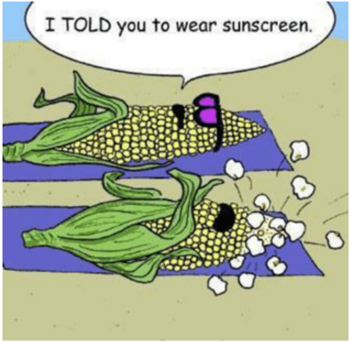
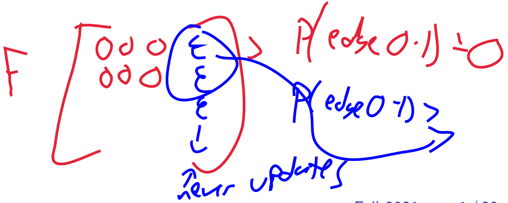
CSCI 4022 Fall 2021
UV Decomposition



An a-maize-ing kernel of truth when sunbathing: losing face really shucks and there is little margarine for error.

Hw 7:

- office hrs after class
- #3: use background probability



Singular Value Decomposition (SVD)

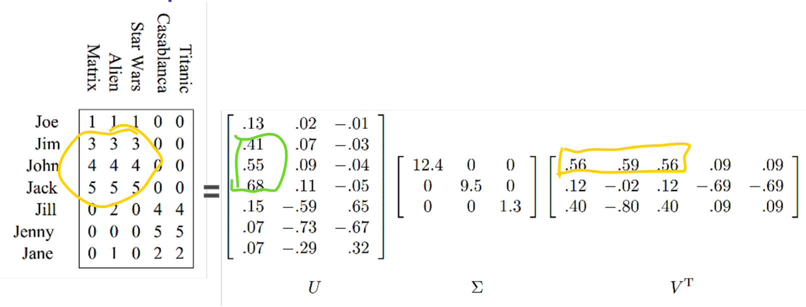
Big idea: Suppose you have an $m \times n$ matrix A, with $\underline{rank(A) = r}$, such as:

- m documents, n terms to compute similarities, make recommendations, do science
- m users, n movies
- Generally: m data points, n entries per data point

Like PCA, we want to decompose data into the most important axes, or concepts...
The Decomposition: $\underline{A = U\Sigma V^T}$ with sizes: $A_{m\times n} = U_{m\times r}\Sigma_{r\times r}V^T_{n\times r}$.
Where each piece is:

1. $U_{m\times r}$ left singular vectors (orthonormal); e.g, m documents, r concepts; orthonormal
2. $\Sigma_{r\times r}$ diagonal matrix of singular values, the strength of each concept, in decreasing order
3. $V_{n\times r}$ right singular vectors (orthonormal); n terms, r concepts; one vector per concept

SVD Example, Rank 3



$$\begin{array}{c|ccccc} & \text{Matrix} & \text{Alien} & \text{Star Wars} & \text{Casablanca} & \text{Titanic} \\ \hline \text{Joe} & 1 & 1 & 1 & 0 & 0 \\ \text{Jim} & 3 & 3 & 3 & 0 & 0 \\ \text{John} & 4 & 4 & 4 & 0 & 0 \\ \text{Jack} & 5 & 5 & 5 & 0 & 0 \\ \text{Jill} & 0 & 2 & 0 & 4 & 4 \\ \text{Jenny} & 0 & 0 & 0 & 5 & 5 \\ \text{Jane} & 0 & 1 & 0 & 2 & 2 \end{array} = \underbrace{\begin{bmatrix} .13 & .02 & -.01 \\ .41 & .07 & -.03 \\ .55 & .09 & -.04 \\ .68 & .11 & -.05 \\ .15 & -.59 & .65 \\ .07 & -.73 & -.67 \\ .07 & -.29 & .32 \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} 12.4 & 0 & 0 \\ 0 & 9.5 & 0 \\ 0 & 0 & 1.3 \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} .56 & .59 & .56 & .09 & .09 \\ .12 & -.02 & .12 & -.69 & -.69 \\ .40 & -.80 & .40 & .09 & .09 \end{bmatrix}}_{V^{\mathrm{T}}}$$

An SVD yields r *singular values* σ_k which represent the relative **strength of concepts**.

1. $U_{m\times r}$ maps users to concepts
2. $\Sigma_{r\times r}$ shows strength/importance of concepts
3. $V_{n\times r}$ maps movies to concepts

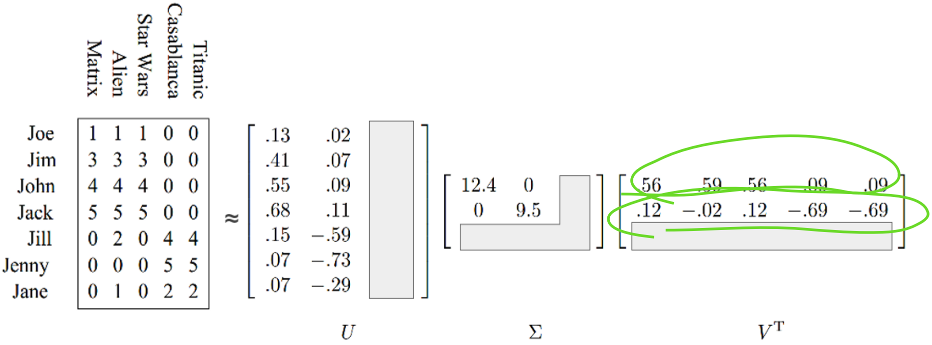
sci-fi

Intuition: what are the “concepts” in our toy example problem?

Dimension Reduction with SVD

How do we actually *reduce* dimension?

1. **Answer:** we *discard* smaller singular values by setting them equal to zero.
2. **Result:** corresponding *columns* of U and *rows* of V^T are no longer used
3. Our modified system is a **low-rank** approximation of M. $M' = U\Sigma V^T \approx M$.



