Chapter IX:
Matrix factorizations

Information Retrieval & Data Mining
Universität des Saarlandes, Saarbrücken
Winter Semester 2011/12

Chapter IX: Matrix factorizations*

*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.1: The general idea

1. The general definition

1.1. Matrix factorizations we've seen so far

1.2. Matrices as data and functions

1.3. Matrix distances and types of matrices

2. Very quick recap of linear algebra

3. Why matrix factorizations

The general definition

- Given *n*-by-*m* matrix ***X***, represent it as a product of two (or more) **factor matrices** ***A*** and ***B***
  - $\mathbf{X} = \mathbf{AB}$
  - We are more interested in *approximate* matrix factorizations

    $\mathbf{X} \approx \mathbf{AB}$
  - Matrix ***A*** is *n*-by-*k*; matrix ***B*** is *k*-by-*m* ($k \leq \min(n, m)$)
    - For more factor matrices, their *inner dimension* must match
- The distance between ***X*** and ***AB*** is the *representation error* of (approximate) factorization
  - E.g. $\|\mathbf{X} - \mathbf{AB}\|_F^2 = \sum_{i=1}^{n} \sum_{j=1}^{m} (x_{ij} - (\mathbf{AB})_{ij})^2$

Variations

- We can change the distance measure
  - Squared element-wise error
  - Absolute element-wise error
- We can restrict the matrices involved
  - Types of values
    - Non-negative
    - Binary
  - Types of factor matrices
    - Upper triangular
    - Diagonal
    - Orthogonal
- We can have more factor matrices
- We can change the matrix multiplication

Matrix factorizations we've seen so far

- Clustering: $\|\mathbf{X} - \mathbf{CM}\|_2^2$
  - $\boldsymbol{C}$ has to be cluster assignment matrix
- Co-clustering: $\|\mathbf{X} - \mathbf{RMC}^\mathsf{T}\|_2^2$
  - $\boldsymbol{R}$ and $\boldsymbol{C}$ are cluster assignment matrices
- Linear regression: $\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2$
  - $\boldsymbol{y}$ is vector, as is $\boldsymbol{\beta}$
  - "decomposes" $\boldsymbol{y}$ – but also is $\boldsymbol{X}$ is known
- Singular value decomposition (SVD) and eigendecomposition
  - Have been mentioned earlier

Two views of a matrix: data or function

- In IR & DM (and most CS) a matrix is a way to write down data
  - A two-dimensional flat database
  - Items and transactions, documents and terms, …
- In linear algebra, a matrix is a linear function between vector spaces
  - n-by-m matrix maps m-dimensional vectors to n-dimensional ones
  - If $\boldsymbol{y} = \boldsymbol{M}\boldsymbol{x}$, then $y_i = \sum_j m_{ij}x_j$
- Different views motivate different techniques

Matrix distances and norms

- Frobenius norm $||\boldsymbol{X}||_F = (\sum_{i,j} x_{ij}^2)^{1/2}$
  - Corresponds to Euclidean norm of vectors
- Sum of absolute values $|\boldsymbol{X}| = \sum_{i,j} x_{ij}$
  - Corresponds to L_1-norm of vectors
- The above elementwise norms are sometimes (imprecisely) called L_2 and L_1 norms
  - Matrix L_1 and L_2 norms are something different altogether
- Operator norm $||\boldsymbol{X}||_p = \max_{\boldsymbol{y}\neq 0} ||\boldsymbol{X}\boldsymbol{y}||_p / ||\boldsymbol{y}||_p$
  - Largest norm of an image of a unit norm vector
  - $||\boldsymbol{X}||_2 \leq ||\boldsymbol{X}||_F \leq \sqrt{(\text{rank}(\boldsymbol{X}))}\ ||\boldsymbol{X}||_2$

Types of matrices

- Diagonal n-by-n matrix
  - Identity matrix $\boldsymbol{I}_n$ is a diagonal n-by-n matrix with 1s in diagonal

$$\begin{pmatrix} x_{1,1} & 0 & 0 & & 0 \\ 0 & x_{2,2} & 0 & \cdots & 0 \\ 0 & 0 & x_{3,3} & & 0 \\ & \vdots & & \ddots & \\ 0 & 0 & 0 & & x_{n,n} \end{pmatrix}$$

- Upper triangular matrix
  - Lower triangular is the transpose
  - If diagonal is full of 0s, matrix is *strictly triangular*

$$\begin{pmatrix} x_{1,1} & x_{1,2} & x_{1,3} & & x_{1,n} \\ 0 & x_{2,2} & x_{2,3} & \cdots & x_{2,n} \\ 0 & 0 & x_{3,3} & & x_{3,n} \\ & \vdots & & \ddots & \\ 0 & 0 & 0 & & x_{n,n} \end{pmatrix}$$

- Permutation matrix
  - Each row and column has exactly one 1, rest are 0

Very quick recap of linear algebra

- An n-by-m matrix $\boldsymbol{X}$ can be represented exactly as a product of n-by-k and k-by-m matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ if and only if rank of $\boldsymbol{X}$ is at most k
  - rank($\boldsymbol{AB}$) $\leq$ min(rank($\boldsymbol{A}$), rank($\boldsymbol{B}$))
  - If rank($\boldsymbol{X}$) = $n \leq m$, we can set $\boldsymbol{A} = \boldsymbol{I}_n$ and $\boldsymbol{B} = \boldsymbol{X}$
  - In general, if $n \leq m$, columns of $\boldsymbol{A}$ are linearly independent *basis vectors* for the subspace spanned by $\boldsymbol{X}$ and columns of $\boldsymbol{B}$ tell the linear combinations of these vectors needed to get the original columns of $\boldsymbol{X}$
- If $\boldsymbol{X}$ is rank-k, it can be written as a sum of k rank-1 matrices, but no fewer
  - Another way to define rank
  - In general, rank($\boldsymbol{A} + \boldsymbol{B}$) $\leq$ rank($\boldsymbol{A}$) + rank($\boldsymbol{B}$)

