10-605/805 – ML for Large Datasets
Lecture 10: Randomized Algorithms

Henry Chai

9/29/22

Front Matter

- HW3 released 9/23, due 10/4 at 11:59 PM
- Midterm exam on 10/11, two weeks from today!
    - Lecture on 10/6 (next Thursday) will be an *optional* practice exam - these will not be collected/graded
        - Recitation on 10/7 (next Friday) will go through the solutions
        - Both the practice exam and the solutions will be released after the Recitation
- Lecture on 10/4 (next Tuesday) will be an AWS tutorial; please bring your laptops to class on that day

Two Specific Applications of Feature Hashing

1. *Count-min sketch* for logistic regression
2. *Locality sensitive hashing* for nearest neighbors/clustering

Two Specific Applications of Feature Hashing

1. *Count-min sketch* for logistic regression
2. *Locality sensitive hashing* for nearest neighbors/clustering

Count-min Sketch

- Data structure used to estimate the frequency of items in some stream of inputs
  - Running example: finding "viral" terms in Google searches (https://trends.google.com/trends/?geo=US)
  - Naïve approach: just keep an array with an index for every possible search term and add one to that index when the term appears
    - Massive array
    - Could be dynamic/need to grow if unseen search terms arrive

Count-min Sketch: Connection to Logistic Regression

- If $\boldsymbol{x}$ is a (sparse) one-hot encoded vector, then we can efficiently compute ${\boldsymbol{w}^{(t)}}^T \boldsymbol{x}$ as

$$\sum_{j\,:\,x_j \neq 0} w_j^{(t)} x_j \approx \sum_{j\,:\,x_j \neq 0} \left(\min_i C[i, h_i(j)] \right) x_j$$

1	2	3	$\cdots$	m
$C[1,1]$	$C[1,2]$	$C[1,3]$	$\cdots$	$C[1,m]$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$C[r,1]$	$C[r,2]$	$C[r,3]$	$\cdots$	$C[r,m]$

- This matrix holds a compact representation of $\boldsymbol{w}^{(t)}$ at each iteration

Count-min Sketch: Algorithm

- Input: r independent hash functions $h_1, \dots, h_r$ that each map to m buckets

1. Initialize an $r \times m$ count matrix, C, to all zeros
2. For each item, s, in some stream of data:
   a. For $i \in \{1, \dots, r\}$:
      i. $C[i, h_i(s)] \mathrel{+}= 1$
3. For any item s, return $\hat{c}_s = \min_i(C[i, h_i(s)])$ as an approximation c_s, the true number of occurrences of s

Observation: Count-min sketch can only overestimate counts!

But by how much?

- Input: r independent hash functions $h_1, \ldots, h_r$ that each map to m buckets

1. Initialize an $r \times m$ count matrix, C, to all zeros
2. For each item, s, in some stream of data:
   a. For $i \in \{1, \ldots, r\}$:
      i. $C[i, h_i(s)] += 1$
3. For any item s, return $\hat{c}_s = \min_i (C[i, h_i(s)])$ as an approximation c_s, the true number of occurrences of s

Count-min Sketch: Theoretical Guarantees

- Given $r > \log_2 {}^1/_\delta$ hash functions that each map to $m > {}^2/_\epsilon$ buckets, then $\forall\, s$

$$P(\hat{c}_s \geq c_s + \epsilon \|\boldsymbol{c}\|_1) \leq \delta$$

where $\|\boldsymbol{c}\|_1$ is the total number of items in the stream.

- Key assumptions:
  - All hash functions are independent of each other

$$P\left(h_i(s) \mid h_j(s')\right) = P\big(h_i(s)\big)\ \forall\, i, j \in \{1, \dots, r\} \text{ and } s, s'$$

  - Each hash function is *pairwise independent*

$$P(h_i(s) = b \cap\ h_i(s') = b') = \frac{1}{m^2}\ \forall\, i \in \{1, \dots, r\},$$
$$b, b' \in \{0, \dots, m\} \text{ and } s, s'$$

$$\Rightarrow P\big(h_i(s) = h_i(s')\big) = \frac{1}{m} \forall\, i \in \{1, \dots, r\} \text{ and } s, s'$$

Source: Cormode and Muthukrishnan, 2005 (http://dimacs.rutgers.edu/~graham/pubs/papers/cm-full.pdf)

Count-min Sketch: Theoretical Guarantees Proof

- Given $r > \log_2 {}^1/_\delta$ hash functions that each map to $m > {}^2/_\epsilon$ buckets, then $\forall\ s$

$$P(\hat{c}_s \geq c_s + \epsilon \|\boldsymbol{c}\|_1) \leq \delta$$

where $\|\boldsymbol{c}\|_1$ is the total number of items in the stream.

- Proof:
  - For an arbitrary hash function, h_i, and item, s:

$$\mathbb{E}[C[i, h_i(s)]] = c_s + \sum_{s' \neq s} P\big(h_i(s) = h_i(s')\big)\, c_{s'}$$

(by pairwise independence) $$= c_s + \sum_{s' \neq s} \frac{1}{m} c_{s'} = c_s + \frac{1}{m} \sum_{s' \neq s} c_{s'}$$

(by definition of m and $\|\boldsymbol{c}\|_1$) $$\leq c_s + \frac{\epsilon}{2} \sum_{s' \neq s} c_{s'} \leq c_s + \frac{\epsilon}{2} \|\boldsymbol{c}\|_1$$

Count-min Sketch: Theoretical Guarantees Proof (Cont.)

- Given $r > \log_2 {}^1\!/_\delta$ hash functions that each map to $m > {}^2\!/_\epsilon$ buckets, then $\forall\ s$

$$P(\hat{c}_s \geq c_s + \epsilon \|\boldsymbol{c}\|_1) \leq \delta$$

where $\|\boldsymbol{c}\|_1$ is the total number of items in the stream.

- Proof:
  - For an arbitrary hash function, h_i, and item, s:

$$P(C[i, h_i(s)] \geq c_s + \epsilon \|\boldsymbol{c}\|_1) = P(C[i, h_i(s)] - c_s \geq \epsilon \|\boldsymbol{c}\|_1)$$

(by Markov's inequality) $$\leq \frac{\mathbb{E}\left[C[i, h_i(s)]\right] - c_s}{\epsilon \|\boldsymbol{c}\|_1}$$

(by bound on $\mathbb{E}\left[C[i, h_i(s)]\right]$) $$\leq \frac{c_s + \frac{\epsilon}{2}\|\boldsymbol{c}\|_1 - c_s}{\epsilon \|\boldsymbol{c}\|_1} = \frac{1}{2}$$

Count-min Sketch: Theoretical Guarantees Proof (Cont.)

- Given $r > \log_2 {}^1\!/_\delta$ hash functions that each map to $m > {}^2\!/_\epsilon$ buckets, then $\forall\, s$

$$P(\hat{c}_s \geq c_s + \epsilon \|\boldsymbol{c}\|_1) \leq \delta$$

where $\|\boldsymbol{c}\|_1$ is the total number of items in the stream.

- Proof:
  - For an arbitrary item, s:

$$P(\hat{c}_s \geq c_s + \epsilon \|\boldsymbol{c}\|_1) = P\left(\min_i C[i, h_i(s)] \geq c_s + \epsilon \|\boldsymbol{c}\|_1\right)$$

$$= P\left(\bigcap_i C[i, h_i(s)] \geq c_s + \epsilon \|\boldsymbol{c}\|_1\right)$$

(by independence) $$= \prod_i P(C[i, h_i(s)] \geq c_s + \epsilon \|\boldsymbol{c}\|_1)$$

(by definition of δ) $$\leq {}^1\!/_{2^r} = \delta$$

■

Count-min Sketch: Summary

- Key idea: use multiple (independent) hash functions to negate the effect of collisions in any single hash function
- Nice theoretical guarantees
- Can address overestimation bias to make unbiased count estimators (see HW3)
- Can be extended to handle weighted updates as in SGD logistic regression → efficient model updates in high-dimensional settings (large k)
  - For more details see: https://arxiv.org/pdf/1711.02305.pdf

Two Specific Applications of Feature Hashing

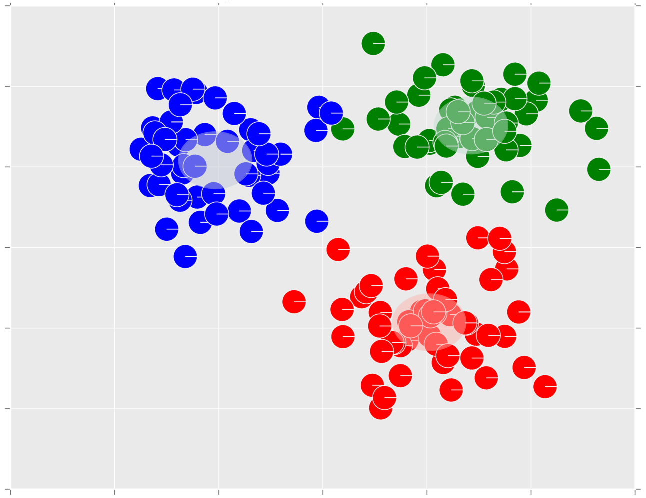
1. *Count-min sketch* for logistic regression
2. *Locality sensitive hashing* for nearest neighbors/clustering

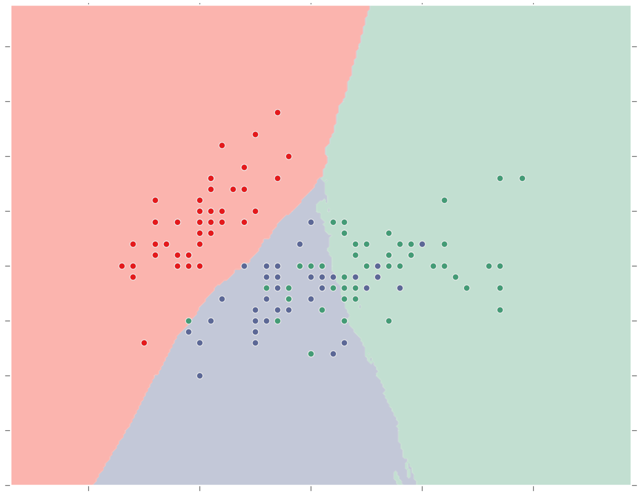
Two Specific Applications of Feature Hashing

