

CSCI 4360/6360 Data Science II

# Randomized SVD

# Singular Value Decomposition (SVD)

- Given a (any!) matrix  $M$ , which is  $n \times m$ , it can be represented as

$$M = U\Sigma V^T$$

- $U$ :  $n \times n$ , unitary matrix (orthogonal)
- $\Sigma$ :  $n \times m$ , diagonal matrix of singular values
- $V^T$ :  $m \times m$ , unitary matrix (orthogonal)

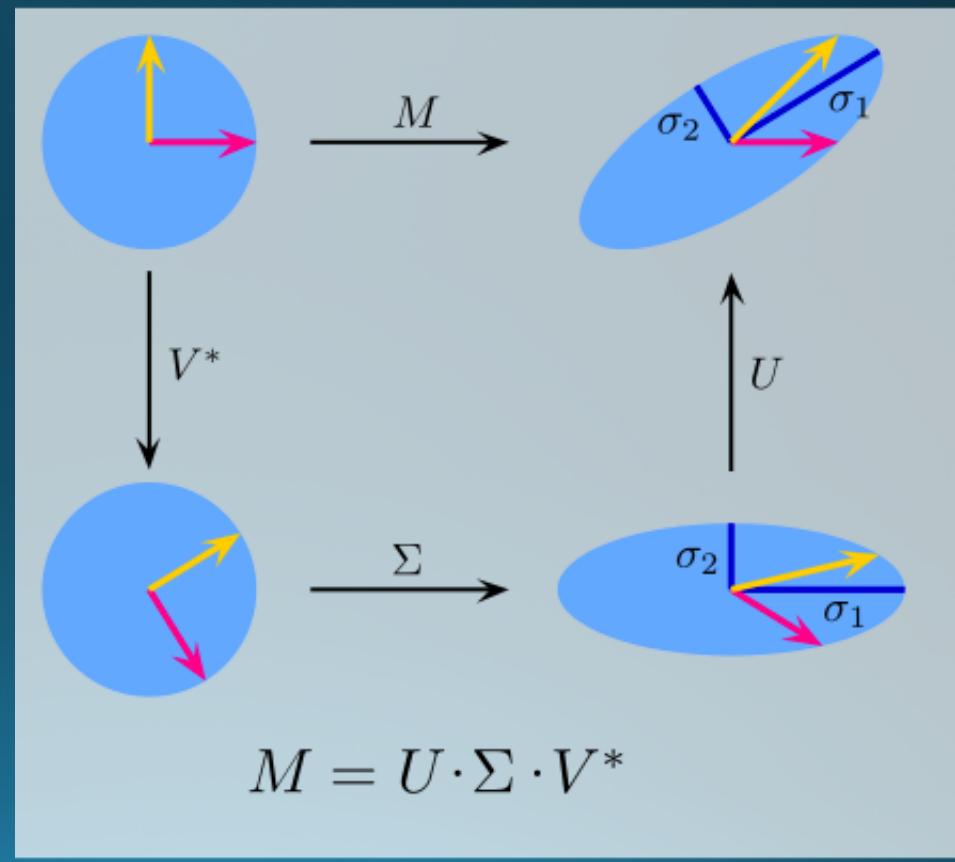
# SVD

$$\mathbf{A}_{[m \times n]} = \mathbf{U}_{[m \times r]} \Sigma_{[r \times r]} (\mathbf{V}_{[n \times r]})^T$$

- **A: Input data matrix**
  - $m \times n$  matrix (e.g.,  $m$  documents,  $n$  terms)
- **U: Left singular vectors**
  - $m \times r$  matrix ( $m$  documents,  $r$  concepts)
- **$\Sigma$ : Singular values**
  - $r \times r$  diagonal matrix (strength of each 'concept')  
( $r$  : rank of the matrix  $\mathbf{A}$ )
- **V: Right singular vectors**
  - $n \times r$  matrix ( $n$  terms,  $r$  concepts)

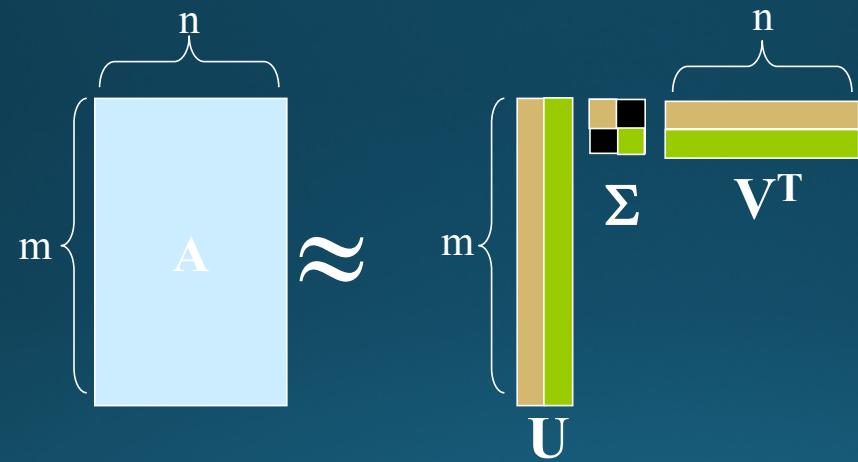
# Singular Value Decomposition (SVD)

- Columns of  $U$  and  $V$  are orthonormal bases
- Singular values are the “strength” of each singular vector



# SVD

$$\mathbf{A} \approx \mathbf{U}\Sigma\mathbf{V}^T = \sum_i \sigma_i \mathbf{u}_i \circ \mathbf{v}_i^\top$$



# SVD

$$\mathbf{A} \approx \mathbf{U}\Sigma\mathbf{V}^T = \sum_i \sigma_i \mathbf{u}_i \circ \mathbf{v}_i^\top$$



$\sigma_i$  ... scalar  
 $\mathbf{u}_i$  ... vector  
 $\mathbf{v}_i$  ... vector

# SVD - Properties

It is **always** possible to decompose a real matrix  $\mathbf{A}$  into  $\mathbf{A} = \mathbf{U} \Sigma \mathbf{V}^T$ , where

- $\mathbf{U}, \Sigma, \mathbf{V}$ : unique
- $\mathbf{U}, \mathbf{V}$ : column orthonormal
  - $\mathbf{U}^T \mathbf{U} = \mathbf{I}; \mathbf{V}^T \mathbf{V} = \mathbf{I}$  ( $\mathbf{I}$ : identity matrix)
  - (Columns are orthogonal unit vectors)
- $\Sigma$ : diagonal
  - Entries (**singular values**) are positive, and sorted in decreasing order ( $\sigma_1 \geq \sigma_2 \geq \dots \geq 0$ )

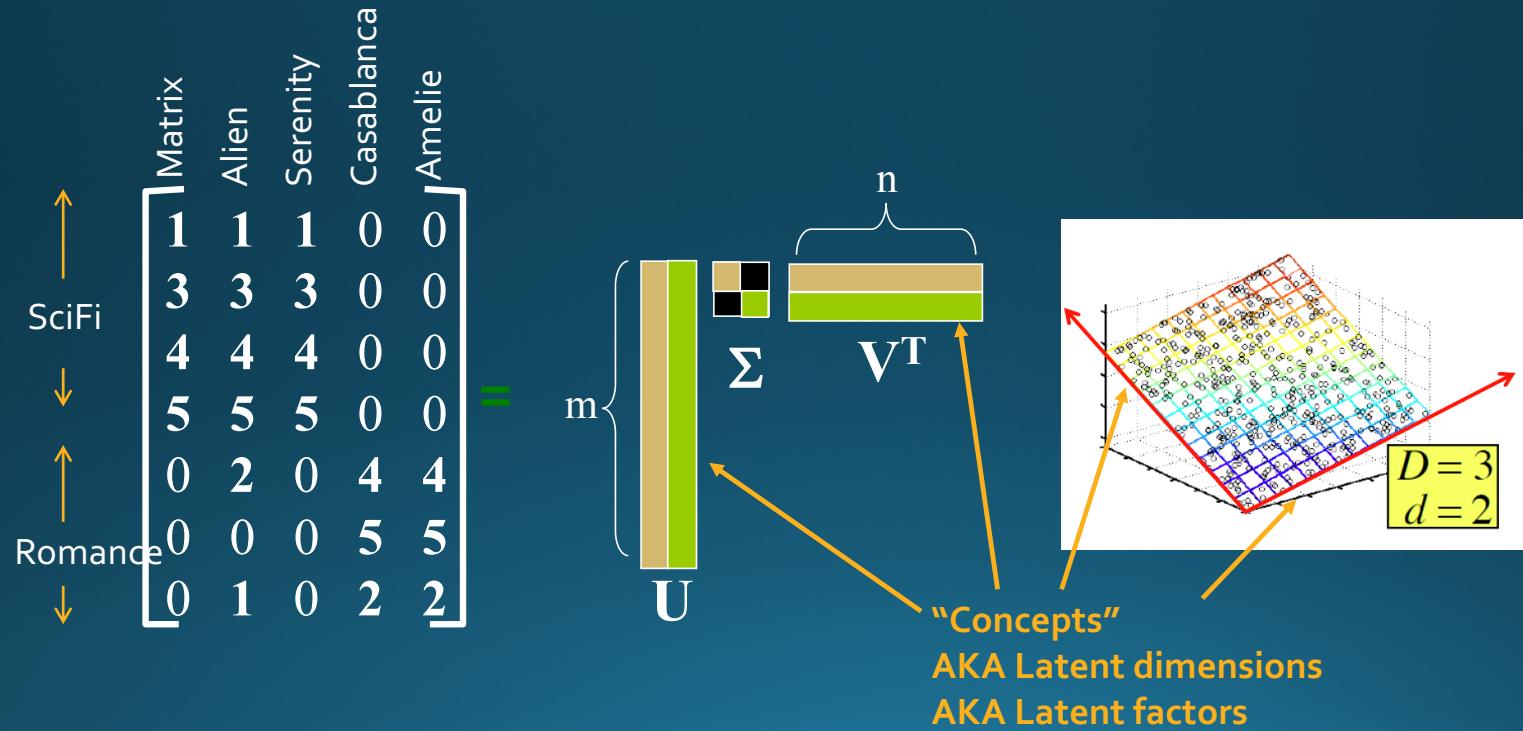
Nice proof of uniqueness: <http://www.mpi-inf.mpg.de/~bast/ir-seminar-wso4/lecture2.pdf>

