大数据分析

Statistics and Counting

程学旗
靳小龙
刘盛华

Finding Similar Items

- Applications
  - Pages with similar words
    - For duplicate detection, classification by topic
  - Customers who purchased similar products
    - Products with similar customer sets
  - Gene methylation-expression correlation networks (850k)
- Approach
  - Find near-neighbors in high-dimensional space

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

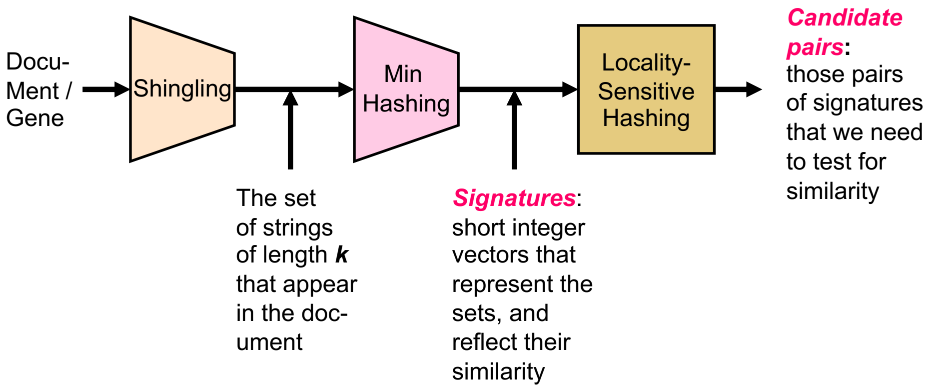
Task: Finding Similar Documents

- Goal: Given a large number (N in the millions or billions) of documents, find "near duplicate" pairs
- Problems:
  - Many small pieces of one document can appear out of order in another
  - Naive solution would take $O(N^2)$

Essential Steps for Similar Docs

- Shingling
  - Convert documents to sets
- Min-Hashing
  - Convert large sets to short signatures, while preserving similarity
- Locality-Sensitive Hashing (LSH)
  - Focus on pairs of signatures likely to be from similar documents (correlated Gene)
  - Candidate pairs

The Big Picture



Step 1: Shingling

- A k-shingle (or k-gram) for a document is a sequence of *k* tokens that appears in the doc
- Example
  - k=2; document D1 = abcab
    Set of 2-shingles: S(D1) = {ab, bc, ca}
  - account for ordering of words

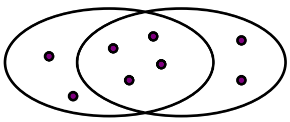
Compressing Shingles

- To compress long shingles, we can hash them to (say) 4 bytes
- Represent a document by the set of hash values of its *k*-shingles
  - Idea: Two documents could (rarely) appear to have shingles in common, when in fact only the hash-values were shared
- Example: k=2; document D_1= abcab
  Set of 2-shingles: $S(D_1)$ = {ab, bc, ca}
  Hash the singles: $h(D_1)$ = {1, 5, 7}

Similarity Metric for Shingles

- Document D_1 is a set of k-shingles $C_1=S(D_1)$
  - Equal to 0/1 vector in the space of *k*-shingles
- Jaccard similarity

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



From Sets to Boolean Matrices

- Rows = elements (shingles)
- Columns = sets (documents)
  - 1 in row *e* and column *s* if and only if *e* is a member of *s*
  - Column similarity is the Jaccard similarity of the corresponding sets
  - 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

9

Working Assumption

- 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

10

Motivation for Minhash/LSH

- Suppose we need to find near-duplicate documents among $N = 1$ million documents
- Naively, computing 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…

Step 2: Minhashing

- Convert large sets to short signatures, while preserving similarity
- Key idea: "hash" each column C to a small signature h(C), such that:
  - h(C) is small enough that the signature fits in RAM
  - sim(C_1, C_2) is the same as the "similarity" of signatures h(C_1) and h(C_2)
- Suitable hash function for the Jaccard similarity

