COMPSCI 514: ALGORITHMS FOR DATA SCIENCE

Andrew McGregor

Lecture 24

SUMMARY

This Class:

- Course wrap up.

PART III: OPTIMIZATION

- Foundational concepts like convexity (line between any two points on curve is above the curve and definition via derivatives), convex sets (line between any two points is in the set), directional derivative (slope of curve if we move in particular direction), and Lipschitzness (slope is bounded).

CONTINUOUS OPTIMIZATION

- Foundational concepts like convexity (line between any two points on curve is above the curve and definition via derivatives), convex sets (line between any two points is in the set), directional derivative (slope of curve if we move in particular direction), and Lipschitzness (slope is bounded).
- Gradient descent greedily tries to find the min value of function $f : \mathbb{R}^d \to \mathbb{R}$ by maintaining a vector $\vec{\theta} \in \mathbb{R}^d$ and at each step moving $\vec{\theta}$ "downhill", i.e., in the direction that minimizes directional derivative

CONTINUOUS OPTIMIZATION

- Foundational concepts like convexity (line between any two points on curve is above the curve and definition via derivatives), convex sets (line between any two points is in the set), directional derivative (slope of curve if we move in particular direction), and Lipschitzness (slope is bounded).
- Gradient descent greedily tries to find the min value of function $f : \mathbb{R}^d \to \mathbb{R}$ by maintaining a vector $\vec{\theta} \in \mathbb{R}^d$ and at each step moving $\vec{\theta}$ "downhill", i.e., in the direction that minimizes directional derivative
- Bounded the number of steps required if f is convex and Lipschitz.

CONTINUOUS OPTIMIZATION

- Foundational concepts like convexity (line between any two points on curve is above the curve and definition via derivatives), convex sets (line between any two points is in the set), directional derivative (slope of curve if we move in particular direction), and Lipschitzness (slope is bounded).
- Gradient descent greedily tries to find the min value of function $f : \mathbb{R}^d \to \mathbb{R}$ by maintaining a vector $\vec{\theta} \in \mathbb{R}^d$ and at each step moving $\vec{\theta}$ "downhill", i.e., in the direction that minimizes directional derivative
- Bounded the number of steps required if f is convex and Lipschitz.
- Simple extension for optimization over a convex constraint set.

CONTINUOUS OPTIMIZATION

- Foundational concepts like convexity (line between any two points on curve is above the curve and definition via derivatives), convex sets (line between any two points is in the set), directional derivative (slope of curve if we move in particular direction), and Lipschitzness (slope is bounded).
- Gradient descent greedily tries to find the min value of function $f : \mathbb{R}^d \to \mathbb{R}$ by maintaining a vector $\vec{\theta} \in \mathbb{R}^d$ and at each step moving $\vec{\theta}$ "downhill", i.e., in the direction that minimizes directional derivative
- Bounded the number of steps required if f is convex and Lipschitz.
- Simple extension for optimization over a convex constraint set.
- Lots that we didn't cover: accelerated methods, adaptive methods, second order methods (quasi-Newton methods). Gave mathematical tools to understand these methods. See CS 690OP for more!

EXAMPLE OF GRADIENTS

- Suppose $f : \mathbb{R}^3 \to \mathbb{R}$ where $f(\vec{\theta}) = \theta_1^3 + \theta_2\theta_3 + \theta_3^2$ then

EXAMPLE OF GRADIENTS

- Suppose $f : \mathbb{R}^3 \to \mathbb{R}$ where $f(\vec{\theta}) = \theta_1^3 + \theta_2\theta_3 + \theta_3^2$ then

$$\nabla f(\vec{\theta}) = \begin{pmatrix} 3\theta_1^2 \\ \theta_3 \\ \theta_2 + 2\theta_3 \end{pmatrix}$$

and

$$\|\nabla f(\vec{\theta})\|_2 = \sqrt{(3\theta_1^2)^2 + (\theta_3)^2 + (\theta_2 + 2\theta_3)^2}$$

EXAMPLE OF GRADIENTS

- Suppose $f : \mathbb{R}^3 \to \mathbb{R}$ where $f(\vec{\theta}) = \theta_1^3 + \theta_2\theta_3 + \theta_3^2$ then

$$\nabla f(\vec{\theta}) = \begin{pmatrix} 3\theta_1^2 \\ \theta_3 \\ \theta_2 + 2\theta_3 \end{pmatrix}$$

  and

$$\|\nabla f(\vec{\theta})\|_2 = \sqrt{(3\theta_1^2)^2 + (\theta_3)^2 + (\theta_2 + 2\theta_3)^2}$$

- Suppose $f : \mathbb{R}^3 \to \mathbb{R}$ where $f(\vec{\theta}) = 3\theta_1 + \theta_2 + 5\theta_3$ then

EXAMPLE OF GRADIENTS

- Suppose $f : \mathbb{R}^3 \to \mathbb{R}$ where $f(\vec{\theta}) = \theta_1^3 + \theta_2\theta_3 + \theta_3^2$ then

$$\nabla f(\vec{\theta}) = \begin{pmatrix} 3\theta_1^2 \\ \theta_3 \\ \theta_2 + 2\theta_3 \end{pmatrix}$$

  and

$$\|\nabla f(\vec{\theta})\|_2 = \sqrt{(3\theta_1^2)^2 + (\theta_3)^2 + (\theta_2 + 2\theta_3)^2}$$

- Suppose $f : \mathbb{R}^3 \to \mathbb{R}$ where $f(\vec{\theta}) = 3\theta_1 + \theta_2 + 5\theta_3$ then

$$\nabla f(\vec{\theta}) = \begin{pmatrix} 3 \\ 1 \\ 5 \end{pmatrix}$$

  and $\|\nabla f(\vec{\theta})\|_2 = \sqrt{3^2 + 1^2 + 5^2}$.

GRADIENT DESCENT

Goal: Find $\vec{\theta} \in \mathbb{R}^d$ that (nearly) minimizes convex function $f : \mathbb{R}^d \to \mathbb{R}$.

GRADIENT DESCENT

Goal: Find $\vec{\theta} \in \mathbb{R}^d$ that (nearly) minimizes convex function $f : \mathbb{R}^d \to \mathbb{R}$.

Algorithm/Analysis: We analyzed the update step:

$$\vec{\theta}^{(i+1)} = \vec{\theta}^{(i)} - \eta \cdot \vec{\nabla} f(\vec{\theta}^{(i)})$$

and showed that after a certain number of steps depending on ϵ, the max gradient of f, and how far the initial point is from the optimal point,

$$\hat{\theta} = \underset{\vec{\theta}_1, \ldots \vec{\theta}_t}{\arg\min} f(\vec{\theta}_i)$$

ensures $f(\hat{\theta}) \leq \left(\min_{\vec{\theta}} f(\vec{\theta})\right) + \epsilon$.