# Why randomize SVD?

- **Runtime**
  - We're good at generating [pseudo-]random numbers
  - Can easily parallelize / distribute this process
  - SVD, like PCA, runs  $O(n^3)$ , making anything beyond  $\sim 10^3$  infeasible

# SVD – Example: Users-to-Movies

- $A = U \Sigma V^T$  - example: Users to Movies



# SVD – Example: Users-to-Movies

- $A = U \Sigma V^T$  - example: Users to Movies

$$\begin{array}{c}
 \text{Matrix} \\
 \begin{array}{c}
 \uparrow \quad \downarrow \\
 \text{SciFi} \quad \text{Romance}
 \end{array} \\
 \left[ \begin{array}{ccccc}
 1 & 1 & 1 & 0 & 0 \\
 3 & 3 & 3 & 0 & 0 \\
 4 & 4 & 4 & 0 & 0 \\
 5 & 5 & 5 & 0 & 0 \\
 0 & 2 & 0 & 4 & 4 \\
 0 & 0 & 0 & 5 & 5 \\
 0 & 1 & 0 & 2 & 2
 \end{array} \right] = \left[ \begin{array}{ccc}
 0.13 & 0.02 & -0.01 \\
 0.41 & 0.07 & -0.03 \\
 0.55 & 0.09 & -0.04 \\
 0.68 & 0.11 & -0.05 \\
 0.15 & -0.59 & 0.65 \\
 0.07 & -0.73 & -0.67 \\
 0.07 & -0.29 & 0.32
 \end{array} \right] \times \left[ \begin{array}{ccc}
 12.4 & 0 & 0 \\
 0 & 9.5 & 0 \\
 0 & 0 & 1.3
 \end{array} \right] \times \left[ \begin{array}{cccccc}
 0.56 & 0.59 & 0.56 & 0.09 & 0.09 \\
 0.12 & -0.02 & 0.12 & -0.69 & -0.69 \\
 0.40 & -0.80 & 0.40 & 0.09 & 0.09
 \end{array} \right]
 \end{array}$$

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, <http://www.mmds.org>

# SVD – Example: Users-to-Movies

- $A = U \Sigma V^T$  - example: Users to Movies

$$\begin{array}{c}
 \text{Matrix} \\
 \begin{bmatrix} & \text{Matrix} & \text{Alien} & \text{Serenity} & \text{Casablanca} & \text{Amelie} \\ 
 \text{SciFi} & \uparrow & 1 & 1 & 1 & 0 & 0 \\ 
 \downarrow & & 3 & 3 & 3 & 0 & 0 \\ 
 \text{Romance} & \uparrow & 4 & 4 & 4 & 0 & 0 \\ 
 \downarrow & & 5 & 5 & 5 & 0 & 0 \\ 
 & & 0 & 2 & 0 & 4 & 4 \\ 
 & & 0 & 0 & 0 & 5 & 5 \\ 
 & & 0 & 1 & 0 & 2 & 2 \end{bmatrix} = 
 \begin{array}{l}
 \text{SciFi-concept} \quad \downarrow \\
 \begin{bmatrix} 0.13 & 0.02 & -0.01 \\ 0.41 & 0.07 & -0.03 \\ 0.55 & 0.09 & -0.04 \\ 0.68 & 0.11 & -0.05 \\ 0.15 & -0.59 & 0.65 \\ 0.07 & -0.73 & -0.67 \\ 0.07 & -0.29 & 0.32 \end{bmatrix} \times 
 \begin{bmatrix} 12.4 & 0 & 0 \\ 0 & 9.5 & 0 \\ 0 & 0 & 1.3 \end{bmatrix} \times 
 \begin{bmatrix} 0.56 & 0.59 & 0.56 & 0.09 & 0.09 \\ 0.12 & -0.02 & 0.12 & -0.69 & -0.69 \\ 0.40 & -0.80 & 0.40 & 0.09 & 0.09 \end{bmatrix} \\
 \text{Romance-concept} \quad \nearrow
 \end{array}
 \end{array}$$

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, <http://www.mmds.org>

# SVD – Example: Users-to-Movies

- $A = U \Sigma V^T$  - example:

$U$  is “user-to-concept”  
similarity matrix

$$\begin{array}{c}
 \text{Matrix} \\
 \begin{array}{c}
 \uparrow \quad \downarrow \\
 \text{SciFi} \quad \text{Romance}
 \end{array} \\
 \left[ \begin{array}{ccccc}
 1 & 1 & 1 & 0 & 0 \\
 3 & 3 & 3 & 0 & 0 \\
 4 & 4 & 4 & 0 & 0 \\
 5 & 5 & 5 & 0 & 0 \\
 0 & 2 & 0 & 4 & 4 \\
 0 & 0 & 0 & 5 & 5 \\
 0 & 1 & 0 & 2 & 2
 \end{array} \right] = \left[ \begin{array}{ccc}
 \text{SciFi-concept} & \text{Romance-concept} \\
 \downarrow & \downarrow \\
 0.13 & 0.02 & -0.01 \\
 0.41 & 0.07 & -0.03 \\
 0.55 & 0.09 & -0.04 \\
 0.68 & 0.11 & -0.05 \\
 0.15 & -0.59 & 0.65 \\
 0.07 & -0.73 & -0.67 \\
 0.07 & -0.29 & 0.32
 \end{array} \right] \times \left[ \begin{array}{ccc}
 12.4 & 0 & 0 \\
 0 & 9.5 & 0 \\
 0 & 0 & 1.3
 \end{array} \right] \times \left[ \begin{array}{cccccc}
 0.56 & 0.59 & 0.56 & 0.09 & 0.09 \\
 0.12 & -0.02 & 0.12 & -0.69 & -0.69 \\
 0.40 & -0.80 & 0.40 & 0.09 & 0.09
 \end{array} \right]
 \end{array}$$

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, <http://www.mmds.org>

# SVD – Example: Users-to-Movies

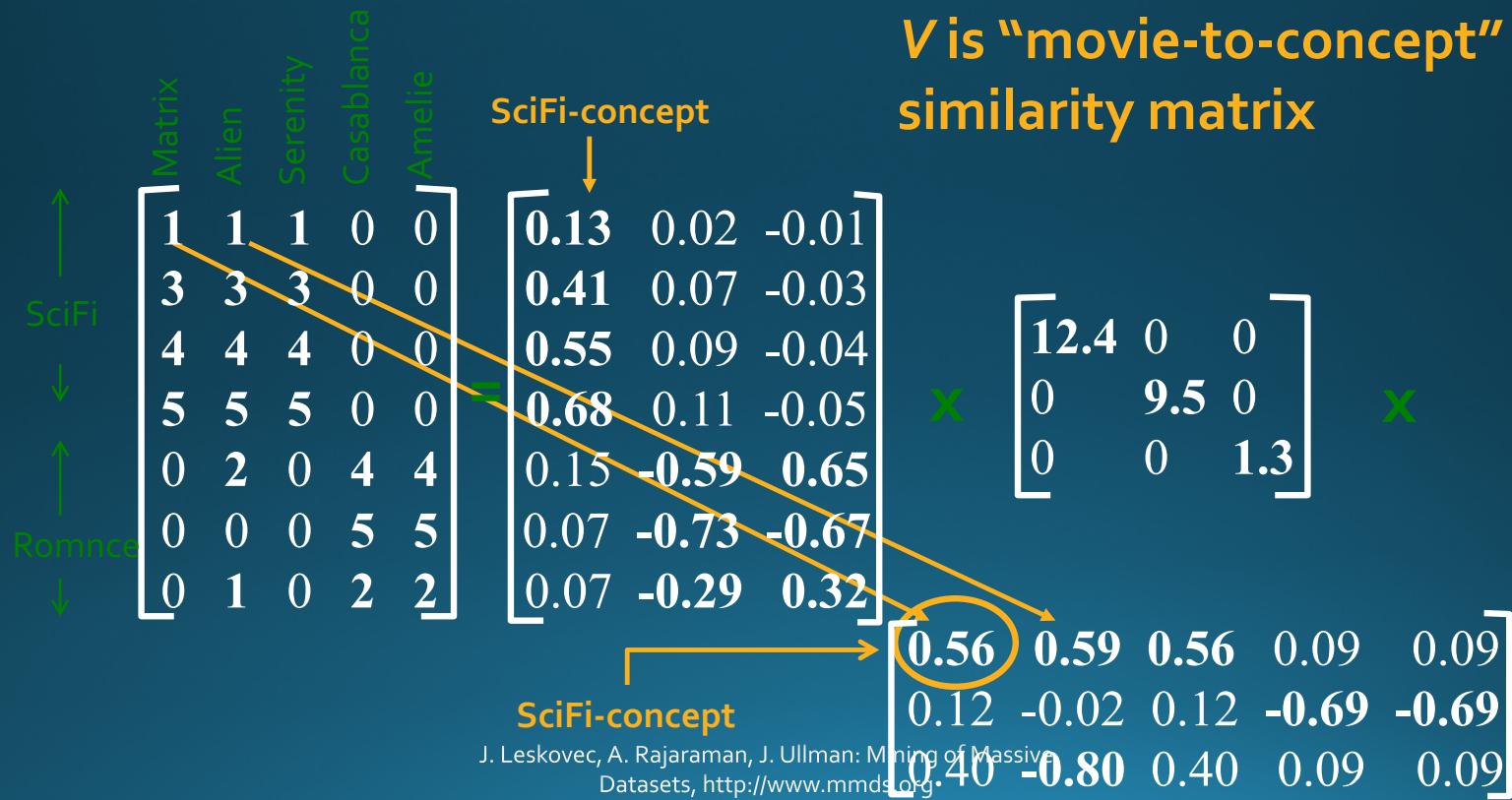
- $A = U \Sigma V^T$  - example:

$$\begin{array}{c}
 \text{Matrix} \\
 \begin{array}{c}
 \uparrow \quad \downarrow \\
 \text{SciFi} \quad \text{Romance}
 \end{array}
 \end{array}
 \begin{bmatrix}
 \text{Matrix} & \text{Alien} & \text{Serenity} & \text{Casablanca} & \text{Amelie} \\
 \begin{bmatrix}
 1 & 1 & 1 & 0 & 0 \\
 3 & 3 & 3 & 0 & 0 \\
 4 & 4 & 4 & 0 & 0 \\
 5 & 5 & 5 & 0 & 0 \\
 0 & 2 & 0 & 4 & 4 \\
 0 & 0 & 0 & 5 & 5 \\
 0 & 1 & 0 & 2 & 2
 \end{bmatrix} & = & \begin{array}{c}
 \text{SciFi-concept} \\
 \downarrow \\
 \begin{bmatrix}
 0.13 & 0.02 & -0.01 \\
 0.41 & 0.07 & -0.03 \\
 \mathbf{0.55} & 0.09 & -0.04 \\
 \mathbf{0.68} & 0.11 & -0.05 \\
 0.15 & -0.59 & 0.65 \\
 0.07 & -0.73 & -0.67 \\
 0.07 & -0.29 & 0.32
 \end{bmatrix} \\
 \times \quad \quad \quad \times
 \end{array} \\
 & & \begin{array}{c}
 \text{"strength" of the SciFi-concept} \\
 \rightarrow \\
 \boxed{12.4} \quad 0 \quad 0 \\
 0 \quad 9.5 \quad 0 \\
 0 \quad 0 \quad 1.3
 \end{array} \\
 & & \begin{bmatrix}
 0.56 & 0.59 & 0.56 & 0.09 & 0.09 \\
 0.12 & -0.02 & 0.12 & -0.69 & -0.69 \\
 0.40 & -0.80 & 0.40 & 0.09 & 0.09
 \end{bmatrix}
 \end{array}$$

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, <http://www.mmds.org>

# SVD – Example: Users-to-Movies

- $A = U \Sigma V^T$  - example:



J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, <http://www.mmds.org>

# SVD - Interpretation #1

'movies', 'users' and 'concepts':

- $U$ : user-to-concept similarity matrix
- $V$ : movie-to-concept similarity matrix
- $\Sigma$ : its diagonal elements:  
‘strength’ of each concept

# SVD - Interpretation #2

## More details

- Q: How exactly is dim. reduction done?
- A: Set smallest singular values to zero

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 2 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 1 & 0 & 2 & 2 \end{bmatrix} = \begin{bmatrix} 0.13 & 0.02 & -0.01 \\ 0.41 & 0.07 & -0.03 \\ 0.55 & 0.09 & -0.04 \\ 0.68 & 0.11 & -0.05 \\ 0.15 & -0.59 & 0.65 \\ 0.07 & -0.73 & -0.67 \\ 0.07 & -0.29 & 0.32 \end{bmatrix} \times \begin{bmatrix} 12.4 & 0 & 0 \\ 0 & 9.5 & 0 \\ 0 & 0 & 1.3 \end{bmatrix} \times \begin{bmatrix} 0.56 & 0.59 & 0.56 & 0.09 & 0.09 \\ 0.12 & -0.02 & 0.12 & -0.69 & -0.69 \\ 0.40 & -0.80 & 0.40 & 0.09 & 0.09 \end{bmatrix}$$

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, <http://www.mmds.org>

# SVD - Interpretation #2

## More details

- Q: How exactly is dim. reduction done?
- A: Set smallest singular values to zero

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ \textcolor{red}{5} & \textcolor{red}{5} & \textcolor{red}{5} & 0 & 0 \\ 0 & 2 & 0 & 4 & 4 \\ 0 & 0 & 0 & \textcolor{blue}{5} & \textcolor{blue}{5} \\ 0 & 1 & 0 & 2 & 2 \end{bmatrix} \approx \begin{bmatrix} 0.13 & 0.02 & -0.01 \\ \textcolor{red}{0.41} & 0.07 & -0.03 \\ \textcolor{blue}{0.55} & 0.09 & -0.04 \\ \textcolor{red}{0.68} & 0.11 & -0.05 \\ 0.15 & \textcolor{red}{-0.59} & \textcolor{blue}{0.65} \\ 0.07 & \textcolor{red}{-0.73} & \textcolor{red}{-0.67} \\ 0.07 & \textcolor{red}{-0.29} & \textcolor{blue}{0.32} \end{bmatrix} \times \begin{bmatrix} 12.4 & 0 & 0 \\ 0 & 9.5 & 0 \\ 0 & 0 & \cancel{1.3} \end{bmatrix} \times \begin{bmatrix} 0.56 & 0.59 & 0.56 & 0.09 & 0.09 \\ 0.12 & -0.02 & 0.12 & \textcolor{red}{-0.69} & \textcolor{red}{-0.69} \\ 0.40 & \textcolor{red}{-0.80} & 0.40 & 0.09 & 0.09 \end{bmatrix}$$

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, <http://www.mmds.org>

# SVD - Interpretation #2

## More details

- Q: How exactly is dim. reduction done?
- A: Set smallest singular values to zero

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ \textcolor{red}{5} & \textcolor{red}{5} & \textcolor{red}{5} & 0 & 0 \\ 0 & 2 & 0 & 4 & 4 \\ 0 & 0 & 0 & \textcolor{blue}{5} & \textcolor{blue}{5} \\ 0 & 1 & 0 & 2 & 2 \end{bmatrix} \approx \begin{bmatrix} 0.13 & 0.02 & -0.01 \\ \textcolor{red}{0.41} & 0.07 & -0.03 \\ \textcolor{blue}{0.55} & 0.09 & -0.04 \\ \textcolor{red}{0.68} & 0.11 & -0.05 \\ 0.15 & \textcolor{red}{-0.59} & \textcolor{red}{0.65} \\ 0.07 & \textcolor{red}{-0.73} & \textcolor{red}{-0.67} \\ 0.07 & \textcolor{red}{-0.29} & \textcolor{red}{0.32} \end{bmatrix} \times \begin{bmatrix} 12.4 & 0 & 0 \\ 0 & 9.5 & 0 \\ 0 & 0 & \textcolor{red}{1.3} \end{bmatrix} \times \begin{bmatrix} 0.56 & 0.59 & 0.56 & 0.09 & 0.09 \\ 0.12 & -0.02 & 0.12 & \textcolor{red}{-0.69} & \textcolor{red}{-0.69} \\ 0.40 & \textcolor{red}{-0.80} & 0.40 & 0.09 & 0.09 \end{bmatrix}$$

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, <http://www.mmds.org>

# SVD - Interpretation #2

## More details

- Q: How exactly is dim. reduction done?
- A: Set smallest singular values to zero

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ \textcolor{red}{5} & \textcolor{red}{5} & \textcolor{red}{5} & 0 & 0 \\ 0 & 2 & 0 & 4 & 4 \\ 0 & 0 & 0 & \textcolor{blue}{5} & \textcolor{blue}{5} \\ 0 & 1 & 0 & 2 & 2 \end{bmatrix} \approx \begin{bmatrix} 0.13 & 0.02 \\ 0.41 & 0.07 \\ \textcolor{blue}{0.55} & 0.09 \\ \textcolor{blue}{0.68} & 0.11 \\ 0.15 & \textcolor{red}{-0.59} \\ 0.07 & \textcolor{red}{-0.73} \\ 0.07 & \textcolor{red}{-0.29} \end{bmatrix} \times \begin{bmatrix} 12.4 & 0 \\ 0 & 9.5 \end{bmatrix} \times \begin{bmatrix} 0.56 & 0.59 & 0.56 & 0.09 & 0.09 \\ 0.12 & -0.02 & 0.12 & \textcolor{red}{-0.69} & \textcolor{red}{-0.69} \end{bmatrix}$$

