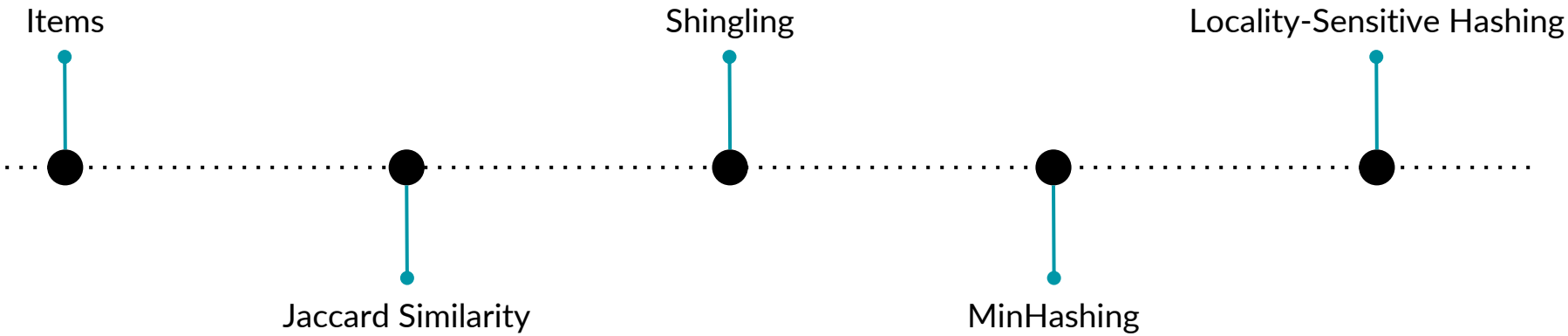
Finding Similar Items in Large Datasets

Zhe Zhou, Qianyan Wu, Zhihao He, Brandon D'Anna

Introduction/Motivation

- Fundamental data mining problem
- Examples include identifying plagiarism, grouping news article from the same source, entity resolution
- Naive approach would require the comparison of each pair of items. The number of pairs can increase drastically as the total number of items increase
- In a world where billion row datasets are common, naive approach is not feasible
- Family of techniques known as locality-sensitive hashing (LSH) focus on pairs likely to be similar
    - Avoids quadratic growth in computation time
    - Introduces false negatives, fraction can be reduced with tuning

Key Ideas


Items
Jaccard Similarity
Shingling
MinHashing
Locality-Sensitive Hashing

Jaccard Similarity

Metric used to calculate how similar two sets are

Given sets X and Y, the Jaccard Similarity and Distance respectively are:

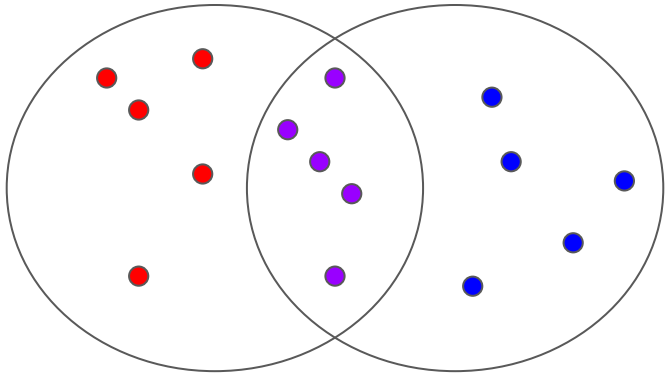
$$sim(X,Y) = |X \cap Y|/|X \cup Y|$$

$$d(X,Y) = 1 - sim(X,Y)$$

$|X \cap Y| = 5$

$|X \cup Y| = 15$

$sim(X,Y) = 1/3$



Shingling

- Shingling is the process of converting documents into sets
- A k-shingle is a sequence of k characters that appear in the document

Example:

Document = “This is AMS 598”, k = 2

Set = {Th, hi, is, s_, _i, _A, AM, MS, S_, _5, 59, 98}

- “_” denote a space in a string
- Notice elements “is” and “s_” only appear once as sets do no contain duplicates

Sets as a Boolean Matrix

- We can determine the similarities between sets by creating a matrix in which:
  - Rows are the universal set of elements
  - Columns each represent a set