Spaces

- Let X be an n-by-m (real-valued) matrix
  - Set $\{\boldsymbol{u} \in \mathbb{R}^n : \boldsymbol{X}\boldsymbol{v} = \boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^m\}$ is the *column space* of $\boldsymbol{X}$
    - Image of $\boldsymbol{X}$
  - Set $\{\boldsymbol{v} \in \mathbb{R}^m : \boldsymbol{X}^T\boldsymbol{u} = \boldsymbol{v}, \boldsymbol{u} \in \mathbb{R}^n\}$ is the *row space* of $\boldsymbol{X}$
    - Image of $\boldsymbol{X}^T$
  - Set $\{\boldsymbol{v} \in \mathbb{R}^m : \boldsymbol{X}\boldsymbol{v} = 0\}$ is the *null space* of $\boldsymbol{X}$
  - Set $\{\boldsymbol{u} \in \mathbb{R}^n : \boldsymbol{X}^T\boldsymbol{u} = 0\}$ is the *left null space* of $\boldsymbol{X}$

Orthogonality and orthonormality

- Two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are **orthogonal** if their inner product $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ is 0
  - Vectors are **orthonormal** if they have unit norm, $\|\boldsymbol{x}\|=\|\boldsymbol{y}\|=1$
- A square matrix $\boldsymbol{X}$ is *orthogonal* if its rows and columns are orthonormal
  - Equivalently, $\boldsymbol{X}^T = \boldsymbol{X}^{-1}$
  - Yet equivalently, $\boldsymbol{X}\boldsymbol{X}^T = \boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{I}$

Why matrix factorizations?

- A general way of writing many problems
  - Makes easier to see similarities & differences
  - May help finding new approaches and tools
- A method to remove noise
  - ”True” matrix $\boldsymbol{A}$ is low-rank
  - Observed matrix $\tilde{\boldsymbol{A}}$ has some noise $\boldsymbol{A} + \boldsymbol{\varepsilon}$ and has full rank
  - Finding a low-rank approximation of $\tilde{\boldsymbol{A}}$ helps remove the noise and leave only the original matrix $\boldsymbol{A}$
  - Here we’re interested in the representation of $\boldsymbol{A}$
- Alternatively we can be interested on the factors…

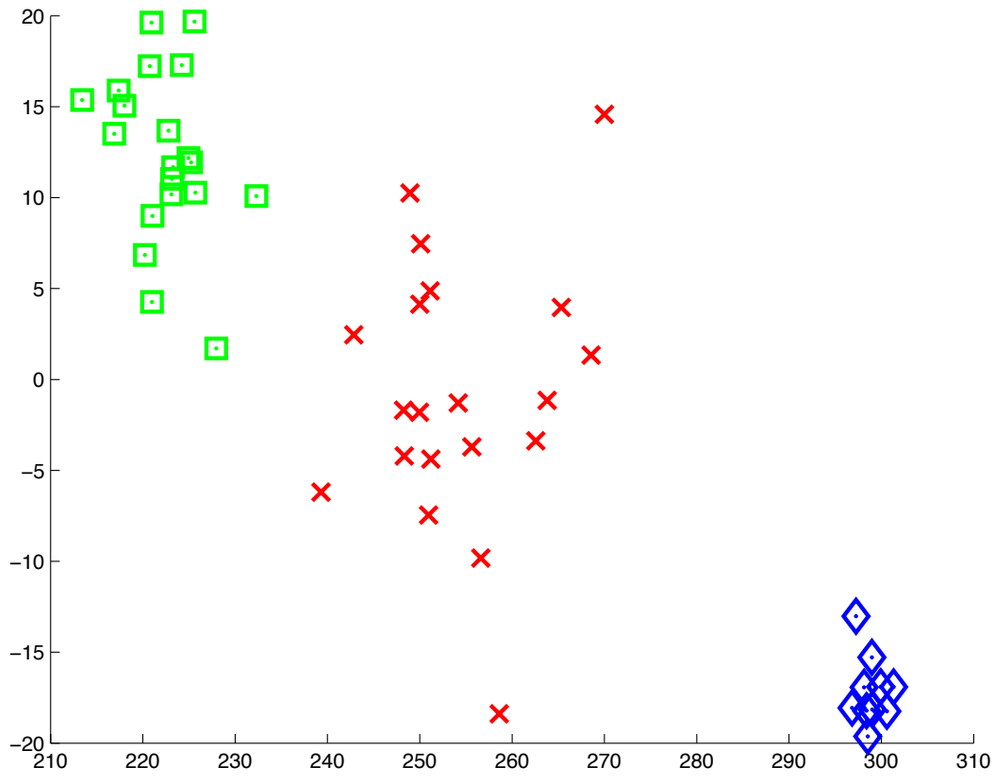
Factors and dimensionality reduction

- Let $\boldsymbol{X}$ be n-by-m, $\boldsymbol{A}$ be n-by-k, $\boldsymbol{B}$ be k-by-m, and $\boldsymbol{X} \approx \boldsymbol{AB}$
  - Rows of $\boldsymbol{A}$ are k-dimensional representations of rows of $\boldsymbol{X}$
  - Columns of $\boldsymbol{B}$ are k-dimensional representations of columns of $\boldsymbol{X}$
  - We can project rows of $\boldsymbol{X}$ to k-dimensional subspace $\boldsymbol{XB}^T$
    - Columns of $\boldsymbol{X}$ are projected with $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{X}$