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, <http://www.mmds.org>

# SVD - Interpretation #2

## More details

- Q: How exactly is dim. reduction done?
- A: Set smallest singular values to zero

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 2 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 1 & 0 & 2 & 2 \end{bmatrix} \approx \begin{bmatrix} 0.92 & 0.95 & 0.92 & 0.01 & 0.01 \\ 2.91 & 3.01 & 2.91 & -0.01 & -0.01 \\ 3.90 & 4.04 & 3.90 & 0.01 & 0.01 \\ 4.82 & 5.00 & 4.82 & 0.03 & 0.03 \\ 0.70 & 0.53 & 0.70 & 4.11 & 4.11 \\ -0.69 & 1.34 & -0.69 & 4.78 & 4.78 \\ 0.32 & 0.23 & 0.32 & 2.01 & 2.01 \end{bmatrix}$$

$\|A-B\|_F = \sqrt{\sum_{ij} (A_{ij}-B_{ij})^2}$   
is “small”

Frobenius norm:

$$\|M\|_F = \sqrt{\sum_{ij} M_{ij}^2}$$

# SVD – Best Low Rank Approx.

$$A = U \Sigma V^T$$

**B is best approximation of A**

$$B = U \Sigma V^T$$

# Relationship to PCA

- SVD can be applied to *any* matrix; PCA only works on symmetric covariance matrices
- However, there is a relationship

$$M^T M = V \Sigma^T U^T U \Sigma V^T = V (\Sigma^T \Sigma) V^T$$
$$M M^T = U \Sigma V^T V \Sigma^T U^T = U (\Sigma \Sigma^T) U^T$$

- Columns of  $V$  are eigenvectors of  $M^T M$
- Columns of  $U$  are eigenvectors of  $M M^T$
- Singular values are square roots of eigenvalues of  $M^T M$  or  $M M^T$

# SVD: Drawbacks

+ Optimal low-rank approximation

in terms of Frobenius norm

- Interpretability problem:

- A singular vector specifies a linear combination of all input columns or rows

- Lack of sparsity:

- Singular vectors are **dense!**

$$\begin{matrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \\ \cdot & \cdot \end{matrix} = \begin{matrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \\ \cdot & \cdot \end{matrix} \Sigma \begin{matrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{matrix} V^T$$

# CUR Decomposition

Frobenius norm:  
 $\|X\|_F = \sqrt{\sum_{ij} X_{ij}^2}$

- Goal: Express  $A$  as a product of matrices  $C, U, R$

Make  $\|A - C \cdot U \cdot R\|_F$  small

- “Constraints” on  $C$  and  $R$ :

$$\left( \begin{array}{c|c|c} & & \\ \hline & A & \\ & & \end{array} \right) \approx \left( \begin{array}{c|c|c|c} & & & \\ \hline & C & & \\ & & & \\ & & & \\ & & & \end{array} \right) \cdot \left( \begin{array}{c} U \\ \hline \end{array} \right) \cdot \left( \begin{array}{c} R \\ \hline \end{array} \right)$$

# CUR Decomposition

Frobenius norm:  
 $\|X\|_F = \sqrt{\sum_{ij} X_{ij}^2}$

- Goal: Express  $A$  as a product of matrices  $C, U, R$

Make  $\|A - C \cdot U \cdot R\|_F$  small

- “Constraints” on  $C$  and  $R$ :

$$\begin{pmatrix} \text{---} \\ A \\ \text{---} \end{pmatrix} \approx \begin{pmatrix} \text{---} \\ C \\ \text{---} \end{pmatrix} \cdot \begin{pmatrix} \text{---} \\ U \\ \text{---} \end{pmatrix} \cdot \begin{pmatrix} \text{---} \\ R \\ \text{---} \end{pmatrix}$$

A                    C                    U                    R

Pseudo-inverse of  
the intersection of  $C$  and  $R$

# CUR: How it Works

- **Sampling columns (similarly for rows):**

Note this is a randomized algorithm; the same column can be sampled more than once

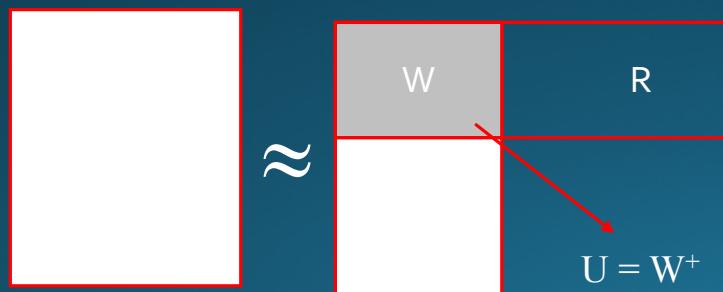
**Input:** matrix  $\mathbf{A} \in \mathbb{R}^{m \times n}$ , sample size  $c$

**Output:**  $\mathbf{C}_d \in \mathbb{R}^{m \times c}$

1. for  $x = 1 : n$  [column distribution]
2.  $P(x) = \sum_i \mathbf{A}(i, x)^2 / \sum_{i,j} \mathbf{A}(i, j)^2$
3. for  $i = 1 : c$  [sample columns]
4. Pick  $j \in 1 : n$  based on distribution  $P(x)$
5. Compute  $\mathbf{C}_d(:, i) = \mathbf{A}(:, j) / \sqrt{cP(j)}$

# Computing U

- Let  $\mathbf{W}$  be the “intersection” of sampled columns  $\mathbf{C}$  and rows  $\mathbf{R}$ 
  - Let SVD of  $\mathbf{W} = \mathbf{X} \mathbf{Z} \mathbf{Y}^T$
- Then:  $\mathbf{U} = \mathbf{W}^+ = \mathbf{Y} \mathbf{Z}^+ \mathbf{X}^T$ 
  - $\mathbf{Z}^+$ : **reciprocals of non-zero singular values**:  $Z_{ii}^+ = 1/Z_{ii}$
  - $\mathbf{W}^+$  is the “**pseudoinverse**”



## Why pseudoinverse works?

$\mathbf{W} = \mathbf{X} \mathbf{Z} \mathbf{Y}^T$  then  $\mathbf{W}^{-1} = \mathbf{X}^{-1} \mathbf{Z}^{-1} \mathbf{Y}^{-1}$

Due to orthonormality

$\mathbf{X}^{-1} = \mathbf{X}^T$  and  $\mathbf{Y}^{-1} = \mathbf{Y}^T$

Since  $\mathbf{Z}$  is diagonal  $\mathbf{Z}^{-1} = 1/Z_{ii}$

Thus, if  $\mathbf{W}$  is nonsingular,  
pseudoinverse is the true  
inverse

# CUR: Pros & Cons

## + Easy interpretation

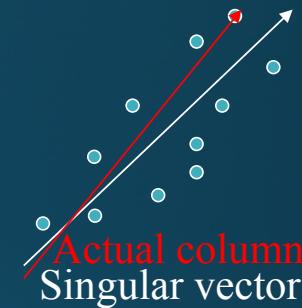
- Since the basis vectors are actual columns and rows

## + Sparse basis

- Since the basis vectors are actual columns and rows

## - Duplicate columns and rows

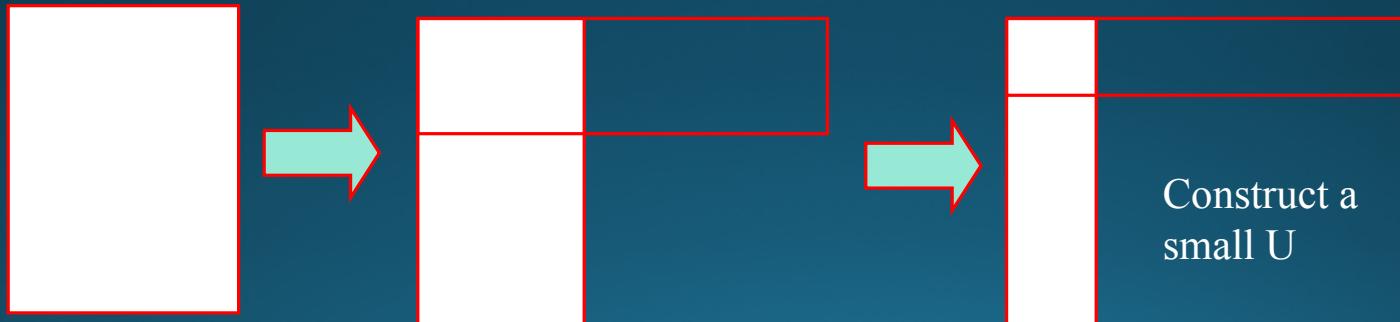
- Columns of large norms will be sampled many times