GRADIENT DESCENT

Goal: Find $\vec{\theta} \in \mathbb{R}^d$ that (nearly) minimizes convex function $f : \mathbb{R}^d \to \mathbb{R}$.

Algorithm/Analysis: We analyzed the update step:

$$\vec{\theta}^{(i+1)} = \vec{\theta}^{(i)} - \eta \cdot \vec{\nabla} f(\vec{\theta}^{(i)})$$

and showed that after a certain number of steps depending on ϵ, the max gradient of f, and how far the initial point is from the optimal point,

$$\hat{\theta} = \arg\min_{\vec{\theta}_1, \ldots \vec{\theta}_t} f(\vec{\theta}_i)$$

ensures $f(\hat{\theta}) \leq \left(\min_{\vec{\theta}} f(\vec{\theta})\right) + \epsilon$.

Projected Gradient Descent: If we want to find $\vec{\theta} \in S$ that (nearly) minimizes convex function f for some convex set S, we just modify the update rule to $\vec{\theta}^{(i+1)} = P_S(\vec{\theta}^{(i)} - \eta \cdot \vec{\nabla} f(\vec{\theta}^{(i)}))$ where P_S is the projection function that maps the input to the closest point in S.

STOCHASTIC GRADIENT DESCENT

Goal: Back to finding $\vec{\theta} \in \mathbb{R}^d$ that (nearly) minimizes convex function $f : \mathbb{R}^d \to \mathbb{R}$.

STOCHASTIC GRADIENT DESCENT

Goal: Back to finding $\vec{\theta} \in \mathbb{R}^d$ that (nearly) minimizes convex function $f : \mathbb{R}^d \to \mathbb{R}$.

Idea for Stochastic Gradient Descent: Rather than computing $\vec{\nabla} f(\vec{\theta}^{(i)})$ in the update step:

$$\vec{\theta}^{(i+1)} = \vec{\theta}^{(i)} - \eta \cdot \vec{\nabla} f(\vec{\theta}^{(i)})$$

instead we do something randomized:

$$\vec{\theta}^{(i+1)} = \vec{\theta}^{(i)} - \eta \cdot D(\vec{\theta}^{(i)})$$

where $D(\vec{\theta}^{(i)})$ is faster to compute and approximates $\vec{\nabla} f(\vec{\theta}^{(i)})$ in expectation.

STOCHASTIC GRADIENT DESCENT

Goal: Back to finding $\vec{\theta} \in \mathbb{R}^d$ that (nearly) minimizes convex function $f : \mathbb{R}^d \to \mathbb{R}$.

Idea for Stochastic Gradient Descent: Rather than computing $\vec{\nabla} f(\vec{\theta}^{(i)})$ in the update step:

$$\vec{\theta}^{(i+1)} = \vec{\theta}^{(i)} - \eta \cdot \vec{\nabla} f(\vec{\theta}^{(i)})$$

instead we do something randomized:

$$\vec{\theta}^{(i+1)} = \vec{\theta}^{(i)} - \eta \cdot D(\vec{\theta}^{(i)})$$

where $D(\vec{\theta}^{(i)})$ is faster to compute and approximates $\vec{\nabla} f(\vec{\theta}^{(i)})$ in expectation. This may increase the number of iterations but each iteration may be much cheaper depending on f and how we generate D.

STOCHASTIC GRADIENT DESCENT

Assume that:

- f is convex and decomposable as $f(\vec{\theta}) = \sum_{j=1}^{n} f_j(\vec{\theta})$.
- Each f_j is $\frac{G}{n}$-Lipschitz.
- Initialize with $\theta^{(1)}$ satisfying $\|\vec{\theta}^{(1)} - \vec{\theta}^*\|_2 \leq R$.

Stochastic Gradient Descent

- Pick some initial $\vec{\theta}^{(1)}$.
- Set step size $\eta = \frac{R}{G\sqrt{t}}$.
- For $i = 1, \ldots, t$
  - $\vec{\theta}^{(i+1)} = \vec{\theta}^{(i)} - \eta \cdot \vec{\nabla} f_j(\vec{\theta}^{(i)})$ where j is chosen randomly from $1, \ldots, n$
- Return $\hat{\theta} = \frac{1}{t} \sum_{i=1}^{t} \vec{\theta}^{(i)}$.

STOCHASTIC GRADIENT DESCENT

Assume that:

- f is convex and decomposable as $f(\vec{\theta}) = \sum_{j=1}^{n} f_j(\vec{\theta})$.
- Each f_j is $\frac{G}{n}$-Lipschitz.
- Initialize with $\theta^{(1)}$ satisfying $\|\vec{\theta}^{(1)} - \vec{\theta}^*\|_2 \leq R$.

Stochastic Gradient Descent

- Pick some initial $\vec{\theta}^{(1)}$.
- Set step size $\eta = \frac{R}{G\sqrt{t}}$.
- For $i = 1, \ldots, t$
  - $\vec{\theta}^{(i+1)} = \vec{\theta}^{(i)} - \eta \cdot \vec{\nabla} f_j(\vec{\theta}^{(i)})$ where j is chosen randomly from $1, \ldots, n$
- Return $\hat{\theta} = \frac{1}{t} \sum_{i=1}^{t} \vec{\theta}^{(i)}$.

We showed that $t = R^2 G^2 / \epsilon^2$ iterations sufficed. We also showed that number of iterations for gradient descent but note assuming each f_j is $\frac{G}{n}$-Lipschitz is a stronger assumption that f is G-Lipschitz.

ONLINE OPTIMIZATION

Online Optimization: In place of a single function f, we see a different objective function at each step: $f_1, f_2, \ldots, f_t : \mathbb{R}^d \to \mathbb{R}$ where we make no assumptions on how the functions are related to each other.

ONLINE OPTIMIZATION

Online Optimization: In place of a single function f, we see a different objective function at each step: $f_1, f_2, \ldots, f_t : \mathbb{R}^d \to \mathbb{R}$ where we make no assumptions on how the functions are related to each other.

- At each step, first pick (play) a parameter vector $\vec{\theta}^{(i)}$.

ONLINE OPTIMIZATION

Online Optimization: In place of a single function f, we see a different objective function at each step: $f_1, f_2, \ldots, f_t : \mathbb{R}^d \to \mathbb{R}$ where we make no assumptions on how the functions are related to each other.

- At each step, first pick (play) a parameter vector $\vec{\theta}^{(i)}$.
- Then are told f_i and incur cost $f_i(\vec{\theta}^{(i)})$.

ONLINE OPTIMIZATION

Online Optimization: In place of a single function f, we see a different objective function at each step: $f_1, f_2, \ldots, f_t : \mathbb{R}^d \to \mathbb{R}$ where we make no assumptions on how the functions are related to each other.