- Low-dimensional views allow
  - Direct study of factors
    - By hand, plotting, etc.
  - Avoidance of *curse of dimensionality* (more on this later)
  - Better scalability / avoidance of noise

Example

- 10-dimensional data
- Clustered using *k*-means in 3 clusters
- Want to visualize the clusters
  - Are they ”natural”?
- Project the data to first two principal components:



IX.2 Matrix factorization methods

1. Eigendecomposition
2. Singular value decomposition (SVD)
3. Principal component analysis (PCA)
4. Non-negative matrix factorization
5. Other matrix factorization methods

   5.1. CX matrix factorization

   5.2. Boolean matrix factorization

   5.3. Regularizers

   5.4. Matrix completion

Eigendecomposition

- If $\mathbf{X}$ is an n-by-n matrix and $\boldsymbol{v}$ is a vector such that $\boldsymbol{X}\boldsymbol{v} = \lambda\boldsymbol{v}$ for some scalar λ, then
  - λ is an **eigenvalue** of $\boldsymbol{X}$
  - $\boldsymbol{v}$ is an **eigenvector** of $\boldsymbol{X}$ associated to λ
- Matrix $\boldsymbol{X}$ has to *diagonalizable*
  - $\boldsymbol{P}\boldsymbol{X}\boldsymbol{P}^{-1}$ is a diagonal matrix for some invertible matrix $\boldsymbol{P}$
- Matrix $\boldsymbol{X}$ has to have n linearly independent eigenvectors
- The **eigendecomposition** of $\boldsymbol{X}$ is $\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{-1}$
  - Columns of $\boldsymbol{Q}$ are the eigenvectors of $\boldsymbol{X}$
  - $\boldsymbol{\Lambda}$ is a diagonal matrix with eigenvalues in the diagonal

Some useful facts

- Not all matrices have eigendecomposition
  - Not all invertible matrices have eigendecomposition
  - Not all matrices that have eigendecomposition are invertible
  - If $\boldsymbol{X}$ is invertible and has eigendecomposition, then $\boldsymbol{X}^{-1} = \boldsymbol{Q}\boldsymbol{\Lambda}^{-1}\boldsymbol{Q}^{-1}$
- If $\boldsymbol{X}$ is symmetric and invertible (and real), then $\boldsymbol{X}$ has eigendecomposition $\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{T}$

How to find eigendecomposition, part 1

- Recall the *power method* for computing the stationary distribution of a Markov chain
  - $\boldsymbol{v}_{t+1} = \boldsymbol{v}_t\boldsymbol{P}$
  - Computes the *dominant* eigenvalue and eigenvector
    - Can't be used to find the full eigendecomposition
- Similar iterative idea is usually used:
  - Let $\boldsymbol{X}_0 = \boldsymbol{X}$ and find orthogonal $\boldsymbol{Q}_t$ such that $\boldsymbol{X}_t = \boldsymbol{Q}_t^T\boldsymbol{X}_{t-1}\boldsymbol{Q}_t$ is "more diagonal" than $\boldsymbol{X}_{t-1}$
  - When $\boldsymbol{X}_t$ is diagonal enough, set $\boldsymbol{\Lambda} = \boldsymbol{X}_t$ and $\boldsymbol{Q} = \boldsymbol{Q}_t\boldsymbol{Q}_{t-1}\boldsymbol{Q}_{t-2}\cdots\boldsymbol{Q}_1$

The Jacobi method for symmetric matrix

- We assume that $\boldsymbol{X}$ is symmetric n-by-n
- The idea is to reduce the quantity

$$\mathrm{off}(\mathbf{X}) = \sqrt{\sum_{i,j:i\neq j} x_{ij}^2}$$

- The *Jacobi rotations* are matrices of form

$$J(p,q,\theta) = \begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{pmatrix} \begin{matrix} \\ \\ p \\ \\ q \\ \\ \\ \end{matrix} \qquad c = \cos(\theta)$$

$$\begin{matrix} & & p & & q & & \end{matrix} \qquad s = \sin(\theta)$$

Basic Jacobi step

1. Choose index pair (p,q) s.t. $1 \le p < q \le n$
2. Compute $c = \cos(\theta)$ and $s = \sin(\theta)$ s.t.

$$\begin{pmatrix} y_{pp} & y_{pq} \\ y_{qp} & y_{qq} \end{pmatrix} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}^{T} \begin{pmatrix} x_{pp} & x_{pq} \\ x_{qp} & x_{qq} \end{pmatrix} \begin{pmatrix} c & s \\ -s & c \end{pmatrix}$$

   is diagonal ($y_{pq} = y_{qp} = 0$)

3. Overwrite $\boldsymbol{X}$ with $\boldsymbol{Y} = \boldsymbol{J}^T\boldsymbol{X}\boldsymbol{J}$ where $\boldsymbol{J} = J(p,q,\theta)$.

Each Jacobi step reduces off-diagonal values of the 2-by-2 matrix by $\mathrm{off}(\mathbf{Y})^2 = \mathrm{off}(\mathbf{X})^2 - 2x_{pq}^2$

Basic Jacobi step

1. Choose index pair (p,q) s.t. $1 \leq$

Symmetric 2-by-2 eigendecomposition

2. Compute c = cos(θ) and s = sin(θ) s.t.

$$\begin{pmatrix} y_{pp} & y_{pq} \\ y_{qp} & y_{qq} \end{pmatrix} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}^{T} \begin{pmatrix} x_{pp} & x_{pq} \\ x_{qp} & x_{qq} \end{pmatrix} \begin{pmatrix} c & s \\ -s & c \end{pmatrix}$$

is diagonal ($y_{pq} = y_{qp} = 0$)

3. Overwrite $\boldsymbol{X}$ with $\boldsymbol{Y} = \boldsymbol{J}^T\boldsymbol{X}\boldsymbol{J}$ where $\boldsymbol{J} = J(p,q,\theta)$.