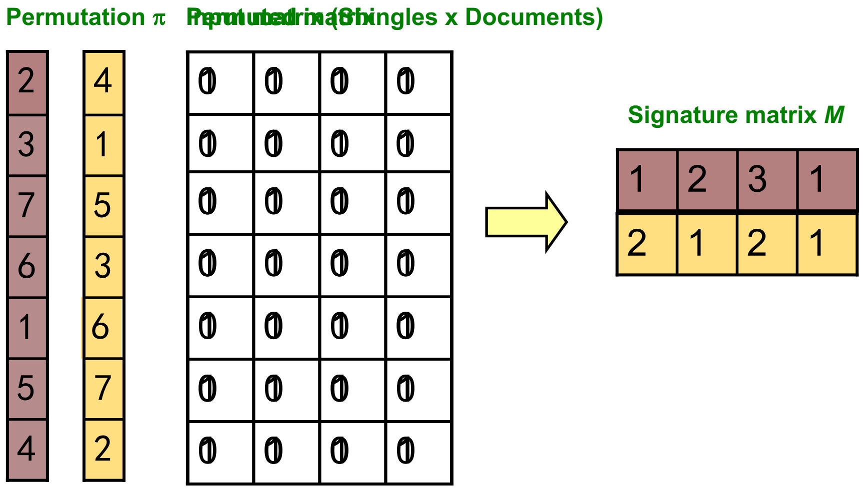
Min-Hashing

- 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:

$$h_\pi(C) = \min_\pi \pi(C)$$ 第一个非零行的index

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

Min-Hashing Example



The Min-Hash Property

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

- Choose a random permutation π
- Claim: $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
  - 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***
  - prob. that both are true is the prob. $y \in C_1 \cap C_2$
  - $Pr[\min(\pi(C_1))=\min(\pi(C_2))]=|C_1 \cap C_2|/|C_1 \cup C_2| = sim(C_1, C_2)$

15

Min-Hash Signatures

1	2	3	1
2	1	2	1

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

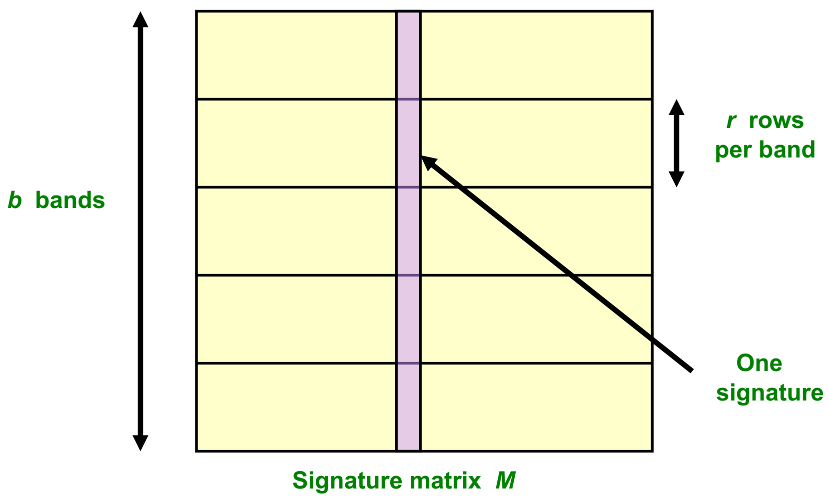
16

Step 3: LSH

- Goal
  - Find documents with Jaccard similarity at least *s*
- General idea
  - tells whether x and y is a candidate pair: a pair of elements whose similarity must be evaluated
- For Min-Hash matrices:
  - Hash columns of signature matrix *M* to many buckets
  - Each pair of documents that hashes into the same bucket is a candidate pair