	Column A	Column B
a	1	1
b	0	1
c	0	1
d	0	0

Revisiting Jaccard Similarity

- We can alternatively define Jaccard Similarity as sim(col A, col B) = a / (a + b + c)
  - Rows of type **a** indicate the element is a member of both sets
  - Rows of type **b** or **c** indicate the element is only a member of one set
  - Rows of type **d** indicate the element is a member of neither

	Column A	Column B
a	1	1
b	1	0
c	0	1
d	0	0

MinHashing

- Signatures take up much less space than sets, allowing for work to be done in main memory instead of with disk

Steps:

1. Permute the rows randomly
2. Define a minhash function h(column) = number of the first row in which the column has a 1 in permuted order
3. Repeat the permutation and minhash function creation upwards of 100 times
4. These create a signature which in turn can be used to create its own matrix

Signature Matrix

Permutation	Col A	Col B	Col C
3 5 3	1	0	1
1 2 5	1	0	0
2 3 1	0	1	0
5 4 2	0	1	0
4 1 4	1	0	0

	3	1	3
	1	3	5
	1	2	3

	A-C	A-B	B-C
Full matrix	1/3	0	0
Sig matrix	1/3	0	0

MinHashing Feasibility

Issue:

- Permutations become infeasible as the number of rows increase
- 1 billion row permutation takes ~ 4 GB of memory, 100 permutations take ~ 0.5 TB

Solution:

- Pick around 100 hash functions
- Create a placeholder for each hash function, i and column, C i.e. **sig(C)[i]**
- Initialize each **sig(C)[i]** to infinity
- Loop through each cell in the matrix.
    - For each 1 found, replace current with value of hash function if k(i) < **sig(C)[i]**

Locality-Sensitive Hashing (LSH)
Focus on pairs of signatures likely to be from similar documents

Basic Idea:

- two points that are very close (similar) in the original space, after being mapped by the LSH hash function, have a high probability that their hashes are the same;
- Similarly, two points that are not close(not similar), after mapping, the probability that their hash values are equal is very small.
- Use a function f(x,y) that tells whether x and y is a **candidate pair**: a pair of elements whose similarity must be evaluated

Locality-Sensitive Hashing (LSH)

Focus on pairs of signatures likely to be from similar documents

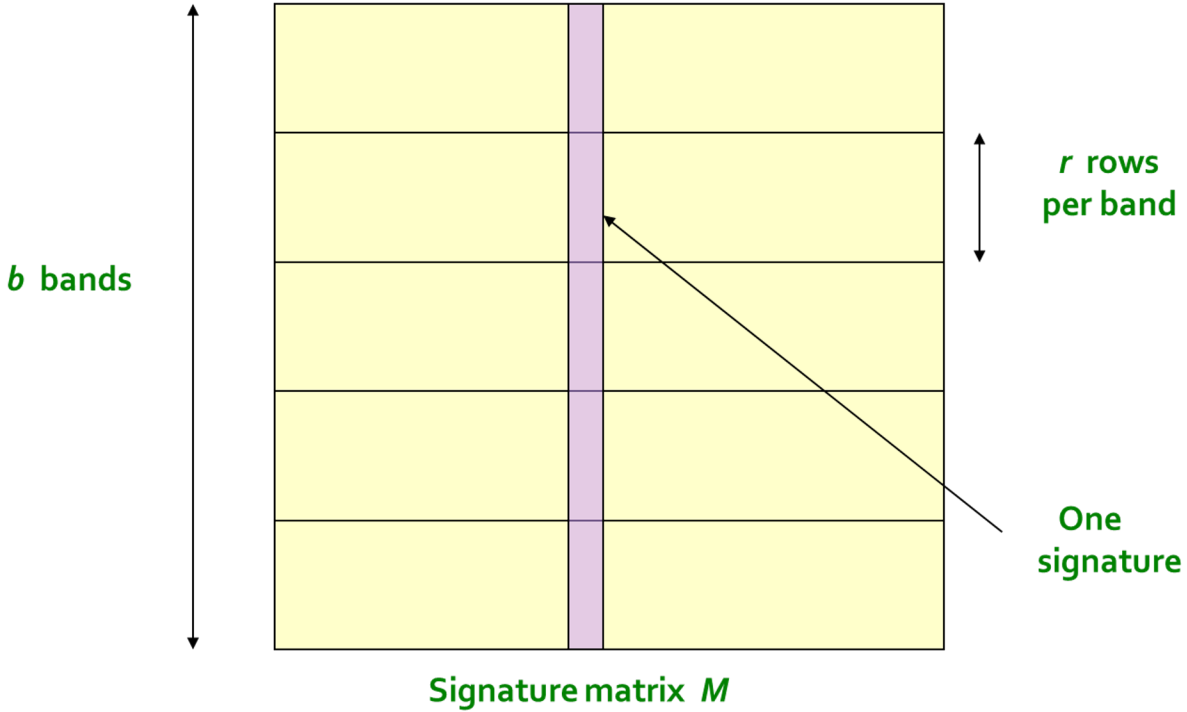
Previous Output: A signature matrix
Next goal: control the relationship between the similarity and the mapping probability

