Chapter IX: Matrix factorizations*

1. The general idea
2. Matrix factorization methods
3. Latent topic models
4. Dimensionality reduction

*Zaki & Meira, Ch. 8; Tan, Steinbach & Kumar, App. B; Manning, Raghavan & Schütze, Ch. 18
Extra reading: Golub & Van Loan: *Matrix computations*. 3rd ed., JHU press, 1996

IX.2 Matrix factorization methods

1. Eigendecomposition
2. Singular value decomposition (SVD)
3. Principal component analysis (PCA)
4. Non-negative matrix factorization
5. Other topics in matrix factorizations
    5.1. CX matrix factorization
    5.2. Boolean matrix factorization
    5.3. Regularizers
    5.4. Matrix completion

Nonnegative matrix factorization (NMF)

- Eigenvectors and singular vectors can have negative entries even if the data is non-negative
  - This can make the factor matrices hard to interpret in the context of the data
- In **nonnegative matrix factorization** we assume the data is nonnegative and we require the factor matrices to be nonnegative
  - Factors have parts-of-whole interpretation
    - Data is represented as a sum of non-negative elements
  - Models many real-world processes

Definition

- Given a nonnegative n-by-m matrix $\boldsymbol{X}$ (i.e. $x_{ij} \geq 0$ for all i and j) and a positive integer k, find an n-by-k nonnegative matrix $\boldsymbol{W}$ and a k-by-m nonnegative matrix $\boldsymbol{H}$ s.t. $\|\boldsymbol{X} - \boldsymbol{W}\boldsymbol{H}\|_F^2$ is minimized.
  - If $k = \min(n,m)$, we can do $\boldsymbol{W} = \boldsymbol{X}$ and $\boldsymbol{H} = \boldsymbol{I}_m$ (or vice versa)
  - Otherwise the complexity of the problem is unknown
- If either $\boldsymbol{W}$ or $\boldsymbol{H}$ is fixed, we can find the other factor matrix in polynomial time
  - Which gives us our first algorithm…

The alternating least squares (ALS)

- Let's forget the nonnegativity constraint for a while
- The alternating least squares algorithm is the following:
  - Intialize $\boldsymbol{W}$ to a random matrix
  - **repeat**
    - Fix $\boldsymbol{W}$ and find $\boldsymbol{H}$ s.t. $\|\boldsymbol{X} - \boldsymbol{W}\boldsymbol{H}\|_F^2$ is minimized
    - Fix $\boldsymbol{H}$ and find $\boldsymbol{W}$ s.t. $\|\boldsymbol{X} - \boldsymbol{W}\boldsymbol{H}\|_\mathrm{F}^2$ is minimized
  - **until** convergence
- For *unconstrained least squares* we can use $\boldsymbol{H} = \boldsymbol{W}^\dagger\boldsymbol{X}$ and $\boldsymbol{W} = \boldsymbol{X}\boldsymbol{H}^\dagger$
- ALS will typically converge to *local optimum*

