Matrix Decompositions

Anthony Sicilia

Motivation

- We very often have a data matrix $\mathbf{X}_{\mathbf{nxd}}$
- **Matrix decompositions:** write (sometimes approximate) the original data-matrix as a product of more simple pieces with nice properties
  - Analyze our data using more descriptive features
  - Solve matrix equations that are otherwise inefficient or impossible
- **Plan:**
  1. Start with **Singular Value Decomposition** (works on any matrix)
  2. Many applications of SVD
  3. **Non-Negative Matrix Factorization** (approximate, non-unique)
  4. Tensor Decomposition (if time allows)

Singular Value Decomposition

SVD of a matrix **A** (a set of n points in $\mathbb{R}^{\mathrm{d}}$ with rank r)

$$\mathbf{A}_{\mathrm{n}\times\mathrm{d}} = \mathbf{U}_{\mathrm{n}\times r}\mathbf{\Sigma}_{r\times r}\mathbf{V}_{r\times\mathrm{d}}^{\mathrm{T}}$$

- **U** : Left Singular Vectors of **A** (symmetric and orthonormal)
- **V** : Right Singular Vectors of **A** (symmetric and orthonormal)
- **Σ**: Rectangular diagonal matrix with positive real entries
- **Note:** sometimes we use $r = d$ with $\sigma_i = 0$ if $i > rank(A)$

$$\mathbf{A} = \begin{bmatrix} \mathrm{u}_1 & \dots & \mathrm{u}_r & \dots & \mathrm{u}_\mathrm{n} \end{bmatrix} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix} \begin{bmatrix} \mathrm{v}_1^\mathrm{T} & \dots & \mathrm{v}_r^\mathrm{T} & \dots & \mathrm{v}_\mathrm{d}^\mathrm{T} \end{bmatrix}$$

$$\mathbf{A} = \mathbf{U\Sigma V}^\mathrm{T} = \mathrm{u}_1\sigma_1\mathrm{v}_1^\mathrm{T} + \dots + \mathrm{u}_\mathrm{k}\sigma_r\mathrm{v}_r^\mathrm{T} = \sum_{\mathrm{i}=1}^{r}\mathrm{u}_\mathrm{i}\sigma_\mathrm{i}\mathrm{v}_\mathrm{i}^\mathrm{T}$$

Credit: Sumedha Singla

Singular Value Decomposition (SVD)

- **Some properties to note:**
- Matrices **U, V** are symmetric and orthonormal.
- **Orthonormal:** if a matrix's rows/columns are orthogonal unit vectors
  - **Note:** geometrically represent rotations and **inverse is exactly the matrix transpose**
- **Σ** (diagonal) contains **the singular values** $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r$
- The columns of **U** span the **column space** of **A**. To see this note,

$$\mathbf{A}\boldsymbol{x} = \sigma_1\boldsymbol{u}_1\boldsymbol{v}_1^T\boldsymbol{x} + \cdots + \sigma_r\boldsymbol{u}_r\boldsymbol{v}_r^T\boldsymbol{x} = \sigma_1(\boldsymbol{v}_1^T\boldsymbol{x})\boldsymbol{u}_1 + \cdots + \sigma_1(\boldsymbol{v}_r^T\boldsymbol{x})\boldsymbol{u}_r$$

- Likewise, looking at $\mathbf{A}^\mathbf{T}$, the columns of **V** span the **row space** of **A**
- **V** is a basis for the rows of **A** and **U** is a basis for the columns of **A**

Construction (proof sketch) of SVD ($\mathbf{A = U\Sigma V^T}$)

Our **goal** is to identify $\mathbf{A = U\Sigma V^T}$ with the r columns of **U** and **V** orthonormal and $\mathbf{\Sigma}$ diagonal. **Symmetric matrices** have a nice property that will allow us to find **U** and **V** so we write as below

$$\mathbf{A^T A} = \left(\mathbf{V\Sigma^T U^T}\right)\left(\mathbf{U\Sigma V^T}\right) = \mathbf{V\Sigma^T \Sigma V^T} \qquad (*)$$

Why is $\mathbf{A^T A}$ symmetric? Any real matrix multiplied with its transpose is always **symmetric**, since

$$(\mathbf{AA^T})^{\boldsymbol{T}} = (\mathbf{A^T})^{\boldsymbol{T}}\mathbf{A^T} = \mathbf{AA^T}$$

Construction (proof sketch) of SVD ($\mathbf{A = U\Sigma V^T}$)

We can use the symmetry of $\mathbf{A^T A}$ because any symmetric has an **eigenvalue decomposition**:

$$\mathbf{A^T A} = \mathbf{Q\Lambda Q^T}$$

where the columns of **Q** are orthogonal **eigenvectors** of $\mathbf{A^T A}$ and $\mathbf{\Lambda}$ is the diagonal matrix of **eigenvalues** for $\mathbf{A^T A}$

$$\mathbf{A^T A}\boldsymbol{q}_i = \boldsymbol{\lambda}_i \boldsymbol{q}_i$$

Construction (proof sketch) of SVD ($\mathbf{A} = \mathbf{U\Sigma V^T}$)

Revisiting $(*)$, we can pick $\mathbf{V} = \mathbf{Q}$ and $\mathbf{\Sigma^T\Sigma} = \mathbf{\Lambda}$ (i.e., each $\sigma_i^2 = \lambda_i$)

$$\mathbf{Q\Lambda Q^T} = \mathbf{A^T A} = (\mathbf{V\Sigma^T U^T})(\mathbf{U\Sigma V^T}) = \mathbf{V\Sigma^T\Sigma V^T} \qquad (*)$$

Now, each column $\boldsymbol{v_i}$ will be orthonormal by default!

