# COMPSCI 514: ALGORITHMS FOR DATA SCIENCE

Cameron Musco

University of Massachusetts Amherst. Spring 2019.

Lecture 9

## LOGISTICS

- Problem Set 2 was released Sunday night and is due Sunday 3/8.
- Please make sure to mark all teammates in the GradeScope submission (don't just write names on the document).
- The Midterm will be on Thursday 3/12. Will cover material through this week.
- Study guide/practice questions to be released soon.

## SUMMARY

Last Class:

- Continued on the frequent elements problem.
- Misra-Greis summaries (deterministic method).
- Started on Count-Min Sketch (randomized method).

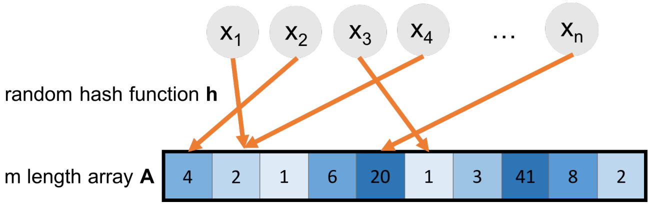
This Class:

- Finish up Count-Min Sketch analysis.
- Start on randomized methods for dimensionality reduction.
- The Johnson-Lindenstrauss Lemma.

## FREQUENT ELEMENTS WITH COUNT-MIN SKETCH

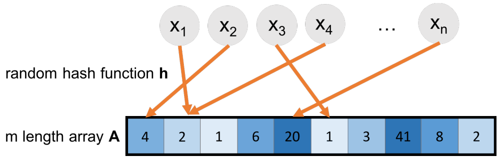
**Frequent Items Problem:** Identify all items with frequency $\geq n/k$ in a stream of $n$ items.

**Count-Min Sketch:** Bloom filter like solution using random hashing.



Use $A[\mathbf{h}(x)]$ to estimate $f(x)$, the frequency of $x$ in the stream. I.e., $|\{x_i : x_i = x\}|$.

## COUNT-MIN SKETCH ACCURACY



Use $A[\mathbf{h}(x)]$ to estimate $f(x)$

**Claim 1:** We always have $A[\mathbf{h}(x)] \geq f(x)$. Why?

- $A[\mathbf{h}(x)]$ counts the number of occurrences of any $y$ with $\mathbf{h}(y) = \mathbf{h}(x)$, including $x$ itself.
- $A[\mathbf{h}(x)] = f(x) + \sum_{y \neq x : \mathbf{h}(y) = \mathbf{h}(x)} f(y)$.

$f(x)$: frequency of $x$ in the stream (i.e., number of items equal to $x$). $\mathbf{h}$: random hash function. $m$: size of count-min sketch array.

## COUNT-MIN SKETCH ACCURACY

$$A[\mathbf{h}(x)] = f(x) + \underbrace{\sum_{y \neq x : \mathbf{h}(y) = \mathbf{h}(x)} f(y)}_{\text{error in frequency estimate}} .$$

**Expected Error:**

$$\mathbb{E}\left[\sum_{y \neq x : \mathbf{h}(y) = \mathbf{h}(x)} f(y)\right] = \sum_{y \neq x} \Pr(\mathbf{h}(y) = \mathbf{h}(x)) \cdot f(y)$$

$$= \sum_{y \neq x} \frac{1}{m} \cdot f(y) = \frac{1}{m} \cdot (n - f(x)) \leq \frac{n}{m}$$

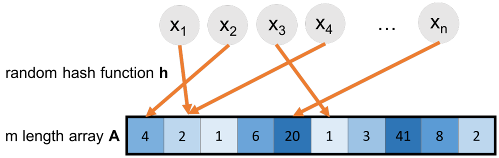
What is a bound on probability that the error is $\geq \frac{3n}{m}$?

**Markov's inequality:** $\Pr\left[\sum_{y \neq x : \mathbf{h}(y) = \mathbf{h}(x)} f(y) \geq \frac{3n}{m}\right] \leq \frac{1}{3}$.

What property of **h** is required to show this bound? 2-universal.

$f(x)$: frequency of $x$ in the stream (i.e., number of items equal to $x$). **h**: random hash function. $m$: size of count-min sketch array.