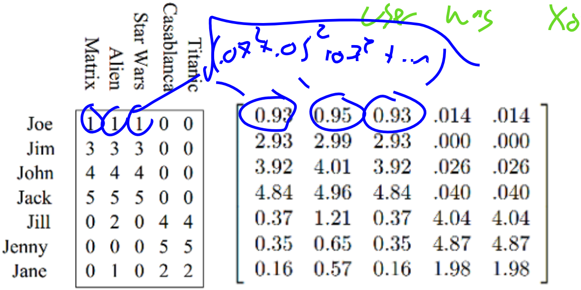
$$\begin{array}{c|ccccc|} & \text{Matrix} & \text{Alien} & \text{Star Wars} & \text{Casablanca} & \text{Titanic} \\ \hline \text{Joe} & 1 & 1 & 1 & 0 & 0 \\ \text{Jim} & 3 & 3 & 3 & 0 & 0 \\ \text{John} & 4 & 4 & 4 & 0 & 0 \\ \text{Jack} & 5 & 5 & 5 & 0 & 0 \\ \text{Jill} & 0 & 2 & 0 & 4 & 4 \\ \text{Jenny} & 0 & 0 & 0 & 5 & 5 \\ \text{Jane} & 0 & 1 & 0 & 2 & 2 \\ \hline \end{array} \approx \underbrace{\begin{bmatrix} .13 & .02 \\ .41 & .07 \\ .55 & .09 \\ .68 & .11 \\ .15 & -.59 \\ .07 & -.73 \\ .07 & -.29 \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} 12.4 & 0 \\ 0 & 9.5 \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} .56 & .59 & .56 & .09 & .09 \\ .12 & -.02 & .12 & -.69 & -.69 \end{bmatrix}}_{V^T}$$

SVD Dimension Reduction

How similar is our approximation (M') to the original matrix (M)?

- Check using the Frobenius norm: $||A||_F = \sqrt{\sum_{i,j} A_{i,j}^2}$
- The actual difference between M and M' is $||M - M'||_F = \sqrt{\sum_{i,j} \left(M_{i,j} - M'_{i,j}\right)^2}$

$\sqrt{.07^2 + .05^2 + .07^2 + \dots}$



$$\begin{array}{r|ccccc|}
 & \text{Matrix} & \text{Alien} & \text{Star Wars} & \text{Casablanca} & \text{Titanic} \\
\text{Joe} & 1 & 1 & 1 & 0 & 0 \\
\text{Jim} & 3 & 3 & 3 & 0 & 0 \\
\text{John} & 4 & 4 & 4 & 0 & 0 \\
\text{Jack} & 5 & 5 & 5 & 0 & 0 \\
\text{Jill} & 0 & 2 & 0 & 4 & 4 \\
\text{Jenny} & 0 & 0 & 0 & 5 & 5 \\
\text{Jane} & 0 & 1 & 0 & 2 & 2
\end{array}
\quad
\begin{bmatrix}
0.93 & 0.95 & 0.93 & .014 & .014 \\
2.93 & 2.99 & 2.93 & .000 & .000 \\
3.92 & 4.01 & 3.92 & .026 & .026 \\
4.84 & 4.96 & 4.84 & .040 & .040 \\
0.37 & 1.21 & 0.37 & 4.04 & 4.04 \\
0.35 & 0.65 & 0.35 & 4.87 & 4.87 \\
0.16 & 0.57 & 0.16 & 1.98 & 1.98
\end{bmatrix}$$

user was $X_0 \cdot V_0 + X_1 \cdot V_1$

$\sim$ close to 11100

$\sim$ close to 00011

SVD Example

How do we decide how many σ 's to keep?

- Define the *energy* as $\Sigma_i \sigma_i^2$
- General rule: keep enough singular values to account for 80-90% of the energy

$\boxed{144 + 160} + 1$ ~ worth $\frac{1}{245}$

keep these, retain $\frac{244}{245}$ 'of the energy'

$$
\begin{array}{r|ccccc|}
 & \text{Matrix} & \text{Alien} & \text{Star Wars} & \text{Casablanca} & \text{Titanic} \\
\text{Joe} & 1 & 1 & 1 & 0 & 0 \\
\text{Jim} & 3 & 3 & 3 & 0 & 0 \\
\text{John} & 4 & 4 & 4 & 0 & 0 \\
\text{Jack} & 5 & 5 & 5 & 0 & 0 \\
\text{Jill} & 0 & 2 & 0 & 4 & 4 \\
\text{Jenny} & 0 & 0 & 0 & 5 & 5 \\
\text{Jane} & 0 & 1 & 0 & 2 & 2
\end{array}
=
\underbrace{\begin{bmatrix}
.13 & .02 & -.01 \\
.41 & .07 & -.03 \\
.55 & .09 & -.04 \\
.68 & .11 & -.05 \\
.15 & -.59 & .65 \\
.07 & -.73 & -.67 \\
.07 & -.29 & .32
\end{bmatrix}}_{U}
\underbrace{\begin{bmatrix}
12.4 & 0 & 0 \\
0 & 9.5 & 0 \\
0 & 0 & 1.3
\end{bmatrix}}_{\Sigma}
\underbrace{\begin{bmatrix}
.56 & .59 & .56 & .09 & .09 \\
.12 & -.02 & .12 & -.69 & -.69 \\
.40 & -.80 & .40 & .09 & .09
\end{bmatrix}}_{V^{\mathrm{T}}}
$$

SVD utility: concept-space

Suppose we have the previous simpler example below, and a new user, Megan, comes along. She has only watched The Matrix, and rated it a 4.