Next, we also require $\mathbf{A}\boldsymbol{v}_i = \sigma_i \boldsymbol{u}_i$ which is accomplished if we pick

$$\boldsymbol{u}_i = 1/\,\sigma_i \cdot \mathbf{A}\boldsymbol{v}_i$$

Note: there will be r non-zero $\boldsymbol{\lambda}_i$ since $\mathbf{A^T A}$ is symmetric, the rank of $\mathbf{A^T A}$ is the same as the rank of $\mathbf{A}$, and any symmetric matrix has as many nonzero eigenvalues as its rank. This means the dimension of **V** and **U** are correct!

Construction (proof sketch) of SVD

We need to verify that the $\boldsymbol{u_i}$ are **also orthonormal**. We can verify this as below. If $i \neq j$

$$\begin{aligned}
\boldsymbol{u}_i^T \boldsymbol{u}_j &= 1/(\sigma_i\sigma_j) \cdot (\mathbf{A}\boldsymbol{v}_i)^T\,(\mathbf{A}\boldsymbol{v}_j) && \text{(choice of } \boldsymbol{u} \text{)}\\
&= 1/(\sigma_i\sigma_j) \cdot \boldsymbol{v}_i{}^T(\mathbf{A}^T\mathbf{A}\boldsymbol{v}_j) && \text{(defn. of transpose)}\\
&= (\sigma_j\,/\,\sigma_i) \cdot \boldsymbol{v}_i{}^T\,\boldsymbol{v}_j && \text{(} \boldsymbol{v}\text{s is an eigenvector)}\\
&= \mathbf{0} && \text{(} \boldsymbol{v}\text{s are orthonormal)}
\end{aligned}$$

If $i = j$ in the above then the result is 1, so the $\boldsymbol{u}$s are also **unit** and we have out result. **U, V** are orthonormal with r columns and $\mathbf{\Sigma}$ is diagonal!

A Nice Guarantee about SVD

- **(Eckart-Young Theorem)** If a matrix **B** has rank k, then

$$||\mathbf{A} - \mathbf{A}_k|| \leq ||\mathbf{A} - \mathbf{B}||$$

$$\text{where } \mathbf{A}_k = \sigma_1 \boldsymbol{u}_1 \boldsymbol{v}_1^T + \cdots + \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k^T$$

- $||\cdot||$ can be **L2 Norm**, **Frobenius Norm**, or **Trace Norm**
- Useful if we wish to **reduce the dimensionality** of **A** with as little error as possible (e.g. approximation according to some norm)
- No other matrix is a better approximation under these constraints

Connections between SVD and PCA

- Can decompose covariance matrix of 0-mean $\mathbf{A}_{nxd}$ using SVD

$$\mathbf{Cov} = \mathbf{A}^{\mathrm{T}}\mathbf{A}/(\mathrm{n}-1) = \mathbf{V}\,\mathbf{\Sigma}\,\mathbf{U}^{\mathrm{T}}\,\mathbf{U}\,\mathbf{\Sigma}\,\mathbf{V}^{\mathrm{T}}/(\mathrm{n}-1) = \mathbf{V}\frac{\mathbf{\Sigma}^2}{(n-1)}\,\mathbf{V}^{\mathrm{T}}$$

- **Cov** is symmetric, so we can also decompose

$$\mathbf{Cov} = \mathbf{Q}\,\mathbf{\Lambda}\,\boldsymbol{Q}^T$$

Connections between SVD and PCA:

- Putting those two equations together shows us

$$\mathbf{Q}\,\boldsymbol{\Lambda}\,\boldsymbol{Q}^{T} = \mathbf{V}\frac{\boldsymbol{\Sigma}^{2}}{(n-1)}\,\mathbf{V}^{\mathrm{T}}$$

- Right singular vectors $\mathbf{V}$ are the **eigenvectors** of the covariance matrix
- The **eigenvalues** are $\lambda_{\mathrm{i}} = \sigma_{\mathrm{i}}^{2}/(\mathrm{n}-1)$
- The **principal components** are $\mathbf{AV} = \mathbf{U}\,\boldsymbol{\Sigma}\,\mathbf{V}^{\mathrm{T}}\mathbf{V} = \mathbf{U}\,\boldsymbol{\Sigma}$
- Remark: similar approach used in computation of Eigenfaces:
  - Eigenfaces are the **eigenvectors** of the covariance matrix

Some Other Brief Applications of SVD

- Determining **range**, **null space**, and **rank** of **A**
- Matrix approximation (e.g. for compression)
- Inverse and Pseudo-inverse:
  - If $\mathbf{A} = \mathbf{U}\,\boldsymbol{\Sigma}\,\mathbf{V}^{\mathrm{T}}$ and $\boldsymbol{\Sigma}$ is full rank, then $\mathbf{A}^{-1} = \mathbf{V}\,\boldsymbol{\Sigma}^{-1}\mathbf{U}^{\mathrm{T}}$.
  - If $\boldsymbol{\Sigma}$ is singular, then its **pseudo-inverse** is given by $\mathbf{A}^{\dagger} = \mathbf{V}\,\boldsymbol{\Sigma}^{\dagger}\mathbf{U}^{\mathrm{T}}$, where $\boldsymbol{\Sigma}^{\dagger}$ is formed by replacing every nonzero entry by its reciprocal
- Least squares:
  - If we need to solve $\mathbf{Ax} = \mathrm{b}$ in the least-squares sense, then $\mathbf{x}_{\mathrm{LS}} = \mathbf{V}\,\boldsymbol{\Sigma}^{\dagger}\mathbf{U}^{\mathrm{T}}\,\mathrm{b}$
- Denoising – small singular values typically correspond to noise.

