COMPSCI 514: ALGORITHMS FOR DATA SCIENCE

Cameron Musco
University of Massachusetts Amherst. Fall 2021.
Lecture 9

SUMMARY

Last Class:

- Finish up distinct elements counting. High-level overview of the HyperLogLog algorithm.
- Introduction of Jaccard similarity and the similarity search problem.

This Class:

- Locality sensitive hashing and fast similarity search.
- MinHashing for Jaccard similarity search.

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

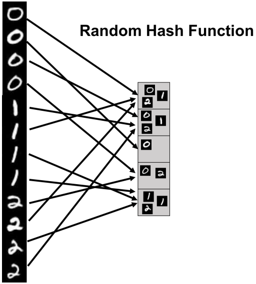
Want Fast Implementations For:

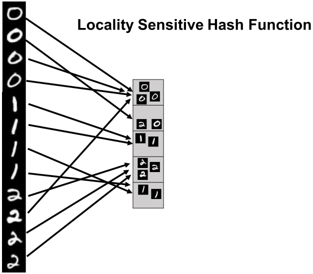
- **Near Neighbor Search:** Have a database of n sets/bit strings and given a set A, want to find if it has high Jaccard similarity to anything in the database. $\Omega(n)$ time with a linear scan.
- **All-pairs Similarity Search:** Have n different sets/bit strings and want to find all pairs with high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

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

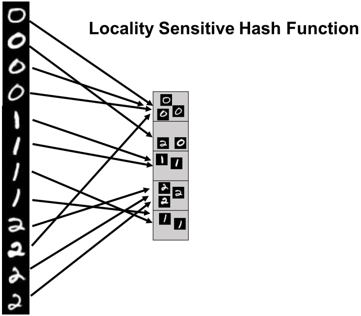
Strategy: Locality sensitive hashing (LSH).

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





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

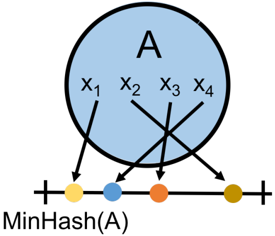


- **Near Neighbor Search:** Given item x, compute $h(x)$. Only search for similar items in the $h(x)$ bucket of the hash table.
- **All-pairs Similarity Search:** Scan through all buckets of the hash table and look for similar pairs within each bucket.

An Example: Locality sensitive hashing for Jaccard similarity.

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

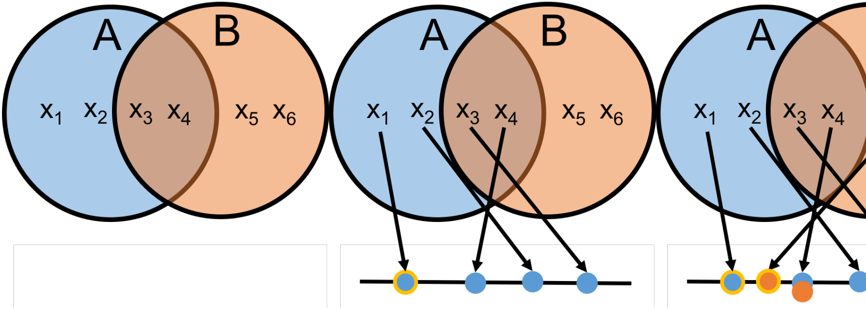
- Let $\mathbf{h} : U \to [0, 1]$ be a random hash function
- $\mathbf{s} := 1$
- For $x_1, \ldots, x_{|A|} \in A$
  - $\mathbf{s} := \min(\mathbf{s}, \mathbf{h}(x_k))$
- Return $\mathbf{s}$



Identical to our distinct elements sketch!

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

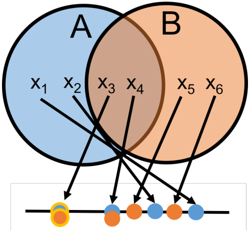
- Since we are hashing into the continuous range $[0, 1]$, we will never have $\mathbf{h}(x) = \mathbf{h}(y)$ for $x \neq y$ (i.e., no spurious collisions)



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

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

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



$$\Pr(MinHash(A) = MinHash(B)) = ? \frac{|A \cap B|}{\text{total \# items hashed}}$$

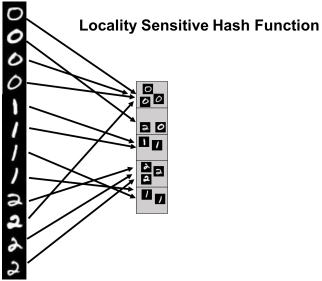
$$= \frac{|A \cap B|}{|A \cup B|} = J(A, B).$$

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

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 $\mathbf{g} : [0, 1] \rightarrow [m]$, and insert every item x into bucket $\mathbf{g}(MinHash(x))$. Search for items similar to y in bucket $\mathbf{g}(MinHash(y))$.