So, Megan's movie vector is $m = [4, 0, 0, 0, 0]$ We can map Megan into concept-space by multiplying mV (V maps movies → concepts). We find that $mv =$

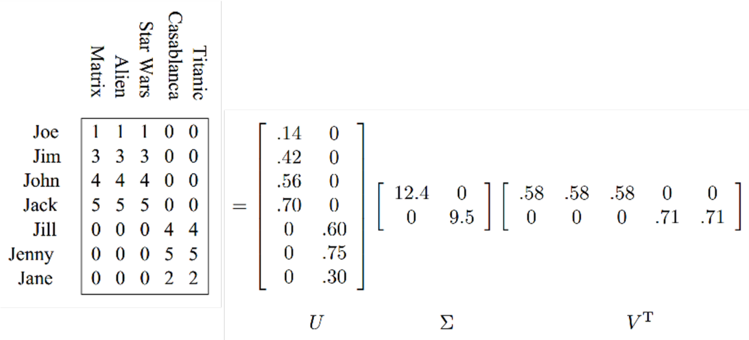
$$\begin{bmatrix} 4 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} .58 & 0 \\ .58 & 0 \\ .58 & 0 \\ 0 & \cancel{.71} \\ 0 & \cancel{.71} \end{bmatrix} = \begin{bmatrix} 2.32 & 0 \end{bmatrix}$$

, so we can infer Megan might be interested in sci-fi movies.

$$\begin{array}{r|ccccc|} & \text{Matrix} & \text{Alien} & \text{Star Wars} & \text{Casablanca} & \text{Titanic} \\ \hline \text{Joe} & 1 & 1 & 1 & 0 & 0 \\ \text{Jim} & 3 & 3 & 3 & 0 & 0 \\ \text{John} & 4 & 4 & 4 & 0 & 0 \\ \text{Jack} & 5 & 5 & 5 & 0 & 0 \\ \text{Jill} & 0 & 0 & 0 & 4 & 4 \\ \text{Jenny} & 0 & 0 & 0 & 5 & 5 \\ \text{Jane} & 0 & 0 & 0 & 2 & 2 \\ \hline \end{array} = \begin{bmatrix} .14 & 0 \\ .42 & 0 \\ .56 & 0 \\ .70 & 0 \\ 0 & .60 \\ 0 & .75 \\ 0 & .30 \end{bmatrix} \begin{bmatrix} 12.4 & 0 \\ 0 & 9.5 \end{bmatrix} \begin{bmatrix} .58 & .58 & .58 & 0 & 0 \\ 0 & 0 & 0 & .71 & .71 \end{bmatrix}$$

SVD utility: concept-space

But we can map Megan *back* into **movie-space**, too! Here, we'd be multiplying mV by V^T, which is the reverse of V (V^T maps movies ← concepts). We find that $(mV)V^T =$ $\begin{bmatrix}\boxed{2.32} & 0\end{bmatrix}\begin{bmatrix} .58 & .58 & .58 & 0 & 0 \\ 0 & 0 & 0 & .71 & .71\end{bmatrix} = \begin{bmatrix}\underline{1.35 \quad 1.35 \quad 1.35} & 0 & 0\end{bmatrix}$, so we can infer Megan might be interested Alien or Star Wars as much as she liked the Matrix.



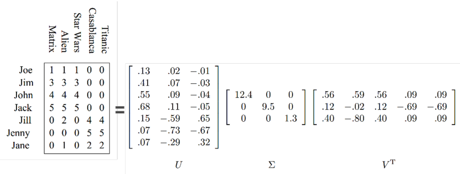
$$\begin{array}{r|ccccc|} & \text{Matrix} & \text{Alien} & \text{Star Wars} & \text{Casablanca} & \text{Titanic} \\ \hline \text{Joe} & 1 & 1 & 1 & 0 & 0 \\ \text{Jim} & 3 & 3 & 3 & 0 & 0 \\ \text{John} & 4 & 4 & 4 & 0 & 0 \\ \text{Jack} & 5 & 5 & 5 & 0 & 0 \\ \text{Jill} & 0 & 0 & 0 & 4 & 4 \\ \text{Jenny} & 0 & 0 & 0 & 5 & 5 \\ \text{Jane} & 0 & 0 & 0 & 2 & 2 \\ \hline \end{array} = \underbrace{\begin{bmatrix} .14 & 0 \\ .42 & 0 \\ .56 & 0 \\ .70 & 0 \\ 0 & .60 \\ 0 & .75 \\ 0 & .30 \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} 12.4 & 0 \\ 0 & 9.5 \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} .58 & .58 & .58 & 0 & 0 \\ 0 & 0 & 0 & .71 & .71 \end{bmatrix}}_{V^{\mathrm{T}}}$$

SVD utility: concept-space

We can also map all users into concept-space and make recommendations by comparing their similarities there. (e.g., using cosine similarity)We can also map all users into concept-space and make recommendations by comparing their similarities there. (e.g., using cosine similarity)
Example: from the slightly more interesting example earlier:

Example: from the slightly more interesting example earlier:

$$MV = \begin{bmatrix} 1.71 & 0.22 & 0 \\ 5.13 & 0.66 & 0 \\ 6.84 & 0.88 & 0 \\ \boxed{8.55} & 1.1 & 0 \\ 1.9 & -5.56 & -0.88 \\ 0.9 & -6.9 & 0.9 \\ 0.95 & -2.78 & -0.44 \end{bmatrix}$$