Credit: Sumedha Singla

SVD for Latent Semantic Indexing (NLP)

- Form a **term-document matrix** where:
  - **Rows:** represents words
  - **Columns**: represents documents
  - **Value**: the **count** of the words in the document
- **Idea**: apply SVD to identify latent **representations** of the words and documents

$$\mathbf{X} = \begin{array}{c} \\ cosmonaut \\ astronaut \\ moon \\ car \\ truck \end{array} \begin{pmatrix} d1 & d2 & d3 & d4 & d5 & d6 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

SVD for Latent Semantic Indexing (NLP)

- Full example using SVD and $k = r = 5$

$$\mathbf{X} = \begin{array}{c} \\ cosmonaut \\ astronaut \\ moon \\ car \\ truck \end{array} \begin{pmatrix} d1 & d2 & d3 & d4 & d5 & d6 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

$$\mathbf{U} = \begin{array}{c} \\ cosmonaut \\ astronaut \\ moon \\ car \\ truck \end{array} \begin{pmatrix} \dim 1 & \dim 2 & \dim 3 & \dim 4 & \dim 5 \\ -0.44 & -0.30 & 0.57 & 0.58 & 0.25 \\ -0.13 & -0.33 & -0.59 & 0.00 & 0.73 \\ -0.48 & -0.51 & -0.37 & 0.00 & -0.61 \\ -0.70 & 0.35 & 0.15 & -0.58 & 0.16 \\ -0.26 & 0.65 & -0.41 & 0.58 & -0.09 \end{pmatrix}$$

$$\mathbf{\Sigma} = \begin{pmatrix} 2.16 & 0 & 0 & 0 & 0 \\ 0 & 1.59 & 0 & 0 & 0 \\ 0 & 0 & 1.28 & 0 & 0 \\ 0 & 0 & 0 & 1.00 & 0 \\ 0 & 0 & 0 & 0 & 0.39 \end{pmatrix}$$

$$\mathbf{V}^T = \begin{array}{c} \\ \dim 1 \\ \dim 2 \\ \dim 3 \\ \dim 4 \\ \dim 5 \end{array} \begin{pmatrix} d1 & d2 & d3 & d4 & d5 & d6 \\ -0.75 & -0.28 & -0.20 & -0.45 & -0.33 & -0.12 \\ -0.29 & -0.53 & -0.19 & 0.63 & 0.22 & 0.41 \\ 0.28 & -0.75 & 0.45 & -0.20 & 0.12 & -0.33 \\ 0 & 0 & 0.58 & 0 & -0.58 & 0.58 \\ -0.53 & 0.29 & -0.63 & 0.19 & 0.41 & -0.22 \end{pmatrix}$$

Credit: Sumedha Singla

SVD for Latent Semantic Indexing (NLP)

- In practice, **X** is **large, noisy,** or **sparse**. We want a low-rank approximation in a **latent space**. We can pick $k = 2$ latent concepts

$$\mathbf{X} = \begin{array}{c} \\ cosmonaut \\ astronaut \\ moon \\ car \\ truck \end{array} \begin{pmatrix} d1 & d2 & d3 & d4 & d5 & d6 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

$$\mathbf{U} = \begin{array}{c} \\ cosmonaut \\ astronaut \\ moon \\ car \\ truck \end{array} \begin{pmatrix} \text{dim 1} & \text{dim 2} & \text{dim 3} & \text{dim 4} & \text{dim 5} \\ -0.44 & -0.30 & 0 & 0 & 0 \\ -0.13 & -0.33 & 0 & 0 & 0 \\ -0.48 & -0.51 & 0 & 0 & 0 \\ -0.70 & 0.35 & 0 & 0 & 0 \\ -0.26 & 0.65 & 0 & 0 & 0 \end{pmatrix}$$

term-to-concept matrix

$$\boldsymbol{\Sigma} = \begin{pmatrix} 2.16 & 0 & 0 & 0 & 0 \\ 0 & 1.59 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

concept strength matrix

$$\mathbf{V}^{\mathrm{T}} = \begin{array}{c} \\ \text{dim 1} \\ \text{dim 2} \\ \text{dim 3} \\ \text{dim 4} \\ \text{dim 5} \end{array} \begin{pmatrix} d1 & d2 & d3 & d4 & d5 & d6 \\ -0.75 & -0.28 & -0.20 & -0.44 & -0.33 & -0.12 \\ -0.29 & -0.53 & -0.19 & 0.65 & 0.22 & 0.41 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

concept-to-doc matrix

Credit: Sumedha Singla

SVD for Latent Semantic Indexing (NLP)

Original space | Reduced latent semantic space

$$Q = \begin{pmatrix} cosmonaut & 1 \\ astronaut & 0 \\ moon & 0 \\ car & 0 \\ truck & 0 \end{pmatrix} \longrightarrow Q' = \begin{pmatrix} \text{dim 1} & -0.44 \\ \text{dim 2} & -0.30 \end{pmatrix}$$

latent representation of cosmonaut

$$\mathbf{X} = \begin{array}{c} \\ cosmonaut \\ astronaut \\ moon \\ car \\ truck \end{array} \begin{pmatrix} d1 & d2 & d3 & d4 & d5 & d6 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} \longrightarrow \mathbf{V}^{\mathrm{T}} = \begin{array}{c} \\ \text{dim 1} \\ \text{dim 2} \\ \text{dim 3} \\ \text{dim 4} \\ \text{dim 5} \end{array} \begin{pmatrix} d1 & d2 & d3 & d4 & d5 & d6 \\ -0.75 & -0.28 & -0.20 & -0.44 & -0.33 & -0.12 \\ -0.29 & -0.53 & -0.19 & 0.65 & 0.22 & 0.41 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

