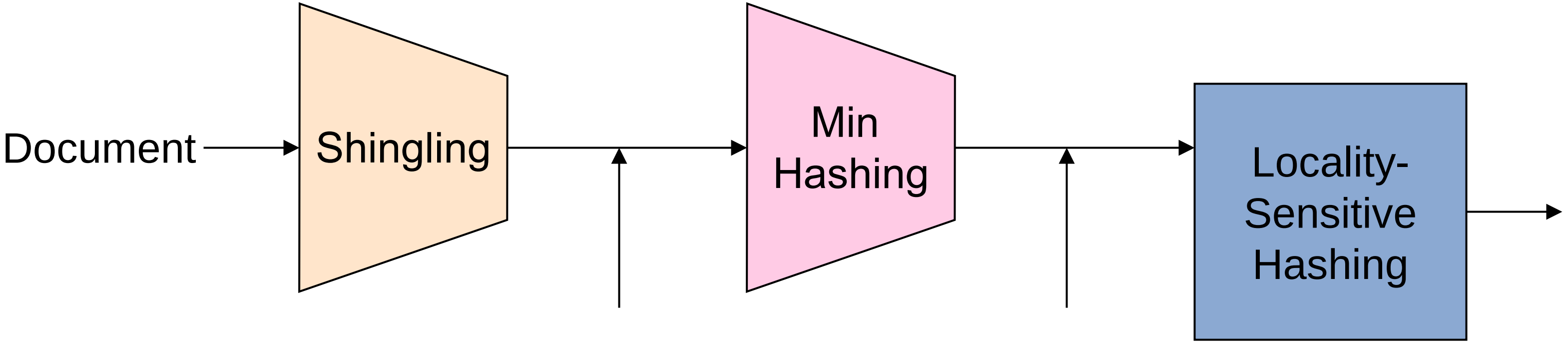


$t \approx (1/b)^{1/r} = 0{,}549$

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



The set of strings of length ***k*** that appear in the document

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

Candidate pairs: those pairs of signatures that we need to test for similarity

Combining the Techniques

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

Combining the Techniques (cont)

4. ***LSH***: Choose a similarity threshold ***t*** to classify a pair of sets (signatures) as a desired "***similar pair***."
   Pick a number of bands ***b*** and a number of rows ***r*** such that $\boldsymbol{br = n}$, and the threshold $\boldsymbol{t \simeq (1/b)^{1/r}}$.
   Tradeoff: To avoid *false negatives* 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 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 ***t***, i.e. if they are similar at least ***t***
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. $d(x,y) > 0$.
2. $d(x,y) = 0$ iff $x = y$.
3. $d(x,y) = d(y,x)$.
4. $d(x,y) < d(x,z) + d(z,y)$ (*triangle inequality*).

Euclidean Distances

In an *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

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

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 $d(x, y) = \text{length}(x) + \text{length}(y) - 2^* \text{length}(LCS)$

- Here LCS is a 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 kind of “***distance***” between points.

Similar points are “***close***.”

Locality-Sensitive Functions – General Definition

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

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

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

1. If $d(x, y) \le d_1$, then the probability that $f(x) = f(y)$ is at least p_1.

2. If $d(x, y) \ge d_2$, then the probability that $f(x) = f(y)$ is at most p_2.