$$\begin{array}{r|ccccc|} & \text{Matrix} & \text{Alien} & \text{Star Wars} & \text{Casablanca} & \text{Titanic} \\ \hline \text{Joe} & 1 & 1 & 1 & 0 & 0 \\ \text{Jim} & 3 & 3 & 3 & 0 & 0 \\ \text{John} & 4 & 4 & 4 & 0 & 0 \\ \text{Jack} & 5 & 5 & 5 & 0 & 0 \\ \text{Jill} & 0 & 2 & 0 & 4 & 4 \\ \text{Jenny} & 0 & 0 & 0 & 5 & 5 \\ \text{Jane} & 0 & 1 & 0 & 2 & 2 \\ \hline \end{array} = \underset{U}{\begin{bmatrix} .13 & .02 & -.01 \\ .41 & .07 & -.03 \\ .55 & .09 & -.04 \\ .68 & .11 & -.05 \\ .15 & -.59 & .65 \\ .07 & -.73 & -.67 \\ .07 & -.29 & .32 \end{bmatrix}} \underset{\Sigma}{\begin{bmatrix} 12.4 & 0 & 0 \\ 0 & 9.5 & 0 \\ 0 & 0 & 1.3 \end{bmatrix}} \underset{V^{T}}{\begin{bmatrix} .56 & .59 & .56 & .09 & .09 \\ .12 & -.02 & .12 & -.69 & -.69 \\ .40 & -.80 & .40 & .09 & .09 \end{bmatrix}}$$

Computing SVD

So how do we actually *do* this?

SVD: $U\Sigma V^T$

Where Σ is a diagonal matrix full of A's singular values σ_i

Recall: Eigenvalue decomposition: for a symmetric matrix A, $\boxed{A = X\Lambda X^T}$

Where Λ is a diagonal matrix full of A's eigenvalues λ_i

Computing SVD

So how do we actually *do* this?

SVD: $U\Sigma V^T$

Where Σ is a diagonal matrix full of A's singular values σ_i

Recall: Eigenvalue decomposition: for a symmetric matrix A, $A = X\Lambda X^T$

Where Λ is a diagonal matrix full of A's eigenvalues λ_i

In SVD, **both** U, V are orthonormal, like X

Question: If we write A using its SVD, what is AA^T? And what is A^TA?

$$AA^T = (U\Sigma V^T)(U\Sigma V^T)^T = (U\Sigma \underbrace{V^T)(V}_{V^TV = I}{}^{\cancel{T}}\Sigma^T U^T)$$

$$= U\Sigma \cdot \Sigma^T U^T = U\Sigma^2 U^T$$

Computing SVD

So how do we actually *do* this?

SVD: $U\Sigma V^T$

Where Σ is a diagonal matrix full of A's singular values σ_i

Recall: Eigenvalue decomposition: for a symmetric matrix A, $A = X\Lambda X^T$

Where Λ is a diagonal matrix full of A's eigenvalues λ_i

In SVD, **both** U, V are orthonormal, like X

Question: If we write A using its SVD, what is AA^T? And what is A^TA?

- $AA^T = U\Sigma V^T(U\Sigma V^T)^T = U\Sigma V^T V\Sigma^T U^T = U\Sigma\Sigma^T U^T$
- $A^TA = \underline{V\Sigma^2}V^T$, similarly
- **Result:** the singular values *squared* Σ^2 are the eigenvalues of both AA^T and A^TA.

Computing SVD

Result: the singular values *squared* Σ^2 are the eigenvalues of both AA^T and A^TA. And the matrices U and V?

We have: $AA^T = U\Sigma^T 2U^T$ and $A^TA = V\Sigma^2 TV^T$

- First, Multiply $\boxed{AA^T}$ on the right by U:

$x = U$ $\lambda = \Sigma$

$AA^T x = \lambda x$.

$$AA^TU = U\Sigma^2U^TU = U\overset{\text{is } \lambda}{\Sigma^2}$$

... or U's columns are *eigenvectors* of AA^T

- Similarly, multiply A^TA on the right by V:

$$A^TAV = V\Sigma^2V^TV = V\Sigma^2$$

... or V's columns are eigenvectors of A^TA

SVD Algorithm

To compute the SVD of a matrix A:

1. Construct AA^T and A^TA
2. Use the generalized power iteration algorithm to compute the eigenvalues and eigenvectors of each (or something faster, maybe trust in your software? Or take an advanced numerics class!)
   (Eigenvalues should be the same! But numerical imprecision.... maybe not :)
3. $\Sigma=$ diag(square roots of the eigenvalues of AA^T or A^TA, in descending order)
4. $U =$ matrix whose columns are the eigenvectors of AA^T (in descending order by their associated eigenvalue)
5. $V=$ matrix whose columns are the eigenvectors of ATA (in descending order by their associated eigenvalue)

1. Construct AA^T and A^TA
2. Use the generalized power iteration algorithm to compute the eigenvalues and eigenvectors of each (or something faster, maybe trust in your software? Or take an advanced numerics class!)
   (Eigenvalues should be the same! But numerical imprecision.... maybe not :)
3. $\Sigma=$ diag(square roots of the eigenvalues of AA^T or A^TA, in descending order)
4. $U =$ matrix whose columns are the eigenvectors of AA^T (in descending order by their associated eigenvalue)
5. $V=$ matrix whose columns are the eigenvectors of ATA (in descending order by their associated eigenvalue)

1. Construct AA^T and A^TA
2. Use the generalized power iteration algorithm to compute the eigenvalues and eigenvectors of each (or something faster, maybe trust in your software? Or take an advanced numerics class!)
   (Eigenvalues should be the same! But numerical imprecision.... maybe not :)
3. $\Sigma =$ diag(square roots of the eigenvalues of AA^T or A^TA, in descending order)
4. $U =$ matrix whose columns are the eigenvectors of AA^T (in descending order by their associated eigenvalue)
5. $V =$ matrix whose columns are the eigenvectors of ATA (in descending order by their associated eigenvalue)
   To reduce dimension:
6. Keep a running total of the sum of squared singular values (as an array, will naturally be in descending order).
7. Once you have all of them, divide the whole array by the total, see where $\approx 80 - 90\%$ is; zero out all the singular values after this point!

