10-605/805 – ML for Large Datasets Lecture 9: Hashing

Henry Chai

9/27/22

Front Matter

- HW3 released 9/23, due 10/4 at 11:59 PM
- Midterm exam on 10/11, two weeks from today!
  - All topics up to and including this Thursday's lecture are in-scope
  - Closed-everything except for a "cheat-sheet", one letter-sized sheet of paper upon which you can put *anything* on the *back and front*
  - 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

Non-numeric Features

- Most of the machine learning methods we've considered in this class require numeric features

Recall: PCA

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}\right)\right\}_{i=1}^{n}, r$

1. Center the data
   A. Optionally, normalize the data by features so that all features are of the same scale
2. Compute the covariance matrix $C_X = X^T X$ ($O(nk^2)$)
3. Collect the top r eigenvectors (corresponding to the r largest eigenvalues), $P \in \mathbb{R}^{k \times r}$ ($O(k^3)$)
4. Project the data into the space defined by P, $Z = XP$ ($O(nkr)$)

Recall: Linear Regression

- A type of supervised learning
- Given:
  - some labelled training dataset $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}$
  - $\ell(y, y') = (y - y')^2$
  - $\mathcal{F}$ = all functions of the form $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}$

1 implicitly prepended

the goal is to find

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} \sum_{i=1}^{n} \left(\boldsymbol{w}^T\boldsymbol{x}^{(i)} - y^{(i)}\right)^2$$

Recall: Logistic Regression

- A type of supervised learning
- Given:
  - some labelled training dataset $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}$
  - the log loss
  - $\mathcal{F}$ = the set of all linear decision boundaries

the goal is to find

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} -\sum_{i=1}^{n}\left(y^{(i)} \log P\left(Y=1 \middle| \boldsymbol{x}^{(i)}, \boldsymbol{w}\right)\right)+\left(1-y^{(i)}\right) \log P\left(Y=0 \middle| \boldsymbol{x}^{(i)}, \boldsymbol{w}\right)$$

$$= \underset{\boldsymbol{w}}{\operatorname{argmin}} -\sum_{i=1}^{n} y^{(i)} \log \frac{P\left(Y=1 \middle| \boldsymbol{x}^{(i)}, \boldsymbol{w}\right)}{P\left(Y=0 \middle| \boldsymbol{x}^{(i)}, \boldsymbol{w}\right)}+\log P\left(Y=0 \middle| \boldsymbol{x}^{(i)}, \boldsymbol{w}\right)$$

$$= \underset{\boldsymbol{w}}{\operatorname{argmin}} -\sum_{i=1}^{n} y^{(i)} \boldsymbol{w}^{T} \boldsymbol{x}^{(i)}-\log \left(1+\exp \left(\boldsymbol{w}^{T} \boldsymbol{x}^{(i)}\right)\right)$$