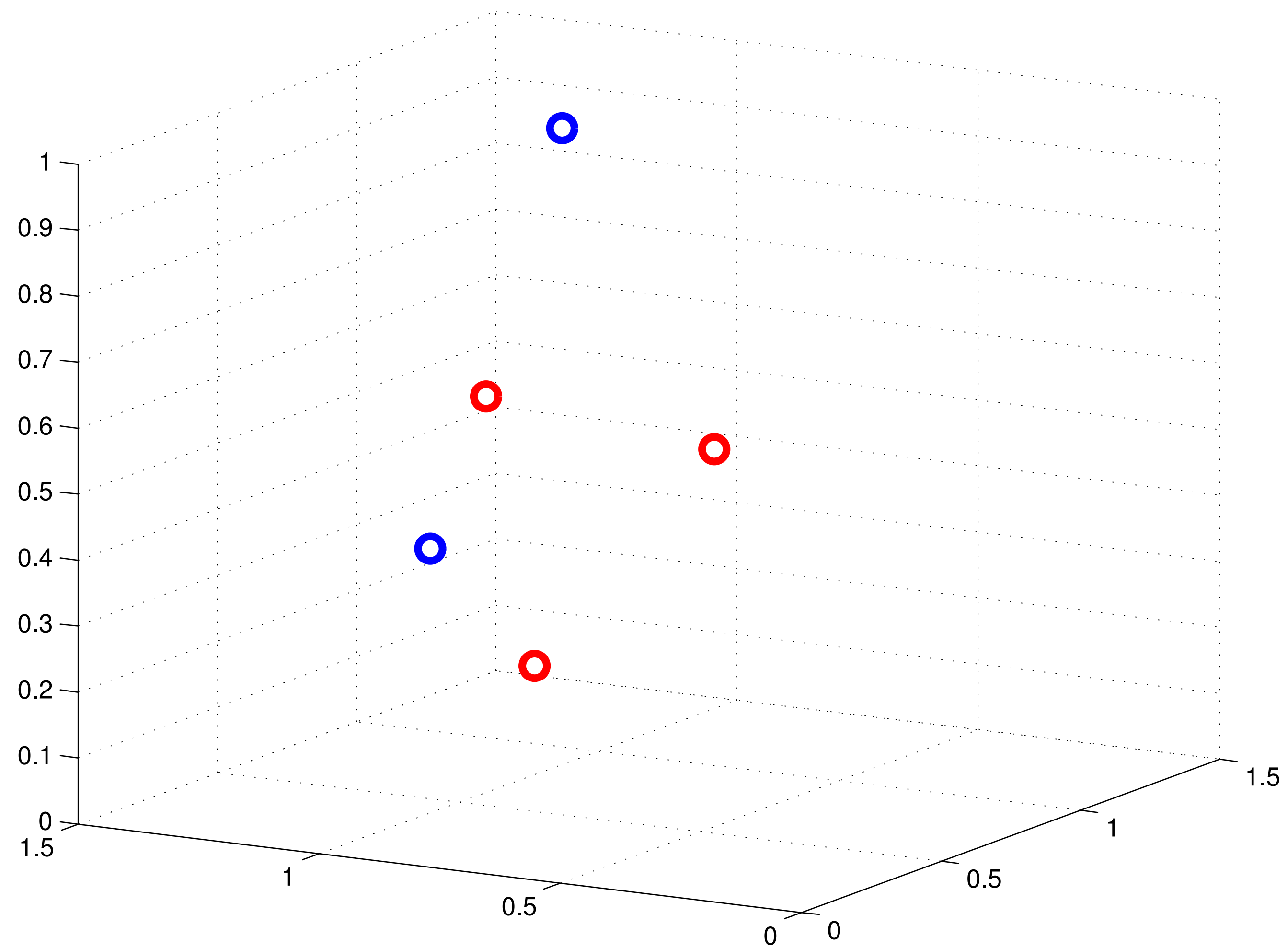
NMF and ALS

- With the nonnegativity constraint pseudo-inverse doesn't work
  - The problem is still *convex* with either of the factor matrices fixed (but not if both are free)
  - We can use *constrained convex optimization*
    - In theory, polynomial time
    - In practice, often too slow
- Poor man's nonnegative ALS:
  - Solve $\boldsymbol{H}$ using pseudo-inverse
  - Set all $h_{ij} < 0$ to 0
  - Repeat for $\boldsymbol{W}$

