

# COMPSCI 514: ALGORITHMS FOR DATA SCIENCE

---

Cameron Musco

University of Massachusetts Amherst. Fall 2019.

Lecture 6

- Problem Set 1 is due next Thursday 9/26 in Gradescope.
- My office hours next week will be Tuesday, 1:15-2:15.

## Last Class:

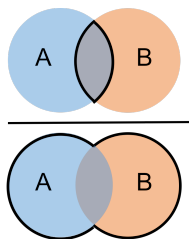
- **Distinct Elements via Hashing:**
  - Distinct elements algorithm using min-of-hashes approach.
  - Analysis using averaging and the 'median trick'.
  - Practical implementations (HyperLogLog) and applications.
- **Jaccard Similarity:**
  - Jaccard similarity as a similarity metric between sets and binary strings. Applications in document comparison and audio fingerprinting.

## This Class:

- See how a min-of-hashes approach (MinHash) is used to estimate the Jaccard similarity.
- Application of MinHash to fast similarity search.
- Locality sensitive hashing.

**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.  $J(x, y) = \frac{\# \text{ shared ones}}{\text{total ones}}.$

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

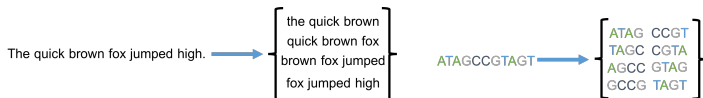
### Two Common Use Cases:

- **Near Neighbor Search:** Have a database of  $n$  sets/bit strings and given a set  $A$ , want to find if it has high similarity to anything in the database.  $O(n \cdot \text{average set size})$  time.
- **All-pairs Similarity Search:** Have  $n$  different sets/bit strings and want to find all pairs with high similarity.  $O(n^2 \cdot \text{average set size})$  time.

Will speed up significantly via hashing.

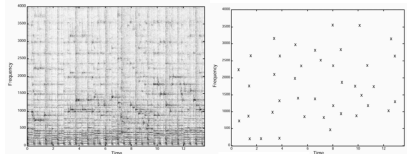
## Document Similarity:

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



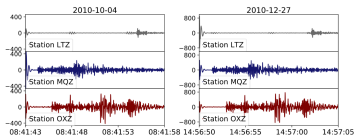
## Audio Fingerprinting:

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

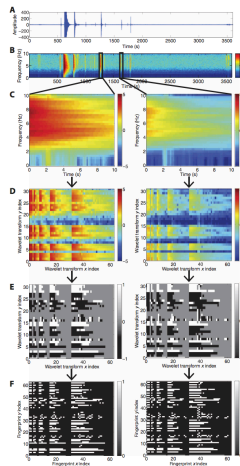


# APPLICATION: EARTHQUAKE DETECTION

Small earthquakes make consistent signatures on seismographs that repeat over time. Detecting repeated signatures lets you detect these otherwise undetectable events.

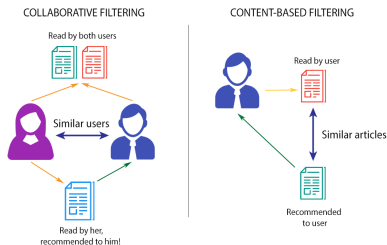


- Split data into overlapping windows of 10 seconds
- Fingerprint each window using the spectrogram (i.e., compute a binary string representing the reading in the window).
- All-pairs search for windows with high Jaccard similarity.



## 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 similar 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.



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.

## WHY JACCARD SIMILARITY?

Why use Jaccard similarity over other metrics like: Hamming distinct (bit strings), correlation (sound waves, seismograms), edit distance (text, genome sequences, etc.)?

### **Two Reasons:**

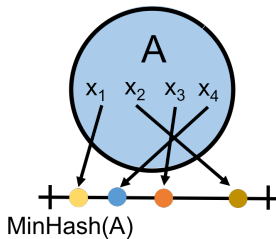
- Depending on the application, often is the right measure.
- Even when not ideal, very efficient to compute and implement near neighbor search and all-pairs similarity search with.