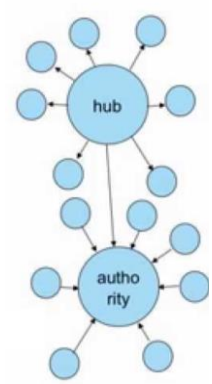
$$\cos(Q, d2) = 0 \longrightarrow \cos(Q', d2) = 0.88$$

Credit: Sumedha Singla

SVD for Latent Semantic Indexing (NLP)

- SVD projects documents and words to a lower dimensional space
- Words and documents are mapped to shared "latent **semantic** space"
  - E.g. King, Pharaoh, Emperor are **semantically** similar
- Rank-lowering combines words into same dimensions
  - Co-occurring words should project on the same dimensions
  - Non-co-occurring words should project onto different dimensions
  - $\{(\mathbf{pot}), (\mathbf{vase}), (\mathbf{dog})\} \rightarrow \{(\mathbf{3.54}\ \times \mathbf{pot}\ + \mathbf{0.36} \times \mathbf{vase}), (\mathbf{dog})\}$
- Mitigates issues of **sparseness** (related to synonymy) and **noisiness**
  - Like our example of **cosmonaut** and **astronaut**

Hyperlink-Induced Topic Search (HITS)

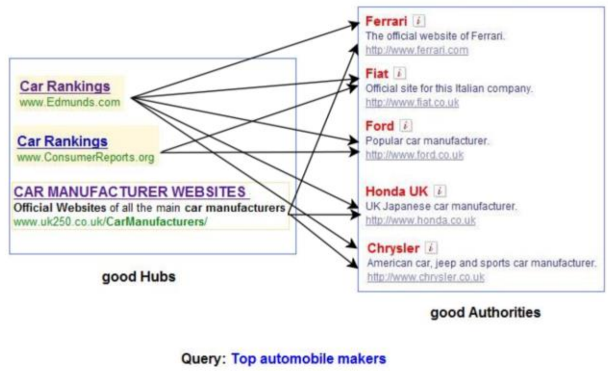


- **(Kleinberg)** ranks web-pages based on query
- Essential vocab:
  - **Authorities**
  - **Hubs**
- **Goal:** Identify good authorities and hubs for a topic
- Each page receives two scores
  - **Authority score** $A(p)$: estimates value of content on page
  - **Hub score** $H(p)$: estimates value of links on page

Credit: Sumedha Singla

Hyperlink-Induced Topic Search (HITS)

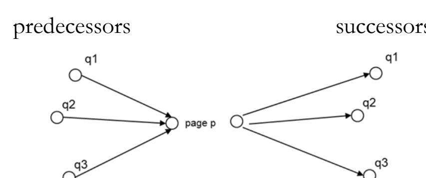
- For a topic, **authorities** are relevant nodes which are referred to by many hubs (**high in-degree**)
- For a topic, **hubs** are nodes which connect many related authorities for that topic (**high out-degree**)

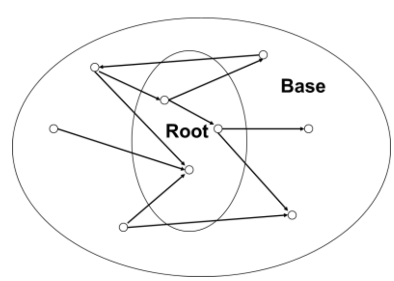


Credit: Sumedha Singla

Hyperlink-Induced Topic Search (HITS)

- Three Steps
  1. Create a focused base-set of the Web
     - Start with a root set (from text-based query)
     - Add immediate **successors** to root set
     - Add immediate **predecessors** to root set (limit d)
     - The extended root set becomes our **base set**
  2. Iteratively compute hub and authority scores
     - A(p): sum of H(q) for all q pointing to p
     - H(q): sum of A(p) for all p pointing to q
     - Starts with all scores as 1, and iteratively repeat till convergence.
  3. Filter out the top hubs and authorities

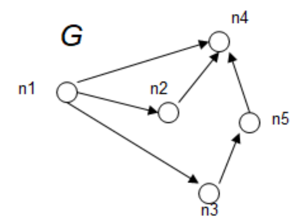




Credit: Sumedha Singla

Hyperlink-Induced Topic Search (HITS)

- **G** (root set) is a directed graph with web pages as **nodes** and their links as **edges**
- **G** can be presented as an adjacency matrix A
  - A(i,j)=1 only if i-th page points to j-th page.
- Authority weights can be represented as a **unit vector a**
  - a_i : The authority weight of the i-th page
- Hub weights can be represented as a **unit vector h**
  - h_i : The hub weight of the i-th page



$$A = \begin{array}{c|ccccc} & n1 & n2 & n3 & n4 & n5 \\ \hline n1 & 0 & 1 & 1 & 1 & 0 \\ n2 & 0 & 0 & 0 & 1 & 0 \\ n3 & 0 & 0 & 0 & 0 & 1 \\ n4 & 0 & 0 & 0 & 0 & 0 \\ n5 & 0 & 0 & 0 & 1 & 0 \end{array}$$

