

COMPSCI 514: Algorithms for Data Science

Cameron Musco

University of Massachusetts Amherst. Fall 2024.

Lecture 10

- Problem Set 2 is due next Friday 10/11 at 11:59pm.
- The midterm is 7-9pm on Thursday 10/27. Midterm study guide will be posted shortly.
- We have a quiz this week, but not next two weeks (due to the problem set and midterm).

Summary

Last Class:

- Analysis of idealized MinHash-based distinct elements algorithm.
- Median trick for boosting success probability.
- Discussion of practical algorithms for distinct elements (LogLog/HyperLogLog).

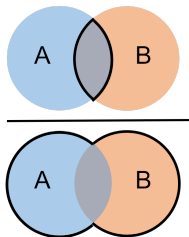
This Class:

- The similarity search problem.
- Locality sensitive hashing for fast similarity search.
- MinHash as a locality sensitive hash function for Jaccard similarity
- Balancing false positives and negatives with LSH signatures and repeated hash tables.

Another Fundamental Problem

Jaccard Index: A similarity measure between two sets.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{\# \text{ shared elements}}{\# \text{ total elements}}.$$



Natural measure for similarity between bit strings – interpret an n bit string as a set, containing the elements corresponding the positions of its ones. $J(x, y) = \frac{\# \text{ shared ones}}{\# \text{ total ones}}.$

Search with Jaccard Similarity

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{\# \text{ shared elements}}{\# \text{ total elements}}.$$

Want Fast Implementations For:

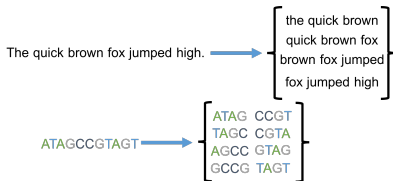
- **Near Neighbor Search:** Have a database of n sets/bit strings and given a set A , want to find if it has high Jaccard similarity to anything in the database. $\Omega(n)$ time with a linear scan.
- **All-pairs Similarity Search:** Have n different sets/bit strings and want to find all pairs with high Jaccard similarity. $\Omega(n^2)$ time if we check all pairs explicitly.

Will speed up via randomized **locality sensitive hashing**.

Application: Document Similarity

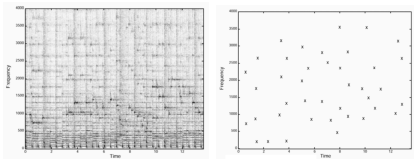
Document Similarity:

- E.g., to detect plagiarism, copyright infringement, duplicate webpages, spam.
- Use Shingling + Jaccard similarity. (n -grams, k -mers)



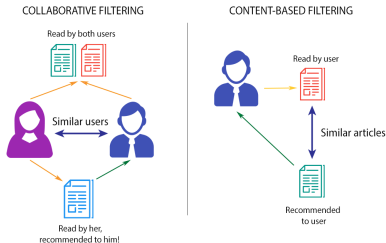
Audio Fingerprinting:

- E.g., in audio search (Shazam), Earthquake detection.
- Represent sound clip via a binary 'fingerprint' then compare with Jaccard similarity.



Application: Collaborative Filtering

Online recommendation systems are often based on **collaborative filtering**. Simplest approach: find similar users and make recommendations based on those users.

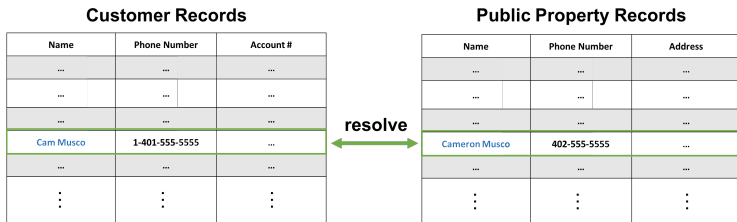


- Twitter: represent a user as the set of accounts they follow. Match users based on the Jaccard similarity of these sets. Recommend that you follow accounts followed by similar users.
- Netflix: look at sets of movies watched. Amazon: look at products purchased, etc.

Application: Entity Resolution

Entity Resolution Problem: Want to combine records from multiple data sources that refer to the same entities.