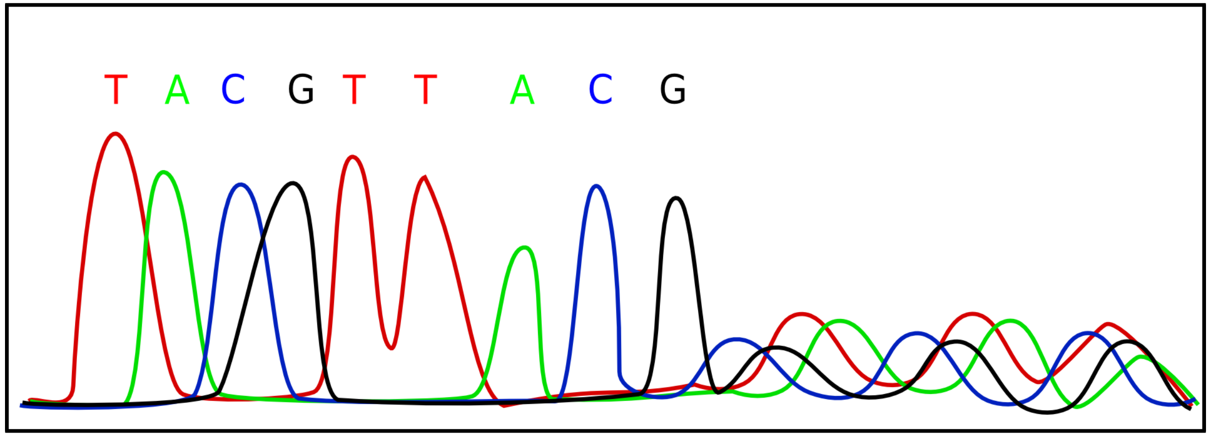
Non-numeric Features

- Most of the machine learning methods we've considered in this class require numeric features
  - Notable exception: decision trees
- However, real-world data is frequently non-numeric e.g.
  - Raw text data

CHAI	Easy course, taught well
CHAI	homework takes way too long
CHAI	See above.
CHAI	Great
CHAI	Too much work
CHAI	This course had a lot of problems but none of them were Henrys fault.

Non-numeric Features

- Most of the machine learning methods we've considered in this class require numeric features
  - Notable exception: decision trees
- However, real-world data is frequently non-numeric e.g.
  - Raw text data
  - Genomic data



Non-numeric Features

- Most of the machine learning methods we’ve considered in this class require numeric features
  - Notable exception: decision trees
- However, real-world data is frequently non-numeric e.g.
  - Raw text data
  - Genomic data
  - Demographic data

2020 Census Results

Learn more about the data from the 2020 Census, including apportionment counts, redistricting data, and public use files.

2020 Census Data Quality

We check the quality of our work every step of the way. When we release data, we make sure they meet our quality standards.

A Look Back at the 2020 Census

It all began on a January afternoon, in the remote Alaskan village of Toksook Bay...

Non-numeric Features

- We’ll consider two types of non-numeric features:
  - Ordinal
    - Values have an intrinsic ordering; spacing between values may be inconsistent
    - E.g. Likert scale ratings: “poor”, “fair”, “acceptable”, “good”, “excellent”
  - Categorical
    - Values do not have an intrinsic relationship with other values
    - E.g. demographic data such as occupation, marital status, gender

Handling Non-numeric Features

- High-level approach: convert non-numeric features to numeric ones
- Idea: assign every possible value a feature can take on to a number e.g. for Likert scale ratings
  - “poor” $\rightarrow 0$
  - “fair” $\rightarrow 1$
  - “acceptable” $\rightarrow 2$
  - “good” $\rightarrow 3$
  - “excellent” $\rightarrow 4$
  - ✔ Preserves ordering for ordinal features
  - Ascribes an equivalence between adjacent ratings

Handling Non-numeric Features

- High-level approach: convert non-numeric features to numeric ones
- Idea: assign every possible value a feature can take on to a number e.g. for occupations
  - "software engineer" $\rightarrow 0$
  - "professor" $\rightarrow 1$
  - "NFL quarterback" $\rightarrow 2$
  - "florist" $\rightarrow 3$

$\vdots$

Introduces spurious relationships between values

One-hot Encoding

- High-level approach: convert non-numeric features to numeric ones
- Better Idea: assign every possible value a feature can take on to a *binary feature* in a new representation
  - "software engineer" $\rightarrow [1\ 0\ 0\ 0\ \cdots]$
  - "professor" $\rightarrow [0\ 1\ 0\ 0\ \cdots]$
  - "NFL quarterback" $\rightarrow [0\ 0\ 1\ 0\ \cdots]$
  - "florist" $\rightarrow [0\ 0\ 0\ 1\ \cdots]$

  $\vdots$

✔ Automatically handles missing values

One-hot Encoding: Example

- Consider this medical dataset

Family History	Blood Pressure	Cholesterol
Yes	Low	Normal
	Medium	Normal
No	Low	Abnormal
Yes		Normal
Yes	High	Abnormal