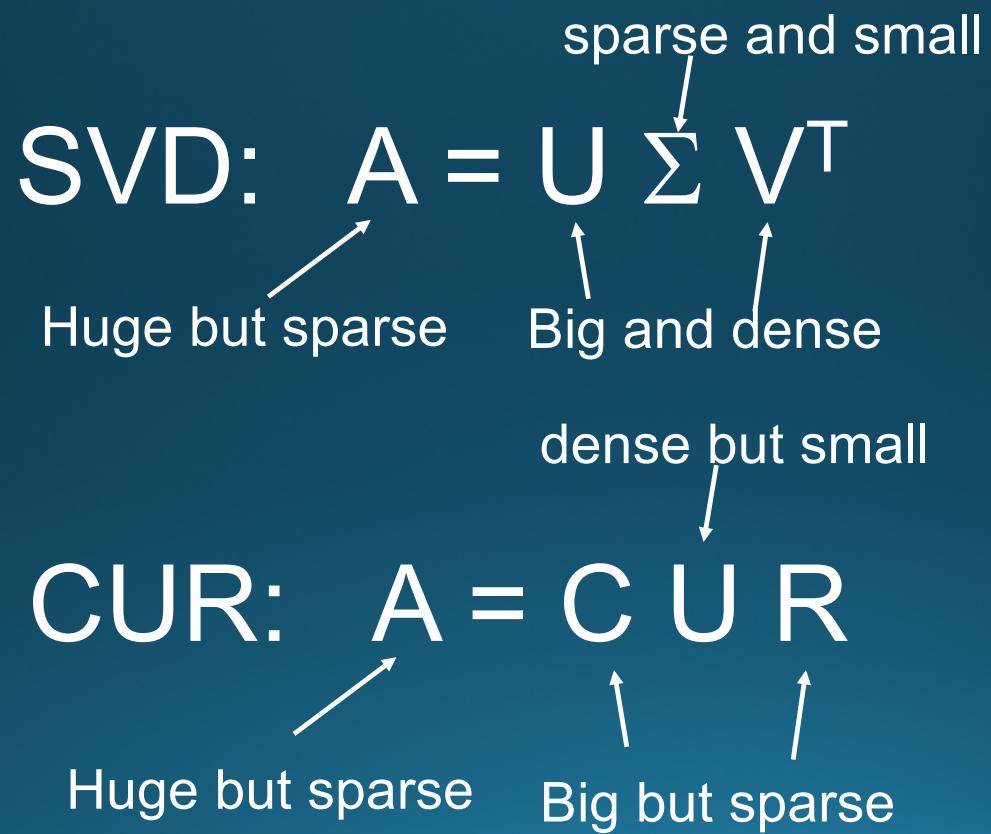
# Solution

- If we want to get rid of the duplicates:

- Throw them away
- Scale (multiply) the columns/rows by the square root of the number of duplicates



# SVD vs. CUR



J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, <http://www.mmds.org>

# Large-Scale Matrix Factorization with Distributed Stochastic Gradient Descent

Rainer Gemulla



talk pilfered from  
→

Peter J. Haas



Yannis Sismanis



Erik Nijkamp



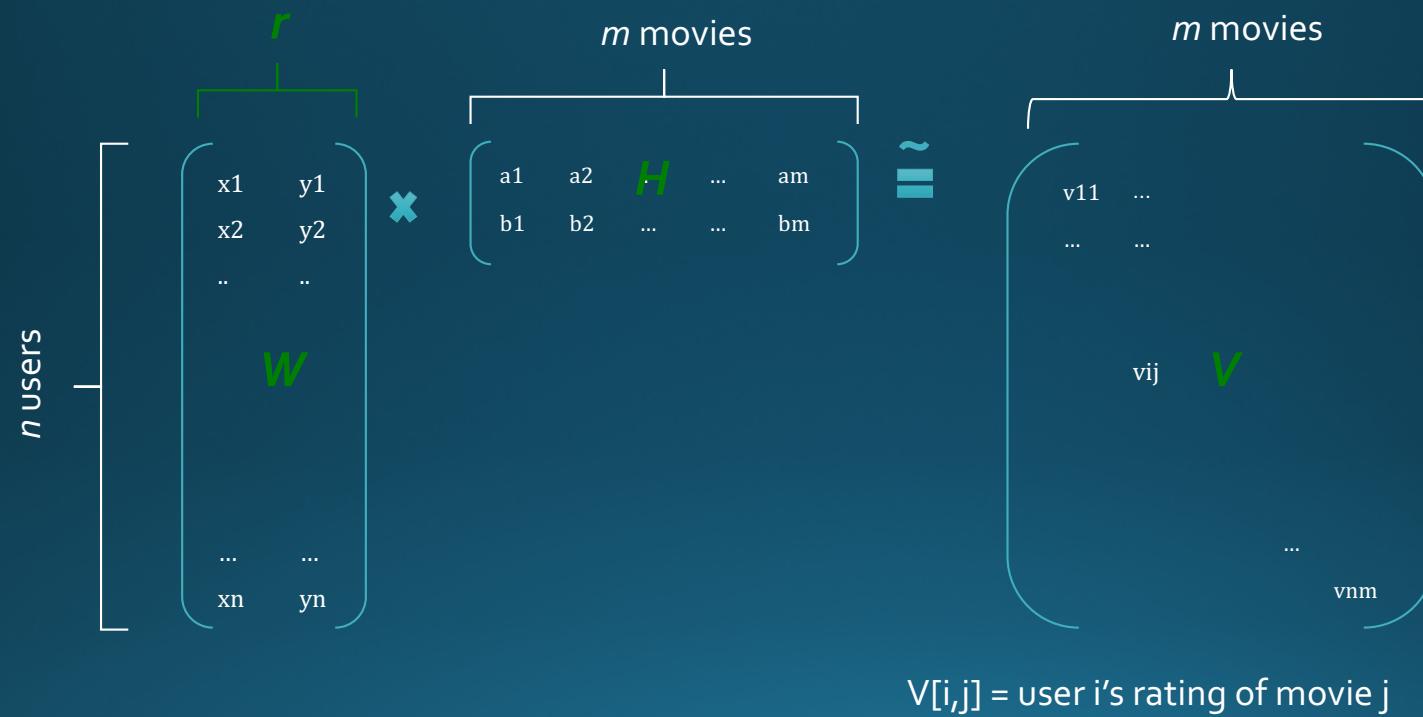
## Collaborative Filtering

- ▶ Problem
  - ▶ Set of users
  - ▶ Set of items (movies, books, jokes, products, stories, ...)
  - ▶ Feedback (ratings, purchase, click-through, tags, ...)
- ▶ Predict additional items a user may like
  - ▶ Assumption: Similar feedback  $\implies$  Similar taste
- ▶ Example

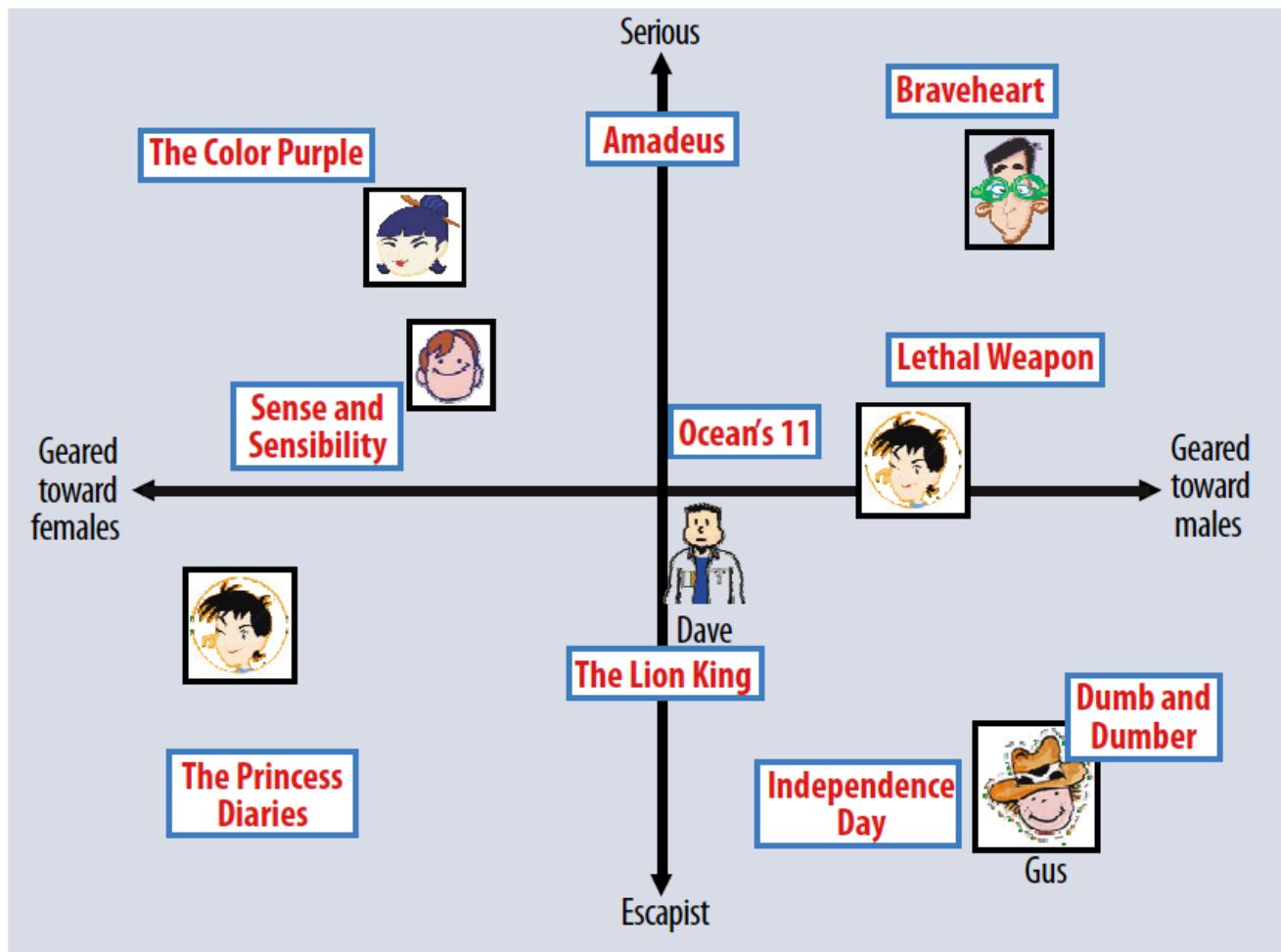
	<i>Avatar</i>	<i>The Matrix</i>	<i>Up</i>
<i>Alice</i>	?	4	2
<i>Bob</i>	3	2	?
<i>Charlie</i>	5	?	3