1. *Count-min sketch* for logistic regression
2. *Locality sensitive hashing* for nearest neighbors/clustering

Locality-Sensitive Hashing

- Key idea: use hash functions that map “similar” items to the same function
  - Plot twist: collisions are the goal!
- Enables efficient comparison between points
- In the context of large-scale machine learning, can be used for clustering or nearest neighbor models in high-dimensional space





Figures courtesy of Matt Gormley

Locality-Sensitive Hashing: Formal Definition

- A *family* of hash functions $\mathcal{F}$ that map $\mathbb{R}^k \rightarrow \{0, \ldots, m\}$ is $(\epsilon, c\epsilon, \delta_1, \delta_2)$-sensitive with respect to some distance metric d iff $\forall$ $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^k$, a hash function h chosen uniformly at random from $\mathcal{F}$ has the property that
  - if $d(\boldsymbol{x}, \boldsymbol{y}) \leq \epsilon$, then $P\big(h(\boldsymbol{x}) = h(\boldsymbol{y})\big) \geq \delta_1$
    - "Points close to each other (according to some distance metric d) are likely to collide"
  - if $d(\boldsymbol{x}, \boldsymbol{y}) \geq c\epsilon$, then $P\big(h(\boldsymbol{x}) = h(\boldsymbol{y})\big) \leq \delta_2$
    - "Points far apart (according to some distance metric d) are unlikely to collide"

Locality-Sensitive Hashing: Example

- Random projection:
  - $m = 1$ so our hash functions map $\mathbb{R}^k$ to $\{0,1\}$
  - Let $\mathcal{F}$ be the set of all linear decision boundaries:

$$\mathcal{F} = \left\{ h : h_{\boldsymbol{w}}(\boldsymbol{x}) = \begin{cases} 1 & \boldsymbol{w}^T\boldsymbol{x} \geq 0 \\ 0 & \text{otherwise} \end{cases} \right\}$$

  - $d(\boldsymbol{x}, \boldsymbol{y}) = 1 - \frac{\boldsymbol{x}^T\boldsymbol{y}}{\|\boldsymbol{x}\|_2 \|\boldsymbol{y}\|_2}$, the *cosine distance*
    - Recall that one way to define a dot product is

$$\boldsymbol{x}^T\boldsymbol{y} = \|\boldsymbol{x}\|_2 \|\boldsymbol{y}\|_2 \cos\theta \rightarrow \cos\theta = \frac{\boldsymbol{x}^T\boldsymbol{y}}{\|\boldsymbol{x}\|_2 \|\boldsymbol{y}\|_2}$$

where θ is the angle between the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$

Locality-Sensitive Hashing: Example

- Random projection:
  - $m = 1$ so our hash functions map $\mathbb{R}^k$ to $\{0,1\}$
  - Let $\mathcal{F}$ be the set of all linear decision boundaries:

$$\mathcal{F} = \left\{ h : h_{\boldsymbol{w}}(\boldsymbol{x}) = \begin{cases} 1 & \boldsymbol{w}^T \boldsymbol{x} \geq 0 \\ 0 & \text{otherwise} \end{cases} \right\}$$

  - $d(\boldsymbol{x}, \boldsymbol{y}) = 1 - \frac{\boldsymbol{x}^T \boldsymbol{y}}{\|\boldsymbol{x}\|_2 \|\boldsymbol{y}\|_2}$, the *cosine distance*
  - For a random weight vector $\boldsymbol{w}$ (e.g., weights sampled independently from a standard Gaussian)

$$P\left(h_{\boldsymbol{w}}(\boldsymbol{x}) = h_{\boldsymbol{w}}(\boldsymbol{y})\right) = 1 - \frac{\arccos 1 - d(\boldsymbol{x}, \boldsymbol{y})}{\pi}$$

  - $\mathcal{F}$ is $\left(\epsilon, c\epsilon, 1 - \frac{\arccos 1-\epsilon}{\pi}, 1 - \frac{\arccos 1-c\epsilon}{\pi}\right)$-sensitive

Locality-Sensitive Hashing: Example

- Random projection:
  - $m = 1$ so our hash functions map $\mathbb{R}^k$ to $\{0,1\}$
  - Let $\mathcal{F}$ be the set of all linear decision boundaries:

$$\mathcal{F} = \left\{ h : h_{\boldsymbol{w}}(\boldsymbol{x}) = \begin{cases} 1 & \boldsymbol{w}^T\boldsymbol{x} \geq 0 \\ 0 & \text{otherwise} \end{cases} \right\}$$

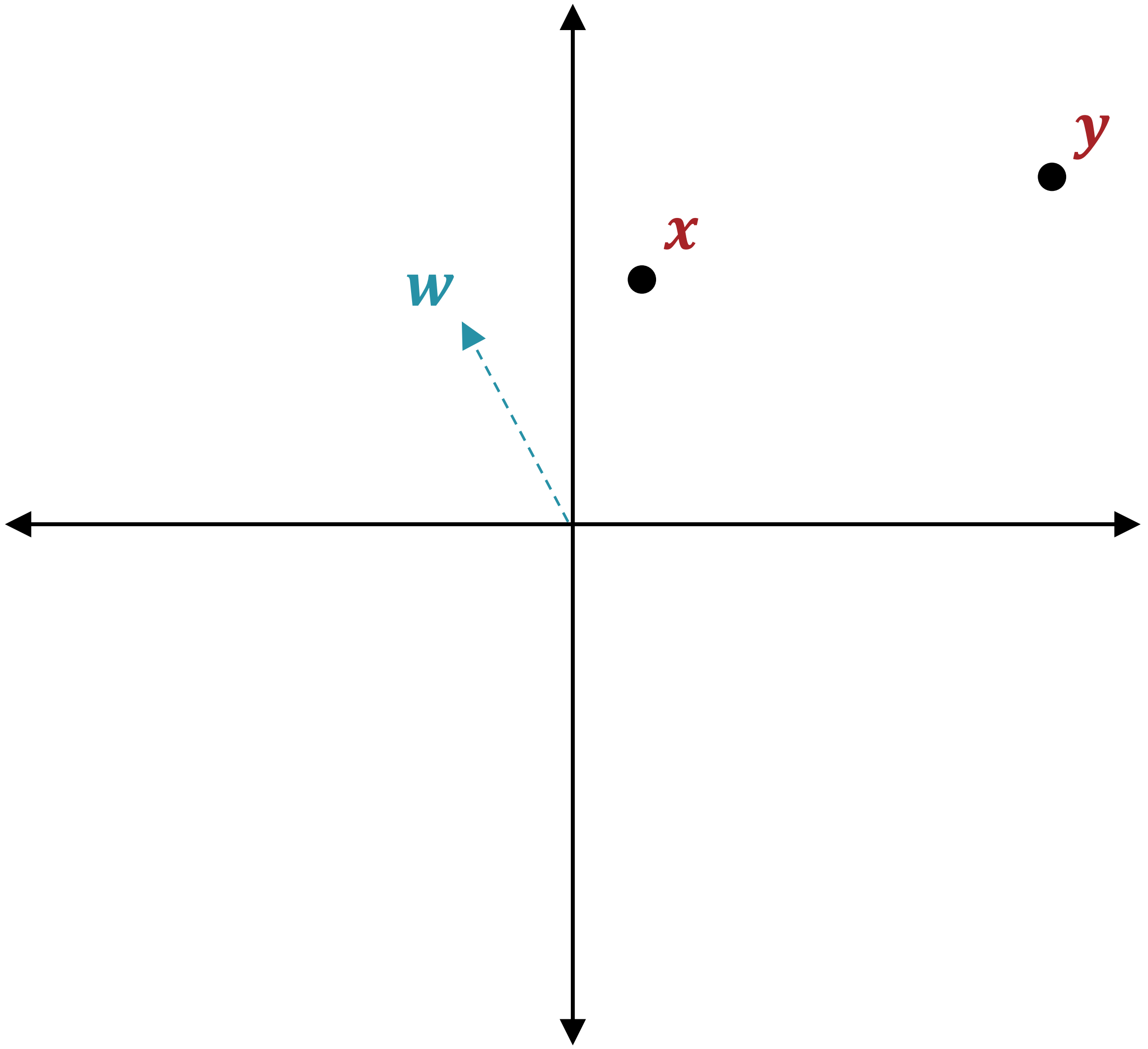
  - $d(\boldsymbol{x}, \boldsymbol{y}) = 1 - \frac{\boldsymbol{x}^T\boldsymbol{y}}{\|\boldsymbol{x}\|_2 \|\boldsymbol{y}\|_2}$, the *cosine distance*
  - For a random weight vector $\boldsymbol{w}$ (e.g., weights sampled independently from a standard Gaussian)

$$P\left(h_{\boldsymbol{w}}(\boldsymbol{x}) = h_{\boldsymbol{w}}(\boldsymbol{y})\right) = 1 - \frac{\theta}{\pi}$$

  - $\mathcal{F}$ is $\left(\epsilon, c\epsilon, 1 - \frac{\arccos 1-\epsilon}{\pi}, 1 - \frac{\arccos 1-c\epsilon}{\pi}\right)$-sensitive