- E.g. data on individuals from voting registrations, property records, and social media accounts. Names and addresses may not exactly match, due to typos, nicknames, moves, etc.
- Still want to match records that all refer to the same person using all pairs similarity search.



See Section 3.8.2 of *Mining Massive Datasets* for a discussion of a real world example involving 1 million customers. Naively this would

Application: Spam and Fraud Detection

Many applications to spam/fraud detection. E.g.

- **Fake Reviews:** Very common on websites like Amazon. Detection often looks for (near) duplicate reviews on similar products, which have been copied. 'Near duplicate' measured with shingles + Jaccard similarity.
- **Lateral phishing:** Phishing emails sent to addresses at a business coming from a legitimate email address at the same business that has been compromised.
 - One method of detection looks at the recipient list of an email and checks if it has small Jaccard similarity with any previous recipient lists. If not, the email is flagged as possible spam.

Search with Jaccard Similarity

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{\# \text{ shared elements}}{\# \text{ total elements}}.$$

Want Fast Implementations For:

- **Near Neighbor Search:** Have a database of n sets/bit strings and given a set A , want to find if it has high Jaccard similarity to anything in the database. $\Omega(n)$ time with a linear scan.
- **All-pairs Similarity Search:** Have n different sets/bit strings and want to find all pairs with high Jaccard similarity. $\Omega(n^2)$ time if we check all pairs explicitly.

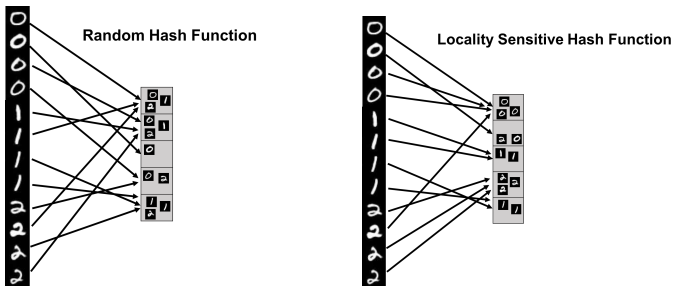
Will speed up via randomized **locality sensitive hashing**.

Locality Sensitive Hashing

Goal: Speed up Jaccard similarity search (near neighbor and all-pairs similarity search).

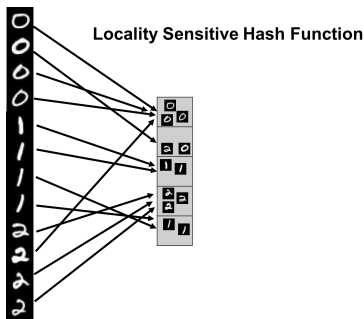
Strategy: Locality sensitive hashing (LSH).

- Design a hash function where the collision probability is higher when two inputs are more similar (can design different functions for different similarity metrics.)



LSH For Similarity Search

How does locality sensitive hashing (LSH) help with similarity search?



- **Near Neighbor Search:** Given item x , compute $h(x)$. Only search for similar items in the $h(x)$ bucket of the hash table.
- **All-pairs Similarity Search:** Scan through all buckets of the hash table and look for similar pairs within each bucket.
- We will use $h(x) = g(\text{MinHash}(x))$ where $g : [0, 1] \rightarrow [n]$ is a random hash function. **Why?**

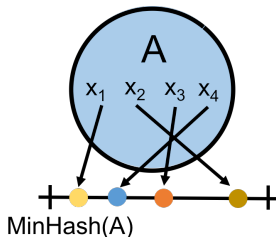
MinHashing

An Example: Locality sensitive hashing for Jaccard similarity.

Strategy: Use random hashing to map each set to a single hash value. The probability that two sets have colliding hash values will be proportional to their Jaccard similarity.