- Map each feature value to a new feature

Family History = “Yes”	Family History = “No”	Blood Pressure = “Low”	Blood Pressure = “Medium”	Blood Pressure = “High”	Cholesterol = “Normal”	Cholesterol = “Abnormal”

One-hot Encoding: Example

- Consider this medical dataset

Family History	Blood Pressure	Cholesterol
Yes	Low	Normal
	Medium	Normal
No	Low	Abnormal
Yes		Normal
Yes	High	Abnormal

- Map each feature value to a new feature

Family History = “Yes”	Family History = “No”	Blood Pressure = “Low”	Blood Pressure = “Medium”	Blood Pressure = “High”	Cholesterol = “Normal”	Cholesterol = “Abnormal”
1	0	1	0	0	1	0

One-hot Encoding: Example

- Consider this medical dataset

Family History	Blood Pressure	Cholesterol
Yes	Low	Normal
	Medium	Normal
No	Low	Abnormal
Yes		Normal
Yes	High	Abnormal

- Map each feature value to a new feature

Family History = “Yes”	Family History = “No”	Blood Pressure = “Low”	Blood Pressure = “Medium”	Blood Pressure = “High”	Cholesterol = “Normal”	Cholesterol = “Abnormal”
1	0	1	0	0	1	0
0	0	0	1	0	1	0

One-hot Encoding: Example

- Consider this medical dataset

Family History	Blood Pressure	Cholesterol
Yes	Low	Normal
	Medium	Normal
No	Low	Abnormal
Yes		Normal
Yes	High	Abnormal

- Map each feature value to a new feature

Family History = "Yes"	Family History = "No"	Blood Pressure = "Low"	Blood Pressure = "Medium"	Blood Pressure = "High"	Cholesterol = "Normal"	Cholesterol = "Abnormal"
1	0	1	0	0	1	0
0	0	0	1	0	1	0
0	1	1	0	0	0	1
1	0	0	0	0	1	0
1	0	0	0	1	0	1

Insight: The resulting matrix is mostly 0's!

Idea: Just store the index and value of the non-zero entries

- Consider this medical dataset

Family History	Blood Pressure	Cholesterol
Yes	Low	Normal
	Medium	Normal
No	Low	Abnormal
Yes		Normal
Yes	High	Abnormal

- Map each feature value to a new feature

Family History = "Yes"	Family History = "No"	Blood Pressure = "Low"	Blood Pressure = "Medium"	Blood Pressure = "High"	Cholesterol = "Normal"	Cholesterol = "Abnormal"
1	0	1	0	0	1	0
0	0	0	1	0	1	0
0	1	1	0	0	0	1
1	0	0	0	0	1	0
1	0	0	0	1	0	1

Sparse Representation of One-hot Encoding: Example

- Consider this medical dataset

Family History	Blood Pressure	Cholesterol
Yes	Low	Normal
	Medium	Normal
No	Low	Abnormal
Yes		Normal
Yes	High	Abnormal

- Map each feature value to a new feature

0	1	2	3	4	5	6
1	0	1	0	0	1	0
0	0	0	1	0	1	0
0	1	1	0	0	0	1
1	0	0	0	0	1	0
1	0	0	0	1	0	1

Sparse Representation of One-hot Encoding: Example

- Consider this medical dataset

Family History	Blood Pressure	Cholesterol
Yes	Low	Normal
	Medium	Normal
No	Low	Abnormal
Yes		Normal
Yes	High	Abnormal

- Map each feature value to a new feature

0	1	2	3	4	5	6
1	0	1	0	0	1	0

- Store just the index and value of the non-zero entries
    - [(0,1), (2,1), (5,1)]

Why do we need to store the values?