Step 1:

- Divide the signature matrix horizontally into some blocks (denoted as bands)
- each band contains the r rows in the signature matrix
- each band of the same column belongs to the same document
- similar columns are likely to hash to the same bucket, with high probability
- We call those that hash to the same bucket **Candidate pairs**

Locality-Sensitive Hashing (LSH)

Focus on pairs of signatures likely to be from similar documents



Locality-Sensitive Hashing (LSH)

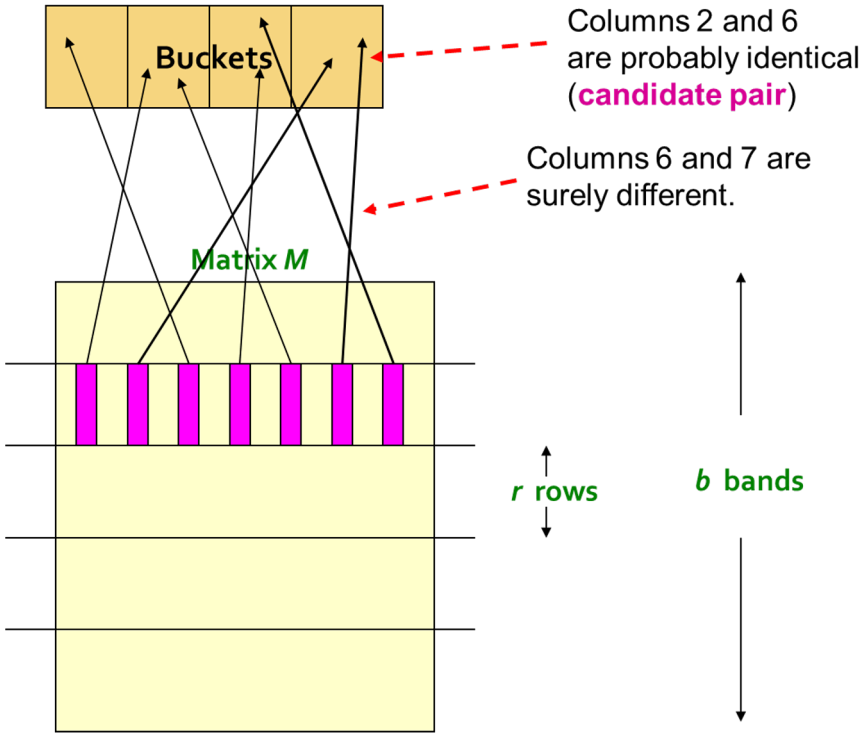
Focus on pairs of signatures likely to be from similar documents

Step 2:

- Calculate the hash value for each band. The hash algorithm here has no special requirements, MD5, SHA1, etc. can be used.
- In general, we need to process these hash values to make them the tags of the pre-set hash bucket, and then "throw" these bands into the hash bucket.
- Candidate column pairs are those that hash to the same bucket for ≥ 1 band
- For each band, hash its portion of each column to a hash table with k buckets
- Tune b and r to catch most similar pairs, but few non-similar pairs

Locality-Sensitive Hashing (LSH)

Focus on pairs of signatures likely to be from similar documents



Locality-Sensitive Hashing (LSH)
Focus on pairs of signatures likely to be from similar documents

Step 3:

- If the bands in the same horizontal direction of two documents are mapped to the same hash value, we will Map these two documents to the same hash bucket, which means that the two documents are similar enough.
- Consequently, by adjusting the parameters, the more similar, the easier it is to be in a hash bucket; the less similar, the less likely it is to be in a hash bucket.
- When the similarity is higher than a certain value, the probability will become very large and quickly approach 1, while when the similarity is lower than a certain value, the probability will become very small and quickly approach 0.