Geometry of NMF

NMF factors

Data points

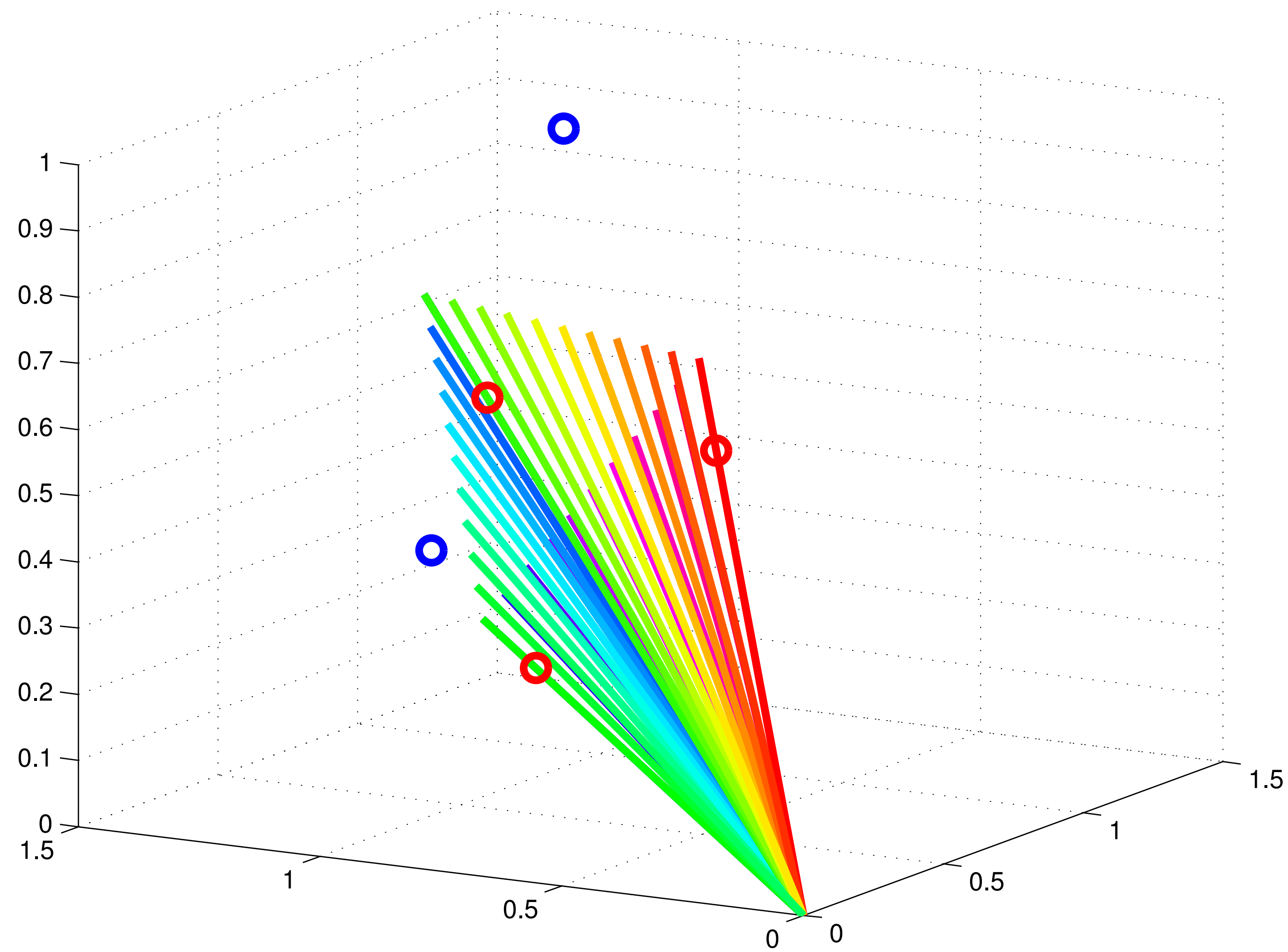


Geometry of NMF

NMF factors

Data points

Convex cone



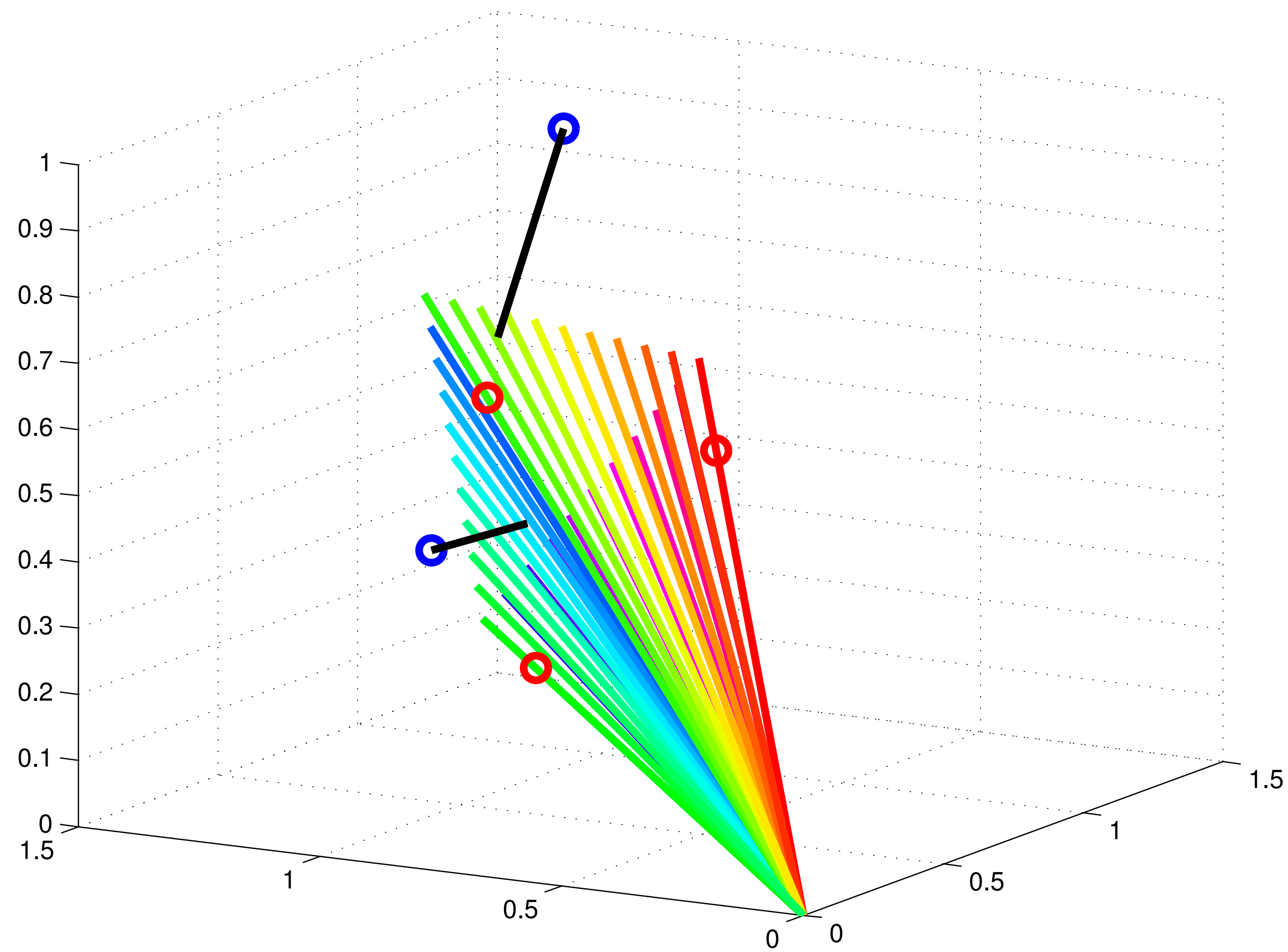
Geometry of NMF

NMF factors

Data points

Convex cone

Projections



Multiplicative update rules

- Idea: update W and H in small steps towards the locally optimum solution
  - Honor the non-negativity constraint
  - Lee & Seung, *Nature, '99*:

1. Initialize $\boldsymbol{W}$ and $\boldsymbol{H}$ randomly to non-negative matrices
2. **repeat**
   2.1. $\boldsymbol{H} = \boldsymbol{H}.*(\boldsymbol{W}^T\boldsymbol{X})./(\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{H} + \varepsilon)$
   2.2. $\boldsymbol{W} = \boldsymbol{W}.*(\boldsymbol{X}\boldsymbol{H}^T)./(\boldsymbol{W}\boldsymbol{H}\boldsymbol{H}^T + \varepsilon)$
3. **until** convergence in $\|\boldsymbol{X} - \boldsymbol{W}\boldsymbol{H}\|_F$

- Here .* is element-wise product, $(\boldsymbol{A}.*\boldsymbol{B})_{ij} = a_{ij}*b_{ij}$, and ./ is element-wise division, $(\boldsymbol{A}./\boldsymbol{B})_{ij} = a_{ij}/b_{ij}$
- Little value ε is added to avoid division by 0