- Consider this medical dataset

Family History	Blood Pressure	Cholesterol
Yes	Low	Normal
	Medium	Normal
No	Low	Abnormal
Yes		Normal
Yes	High	Abnormal

- Map each feature value to a new feature

0	1	2	3	4	5	6
1	0	1	0	0	1	0

- Store just the index and value of the non-zero entries
  - [(0,1), (2,1), (5,1)]

Recall: Bag of Words Model

- Running Example: Sentiment analysis of course evaluations
- Training dataset
  - Feature engineering - transform observations into a form appropriate for the machine learning method
  - Example: bag of words model

this course had lot problems but none of the course problems were henry fault

Vocabulary	
but	1
course	2
easy	0
fault	1
great	0
henry	1
homework	0
long	0
⋮	⋮

Sparse Representation of Bag of Words Model

- English language consists of > 1M words
- Storing 1M course evaluations using a Bag of Words model over this vocabulary:
  - Counts represented as `int`s (4 bytes per `int`)
  - $4 * 10^6 * 10^6 = 4$ TB
- Most course evaluations don't use every word in the English language
  - Assume 0.01% non-zero entries
  - Each non-zero entry requires storing 2 `int`s
  - $2 * 4 * 0.0001 * 10^6 * 10^6 = 0.8$ GB (5000x reduction)

One-hot Encoding: Summary

- Pros
  - Inherently sparse representation
  - Naturally handles missing values
- Cons
  - Greatly increases the dimensionality of the data
    - Makes learning less efficient (if the complexity of the algorithm depends on k) and more difficult (many new features will be of limited predictive value)
    - Increases the tendency of models to overfit
    - Increases communication costs in distributed settings
  - Can (potentially) create multicollinearity between features

Alright, so let's just do some Dimensionality Reduction, no big deal right?

- Pros
  - Inherently sparse representation
  - Naturally handles missing values
- Cons
  - Greatly increases the dimensionality of the data
    - Makes learning less efficient (if the complexity of the algorithm depends on k) and more difficult (many new features will be of limited predictive value)
    - Increases the tendency of models to overfit
    - Increases communication costs in distributed settings
  - Can (potentially) create multicollinearity between features

Dimensionality Reduction

- In order to perform dimensionality reduction of one-hot encodings, we first need to compute these (potentially massive) encodings

Dimensionality Reduction via Feature Hashing

- In order to perform dimensionality reduction of one-hot encodings, we first need to compute these (potentially massive) encodings
- Feature hashing is an alternative mechanism for converting non-numeric features to numeric ones that
  - Avoids having to compute expensive intermediate entities
  - Also gives rise to sparse representations
  - Has some nice theoretical guarantees
  - Good empirical performance on a variety of tasks

Hashing

- A *hash function* maps an input to one of m "buckets"
  - A good hash function should be easy to compute and (roughly) evenly distribute inputs across all m buckets
  - Unrealistically simple examples:

$$h(\text{int } x) = \big(x(x+3)\big) \bmod m$$

$$h(\text{string } s) = \left(\sum_{c=1}^{len(s)} s[c] * 2^c\right) \bmod m$$

- In the context of feature hashing
  - Inputs to our hash functions will be feature values
  - $m \ll$ the total number of distinct feature values (i.e., the dimensionality of the one-hot encoding)
  - Bucket indices will be the new features

Hashing: Example

- Consider this medical dataset

Family History	Blood Pressure	Cholesterol
Yes	Low	Normal
	Medium	Normal
No	Low	Abnormal
Yes		Normal
Yes	High	Abnormal

0	1	2	3

- Define the hash function h explicitly ($m = 4$)
  - $h(\text{Family History} = \text{“Yes”}) = 0$
  - $h(\text{Family History} = \text{“No”}) = 2$
  - $h(\text{Blood Pressure} = \text{“Low”}) = 1$
  - $h(\text{Blood Pressure} = \text{“Medium”}) = 1$
  - $h(\text{Blood Pressure} = \text{“High”}) = 2$
  - $h(\text{Cholesterol} = \text{“Normal”}) = 3$
  - $h(\text{Cholesterol} = \text{“Abnormal”}) = 0$

Hashing: Example

- Consider this medical dataset

Family History	Blood Pressure	Cholesterol
Yes	Low	Normal
	Medium	Normal
No	Low	Abnormal
Yes		Normal
Yes	High	Abnormal

