15-853:Algorithms in the Real World

Announcement:

- HW3 was released on Tuesday
  - Due on Nov. 20 11:59pm
  - Small typo corrected in Problem 3.2:
    - Use part **(a)** -> Use part **(1)**
- No recitation tomorrow: Work on HW
- Naama will have two office hours: covering Francisco's slot as well
- Scribe volunteer
  - Shorter turn around time for scribing this and the previous lecture? By Monday Nov 18?

15-853:Algorithms in the Real World

Announcement:

Next lecture:

Dimensionality reduction: JL, PCA and time permitting one other topic.

Next Thursday, we will have a recap of the whole course.

Final exam: Nov. 26th

15-853:Algorithms in the Real World

Hashing:

- Concentration bounds
- Load balancing: balls and bins
- Hash functions
- Data streaming model (cont.)
- Hashing for finding similarity

Recall: Data streaming model

- Elements going past in a “stream”
- Limited storage space: Insufficient to store all the elements

A useful abstraction:

Viewing streams as vectors (in high dimensional space)

Stream at time t as a vector $x^t \in Z^{|U|}$

$$x^t = (x^t_1, x^t_2, \ldots, x^t_{|U|})$$

Element i = #times i^{th} element of U has been seen until time t

Leads to an extension of the model where each element of the stream is either

(1) A new element or (2) old element departing (i.e. deletions).

Recall: Heavy hitters

Many ways to formalize the heavy hitters problem.

ε-heavy-hitters: Indices i such that $x_i > \varepsilon \| x \|_1$

Let us consider a simpler problem.

Count-Query:

At any time t, given an index i, output the value of x^t_i with an error of at most $\varepsilon\|x^t\|_1$. I.e., output an estimate

$$y_i \in x_i \pm \varepsilon \| x \|_1$$

Recall: Count-min Sketch

A hashing based solution

Let h: U -> [M] be a hash function

Let a A[1...M] be an array capable of storing non-negative integers.

When update a_t arrives

```
        If (a_t == (add, i))
                then A[h(i)]++;
else // a_t == (del, i)
                A[h(i)]--;
```

Estimate for x^t_i: y_i = A[h(i)]

Continue on board

15-853:Algorithms in the Real World

Hashing:

- Concentration bounds
- Load balancing: balls and bins
- Hash functions
- Data streaming model
- Hashing for finding similarity

Material based largely on "Mining of Massive Datasets" book (available free for download!)

Applications

Applications of finding Similar (near-neighbor) Items

- Filter duplicate docs in search engine
- Plagiarism, mirror pages
- Recommend items (e.g., products, movies) to users that were liked by other users who have similar tastes
  - Collaborative Filtering
    - represent movie as a vector of ratings by users
    - represent product by binary vector x: x(j) = 1 if user j bought the item, 0 otherwise

We will specifically focus on the application of **finding similar text documents**

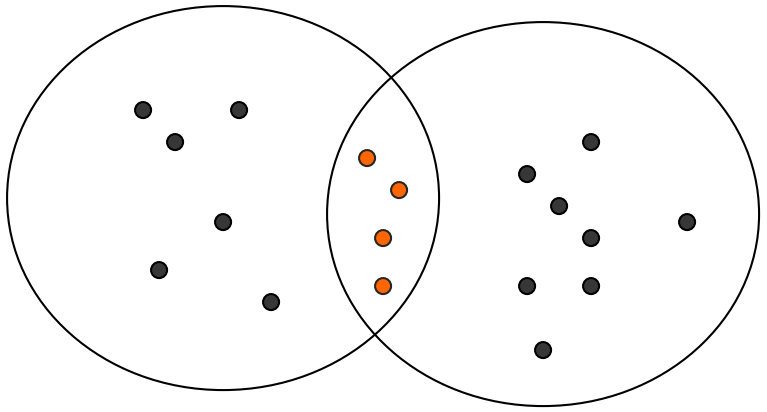
Defining Similarity of Sets

Many ways to define similarity.

One similarity metric, “distance”, for sets

Jaccard similarity

$$\mathrm{SIM}(A, B) := \frac{A \cap B}{A \cup B}$$



A B

4 common
18 total

SIM(A,B) = 4/18 = 2/9

Jaccard distance is 1 – SIM(A, B)

Representing documents as sets: Shingling

Document = string of characters

k-shingle = any substring of length k found within the document

E.g.: 4-shingles of abacbdaeacf -> abac, bacb, acbd, cbda, bdae,…

How to choose k?

If too small:

- Most shingles will appear in most documents
- Documents with no common phrases also will have high similarity

How large should k be?

- Choose k so that any shingle is unlikely to occur in any doc.

Representing documents as sets: Shingling

E.g.,: Emails which are quite short, k = 5 has been found to work well.

k = 5 would mean 27^5 ~ 14M possible shingles.

(27 = letters plus space)

Longer documents need need larger k.

E.g.: For research documents k=9 has been found to work well.