Locality-Sensitive Hashing (LSH)

Focus on pairs of signatures likely to be from similar documents

b bands, r rows:

- $s \in [0,1]$ is the similarity of the two columns (documents).
- For any band of two documents (r rows), the probability that the two band values are the same is: s^r
- In other words, the probability that the two bands are not the same is: $1-s^r$
- There are b bands in these two documents, and the probability that the b bands are all different is: $(1-s^r)^b$
- At least one of these b bands has the same probability is $1-(1-s^r)^b$

Locality-Sensitive Hashing (LSH)

Focus on pairs of signatures likely to be from similar documents

In this way, the probability that two documents are mapped to the same hash bucket can actually be controlled by controlling the value of the parameter.

Say, b=20, r=5:

When s = 0.8:

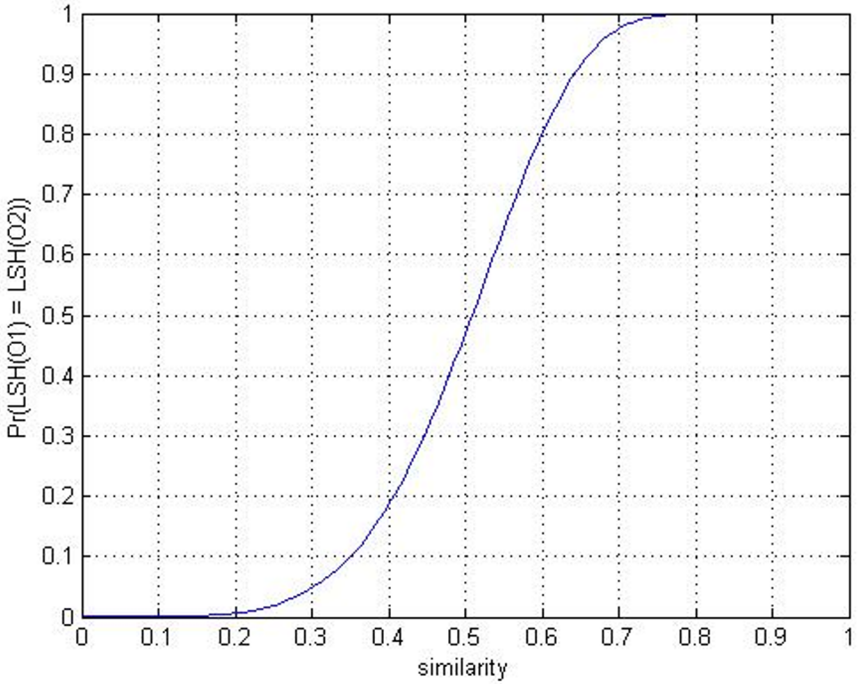
$$Pr(LSH(O_1) = LSH(O_2))=1-(1-0.8^5)^5=0.999643942109 4793$$

When s = 0.2:

$$Pr(LSH(O_1) = LSH(O_2))=1-(1-0.2^5)^5=0.0063805813047682$$

Locality-Sensitive Hashing (LSH)

Focus on pairs of signatures likely to be from similar documents



Example: Find Images With Similar Features

Example: Find Images With Similar Features

- **Introduction**
  - Use the algorithm introduced previously to find images with the most similar features
  - For instance, find the nearest neighbors from a collection of 2 million images

Example: Find Images With Similar Features

- **Dataset: Fashion Product Images Dataset**
  - Reference: https://www.kaggle.com/datasets/paramaggarwal/fashion-product-images-dataset
  - Description: large image dataset contains high resolution product images from e-commerce industry; a table indicate category of images
    - Image Resolution: 60 x 80
    - # of images: around 50000
  - In this project, we want to find all the similar images for a given picture
    - e.g. If the given picture is a [illegible] want [illegible] image [illegible] in a shoe