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

**Strategy:** Use random hashing to map each set to a very compressed representation. Jaccard similarity can be estimated from these representations.

**MinHash(A):**

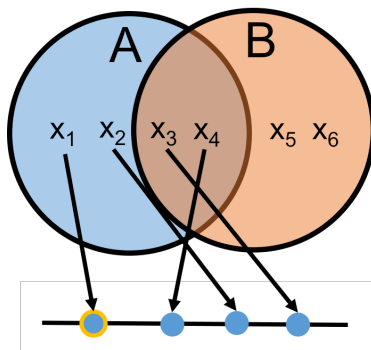
- Let  $h : U \rightarrow [0, 1]$  be a random hash function
- $s := 1$
- For  $x_1, \dots, x_{|A|} \in A$ 
  - $s := \min(s, h(x_k))$
- Return  $s$



Identical to our distinct elements sketch!

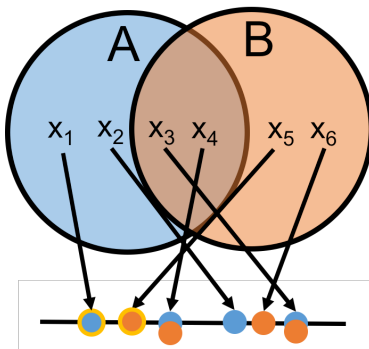
For two sets  $A$  and  $B$ , what is  $\Pr(\text{MinHash}(A) = \text{MinHash}(B))$ ?

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



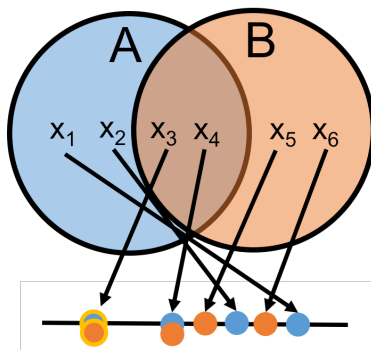
For two sets  $A$  and  $B$ , what is  $\Pr(\text{MinHash}(A) = \text{MinHash}(B))$ ?

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



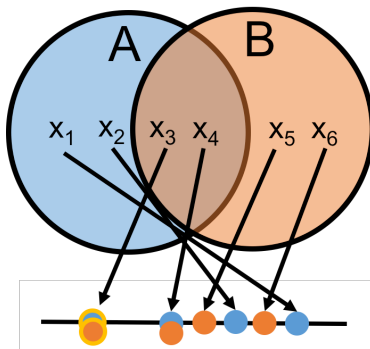
For two sets  $A$  and  $B$ , what is  $\Pr(\text{MinHash}(A) = \text{MinHash}(B))$ ?

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



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

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

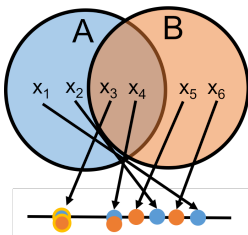


- $\text{MinHash}(A) = \text{MinHash}(B)$  only if an item in  $A \cap B$  has the minimum hash value in both sets.

# MINHASH

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.



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

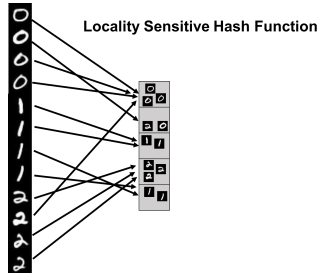
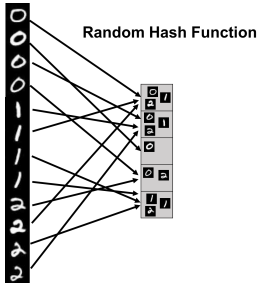


# LOCALITY SENSITIVE HASHING

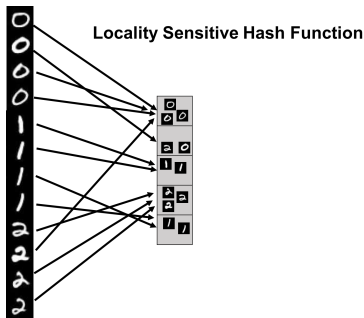
**Upshot:** MinHash reduces estimating the Jaccard similarity to checking equality of a *single number*.

$$\Pr(\text{MinHash}(A) = \text{MinHash}(B)) = J(A, B).$$

- An instance of **locality sensitive hashing** (LSH).
- A hash function where the collision probability is higher when two inputs are more similar (can design different functions for different similarity metrics.)