## COUNT-MIN SKETCH ACCURACY



**Claim:** For any $x$, with probability at least 2/3,

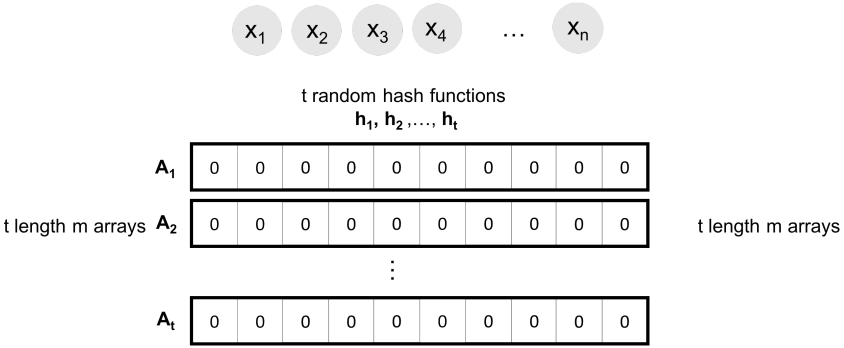
$$f(x) \leq A[\mathbf{h}(x)] \leq f(x) + \frac{3n}{m} . \frac{\epsilon n}{k} .$$

To solve the $(\epsilon, k)$-Frequent elements problem, set $m = \frac{3k}{\epsilon}$.

How can we improve the success probability? Repetition.

$f(x)$: frequency of $x$ in the stream (i.e., number of items equal to $x$). $\mathbf{h}$: random hash function. $m$: size of count-min sketch array.

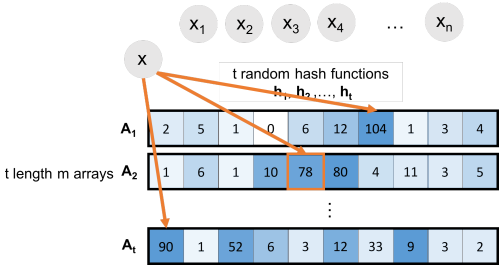
## COUNT-MIN SKETCH ACCURACY



Estimate $f(x)$ with $\tilde{f}(x) = \min_{i \in [t]} A_i[\mathbf{h}_i(x)]$. (count-min sketch)

Why min instead of mean or median? The minimum estimate is always the most accurate since they are all overestimates of the true frequency!

## COUNT-MIN SKETCH ANALYSIS



Estimate $f(x)$ by $\tilde{f}(x) = \min_{i\in[t]} A_i[\mathbf{h}_i(x)]$

- For every $x$ and $i \in [t]$, we know that for $m = O(k/\epsilon)$, with probability $\geq 2/3$:

$$f(x) \leq A_i[\mathbf{h}_i(x)] \leq f(x) + \frac{\epsilon n}{k}.$$

- What is $\Pr[f(x) \leq \tilde{f}(x) \leq f(x) + \frac{\epsilon n}{k}]$? $1 - 1/3^t$.
- To get a good estimate with probability $\geq 1 - \delta$, set $t = O(\log(1/\delta))$.

## COUNT-MIN SKETCH

**Upshot:** Count-min sketch lets us estimate the frequency of every item in a stream up to error $\frac{\epsilon n}{k}$ with probability $\geq 1 - \delta$ in $O\left(\log(1/\delta) \cdot k/\epsilon\right)$ space.

- Accurate enough to solve the $(\epsilon, k)$-Frequent elements problem.
- Actually identifying the frequent elements quickly requires a little bit of further work.

  **One approach:** Separately store a list of potential frequent elements as they come in. At step $i$, keep any elements whose estimated frequency is $\geq i/k$. List contains at most $O(k)$ items at any step and has all items with frequency $\geq n/k$ stored at the end of the stream.

Questions on Frequent Elements?

## HIGH DIMENSIONAL DATA

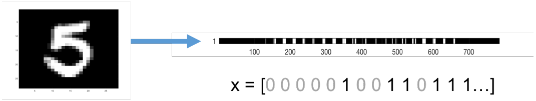
'Big Data' means not just many data points, but many measurements per data point. I.e., very high dimensional data.

- Twitter has 321 million active monthly users. Records (tens of) thousands of measurements per user: who they follow, who follows them, when they last visited the site, timestamps for specific interactions, how many tweets they have sent, the text of those tweets, etc.
- A 3 minute Youtube clip with a resolution of $500 \times 500$ pixels at 15 frames/second with 3 color channels is a recording of $\geq 2$ billion pixel values. Even a $500 \times 500$ pixel color image has $750,000$ pixel values.
- The human genome contains 3 billion+ base pairs. Genetic datasets often contain information on 100s of thousands+ mutations and genetic markers.