- At each step, first pick (play) a parameter vector $\vec{\theta}^{(i)}$.
- Then are told f_i and incur cost $f_i(\vec{\theta}^{(i)})$.
- Minimize "Regret" $= \sum_{i=1}^{t} f_i(\vec{\theta}^{(i)}) - \sum_{i=1}^{t} f_i(\vec{\theta}^{off})$ where

$$\vec{\theta}^{off} = \arg\min_{\vec{\theta}} \sum_{i=1}^{t} f_i(\vec{\theta})$$

ONLINE OPTIMIZATION

Online Optimization: In place of a single function f, we see a different objective function at each step: $f_1, f_2, \ldots, f_t : \mathbb{R}^d \to \mathbb{R}$ where we make no assumptions on how the functions are related to each other.

- At each step, first pick (play) a parameter vector $\vec{\theta}^{(i)}$.
- Then are told f_i and incur cost $f_i(\vec{\theta}^{(i)})$.
- Minimize "Regret" $= \sum_{i=1}^{t} f_i(\vec{\theta}^{(i)}) - \sum_{i=1}^{t} f_i(\vec{\theta}^{off})$ where

$$\vec{\theta}^{off} = \arg\min_{\vec{\theta}} \sum_{i=1}^{t} f_i(\vec{\theta})$$

- **Algorithm/Analysis:** We analyzed the update step:

$$\vec{\theta}^{(i+1)} = \vec{\theta}^{(i)} - \eta \cdot \vec{\nabla} f_i(\vec{\theta}^{(i)})$$

and showed that $Regret/t \to 0$ as $t \to \infty$

PART II: LINEAR ALGEBRA

DIMENSIONALITY REDUCTION

Methods for working with (compressing) high-dimensional data

Methods for working with (compressing) high-dimensional data

- Started with randomized dimensionality reduction and the JL lemma: compression from *any* d-dimensions to $O(\log n/\epsilon^2)$ dimensions while preserving pairwise distances.

DIMENSIONALITY REDUCTION

Methods for working with (compressing) high-dimensional data

- Started with randomized dimensionality reduction and the JL lemma: compression from *any* d-dimensions to $O(\log n/\epsilon^2)$ dimensions while preserving pairwise distances.
- Dimensionality reduction via low-rank approximation and optimal solution with PCA/eigendecomposition/SVD.

DIMENSIONALITY REDUCTION

Methods for working with (compressing) high-dimensional data

- Started with randomized dimensionality reduction and the JL lemma: compression from *any* d-dimensions to $O(\log n/\epsilon^2)$ dimensions while preserving pairwise distances.
- Dimensionality reduction via low-rank approximation and optimal solution with PCA/eigendecomposition/SVD.
- Spectral graph theory – nonlinear dimension reduction and spectral clustering for community detection.

DIMENSIONALITY REDUCTION

Methods for working with (compressing) high-dimensional data

- Started with randomized dimensionality reduction and the JL lemma: compression from *any* d-dimensions to $O(\log n/\epsilon^2)$ dimensions while preserving pairwise distances.
- Dimensionality reduction via low-rank approximation and optimal solution with PCA/eigendecomposition/SVD.
- Spectral graph theory – nonlinear dimension reduction and spectral clustering for community detection.
- In the process covered linear algebraic tools that are very broadly useful in ML and data science: eigendecomposition, singular value decomposition.

REDUCING DIMENSION: JOHNSON LINDENSTRAUSS

- Let $\vec{\pi} \in \mathbb{R}^d$ have random $\mathcal{N}(0,1)$ entries. Then for any $\vec{x} \in \mathbb{R}^d$,

$$\mathbb{E}[\langle \vec{\pi}, \vec{x} \rangle^2] = \|\vec{x}\|_2^2$$

  Proof just uses linearity of expectation and variance.

REDUCING DIMENSION: JOHNSON LINDENSTRAUSS

- Let $\vec{\pi} \in \mathbb{R}^d$ have random $\mathcal{N}(0,1)$ entries. Then for any $\vec{x} \in \mathbb{R}^d$,

$$\mathbb{E}[\langle \vec{\pi}, \vec{x} \rangle^2] = \|\vec{x}\|_2^2$$

  Proof just uses linearity of expectation and variance.

- Let $\mathbf{\Pi} \in \mathbb{R}^{k \times d}$ where $k = O\left(\epsilon^{-2} \log n\right)$ with $\mathcal{N}(0, 1/k)$ entries, then for any $\vec{x} \in \mathbb{R}^d$,

$$(1-\epsilon)\|\vec{x}\|_2^2 \leq \|\mathbf{\Pi}\vec{x}\|_2^2 \leq (1+\epsilon)\|\vec{x}\|_2^2$$

REDUCING DIMENSION: JOHNSON LINDENSTRAUSS

- Let $\vec{\pi} \in \mathbb{R}^d$ have random $\mathcal{N}(0,1)$ entries. Then for any $\vec{x} \in \mathbb{R}^d$,

$$\mathbb{E}[\langle \vec{\pi}, \vec{x} \rangle^2] = \|\vec{x}\|_2^2$$

  Proof just uses linearity of expectation and variance.

- Let $\mathbf{\Pi} \in \mathbb{R}^{k \times d}$ where $k = O\left(\epsilon^{-2} \log n\right)$ with $\mathcal{N}(0, 1/k)$ entries, then for any $\vec{x} \in \mathbb{R}^d$,

$$(1-\epsilon)\|\vec{x}\|_2^2 \leq \|\mathbf{\Pi}\vec{x}\|_2^2 \leq (1+\epsilon)\|\vec{x}\|_2^2$$

- Furthermore, for any $\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n \in \mathbb{R}^d$,

$$(1-\epsilon)\|\vec{x}_i - \vec{x}_j\|_2^2 \leq \|\mathbf{\Pi}\vec{x}_i - \mathbf{\Pi}\vec{x}_j\|_2^2 \leq (1+\epsilon)\|\vec{x}_i - \vec{x}_j\|_2^2$$

  i.e., random projections preserve distances between vectors.

REDUCING DIMENSION: PCA

- The $\mathcal{V}$ be the k-dimension subspace of $\mathbb{R}^d$ and let $\mathbf{V} \in \mathbb{R}^{d \times k}$ be the matrix whose columns are an orthonormal basis for $\mathcal{V}$. Then,

$$\mathbf{V}\mathbf{V}^T \vec{x} = \arg\min_{\vec{z} \in \mathcal{V}} \|\vec{z} - \vec{x}\|_2$$

REDUCING DIMENSION: PCA

- The $\mathcal{V}$ be the k-dimension subspace of $\mathbb{R}^d$ and let $\mathbf{V} \in \mathbb{R}^{d \times k}$ be the matrix whose columns are an orthonormal basis for $\mathcal{V}$. Then,

$$\mathbf{V}\mathbf{V}^T\vec{x} = \arg\min_{\vec{z} \in \mathcal{V}} \|\vec{z} - \vec{x}\|_2$$

- If we have n points (rows of $\mathbf{X} \in \mathbb{R}^{n \times d}$), and want to project them all into a k-dimensional space $\mathcal{V}$, how to we chose $\mathcal{V}$ to minimizes the total error?