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 only look for similar pairs within each bucket.

**Goal:** Use MinHash to perform Jaccard similarity search of documents efficiently.

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

### Simple 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  $g(\text{MinHash}(y))$ .
- For every document  $x$  in your database with  $J(x, y) = 1/2$  what is the probability you will find  $x$  in bucket  $g(\text{MinHash}(y))$ , assuming  $g$  is collision free?  $1/2$ .

## 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?**

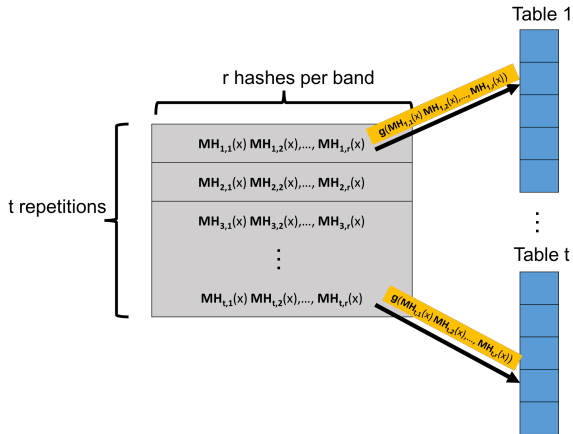
**Repetition:** Run MinHash  $t$  times independently, to produce hash values  $MH_1(x), \dots, MH_t(x)$ . Apply random hash function  $\mathbf{g}$  to map all these values to locations in  $t$  hash tables.

- To search for items similar to  $y$ , look at all items in bucket  $\mathbf{g}(MH_1(y))$  of the 1<sup>st</sup> table, bucket  $\mathbf{g}(MH_2(y))$  of the 2<sup>nd</sup> 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**.

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  $i$ .

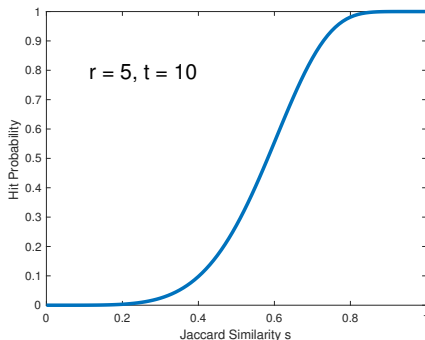
$$\Pr [MH_{i,1}(x), \dots, MH_{i,r}(x) = MH_{i,1}(y), \dots, MH_{i,r}(y)] = s^r.$$

- 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:

$$\text{Hit Probability: } 1 - (1 - s^r)^t.$$

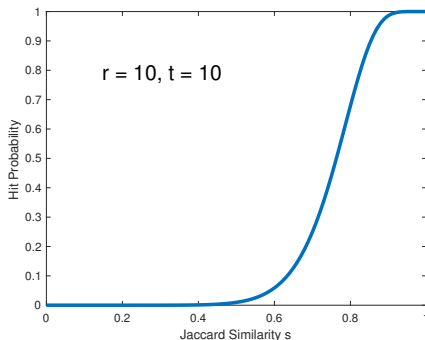
## THE S-CURVE

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



## THE S-CURVE

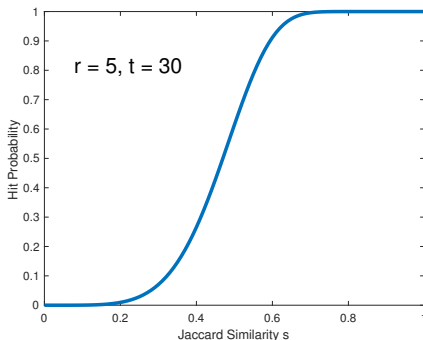
Using  $t$  repetitions each with a signature of  $r$  MinHash values, the probability that  $x$  and  $y$  with Jaccard similarity  $J(x, y) = s$  match in at least one repetition is:  $1 - (1 - s^r)^t$ .