Discussion on multiplicative updates

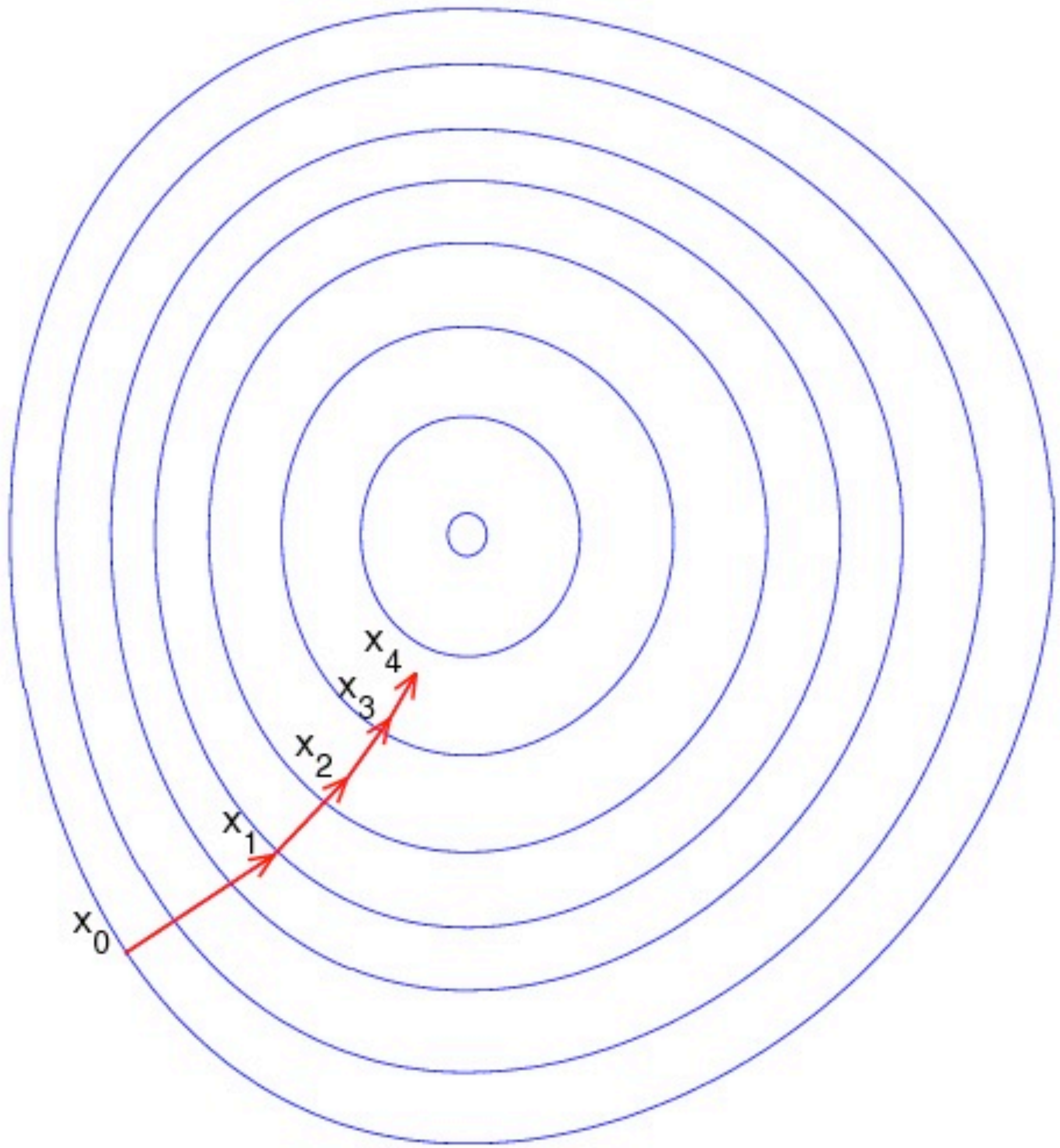
- If $\boldsymbol{W}$ and $\boldsymbol{H}$ are initialized to strictly positive matrices, they stay strictly positive throughout the algorithm
  - Multiplicative form of updates
- If $\boldsymbol{W}$ and $\boldsymbol{H}$ have zeros, the zeros stay
- Converges slowly
  - And has issues when the limit point lies in the boundary
- Lots of computation per update
  - Clever implementation helps
  - Simple to implement

Gradient descent

- Consider the representation error as a function of $\boldsymbol{W}$ and $\boldsymbol{H}$
  - $f: \mathbb{R}^{n \times k} \times \mathbb{R}^{k \times m} \to \mathbb{R}_+$, $f(\boldsymbol{W}, \boldsymbol{H}) = \|\boldsymbol{X} - \boldsymbol{W}\boldsymbol{H}\|_F^2$
  - We can compute the partial derivatives $\partial f/\partial \boldsymbol{W}$ and $\partial f/\partial \boldsymbol{H}$
- **Observation**: The biggest decrease in f at point $(\boldsymbol{W}, \boldsymbol{H})$ happens at the opposite direction of the gradient
  - But this only holds in an ε-neighborhood of $(\boldsymbol{W},\boldsymbol{H})$
  - Therefore, we make small steps opposite to gradient and re-compute the gradient

Example of gradient descent

Image: Wikipedia



NMF and gradient descent

1. Initialize $\boldsymbol{W}$ and $\boldsymbol{H}$ randomly to non-negative matrices
2. **repeat**
   2.1. $\boldsymbol{H} = \boldsymbol{H} - \varepsilon_{\boldsymbol{H}} \partial f / \partial \boldsymbol{H}$
   2.2. $\boldsymbol{W} = \boldsymbol{W} - \varepsilon_{\boldsymbol{W}} \partial f / \partial \boldsymbol{W}$
3. **until** convergence in $\|\boldsymbol{X} - \boldsymbol{WH}\|_F$

NMF and gradient descent

Step size

1. Initialize $\boldsymbol{W}$ and $\boldsymbol{H}$ randomly to non-negative matrices
2. **repeat**
   2.1. $\boldsymbol{H} = \boldsymbol{H} - \varepsilon_{\boldsymbol{H}} \partial f / \partial \boldsymbol{H}$
   2.2. $\boldsymbol{W} = \boldsymbol{W} - \varepsilon_{\boldsymbol{W}} \partial f / \partial \boldsymbol{W}$
3. **until** convergence in $\|\boldsymbol{X} - \boldsymbol{W}\boldsymbol{H}\|_F$

Step size

Issues with gradient descent

- Step sizes are important
  - Too big step size: error increases, not decrease
  - Too small step size: very slow convergence
  - Fixed step sizes don't work
    - Have to adjust somehow
  - Lots of research work put on this
- Ensuring the non-negativity
  - The updates can make factors negative
  - Easiest option: change all negative values to 0 after each update
- Updates are expensive
- Multiplicative update is a type of gradient descent
  - Essentially, the step size is adjusted