Best $\mathcal{V}$ is the one spanned by top k eigenvectors of $\mathbf{X}^T\mathbf{X}$

REDUCING DIMENSION: PCA

- The $\mathcal{V}$ be the k-dimension subspace of $\mathbb{R}^d$ and let $\mathbf{V} \in \mathbb{R}^{d \times k}$ be the matrix whose columns are an orthonormal basis for $\mathcal{V}$. Then,

$$\mathbf{V}\mathbf{V}^T\vec{x} = \arg\min_{\vec{z} \in \mathcal{V}} \|\vec{z} - \vec{x}\|_2$$

- If we have n points (rows of $\mathbf{X} \in \mathbb{R}^{n \times d}$), and want to project them all into a k-dimensional space $\mathcal{V}$, how to we chose $\mathcal{V}$ to minimizes the total error?

Best $\mathcal{V}$ is the one spanned by top k eigenvectors of $\mathbf{X}^T\mathbf{X}$

- I.e., if $\mathbf{V}_k$ is the matrix with the first k eigenvectors as columns,

$$\mathbf{V}_k = \arg\min_{\text{orthonormal } \mathbf{V}} \|\mathbf{X} - \mathbf{X}\mathbf{V}\mathbf{V}^T\|_F$$

and $\|\mathbf{X} - \mathbf{X}\mathbf{V_k}\mathbf{V_k}^T\|_F = \lambda_{k+1} + \lambda_{k+2} + \ldots$ where $\lambda_1 \geq \lambda_2 \geq \ldots$ are the eigenvalues of $\mathbf{X}^T\mathbf{X}$.

FINDING TOP EIGENVECTORS: POWER METHOD

- **Power Method:** The most fundamental iterative method for approximate SVD/eigendecomposition.

FINDING TOP EIGENVECTORS: POWER METHOD

- **Power Method:** The most fundamental iterative method for approximate SVD/eigendecomposition.
- **Goal:** Given a matrix $\mathbf{A} \in \mathbb{R}^{d \times d}$, find an approximation to the top eigenvector $\vec{v}_1$ of $\mathbf{A}$.

FINDING TOP EIGENVECTORS: POWER METHOD

- **Power Method:** The most fundamental iterative method for approximate SVD/eigendecomposition.
- **Goal:** Given a matrix $\mathbf{A} \in \mathbb{R}^{d \times d}$, find an approximation to the top eigenvector $\vec{v}_1$ of $\mathbf{A}$.
- **Algorithm:**
  - Choose $\vec{z}^{(0)}$ randomly: each $\vec{z}^{(0)}(i) \sim \mathcal{N}(0,1)$.
  - For $i = 1, \ldots, t$
    - $\vec{z}^{(i)} := \mathbf{A} \cdot \vec{z}^{(i-1)}$
    - $\vec{z}_i := \vec{z}^{(i)} / \|\vec{z}^{(i)}\|_2$

    Return $\vec{z}_t$
- With high probability, after $t = O\left(\gamma^{-1} \ln(d/\epsilon)\right)$ steps $\|\vec{z}^{(t)} - \vec{v}_1\|_2 \leq \epsilon$ where $\gamma = 1 - |\lambda_2|/|\lambda_1|$.

EIGENVALUES EXAMPLE

Consider matrix

$$A = \begin{pmatrix} 4 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 7 \end{pmatrix}$$

EIGENVALUES EXAMPLE

Consider matrix

$$A = \begin{pmatrix} 4 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 7 \end{pmatrix}$$

- A is a rank 3 ("full rank") matrix because it is impossible to write any row as a linear combination of the other rows. (Or equivalently

EIGENVALUES EXAMPLE

Consider matrix

$$A = \begin{pmatrix} 4 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 7 \end{pmatrix}$$

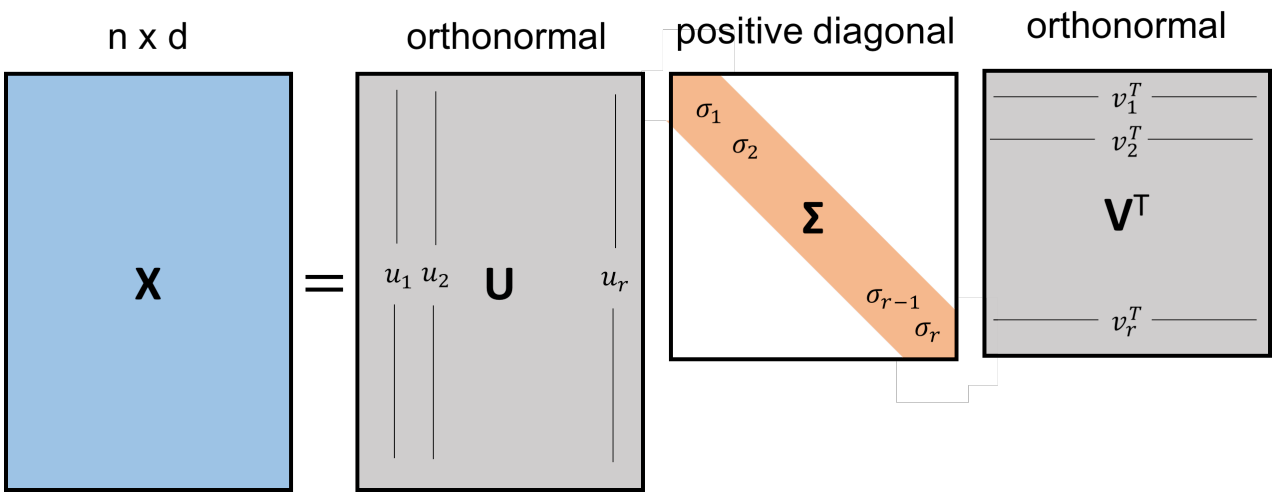
- A is a rank 3 ("full rank") matrix because it is impossible to write any row as a linear combination of the other rows. (Or equivalently
- λ is an eigenvalue if

$$A - \lambda I = \begin{pmatrix} 4-\lambda & 0 & 2 \\ 0 & 1-\lambda & 0 \\ 0 & 0 & 7-\lambda \end{pmatrix}$$

is not full rank. E.g., 4, 1, and 7 are eigenvalues in this case. In fact the eigenvalues of an upper triangular matrix are always the diagonal entries. This isn't true in general.