- ▶ Netflix competition: 500k users, 20k movies, 100M movie ratings, 3M question marks

# Recovering latent factors in a matrix



## Semantic Factors (Koren et al., 2009)



## Latent Factor Models

- ▶ Discover latent factors ( $r = 1$ )

	<b>Avatar</b> (2.24)	<b>The Matrix</b> (1.92)	<b>Up</b> (1.18)
<b>Alice</b> (1.98)		<b>4</b> (3.8)	<b>2</b> (2.3)
<b>Bob</b> (1.21)	<b>3</b> (2.7)	<b>2</b> (2.3)	
<b>Charlie</b> (2.30)	<b>5</b> (5.2)		<b>3</b> (2.7)

- ▶ Minimum loss

$$\min_{\mathbf{W}, \mathbf{H}} \sum_{(i,j) \in Z} (\mathbf{V}_{ij} - [\mathbf{WH}]_{ij})^2$$

## Latent Factor Models

- ▶ Discover latent factors ( $r = 1$ )

	Avatar (2.24)	The Matrix (1.92)	Up (1.18)
Alice (1.98)	?	4 (3.8)	2 (2.3)
Bob (1.21)	3 (2.7)	2 (2.3)	?
Charlie (2.30)	5 (5.2)	?	3 (2.7)

- ▶ Minimum loss

$$\min_{\mathbf{W}, \mathbf{H}, \mathbf{u}, \mathbf{m}} \sum_{(i,j) \in Z} (\mathbf{v}_{ij} - \mu - \mathbf{u}_i - \mathbf{m}_j - [\mathbf{WH}]_{ij})^2 + \lambda (\|\mathbf{W}\| + \|\mathbf{H}\| + \|\mathbf{u}\| + \|\mathbf{m}\|)$$

- ▶ Bias, regularization

# Matrix factorization as SGD

require that the loss can be written as

$$L = \sum_{(i,j) \in Z} l(\mathbf{V}_{ij}, \mathbf{W}_{i*}, \mathbf{H}_{*j})$$

---

## Algorithm 1 SGD for Matrix Factorization

---

**Require:** A training set  $Z$ , initial values  $\mathbf{W}_0$  and  $\mathbf{H}_0$

**while** not converged **do** {step}

    Select a training point  $(i, j) \in Z$  uniformly at random.

$$\mathbf{W}'_{i*} \leftarrow \mathbf{W}_{i*} - \epsilon_n N \frac{\partial}{\partial \mathbf{W}_{i*}} l(\mathbf{V}_{ij}, \mathbf{W}_{i*}, \mathbf{H}_{*j})$$

$$\mathbf{H}_{*j} \leftarrow \mathbf{H}_{*j} - \epsilon_n N \frac{\partial}{\partial \mathbf{H}_{*j}} l(\mathbf{V}_{ij}, \mathbf{W}_{i*}, \mathbf{H}_{*j})$$

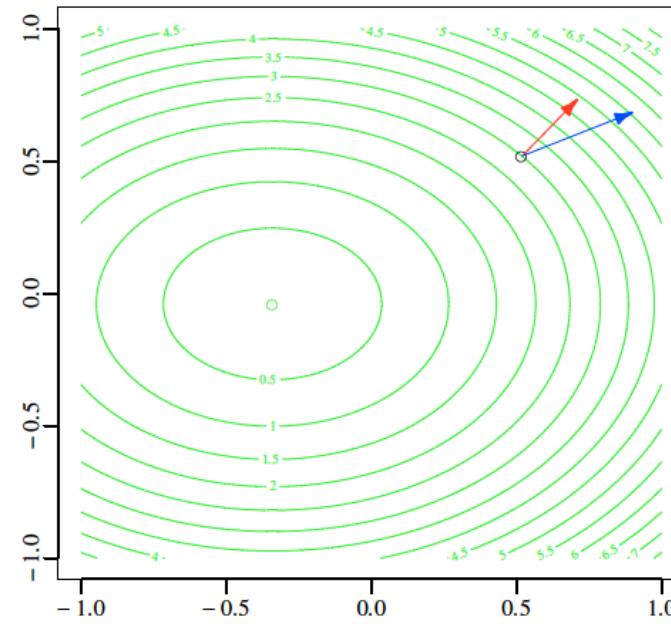
$$\mathbf{W}_{i*} \leftarrow \mathbf{W}'_{i*}$$

**end while**

why does this work?

## Stochastic Gradient Descent

- ▶ Find minimum  $\theta^*$  of function  $L$
- ▶ Pick a starting point  $\theta_0$
- ▶ Approximate gradient  $\hat{L}'(\theta_0)$

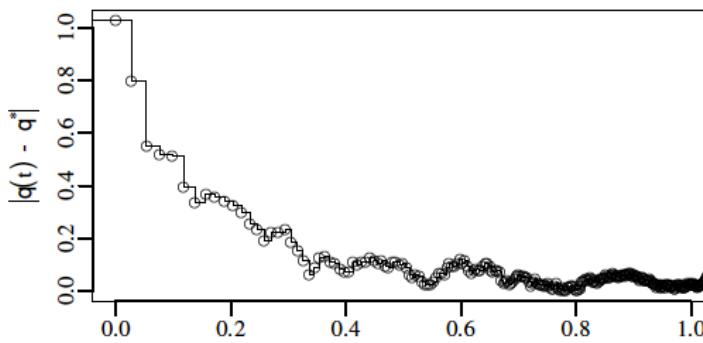
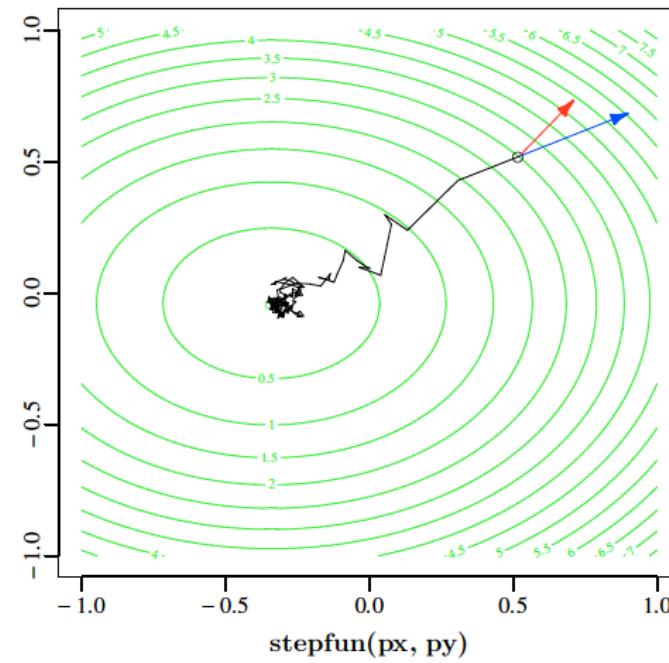


## Stochastic Gradient Descent

- ▶ Find minimum  $\theta^*$  of function  $L$
- ▶ Pick a starting point  $\theta_0$
- ▶ Approximate gradient  $\hat{L}'(\theta_0)$
- ▶ Jump “approximately” downhill
- ▶ Stochastic difference equation

$$\theta_{n+1} = \theta_n - \epsilon_n \hat{L}'(\theta_n)$$

- ▶ Under certain conditions, asymptotically approximates (continuous) gradient descent



# Why does this work?

require that the loss can be written as

$$L = \sum_{(i,j) \in Z} l(\mathbf{V}_{ij}, \mathbf{W}_{i*}, \mathbf{H}_{*j})$$

---

## Algorithm 1 SGD for Matrix Factorization

---

**Require:** A training set  $Z$ , initial values  $\mathbf{W}_0$  and  $\mathbf{H}_0$

**while** not converged **do** {step}

    Select a training point  $(i, j) \in Z$  uniformly at random.

