# Similarity Estimation

Lecture 13

October 11, 2022

# Similar Items

Modern data: often unstructured and high-dimensional

**Examples:** documents, web pages, reviews, images, audio, video

# Similar Items

Modern data: often unstructured and high-dimensional

**Examples:** documents, web pages, reviews, images, audio, video

Given a collection of objects from a data collection:

- find all "similar" items (application: duplicate detection in documents)
- for an item $x$ find all items in the collection similar to $x$ (near-neighbor search, many applications)

# Similar Items

Modern data: often unstructured and high-dimensional

**Examples:** documents, web pages, reviews, images, audio, video

Given a collection of objects from a data collection:

- find all "similar" items (application: duplicate detection in documents)
- for an item $x$ find all items in the collection similar to $x$ (near-neighbor search, many applications)

Comparing two items expensive. Comparing all pairs, infeasible.

# High-level Ideas

- How to measure similarity/dissimilarity? Proxy functions for estimating/capturing similarity
- Focus only on *highly* similar items rather than try to find similarity for all pairs
- Compression/sketching/hashing to create compact representations of objects
- Fast/approximate near-neighbor search via ideas such as locality-sensitive-hashing, clustering etc

# Topics

- Jaccard similarity for sets and minhash
- Angular distance and simhash
- Locality-sensitive hashing

# Part I

# Jaccard Similarity and Min-wise independent Hashing

# Set Similarity

**Motivation:** How do we detect near-duplicate text documents? Web pages, papers, homeworks, ...?

# Set Similarity

**Motivation:** How do we detect near-duplicate text documents? Web pages, papers, homeworks, . . .?

Model documents as (multi)sets of “words” or more generally “shingles”

- A very large set of words/shingles
- Each document is a (small) set of words/shingles
- Large number of documents and each document is sparse in space of words/shingles

# Jaccard similarity of sets

**Definition:** given two subsets $S, T$ of a common universe the Jaccard similarity between $S$ and $T$ is defined as

$$\frac{|S \cap T|}{|S \cup T|}$$

and denoted by $\text{SIM}(S, T)$.

# Jaccard similarity of sets

**Definition:** given two subsets $S, T$ of a common universe the Jaccard similarity between $S$ and $T$ is defined as

$$\frac{|S \cap T|}{|S \cup T|}$$

and denoted by $\mathrm{SIM}(S, T)$.

**Assumption:** $S, T$ very similar if $\mathrm{SIM}(S, T) \geq \alpha$ for some fixed threshold $\alpha$. Say $\alpha = 0.7$

# Jaccard similarity of sets

**Definition:** given two subsets $S, T$ of a common universe the Jaccard similarity between $S$ and $T$ is defined as

$$\frac{|S \cap T|}{|S \cup T|}$$

and denoted by $\text{SIM}(S, T)$.

**Assumption:** $S, T$ very similar if $\text{SIM}(S, T) \geq \alpha$ for some fixed threshold $\alpha$. Say $\alpha = 0.7$

**Question:** Given many documents how do we find similar documents?

# Min Hashing

Let $n$ be the size of vocabulary

For a permutation $\sigma$ of $[n]$ and set $S$ let

$$\sigma_{\min}(S) = \text{first element of } S \text{ in } \sigma$$

Formally: $\sigma_{\min}(S) = \arg\min_{i \in S} \sigma(i)$

# Min Hashing

Let $n$ be the size of vocabulary

For a permutation $\sigma$ of $[n]$ and set $S$ let

$$\sigma_{\min}(S) = \text{first element of } S \text{ in } \sigma$$

Formally: $\sigma_{\min}(S) = \arg\min_{i \in S} \sigma(i)$

**Example:**

- $[7] = \{1, 2, 3, 5, 6, 7\}$. $\sigma = 3, 5, 7, 1, 6, 2, 4$
- $S = \{2, 3, 7\}$. $\sigma_{\min}(S) = 3$
- $T = \{2, 7\}$. $\sigma_{\min}(T) = 7$.