SINGULAR VALUE DECOMPOSITION

- Any symmetric matrix $\mathbf{A}$ can be written as $\mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$ corresponding to eigenvectors and eigenvectors.
- The Singular Value Decomposition (SVD) extends eigendecomposition.
- Any $\mathbf{X} \in \mathbb{R}^{n \times d}$ with $\text{rank}(\mathbf{X}) = r$ can be written as $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$.
  - $\mathbf{U}$ has orthonormal columns $\vec{u}_1, \ldots, \vec{u}_r \in \mathbb{R}^n$ (left singular vectors).
  - $\mathbf{V}$ has orthonormal columns $\vec{v}_1, \ldots, \vec{v}_r \in \mathbb{R}^d$ (right singular vectors).
  - $\mathbf{\Sigma}$ is diagonal with elements $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$ (singular values).



- Note $\mathbf{X}^T\mathbf{X} = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T$ and $\mathbf{X}\mathbf{X}^T = \mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^T$, i.e., the left/right singular vectors are the eigenvectors of $\mathbf{X}\mathbf{X}^T$ and $\mathbf{X}\mathbf{X}^T$ respectively.

- Let $\mathbf{U}_k$, $\mathbf{\Sigma}_k$, $\mathbf{V}_k$ be truncations of $\mathbf{U}$, $\mathbf{\Sigma}$, $\mathbf{V}$ to first k columns. The best rank k approximation of $\mathbf{X}$ is $\mathbf{X}\mathbf{V}_k\mathbf{V}_k^T = \mathbf{U_k}\mathbf{U_k}^T\mathbf{X} = \mathbf{U_k}\mathbf{\Sigma_k}\mathbf{V_k}^T$.

APPLICATIONS

- Let $\mathbf{U}_k$, $\mathbf{\Sigma}_k$, $\mathbf{V}_k$ be truncations of $\mathbf{U}$, $\mathbf{\Sigma}$, $\mathbf{V}$ to first k columns. The best rank k approximation of $\mathbf{X}$ is $\mathbf{X}\mathbf{V}_k\mathbf{V}_k^T = \mathbf{U_k}\mathbf{U_k}^T\mathbf{X} = \mathbf{U_k}\mathbf{\Sigma_k}\mathbf{V_k}^T$.
- Applications include: Approximating an "incomplete" matrix $\mathbf{X}$ by a low rank in the hope that the approximation "fills in" the missing values. LSA uses the rows of $\mathbf{U}$ to approximate the documents in the document/term matrix.

APPLICATIONS

- Let $\mathbf{U}_k$, $\mathbf{\Sigma}_k$, $\mathbf{V}_k$ be truncations of $\mathbf{U}$, $\mathbf{\Sigma}$, $\mathbf{V}$ to first k columns. The best rank k approximation of $\mathbf{X}$ is $\mathbf{X}\mathbf{V}_k\mathbf{V}_k^T = \mathbf{U_k}\mathbf{U_k}^T\mathbf{X} = \mathbf{U_k}\mathbf{\Sigma_k}\mathbf{V_k}^T$.
- Applications include: Approximating an "incomplete" matrix $\mathbf{X}$ by a low rank in the hope that the approximation "fills in" the missing values. LSA uses the rows of $\mathbf{U}$ to approximate the documents in the document/term matrix.
- Applications to graphs: Given adjacency matrix $\mathbf{A}$ projecting nodes on the top k eigenvalues of $\mathbf{A}^T\mathbf{A}$ allows us to map nodes to k-dimensional space such that close nodes are still close.

APPLICATIONS

- Let $\mathbf{U}_k$, $\mathbf{\Sigma}_k$, $\mathbf{V}_k$ be truncations of $\mathbf{U}$, $\mathbf{\Sigma}$, $\mathbf{V}$ to first k columns. The best rank k approximation of $\mathbf{X}$ is $\mathbf{X}\mathbf{V}_k\mathbf{V}_k^T = \mathbf{U_k}\mathbf{U_k}^T\mathbf{X} = \mathbf{U_k}\mathbf{\Sigma_k}\mathbf{V_k}^T$.
- Applications include: Approximating an "incomplete" matrix $\mathbf{X}$ by a low rank in the hope that the approximation "fills in" the missing values. LSA uses the rows of $\mathbf{U}$ to approximate the documents in the document/term matrix.
- Applications to graphs: Given adjacency matrix $\mathbf{A}$ projecting nodes on the top k eigenvalues of $\mathbf{A}^T\mathbf{A}$ allows us to map nodes to k-dimensional space such that close nodes are still close.
- Spectral Clustering Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$ satisfies $\vec{v}^T\mathbf{L}\vec{v} = \sum_{ij\in E}(v_i - v_j)^2$. The 2nd smallest eigenvector of $\mathbf{L}$ gives way to decompose the graph into roughly balanced groups such that the number of cross edges in minimized: put all nodes with negative entries in one group and all nodes with positive entires in the other.

STOCHASTIC BLOCK MODEL

STOCHASTIC BLOCK MODEL

- Stochastic Block Model is a generative model for generating graphs we could cluster: n nodes are partitioned into two groups A and B, edges between nodes in same group are present with probability p and edges between nodes in different groups are present with probability $q < p$.

STOCHASTIC BLOCK MODEL

- Stochastic Block Model is a generative model for generating graphs we could cluster: n nodes are partitioned into two groups A and B, edges between nodes in same group are present with probability p and edges between nodes in different groups are present with probability $q < p$.
- We showed the second smallest eigenvector of $\mathbb{E}[\mathbf{L}]$ allows us to find A and B exactly.

STOCHASTIC BLOCK MODEL

- Stochastic Block Model is a generative model for generating graphs we could cluster: n nodes are partitioned into two groups A and B, edges between nodes in same group are present with probability p and edges between nodes in different groups are present with probability $q < p$.
- We showed the second smallest eigenvector of $\mathbb{E}[\mathbf{L}]$ allows us to find A and B exactly. But the input to Spectral Clustering is $\mathbf{L}$, not $\mathbb{E}[\mathbf{L}]$!

STOCHASTIC BLOCK MODEL

- Stochastic Block Model is a generative model for generating graphs we could cluster: n nodes are partitioned into two groups A and B, edges between nodes in same group are present with probability p and edges between nodes in different groups are present with probability $q < p$.
- We showed the second smallest eigenvector of $\mathbb{E}[\mathbf{L}]$ allows us to find A and B exactly. But the input to Spectral Clustering is $\mathbf{L}$, not $\mathbb{E}[\mathbf{L}]$!
- Fortunately, we could show the 2nd smallest eigenvectors of $\mathbf{L}$ and $\mathbb{E}[\mathbf{L}]$ are sufficiently similar that we learn A and B we only a few mistakes.

PART I: RANDOMIZED TECHNIQUES

Randomization as a computational resource for massive datasets.