- What is $\Pr\left[\mathbf{g}(MinHash(x)) = \mathbf{g}(MinHash(y))\right]$ assuming $J(x, y) = 1/2$

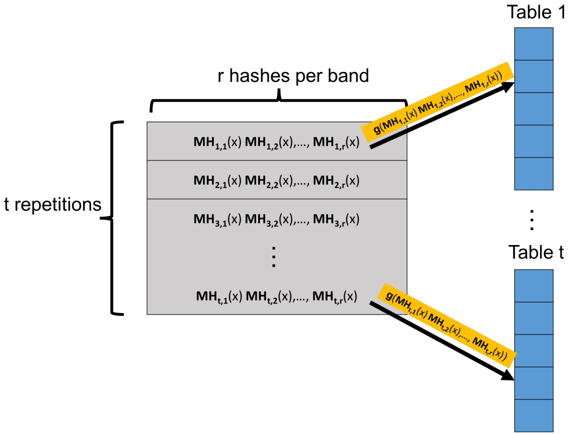
With a simple use of MinHash, we miss a match x with $J(x, y) = 1/2$ with probability 1/2. How can we reduce this false negative rate?

Repetition: Run MinHash t times independently, to produce hash values $MH_1(x), \ldots, MH_t(x)$. 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^{st} table, bucket $\mathbf{g}(MH_2(y))$ of the 2^{nd} 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?
  $1-$ (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?
  $1-$ (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.

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.

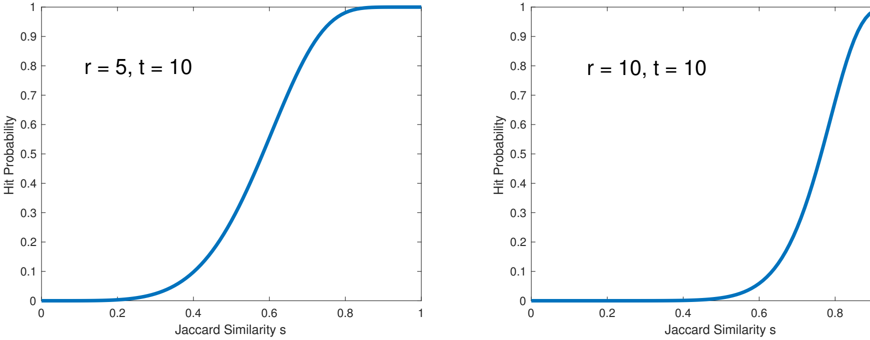
BALANCING HIT RATE AND QUERY TIME

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.
  $\Pr\left[MH_{i,j}(x) = MH_{i,j}(y)\right] = J(x, y) = s$.
- Probability that x and y having matching signatures in repetition i.
  $\Pr\left[MH_{i,1}(x), \ldots, MH_{i,r}(x) = MH_{i,1}(y), \ldots, MH_{i,r}(y)\right] = s^r$.
- 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:

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

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.

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)

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

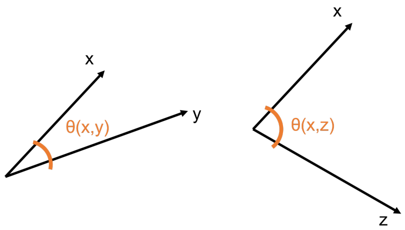
HASHING FOR DUPLICATE DETECTION

	Hash Table	Bloom Filters	MinHash Similarity Search	Distinct Elements
Goal	Check if x is a duplicate of any y in database and return y.	Check if x is a duplicate of y in database.	Check if x is a duplicate of any y in database and return y.	Count # of items, excluding duplicates.
Space	$O(n)$ items	$O(n)$ bits	$O(n \cdot t)$ items (when t tables used)	$O\left(\frac{\log \log n}{\epsilon^2}\right)$
Query Time	$O(1)$	$O(1)$	Potentially o(n)	NA
Approximate Duplicates?	✗	✗	✓	✗

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

Repetition and s-curve tuning can be used for fast similarity search with any similarity metric, given a locality sensitive hash function for that metric.

- LSH schemes exist for many similarity/distance measures: hamming distance, cosine similarity, etc.



Cosine Similarity: $\cos(\theta(x,y)) = \frac{\langle x,y\rangle}{\|x\|_2 \cdot \|y\|_2}$.

- $\cos(\theta(x,y)) = 1$ when $\theta(x,y) = 0°$ and $\cos(\theta(x,y)) = 0$ when $\theta(x,y) = 90°$, and $\cos(\theta(x,y)) = -1$ when $\theta(x,y) = 180°$