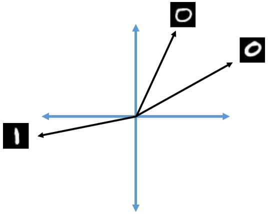
## DATA AS VECTORS AND MATRICES

In data analysis and machine learning, data points with many attributes are often stored, processed, and interpreted as high dimensional vectors, with real valued entries.

ATAGCCGTAGT $\longrightarrow$ x = [1 2 1 3 4 4 3 2 1 3 4]



x = [0 0 0 0 0 1 0 0 1 1 0 1 1 1...]



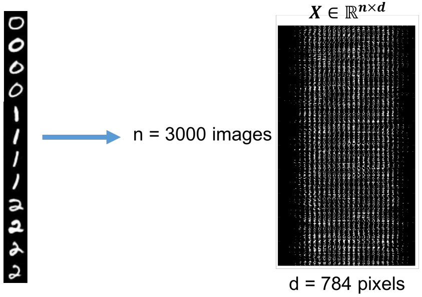
Similarities/distances between vectors (e.g., $\langle x, y\rangle$, $\|x - y\|_2$) have meaning for underlying data points.

## DATASETS AS VECTORS AND MATRICES

Data points are interpreted as high dimensional vectors, with real valued entries. Data set is interpreted as a matrix.

**Data Points:** $\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n \in \mathbb{R}^d$.

**Data Set:** $X \in \mathbb{R}^{n \times d}$ with $i^{th}$ rows equal to $\vec{x}_i$.



Many data points $n \implies$ tall. Many dimensions $d \implies$ wide.

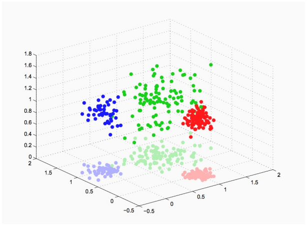
## DIMENSIONALITY REDUCTION

**Dimensionality Reduction:** Compress data points so that they lie in many fewer dimensions.

$$\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{R}^d \rightarrow \tilde{x}_1, \ldots, \tilde{x}_n \in \mathbb{R}^m \text{ for } m \ll d.$$

$\longrightarrow$ $x$ = [0 0 0 0 0 1 0 0 1 1 0 1 1 1...] $\longrightarrow$ $\tilde{x}$ = [-5.5 4 3.2 -1]

'Lossy compression' that still preserves important information about the relationships between $\vec{x}_1, \ldots, \vec{x}_n$.

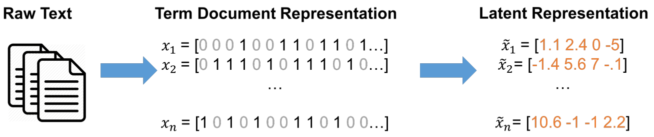


Generally will not consider directly how well $\tilde{x}_i$ approximates $\vec{x}_i$.

## DIMENSIONALITY REDUCTION

Dimensionality reduction is one of the most important techniques in data science.

- Principal component analysis
- Latent semantic analysis (LSA)



Raw Text
Term Document Representation
Latent Representation
$x_1$ = [0 0 0 1 0 0 1 1 0 1 1 0 1...]
$x_2$ = [0 1 1 1 0 1 0 1 1 1 0 1 0...]
...
$x_n$ = [1 0 1 0 1 0 0 1 1 0 1 0 0...]
$\tilde{x}_1$ = [1.1 2.4 0 -5]
$\tilde{x}_2$= [-1.4 5.6 7 -.1]
...
$\tilde{x}_n$= [10.6 -1 -1 2.2]

- Linear discriminant analysis
- Autoencoders

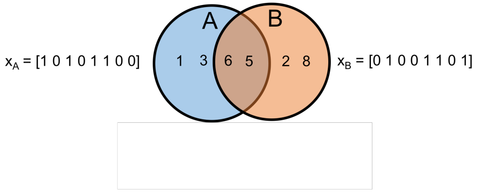
Compressing data makes it more efficient to work with. May also remove extraneous information/noise.