Randomization as a computational resource for massive datasets.

- Focus on problems that are easy on small datasets but hard at massive scale – set size estimation, load balancing, distinct elements counting (MinHash), checking set membership (Bloom Filters), frequent items counting (Count-min sketch), near neighbor search (locality sensitive hashing).

Randomization as a computational resource for massive datasets.

- Focus on problems that are easy on small datasets but hard at massive scale – set size estimation, load balancing, distinct elements counting (MinHash), checking set membership (Bloom Filters), frequent items counting (Count-min sketch), near neighbor search (locality sensitive hashing).
- Just the tip of the iceberg on randomized streaming/sketching/hashing algorithms.

Randomization as a computational resource for massive datasets.

- Focus on problems that are easy on small datasets but hard at massive scale – set size estimation, load balancing, distinct elements counting (MinHash), checking set membership (Bloom Filters), frequent items counting (Count-min sketch), near neighbor search (locality sensitive hashing).
- Just the tip of the iceberg on randomized streaming/sketching/hashing algorithms.
- In the process covered probability/statistics tools that are very useful beyond algorithm design: concentration inequalities, higher moment bounds, law of large numbers, central limit theorem, linearity of expectation and variance, union bound, median as a robust estimator.

USEFUL PROBABILITY FACTS (1/2)

- Linearity of Expectation: For any random variables $X_1, \ldots, X_n$ and constants $c_1, \ldots, c_n$,

$$\mathbb{E}[c_1 X_1 + \ldots + c_n X_n] = c_1 \mathbb{E}[X_1] + \ldots + c_n \mathbb{E}[X_n]$$

USEFUL PROBABILITY FACTS (1/2)

- Linearity of Expectation: For any random variables $X_1, \ldots, X_n$ and constants $c_1, \ldots, c_n$,

$$\mathbb{E}[c_1 X_1 + \ldots + c_n X_n] = c_1 \mathbb{E}[X_1] + \ldots + c_n \mathbb{E}[X_n]$$

- Independent Random Variables: $X_1, X_2, \ldots X_n$ are independent random variables if for any set $S \subset [n]$ and values $a_1, a_2, \ldots, a_n$

$$\Pr(X_i = a_i \text{ for all } i \in S) = \prod_{i \in S} \Pr(X_i = a_i) \ .$$

  They are k-wise independent if this holds for S with $|S| \leq k$.

USEFUL PROBABILITY FACTS (1/2)

- Linearity of Expectation: For any random variables $X_1, \ldots, X_n$ and constants $c_1, \ldots, c_n$,

$$\mathbb{E}[c_1 X_1 + \ldots + c_n X_n] = c_1 \mathbb{E}[X_1] + \ldots + c_n \mathbb{E}[X_n]$$

- Independent Random Variables: $X_1, X_2, \ldots X_n$ are independent random variables if for any set $S \subset [n]$ and values $a_1, a_2, \ldots, a_n$

$$\Pr(X_i = a_i \text{ for all } i \in S) = \prod_{i \in S} \Pr(X_i = a_i) \ .$$

  They are k-wise independent if this holds for S with $|S| \leq k$.

- Linearity of Variance: If$X_1, \ldots, X_n$ are independent (in fact 2-wise independent suffices) then for any constants $c_1, \ldots, c_n$

$$\text{Var}[c_1 X_1 + \ldots + c_n X_n] = c_1^2 \text{Var}[X_1] + \ldots + c_n^2 \text{Var}[X_n]$$

USEFUL PROBABILITY FACTS (2/2)

- Union Bound: For any events $A_1, A_2, A_3, \ldots$

$$\Pr\left[\bigcup A_i\right] \leq \sum_i \Pr[A_i] \ .$$

- An indicator random variable X just takes the values 0 or 1:

$$\mathbb{E}[X] = p \qquad \text{Var}[X] = p(1-p) \qquad \text{where } p = \Pr[X = 1]$$

- If $Y = X_1 + \ldots + X_n$ where each X_i are independent and $p = \Pr[X_1 = 1] = \ldots = \Pr[X_n = 1]$ then Y is a binomial random variable. Using linearity of expectation and variance,

$$\mathbb{E}[X] = np \qquad \text{Var}[X] = np(1-p)$$

BALLS AND BINS

- Most of the analysis of hash functions that we've considered can be abstracted as "balls and bins" problems: we throw n balls and each ball is equally likely to land in one of m bins.
- Let R_i be number of balls bin i. Then $R_i \sim \text{Bin}(n, \frac{1}{m})$ and $\mathbb{E}[R_i] = \frac{n}{m}$, $\text{Var}[R_i] = \frac{n}{m} \cdot (1 - \frac{1}{m})$. R_i and R_j not independent!
- Union Bound implies $\Pr[\max(R_1, \ldots, R_m) > t] \leq \sum_i \Pr[R_i > t]$
- $Pr[\text{no collisions}] = \frac{m-1}{m} \frac{m-2}{m} \ldots \frac{m-(n-1)}{m}$

$$\Pr[\text{collisions}] = \Pr[\max(R_1, \ldots, R_m) > 1] \leq 1/8 \text{ if } m > 4n^2$$

and more generally

$$\Pr[\max(R_1, \ldots, R_m) \geq 2n/m] \leq m^2/n$$

- In the exam, you'll be expected to do calculations like these.

- Hash function $\mathbf{h} : U \to [n]$ is two universal if:

$$\Pr[\mathbf{h}(x) = \mathbf{h}(y)] \leq \frac{1}{n}.$$

HASH FUNCTIONS

- Hash function $\mathbf{h} : U \to [n]$ is two universal if:

$$\Pr[\mathbf{h}(x) = \mathbf{h}(y)] \leq \frac{1}{n}.$$

- Hash function $\mathbf{h} : U \to [n]$ is k-wise independent if $\{h(e)\}_{e \in U}$ are k-wise independent and each $h(e)$ is uniform in $[n]$.

HASH FUNCTIONS

- Hash function $\mathbf{h} : U \to [n]$ is two universal if:

$$\Pr[\mathbf{h}(x) = \mathbf{h}(y)] \leq \frac{1}{n}.$$

- Hash function $\mathbf{h} : U \to [n]$ is k-wise independent if $\{h(e)\}_{e \in U}$ are k-wise independent and each $h(e)$ is uniform in $[n]$.
- Hash function $\mathbf{h} : U \to [n]$ is fully independent if $\{h(e)\}_{e \in U}$ are independent and each $h(e)$ is uniform in $[n]$.