$\longrightarrow$

0	1	2	3
1	1	0	1

- Define the hash function h explicitly ($m = 4$)
  - $h(\text{Family History} = \text{"Yes"}) = 0$
  - $h(\text{Family History} = \text{"No"}) = 2$
  - $h(\text{Blood Pressure} = \text{"Low"}) = 1$
  - $h(\text{Blood Pressure} = \text{"Medium"}) = 1$
  - $h(\text{Blood Pressure} = \text{"High"}) = 2$
  - $h(\text{Cholesterol} = \text{"Normal"}) = 3$
  - $h(\text{Cholesterol} = \text{"Abnormal"}) = 0$

Hashing: Example

- Consider this medical dataset

Family History	Blood Pressure	Cholesterol
Yes	Low	Normal
	Medium	Normal
No	Low	Abnormal
Yes		Normal
Yes	High	Abnormal

→

0	1	2	3
1	1	0	1
0	1	0	1
1	1	1	0
1	0	0	1
2	0	1	0

hash collision

- Define the hash function h explicitly ($m = 4$)
  - $h(\text{Family History} = \text{“Yes”}) = 0$
  - $h(\text{Family History} = \text{“No”}) = 2$
  - $h(\text{Blood Pressure} = \text{“Low”}) = 1$
  - $h(\text{Blood Pressure} = \text{“Medium”}) = 1$
  - $h(\text{Blood Pressure} = \text{“High”}) = 2$
  - $h(\text{Cholesterol} = \text{“Normal”}) = 3$
  - $h(\text{Cholesterol} = \text{“Abnormal”}) = 0$

Distributed Feature Hashing

```
trainHashed = trainData.map(apply_hash_function)
```

- Hash function can be computed locally
- Hash functions are typically very fast
- Hashed features can be stored in a sparse representation

Feature Hashing preserves Relative Distances

- Suppose we have one-hot encoded features $\boldsymbol{x}$
- Consider two hash functions:
  - $h: \mathbb{N} \rightarrow \{1, \ldots, m\}$
  - A sign hash function $\xi: \mathbb{N} \rightarrow \{-1, +1\}$ with equal probability

$$P(\xi(i) = -1) = P(\xi(i) = +1) = {}^{1}/_{2}$$

- Define a hash kernel as $K(\boldsymbol{x}, \boldsymbol{y}) = \phi(\boldsymbol{x})^T \phi(\boldsymbol{y})$ where

$$\phi(\boldsymbol{x})^T = \left[\sum_{i:h(i)=1} \xi(i) x_i \quad \sum_{i:h(i)=2} \xi(i) x_i \quad \cdots \quad \sum_{i:h(i)=m} \xi(i) x_i \right]$$

- This implies two (related) things:
  1. $\mathbb{E}_{\xi}[K(\boldsymbol{x}, \boldsymbol{y})] = \boldsymbol{x}^T \boldsymbol{y}$ i.e., the hash kernel is *unbiased*
  2. Given a dataset $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}\right)\right\}_{i=1}^{n}$ and $\epsilon > 0$, if $m \geq \Omega\left(\frac{1}{\epsilon^2} \log \frac{N}{\delta}\right)$ then

$$\left\|\phi\left(\boldsymbol{x}^{(i)}\right) - \phi\left(\boldsymbol{x}^{(j)}\right)\right\|_2^2 \in \left[(1-\epsilon)\left\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\right\|_2^2, (1+\epsilon)\left\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\right\|_2^2\right]$$

$\forall\, \boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)} \in \mathcal{D}$ with probability at least $1 - \delta$

Source: Weinberger et al., 2009 (https://arxiv.org/pdf/0902.2206.pdf)

Two Specific Applications of Feature Hashing

- *Count-min sketch* for logistic regression
- *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

Okay, but what does this have to do with Logistic Regression?