$$\mathbf{W}'_{i*} \leftarrow \mathbf{W}_{i*} - \epsilon_n N \frac{\partial}{\partial \mathbf{W}_{i*}} l(\mathbf{V}_{ij}, \mathbf{W}_{i*}, \mathbf{H}_{*j})$$

$$\mathbf{H}_{*j} \leftarrow \mathbf{H}_{*j} - \epsilon_n N \frac{\partial}{\partial \mathbf{H}_{*j}} l(\mathbf{V}_{ij}, \mathbf{W}_{i*}, \mathbf{H}_{*j})$$

$$\mathbf{W}_{i*} \leftarrow \mathbf{W}'_{i*}$$

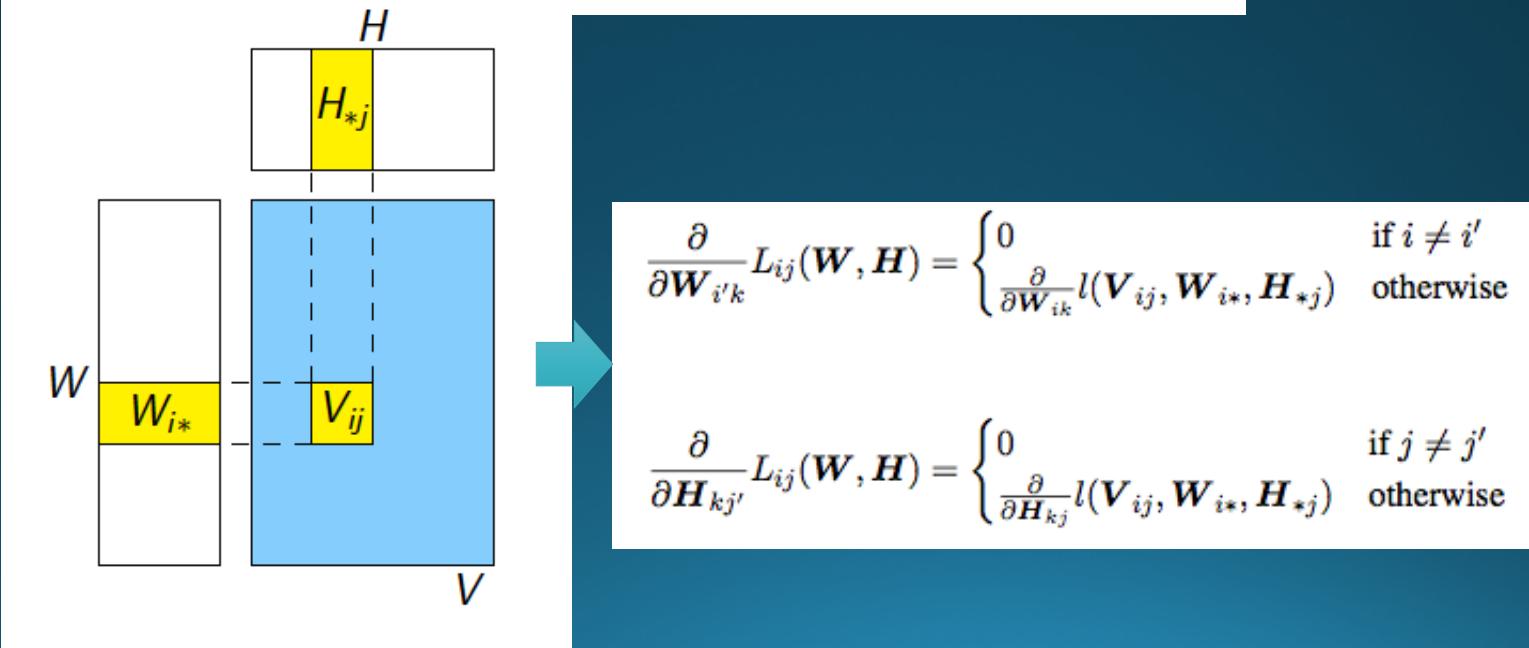
**end while**

---

# Key Claim

require that the loss can be written as

$$L = \sum_{(i,j) \in Z} l(\mathbf{V}_{ij}, \mathbf{W}_{i*}, \mathbf{H}_{*j})$$



# Checking the claim

$$\frac{\partial}{\partial \mathbf{W}_{i*}} L(\mathbf{W}, \mathbf{H}) = \frac{\partial}{\partial \mathbf{W}_{i*}} \sum_{(i',j) \in Z} L_{i'j}(\mathbf{W}_{i'*}, \mathbf{H}_{*j}) = \sum_{j \in Z_{i*}} \frac{\partial}{\partial \mathbf{W}_{i*}} L_{ij}(\mathbf{W}_{i*}, \mathbf{H}_{*j}),$$

where  $Z_{i*} = \{ j : (i, j) \in Z \}$ .

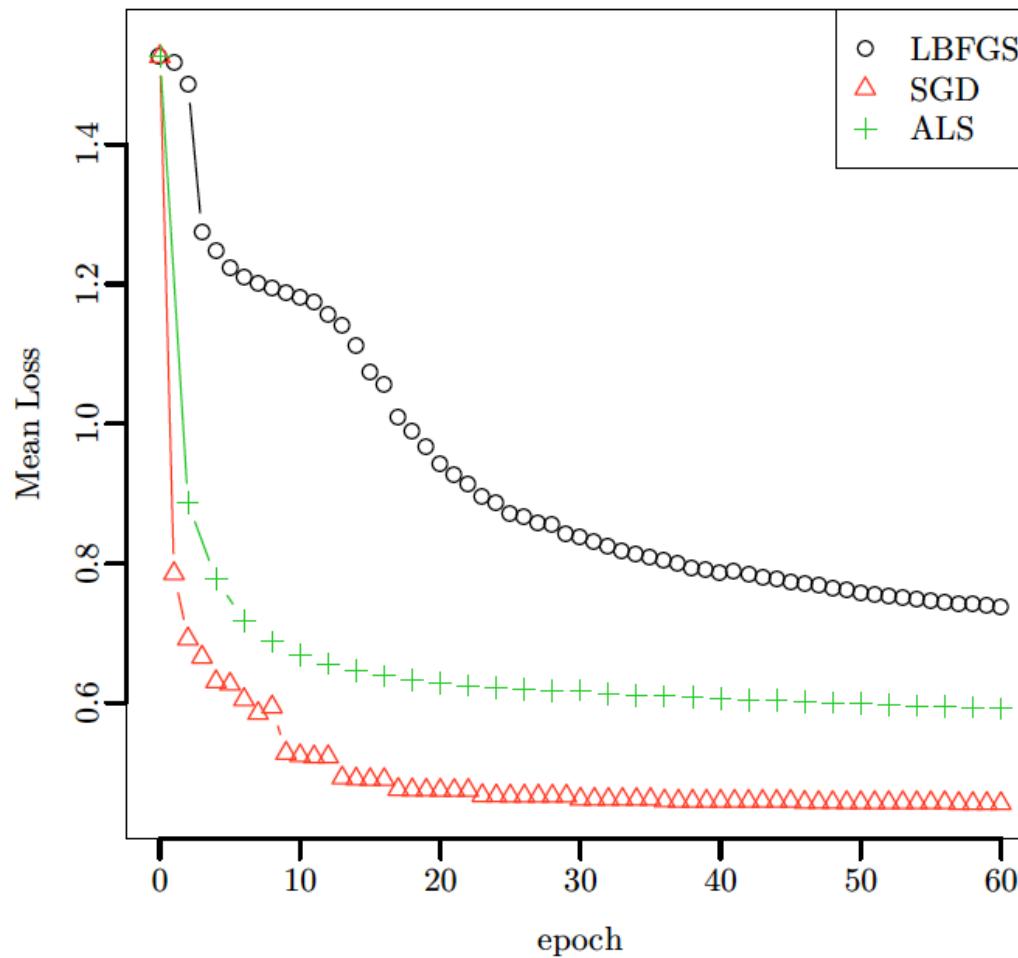
$$\frac{\partial}{\partial \mathbf{H}_{*j}} L(\mathbf{W}, \mathbf{H}) = \sum_{i \in Z_{*j}} \frac{\partial}{\partial \mathbf{W}_{*j}} L_{ij}(\mathbf{W}_{i*}, \mathbf{H}_{*j}),$$

where  $Z_{*j} = \{ i : (i, j) \in Z \}$ .