Credit: Sumedha Singla

Hyperlink-Induced Topic Search (HITS)

- Updating authority weights: $a = A^T h$
- Updating hub weights: $h = Aa$
- After k iterations:

$$a_1 = A^T h_0$$
$$h_1 = Aa_1$$
$$\rightarrow h_1 = AA^T h_0$$
$$\rightarrow h_k = (AA^T)_k h_0$$

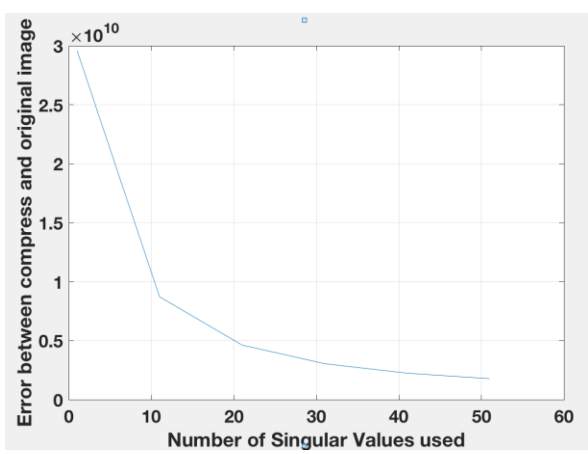
- Convergence
  - a_k: Converges to principal eigenvector of $A^T A$ (singular vector)
  - h_k: Converges to principal eigenvector of AA^T (singular vector)

Credit: Sumedha Singla

Using SVD to Reconstruct an Image

(**Credit:** Kostas Pelechrinis)

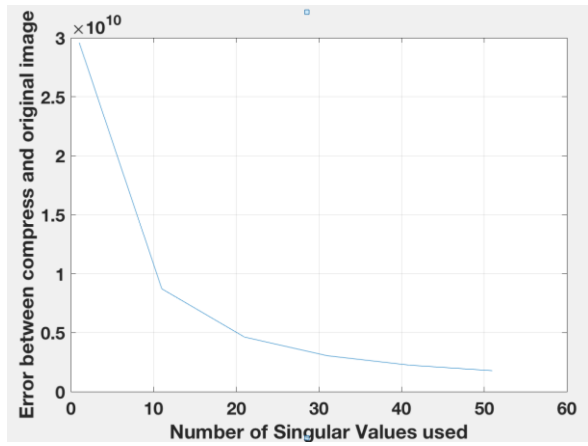
Using SVD to Reconstruct an Image



(**Credit:** Kostas Pelechrinis)

Practical Concerns: How to pick k

- Note we can often pick k based on the **elbow method** (heuristic).
- When does the rate of improvement decrease?
- Can also use a **Scree Plot**
  - Plot of eigenvalues
- For NMF, we will see a more principled procedure



(**Credit:** Kostas Pelechrinis)

Non-Negative Matrix Factorization

- For a non-negative matrix $\mathbf{X}$ we seek factors $\mathbf{X} \approx \mathbf{W}_{nxr}\mathbf{H}_{rxn}$

$$\min_{\boldsymbol{W},\boldsymbol{H}} \|\mathbf{X} - \mathbf{WH}\| \quad \text{s.t. } \mathbf{W}_{ij} \geq 0, \forall i,j \text{ and } \mathbf{H}_{ij} \geq 0, \forall i,j$$

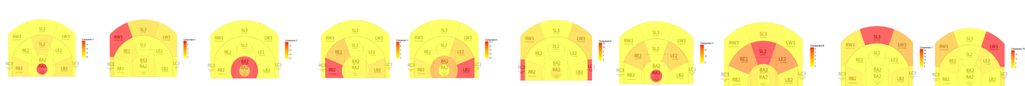
- This decomposes rows and columns of $\mathbf{X}$ into an r dim. feature space **Recall** that SVD provided the best rank r approximation! **Why NMF?**
  - Operates on non-negative data/gives non-negative factors (**intuitive** for counts)
  - **Non-unique**. Useful if connected to a privacy application; e.g., for certain invertible $\mathbf{B}$ we have $\mathbf{WH} = \mathbf{WBB}^{-1}\mathbf{H}$ with the factors still positive
- Norms used may be Frobenius or Matrix Divergence (identical to applying KL Divergence on the elements of a matrix)

NMF Example: Basketball

- **Dataset:** 184,209 shot locations from an NBA season
- **Problem:** Suppose we want to describe the shooting patterns of NBA players. If we discretize the court into 1ft x 1ft squares, there is still more than 2000 locations (high dimensional and probably sparse)
- Our data matrix rows are players, while columns represent locations
  - **Option 1:** court zones (13 columns)
  - **Option 2:** grid cells 1x1 (2,350 columns)
- $\mathbf{X}_{ij}$ is the number of shots player i took from location j

NMF Example: Basketball (Option 1)

- **W** has dimension $\#\text{players} \times r$; rows are player reps. in terms latent patterns
- **H** has dimension $r \times \#\text{locations}$ gives latent shooting patterns