ALS vs. gradient descent

- Both are *general* techniques
  - Not tied to NMF
- More general version of ALS is called *alternating projections*
  - Like ALS, but not tied to least-squares optimization
  - We must know how to optimize one factor given the other
    - Or we can approximate this, too…
- In gradient descent function must be derivable
  - (Quasi-)Newton methods study also the second derivative
    - Even more computationally expensive
  - Stochastic gradient descent updates random parts of factors
    - Computationally cheaper but can yield slower convergence

Other topics in matrix factorizations

- Eigendecomposition, SVD, PCA, and NMF are just few examples of possible factorizations
- New factorizations try to address specific issues
  - Sparsity of the factors (number of non-zero elements)
  - Interpretability of the factors
  - Other loss functions (sum-of-absolute differences, …)
  - Over- and underfitting
  - …

The CX factorization

- Given a data matrix $\boldsymbol{D}$, find a subset of columns of $\boldsymbol{D}$ in matrix $\boldsymbol{C}$ and a matrix $\boldsymbol{X}$ s.t. $\|\boldsymbol{D} - \boldsymbol{CX}\|_F$ is minimized
  - Interpretability: if columns of $\boldsymbol{D}$ are easy to interpret, so are columns of $\boldsymbol{C}$
  - Sparsity: if all columns of $\boldsymbol{D}$ are sparse, so are columns of $\boldsymbol{C}$
  - Feature selection: selects actual columns
  - Approximation accuracy: if $\boldsymbol{D}_k$ is the rank-k truncated SVD of $\boldsymbol{D}$ and $\boldsymbol{C}$ has k columns, then with high probability

$$\|D - CX\|_F \leqslant O(k\sqrt{\log k})\,\|D - D_k\|_F$$

[Boutsidis, Mahoney & Drineas, KDD ’08, SODA ’09]

Tiling databases

- Let X be n-by-m binary matrix (e.g. transaction data)
  - Let $\boldsymbol{r}$ be a p-dimensional vector of row indices ($1 \leq \boldsymbol{r}_i \leq n$)
  - Let $\boldsymbol{c}$ be a q-dimensional vector of column indices ($1 \leq \boldsymbol{c}_j \leq m$)
  - The p-by-q *combinatorial submatrix* induced by $\boldsymbol{r}$ and $\boldsymbol{c}$ is

$$\mathbf{X}(\mathbf{r}, \mathbf{c}) = \begin{pmatrix} x_{r_1 c_1} & x_{r_1 c_2} & x_{r_1 c_3} & & x_{r_1 c_q} \\ x_{r_2 c_1} & x_{r_2 c_2} & x_{r_2 c_3} & \cdots & x_{r_2 c_q} \\ x_{r_3 c_1} & x_{r_3 c_2} & x_{r_3 c_3} & & x_{r_3 c_q} \\ & \vdots & & \ddots & \vdots \\ x_{r_p c_1} & x_{r_p c_2} & x_{r_p c_3} & \cdots & x_{r_p c_q} \end{pmatrix}$$

  - $\boldsymbol{X}(\boldsymbol{r},\boldsymbol{c})$ is *monochromatic* if all of its values have the same value (0 or 1 for binary matrices)
    - If $\boldsymbol{X}(\boldsymbol{r},\boldsymbol{c})$ is monochromatic 1, it (and $(\boldsymbol{r},\boldsymbol{c})$ pair) is called a *tile*

[Geerts, Goethals & Mielikäinen, DS '04]

Tiling problems

- **Minimum tiling.** Given $\boldsymbol{X}$, find the least number of tiles $(\boldsymbol{r},\boldsymbol{c})$ such that
  - For all (i,j) s.t. $x_{ij} = 1$, there exists at least one pair $(\boldsymbol{r},\boldsymbol{c})$ such that $i \in \boldsymbol{r}$ and $j \in \boldsymbol{c}$ (i.e. $x_{ij} \in \boldsymbol{X}(\boldsymbol{r},\boldsymbol{c})$)
    - $i \in \boldsymbol{r}$ if exists j s.t. $r_j = i$
- **Maximum k-tiling.** Given $\boldsymbol{X}$ and integer k, find k tiles $(\boldsymbol{r}, \boldsymbol{c})$ such that
  - The number of elements $x_{ij} = 1$ that do not belong in some $\boldsymbol{X}(\boldsymbol{r},\boldsymbol{c})$ is minimized

Tiling and itemsets

- Each tile defines an itemset and a set of transactions where the itemset appears
  - Minimum tiling: each recorded transaction–item pair must appear in some tile
  - Maximum *k*-tiling: minimize the number of transaction–item pairs *not* appearing on selected tiles
- Itemsets are local patterns, but tiling is global

Algorithm for tiling

- Algorithm for tiling:
  - Find all itemset, inducing tiles
  - Select first the biggest-area tile (pq is largest) and mark the submatrix *covered*
  - Select the tile that has most not-yet covered elements and mark it covered
  - Repeat previous step until
    - all transaction–item pairs are covered or
    - we have selected k tiles
- Problem: exponential number of itemsets
  - Heuristic solution: mine only reasonably frequent closed itemsets

Tiling and matrix factorizations

- An index vector can be represented using an *incidence vector*
  - The incidence vector of $\boldsymbol{r}$ is a binary n-dimensional vector $\chi(\boldsymbol{r})$ s.t. $\chi(\boldsymbol{r})_i = 1$ iff $i \in \boldsymbol{r}$
- The submatrix $\boldsymbol{X}(\boldsymbol{r},\boldsymbol{c})$ can be written as $\chi(\boldsymbol{r})\chi(\boldsymbol{c})^T$
  - n-by-m binary matrix with $(\chi(\boldsymbol{r})\chi(\boldsymbol{c})^T)_{ij} = 1$ iff $i \in \boldsymbol{r}$ and $j \in \boldsymbol{c}$
  - Columns of $\boldsymbol{R}$ are the incidence vectors of k row indices for tiles
    - Columns of $\boldsymbol{C}$ are the incidence vectors of k column indices for tiles
  - The non-zeros of $\boldsymbol{RC}^T$ define the transaction–item pairs in the tiling