Each Jacobi step reduces off-diagonal values of the 2-by-2 matrix by $\text{off}(\mathbf{Y})^2 = \text{off}(\mathbf{X})^2 - 2x_{pq}^2$

Basic Jacobi step

1. Choose index pair (p,q) s.t. $1 \le p < q \le n$
2. Compute $c = \cos(\theta)$ and $s = \sin(\theta)$ s.t.

$$\begin{pmatrix} y_{pp} & y_{pq} \\ y_{qp} & y_{qq} \end{pmatrix} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}^{T} \begin{pmatrix} x_{pp} & x_{pq} \\ x_{qp} & x_{qq} \end{pmatrix} \begin{pmatrix} c & s \\ -s & c \end{pmatrix}$$

   is diagonal ($y_{pq} = y_{qp} = 0$)

3. Overwrite $\boldsymbol{X}$ with $\boldsymbol{Y} = \boldsymbol{J}^T\boldsymbol{X}\boldsymbol{J}$ where $\boldsymbol{J} = J(p,q,\theta)$.

Each Jacobi step reduces off-diagonal values of the 2-by-2 matrix by $\mathrm{off}(\mathbf{Y})^2 = \mathrm{off}(\mathbf{X})^2 - 2x_{pq}^2$

How to select c and s

- We want to have $c = \cos(\theta)$ and $s = \sin(\theta)$ s.t. $0 = y_{pq} = x_{pq}(c^2 - s^2) + (x_{pp} - x_{qq})cs$
- If $x_{pq} = 0$, set c = 1 and s = 0
- Else set $\tau = (x_{qq} - x_{pp})/(2x_{pq})$
- If $\tau \geq 0$, set $t = 1/(\tau + \sqrt{(1 + \tau^2)})$
  - Else set $t = -1/(-\tau + \sqrt{(1 + \tau^2)})$
- Set $c = 1/\sqrt{(1 + t^2)}$ and $s = tc$

How to select p and q

- In *Classical Jacobi* select (p,q) such that
  $|x_{pq}| = \max_{i \neq j} |x_{ij}|$
  - Finding this value takes $O(n^2)$ time
- In *Cyclic Jacobi* go thru the off-diagonal elements in a fixed order
  - E.g. $(p,q) = (1,2), (1,3), (1,4), (2,3), (2,4), (3,4), (1,2), \ldots$

Jacobi in nutshell

1. Set $\boldsymbol{V} = \boldsymbol{I}_n$; $eps = tol \times ||\boldsymbol{X}||_F$; $\boldsymbol{Y} = \boldsymbol{X}$
2. **while** off($\boldsymbol{Y}$) > eps
   - 2.1. Choose (p,q) so $|x_{pq}| = \max_{i \neq j} |x_{ij}|$ (or use cyclic order)
   - 2.2. Compute cosine–sin pair (c,s)
   - 2.3. $\boldsymbol{Y} = J(p,q,\theta)^T \boldsymbol{Y} J(p,q,\theta)$
   - 2.4. $\boldsymbol{V} = \boldsymbol{V} J(p,q,\theta)$
3. **end while**
4. **return** $\boldsymbol{\Lambda} = \boldsymbol{Y}$ and $\boldsymbol{Q} = \boldsymbol{V}$ ($\boldsymbol{X} \approx \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$)