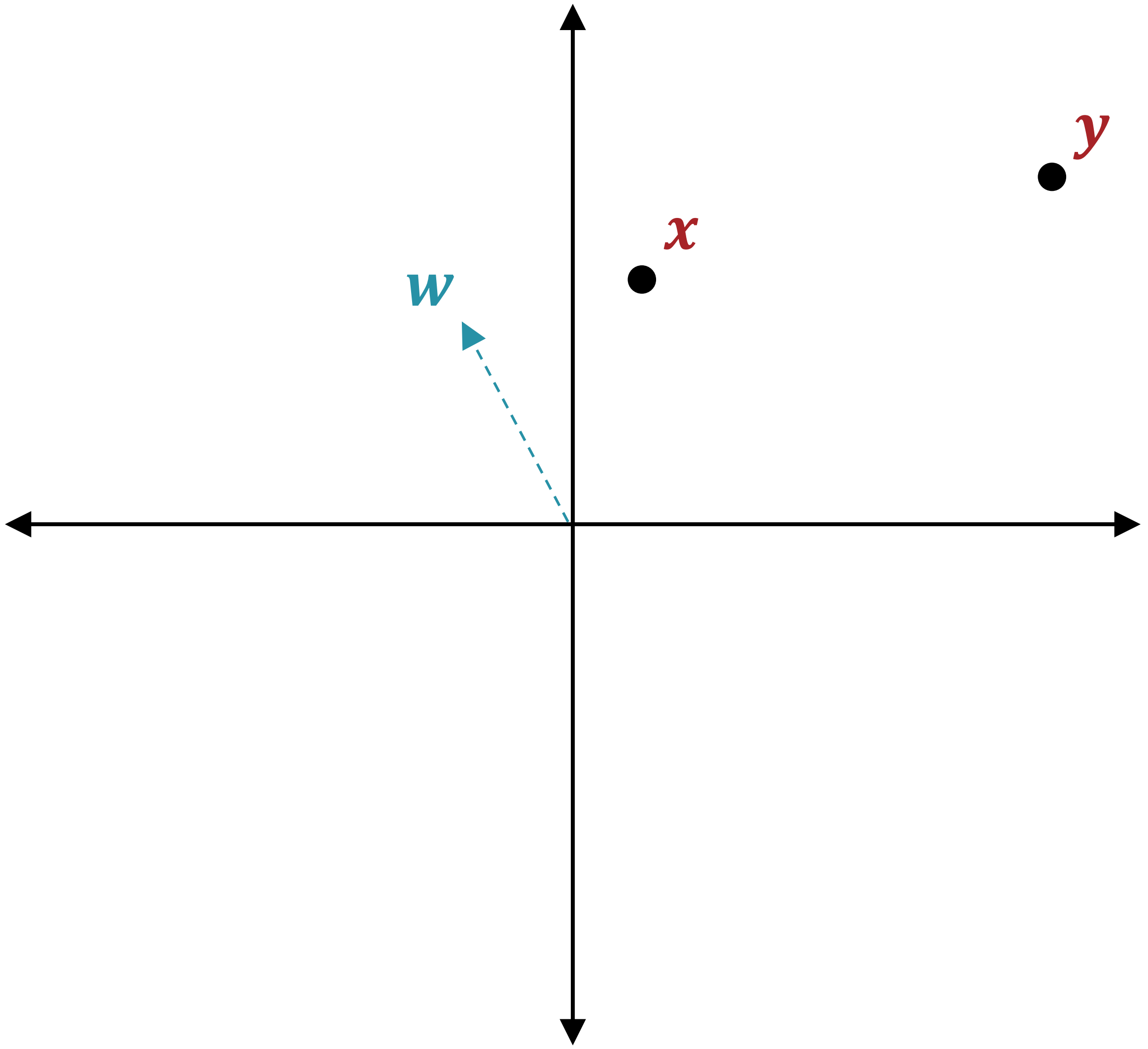
Locality-Sensitive Hashing: Example

- Random projection:



What does the decision boundary corresponding to $\boldsymbol{w}$ look like?

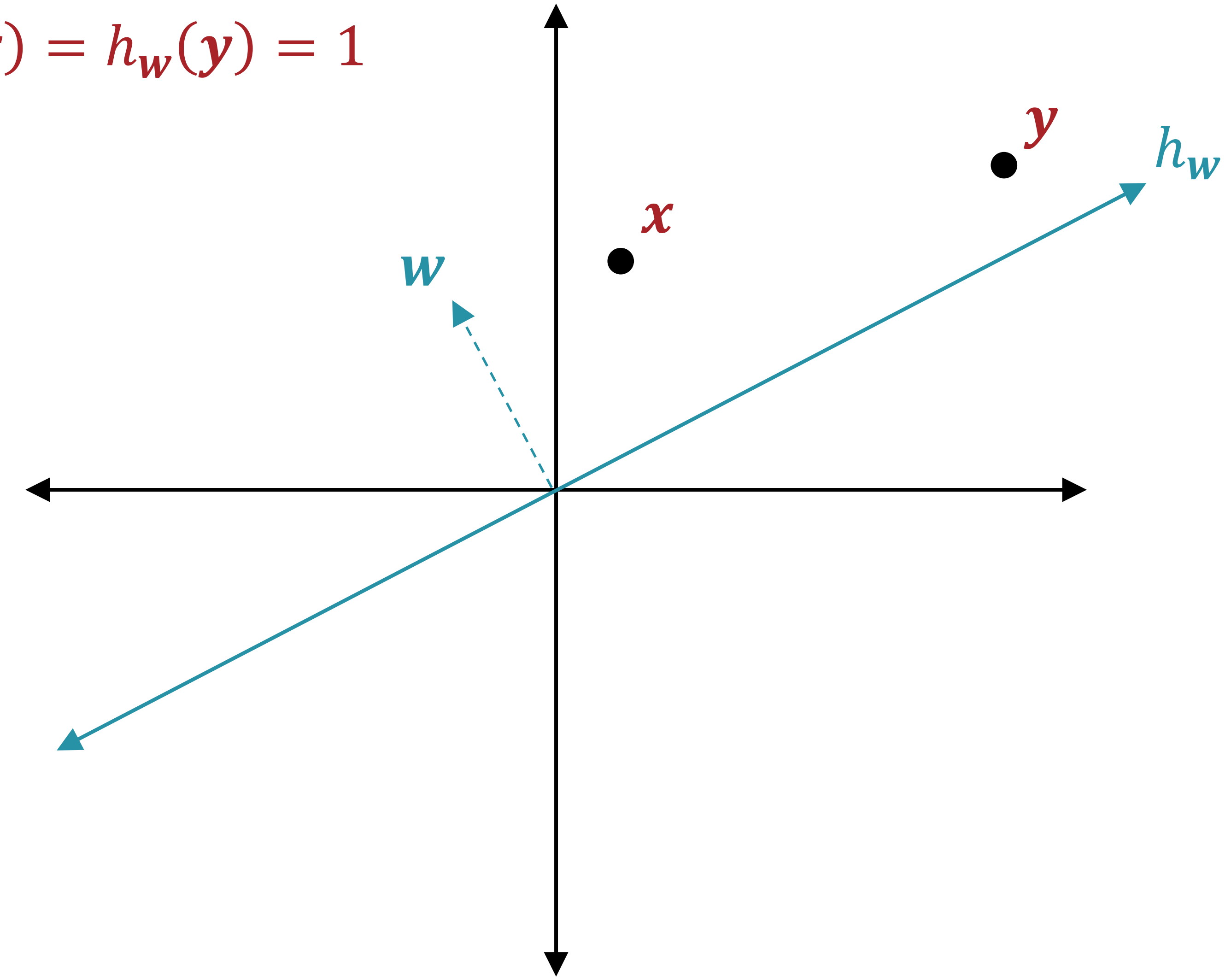
- Random projection:



Locality-Sensitive Hashing: Example

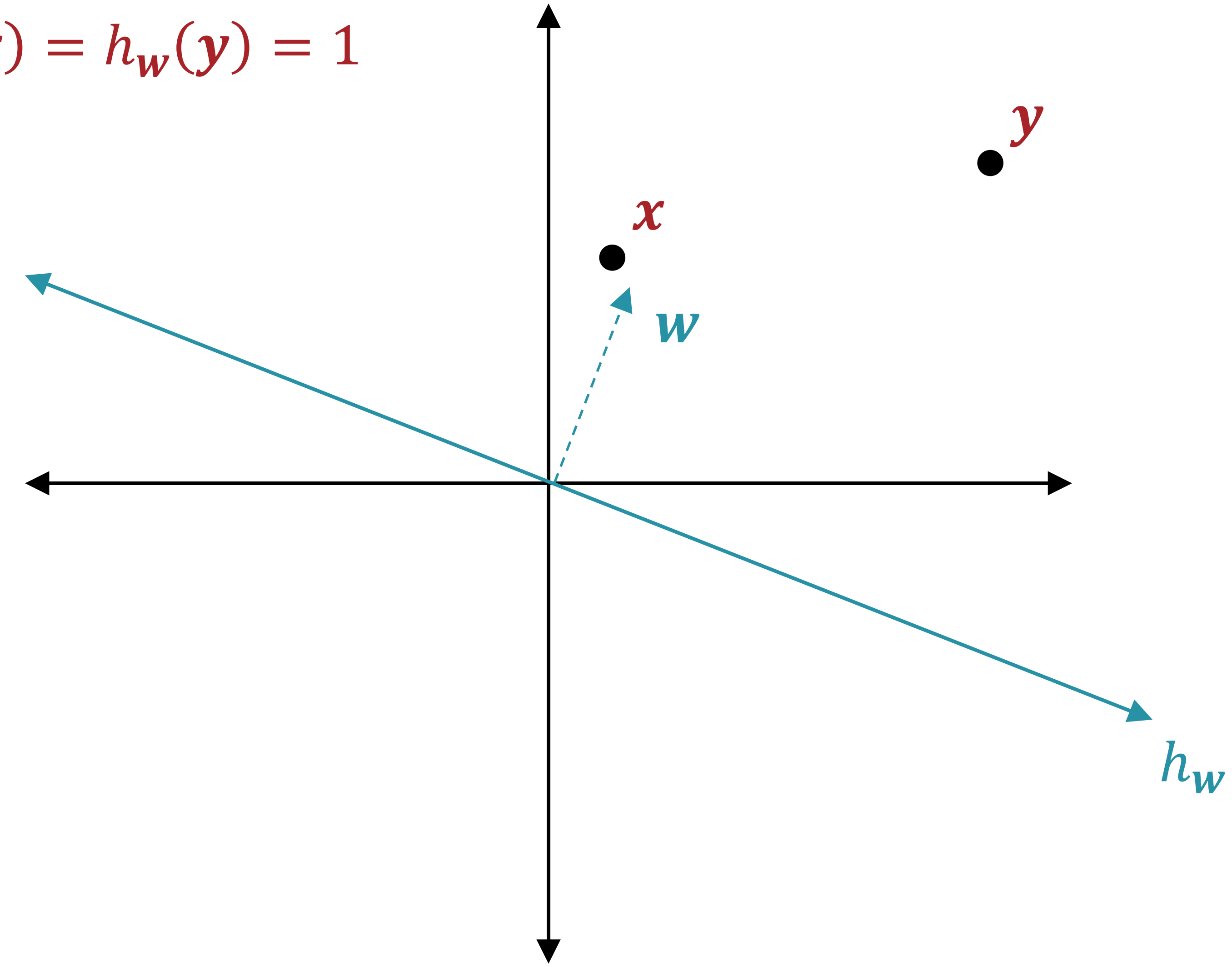
- Random projection:

$h_{\boldsymbol{w}}(\boldsymbol{x}) = h_{\boldsymbol{w}}(\boldsymbol{y}) = 1$



Locality-Sensitive Hashing: Example

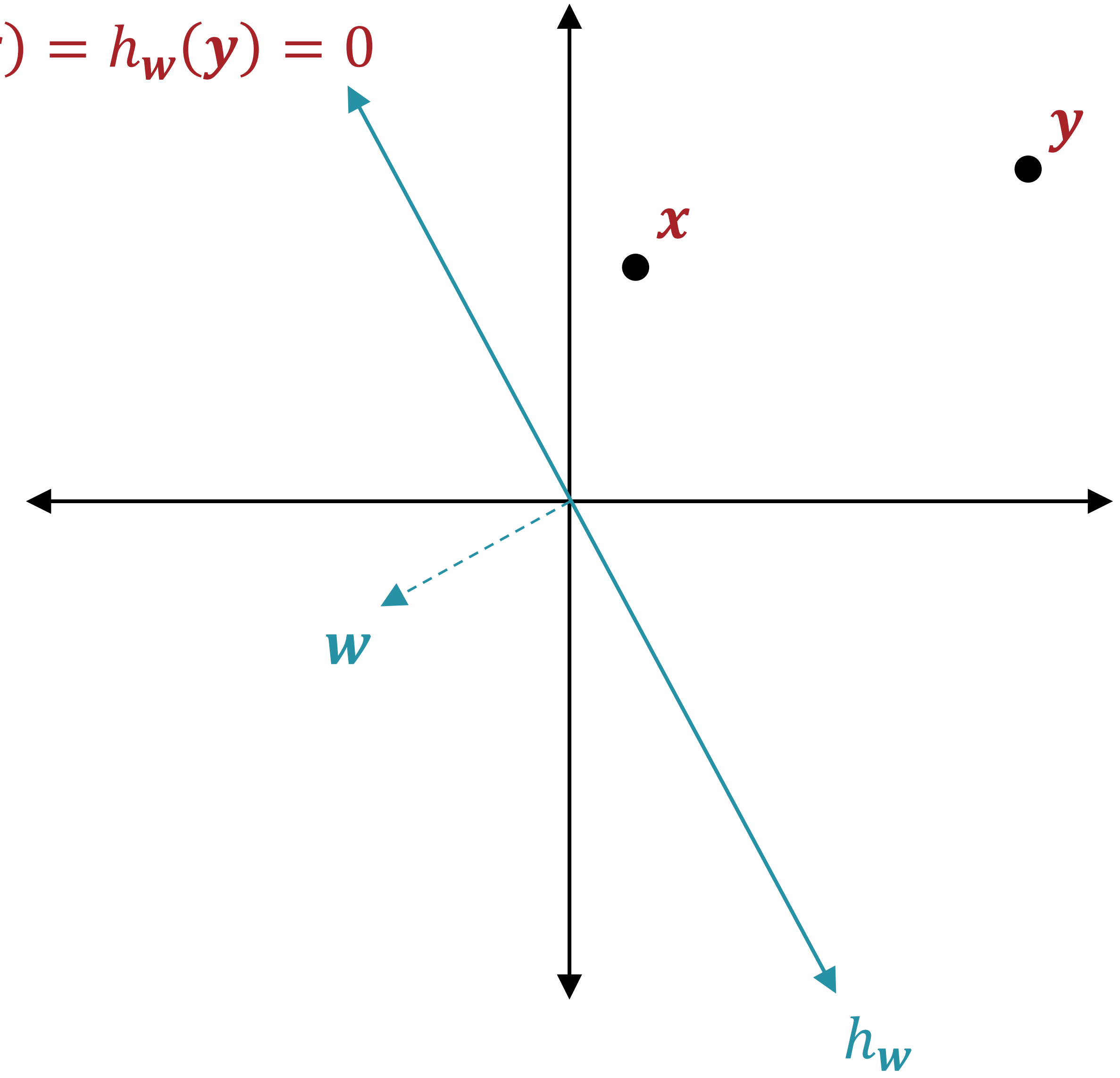
- Random projection:
  $h_{\boldsymbol{w}}(\boldsymbol{x}) = h_{\boldsymbol{w}}(\boldsymbol{y}) = 1$



Locality-Sensitive Hashing: Example

- Random projection:

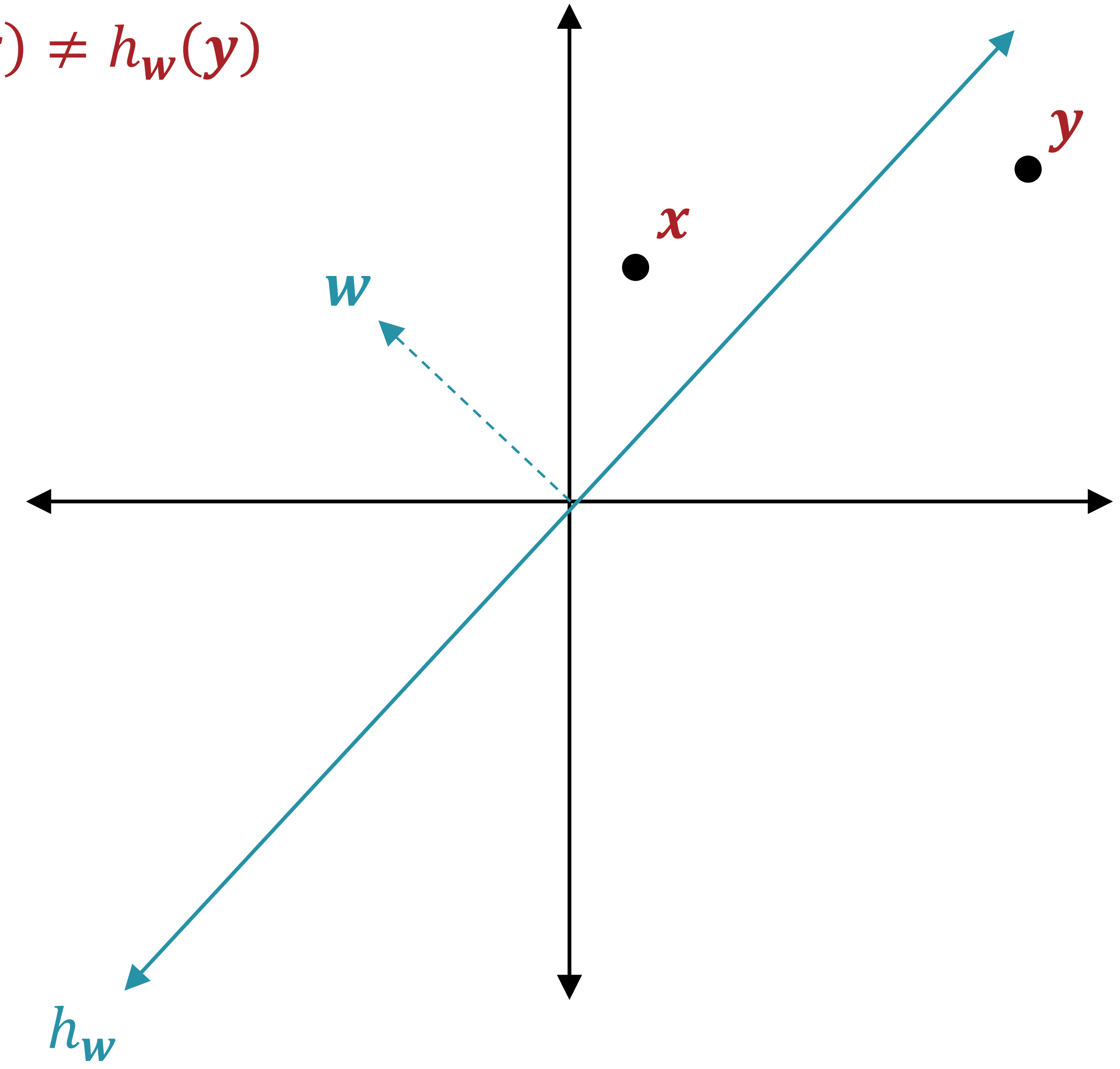
$h_{\boldsymbol{w}}(\boldsymbol{x}) = h_{\boldsymbol{w}}(\boldsymbol{y}) = 0$