- 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

Recall: Gradient Descent for Logistic Regression

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}, \alpha$

1. Initialize $\boldsymbol{w}^{(0)}$ to all zeros and set $t = 0$
2. While TERMINATION CRITERION is not satisfied
   a. Compute the gradient:
   $$\nabla_{\boldsymbol{w}} L_{\mathcal{D}}\left(\boldsymbol{w}^{(t)}\right) = \sum_{i=1}^{n}\left(\sigma\left({\boldsymbol{w}^{(t)}}^T \boldsymbol{x}^{(i)}\right) - y^{(i)}\right)\boldsymbol{x}^{(i)}$$
   b. Update $\boldsymbol{w}$: $\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \alpha \nabla_{\boldsymbol{w}} L_{\mathcal{D}}\left(\boldsymbol{w}^{(t)}\right)$
   c. Increment t: $t \leftarrow t + 1$

- Output: $\boldsymbol{w}^{(t)}$

Stochastic Gradient Descent for Logistic Regression

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}, \alpha$

1. Initialize $\boldsymbol{w}^{(0)}$ to all zeros and set $t = 0$
2. While TERMINATION CRITERION is not satisfied
   a. Randomly sample a point from the dataset $\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)$
   b. Compute the *pointwise* gradient:
   $$\nabla_{\boldsymbol{w}} L_{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)}\left(\boldsymbol{w}^{(t)}\right) = \left(\sigma\left({\boldsymbol{w}^{(t)}}^T \boldsymbol{x}^{(i)}\right) - y^{(i)}\right)\boldsymbol{x}^{(i)}$$
   c. Update $\boldsymbol{w}$: $\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \alpha \nabla_{\boldsymbol{w}} L_{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)}$
   d. Increment t: $t \leftarrow t + 1$

- Output: $\boldsymbol{w}^{(t)}$

Count-min Sketch for Logistic Regression

- Suppose observations are one-hot encoded
- Issue: weight vector might be prohibitively big (large k)
  - Secondary issue: in online/streaming settings, the set of all unique feature values may be unknown so the weight vector may need to grow dynamically
- Insight: if we use one-hot encoded features, we can write the stochastic gradient descent update for logistic regression as a weighted *count* of each feature

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \alpha\left(\sigma\left({\boldsymbol{w}^{(t)}}^T \boldsymbol{x}^{(i)}\right) - y^{(i)}\right)\boldsymbol{x}^{(i)}$$

- Idea: use count-min sketch to keep track of this weighted *count*

Count-min Sketch for Logistic Regression

- Suppose observations are one-hot encoded
- Issue: weight vector might be prohibitively big (large k)
  - Secondary issue: in online/streaming settings, the set of all unique feature values may be unknown so the weight vector may need to grow dynamically
- Insight: if we use one-hot encoded features, we can write the stochastic gradient descent update for logistic regression as a ~~weighted~~ *count* of each feature

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \alpha\left(\sigma\left({\boldsymbol{w}^{(t)}}^T \boldsymbol{x}^{(i)}\right) - y^{(i)}\right)\boldsymbol{x}^{(i)}$$

- Idea: use count-min sketch to keep track of this ~~weighted~~ *count*

Count-min Sketch: Example

- Suppose we want to count search terms in a stream of Google searches

Youtube Amazon Youtube Google Facebook Weather

- Using a hash function h ($m = 5$):

$$h(\text{“Youtube”}) = 3$$

1	2	3	4	5

Count-min Sketch: Example

- Suppose we want to count search terms in a stream of Google searches

Youtube Amazon Youtube Google Facebook Weather

- Using a hash function h $(m = 5)$:

$$h(\text{“Youtube”}) = 3$$

1	2	3	4	5
		$0 + 1$		

Count-min Sketch: Example

- Suppose we want to count search terms in a stream of Google searches

Youtube Amazon Youtube Google Facebook Weather

- Using a hash function h $(m = 5)$:

$$h(\text{“Amazon”}) = 2$$

1	2	3	4	5
	$0+1$	1		

Count-min Sketch: Example

- Suppose we want to count search terms in a stream of Google searches

Youtube Amazon Youtube Google Facebook Weather

- Using a hash function h $(m = 5)$:

$$h(\text{“Youtube”}) = 3$$

1	2	3	4	5
	1	$1 + 1$		

Count-min Sketch: Example

- Suppose we want to count search terms in a stream of Google searches

Youtube Amazon Youtube Google Facebook Weather

- Using a hash function h ($m = 5$):

$$h(\text{“Google”}) = 5$$

1	2	3	4	5
	1	2		$0+1$

Count-min Sketch: Example

- Suppose we want to count search terms in a stream of Google searches

Youtube Amazon Youtube Google Facebook Weather

- Using a hash function h $(m = 5)$:

$$h(\text{“Facebook”}) = 2$$

1	2	3	4	5
	$1+1$	2		1

Collision!

Count-min Sketch: Example

- Suppose we want to count search terms in a stream of Google searches

Youtube Amazon Youtube Google Facebook Weather

- Using a hash function h ($m = 5$):

$$h(\text{“Weather”}) = 5$$

1	2	3	4	5
	2	2		$1+1$

Collision!

Count-min Sketch: Example

- Suppose we want to count search terms in a stream of Google searches

Youtube Amazon Youtube Google Facebook Weather

- Using a hash function h $(m = 5)$:

$$h(\text{“Weather”}) = 5$$

1	2	3	4	5
	2	2		2

- Collisions are common, especially when m is small
- Idea: use more than one hash function!

Count-min Sketch: Example

- Suppose we want to count search terms in a stream of Google searches

Youtube Amazon Youtube Google Facebook Weather

- $r = 3$ independent hash functions $h_1, \ldots, h_r$ ($m = 5$):

$$h_1(\text{"Youtube"}) = 3, h_2(\text{"Youtube"}) = 1, h_3(\text{"Youtube"}) = 4$$

1	2	3	4	5
		$0+1$		
$0+1$				
			$0+1$	

Count-min Sketch: Example

- Suppose we want to count search terms in a stream of Google searches

Youtube Amazon Youtube Google Facebook Weather

- $r = 3$ independent hash functions $h_1, \dots, h_r$ ($m = 5$):

$h_1(\text{“Amazon”}) = 2, h_2(\text{“Amazon”}) = 2, h_3(\text{“Amazon”}) = 5$

1	2	3	4	5
	$0+1$	1		
1	$0+1$			
			1	$0+1$

Count-min Sketch: Example

- Suppose we want to count search terms in a stream of Google searches

Youtube Amazon Youtube Google Facebook Weather

- $r = 3$ independent hash functions $h_1, \dots, h_r$ $(m = 5)$:

$h_1(\text{“Youtube”}) = 3, h_2(\text{“Youtube”}) = 1, h_3(\text{“Youtube”}) = 4$

1	2	3	4	5
	1	$1+1$		
$1+1$	1			
			$1+1$	1

Count-min Sketch: Example

- Suppose we want to count search terms in a stream of Google searches

Youtube Amazon Youtube Google Facebook Weather

- $r = 3$ independent hash functions $h_1, \ldots, h_r$ ($m = 5$):

$h_1(\text{"Google"}) = 5, h_2(\text{"Google"}) = 3, h_3(\text{"Google"}) = 2$

1	2	3	4	5
	1	2		$0 + 1$
2	1	$0 + 1$		
	$0 + 1$		2	1

Count-min Sketch: Example

- Suppose we want to count search terms in a stream of Google searches

Youtube Amazon Youtube Google Facebook Weather

- $r = 3$ independent hash functions $h_1, \ldots, h_r$ ($m = 5$):

$$h_1(\text{“Facebook”}) = 2, h_2(\text{“Facebook”}) = 1, h_3(\text{“Facebook”}) = 1$$