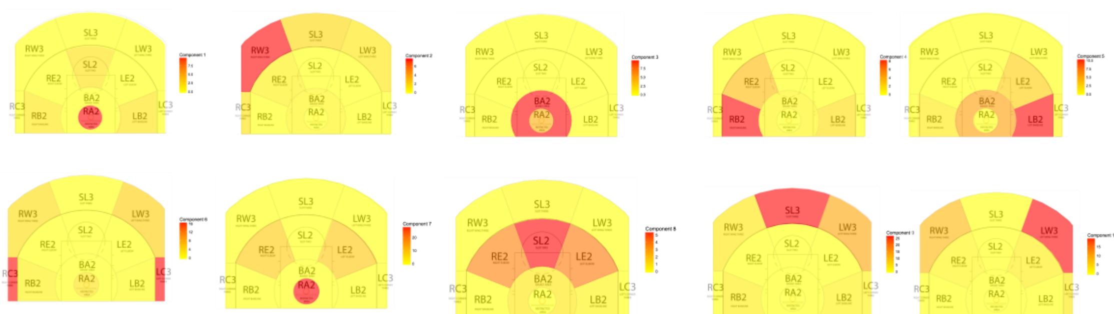


0	21.3	8.6	3.2	4.6	1	8.7	9	4.7	4.6
9.6	19.6	2.5	0.8	0	5.4	3.6	0.79	0.64	2.5
20.2	7.6	8.1	2.44	10.7	0.4	7.7	2.4	1.6	5.2
0	0	1.1	2.12	34.2	0.5	11	19.5	0.7	1.3
0	4	14.2	16.2	4.5	0.8	2.4	15.1	3.5	3.7

(**Credit:** Kostas Pelechrinis)

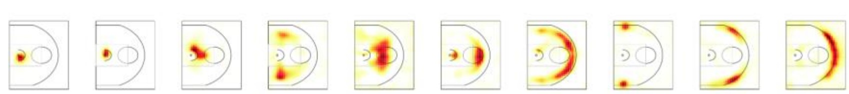
NMF Example: Basketball (Option 1)

- Using a decomposition instead of simply counting frequencies allows disjoint areas to be paired together. E.g., see the **latent concept** of the **corner three** (bottom left)



NMF Example: Basketball (Option 2)

- Miller, Bornn, Adams and Goldsberry (ICML 2014) used a grid over the court and an extension of NMF



LeBron James	0.21	0.16	0.12	0.09	0.04	0.07	0.00	0.07	0.08	0.17
Brook Lopez	0.06	0.27	0.43	0.09	0.01	0.03	0.08	0.03	0.00	0.01
Tyson Chandler	0.26	0.65	0.03	0.00	0.01	0.02	0.01	0.01	0.02	0.01
Marc Gasol	0.19	0.02	0.17	0.01	0.33	0.25	0.00	0.01	0.00	0.03
Tony Parker	0.12	0.22	0.17	0.07	0.21	0.07	0.08	0.06	0.00	0.00
Kyrie Irving	0.13	0.10	0.09	0.13	0.16	0.02	0.13	0.00	0.10	0.14
Stephen Curry	0.08	0.03	0.07	0.01	0.10	0.08	0.22	0.05	0.10	0.24
James Harden	0.34	0.00	0.11	0.00	0.03	0.02	0.13	0.00	0.11	0.26
Steve Novak	0.00	0.01	0.00	0.02	0.00	0.00	0.01	0.27	0.35	0.34

Implementing NMF: How to Solve For Factors

- Solving $\min_{W,H} ||\mathbf{X} - \mathbf{WH}||$ is equivalent to solving $\min_{W,H} ||\mathbf{X} - \mathbf{WH}||^2$
- In general, we will fix **W** and solve for $\nabla_{\mathbf{H}} ||\mathbf{X} - \mathbf{WH}||^2$ or vice-versa
- This simplifies things because $\min_{H} ||\mathbf{X} - \mathbf{WH}||^2$ is convex
- We will use $\nabla_{\mathbf{H}} ||\mathbf{X} - \mathbf{WH}||^2$ and $\nabla_{\mathbf{W}} ||\mathbf{X} - \mathbf{WH}||^2$ frequently, so we'll compute them

- **First some useful facts:**

A. $||\mathrm{X}|| = \sqrt{\mathrm{tr}(\mathrm{X}^{\mathrm{T}}\mathrm{X})}$
B. $\mathrm{tr}(\mathrm{A} + \mathrm{B}) = \mathrm{tr}(\mathrm{A}) + \mathrm{tr}(\mathrm{B})$
C. $\mathrm{tr}(\mathrm{ABC}) = \mathrm{tr}(\mathrm{CAB}) = \mathrm{tr}(\mathrm{BCA})$
D. $\nabla_{\mathrm{X}} \mathrm{tr}(\mathrm{AX}) = \mathrm{A}^{\mathrm{T}}$
E. $\nabla_{\mathrm{X}} \mathrm{tr}(\mathrm{X}^{\mathrm{T}}\mathrm{A}) = \mathrm{A}$
F. $\nabla_{\mathrm{X}} \mathrm{tr}(\mathrm{X}^{\mathrm{T}}\mathrm{AX}) = (\mathrm{A} + \mathrm{A}^{\mathrm{T}})\mathrm{X}$
G. $\nabla_{\mathrm{X}} \mathrm{tr}(\mathrm{XAX}^{\mathrm{T}}) = \mathrm{X}(\mathrm{A}^{\mathrm{T}} + \mathrm{A})$

Analytic Solutions for Factors

Gradient computation for the Frobenius norm w.r.t **H**:

$\boldsymbol{\nabla_H} ||\boldsymbol{X} - \boldsymbol{WH}||^2 = \boldsymbol{\nabla_H}\mathbf{tr[(X{-}WH)}^{\boldsymbol{T}}\mathbf{(X{-}WH)]}$ b/c $||\mathrm{A}|| = \sqrt{\mathrm{tr}(\mathrm{A}^{\mathrm{T}}\mathrm{A})}$