# Min Hashing

**Lemma**

*Let* $S, T$ *be two subsets of* $[n]$. *Suppose* $\sigma$ *is a random permutation of* $[n]$. *Then*

$$\Pr[\sigma_{\min}(S) = \sigma_{\min}(T)] = \frac{|S \cap T|}{|S \cup T|}.$$

# Min Hashing

- Pick $\ell$ random permutations $\sigma^1, \sigma^2, \ldots, \sigma^\ell$
- For each set $S$ store a $\ell$-tuple $(\sigma^1_{\min}(S), \ldots, \sigma^\ell_{\min}(S))$
- To check similarity between $S$ and $T$ let $s = |\{i \mid \sigma^i_{\min}(S) = \sigma^i_{\min}(T)\}|$. Output estimator $Z = \mathrm{SIM}(S, T) = s/\ell$

# Min Hashing

- Pick $\ell$ random permutations $\sigma^1, \sigma^2, \ldots, \sigma^\ell$
- For each set $S$ store a $\ell$-tuple $(\sigma^1_{\min}(S), \ldots, \sigma^\ell_{\min}(S))$
- To check similarity between $S$ and $T$ let $s = |\{i \mid \sigma^i_{\min}(S) = \sigma^i_{\min}(T)\}|$. Output estimator $Z = \mathrm{SIM}(S, T) = s/\ell$

$Z$ is an exact estimator for $\mathrm{SIM}(S, T)$.

**Exercise:** Suppose $\mathrm{SIM}(S, T) \geq \alpha$. How large should $\ell$ be such that $\Pr[Z < (1-\epsilon)\alpha] < \delta$?

# Min Hashing

In practice:

- Pick some sufficiently large $\ell$
- Use "shingles" instead of "words": depends on application
- Store for each $S$ the compact "sketch/signature" $(\sigma_{\min}^{1}(S), \ldots, \sigma_{\min}^{\ell}(S))$
- Do further optimizations for performance/space

See Chapter 3 in Mining Massive Data Sets book by Leskovic, Rajaraman, Ullman.

# Random permutation?

Random permutation like a random hash function is complex

- Cannot store compactly
- Computing $\sigma_{\min}(S)$ expensive

Need pseudorandom permutations that suffice.

# Minwise Independent Permutations

[Broder-Charikar-Frieze-Mitzemacher]

Given $n$, $S_n$ is the set of $n!$ permutations

Want a family $\mathcal{F} \subseteq S_n$ of permutations such that picking a random $\sigma$ from $\mathcal{F}$ behaves like a random permutation for Jaccard similarity

# Minwise Independent Permutations

[Broder-Charikar-Frieze-Mitzemacher]

Given $n$, $S_n$ is the set of $n!$ permutations

Want a family $\mathcal{F} \subseteq S_n$ of permutations such that picking a random $\sigma$ from $\mathcal{F}$ behaves like a random permutation for Jaccard similarity

**Definition**

A family $\mathcal{F} \subseteq S_n$ is a minwise independent family of permutations if for every $X \subseteq [n]$ and $a \in X$, for a $\sigma$ chosen uniformly from $\mathcal{F}$,

$$\Pr[\sigma_{\min}(X) = a] = \frac{1}{|X|}.$$

# Minwise Independent Permutations

**Definition**

A family $\mathcal{F} \subseteq S_n$ is a minwise independent family of permutations if for every $X \subseteq [n]$ and $a \in X$, for a $\sigma$ chosen uniformly from $\mathcal{F}$,

$$\Pr[\sigma_{\min}(X) = a] = \frac{1}{|X|}.$$

**Exercise:** show that minwise independent permutations suffice for Jaccard similarity estimation.

**Question:** is there an $\mathcal{F}$ which is not $S_n$? Not obvious!

- There exist minwise independent families of size $4^n$
- Any minwise independent family must have size $e^{(1-o(1))n}$

Exponential lower bound means we need to relax the requirement further.

# Minwise Independent Permutations

## Definition

A family $\mathcal{F} \subseteq S_n$ is a minwise independent family of permutations if for every $X \subseteq [n]$ and $a \in X$, for a $\sigma$ chosen uniformly from $\mathcal{F}$, $\Pr[\sigma_{\min}(X) = a] = \frac{1}{|X|}$.

Two relaxations:

- $\epsilon$-approximate minwise independence.

$$\frac{1-\epsilon}{|X|} \leq \Pr[\sigma_{\min}(X) = a] \leq \frac{1+\epsilon}{|X|}.$$

- Need condition to hold only for sets $X$ where $|X| \leq k$ for some parameter $k < n$. Sufficient for applications where sets are much smaller than $n$

# Relaxation of Minwise Independence

**Definition**

A family $\mathcal{F} \subseteq S_n$ is $(\epsilon, k)$ min-wise independent family if for all $X \subset [n]$ such that $|X| \leq k$, if $\sigma$ is chosen uniformly from $\mathcal{F}$,

$$\frac{1-\epsilon}{|X|} \leq \Pr[\sigma_{\min}(X) = a] \leq \frac{1+\epsilon}{|X|}.$$

# Minwise Independence and Hashing

**Question:** Is there a connection between minwise independent permutations and hashing?

Suppose $\mathcal{H}$ is a family of $t$-wise independent hash functions from $[n]$ to $[n]$. Let $h \in \mathcal{H}$. Why is $h$ not a permutation?