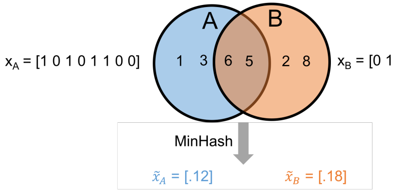
## LOW DISTORTION EMBEDDING

**Low Distortion Embedding:** Given $\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{R}^d$, distance function $D$, and error parameter $\epsilon \geq 0$, find $\tilde{x}_1, \ldots, \tilde{x}_n \in \mathbb{R}^m$ (where $m \ll d$) and distance function $\tilde{D}$ such that for all $i, j \in [n]$:

$$(1-\epsilon) D(\vec{x}_i, \vec{x}_j) \leq \tilde{D}(\tilde{x}_i, \tilde{x}_j) \leq (1+\epsilon) D(\vec{x}_i, \vec{x}_j).$$

Have already seen one example in class: MinHash.





With large enough signature size $r$, $\frac{\#\text{ matching entries in } \tilde{x}_A, \tilde{x}_B}{r} \approx J(\vec{x}_A, \vec{x}_B)$.

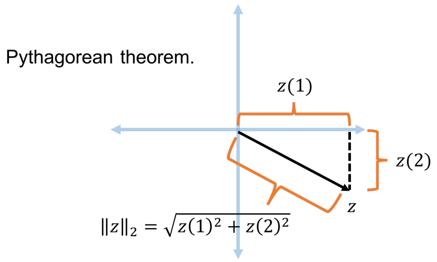
- Reduce dimension from $d = |U|$ to $r$. Note: here $J(\vec{x}_A, \vec{x}_B)$ is a similarity rather than a distance. So this is not quite low distortion

## EMBEDDINGS FOR EUCLIDEAN SPACE

**Euclidean Low Distortion Embedding:** Given $\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{R}^d$ and error parameter $\epsilon \geq 0$, find $\tilde{x}_1, \ldots, \tilde{x}_n \in \mathbb{R}^m$ (where $m \ll d$) such that for all $i, j \in [n]$:

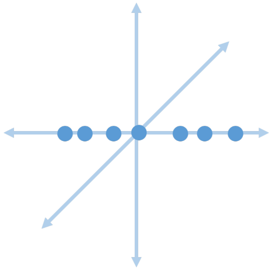
$$(1-\epsilon)\|\vec{x}_i - \vec{x}_j\|_2 \leq \|\tilde{x}_i - \tilde{x}_j\|_2 \leq (1+\epsilon)\|\vec{x}_i - \vec{x}_j\|_2.$$

Recall that for $\vec{z} \in \mathbb{R}^n$, $\|\vec{z}\|_2 = \sqrt{\sum_{i=1}^n \vec{z}(i)^2}$.



## EMBEDDING WITH ASSUMPTIONS

**A very easy case:** Assume that $\vec{x}_1, \ldots, \vec{x}_n$ all lie on the 1st axis in $\mathbb{R}^d$.

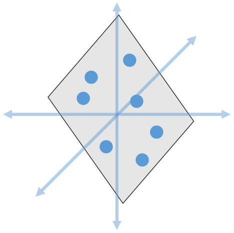


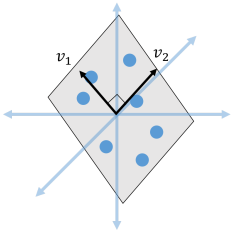
Set $m = 1$ and $\tilde{x}_i = \vec{x}_i(1)$ (i.e., $\tilde{x}_i$ is just a single number).

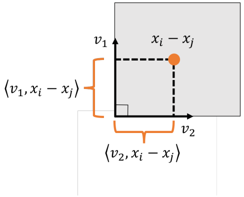
- $\|\tilde{x}_i - \tilde{x}_j\|_2 = \sqrt{[\vec{x}_i(1) - \vec{x}_j(1)]^2} = |\vec{x}_i(1) - \vec{x}_j(1)| = \|\vec{x}_i - \vec{x}_j\|_2$.
- An embedding with no distortion from any $d$ into $m = 1$.

## EMBEDDING WITH ASSUMPTIONS

**A pretty easy case:** Assume that $\vec{x}_1, \ldots \vec{x}_n$ lie in any $k$-dimensional subspace $\mathcal{V}$ of $\mathbb{R}^d$.






