COMPSCI 514: Algorithms for Data Science

Cameron Musco

University of Massachusetts Amherst. Spring 2026.

Lecture 9

Logistics

- Problem Set 2 is due Sunday 3/8 at 11:59pm.
- The midterm is next Thursday, 3/12 in class.
- No quiz this week – focus on the problem set and studying.
- At least part of Tuesday's lecture will be midterm review. Additional midterm review at office hours, Tuesday 2:30pm and Wednesday 11am.
- Material from today may be on the midterm.

Summary

Last Class:

- Distinct elements counting in streams via MinHashing.
- Track minimum hash value of incoming items and use it as a proxy for the number of distinct items.
- Boosting success probability with the median trick.

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.

Fast Similarity Search

Have a database of items – e.g., documents, images, audio clips, etc. Often they have transformed into 'embeddings' – i.e., representations as vectors or sets.

Define a similarity metric over these items – e.g., cosine similarity (dot product) if they are represented as vectors, or Jaccard similarity if represented as sets.

Want Fast Implementations For:

- **Near Neighbor Search:** Given a query item q, find if it has high similarity to any database item. $\Omega(n)$ time with a linear scan.
- **All-pairs Similarity Search:** Have n different query vectors and want to find all pairs with high similarity. $\Omega(n^2)$ time if we check all pairs explicitly.

Difficulty is that q almost never has an exact match in the database – if it did we could solve search in $O(1)$ time using a hash table.

Applications of Fast Similarity Search

Huge number of applications including:

- Reverse image search (e.g., Google)
- Audio search (e.g., Shazam)
- Approximate document matching (e.g., for plagarism detection)
- Retrieval augmented generation (RAG) for LLMs.

Often implemented by 'vector databases' like:

Pinecone Weaviate Chroma Pgvector

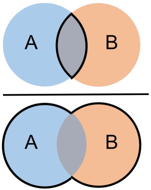
This area has blown up in the last year or two due to the importance of RAG in particular.

Jaccard Similarity

The most popular similarity metrics in practice are cosine similarity or ℓ_2 distance between real-valued vectors.

Today we will focus on Jaccard simlarity between sets. Many of the ideas carry over to similarity search in other metrics.

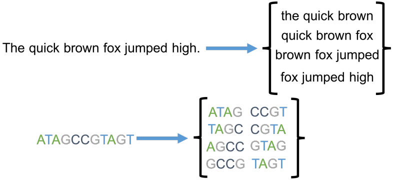
$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{\#\text{ shared elements}}{\#\text{ total elements}}.$$



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

Document Similarity:

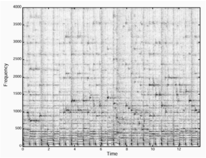
- E.g., to detect plagiarism, copyright infringement, duplicate webpages, spam.
- Use Shingling + Jaccard similarity. (n-grams, k-mers)

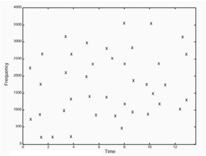


Application: Audio Search

Audio Fingerprinting:

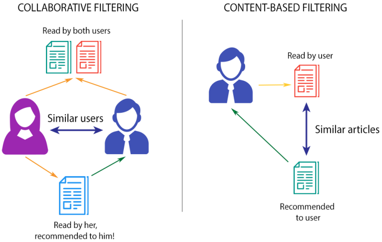
- E.g., in audio search (Shazam), Earthquake detection.
- Represent sound clip via a binary ‘fingerprint’ then compare with Jaccard similarity.





Application: Collaborative Filtering

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



- E.g., 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.

Customer Records

Name	Phone Number	Account #
...	...	...
...	...	...
...	...	...
Cam Musco	1-401-555-5555	...
...	...	...
⋮	⋮	⋮

resolve ⟷

Public Property Records

Name	Phone Number	Address
...	...	...
...	...	...
...	...	...
Cameron Musco	402-555-5555	...
...	...	...
⋮	⋮	⋮

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

Fast Similarity Search

Recall: Our goal is to design much faster algorithms for the following problems:

- **Near Neighbor Search:** Given a query item q, find if it has high similarity to any database item. $\Omega(n)$ time with a linear scan.
- **All-pairs Similarity Search:** Have n different query vectors and want to find all pairs with high similarity. $\Omega(n^2)$ time if we check all pairs explicitly.

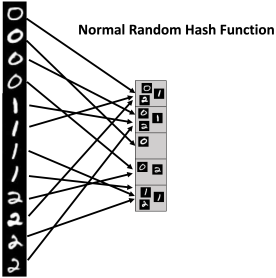
Many interesting approaches: locality sensitive hashing (LSH), graph based methods (HNSW, DiskANN, Vamana), clustering/tree based methods, product quantization, etc.