Boolean matrix multiplication

- We want to write:
  - Minimum tiling: find $\boldsymbol{R}$ and $\boldsymbol{C}$ s.t. $\boldsymbol{X} = \boldsymbol{R}\boldsymbol{C}^T$
  - Maximum k-tiling: find $\boldsymbol{R}$ and $\boldsymbol{C}$ s.t. $|\boldsymbol{X} - \boldsymbol{R}\boldsymbol{C}^T|$ is minimized
- But this is wrong
  - $\boldsymbol{R}\boldsymbol{C}^T$ is not binary, can have values > 1 (overlap)
  - Notice how clustering avoids this!
- Intuitively we do set union
  - If x_{ij} belongs to many tiles, we still count it only once
- Solution: *Boolean matrix multiplication*

$$(\mathbf{R} \circ \mathbf{C}^T)_{ij} = \bigvee_{l=1}^{k} r_{il} c_{jl}$$

Boolean matrix multiplication

- We want to write:
  - Minimum tiling: find $\boldsymbol{R}$ and $\boldsymbol{C}$ s.t. $\cancel{\boldsymbol{X} = \boldsymbol{R}\boldsymbol{C}^T}$ $X = R \circ C^T$
  - Maximum k-tiling: find $\boldsymbol{R}$ and $\boldsymbol{C}$ s.t. $\cancel{|\boldsymbol{X} - \boldsymbol{R}\boldsymbol{C}^T|}$ is minimized $|X - R \circ C^T|$
- But this is wrong
  - $\boldsymbol{R}\boldsymbol{C}^T$ is not binary, can have values > 1 (overlap)
  - Notice how clustering avoids this!
- Intuitively we do set union
  - If x_{ij} belongs to many tiles, we still count it only once
- Solution: *Boolean matrix multiplication*

$$(\mathbf{R} \circ \mathbf{C}^T)_{ij} = \bigvee_{l=1}^{k} r_{il} c_{jl}$$

Boolean matrix factorization (BMF)

- Tiling still requires that the tiles are monochromatic
  - If $(\boldsymbol{R} \circ \boldsymbol{C}^T)_{ij} = 1$ then $\boldsymbol{X}_{ij} = 1$
  - This can be problematic if data has noise
    - Tiles must be broke down
- Removing the monochromaticity requirement gives *Boolean matrix factorization*:
  - Given binary $\boldsymbol{X}$ and nonnegative k, find n-by-k $\boldsymbol{A}$ and k-by-m $\boldsymbol{B}$ s.t. $|\boldsymbol{X} - \boldsymbol{A} \circ \boldsymbol{B}|$ is minimized
- BMF generalizes tiling by allowing noise
- BMF generalizes clustering by allowing overlaps

BMF example

$$\mathbf{X} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

BMF example

$$\mathbf{X} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 0 \end{pmatrix} = \mathbf{b}_1$$

$$\mathbf{a}_1 = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

BMF example

$$\mathbf{X} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 0 \end{pmatrix} = \mathbf{b}_1$$

$$\mathbf{a}_1 = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \mathbf{a}_1\mathbf{b}_1$$

BMF example

$$\mathbf{X} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 0 \end{pmatrix} = \mathbf{b}_1$$

$$\mathbf{a}_1 = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \mathbf{a}_1\mathbf{b}_1$$

$$\begin{pmatrix} 0 & 1 & 1 \end{pmatrix} = \mathbf{b}_2$$

$$\mathbf{a}_2 = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

BMF example

$$\mathbf{X} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 0 \end{pmatrix} = \mathbf{b}_1$$

$$\mathbf{a}_1 = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \mathbf{a}_1 \mathbf{b}_1$$

$$\begin{pmatrix} 0 & 1 & 1 \end{pmatrix} = \mathbf{b}_2$$

$$\mathbf{a}_2 = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} = \mathbf{a}_2 \mathbf{b}_2$$

BMF example

$$\mathbf{X} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} = \mathbf{A} \circ \mathbf{B}$$

$$\begin{pmatrix} 1 & 1 & 0 \end{pmatrix} = \mathbf{b}_1$$

$$\mathbf{a}_1 = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \mathbf{a}_1\mathbf{b}_1$$

$$\begin{pmatrix} 0 & 1 & 1 \end{pmatrix} = \mathbf{b}_2$$

$$\mathbf{a}_2 = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} = \mathbf{a}_2\mathbf{b}_2$$

BMF example

$$\mathbf{X} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} = \mathbf{A} \circ \mathbf{B}$$

$$\begin{pmatrix} 1 & 1 & 0 \end{pmatrix} = \mathbf{b}_1$$

$$\mathbf{a}_1 = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \mathbf{a}_1 \mathbf{b}_1$$

$$\begin{pmatrix} 0 & 1 & 1 \end{pmatrix} = \mathbf{b}_2$$

$$\mathbf{a}_2 = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} = \mathbf{a}_2 \mathbf{b}_2$$

BMF example

$$\mathbf{X} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} = \mathbf{A} \circ \mathbf{B}$$

$$\begin{pmatrix} 1 & 1 & 0 \end{pmatrix} = \mathbf{b}_1$$

$$\mathbf{a}_1 = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \mathbf{a}_1\mathbf{b}_1$$

$$\begin{pmatrix} 0 & 1 & 1 \end{pmatrix} = \mathbf{b}_2$$

$$\mathbf{a}_2 = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} = \mathbf{a}_2\mathbf{b}_2$$

BMF example

$$\mathbf{X} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} = \mathbf{A} \circ \mathbf{B}$$

$$\begin{pmatrix} 1 & 1 & 0 \end{pmatrix} = \mathbf{b}_1$$

$$\mathbf{a}_1 = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \mathbf{a}_1 \mathbf{b}_1$$

$$\begin{pmatrix} 0 & 1 & 1 \end{pmatrix} = \mathbf{b}_2$$

$$\mathbf{a}_2 = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} = \mathbf{a}_2 \mathbf{b}_2$$