Locality-Sensitive Hashing: Example

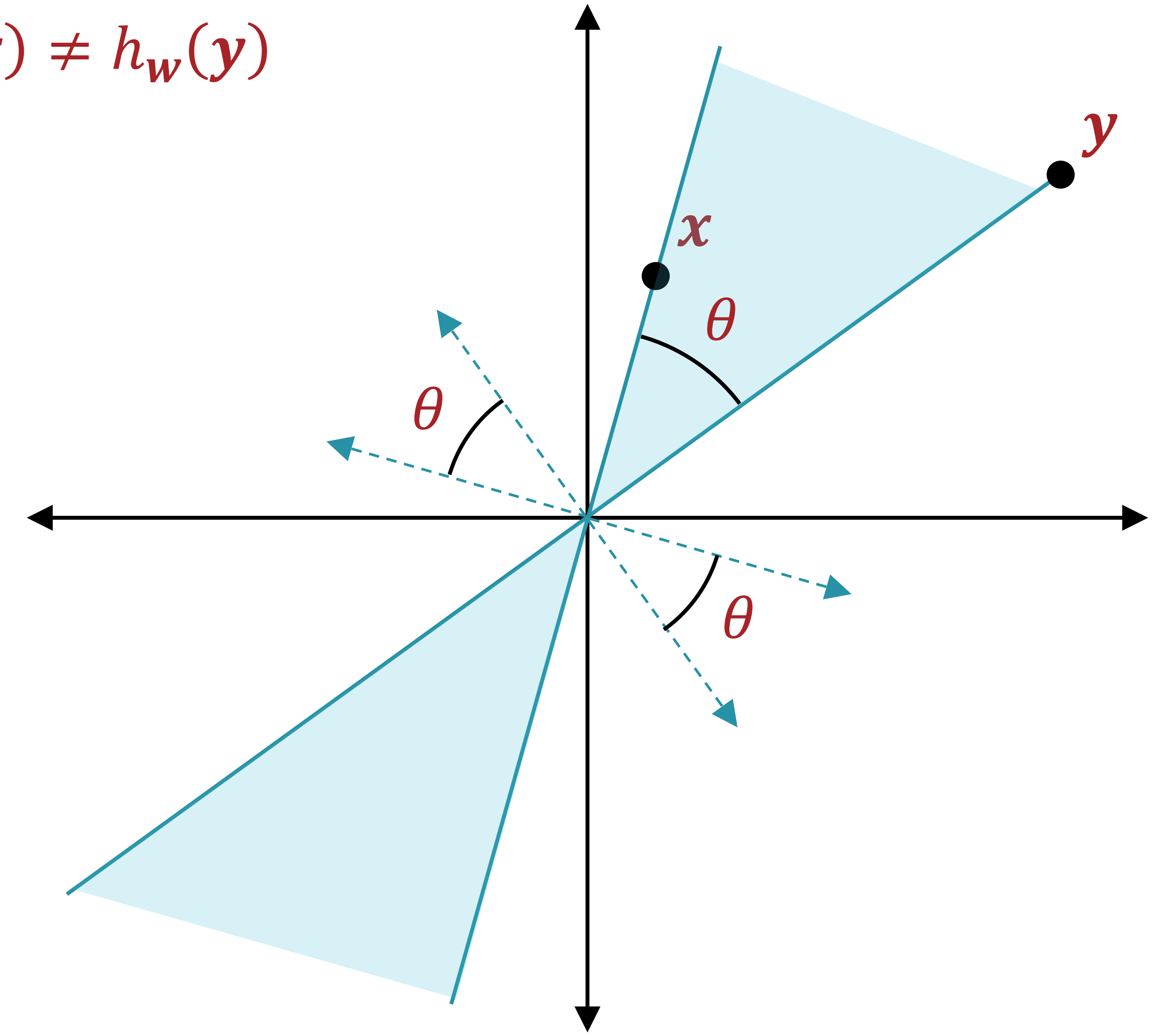
- Random projection:

  $h_{\boldsymbol{w}}(\boldsymbol{x}) \neq h_{\boldsymbol{w}}(\boldsymbol{y})$



Locality-Sensitive Hashing: Example

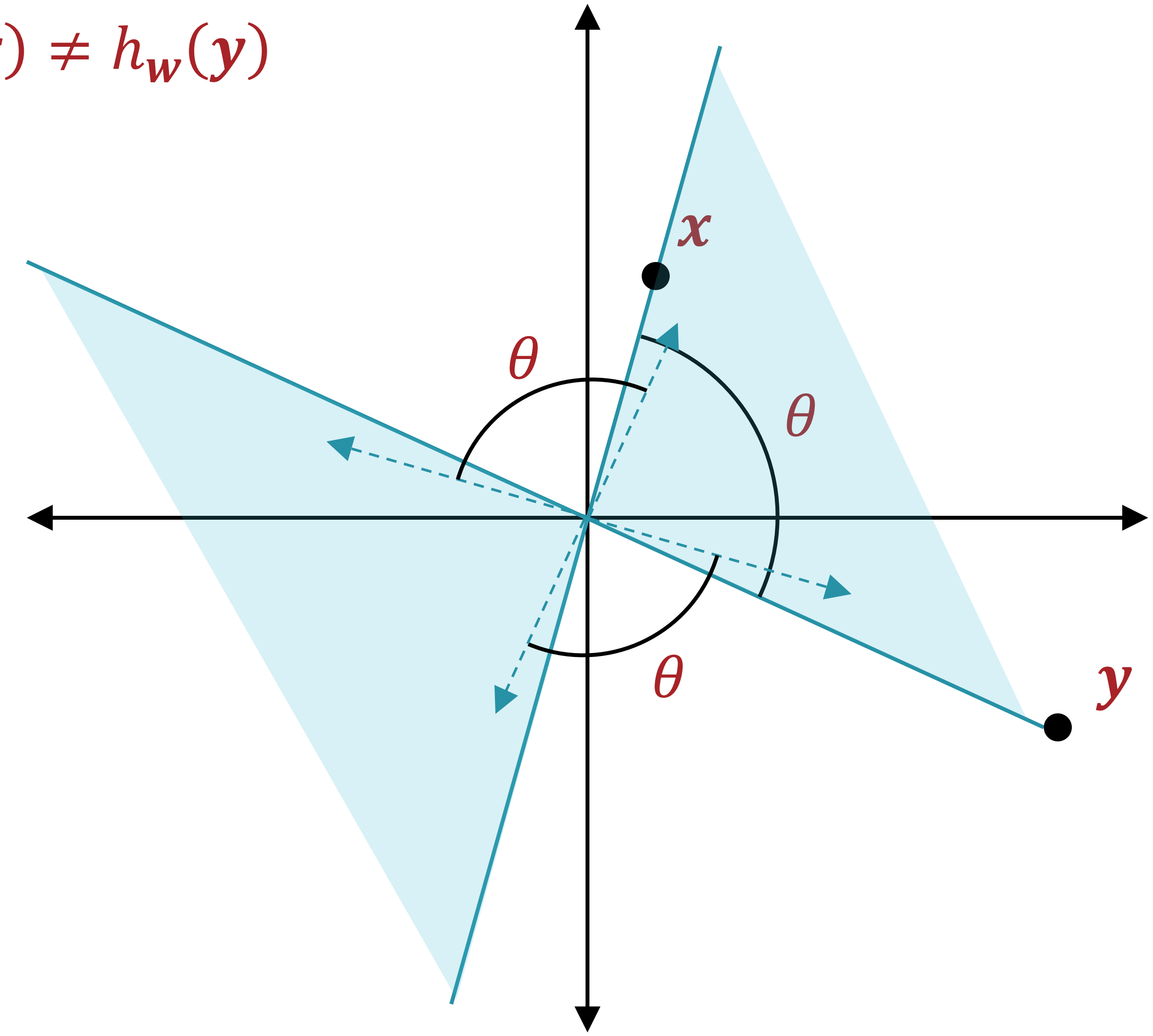
- Random projection:
  $h_{\boldsymbol{w}}(\boldsymbol{x}) \neq h_{\boldsymbol{w}}(\boldsymbol{y})$



Locality-Sensitive Hashing: Example

- Random projection:

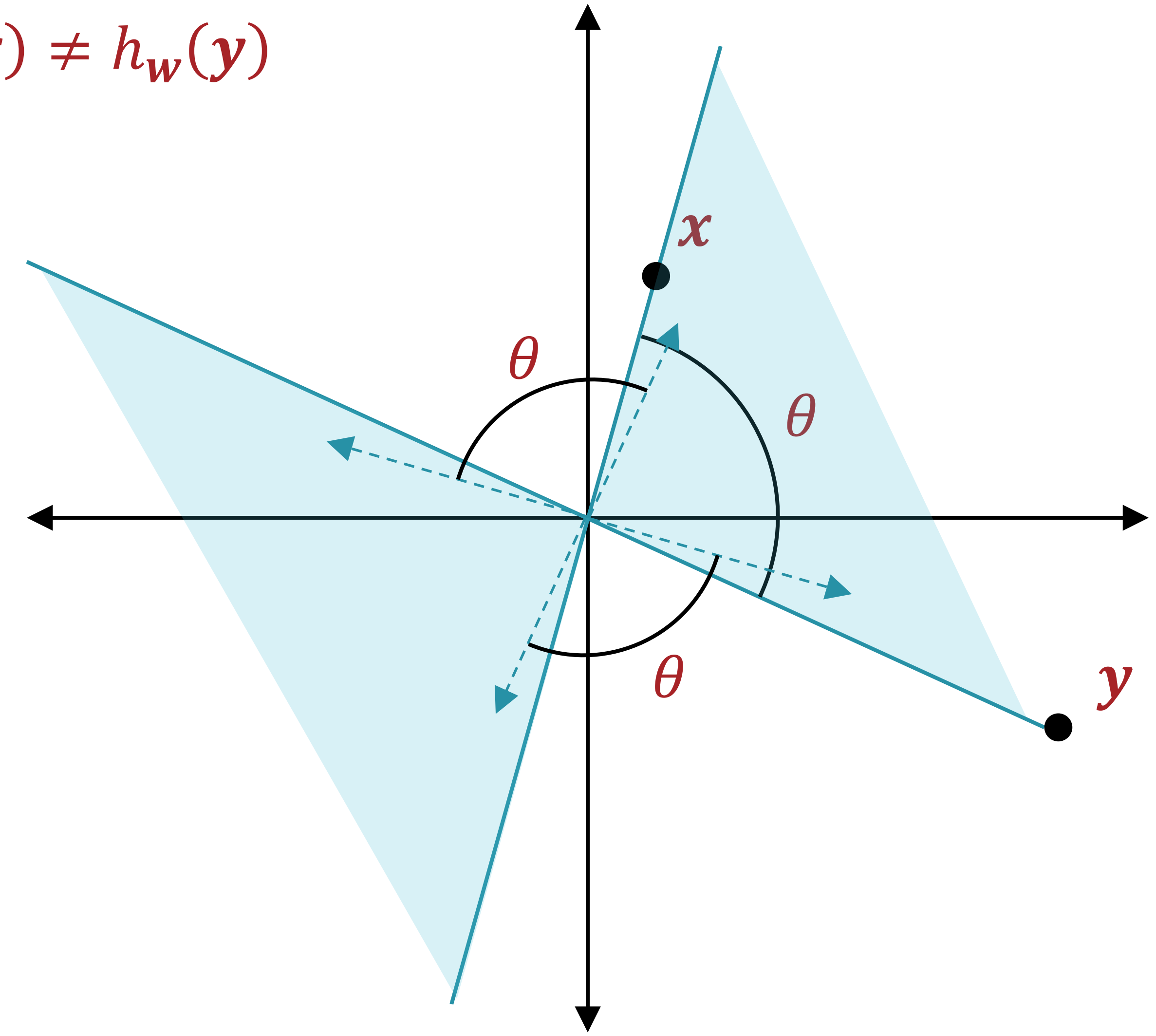
$h_{\boldsymbol{w}}(\boldsymbol{x}) \neq h_{\boldsymbol{w}}(\boldsymbol{y})$



But why stop at just one weight vector?