Example: Find Images With Similar Features

- **Preprocess images**

a imag
ge.ope
return
round(r
return
[0.73,
vert_in

bject
ze((30,
e (80, 6
1, 0, 1]

e value

  - convert colored pixels into black pixels
- to_2D_matrix(img)
  - return array has shape(80, 60) (e.g. [1, 0, 1] → 0 & [0, 0, 0] → 1)

Example: Find Images With Similar Features

- **Shingling**
  - Images that have lots of shingles in common have similar 5 by 5 pixels blocks, even if the blocks appears in different order
    - K = 5, 5 by 5 pixels block
      - Every time, move 1 pixel to obtain a new shingle
      - Image Resolution = 30 x 40, total shingles = 25 x 35 = 875
    - Shingles will be stored as a 5 x 5 numpy matrix first, and then it will be flatten into a numpy array containing 25 elements
  - Create Boolean Matrices
- **Min-Hashing**
- **Locality-Sensitive Hashing**

Example: Find Images With Similar Features

- **Parallel Computing in the Algorithm**
  - **Case 1: Find Images With Similar Features**
    - **Basic Idea**
      - Each processor takes a portion of the images
      - First column represents the given image
      - Return a list of names of similar images for a given picture
      - Send the list to root node
  - **Case 2: Group Images With Similar Features**
    - **Basic Idea**
      - Each processor takes a portion of the images
      - Return a list of tuples (image name, shingles)
      - Send the list to root node
      - Root node run the algorithm (Min-Hashing, LSH) to group images

Implementation with mpi4py

Min-Hashing

```
def min_hash(matrix):
    row = matrix.shape[0]
    col = matrix.shape[1]
    num = 1000
    perm = list(range(row))
    sig = np.zeros([num, col])
    for n in range(num):
        rd.shuffle(perm)
        for i in range(col):
            index = 1
            for j in perm:
                if matrix[j, i] == 1:
                    sig[n, i] = index
                    break
                else:
                    index += 1
    return sig
```

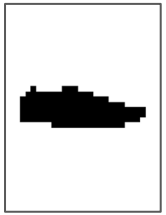
Locality-Sensitive Hashing

```
def LSH(sig, band):
    given = np.split(sig[:,0], band)
    candidates = []
    for i in range( sig.shape[1]):
        col = np.split(sig[:, i], band)
        for j in range(band):
            if np.array_equal(given[j], col[j]):
                candidates.append(i)
                break
    return candidates
```

Example: Find Images With Similar Features

- **Given Image**





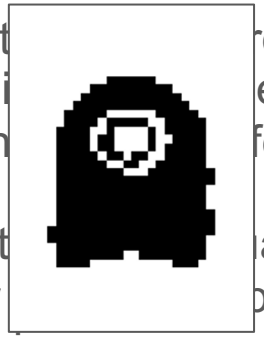
- **Images With Similar Features**
  - # of images found: 185
    - # of shoes images: 133
    - # of other images: 52
  - Accuracy: 133 / 185 = 72%

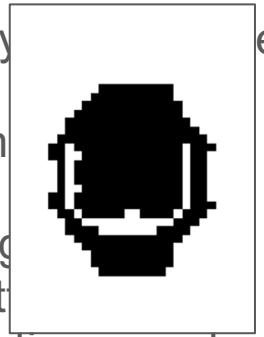



Example: Find Images With Similar Features

- **Deficiency**
  - np.r minat round may e colore . whi e ren e proc ns wh fecte rithm s, whi ease n of t ality puting differe may patter & bot
  - Algorithm may miss comparing some parts of images (permutations maybe not enough)
- **Improvement**
  - Consider colors in the algorithm
  - Increase the resolution of images
  - Use better hashing method (e.g. perceptual hash, usually used in fingerprint recognition)







Another Application: Find ‘Near Duplicate’ Documents

- **Shingling**
  - For example, K = 2; Document D1 = abcab
    - Set of 2-shingles: S(D1) = {ab, bc, ca}
  - Documents that have lots of shingles in common have similar text, even if the text appears in different order
- **Min-Hashing**
- **Locality-Sensitive Hashing**

Question?

Thank you!