Regularizers

- We used regularizers with linear regression to prevent over-fitting
- Similar ideas work with matrix factorization
  - With so-called L_2-regularizer the squared loss function is

$$\|\mathbf{X} - \mathbf{AB}\|_F^2 + \lambda_1 \|\mathbf{A}\|_F^2 + \lambda_2 \|\mathbf{B}\|_F^2$$

    - λ_1 and λ_2 are regularizer parameters
    - The problem is still convex (and quadratic) if one factor is fixed
  - We can mix-and-match distances and regularizers:

$$\|\mathbf{X} - \mathbf{AB}\|_F^2 + \lambda_1 |\mathbf{A}| + \lambda_2 |\mathbf{B}|$$

Matrix completion

- The standard matrix factorization formulation assumes that all values of $\boldsymbol{X}$ are known
- In **matrix completion** setting, some values are unknown
- The idea is to compute a factorization of the data using the known values and fill in the unknown based on this factorization
  - When computing the factorization, unknown values do not cause error

Completion example

$$\mathbf{A} = \begin{pmatrix} ? & 10 & 16 & ? \\ 4 & 9.5 & ? & 19.5 \\ 11 & ? & 39 & ? \end{pmatrix}$$

Completion example

$$\mathbf{A} = \begin{pmatrix} ? & 10 & 16 & ? \\ 4 & 9.5 & ? & 19.5 \\ 11 & ? & 39 & ? \end{pmatrix}$$

$$\mathbf{X} = \begin{pmatrix} 1 & 2 \\ 2 & 0.5 \\ 4 & 3 \end{pmatrix} \qquad \mathbf{Y} = \begin{pmatrix} 2 & 4 & 6 & 8 \\ 1 & 3 & 5 & 7 \end{pmatrix}$$

Completion example

$$\mathbf{A} = \begin{pmatrix} ? & 10 & 16 & ? \\ 4 & 9.5 & ? & 19.5 \\ 11 & ? & 39 & ? \end{pmatrix}$$

$$\mathbf{XY} = \begin{pmatrix} 4 & 10 & 16 & 22 \\ 4 & 9.5 & 14.5 & 19.5 \\ 11 & 25 & 39 & 53 \end{pmatrix}$$

Recommender systems

- Data about users and products
  - Which products users liked/purchased/rented/watched
- Lots of unknowns
  - If user hasn't seen the product, we don't know would she like/buy/rent/watch it
- Goal: recommend new products to users based on what people with similar tastes also liked
  - User's taste is learned from the data
- One way to do this: matrix completion
  - Each column factor corresponds to a "group" of users with similar tastes
  - Each row factor corresponds to a "group" of similarly-liked products

Netflix example

- Data: users and movie ratings (1–5 stars)
  - 1/2 million users, 18 000 movies, 100 million ratings
    - 99% of values were unknown
- Algorithms were compared based on how well they predicted the values in *test set*
  - Ratings known by the jury but unknown to the competitors
- Winner was awarded $1 000 000
- Winning algorithm was an ensemble method
  - Matrix factorization gave very important contribution

IX.3 Latent topic models

1. Basic idea
2. Latent semantic indexing (LSI)
3. Probabilistic latent semantic indexing (pLSI)
4. Latent Dirichlet allocation (LDA)

Basic idea

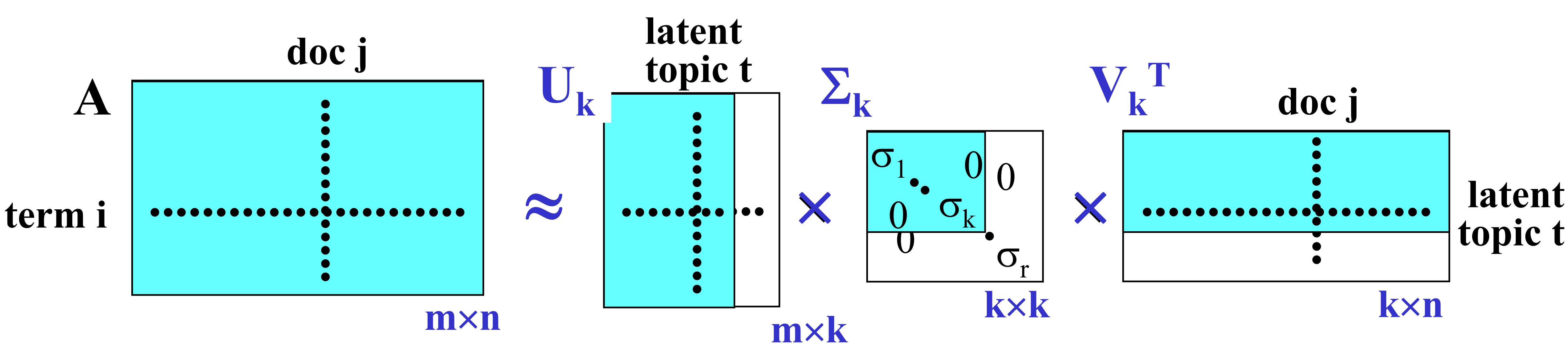
- Consider a terms-by-documents matrix
  - Some terms are synonymous
    - ‘Internet’ and ‘web’
  - Some terms are polysemic
    - ‘Java’ can be island, coffee, or programming language
- We aim to ‘group’ similar terms together
  - We also want to group documents together

Latent topic models

- We assume there's a small number of *latent topics*
- Generative process for documents:
  - Choose (latent) topic
  - Choose terms based on the (latent) topic
- We need to find
  - Mapping between documents and topics
  - Mapping between topics and terms
- But if we want linear mappings, then this is matrix factorization…

Latent semantic indexing (LSI)