# Minwise Independence and Hashing

**Question:** Is there a connection between minwise independent permutations and hashing?

Suppose $\mathcal{H}$ is a family of $t$-wise independent hash functions from $[n]$ to $[n]$. Let $h \in \mathcal{H}$. Why is $h$ not a permutation? Because of collisions

Suppose $h : [n] \rightarrow [m]$ where $m \gg n$ then $h$ has very low probability of collisions. Then would $h$ behave like a minwise independent permutation?

# Minwise Independence and Hashing

**Theorem (Indyk)**

*Let $\mathcal{H}$ be a $t$-wise independent family of hash functions from $[n]$ to $[n]$ where $t = \Omega(\log \frac{1}{\epsilon})$. Then $\mathcal{H}$ is a $(\epsilon, k)$ minwise-independent family of permutations for $k = O(\epsilon n)$.*

Thus hash functions from $[n]$ to $[n]$ effectively suffice for minwise independence and can be used in minhashing.

# Minwise independence and Distinct Elements

Do you see connection between minwise independent permutations/hashing and Distinct Element sampling?

**Exercise:** How would you used minwise independent permutations to sample near-uniformly from the set of distinct elements in a stream?

# Part II

# Angular Distance and Simhash

# Angular distance

Given a collection of vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ in $\mathbb{R}^d$ representing some data objects ($\mathbf{v}_i$ is feature vector for object $\mathbf{i}$).

Two vectors $\mathbf{u}, \mathbf{v}$ "similar" if they point roughly in the same direction

Define $\text{dist}(\mathbf{u}, \mathbf{v}) = \theta(\mathbf{u}, \mathbf{v})/\pi$ where $\theta(\mathbf{u}, \mathbf{v})$ is angle between vectors $\mathbf{u}$ and $\mathbf{v}$. Assuming $\mathbf{u}, \mathbf{v}$ are unit vectors wlog we have $\mathbf{u} \cdot \mathbf{v} = \mathbf{cos}(\theta(\mathbf{u}, \mathbf{v}))$.
Similarity is $1 - \text{dist}(\mathbf{u}, \mathbf{v})$

# Sim Hash

[Charikar] as a special case of a connection between rounding algorithms and hashing

- Pick random hyperplane/unit vector $\boldsymbol{r}$
- For each $\boldsymbol{v_i}$ set $\boldsymbol{h_r}(\boldsymbol{v_i}) = \text{sign}(\boldsymbol{r} \cdot \boldsymbol{v_i})$

# Sim Hash

[Charikar] as a special case of a connection between rounding algorithms and hashing

- Pick random hyperplane/unit vector $\boldsymbol{r}$
- For each $\boldsymbol{v_i}$ set $\boldsymbol{h_r}(\boldsymbol{v_i}) = \text{sign}(\boldsymbol{r} \cdot \boldsymbol{v_i})$

**Lemma**

$\Pr[\boldsymbol{h_r}(\boldsymbol{v_i}) = \boldsymbol{h_r}(\boldsymbol{v_j})] = \theta(\boldsymbol{v_i}, \boldsymbol{v_j})/\pi$.

# Sim Hash

[Charikar] as a special case of a connection between rounding algorithms and hashing

- Pick random hyperplane/unit vector $\boldsymbol{r}$
- For each $\boldsymbol{v_i}$ set $\boldsymbol{h_r}(\boldsymbol{v_i}) = \text{sign}(\boldsymbol{r} \cdot \boldsymbol{v_i})$

**Lemma**

$\Pr[\boldsymbol{h_r}(\boldsymbol{v_i}) = \boldsymbol{h_r}(\boldsymbol{v_j})] = \theta(\boldsymbol{v_i}, \boldsymbol{v_j})/\pi$.

Using several random hyperplanes $\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_\ell$ we create a compact hash value/sketch for angle similarity. Need need pseudorandom hyperplanes ...