Think for SGD for logistic regression

- LR loss = compare  $y$  and  $\hat{y} = \text{dot}(w, x)$
- similar but now update  $w$  (user weights) and  $x$  (movie weight)

## Stochastic Gradient Descent on Netflix Data



# Stochastic SVD (SSVD)

- Uses **random projections** to find close approximation to SVD
- Combination of probabilistic strategies to maximize convergence likelihood
- Easily scalable to *massive* linear systems

# Basic goal

- Matrix  $A$ 
  - Find a low-rank approximation of  $A$
  - Basic dimensionality reduction

$$\|A - QQ^*A\| < \epsilon$$



# Approximating range of $A$

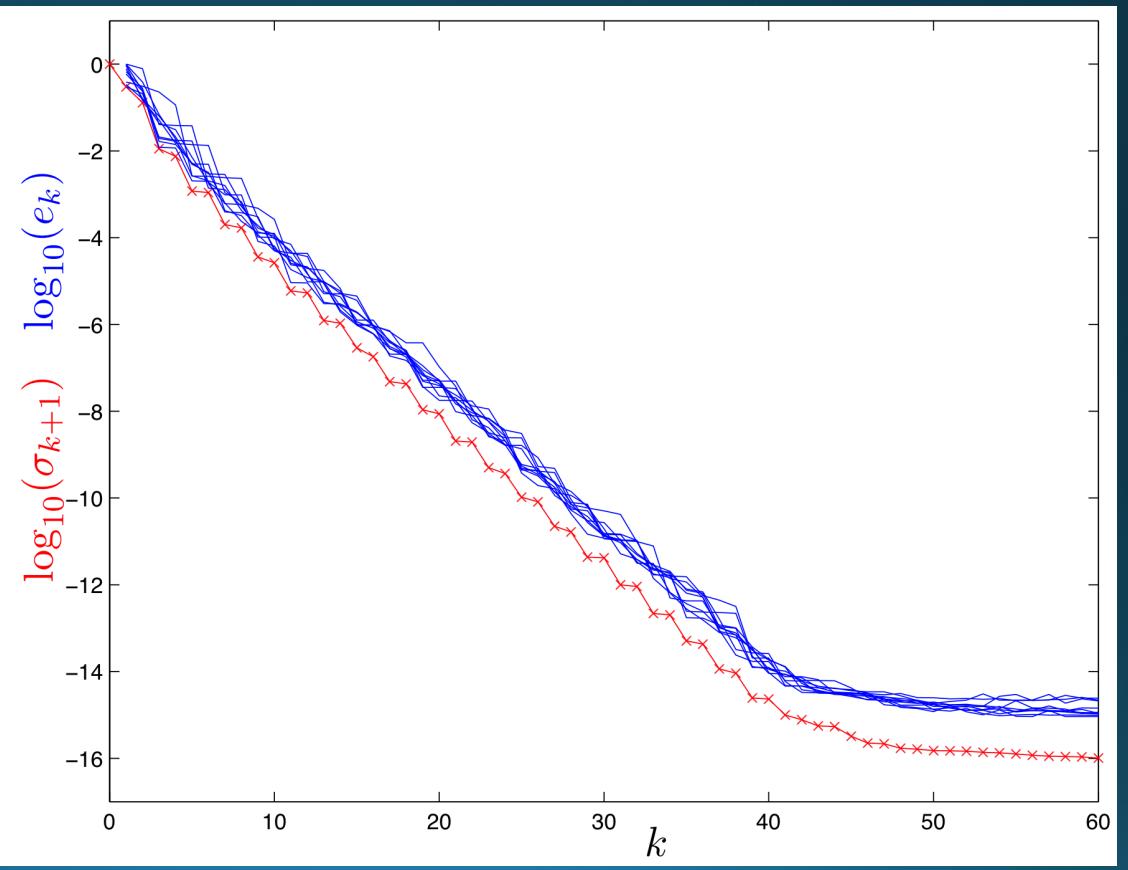
- INPUT:  $A, k, p$
  - OUTPUT:  $Q$
1. Draw Gaussian  $n \times k$  test matrix  $\Omega$
  2. Form product  $Y = A\Omega$
  3. Orthogonalize columns of  $Y \rightarrow Q$

# Approximating SVD of $A$

- INPUT:  $Q$
  - OUTPUT: Singular vectors  $U$
1. Form  $k \times n$  matrix  $B = Q^T A$
  2. Compute SVD of  $B = \hat{U} \Sigma V^T$
  3. Compute singular vectors  $U = Q \hat{U}$

# Empirical Results

- 1000x1000 matrix
- Several runs of empirical results (blue) to theoretical lower bound (red)
- **Error seems to be systemic**



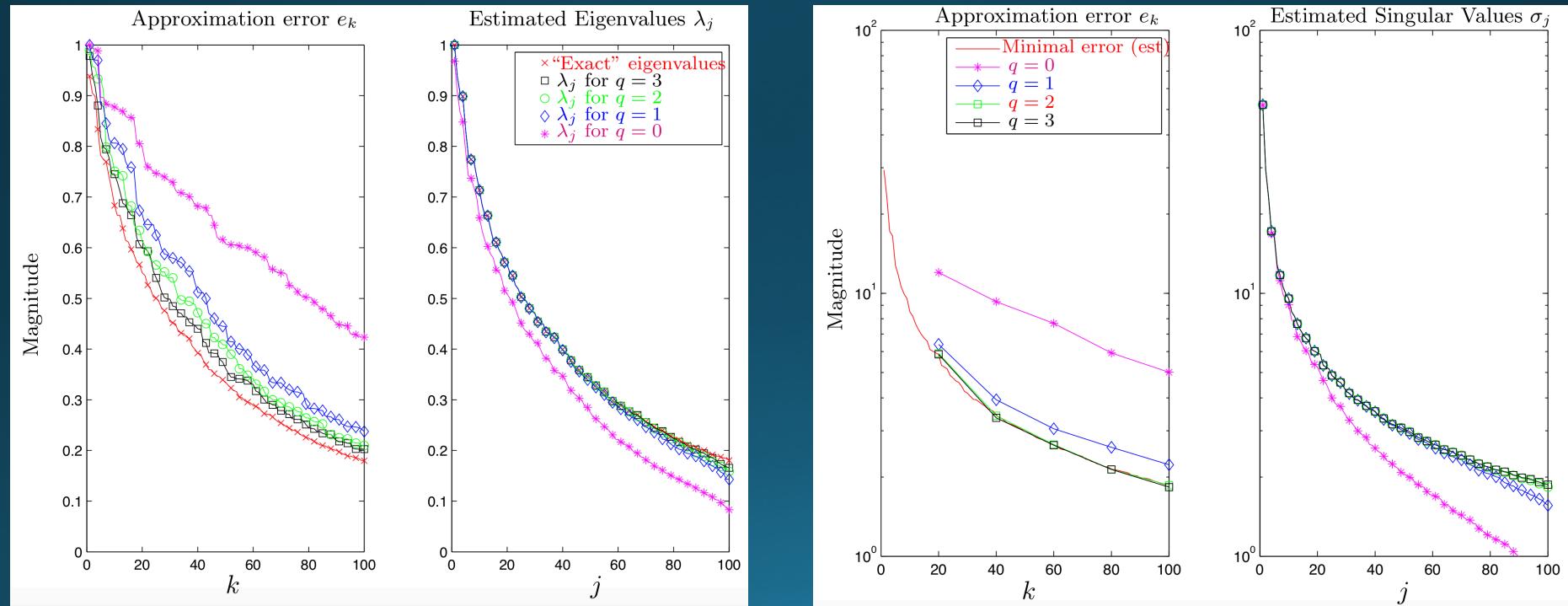
# Power iterations

- Affects decay of eigenvalues / singular values

$$Y = \cancel{A} \Omega.$$

$$Y = (A A^*)^q A \Omega$$

# Empirical Results



# Why does this work?

- Three primary reasons:

## 1. Johnson-Lindenstrauss Lemma

- Low-dimensional embeddings preserve pairwise distances

$$(1 - \varepsilon) \|u - v\|^2 \leq \|f(u) - f(v)\|^2 \leq (1 + \varepsilon) \|u - v\|^2$$

## 2. Concentration of measure

- Geometric interpretation of classical idea: regular functions of independent random variables rarely deviate far from their means

## 3. Preconditioning

- Condition number: how much change in output is produced from change in input (relation to #1)
- $Q$  matrix lowers condition number while preserving overall system

$$\kappa = \frac{|\lambda_{\max}|}{|\lambda_{\min}|}$$

# Summary

- Relationship of SVD and PCA
  - PCA: eigenvectors and eigenvalues of the covariance matrix (or kernel matrix, for Kernel PCA)
  - SVD: Low-rank approximation for *any* matrix
- CUR
  - Randomly sample columns of data matrix  $A$  to use as basis
  - Interpretable and sparse, but potentially oversample high-magnitude columns
- SVD via SGD
  - Reframe SVD as a matrix completion problem
  - Use SGD in alternating least-squares to infer “missing” components
- SSVD
  - Full-blown Johnson-Lindenstrauss exploitation
  - Use random projections to approximate SVD to high accuracy
  - Requires some empirical tweaks (oversampling, power iterations)

# References

- “Randomized methods for computing low-rank approximations of matrices”,  
[https://amath.colorado.edu/faculty/martinss/Pubs/2012\\_halko\\_dissertation.pdf](https://amath.colorado.edu/faculty/martinss/Pubs/2012_halko_dissertation.pdf)
- “Large-scale matrix factorization with distributed stochastic gradient descent”,  
<https://pdfs.semanticscholar.org/b996/0e7cob1acbd0711b9a6ba32770obe64abeb5.pdf>
- “CUR decomposition for compression and compressed sensing of large-scale traffic data”, <https://dspace.mit.edu/openaccess-disseminate/1721.1/86879>