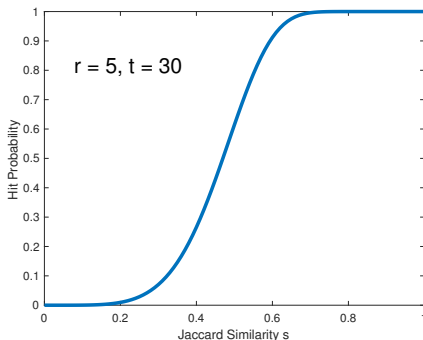
## THE S-CURVE

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



## 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)

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

# HASHING FOR DUPLICATE DETECTION

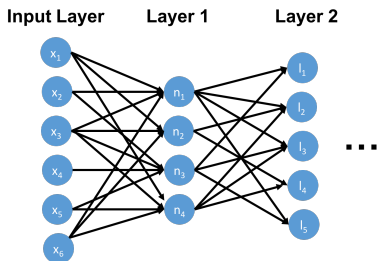
	<b>Bloom Filters</b>	<b>Hash Table</b>	<b>MinHash</b>	<b>Distinct Elements</b>
<b>Goal</b>	Check if x is a duplicate of y in database.	Check if x is a duplicate of any y in database and return y.	Check if x is a duplicate of any y in database and return y.	Count # of items, excluding duplicates.
<b>Approximate Duplicates?</b>	✘	✘	✔	✘

All different variants of detecting duplicates/finding matches in large datasets. This is an important problem in many contexts!

Locality sensitive hashing (LSH) schemes have been developed for many similarity/distance measures: hamming distance (bit sampling), cosine similarity (random projections), etc.

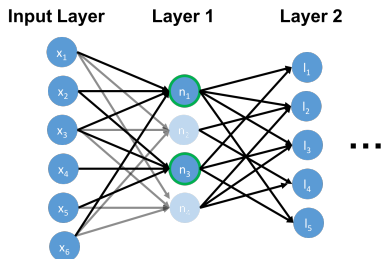
- Typically used for fast near neighbor search and all-pairs similarity search.
- Anshumali Shrivastava at Rice has proposed a huge number of other interesting applications – from speeding up neural net evaluation, to sampling points with large gradients in stochastic gradient descent.

## SPEEDING UP NEURAL NETWORKS



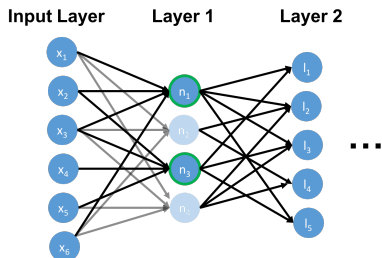
- Evaluating the output for input  $x$  requires  $|x| \cdot |\text{layer 1}| + |\text{layer 1}| \cdot |\text{layer 2}| + \dots$  multiplications if fully connected. Can be expensive, especially on constrained devices like cellphones, cameras, etc.

# SPEEDING UP NEURAL NETWORKS



- Evaluating the output for input  $x$  requires  $|x| \cdot |\text{layer 1}| + |\text{layer 1}| \cdot |\text{layer 2}| + \dots$  multiplications if fully connected. Can be expensive, especially on constrained devices like cellphones, cameras, etc.
- For approximate evaluation, suffices to identify the neurons in each layer with the **highest activation** when  $x$  is presented.

# SPEEDING UP NEURAL NETWORKS



- Evaluating the output for input  $x$  requires  $|x| \cdot |\text{layer 1}| + |\text{layer 1}| \cdot |\text{layer 2}| + \dots$  multiplications if fully connected. Can be expensive, especially on constrained devices like cellphones, cameras, etc.
- For approximate evaluation, suffices to identify the neurons in each layer with the **highest activation** when  $x$  is presented.
- That is,  $n_j$  where  $\sum_{i=1}^6 x_i \cdot w(x_i, n_j) = \langle x, w(n_j) \rangle$  is large. **Can be identified rapidly using LSH for cosine similarity!**



Questions?