Partition *M* into *b* Bands

1	2	3	1
2	1	2	1

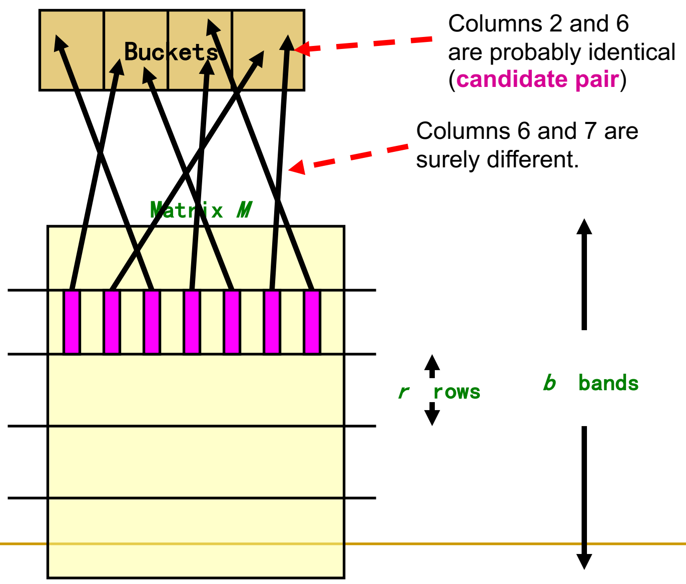


Partition M into Bands

- 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
- Candidate column pairs:
  - hash to the same bucket for ≥ 1 band
- Tune b and r to catch most similar pairs, but few non-similar pairs

19

Hashing Bands



20

Example of Bands

- Find pairs of ≥ s=0.8 similarity, set b=20, r=5
  - C_1, C_2 to be a candidate pair: hash to at least 1 common bucket
    - Probability: $(0.8)^5 = 0.328$
- Probability C_1, C_2 are not similar in all of bands:

$$(1\text{-}0.328)^{20} = 0.00035$$

  - i.e., about 1/3000th of the 80%-similar column pairs are false negatives (we miss them)
  - We would find 99.965% pairs of truly similar documents

21

22

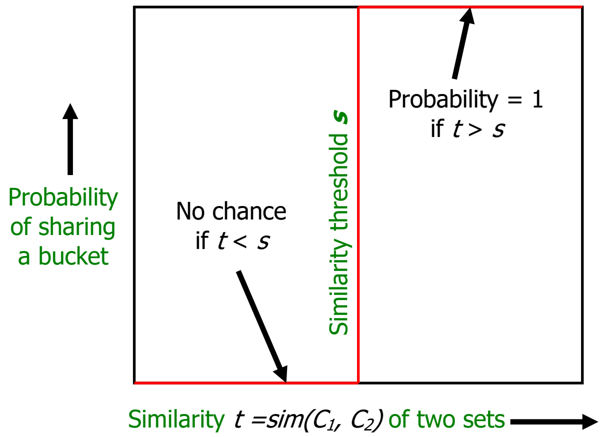
C_1, C_2 are 30% Similar

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

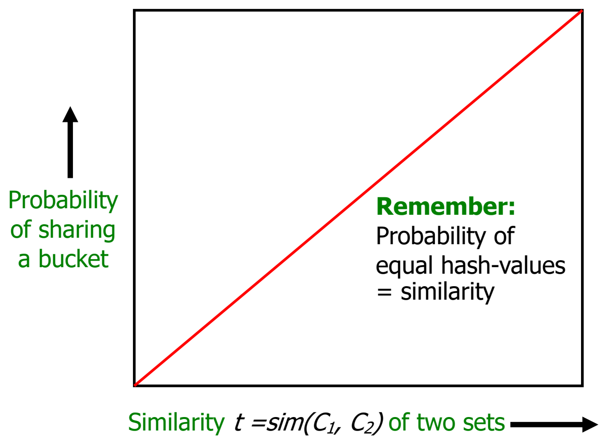
- **Find pairs of ≥ s=0.8 similarity, set b=20, r=5**
- **Assume: sim(C_1, C_2) = 0.3**
  - **Since sim(C_1, C_2) < s we want C_1, C_2 to hash to NO common buckets (all bands should be different)**
- **Probability C_1, C_2 identical in one particular band: $(0.3)^5$ = 0.00243**
- **Probability C_1, C_2 identical in at least 1 of 20 bands: 1 - $(1 - 0.00243)^{20}$ = 0.0474**
  - **In other words, approximately 4.74% pairs of docs with similarity 0.3% end up becoming candidate pairs**
    - **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**