$= \boldsymbol{\nabla_H}\mathbf{tr[X}^{\boldsymbol{T}}\mathbf{X} - \mathbf{X}^{\boldsymbol{T}}\mathbf{WH} - \mathbf{H}^{\boldsymbol{T}}\mathbf{W}^{\boldsymbol{T}}\mathbf{X} + \mathbf{H}^{\boldsymbol{T}}\mathbf{W}^{\boldsymbol{T}}\mathbf{WH]}$

$= \boldsymbol{\nabla_H}\mathbf{tr(X}^{\boldsymbol{T}}\mathbf{X)} - \boldsymbol{\nabla_H}\mathbf{tr(X}^{\boldsymbol{T}}\mathbf{WH)} - \boldsymbol{\nabla_H}\mathbf{tr(H}^{\boldsymbol{T}}\mathbf{W}^{\boldsymbol{T}}\mathbf{X)} + \boldsymbol{\nabla_H}\mathbf{tr(H}^{\boldsymbol{T}}\mathbf{W}^{\boldsymbol{T}}\mathbf{WH)}$ b/c tr is linear

$= \mathbf{0} - \mathbf{W}^{\boldsymbol{T}}\mathbf{X} - \mathbf{W}^{\boldsymbol{T}}\mathbf{X} + (\mathbf{W}^{\boldsymbol{T}}\mathbf{W} + \mathbf{W}^{\boldsymbol{T}}\boldsymbol{W})\mathbf{H}$

b/c $\nabla_{\mathrm{X}} \mathrm{tr}(\mathrm{AX}) = \mathrm{A}^{\mathrm{T}}$, $\nabla_{\mathrm{X}} \mathrm{tr}(\mathrm{X}^{\mathrm{T}}\mathrm{A}) = \mathrm{A}$, and $\nabla_{\mathrm{X}} \mathrm{tr}(\mathrm{X}^{\mathrm{T}}\mathrm{AX}) = (\mathrm{A} + \mathrm{A}^{\mathrm{T}})\mathrm{X}$

$= -\mathbf{2} \cdot \mathbf{W}^{\boldsymbol{T}}\mathbf{X} + \mathbf{2} \cdot \mathbf{W}^{\boldsymbol{T}}\mathbf{WH}$. Setting to zero gives $\mathbf{W}^{\boldsymbol{T}}\mathbf{WH} = \mathbf{W}^{\boldsymbol{T}}\mathbf{X}$

Alternating Least Squares For NMF

A simple approach to finding **W, H** is alternating least squares (ALS)

1. Initialize **W** randomly
2. Estimate **H** from $\mathbf{W}^T\mathbf{W}\mathbf{H} = \mathbf{W}^T\mathbf{X}$ by solving $\min_{\mathbf{H}} \|\mathbf{X} - \mathbf{W}\mathbf{H}\|_{\mathbf{F}}^{\mathbf{2}}$

   (recall, **W** is fixed so we can use standard solver for above)
3. Set all negative elements of **H** to zero or a small positive value
4. Estimate **W** from $\mathbf{H}\mathbf{H}^T\mathbf{W} = \mathbf{H}^{\mathrm{T}}\mathbf{X}$ by solving $\min_{\mathbf{W}} \left\|\mathbf{X}^T - \mathbf{H}^T\mathbf{W}^T\right\|_{\mathbf{F}}^{\mathbf{2}}$
5. Set all negative elements of **W** to zero or a small positive value

- There are simple and more complicated improvements **(Cichoki, 09)**

Multiplicative Update Rules for NMF

- **(Lee and Seung, 01)** provide some guarantees on the below multiplicative update rules (i.e., they are non-decreasing under some assumptions)
- For Frobenius Norm (ops are element-wise),

$$\mathbf{H} \leftarrow \mathbf{H} \otimes \left(\mathbf{W}^T\mathbf{X}\right) \oslash \left(\mathbf{W}^T\mathbf{W}\mathbf{H}\right),$$

$$\mathbf{W} \leftarrow \mathbf{W} \otimes \left(\mathbf{X}\mathbf{H}^T\right) \oslash \left(\mathbf{W}\mathbf{H}\mathbf{H}^T\right)$$

- For the KL Divergence,

$$\mathbf{H}_{au} \leftarrow \mathbf{H}_{au} \frac{\sum_i \mathbf{W}_{ia}\mathbf{X}_{iu}/(\mathbf{W}\mathbf{H})_{iu}}{\sum_k \mathbf{W}_{ka}}, \mathbf{W}_{au} \leftarrow \mathbf{W}_{ia} \frac{\sum_u \mathbf{H}_{au}\mathbf{X}_{iu}/(\mathbf{W}\mathbf{H})_{iu}}{\sum_v \mathbf{H}_{av}}$$

Multiplicative Updates vs. Gradient Based

- We can understand these by their relation to traditional gradient descent
- This is demonstrated for the Frobenius Norm below
- The below additive rule is equivalent to conventional gradient descent if η_{au} is the same for all indices

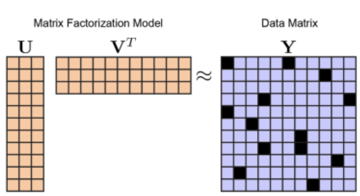
$$\mathbf{H}_{au} \leftarrow \mathbf{H}_{au} + \eta_{au}[(\mathbf{W}^T\mathbf{X})_{au} - (\mathbf{W}^T\mathbf{W}\mathbf{H})_{au}]$$