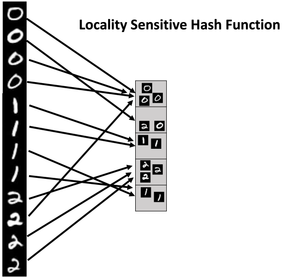
We will focus on LSH today – currently has the strongest theoretical foundations.

Locality Sensitive Hashing

Locality sensitive hashing (LSH) Strategy:

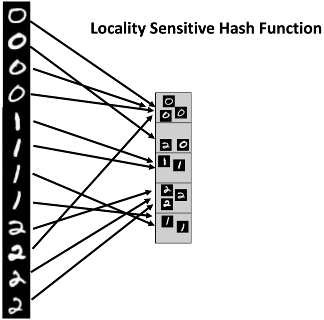
- Design a hash function where the collision probability is higher when two inputs are more similar (can design different functions for different similarity metrics.)





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.

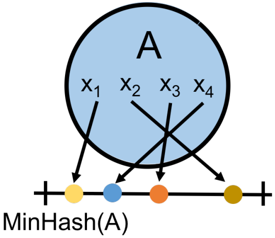
MinHashing

An Example: Locality sensitive hashing for Jaccard similarity.

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

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

- Let $\mathbf{h} : U \rightarrow [0,1]$ 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 $\mathbf{s}$



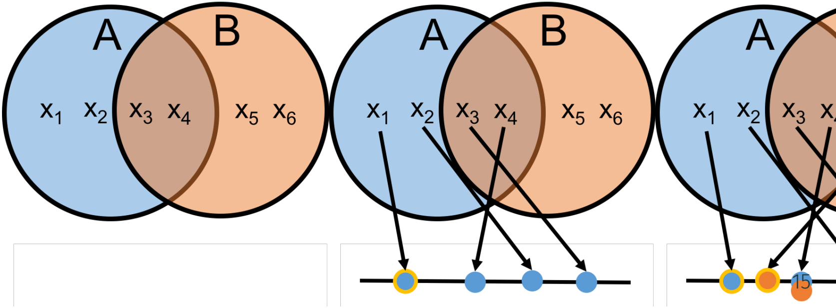
Identical to our distinct elements sketch!

MinHash Analysis

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

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

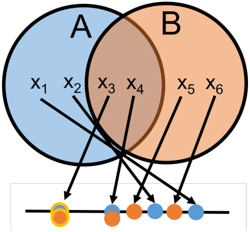
- Since we are hashing into the continuous range $[0,1]$, we will never have $\mathbf{h}(x) = \mathbf{h}(y)$ for $x \neq y$ (i.e., no spurious collisions)



MinHash Analysis

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.



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

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

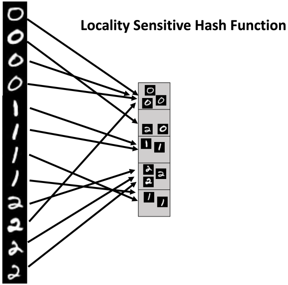
Locality sensitive: the higher $J(A, B)$ is, the more likely $MinHash(A), MinHash(B)$ are to collide.

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



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?

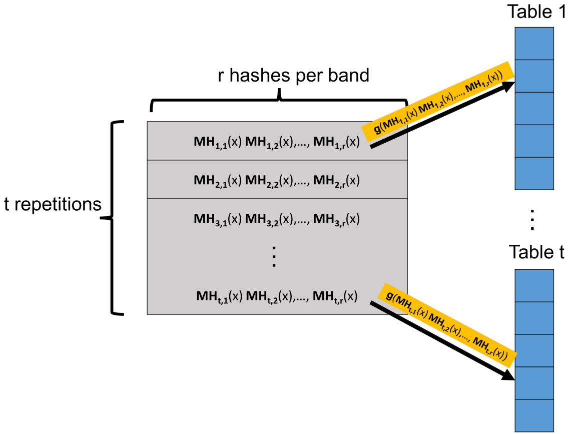
Repetition: Run MinHash t times independently, to produce hash values $MH_1(x), \ldots, MH_t(x)$. Apply random hash function $\mathbf{g}$ to map all these values to locations in t hash tables.

- 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.