23

Analysis of LSH – What We Want



What 1 Band of 1 Row Gives You



24

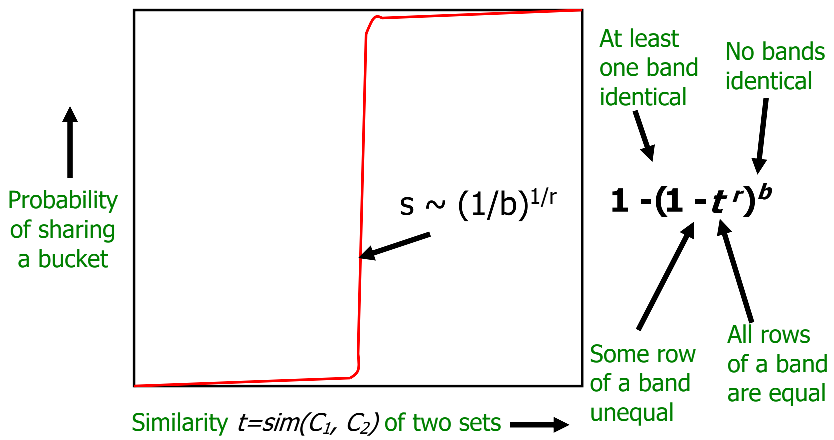
LSH Involves a Tradeoff

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

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

25

What *b* Bands of *r* Rows Gives You



26

27

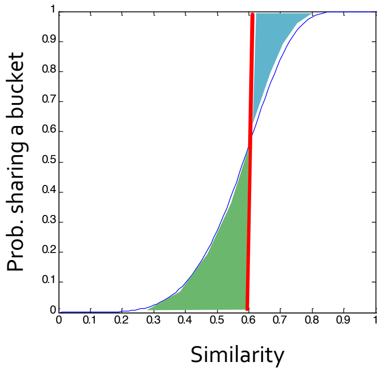
Example: *b* = 20; *r* = 5

- Similarity threshold s
- Prob. that at least 1 band is identical:

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

Picking *r* and *b*: The S-curve

- Picking *r* and *b* to get the best S-curve
  - 50 hash-functions (r=5, b=10)



Blue area: False Negative rate
Green area: False Positive rate

28

LSH Summary

- Tune M, b, r to get almost all pairs with similar signatures, but eliminate most pairs that do not have similar signatures
- Check in main memory that candidate pairs really do have similar signatures

Summary: 3 Steps

- Shingling: Convert documents to sets
  - use hashing to assign each shingle an ID
- Min-Hashing: Convert large sets to short signatures
  - use similarity preserving hashing to generate signatures with property $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
  - use hashing to get around generating random permutations
- Locality-Sensitive Hashing: Focus on pairs of signatures likely to be from similar documents
  - use hashing to find candidate pairs of similarity $\geq$ s

Count-Min sketch(CMS)

Management = Measurement + Control

- Traffic engineering
  - Identify large traffic aggregates, traffic changes
  - Understand flow characteristics (flow size, delay, etc.)

- Performance diagnosis
  - Why my application has high delay, low throughput?

- Accounting
  - Count resource usage for tenants

Measurement is Increasingly Important

- Increasing network utilization in larger networks
  - Hundreds of thousands of servers and switches
  - Up to 100Gbps in data centers
  - Google drives WAN links to 100% utilization
- Requires better measurement support
  - Collect fine-grained flow information
  - Timely report of traffic changes
  - Automatic performance diagnosis

Yet, measurement is underexplored

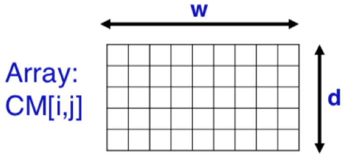
- Vendors view measurement as a secondary citizen
  - Control functions are optimized w/ many resources
  - NetFlow/sFlow are too coarse-grained
- Operators rely on postmoterm analysis
  - No control on what (not) to measure
  - Infer missing information from massive data
- Network-wide view of traffic is especially difficult
  - Data are collected at different times/places

Intro to sketches

- "Sketch" data structures are compact, randomized summaries
- Common sketch properties
  - Approximate a holistic function
  - Sublinear in size of the input
  - Linear transform of input
  - Can easily merge sketches

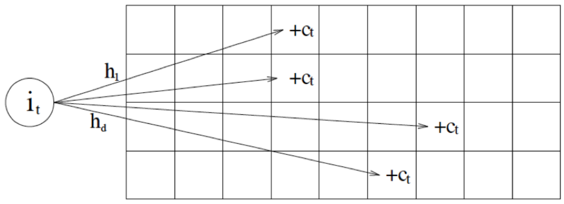
Count-Min sketch(CMS)