Other aspects come into picture: Should really think of having only 20 letters (excluding rare characters such as x, q, z, etc)

Say choose k=8 or so, so $20^8 \sim 2^{32}$

Representing documents as sets: Shingling

Finally, hash shingles to 32 bit words (“cheap compression”).

Instead of using shingles directly, we hash the strings of length k to some number of buckets and treat the resulting bucket number as the shingle.

Helps to manipulate using single-word machine operations.

Similarity-Preserving Signatures

Too large space needed to store documents using sets of shingles

Even when hashed to 4 bytes each, takes 4x the space

Goal: Compute a smaller representation called "signature" for each set, so that similar documents have similar signatures (and dissimilar docs are unlikely to have similar signatures).

That is, we want to be able to estimate Jaccard similarity between two sets using their signatures alone

Trade-off: length of signature vs. accuracy

Characteristic Matrix of Sets

Element num	Set1	Set2	Set3	Set4
0	1	0	0	1
1	0	0	1	0
2	0	1	0	1
3	1	0	1	1
4	0	0	1	0
…				

Stored as a sparse matrix in practice.

Example from "Mining of Massive Datasets" book by Rajaraman and Ullman

Minhashing

Minhash(π) of a set is the number of the row (element) with first non-zero in the **permuted order π**.

Element num	Set1	Set2	Set3	Set4
1	0	0	1	0
4	0	0	1	0
0	1	0	0	1
3	1	0	1	1
2	0	1	0	1
…				

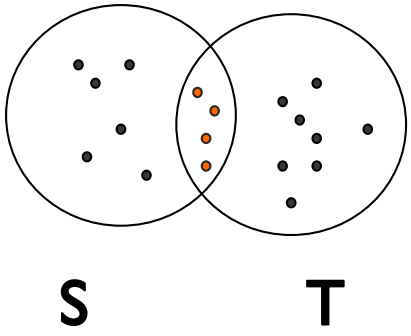
Π=(1,4,0,3,2)

Example from "Mining of Massive Datasets" book by Rajaraman and Ullman

Minhash and Jaccard similarity

Theorem:

P(minhash(S) = minhash(T)) = SIM(S,T)



Proof:

X = rows with 1 for both S and T

Y = rows with either S or T have 1, but not both

Z = rows with both 0

Q: Jaccard similarity?

Q: Probability that row of type X is before type Y in a random permuted order is _____

Representing collection of sets: Minhash signature

Let $h_1, h_2, \ldots, h_n$ be different minhash functions (i.e., independent permutations).

Then **signature** for set S is:

$SIG(S) = [h_1(S), h_2(S), \ldots, h_n(S)]$

Signature matrix:

Rows are minhash functions

Columns are sets

Minhash signature

Signature for set S is:

$SIG(S) = [h_1(S), h_2(S), \ldots, h_n(S)]$

Now how to compute estimate of the Jaccard similarity between S and T using minhash-signatures?

SIM(S,T) ≈ fraction of coordinates where SIG(S) and SIG(T) are the same

Approximating Minhashes

Permuting large a characteristic matrix is infeasible.

(millions to billions of rows)

Solution:

use a **good hash function** that maps rows to positions.

If the rows mapped to distinct positions,
perhaps behaves like random permutation.

Properties of random hashes?

We assume the # collisions is small vs. number of items.

Algorithm

Pick n independent hash functions.

Let SIG(i, c) be the element of the signature matrix for i^{th} hash function and column c.

Initialize SIG(i, c) = ∞

For each row r = 0, 1, …, N-1 of the characteristic matrix:

1. Compute $h_1(r)$, $h_2(r)$, …, $h_n(r)$
2. For each column c:
    1. If column c has 0 in row r, do nothing
    2. Otherwise, for each i = 1,2, …, n set
       SIG(i, c) = min($h_i(r)$, SIG(i, c))

Worked example (on blackboard)

Element num	Set1	Set2	Set3	Set4	x + 1 mod 5	3x +1 mod 5
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3
…						

Signature matrix

	Set1	Set2	Set3	Set4
H1	∞	∞	∞	∞
H2	∞	∞	∞	∞

Worked example (on blackboard)

Element num	Set1	Set2	Set3	Set4	x + 1 mod 5	3x +1 mod 5
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3
…						

Signature matrix

	Set1	Set2	Set3	Set4
H1	1	3	0	1
H2	0	2	0	0

Worked example (on blackboard)

Element num	Set1	Set2	Set3	Set4	x + 1 mod 5	3x +1 mod 5
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3
…						

Signature matrix

	Set1	Set2	Set3	Set4
H1	1	3	0	1
H2	0	2	0	0

LOCALITY SENSITIVE HASHING USING MINHASH

Nearest Neighbors

Assume that we construct a 1,000 byte minhash signature for each document.

Million documents can now fit into 1 gigabyte of RAM.

But how much does it cost to find the nearest neighbor of a document?

- Brute force: N signature-signature matches.

(Closest pair takes N^2 time.)

→ Need a way to reduce the number of comparisons.

LSH requirements

A good LSH hash function will divide input into large number of **buckets.**

To find nearest neighbors for a query item q, we want to only compare with items in the bucket hash(q): "**candidates**".