- Idea: apply SVD to vector space model
- $\boldsymbol{A}$ is m-by-n term-document matrix and $\boldsymbol{A} = \boldsymbol{U\Sigma V}^T$ its SVD
  - $\boldsymbol{U}_k$, $\boldsymbol{V}_\mathrm{k}$, and $\boldsymbol{\Sigma}_\mathrm{k}$ contain the first k singular vectors and values
- We interpret:
  - $\boldsymbol{U}_k$ maps terms to topics
  - $\boldsymbol{V}_k$ maps documents to topics (or $\boldsymbol{\Sigma V}^T$ topics to documents)



Operations in latent topic space

- An m-dimensional vector $\boldsymbol{q}$ in term space is mapped to the k-dimensional topic space by $\boldsymbol{q} \mapsto \boldsymbol{U}_k{}^T\boldsymbol{q}$
  - Vector $\boldsymbol{q}$ could be a *query* of terms
- The mapped query is evaluated in the topic vector space $\boldsymbol{V}_k$
  - Scalar-product similarity: $\boldsymbol{V}_k{}^T\boldsymbol{q}' = \boldsymbol{V}_k{}^T\boldsymbol{U}_k{}^T\boldsymbol{q}$
  - Alternatively e.g. cosine similarity can be used
- A new document can be transformed to the topic space similarly and then appended to $\boldsymbol{V}_k{}^T$ as a new column
  - Quality deteriorates over time

LSI example (1)

m=6 terms
- t1: bak(e,ing)
- t2: recipe(s)
- t3: bread
- t4: cake
- t5: pastr(y,ies)
- t6: pie

n=5 documents
- d1: How to bake bread without recipes
- d2: The classic art of Viennese Pastry
- d3: Numerical recipes: the art of scientific computing
- d4: Breads, pastries, pies and cakes: quantity baking recipes
- d5: Pastry: a book of best French recipes

$$A = \begin{pmatrix} 0.5774 & 0.0000 & 0.0000 & 0.4082 & 0.0000 \\ 0.5774 & 0.0000 & 1.0000 & 0.4082 & 0.7071 \\ 0.5774 & 0.0000 & 0.0000 & 0.4082 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.4082 & 0.0000 \\ 0.0000 & 1.0000 & 0.0000 & 0.4082 & 0.7071 \\ 0.0000 & 0.0000 & 0.0000 & 0.4082 & 0.0000 \end{pmatrix}$$

LSI example (2)

$$A = \begin{pmatrix} 0.2670 & -0.2567 & 0.5308 & -0.2847 \\ 0.7479 & -0.3981 & -0.5249 & 0.0816 \\ 0.2670 & -0.2567 & 0.5308 & -0.2847 \\ 0.1182 & -0.0127 & 0.2774 & 0.6394 \\ 0.5198 & 0.8423 & 0.0838 & -0.1158 \\ 0.1182 & -0.0127 & 0.2774 & 0.6394 \end{pmatrix} \qquad \boldsymbol{U}$$

$$\times \begin{pmatrix} 1.6950 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 1.1158 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.8403 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.4195 \end{pmatrix} \qquad \boldsymbol{\Sigma}$$

$$\times \begin{pmatrix} 0.4366 & 0.3067 & 0.4412 & 0.4909 & 0.5288 \\ -0.4717 & 0.7549 & -0.3568 & -0.0346 & 0.2815 \\ 0.3688 & 0.0998 & -0.6247 & 0.5711 & -0.3712 \\ -0.6715 & -0.2760 & 0.1945 & 0.6571 & -0.0577 \end{pmatrix} \qquad \boldsymbol{V}^T$$

LSI example (3)

$$A_3 = \begin{pmatrix} 0.4971 & -0.0330 & 0.0232 & 0.4867 & -0.0069 \\ 0.6003 & 0.0094 & 0.9933 & 0.3858 & 0.7091 \\ 0.4971 & -0.0330 & 0.0232 & 0.4867 & -0.0069 \\ 0.1801 & 0.0740 & -0.0522 & 0.2320 & 0.0155 \\ -0.0326 & 0.9866 & 0.0094 & 0.4402 & 0.7043 \\ 0.1801 & 0.0740 & -0.0522 & 0.2320 & 0.0155 \end{pmatrix} = \boldsymbol{U}_3 \boldsymbol{\Sigma}_3 \boldsymbol{V}_3{}^T$$

LSI example (4)

- Query q: baking bread
  - $\boldsymbol{q} = (1\ 0\ 1\ 0\ 0\ 0)^T$
  - $\boldsymbol{q'} = \boldsymbol{U}_3{}^T\boldsymbol{q} = (0.5340\ –0.5134\ 1.0616)^T$
- Scalar product similarity in topic space
  - $\text{sim(q, d1)} = \langle \boldsymbol{V}_3(:,1)^T, \boldsymbol{q'} \rangle \approx 0.86$
  - $\text{sim(q, d2)} = \langle \boldsymbol{V}_3(:,2)^T, \boldsymbol{q'} \rangle \approx –0.12$
  - $\text{sim(q, d3)} = \langle \boldsymbol{V}_3(:,3)^T, \boldsymbol{q'} \rangle \approx –0.24$
- Adding document d6: ”algorithmic recipes for the computation of pie”
  - $\boldsymbol{d} = (0\ 0.7071\ 0\ 0\ 0\ 0.7071)^T$
  - $\boldsymbol{d'} = \boldsymbol{U}_3{}^T\boldsymbol{d} \approx (0.5\ –0.28\ –0.15)^T$
  - $\boldsymbol{d'}$ becomes a new column of $\boldsymbol{V}_k{}^T$

Issues with LSI

- How to select proper *k*?
  - Different *k* makes different terms related
  - We don't know *a priori* which terms are related and which are not
- Memory consumption
  - Terms-by-documents matrices are sparse
    - Most terms don't appear on most documents
  - SVD factors $\boldsymbol{U}$ and $\boldsymbol{V}$ are (almost) never sparse
    - Even if we have relatively small *k*, we might need more space to store the factors than to store the original matrix
- Has not shown convincing results for Web search engines