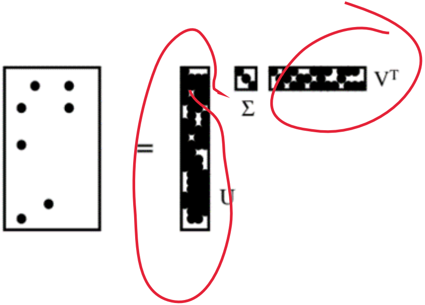
SVD: Pros and Cons

Pros:

1. Optimal low-rank approximation
2. Always exists, for any matrix

Cons:

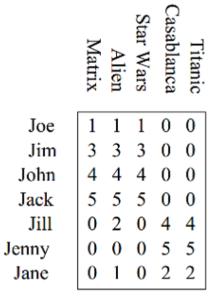
1. Interpretability problem
2. A singular vector specifies a linear combination of all input columns/rows – what does this mean?
3. Lack of sparsity
4. Our original matrix may have been sparse (users/movies/ratings, e.g., certainly is!), but U and V have lots of nonzero entries



The Missing Data Problem

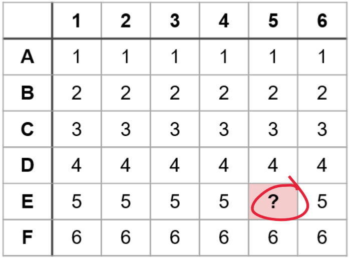
There's an issue with our treatment of this matrix: as in recommendation systems, we often want to treat "0" entries as *unknown* rather than part of the inherent/crucial structure of the matrix.

Example: What would Joe's rating for Casablanca be?



	Matrix	Alien	Star Wars	Casablanca	Titanic
Joe	1	1	1	0	0
Jim	3	3	3	0	0
John	4	4	4	0	0
Jack	5	5	5	0	0
Jill	0	2	0	4	4
Jenny	0	0	0	5	5
Jane	0	1	0	2	2

Example: what is the missing value?

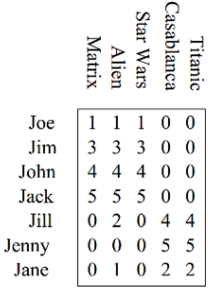


	1	2	3	4	5	6
A	1	1	1	1	1	1
B	2	2	2	2	2	2
C	3	3	3	3	3	3
D	4	4	4	4	4	4
E	5	5	5	5	?	5
F	6	6	6	6	6	6

The Missing Data Problem

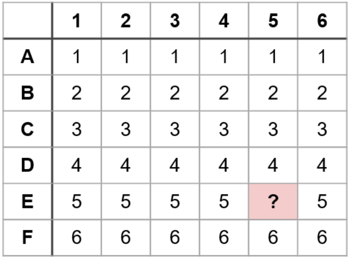
There's an issue with our treatment of this matrix: as in recommendation systems, we often want to treat "0" entries as *unknown* rather than part of the inherent/crucial structure of the matrix.

Example: What would Joe's rating for Casablanca be?



	Matrix	Alien	Star Wars	Casablanca	Titanic
Joe	1	1	1	0	0
Jim	3	3	3	0	0
John	4	4	4	0	0
Jack	5	5	5	0	0
Jill	0	2	0	4	4
Jenny	0	0	0	5	5
Jane	0	1	0	2	2

Example: what is the missing value?



	1	2	3	4	5	6
A	1	1	1	1	1	1
B	2	2	2	2	2	2
C	3	3	3	3	3	3
D	4	4	4	4	4	4
E	5	5	5	5	**?**	5
F	6	6	6	6	6	6

I like 5! But how can we justify that? And how can we automate that calculation?

The Missing Data Decomposition

SVD and PCA: We approximate the $m \times n$ utility matrix M by a low-rank/lower-dimensional representation:

$$M = U\Sigma V^T; \quad \text{take only k largest singular values}$$

What if we tried to make it even simpler?

Can we find only *two* matrices U and V such that $M \approx UV$? We would needs U to be $m \times d$ and V to be $d \times n$ for *some* d... smaller sounds nicer!

$[6\times 1] \times [1\times 6] =$

The Missing Data Decomposition

SVD and PCA: We approximate the $m \times n$ utility matrix M by a low-rank/lower-dimensional representation:

$$M = U\Sigma V^T; \quad \text{take only k largest singular values}$$

What if we tried to make it even simpler?

Can we find only *two* matrices U and V such that $M \approx UV$? We would needs U to be $m \times d$ and V to be $d \times n$ for *some* d... smaller sounds nicer!

UV Decomposition: *Find* entries of U and V so that $M' = UV$ is close to M.

V is 2x5

$$\underbrace{\begin{bmatrix} 5 & 2 & 4 & 4 & 3 \\ 3 & 1 & 2 & 4 & 1 \\ 2 & & 3 & 1 & 4 \\ 2 & 5 & 4 & 3 & 5 \\ 4 & 4 & 5 & 4 & \end{bmatrix}}_{\text{GOAL}} \heartsuit = \underbrace{\begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \\ u_{31} & u_{32} \\ u_{41} & u_{42} \\ u_{51} & u_{52} \end{bmatrix}}_{\text{U is 5x2}} \times \begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} & v_{15} \\ v_{21} & v_{22} & v_{23} & v_{24} & v_{25} \end{bmatrix}$$

The Missing Data Decomposition

UV Decomposition: *Find* entries of U and V so that $M' = UV$ is close to M.

$$\begin{bmatrix} 5 & 2 & 4 & 4 & 3 \\ 3 & 1 & 2 & 4 & 1 \\ 2 & & 3 & 1 & 4 \\ 2 & 5 & 4 & 3 & 5 \\ 4 & 4 & 5 & 4 & \end{bmatrix} \overset{\heartsuit}{=} \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \\ u_{31} & u_{32} \\ u_{41} & u_{42} \\ u_{51} & u_{52} \end{bmatrix} \times \begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} & v_{15} \\ v_{21} & v_{22} & v_{23} & v_{24} & v_{25} \end{bmatrix}$$

What do we mean by "close to?" As usual, we need a *distance* or *error* measure. Options include Frobenius norm, maximum absolute difference of any entry, etc.