MinHash(A): [Andrei Broder, 1997 at Altavista]

- Let $h : U \rightarrow [0, 1]$ be a random hash function
- $s := 1$
- For $x_1, \dots, x_{|A|} \in A$
 - $s := \min(s, h(x_k))$
- Return s



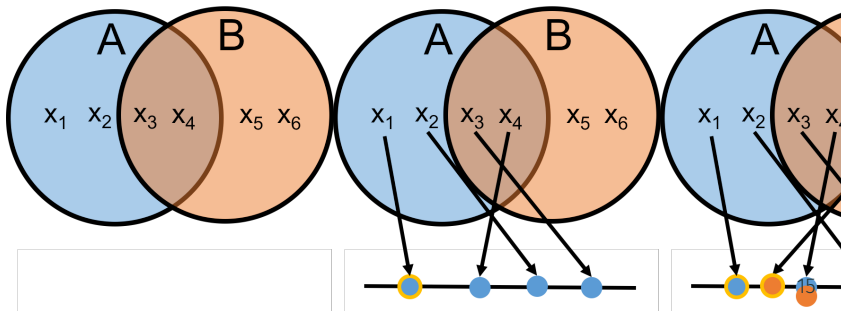
Identical to our distinct elements sketch!

MinHash Analysis

For two sets A and B , what is $\Pr(\text{MinHash}(A) = \text{MinHash}(B))$?

$$\Pr\left(\min_{x \in A} h(x) = \min_{y \in B} h(y)\right) = ?$$

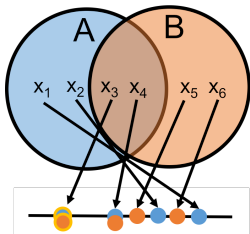
- Since we are hashing into the continuous range $[0, 1]$, we will never have $h(x) = h(y)$ for $x \neq y$ (i.e., no spurious collisions)



MinHash Analysis

For two sets A and B , what is $\Pr(\text{MinHash}(A) = \text{MinHash}(B))$?

Claim: $\text{MinHash}(A) = \text{MinHash}(B)$ only if an item in $A \cap B$ has the minimum hash value in both sets.



$$\begin{aligned}\Pr(\text{MinHash}(A) = \text{MinHash}(B)) &= ? \frac{|A \cap B|}{\text{total \# items hashed}} \\ &= \frac{|A \cap B|}{|A \cup B|} = J(A, B).\end{aligned}$$

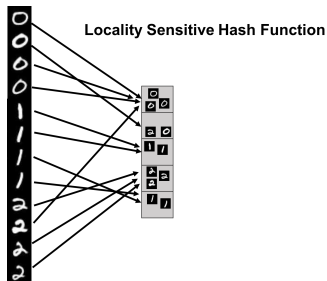
Locality sensitive: the higher $J(A, B)$ is, the more likely $\text{MinHash}(A), \text{MinHash}(B)$ are to collide.

Similarity Search with MinHash

Goal: Given a document y , identify all documents x in a database with Jaccard similarity (of their shingle sets) $J(x,y) \geq 1/2$.

Our Approach:

- Create a hash table of size m , choose a random hash function $g : [0, 1] \rightarrow [m]$, and insert every item x into bucket $g(\text{MinHash}(x))$. Search for items similar to y in bucket $g(\text{MinHash}(y))$.



Reducing False Negatives

With a simple use of MinHash, we miss a match x with $J(x, y) = 1/2$ with probability $1/2$. **How can we reduce this false negative rate?**

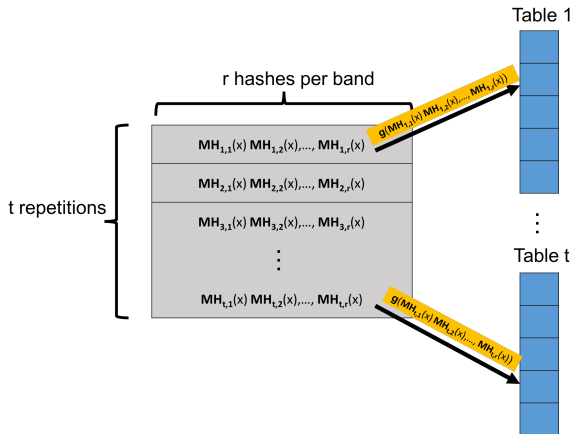
Repetition: Run MinHash t times independently, to produce hash values $MH_1(x), \dots, MH_t(x)$. Apply random hash function \mathbf{g} to map all these values to locations in t hash tables.