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

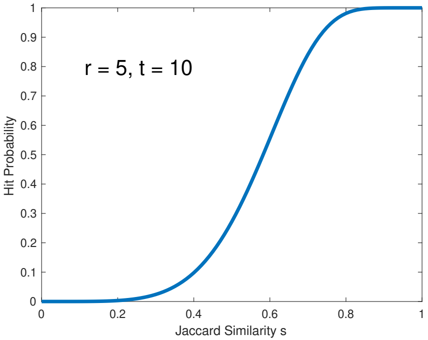
Consider searching for matches in t hash tables, using MinHash signatures of length r. For x and y with Jaccard similarity $J(x, y) = s$:

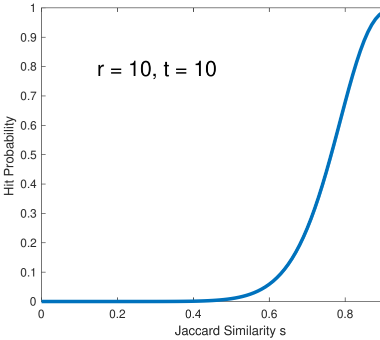
- Probability that a single hash matches. $\Pr\left[MH_{i,j}(x) = MH_{i,j}(y)\right] = J(x, y) = s$.
- Probability that x and y having matching signatures in repetition i. $\Pr\left[MH_{i,1}(x), \ldots, MH_{i,r}(x) = MH_{i,1}(y), \ldots, MH_{i,r}(y)\right] = s^r$.
- Probability that x and y don't match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

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

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 $\leq 1-(1-.7^{25})^{50} \approx .007$

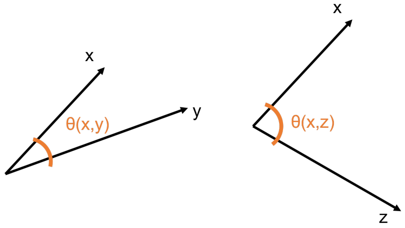
Expected Number of Items Scanned: (proportional to query time)

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

Generalizing Locality Sensitive Hashing

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.

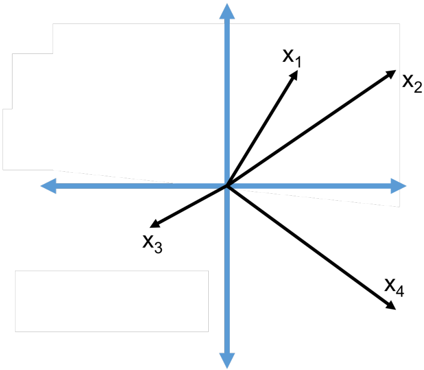


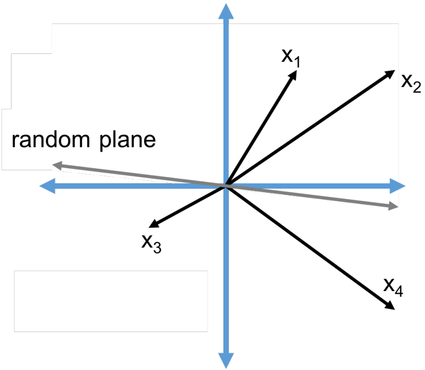
Cosine Similarity: $\cos(\theta(x,y)) = \frac{\langle x,y \rangle}{\|x\|_2 \cdot \|y\|_2}$.

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

SimHash for Cosine Similarity

SimHash Algorithm: LSH for cosine similarity.



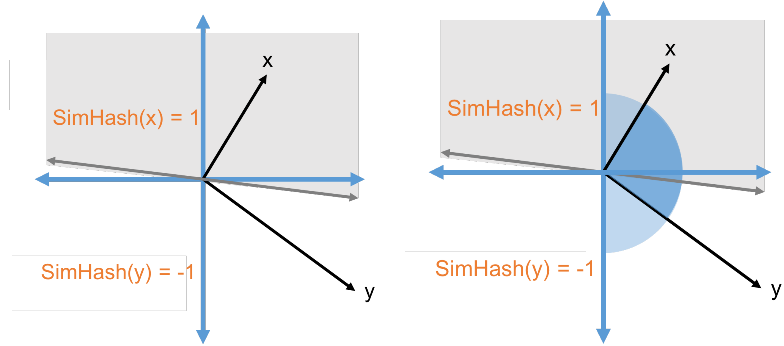


$SimHash(x) = \text{sign}(\langle x, t \rangle)$ for a random vector t.

SimHash for Cosine Similarity

What is $\Pr[SimHash(x) = SimHash(y)]$?

$SimHash(x) \neq SimHash(y)$ when the plane separates x from y.



- $\Pr[SimHash(x) \neq SimHash(y)] = \frac{\theta(x,y)}{\pi}$
- $\Pr[SimHash(x) = SimHash(y)] = 1 - \frac{\theta(x,y)}{\pi} \approx \frac{\cos(\theta(x,y))+1}{2}$

Questions on MinHash and Locality Sensitive Hashing?