$v_1$
$x_i - x_j$
$\langle v_1, x_i - x_j \rangle$
$v_2$
$\langle v_2, x_i - x_j \rangle$

- Let $\vec{v}_1, \vec{v}_2, \ldots \vec{v}_k$ be an orthonormal basis for $\mathcal{V}$ and let $\mathbf{V} \in \mathbb{R}^{d \times k}$ be the matrix with these vectors as its columns.
- For all $i, j$ we have $\vec{x}_i - \vec{x}_j \in \mathcal{V}$ and (a good exercise!):

$$\|\vec{x}_i - \vec{x}_j\|_2 = \sqrt{\sum_{\ell=1}^{k} \langle v_\ell, \vec{x}_i - \vec{x}_j \rangle^2} = \|\mathbf{V}^T(\vec{x}_i - \vec{x}_j)\|_2.$$

## EMBEDDING WITH NO ASSUMPTIONS

What about when we don't make any assumptions on $\vec{x}_1, \ldots \vec{x}_n$. I.e., they can be scattered arbitrarily around $d$-dimensional space?

- Can we find a no-distortion embedding into $m \ll d$ dimensions? No. Require $m = d$.
- Can we find an $\epsilon$-distortion embedding into $m \ll d$ dimensions for $\epsilon > 0$? Yes! Always, with $m$ depending on $\epsilon$.

$$\text{For all } i, j : (1-\epsilon)\|\vec{x}_i - \vec{x}_j\|_2 \leq \|\tilde{x}_i - \tilde{x}_j\|_2 \leq (1+\epsilon)\|\vec{x}_i - \vec{x}_j\|_2.$$

## THE JOHNSON-LINDENSTRAUSS LEMMA

**Johnson-Lindenstrauss Lemma:** For any set of points $\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{R}^d$ and $\epsilon > 0$ there exists a linear map $\mathbf{\Pi} : \mathbb{R}^d \to R^m$ such that $m = O\left(\frac{\log n}{\epsilon^2}\right)$ and letting $\tilde{x}_i = \mathbf{\Pi} x_i$:

$$\text{For all } i, j: \ (1-\epsilon)\|\vec{x}_i - \vec{x}_j\|_2 \leq \|\tilde{x}_i - \tilde{x}_j\|_2 \leq (1+\epsilon)\|\vec{x}_i - \vec{x}_j\|_2.$$

Further, if $\mathbf{\Pi} \in \mathbb{R}^{m \times d}$ has each entry chosen i.i.d. from $\mathcal{N}(0, 1/m)$, it satisfies the guarantee with high probability.

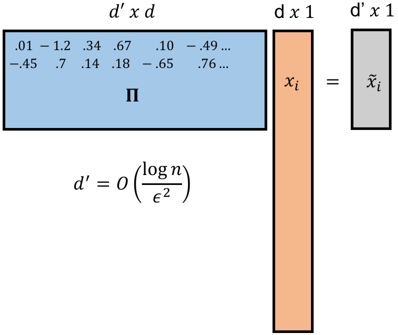
For $d = 1$ trillion, $\epsilon = .05$, and $n = 100,000$, $m \approx 6600$.

Very surprising! Powerful result with a simple construction: applying a random linear transformation to a set of points preserves distances between all those points with high probability.

## RANDOM PROJECTION

For any $\vec{x}_1, \ldots, \vec{x}_n$ and $\mathbf{\Pi} \in \mathbb{R}^{m \times d}$ with each entry chosen i.i.d. from $\mathcal{N}(0, 1/m)$, with high probability, letting $\tilde{x}_i = \mathbf{\Pi}\vec{x}_i$:

$$\text{For all } i, j: \; (1-\epsilon)\|\vec{x}_i - \vec{x}_j\|_2 \leq \|\tilde{x}_i - \tilde{x}_j\|_2 \leq (1+\epsilon)\|\vec{x}_i - \vec{x}_j\|_2.$$



- $\mathbf{\Pi}$ is known as a random projection.
- $\mathbf{\Pi}$ is data oblivious. Stark contrast to methods like PCA.

## ALGORITHMIC CONSIDERATIONS

- Many alternative constructions: $\pm 1$ entries, sparse (most entries 0), Fourier structured, etc. $\implies$ more efficient computation of $\tilde{x}_i = \mathbf{\Pi}\vec{x}_i$.
- Data oblivious property means that once $\mathbf{\Pi}$ is chosen, $\tilde{x}_1, \ldots, \tilde{x}_n$ can be computed in a stream with little memory.
- Memory needed is just $O(d + nm)$ vs. $O(nd)$ to store the full data set.
- Compression can also be easily performed in parallel on different servers.
- When new data points are added, can be easily compressed, without updating existing points.