RMSE: root-mean-squared-error is given by

$$RMSE = \sqrt{\frac{1}{N} \sum_{i,j} \left(M_{i,j} - M'_{i,j} \right)^2}$$

where the sum extends over only the *non-blank* entries of M, and N is the total number of such entries.

The Missing Data Decomposition

UV Decomposition: *Find* entries of U and V so that $M' = UV$ is close to M.

$$\begin{bmatrix} 5 & 2 & 4 & 4 & 3 \\ 3 & 1 & 2 & 4 & 1 \\ 2 & & 3 & 1 & 4 \\ 2 & 5 & 4 & 3 & 5 \\ 4 & 4 & 5 & 4 & \end{bmatrix} \overset{\heartsuit}{=} \underbrace{\begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \\ u_{31} & u_{32} \\ u_{41} & u_{42} \\ u_{51} & u_{52} \end{bmatrix}}_{d} \times \left.\begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} & v_{15} \\ v_{21} & v_{22} & v_{23} & v_{24} & v_{25} \end{bmatrix}\right\}$$

Plan: We'll develop an iterative solution to optimize the entries of U and V

- Need some starting values
- Could pick all 1s, random numbers, all equal to $(1/d) \cdot mean$ of M's non-blank entries, etc.

Whatever we do, there are *lots* of degrees of freedom in our selection of the u_{ij} and v_{ij}

- Many local minima in the RMSE surface
- Want to use a handful of different starting values for U and V and compare

UV Decomposition: Idea

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} \boxed{2} & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & \boxed{2} & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix} \text{ vs } \begin{bmatrix} 5 & \ddots & & \\ 3 & & & \\ 2 & & | & \\ 2 & & & \\ 4 & & \ddots & \end{bmatrix}$$

Suppose we start with some non-random initialization of U, V all-1s.

- Then our first estimate of M would be M' consisting of all 2s, which... isn't very good.
- We want to **optimize** over the values in U and V

UV Decomposition: Idea

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} x\cdot 1+1\cdot 1 & x\cdot 1+1\cdot 1 & x+1 & x+1 & x+1 \\ & & & & \\ & & & & \end{bmatrix} \approx \begin{bmatrix} 5 & 2 & 4 & 4 & 3 \\ & & & & \\ & & & & \end{bmatrix}$$

doesn't change

Suppose we start with some non-random initialization of U, V all-1s.

- Then our first estimate of M would be M' consisting of all 2s, which... isn't very good.
- We want to **optimize** over the values in U and V
- Let's *improve* on U by making the first entry - u_{11} - better. Call this value x

UV Decomposition: Idea

$$\begin{bmatrix} x & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} x+1 & x+1 & x+1 & x+1 & x+1 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

Suppose we start with some non-random initialization of U, V all-1s.

- Then our first estimate of M would be M' consisting of all 2s, which... isn't very good.
- We want to **optimize** over the values in U and V
- Let's *improve* on U by making the first entry - u_{11} - better. Call this value x
- What's the best x? The one that provides the best M'!
- **Choose** x that minimizes RMSE. But **note:** x *only affects* the top row of M'.

UV

$U_{1,1} \cdot V_{1,3} + U_{1,2} \cdot V_{2,3}$

UV Decomposition: Optimization

$$\begin{bmatrix} x & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} x+1 & x+1 & x+1 & x+1 & x+1 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

$\{5\}\ \{2, \{4\ 4\}\}$

u5

Choose x that minimizes RMSE. But **note:** x *only affects* the top row of M'
...Reminder that $M_{1,:}$ =[5 2 4 4 3]

... and the best choice is a classic calculus problem: find the deriv, set equal to zero!

$M_{\text{row } 1, \text{col } 3}$

error: $((x+1) - 5)^2 + ((x+1) - 2)^2 + ((x+1) - 4)^2 + ((x+1) - 4)^2 + ((x+1) - 3)^2$

UV Decomposition: Optimization

$$\begin{bmatrix} x & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} x+1 & x+1 & x+1 & x+1 & x+1 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

Choose x that minimizes RMSE. But **note:** x *only affects* the top row of M'
...Reminder that $M_{1,:}$ =[5 2 4 4 3]

$$\begin{aligned} SS &= (5-(x+1))^2 + (2-(x+1))^2 + (4-(x+1))^2 + (4-(x+1))^2 + (3-(x+1))^2 \\ &= (4-x)^2 + (1-x)^2 + (3-x)^2 + (3-x)^2 + (2-x)^2 \end{aligned}$$

... and the best choice is a classic calculus problem: find the deriv, set equal to zero!

UV Decomposition: Optimization

$$\begin{bmatrix} x & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} x+1 & x+1 & x+1 & x+1 & x+1 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

Choose x that minimizes RMSE. But **note:** x *only affects* the top row of M'

...Reminder that $M_{1,:}$ =[5 2 4 4 3]

$$\begin{aligned} SS &= (5-(x+1))^2 + (2-(x+1))^2 + (4-(x+1))^2 + (4-(x+1))^2 + (3-(x+1))^2 \\ &= (4-x)^2 + (1-x)^2 + (3-x)^2 + (3-x)^2 + (2-x)^2 \end{aligned}$$

... and the best choice is a classic calculus problem: find the deriv, set equal to zero!

$$\frac{d}{dx}SS = -2[(4-x)+(1-x)+(3-x)+(3-x)+(2-x)]$$

$$0 \overset{set}{=} -2[(13-5x)] \implies x = 13/5 = 2.6.$$

UV Decomposition: Optimization

So we update U, and then also update M'.

$$\begin{bmatrix} 2.6 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 3.6 & 3.6 & 3.6 & 3.6 & 3.6 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 2.6y+1 \\ y+1 \\ y+1 \\ y+1 \\ y+1 \end{bmatrix}$$

.. and we move on with our optimization. We choose another entry of either u or v. Suppose we move onto the first entry in V, $\underline{v_{11}}$. Call it y.