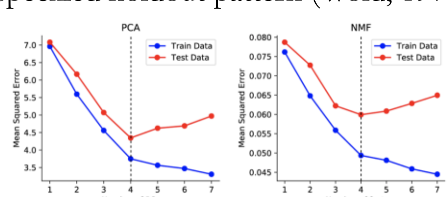
- We can pick η_{au} to arrive at the multiplicative update

$$\eta_{au} = \mathbf{H}_{au}/(\mathbf{W}^T\mathbf{W}\mathbf{H})_{au}$$

Cross Validation via Imputation for NMF

- Why implement ALS? If we wish to cross-validate our NMF through imputation we can do so by implementing ALS and applying a mask (this isn't available in Sklearn)
- Cross-validation through imputation in which we modify the traditional decomposition objective by a mask $\mathbf{M}$ giving $\min_{\boldsymbol{W},\boldsymbol{H}} ||\mathbf{M} \otimes (\mathbf{X} - \mathbf{WH})||$
- **Cannot** holdout entire rows. Below shows a speckled holdout pattern (Wold, 1978)





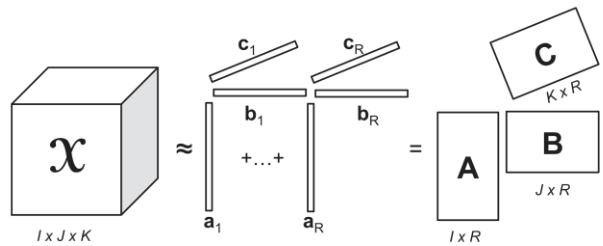
See http://alexhwilliams.info/itsneuronalblog/2018/02/26/crossval/ for a more detailed discussion
And the original paper by Wold: https://www.jstor.org/stable/pdf/1267639.pdf

Decomposing a General Tensor

- In general, we may be interested in decomposing data indexable along ≥ 2 axes
- E.g. term, document, and time **or** term, document, and news organization
- For a brief video overview: https://www.youtube.com/watch?v=L8uT6hgMt00
- For a good textual overview: https://www.cs.ucr.edu/~epapalex/papers/tist16-tensors.pdf

In general, we can write more higher **mode** decompositions as a linear combination of outer products of vectors.

Recall for SVD: $\mathbf{A} = \sum \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T$



Decomposing a General Tensor

- Decomposing a general tensor can be framed as a canonical polyadic (CP) decomposition (also called CANDECOMP and PARAFAC)
- CP Decomposition for a 3-mode tensor into $r = 1{:}R$ components is
  $\mathcal{X} \approx \sum_r \boldsymbol{a}^r \circ \boldsymbol{b}^r \circ \boldsymbol{c}^r$ where $(\boldsymbol{a}^r \circ \boldsymbol{b}^r \circ \boldsymbol{c}^r)_{i,j,k} = \boldsymbol{a}_i^r \times \boldsymbol{b}_j^r \times \boldsymbol{c}_k^r$
- If $\mathcal{X}$ is $nxmxk$ then $\boldsymbol{a}^*$ is nx1, $\boldsymbol{b}^*$ is mx1, $\boldsymbol{c}^*$ is kx1
- We say $\mathcal{X}$ is approximated by a sum of **(tensor) rank-1** tensors
- Easily extends to N-mode case (e.g. include $\boldsymbol{d}^*$)
- CP is unique under fairly mild conditions!
- Variants of ALS are widely used (derived in aforementioned review)

Applications of Tensor Decomposition

- https://www.sciencedirect.com/science/article/pii/S0165027015001016

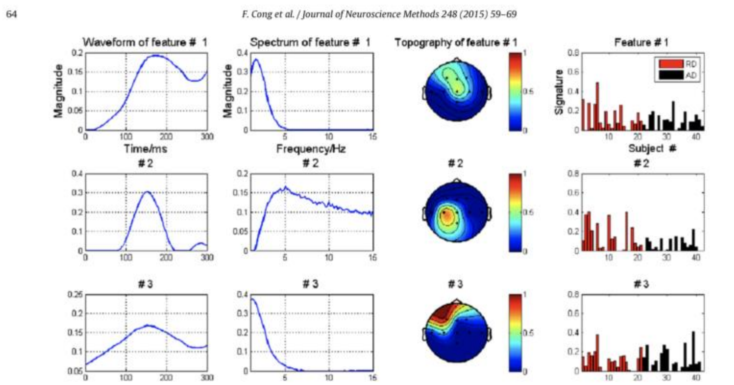


Fig. 3. Example of CPD of a fourth-order ERP tensor including TFR of MMN data with the nonnegativity constraints. The first three components in each mode are shown.

Relevant Packages

- Scikit-Learn has implementations of both NMF and SVD (truncated)
  - https://scikit-learn.org/stable/
- Numpy/Scipy also have an implementation of SVD (full)
  - https://www.scipy.org
- Gensim has an implementation of LSI model that supports updates
  - https://radimrehurek.com/gensim/
- Tensorly has implementations of some tensor decompositions
  - http://tensorly.org/stable/index.html

References

- https://papers.nips.cc/paper/1861-algorithms-for-non-negative-matrix-factorization.pdf
- http://math.mit.edu/classes/18.095/2016IAP/lec2/SVD_Notes.pdf
- http://math.mit.edu/~gs/learningfromdata/
- http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.214.6398&rep=rep1&type=pdf
- https://www.jjburred.com/research/pdf/jjburred_nmf_updates.pdf