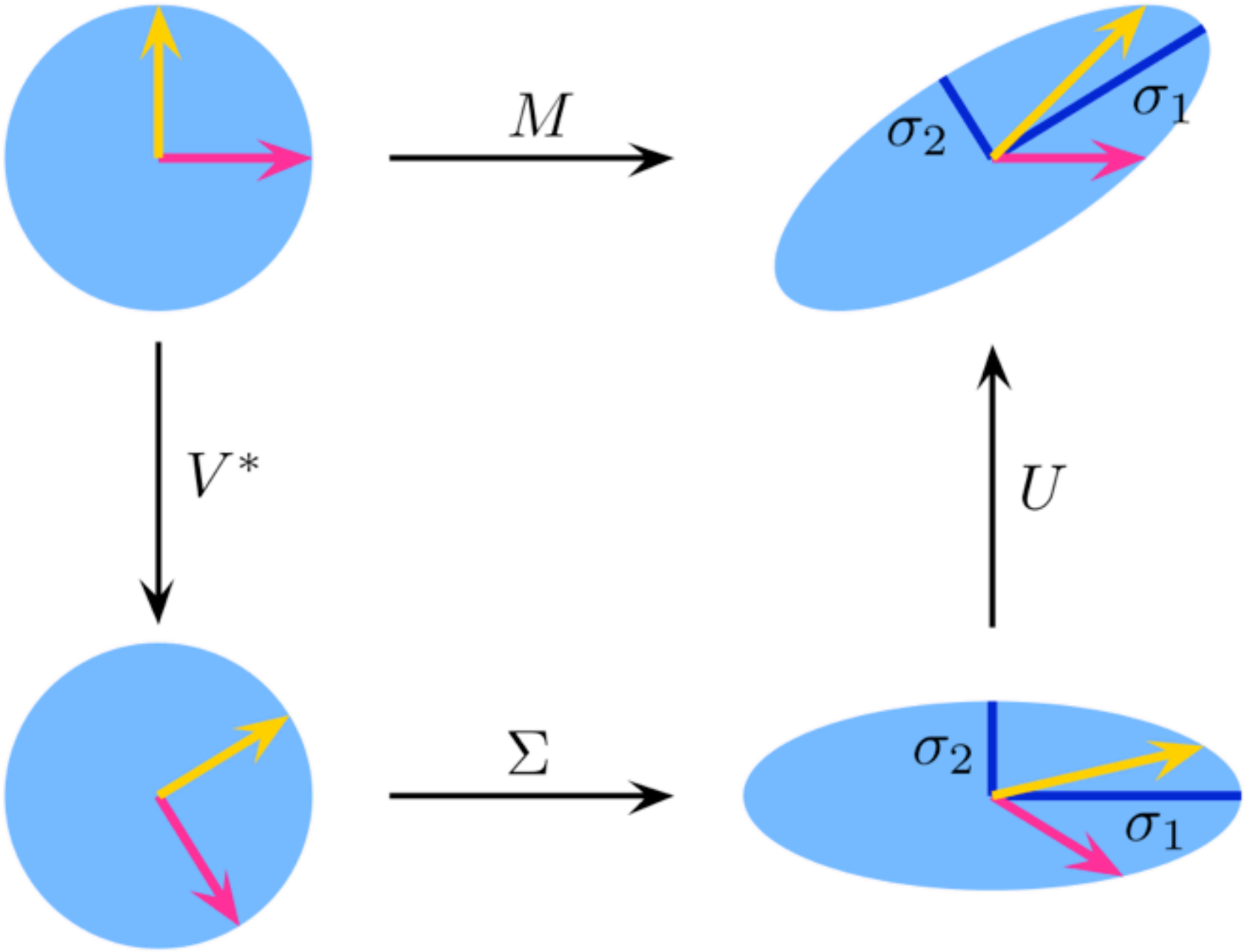
Some notes

- The quality (and running time) depends on parameter $tol > 0$
- Jacobi method is easy to parallellize
  - Split the update in non-conflicting steps
- Other methods exist
  - Symmetric QR algorithm
  - Tri-diagonal methods
    - Bisecting algorithm
    - Divide-and-conquer
- Numerical stability is an issue with all these methods

Singular value decomposition (SVD)

- Not every matrix has eigendecomposition, but:
  Theorem. If $\boldsymbol{X}$ is n-by-m real matrix, there exists n-by-n orthogonal matrix $\boldsymbol{U}$ and m-by-m orthogonal matrix $\boldsymbol{V}$ such that $\boldsymbol{U}^T\boldsymbol{X}\boldsymbol{V}$ is n-by-m matrix $\boldsymbol{\Sigma}$ with values $\sigma_1, \sigma_2, \ldots, \sigma_{\min(n,m)}$, $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{\min(n,m)} \geq 0$, in its diagonal.
  - In other words, $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$
  - Values σ_i are the **singular values** of $\boldsymbol{X}$
  - Columns of $\boldsymbol{U}$ are the **left singular vectors** and columns of $\boldsymbol{V}$ the **right singular vectors** of $\boldsymbol{X}$

Example



$$M = U \cdot \Sigma \cdot V^*$$

Properties of SVD, part 1

- rank($\boldsymbol{X}$) = r iff $\boldsymbol{X}$ has exactly r non-zero singular values ($\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > \sigma_{r+1} = \ldots = \sigma_{\min(n,m)} = 0$)
- Vectors $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_r$ are a basis for the column space of $\boldsymbol{X}$
- Vectors $\boldsymbol{u}_{r+1}, \boldsymbol{u}_{r+2}, \ldots, \boldsymbol{u}_n$ are a basis for the left null space of $\boldsymbol{X}$
- Vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_r$ are a basis for the row space of $\boldsymbol{X}$
- Vectors $\boldsymbol{v}_{r+1}, \boldsymbol{v}_{r+2}, \ldots, \boldsymbol{v}_m$ are a basis for the null space of $\boldsymbol{X}$

Properties of SVD, part 2

- If $\boldsymbol{X}$ is rank-r, then $\mathbf{X} = \sum_{i=1}^{r} \sigma_i u_i v_i^T$
  - $\boldsymbol{X}$ is a sum of r rank-1 matrices scaled with singular values
- $\|\mathbf{X}\|_F^2 = \sigma_1^2 + \sigma_2^2 + \cdots + \sigma_{\min(n,m)}^2$
- $\|\mathbf{X}\|_2 = \sigma_1$
- **Eckart–Young theorem.** Let $\boldsymbol{X}$ be of rank-r and let $\boldsymbol{U\Sigma V}^T$ be its SVD. Denote by $\boldsymbol{U}_k$ the first k columns of $\boldsymbol{U}$, by $\boldsymbol{V}_k$ the first k columns of $\boldsymbol{V}$ and by $\boldsymbol{\Sigma}_k$ the upper-left k-by-k corner of $\boldsymbol{\Sigma}$. Then $\boldsymbol{X}_k = \boldsymbol{U}_k\boldsymbol{\Sigma}_k\boldsymbol{V}_k^T$ is the best rank-k approximation of $\boldsymbol{X}$ in the sense that $\|\mathbf{X} - \mathbf{X}_k\|_F \leqslant \|\mathbf{X} - \mathbf{Y}\|_F$ and $\|\mathbf{X} - \mathbf{X}_k\|_2 \leqslant \|\mathbf{X} - \mathbf{Y}\|_2$ for any rank-k matrix $\boldsymbol{Y}$.