THREE MAIN CONCENTRATION BOUNDS

- Markov. For any non-negative random variable X and $t > 0$,

$$\Pr[X \geq t] \leq \mathbb{E}[X]/t \ .$$

THREE MAIN CONCENTRATION BOUNDS

- Markov. For any non-negative random variable X and $t > 0$,

$$\Pr[X \geq t] \leq \mathbb{E}[X]/t \ .$$

- Chebyshev. For any random variable X and $t > 0$,

$$\Pr[|X - \mathbb{E}[X]| \geq t] \leq \mathrm{Var}[X]/t^2 \ .$$

THREE MAIN CONCENTRATION BOUNDS

- Markov. For any non-negative random variable X and $t > 0$,

$$\Pr[X \geq t] \leq \mathbb{E}[X]/t \ .$$

- Chebyshev. For any random variable X and $t > 0$,

$$\Pr[|X - \mathbb{E}[X]| \geq t] \leq \mathrm{Var}[X]/t^2 \ .$$

- Chernoff. Let $X_1, \ldots, X_n$ be independent $\{0, 1\}$ random variables with $\mu = \mathbb{E}[\sum_i X_i]$. Then for any $\delta > 0$,

$$\Pr[|(\sum_i X_i) - \mu| \geq \delta\mu] \leq 2 \exp\left(-\frac{\delta^2 \mu}{\delta + 2}\right) \ .$$

THREE MAIN CONCENTRATION BOUNDS

- Markov. For any non-negative random variable X and $t > 0$,

$$\Pr[X \geq t] \leq \mathbb{E}[X]/t \ .$$

- Chebyshev. For any random variable X and $t > 0$,

$$\Pr[|X - \mathbb{E}[X]| \geq t] \leq \mathrm{Var}[X]/t^2 \ .$$

- Chernoff. Let $X_1, \ldots, X_n$ be independent $\{0, 1\}$ random variables with $\mu = \mathbb{E}[\sum_i X_i]$. Then for any $\delta > 0$,

$$\Pr[|(\sum_i X_i) - \mu| \geq \delta\mu] \leq 2\exp\left(-\frac{\delta^2\mu}{\delta + 2}\right) \ .$$

- Generally, Chernoff gives better results then Chebyshev and Chebyshev gives better results than Markov. So choose bound based on how much you know about X.

THREE MAIN CONCENTRATION BOUNDS

- Markov. For any non-negative random variable X and $t > 0$,

$$\Pr[X \geq t] \leq \mathbb{E}[X]/t \ .$$

- Chebyshev. For any random variable X and $t > 0$,

$$\Pr[|X - \mathbb{E}[X]| \geq t] \leq \mathrm{Var}[X]/t^2 \ .$$

- Chernoff. Let $X_1, \ldots, X_n$ be independent $\{0, 1\}$ random variables with $\mu = \mathbb{E}[\sum_i X_i]$. Then for any $\delta > 0$,

$$\Pr[|(\sum_i X_i) - \mu| \geq \delta\mu] \leq 2\exp\left(-\frac{\delta^2\mu}{\delta + 2}\right) \ .$$

- Generally, Chernoff gives better results then Chebyshev and Chebyshev gives better results than Markov. So choose bound based on how much you know about X.
- Bernstein generalizes Chernoff to arbitrary bounded X_i variables.

AVERAGING AND THE MEDIAN TRICK

- Want to learn a quantity q. Suppose you have a randomized algorithm that returns X that has expectation q and variance σ^2.

AVERAGING AND THE MEDIAN TRICK

- Want to learn a quantity q. Suppose you have a randomized algorithm that returns X that has expectation q and variance σ^2.
- To get a good estimate of q, repeat algorithm t times to get $X_1, \ldots, X_t$ and let $A = (X_1 + \ldots + X_t)/t$. Then, if $t = \frac{\sigma^2}{\delta \epsilon^2 q^2}$

$$\Pr[|A - q| \geq \epsilon q] \leq \frac{\mathrm{Var}[A]}{\epsilon^2 q^2}$$

AVERAGING AND THE MEDIAN TRICK

- Want to learn a quantity q. Suppose you have a randomized algorithm that returns X that has expectation q and variance σ^2.
- To get a good estimate of q, repeat algorithm t times to get $X_1, \ldots, X_t$ and let $A = (X_1 + \ldots + X_t)/t$. Then, if $t = \frac{\sigma^2}{\delta \epsilon^2 q^2}$

$$\Pr[|A - q| \geq \epsilon q] \leq \frac{\mathrm{Var}[A]}{\epsilon^2 q^2} = \frac{\sigma^2/t}{\epsilon^2 q^2}$$

AVERAGING AND THE MEDIAN TRICK

- Want to learn a quantity q. Suppose you have a randomized algorithm that returns X that has expectation q and variance σ^2.
- To get a good estimate of q, repeat algorithm t times to get $X_1, \ldots, X_t$ and let $A = (X_1 + \ldots + X_t)/t$. Then, if $t = \frac{\sigma^2}{\delta\epsilon^2 q^2}$

$$\Pr[|A - q| \geq \epsilon q] \leq \frac{\mathrm{Var}[A]}{\epsilon^2 q^2} = \frac{\sigma^2/t}{\epsilon^2 q^2} = \delta$$

AVERAGING AND THE MEDIAN TRICK

- Want to learn a quantity q. Suppose you have a randomized algorithm that returns X that has expectation q and variance σ^2.
- To get a good estimate of q, repeat algorithm t times to get $X_1, \ldots, X_t$ and let $A = (X_1 + \ldots + X_t)/t$. Then, if $t = \frac{\sigma^2}{\delta \epsilon^2 q^2}$

$$\Pr[|A - q| \geq \epsilon q] \leq \frac{\mathrm{Var}[A]}{\epsilon^2 q^2} = \frac{\sigma^2/t}{\epsilon^2 q^2} = \delta$$

- Median Trick: Let $t = t_1 t_2$ where $t_1 = \frac{4\sigma^2}{\epsilon^2 q^2}$ and $t_2 = O(\log \frac{1}{\delta})$. Let A_1 be average of first t_1 results, let A_2 be average of next t_1 results etc. Then,

$$\Pr[|A_i - q| \geq \epsilon q] \leq 1/4$$

and $\Pr[|\mathrm{median}(A_1, \ldots, A_{t_2}) - q| \geq \epsilon q] \leq \delta$.

2-level hash tables vs. bloom filter

- Input to both is a set of items S and and both support queries of the form "Is $x \in S$?" in constant time.

2-level hash tables vs. bloom filter

- Input to both is a set of items S and and both support queries of the form "Is $x \in S$?" in constant time.
- 2-Level Hash Table:
  - Space is $O(|S|) \times$ "space required to store an element of S"

2-level hash tables vs. bloom filter

- Input to both is a set of items S and and both support queries of the form "Is $x \in S$?" in constant time.
- 2-Level Hash Table:
  - Space is $O(|S|) \times$ "space required to store an element of S"
- Bloom Filter:

2-level hash tables vs. bloom filter

- Input to both is a set of items S and and both support queries of the form "Is $x \in S$?" in constant time.
- 2-Level Hash Table:
  - Space is $O(|S|) \times$ "space required to store an element of S"
- Bloom Filter:
  - Does not actually store the items in S, just a binary array from which we make various deductions.