# A general observation

For Jaccard similarity and angular similarity we had the property that there is a family of hash functions $\mathcal{H}$ such that for $h$ chosen randomly from $\mathcal{H}$

$$\Pr[h(A) = h(B)] = \text{sim}(A, B)$$

# A general observation

For Jaccard similarity and angular similarity we had the property that there is a family of hash functions $\mathcal{H}$ such that for $h$ chosen randomly from $\mathcal{H}$

$$\Pr[h(A) = h(B)] = \text{sim}(A, B)$$

**Question:** When is the above true in general?

# A general observation

For Jaccard similarity and angular similarity we had the property that there is a family of hash functions $\mathcal{H}$ such that for $h$ chosen randomly from $\mathcal{H}$

$$\Pr[h(A) = h(B)] = \text{sim}(A, B)$$

**Question:** When is the above true in general?

**Lemma (Charikar)**

*If there is a hash family for a similarity measure $sim(\cdot,\cdot)$ with the preceding property then $d(\cdot,\cdot) = 1 - sim(\cdot,\cdot)$ is a metric and further $d$ is embeddable in generalized Hamming distance.*

Part III

# Similarity and Distance Measures

# Similarity and Distance

Different objects and applications drive similarity measures

Similarity between $\boldsymbol{x}$ and $\boldsymbol{y}$ large implies they are close to being identical

Another common way is to use *distances* where small distances mean higher similarity

# Some common measures

- Jaccard similarity measure of sets
- Cosine angle between vectors
- Distance measures: norm based measures $\|x - y\|_p$ say $p = 1, 2, \ldots$
- Hamming distance between vectors
- Edit distance between strings
- Distance measures between probability distributions: earth-mover distance, KL divergence/relative entropy (not symmetric),

For distance measures: dimensionality reduction like JL provides way to speed up pairwise distance computation.

Part IV

# Near-Neighbor Search

# Similarity estimation and search

Collection of data items/objects $\mathcal{D}$

We saw ways to compress objects to speed up similarity estimation between objects

# Similarity estimation and search

Collection of data items/objects $\mathcal{D}$

We saw ways to compress objects to speed up similarity estimation between objects

Still two problems remain:

- find all highly similar pairs — cannot do quadratic time even with compressed hashes
- new point $x$: want to know all points "similar" to $x$ in $\mathcal{D}$. linear search is not feasible

# Near-Neighbor Search

Collection of data items/objects $\mathcal{D}$

Preprocess $\mathcal{D}$ using small space so that given query $x$, output all $y \in \mathcal{D}$ with high similarity to $x$ (or small distance to $x$)

# Near-Neighbor Search

Collection of data items/objects $\mathcal{D}$

Preprocess $\mathcal{D}$ using small space so that given query $x$, output all $y \in \mathcal{D}$ with high similarity to $x$ (or small distance to $x$)

Fundamental data structure problem with many applications

# Near-Neighbor Search

Collection of data items/objects $\mathcal{D}$

Preprocess $\mathcal{D}$ using small space so that given query $x$, output all $y \in \mathcal{D}$ with high similarity to $x$ (or small distance to $x$)

Fundamental data structure problem with many applications

**Classical (exact) solution approaches from geometry:** Voronoi diagrams, $k$-d trees, space partition/filling approaches.

# Near-Neighbor Search

Collection of data items/objects $\mathcal{D}$

Preprocess $\mathcal{D}$ using small space so that given query $x$, output all $y \in \mathcal{D}$ with high similarity to $x$ (or small distance to $x$)

Fundamental data structure problem with many applications

**Classical (exact) solution approaches from geometry:** Voronoi diagrams, $k$-d trees, space partition/filling approaches.
**Major drawback:** curse of dimensionality for exact search

# Near-Neighbor Search

Collection of data items/objects $\mathcal{D}$

Preprocess $\mathcal{D}$ using small space so that given query $x$, output all $y \in \mathcal{D}$ with high similarity to $x$ (or small distance to $x$)

Fundamental data structure problem with many applications

**Classical (exact) solution approaches from geometry:** Voronoi diagrams, $k$-d trees, space partition/filling approaches.
**Major drawback:** curse of dimensionality for exact search

**Modern/recent approaches:** *approximate* NN search via locality-sensitive hashing (LSH), randomized $k$-d trees, etc

# LSH approach

Initially developed for NN search in high-dimensional Euclidean space and then generalized to other similarity/distance measures.

High-level ideas:

- collection of $n$ objects $p_1, p_2, \ldots, p_n$ in some space
- some distance/similarity measure $d$ on pairs of objects
- create a hash function family $\mathcal{H}$ with the property that each hash function $h$ has "locality" preserving property
- $h$ maps points similar to each other (or closer in distance) to the same bucket with higher probability than it would map points that are not so similar
- Use multiple independent hash functions to create a data structure
- Hashing family depends on the similarity/distance measure