SVD and pseudo-inverse

- Recall that if $\boldsymbol{X}$ is n-by-m with rank($\boldsymbol{X}$) = $m \leq n$, the *pseudo-inverse* of $\boldsymbol{X}$ is $\boldsymbol{X}^{\dagger} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T$
- If rank($\boldsymbol{X}$) = r and $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$, then we can define $\boldsymbol{X}^{\dagger} = \boldsymbol{V}\boldsymbol{\Sigma}^{\dagger}\boldsymbol{U}^T$
  - $\boldsymbol{\Sigma}^{\dagger}$ is a diagonal matrix with $1/\sigma_i$ in its ith position
  - More general than the above definition
- This gives the least-squares solution to the following problem: given $\boldsymbol{A}$ and $\boldsymbol{X}$, find $\boldsymbol{Y}$ s.t. $||\boldsymbol{A} - \boldsymbol{X}\boldsymbol{Y}||_F^2$ is minimized
  - Setting $\boldsymbol{Y} = \boldsymbol{X}^{\dagger}\boldsymbol{A}$ minimizes the squared Frobenius

SVD and eigendecomposition

- Let $\boldsymbol{X}$ be n-by-m and $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ its SVD
- Recall that the *Gram matrix* of the columns of $\boldsymbol{X}$ is $\boldsymbol{X}^T\boldsymbol{X}$
  - For the rows it is $\boldsymbol{X}\boldsymbol{X}^T$
- Now $\boldsymbol{X}^T\boldsymbol{X} = (\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T)^T(\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T) = \boldsymbol{V}\boldsymbol{\Sigma}^T\boldsymbol{U}^T\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$
  $= \boldsymbol{V}\boldsymbol{\Sigma}^T\boldsymbol{\Sigma}\boldsymbol{V}^T = \boldsymbol{V}\boldsymbol{\Sigma}_m^2\boldsymbol{V}^T$
  - $\boldsymbol{\Sigma}_m^2$ is an m-by-m diagonal matrix with σ_i^2 in its ith position
- Similarly $\boldsymbol{X}\boldsymbol{X}^T = \boldsymbol{U}\boldsymbol{\Sigma}_n^2\boldsymbol{U}^T$
- Therefore
  - Columns of $\boldsymbol{U}$ are the eigenvectors of $\boldsymbol{X}\boldsymbol{X}^T$
  - Columns of $\boldsymbol{V}$ are the eigenvectors of $\boldsymbol{X}^T\boldsymbol{X}$
  - Singular values are square roots of the associated eigenvalues

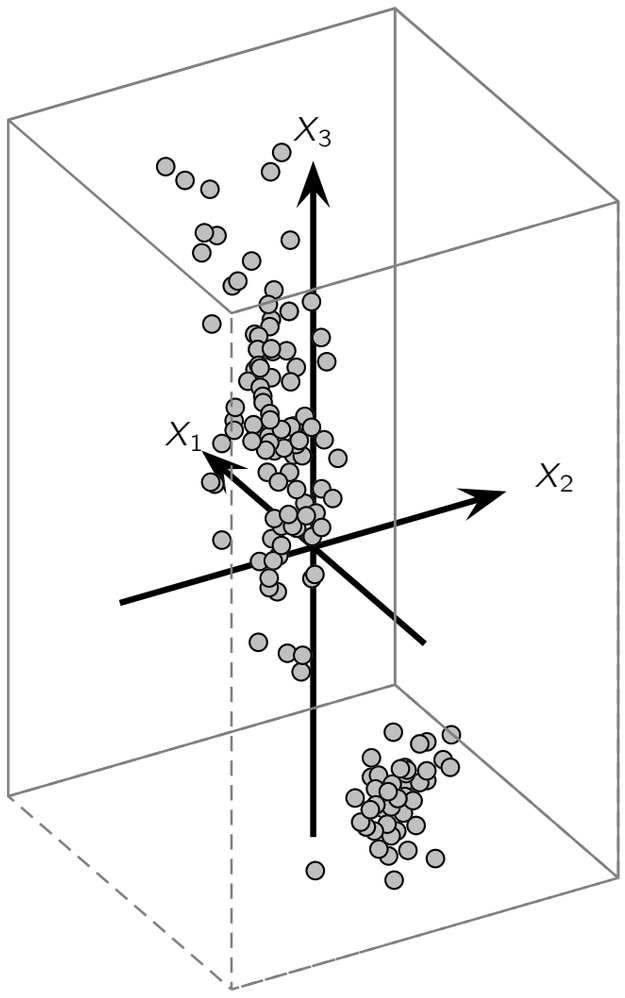
Computing the SVD

- Simple idea: Compute the eigendecompositions of $\boldsymbol{XX}^T$ and $\boldsymbol{X}^T\boldsymbol{X}$
  - Bad for numerical stability
- We can adapt the Jacobi method:
  - At each step find a Jacobi rotation $J(p,q,\theta)$ such that columns p and q of $\boldsymbol{X}J(p,q,\theta)$ are orthogonal
    - Corresponds to zeroing (p,q) and (q,p) in $\boldsymbol{X}^T\boldsymbol{X}$
    - The product of this sequence of Jacobi rotations gives orthogonal $\boldsymbol{V}$
    - Rest follows by $\boldsymbol{AV} = \boldsymbol{U\Sigma}$
  - This is called one-sided Jacobi