- Model incremental Stream as a vector of dimension n
  - Each dimension represents an entry index
  - Current state at time t is $a(t) = [a_1(t), \ldots, a_n(t)]$
    - $a_i(t)$ means the number of entry i at time t
    - d hash functions $h_1 \ldots h_d: \{1, \ldots, n\} \rightarrow \{1, \ldots, w\}$



Update procedure of CMS

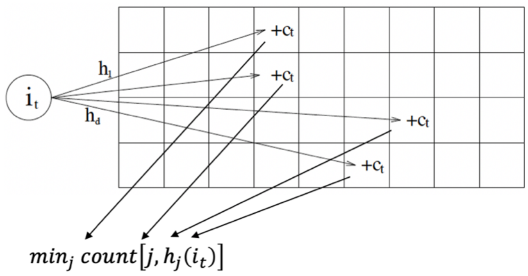
- The tth update is (i_t, c_t), meaning that
  - $a_{i_t}(t) = a_{i_t}(t-1) + c_t$, and $a_{i'}(t) = a_{i'}(t-1)$ for all $i' \neq i_t$
  - Update CMS: Add c_t to one count determined by $h_j$in each row.



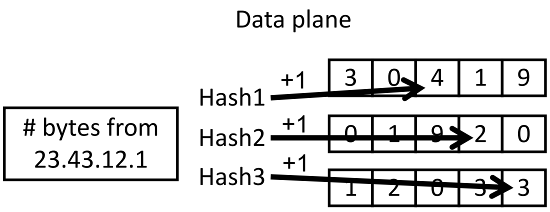
Formally, $\forall 1 \leq j \leq d: count[j, h_j(i_t)] \leftarrow count[j, h_j(i_t)] + c_t$

Point query

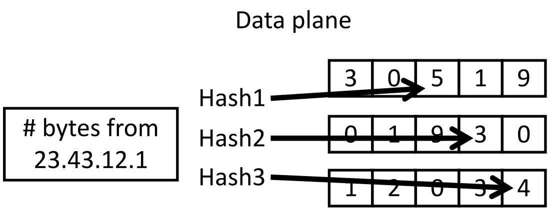
- At any time t, for $i \in [n]$, return an approximation of a_{i_t}.
- Estimation:
  - the approximated result is $\hat{a}_{i_t} = min_j \ count[j, h_j(i_t)]$



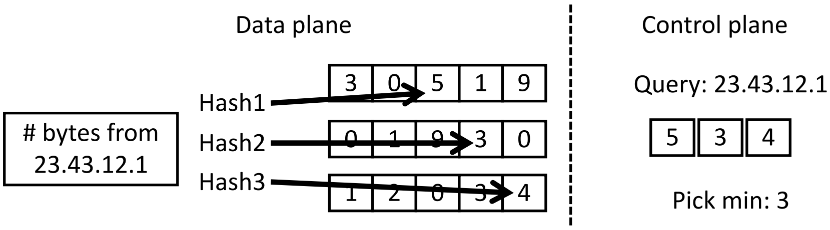
Example



Example



Example



Point query problem

- If $w = 2/\epsilon$and $d = log_2 \delta^{-1}$, we can find an estimate $\hat{a}_{i_t}$ for a_{i_t} that satisfies and with probability at least $1 - \delta$,

$$\hat{a}_{i_t} \le a_{i_t} + \epsilon \|\boldsymbol{a}\|_1,$$

where $\|\boldsymbol{a}\|_1 = \sum_{i=1}^{n} |a_{i_t}(t)|$.

- Memory used is $O(\epsilon^{-1} log_2 \delta^{-1})$.

Range query problem

- At any time t, for $l, r \in [n]$, return an approximation of $\mathrm{a}[l,r] = \sum_{i=l}^{r} a_{i_t}$,
- Range Query Theorem
  - If $w = 2/\epsilon$ and $d = log_2 \delta^{-1}$, we can find an estimate $\hat{a}[l,r]$for $a[l,r]$ that satisfies $a[l,r] \le \hat{a}[l,r]$ and with probability at least $1-\delta$,

$$\hat{a}[l,r] \le a[l,r] + 2\epsilon \log \mathrm{n} \, \|\boldsymbol{a}\|_1$$