- Random projection:

  $h_{\boldsymbol{w}}(\boldsymbol{x}) \neq h_{\boldsymbol{w}}(\boldsymbol{y})$



Locality-Sensitive Hashing: Amplification

- Random projections:
  - $m = 1$ so our hash functions map $\mathbb{R}^k$ to $\{0,1\}$
  - Let $\mathcal{F}$ be the set of all linear decision boundaries:

$$\mathcal{F} = \left\{ h : h_{\boldsymbol{w}}(\boldsymbol{x}) = \begin{cases} 1 & \boldsymbol{w}^T\boldsymbol{x} \geq 0 \\ 0 & \text{otherwise} \end{cases} \right\}$$

  - $d(\boldsymbol{x}, \boldsymbol{y}) = 1 - \frac{\boldsymbol{x}^T\boldsymbol{y}}{\|\boldsymbol{x}\|_2 \|\boldsymbol{y}\|_2}$, the *cosine distance*
  - Sample r weight vectors $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_r$ and define a new *bit vector* representation for $\boldsymbol{x}$ as

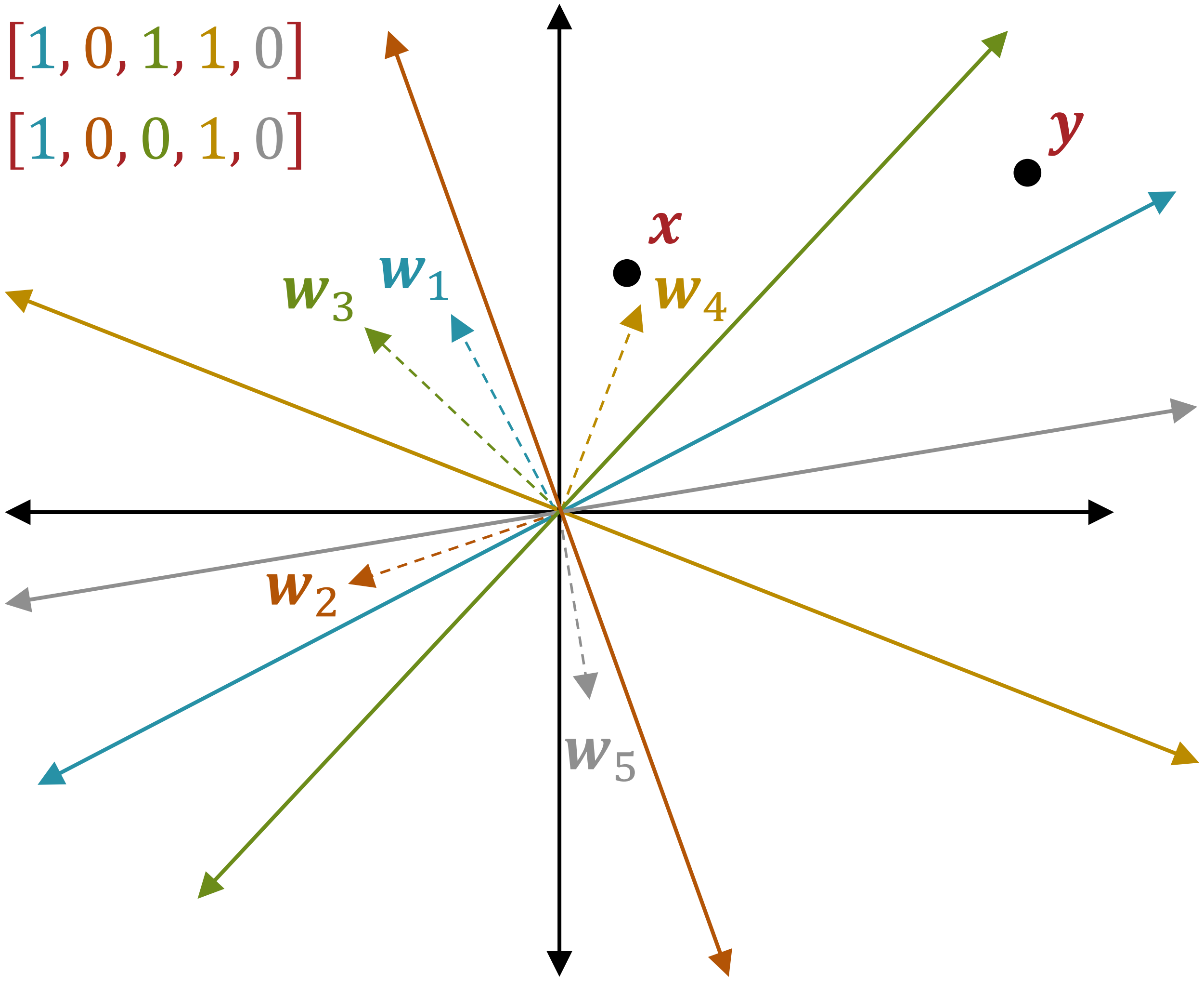
$$\boldsymbol{x}' = \left[h_{\boldsymbol{w}_1}(\boldsymbol{x}), h_{\boldsymbol{w}_2}(\boldsymbol{x}), \ldots, h_{\boldsymbol{w}_r}(\boldsymbol{x})\right]$$

Locality-Sensitive Hashing: Amplification

- Random projections:

  $\boldsymbol{x}' = [1, 0, 1, 1, 0]$

  $\boldsymbol{y}' = [1, 0, 0, 1, 0]$



Locality-Sensitive Hashing: Amplification

- Random projections:
  - $m = 1$ so our hash functions map $\mathbb{R}^k$ to $\{0,1\}$
  - Let $\mathcal{F}$ be the set of all linear decision boundaries:

$$\mathcal{F} = \left\{ h : h_{\boldsymbol{w}}(\boldsymbol{x}) = \begin{cases} 1 & \boldsymbol{w}^T\boldsymbol{x} \geq 0 \\ 0 & \text{otherwise} \end{cases} \right\}$$

  - $d(\boldsymbol{x}, \boldsymbol{y}) = 1 - \frac{\boldsymbol{x}^T\boldsymbol{y}}{\|\boldsymbol{x}\|_2 \|\boldsymbol{y}\|_2}$, the *cosine distance*
  - Sample r weight vectors $\boldsymbol{w}_1, \dots, \boldsymbol{w}_r$ and define a new *bit vector* representation for $\boldsymbol{x}$ as

$$\boldsymbol{x}' = \left[h_{\boldsymbol{w}_1}(\boldsymbol{x}), h_{\boldsymbol{w}_2}(\boldsymbol{x}), \dots, h_{\boldsymbol{w}_r}(\boldsymbol{x})\right]$$

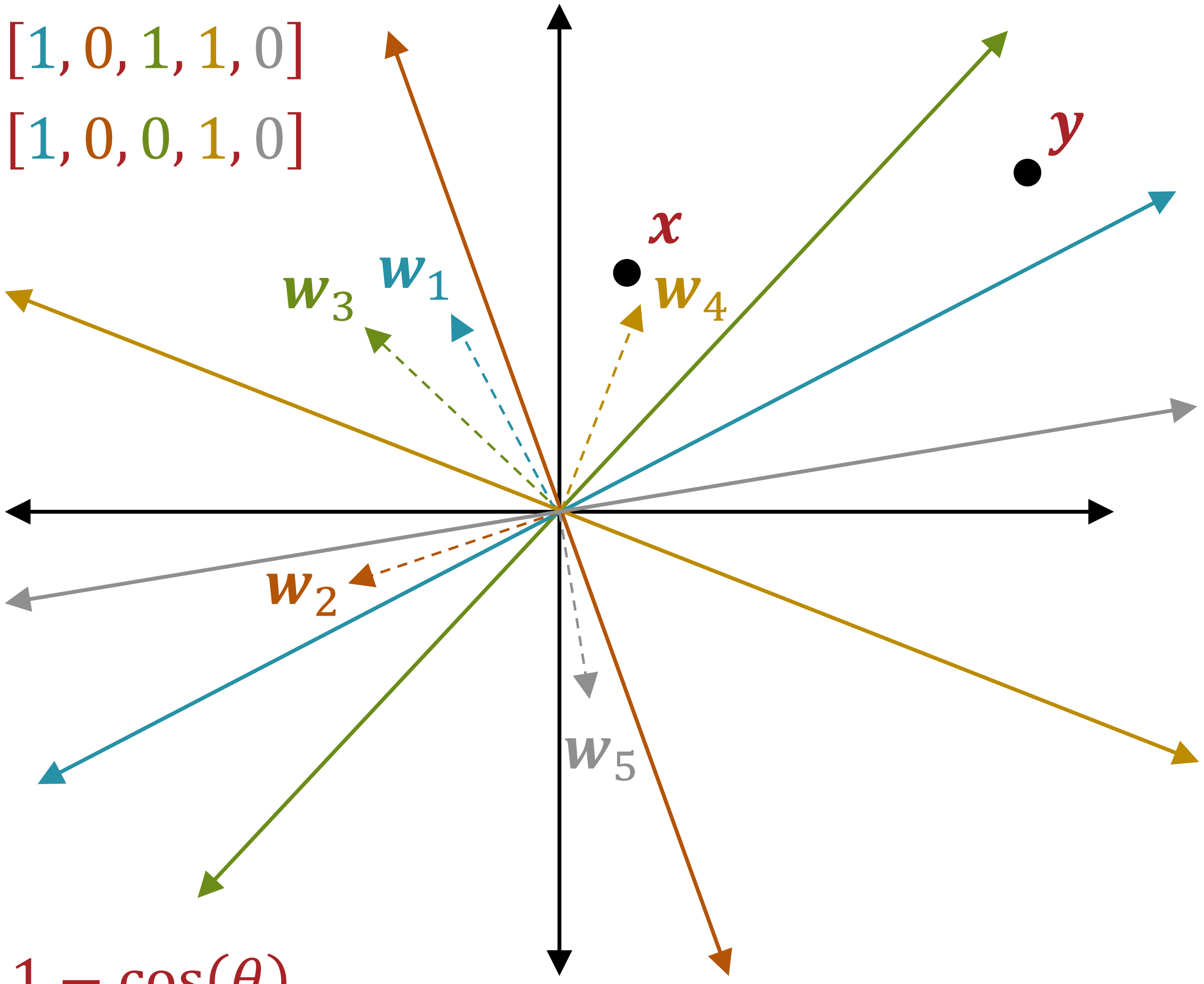
  - The Hamming distance between two bit vectors is