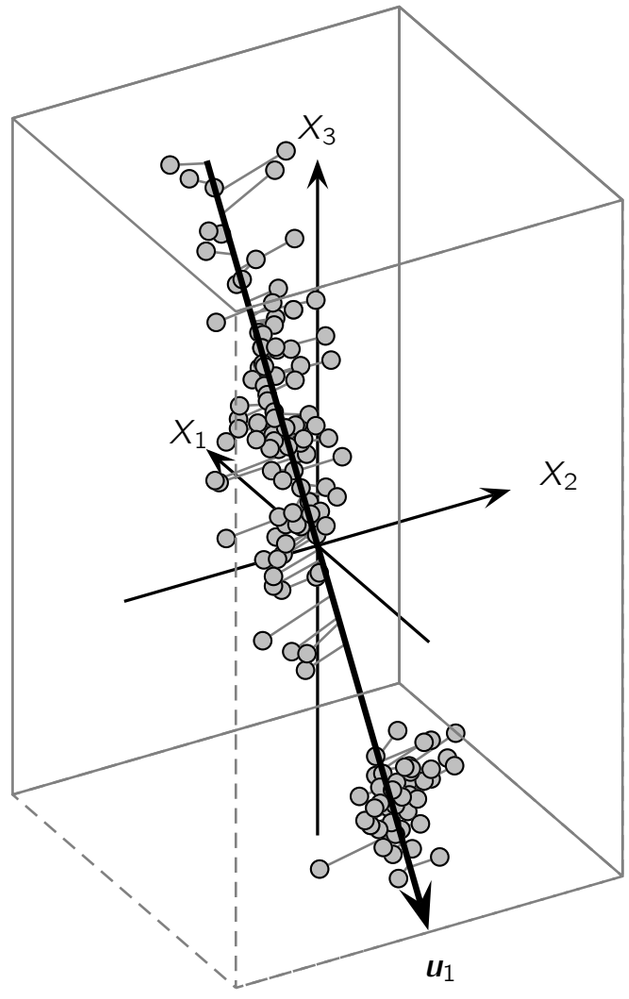
Principal component analysis (PCA)

- Let rows of matrix denote observations and columns denote variables
- In **principal component analysis** (PCA) we want to find new variables (dimensions) that capture the variance of the data
  - First variable has as much variance as possible
  - Second variable is orthogonal to the first and captures as much as possible of the remaining variance
  - Third variable …

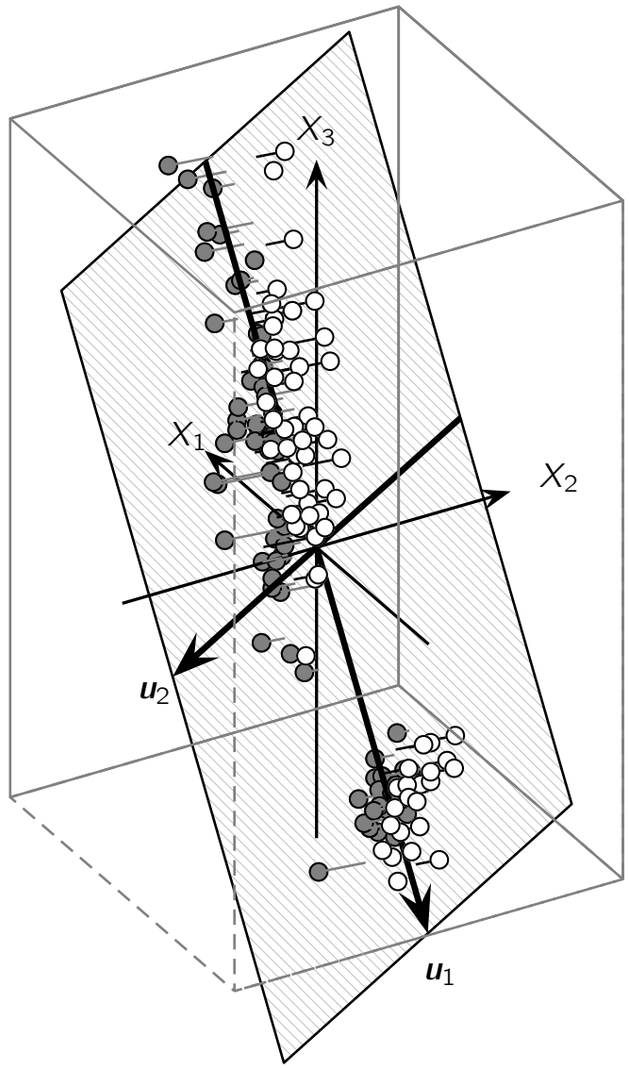
Example



Example



Example



Computing the PCA

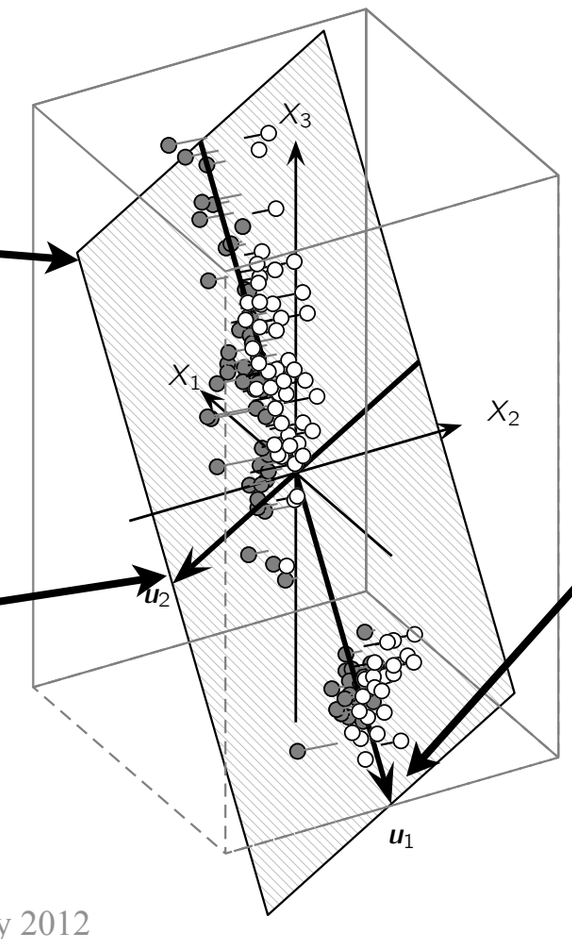
- First, data is centered
  - The mean of each column is subtracted from the column
- Then, the m-by-m covariance matrix $\boldsymbol{S}$ is computed
  - s_{ij} is the covariance between ith and jth column (variable)
  - For centered data $\boldsymbol{X}$, $\boldsymbol{S} = 1/n\ \boldsymbol{X}^T\boldsymbol{X}$
- The first principal vector is given by the eigenvector of $\boldsymbol{S}$ associated with the highest eigenvalue λ_1
  - $\lambda 1$ gives the amount of variance explained
- The second principal vector is given by the second eigenvector, etc.
- The total variance of the data is $\lambda_1 + \lambda_2 + \ldots + \lambda_m$