UV Decomposition: Optimization

So we update U, and then also update M'.

$$\begin{bmatrix} 2.6 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} y & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} =$$

.. and we move on with our optimization. We choose another entry of either u or v. Suppose we move onto the first entry in V, v_{11}. Call it y.

- What's the best y? The one that provides the best M'!
- **Choose** y that minimizes RMSE. But **note:** y *only affects* the first column of M'.

UV Decomposition: Optimization

So we update U, and then also update M'.

$$\begin{bmatrix} 2.6 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} y & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 2.6y+1 & 3.6 & 3.6 & 3.6 & 3.6 \\ y+1 & 2 & 2 & 2 & 2 \\ y+1 & 2 & 2 & 2 & 2 \\ y+1 & 2 & 2 & 2 & 2 \\ y+1 & 2 & 2 & 2 & 2 \end{bmatrix}$$

.. and we move on with our optimization. We choose another entry of either u or v. Suppose we move onto the first entry in V, v_{11}. Call it y.

- What's the best y? The one that provides the best M'!
- **Choose** y that minimizes RMSE. But **note:** y *only affects* the first column of M'.
- ...Reminder that $M_{:,1} =$[5 3 2 2 4]^T, so we solve:

$$SS = (5-(2.6y+1))^2 + (3-(y+1))^2 + (2-(y+1))^2 + (2-(y+1))^2 + (4-(y+1$$

$$= (4-2.6y)^2 + (2-y)^2 + (1-y)^2 + (1-y)^2 + (3-y)^2$$

$$\frac{d}{dy}SS = -2[(2.6)(4-2.6y) + (2-y) + (1-y) + (1-y) + (3-y)] \implies y = 17.4/10.76$$

UV Decomposition: Optimization

...and we repeat

$$\begin{bmatrix} 2.6 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1.617 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 5.204 & 3.6 & 3.6 & 3.6 & 3.6 \\ 2.617 & 2 & 2 & 2 & 2 \\ 2.617 & 2 & 2 & 2 & 2 \\ 2.617 & 2 & 2 & 2 & 2 \\ 2.617 & 2 & 2 & 2 & 2 \end{bmatrix}$$

For terms like u_{31} we have to deal with blanks in M. What do we do?

UV Decomposition: Optimization

...and we repeat

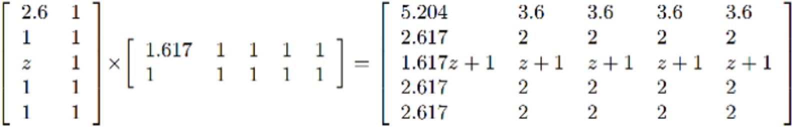
$$\begin{bmatrix} 2.6 & 1 \\ 1 & 1 \\ \boxed{z} & \boxed{1} \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} \boxed{1.617} & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 5.204 & 3.6 & 3.6 & 3.6 & 3.6 \\ 2.617 & 2 & 2 & 2 & 2 \\ 1.617z+1 & z+1 & z+1 & z+1 & z+1 \\ 2.617 & 2 & 2 & 2 & 2 \\ 2.617 & 2 & 2 & 2 & 2 \end{bmatrix}$$

For terms like u_{31} we have to deal with blanks in M. What do we do?

- **Choose** z that minimizes RMSE. Now we're in *row 3* of M'.

UV Decomposition: Optimization

...and we repeat



$$\begin{bmatrix} 2.6 & 1 \\ 1 & 1 \\ z & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1.617 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 5.204 & 3.6 & 3.6 & 3.6 & 3.6 \\ 2.617 & 2 & 2 & 2 & 2 \\ 1.617z+1 & z+1 & z+1 & z+1 & z+1 \\ 2.617 & 2 & 2 & 2 & 2 \\ 2.617 & 2 & 2 & 2 & 2 \end{bmatrix}$$

For terms like u_{31} we have to deal with blanks in M. What do we do?

- **Choose** z that minimizes RMSE. Now we're in *row 3* of M'.
- ...Reminder that $M_{3,:}$ =[2 _ 3 1 4]^T, so we simply omit that term.

$$SS = (2 - (1.617z + 1))^2 + (3 - (z+1))^2 + (1 - (z+1))^2 + (4 - (z+1))^2$$

$$\frac{d}{dz} SS = -2[1.617(2 - (1.617z + 1)) + (3 - (z+1)) + (1 - (z+1)) + (4 - (z+1))]$$

$$\implies z = 6.617/5.615 = 1.178$$

UV Decomposition: Optimization

$$\begin{bmatrix} 2.6 & 1 \\ 1 & 1 \\ 1.178 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1.617 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 5.204 & 3.6 & 3.6 & 3.6 & 3.6 \\ 2.617 & 2 & 2 & 2 & 2 \\ 2.905 & 2.178 & 2.178 & 2.178 & 2.178 \\ 2.617 & 2 & 2 & 2 & 2 \\ 2.617 & 2 & 2 & 2 & 2 \end{bmatrix}$$

We continue until we have updated all the elements of both U and V. Some thoughts:

1. How do we pick which order to update the elements of U and V?

   We could pick some arbitrary order and stick with it each round of updating

   ...But that might introduce some kind of bias (if the columns/rows are ordered)

   If you can, *permuting* the order each time can avoid this (but if the matrices are large, then generating a permutation of their elements can be computationally taxing)