If two A and B are similar, we want the probability that hash(A) = hash(B) be high.

- *False positives*: sets that are not similar, but are hashed into same bucket.
- *False negatives*: sets that are similar, but hashed into different buckets.

LSH based on minhash

We will consider a specific form of LSH designed for documents represented by shingle-sets and minhahsed to short signatures.

Idea:

divide the signature matrix rows into **b** bands of **r** rows

hash the columns in each band with a basic hash-function

→ each band divided to buckets [i.e., a hashtable for each band]

LSH based on minhash

Idea:

divide the signature matrix rows into **b** bands of **r** rows

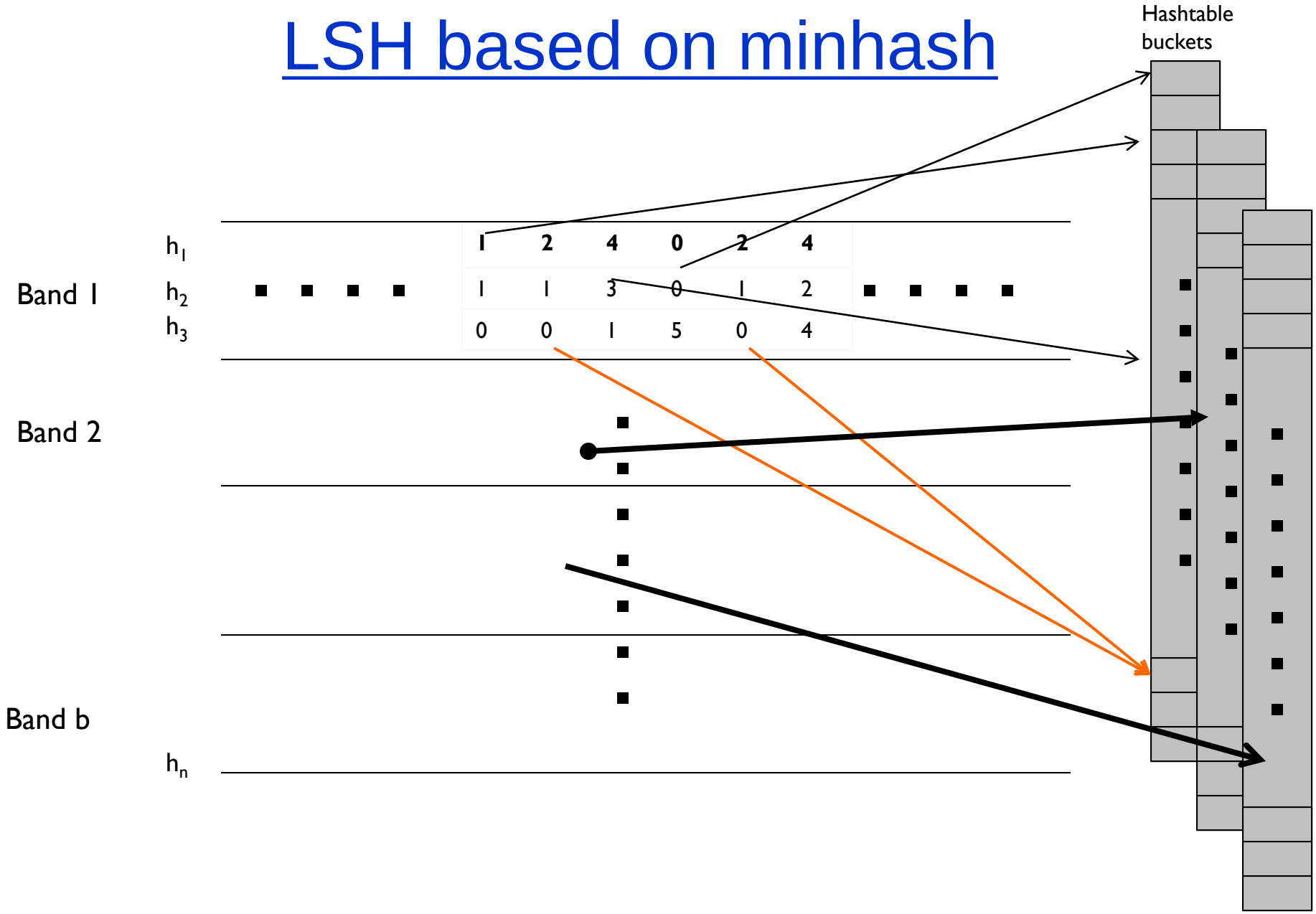
hash the columns in each band with a basic hash-function

→ each band divided to buckets [i.e., a hashtable for each band]

If sets S and T have same values in a band,
they will be hashed into the same bucket in that band.

For nearest-neighbor, the candidates are
the items in the same bucket as query item, in each band.

LSH based on minhash



Analysis

Consider the probability that we find T with query document Q

Let

s = SIM(Q,T) = P{ $h_i(Q) = h_i(T)$ }

b = # of bands

r = # rows in one band

What is the probability that rows of signature matrix agree for columns Q and T in one band?

We will continue in the next class