1	2	3	4	5
	$1+1$	2		1
$2+1$	1	1		
$0+1$	1		2	1

Count-min Sketch: Example

- Suppose we want to count search terms in a stream of Google searches

Youtube Amazon Youtube Google Facebook Weather

- $r = 3$ independent hash functions $h_1, \ldots, h_r$ ($m = 5$):

$h_1(\text{"Weather"}) = 5, h_2(\text{"Weather"}) = 4, h_3(\text{"Weather"}) = 2$

1	2	3	4	5
	2	2		$1+1$
3	1	1	$0+1$	
1	$1+1$		2	1

Count-min Sketch: Example

- Suppose we want to count search terms in a stream of Google searches

Youtube Amazon Youtube Google Facebook Weather

- $r = 3$ independent hash functions $h_1, \dots, h_r$ ($m = 5$):

1	2	3	4	5
	2	2		2
3	1	1	1	
1	2		2	1

- Take the minimum across hashes to *approximate* counts

Count-min Sketch: Example

- Suppose we want to count search terms in a stream of Google searches

Youtube Amazon Youtube Google Facebook Weather

- $r = 3$ independent hash functions $h_1, \ldots, h_r$ ($m = 5$):

$$h_1(\text{“Youtube”}) = 3, h_2(\text{“Youtube”}) = 1, h_3(\text{“Youtube”}) = 4$$

1	2	3	4	5
	2	**2**		2
3	1	1	1	
1	2		**2**	1

- $\hat{c}_{\text{Youtube}} = \min_i (C[i, h_i(\text{“Youtube”})]) = 2$

Count-min Sketch: Example

- Suppose we want to count search terms in a stream of Google searches

Youtube   Amazon   Youtube   Google   Facebook   Weather

- $r = 3$ independent hash functions $h_1, \ldots, h_r$ $(m = 5)$:

$h_1(\text{“Amazon”}) = 2, h_2(\text{“Amazon”}) = 2, h_3(\text{“Amazon”}) = 5$

1	2	3	4	5
	2	2		2
3	**1**	1	1	
1	2		2	**1**

- $\hat{c}_{\text{Amazon}} = \min_i (C[i, h_i(\text{“Amazon”})]) = 1$

Count-min Sketch: Example

- Suppose we want to count search terms in a stream of Google searches

Youtube Amazon Youtube Google Facebook Weather

- $r = 3$ independent hash functions $h_1, \dots, h_r$ ($m = 5$):

$$h_1(\text{“Google”}) = 5, h_2(\text{“Google”}) = 3, h_3(\text{“Google”}) = 2$$

1	2	3	4	5
	2	2		**2**
3	1	**1**	1	
1	**2**		2	1

- $\hat{c}_{\text{Google}} = \min_i (C[i, h_i(\text{“Google”})]) = 1$

Count-min Sketch: Example

- Suppose we want to count search terms in a stream of Google searches

Youtube Amazon Youtube Google Facebook Weather

- $r = 3$ independent hash functions $h_1, \ldots, h_r$ $(m = 5)$:

$$h_1(\text{"Facebook"}) = 2, h_2(\text{"Facebook"}) = 1, h_3(\text{"Facebook"}) = 1$$

1	2	3	4	5
	2	2		2
3	1	1	1	
1	2		2	1

- $\hat{c}_{\text{Facebook}} = \min_i (C[i, h_i(\text{"Facebook"})]) = 1$

Count-min Sketch: Example

- Suppose we want to count search terms in a stream of Google searches

Youtube Amazon Youtube Google Facebook Weather

- $r = 3$ independent hash functions $h_1, \dots, h_r$ $(m = 5)$:

$$h_1(\text{“Weather”}) = 5, h_2(\text{“Weather”}) = 4, h_3(\text{“Weather”}) = 2$$

1	2	3	4	5
	2	2		**2**
3	1	1	**1**	
1	**2**		2	1

- $\hat{c}_{\text{Weather}} = \min_i (C[i, h_i(\text{“Weather”})]) = 1$

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

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

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. $C[i, h_i(s)] \mathrel{+}= 1$
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