$$d_h(\boldsymbol{x}', \boldsymbol{y}') = \sum_{i=1}^{r} \mathbb{1}(x_i' \neq y_i') = \|\boldsymbol{x}' - \boldsymbol{y}'\|_1$$

Locality-Sensitive Hashing: Amplification

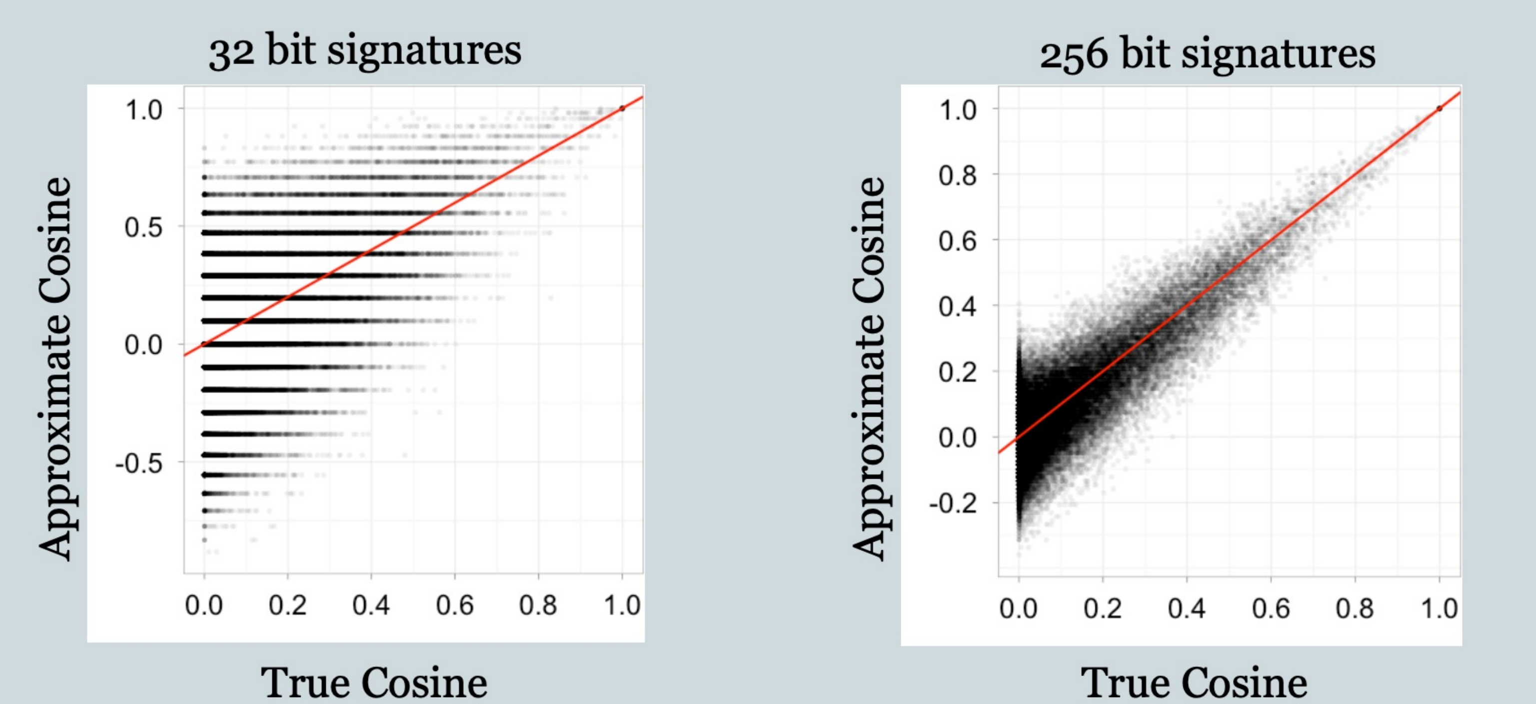
- Random projections:

$$x' = [1, 0, 1, 1, 0]$$
$$y' = [1, 0, 0, 1, 0]$$



$$d(x, y) = 1 - \cos(\theta)$$
$$\approx 1 - \cos\left(\frac{d_h(x', y')}{r}\pi\right)$$

Locality-Sensitive Hashing: Evaluation



Cheaper, less accurate approximation

Expensive, more accurate approximation

Source: https://www.cs.jhu.edu/~vandurme/papers/VanDurmeLallACL10-slides.pdf

Locality-Sensitive Hashing: Summary

- Key idea: use hash functions that map "similar" items to the same function
  - Plot twist: collisions are the goal!
- Random projections can be used to efficiently approximate the cosine distance between points
  - Reduces to computing the Hamming distance between bit vectors consisting of multiple hash functions
- Can be used to efficiently cluster data and compute the nearest neighbors to a query point

Distributed k-NN

1. Distribute training dataset across some number of worker machines
2. Communicate query point $\boldsymbol{x}$ to each worker machine
3. Map: compute distance between $\boldsymbol{x}$ and each training data point
   - If the data is high-dimensional, first compute random projection bit vectors to approximate distances
4. Reduce: return the k nearest neighbors from each worker (and their distances)
5. Driver computes the global k nearest neighbors

Key Takeaways

- Non-numeric data can be a challenge for many machine learning models
- One-hot encodings are impractical for large datasets
  - Feature hashing is a reasonable alternative
  - Hash kernels are unbiased estimators of dot products that preserve distances (with high probability)
- For algorithms where updates are weighted counts of feature values, we can use a weighted version of count-min sketch, an application of hashing with nice theoretical guarantees and good empirical performance
- For algorithms where distances between points is relevant, locality sensitive hashing can be used to efficiently approximate certain distance metrics

Looking unreasonably far ahead…

- Let’s talk about HW4!
- Main goals:
  - Give you all hands-on experience working with cloud computing, specifically Amazon Web Services
  - Work with an actual large dataset in Spark (the entire Million Song Dataset ~280 GB)
  - Execute an end-to-end machine learning pipeline in a distributed manner
- Warning: this homework will be more open-ended than previous ones

Looking unreasonably far ahead…

- Let's talk about HW4!
- Main goals:
  - Give you all hands-on experience working with **cloud computing**, specifically **Amazon Web Services**
  - Work with an actual large dataset in Spark (the entire Million Song Dataset ~280 GB)
  - Execute an end-to-end machine learning pipeline in a distributed manner
- Warning: this homework will be more open-ended than previous ones

Recall: Cloud Computing

- Enables distributed processing by democratizing access to storage and computational resources
- Similar to a public utility, you can access as much or as little of it as you need

Recall: Cloud Computing

- Cloud computing relies on sharing of resources to achieve coherence and typically using a "pay-as-you-go" model which can help in reducing capital expenses but may also lead to unexpected operating expenses for users.

Source: https://en.wikipedia.org/wiki/Cloud_computing

Data Centers: Amazon Data Center in Ashburn, Virginia

Source: https://www.geekwire.com/2018/crystal-city-just-latest-addition-amazons-sprawling-northern-virginia-cloud-operation/

Data Centers: Facebook Data Center in Prineville, Oregon

Source: https://www.geekwire.com/2018/crystal-city-just-latest-addition-amazons-sprawling-northern-virginia-cloud-operation/

Data Centers: Amazon Data Center in Brétigny-sur-Orge, France

Source: https://www.datacenterdynamics.com/en/news/br%C3%A9tigny-sur-orge-residents-protest-proposed-amazon-web-services-data-center/

Cloud Computing: Pros and Cons

- Pros:
  - On-demand computing/storage allows for scalability and elasticity (only pay for what you use)
  - Eliminates infrastructure and **maintenance** costs
- Cons:
  - Depending on usage, overall (monetary) costs may be higher than being “on-perm”, i.e., buying your own compute resources
  - Data gravity – difficult to switch providers once a lot of resources have been committed to a single one
  - Data privacy – do you really trust Amazon?
  - Environmental impacts – densely packed computing resources require a lot of energy to run and **cool**

Source: https://datacenterfrontier.com/florida-data-centers-brace-for-powerful-hurricane-ian/

Source: https://www.datacenterknowledge.com/cloud/apple-recovers-massive-outage

Cloud Computing: Maintenance

Florida Data Centers Brace for Powerful Hurricane Ian

BY RICH MILLER - SEPTEMBER 28, 2022 — LEAVE A COMMENT

Data Center Fire: Google Suffers 'Electrical Incident,' 3 Injured

News of yet another data center fire in just two years: OVH in 2021 and now Google. Here's how to make sure you're not next.