UV Decomposition: General Case

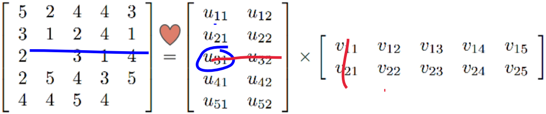
Let $M := m \times n$

$M := m \times n$

$U := m \times d$

$V := d \times n$

$P := UV = M'$



$$\begin{bmatrix} 5 & 2 & 4 & 4 & 3 \\ 3 & 1 & 2 & 4 & 1 \\ \cancel{2} & & \cancel{3} & \cancel{1} & \cancel{4} \\ 2 & 5 & 4 & 3 & 5 \\ 4 & 4 & 5 & 4 & \end{bmatrix} \overset{\heartsuit}{=} \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \\ \boxed{\cancel{u_{31}}} & \cancel{u_{32}} \\ u_{41} & u_{42} \\ u_{51} & u_{52} \end{bmatrix} \times \begin{bmatrix} \cancel{v_{11}} & v_{12} & v_{13} & v_{14} & v_{15} \\ \cancel{v_{21}} & v_{22} & v_{23} & v_{24} & v_{25} \end{bmatrix}$$

In general, what is the update for an arbitrary element of U or V? Consider u_{rs}.

1. $\boxed{u_{rs}}$ only affects row r of P.
   $r=3$ $s=1$
2. The elements of that row of P are sum of the dot products of row r of u with *every* column of v. In other words:

$$p_{rj} = \sum_{k=1}^{d} u_{rk} v_{kj} = \sum_{k \neq s} \overset{u_{32}}{u_{rk}} v_{kj} + x v_{sj}$$

$x = u_{31}$

UV Decomposition: General Case

Consider updating u_{rs}. Then $p_{rj} = \sum_{k=1}^{d} u_{rk}v_{kj} = \sum_{k \neq s} u_{rk}v_{kj} + xv_{sj}$ and the resulting sum of squares error per term is from $M - P$:

$$(m_{rj} - p_{rj})^2 = \left(m_{rj} - \sum_{k \neq s} u_{rk}v_{kj} + xv_{sj} \right)^2$$

for a total error in row r of

$$\sum_{j} \left(m_{rj} - \sum_{k \neq s} u_{rk}v_{kj} + xv_{sj} \right)^2$$

We take a derivative with respect to x, set it equal to zero:

$$0 = \sum_{j} -2v_{sj} \left(m_{rj} - \sum_{k \neq s} u_{rk}v_{kj} + xv_{sj} \right)$$

UV Decomposition: General Case

Solving for $x = u_{rs}$ looks kind of like an average, but a bit gross:

$$x = \frac{\sum_j v_{sj} \left(\boxed{m_{rj}} - \underline{\sum_{k \neq s} u_{rk} v_{kj}} \right)}{\sum_j v_{sj}^2}$$

and a nearly identical calculation that replaces rows for columns shows that to update v_{rs}:

$$y = \frac{\sum_i u_{ir} \left(m_{is} - \sum_{k \neq r} u_{ik} v_{ks} \right)}{\sum_i u_{ir}^2}$$

Sanity check: are these derivative-equals-zero values actually local *minima*? How would we prove this?

UV Decomposition: Implementation Notes

Recall from Recommender Systems that frequently we want to normalize the utility matrix M to account for differences in the ways users rate items, or differences in item ratings.

Some options: (in context of rec. systems but maps to other applications too)

1. Subtract from each non-blank element m_{ij} the average rating of user i
2. Subtract from each non-blank element in column j the average rating of item j
3. Do both of these, in either order
4. From element m_{ij} subtract $\frac{1}{2}\cdot$ (the average of user i + the average of item j)
5. Or similar.

Crucially, whatever we do, if we make predictions, must add back in whatever was subtracted out after we've computed U and V.

UV Decomposition: Implementation Notes

One danger we may run into in optimization/estimation problems (UV, and others) is *overfitting* our solution to the training data, and not performing well on out-of-sample or new data.

Some options to combat this:

1. Only move an element a fraction of the way towards the RMSE-minimizing value (of course, using a convergence/tolerance check may make us just move all the way there, eventually!)
2. Take several different UV decompositions (different initial conditions), and average all of their predictions for a particular entry in M

   If we do this, we can weight by each solutions eventual RMSE
3. Can use a loose convergence criterion to stop revisiting elements in U and V

   Idea: Maintain a list of "good" estimates, only work on others.

UV Decomposition: Big Picture

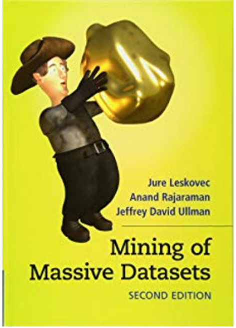
Given an $n \times m$ utility or data matrix M, with some unknown/blank entries...

1. Preprocess M: e.g. normalize
2. Initialize: decompose with *many* different initial U and V because local minima lurk
3. Iterate: do multiple training epochs, looping over all elements in U and V each time
4. Converge: don't expect RMSE to reach 0, but once its epoch-to-epoch change is $<$ tolerance we can break. We can also stop once no component improves RMSE by more than some threshold (e.g. a max noem)
5. Would another choice of d work better? Try it, and recall **elbow** plots of d as the x-axis and $RMSE$ as the y-axis. More d is always (expected to be) less $RMSE$, but with diminishing returns.

Acknowledgments

Next time: Notebooks, then some other matrix properties

Some material is adapted/adopted from Mining of Massive Data Sets, by Jure Leskovec, Anand Rajaraman, Jeff Ullman (Stanford University) `http://www.mmds.org`



Special thanks to Tony Wong for sharing his original adaptation and adoption of slide material.