2-level hash tables vs. bloom filter

- Input to both is a set of items S and and both support queries of the form "Is $x \in S$?" in constant time.
- 2-Level Hash Table:
  - Space is $O(|S|)\times$ "space required to store an element of S"
- Bloom Filter:
  - Does not actually store the items in S, just a binary array from which we make various deductions.
  - Uses only $O(|S|)$ space but at the cost of sometimes answering "yes" when answer should be "no" (a false positive)

2-level hash tables vs. Bloom filter

- Input to both is a set of items S and and both support queries of the form "Is $x \in S$?" in constant time.
- 2-Level Hash Table:
  - Space is $O(|S|) \times$ "space required to store an element of S"
- Bloom Filter:
  - Does not actually store the items in S, just a binary array from which we make various deductions.
  - Uses only $O(|S|)$ space but at the cost of sometimes answering "yes" when answer should be "no" (a false positive)
  - If the Bloom Filter array is length m, false positive probability is roughly $(1 - e^{-k|S|/m})^k$ where k is the number of hash functions used. Picking $k = \ln 2 \cdot m/|S|$ gives probability $1/2^{(\ln 2)m/|S|}$

LOCALITY SENSITIVE HASHING

- Designed a hash function for hashing sets such that for sets A and B, $\Pr[MH(A) = MH(B)] = J(A, B) = \frac{|A \cap B|}{|A \cup B|}$.

$$MH(A) = \min_{x \in A} h(x) \quad \text{where} \quad h : U \to [0, 1] \text{ is fully independent}$$

LOCALITY SENSITIVE HASHING

- Designed a hash function for hashing sets such that for sets A and B, $\Pr[MH(A) = MH(B)] = J(A, B) = \frac{|A \cap B|}{|A \cup B|}$.

$$MH(A) = \min_{x \in A} h(x) \quad \text{where} \quad h : U \to [0, 1] \text{ is fully independent}$$

- Can form signature of set A using r independent hash functions:

$$\text{signature}(A) = (MH_1(A), \ldots, MH_r(A))$$

Note $\Pr[\text{signature}(A) = \text{signature}(B)] = J(A, B)^r$.

LOCALITY SENSITIVE HASHING

- Designed a hash function for hashing sets such that for sets A and B, $\Pr[MH(A) = MH(B)] = J(A, B) = \frac{|A \cap B|}{|A \cup B|}$.

$$MH(A) = \min_{x \in A} h(x) \quad \text{where} \quad h : U \to [0, 1] \text{ is fully independent}$$

- Can form signature of set A using r independent hash functions:

$$\text{signature}(A) = (MH_1(A), \ldots, MH_r(A))$$

  Note $\Pr[\text{signature}(A) = \text{signature}(B)] = J(A, B)^r$.

- Given rt independent hash functions, we can form t signatures $\text{signature}_1(A), \ldots \text{signature}_t(A)$. Then if $s = J(A, B)$,

$$\Pr[\text{signature}_i(A) = \text{signature}_i(B) \text{ for some } i] = 1 - (1 - s^r)^t \ .$$

LOCALITY SENSITIVE HASHING

- Designed a hash function for hashing sets such that for sets A and B, $\Pr[MH(A) = MH(B)] = J(A, B) = \frac{|A \cap B|}{|A \cup B|}$.

$$MH(A) = \min_{x \in A} h(x) \quad \text{where} \quad h : U \to [0, 1] \text{ is fully independent}$$

- Can form signature of set A using r independent hash functions:

$$\text{signature}(A) = (MH_1(A), \ldots, MH_r(A))$$

  Note $\Pr[\text{signature}(A) = \text{signature}(B)] = J(A, B)^r$.

- Given rt independent hash functions, we can form t signatures $\text{signature}_1(A), \ldots \text{signature}_t(A)$. Then if $s = J(A, B)$,

$$\Pr[\text{signature}_i(A) = \text{signature}_i(B) \text{ for some } i] = 1 - (1 - s^r)^t \ .$$

- To find all pairs of similar sets amongst $A_1, A_2, A_3, \ldots$ only compare a pair if there exists i, their ith signatures match.

DATA STREAMS ALGORITHMS

- We want to compute something about the stream $x_1, x_2, \ldots, x_m$ with only one pass over the stream and limited space.

DATA STREAMS ALGORITHMS

- We want to compute something about the stream $x_1, x_2, \ldots, x_m$ with only one pass over the stream and limited space.
- Let f_i be the number of values in stream that equal i.

DATA STREAMS ALGORITHMS

- We want to compute something about the stream $x_1, x_2, \ldots, x_m$ with only one pass over the stream and limited space.
- Let f_i be the number of values in stream that equal i.
  - Distinct Items: Can estimate $D = |\{i : f_i > 0\}|$ up to a factor $1 + \epsilon$ with probability $1 - \delta$ in $O(\epsilon^{-2} \log 1/\delta)$ space.

- We want to compute something about the stream $x_1, x_2, \ldots, x_m$ with only one pass over the stream and limited space.
- Let f_i be the number of values in stream that equal i.
  - Distinct Items: Can estimate $D = |\{i : f_i > 0\}|$ up to a factor $1 + \epsilon$ with probability $1 - \delta$ in $O(\epsilon^{-2} \log 1/\delta)$ space.
  - Frequently Elements Items: Can return a set S such that:

$$f_i \geq m/k \text{ implies } i \in S \qquad \text{and} \qquad i \in S \text{ implies } f_i \geq m(1-\epsilon)/k$$

with probability $1 - \delta$ in $O(k/\epsilon \cdot \log 1/\delta)$ space.

- We want to compute something about the stream $x_1, x_2, \ldots, x_m$ with only one pass over the stream and limited space.
- Let f_i be the number of values in stream that equal i.
  - Distinct Items: Can estimate $D = |\{i : f_i > 0\}|$ up to a factor $1 + \epsilon$ with probability $1 - \delta$ in $O(\epsilon^{-2} \log 1/\delta)$ space.
  - Frequently Elements Items: Can return a set S such that:

    $$f_i \geq m/k \text{ implies } i \in S \qquad \text{and} \qquad i \in S \text{ implies } f_i \geq m(1-\epsilon)/k$$

    with probability $1 - \delta$ in $O(k/\epsilon \cdot \log 1/\delta)$ space.
  - Sampling and Averaging Distinct Elements: Apply hash function $h : U \to [0, 1]$ to each stream element. The element x with the smallest value of $h(x)$ is a uniform sample from the stream.

Thanks for a great semester!