Apple Recovers From Massive Outage

Yet another example
rely heavily on the clc

Jeff Baumgartner | Mar 22, :

TECH • FACEBOOK

Facebook's outage cost the company nearly $100 million in revenue

BY CHRIS MORRIS

October 4, 2021 4:30 PM EDT

Updated October 5, 2021 11:25 AM EDT

Source: https://www.datacenterknowledge.com/google-alphabet/data-center-fire-google-suffers-electrical-incident-3-injured

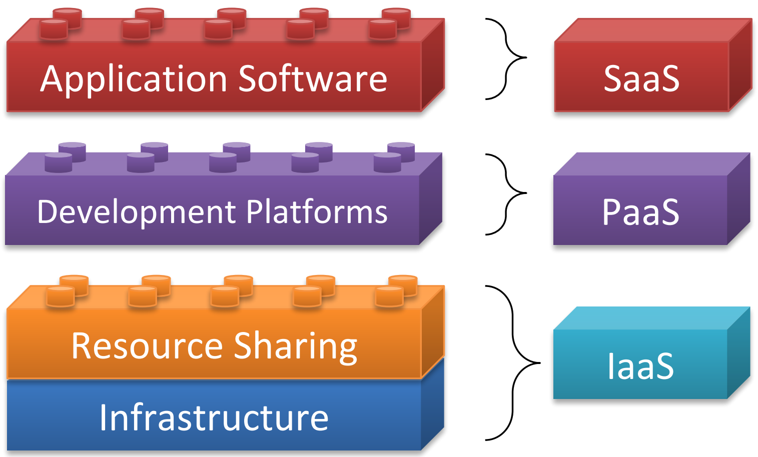
Source: https://www.yahoo.com/now/facebook-outage-already-cost-company-203040137.html

Cloud Computing: Pros and Cons

- Pros:
  - On-demand computing/storage allows for scalability and elasticity (only pay for what you use)
  - Eliminates infrastructure and **maintenance** costs
- Cons:
  - Depending on usage, overall (monetary) costs may be higher than being “on-perm”, i.e., buying your own compute resources
  - Data gravity – difficult to switch providers once a lot of resources have been committed to a single one
  - Data privacy – do you really trust Amazon?
  - Environmental impacts – densely packed computing resources require a lot of energy to run and **cool**

Cloud Computing: Services

- Cloud computing services can be thought of as different levels of abstraction
  - Software as a service (SaaS)
  - Platform as a service (PaaS)
  - Infrastructure as a service (IaaS)



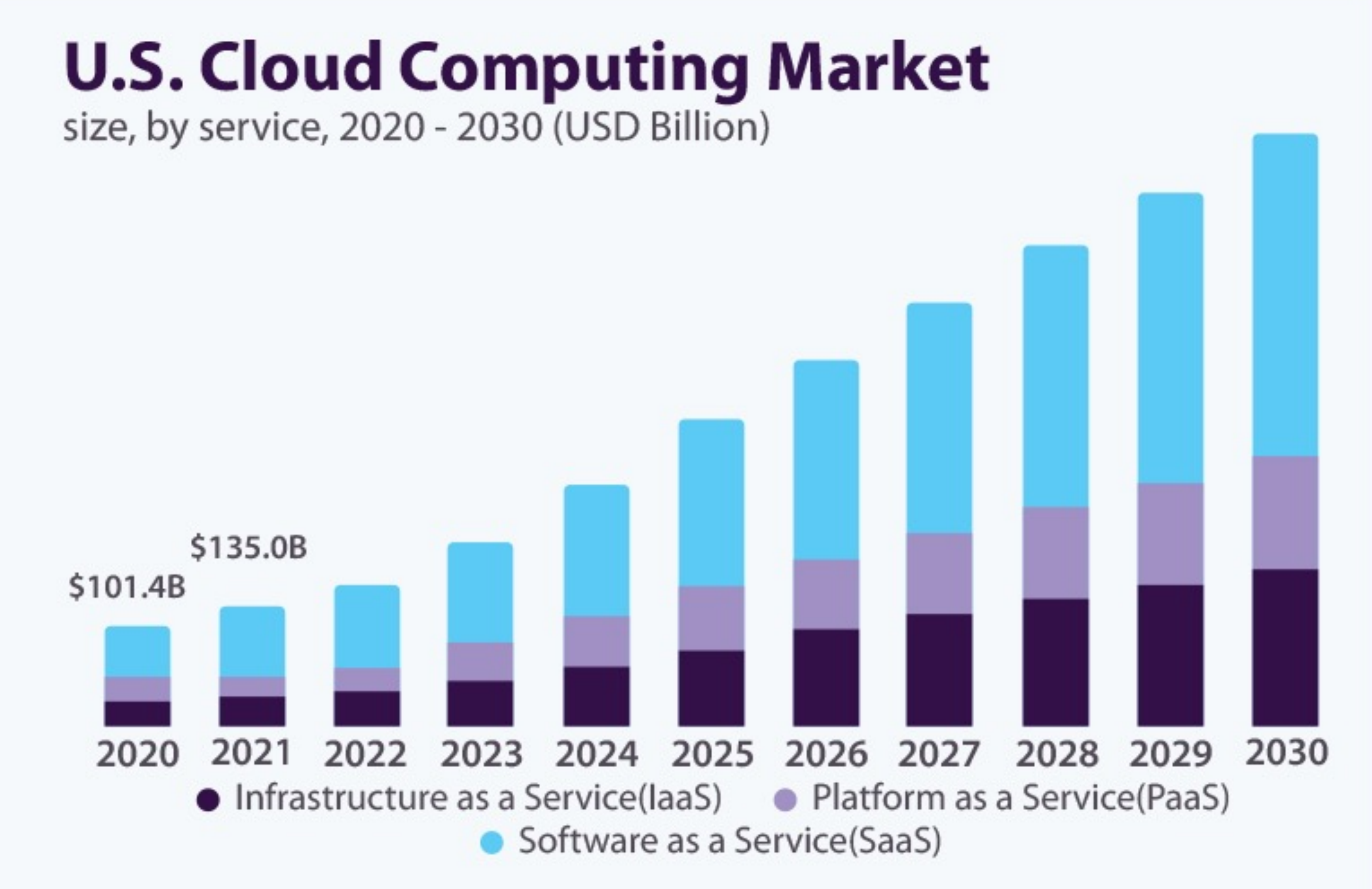
e.g., Gmail

e.g., Amazon EMR

e.g., Amazon EC2

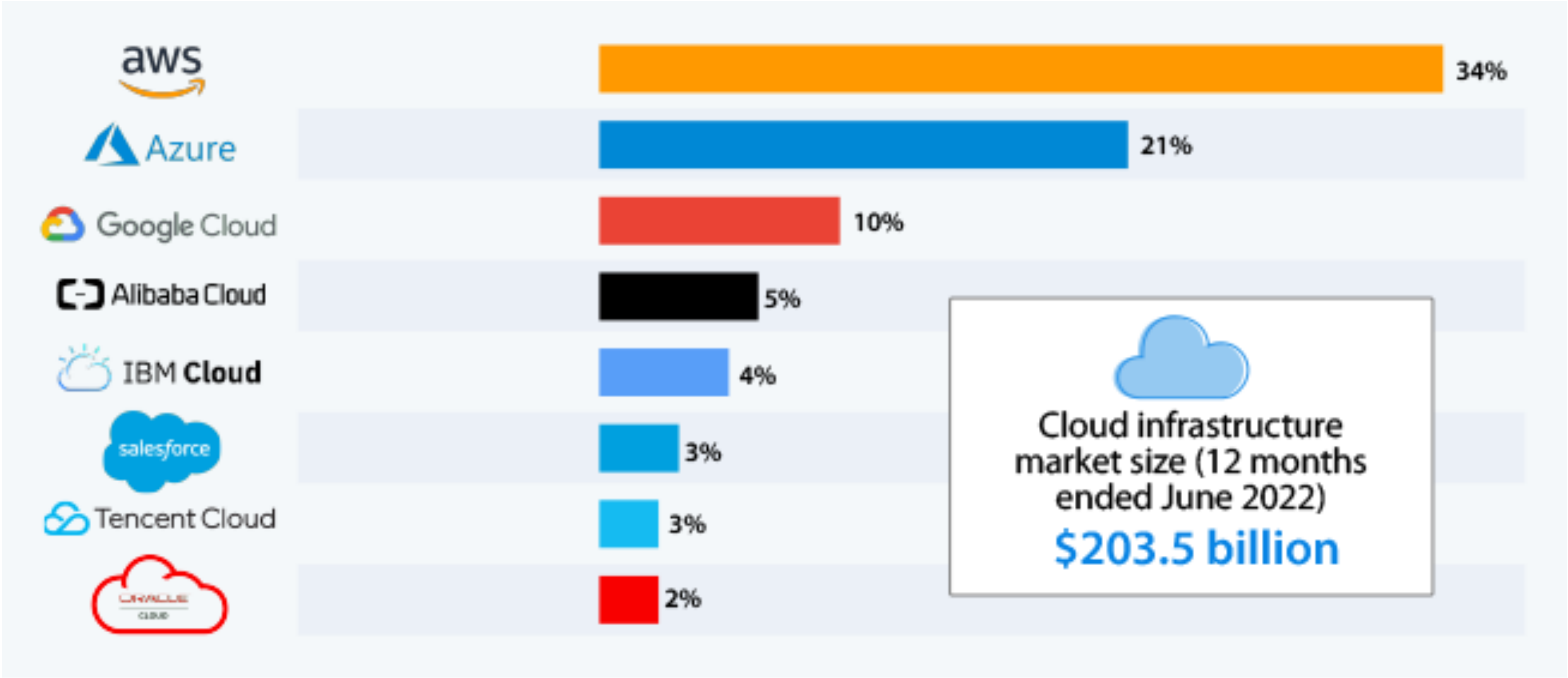
Figures courtesy of Majd Sakr

Cloud Computing: Market



Source: https://www.wpoven.com/blog/cloud-market-share/

Cloud Computing: Market



Source: https://www.wpoven.com/blog/cloud-market-share/