- To search for items similar to y , look at all items in bucket $\mathbf{g}(MH_1(y))$ of the 1st table, bucket $\mathbf{g}(MH_2(y))$ of the 2nd table, etc.
- **What is the probability that x with $J(x, y) = 1/2$ is in at least one of these buckets, assuming for simplicity \mathbf{g} has no collisions?**
 $1 - (\text{probability in no buckets}) = 1 - \left(\frac{1}{2}\right)^t \approx .99$ for $t = 7$.
- **What is the probability that x with $J(x, y) = 1/4$ is in at least one of these buckets, assuming for simplicity \mathbf{g} has no collisions?**
 $1 - (\text{probability in no buckets}) = 1 - \left(\frac{3}{4}\right)^t \approx .87$ for $t = 7$.

Potential for a lot of false positives! Slows down search time.

Balancing Hit Rate and Query Time

We want to balance a small probability of false negatives (a high hit rate) with a small probability of false positives (a small query time.)



Create t hash tables. Each is indexed into not with a single MinHash value, but with r values, appended together. A length r signature.

Balancing Hit Rate and Query Time

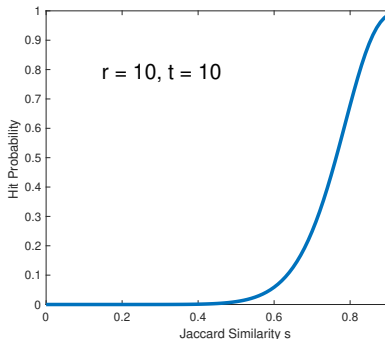
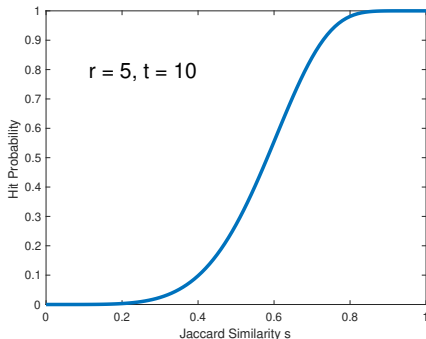
Consider searching for matches in t hash tables, using MinHash signatures of length r . For x and y with Jaccard similarity $J(x, y) = s$:

- Probability that a single hash matches.
 $\Pr [MH_{i,j}(x) = MH_{i,j}(y)] = J(x, y) = s$.
- Probability that x and y having matching signatures in repetition i . $\Pr [MH_{i,1}(x), \dots, MH_{i,r}(x) = MH_{i,1}(y), \dots, MH_{i,r}(y)] = s^r$.
- Probability that x and y don't match in repetition i : $1 - s^r$.
- Probability that x and y don't match in *all repetitions*: $(1 - s^r)^t$.
- Probability that x and y match in at least one repetition:

Hit Probability: $1 - (1 - s^r)^t$.

The s-curve

Using t repetitions each with a signature of r MinHash values, the probability that x and y with Jaccard similarity $J(x, y) = s$ match in at least one repetition is: $1 - (1 - s^r)^t$.



r and t are tuned depending on application. 'Threshold' when hit probability is $1/2$ is $\approx (1/t)^{1/r}$. E.g., $\approx (1/30)^{1/5} = .51$ in this case.

s-curve Example

For example: Consider a database with 10,000,000 audio clips. You are given a clip x and want to find any y in the database with $J(x, y) \geq .9$.

- There are 10 **true matches** in the database with $J(x, y) \geq .9$.
- There are 10,000 **near matches** with $J(x, y) \in [.7, .9]$.

With signature length $r = 25$ and repetitions $t = 50$, hit probability for $J(x, y) = s$ is $1 - (1 - s^{25})^{50}$.

- Hit probability for $J(x, y) \geq .9$ is $\geq 1 - (1 - .9^{25})^{50} \approx .98$
- Hit probability for $J(x, y) \in [.7, .9]$ is $\leq 1 - (1 - .9^{25})^{50} \approx .98$
- Hit probability for $J(x, y) \leq .7$ is $\leq 1 - (1 - .7^{25})^{50} \approx .007$

Expected Number of Items Scanned: (proportional to query time)

$$\leq 10 + .98 * 10,000 + .007 * 9,989,990 \approx 80,000 \ll 10,000,000.$$