PCA and SVD

- Alternatively, we can just compute the SVD of centered data ***X***'
  - Now the principal vectors are columns of ***V***
  - Therefore, PCA is *SVD done with centered data*
- We can project the data ***X***' into its principal space by ***X'V***

Subspace spanned by the principal vectors

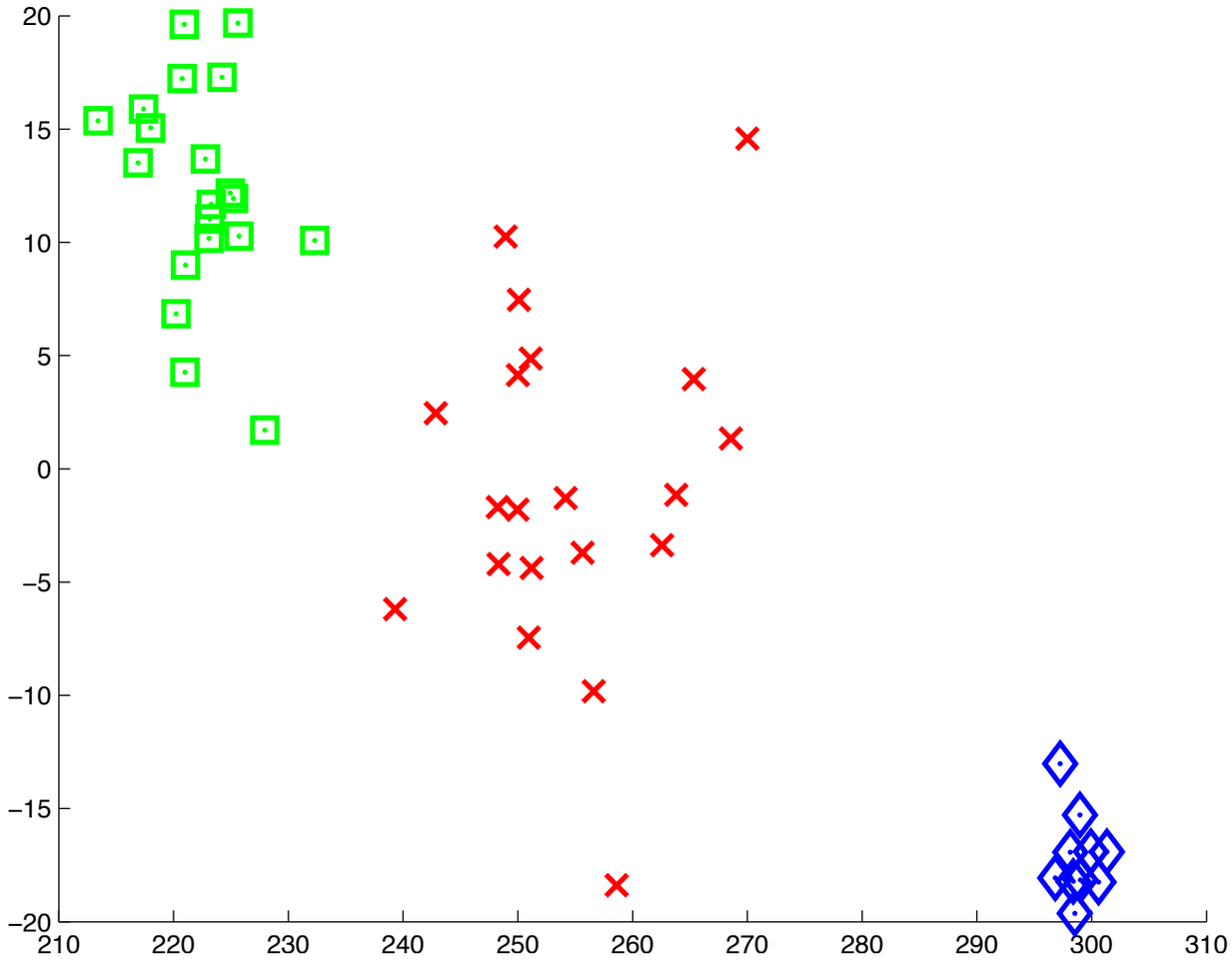
2nd principal vector



1st principal vector

How many principal vectors?

- Rule of thumb: keep 90% of variance
  - Select k s.t. $(\lambda_1 + \lambda_2 + \ldots + \lambda_k)/(\lambda_1 + \lambda_2 + \ldots + \lambda_m) \geq 0.9$
  - Same as $(\sigma_1^2 + \sigma_2^2 + \ldots + \sigma_k^2)/(\sigma_1^2 + \sigma_2^2 + \ldots + \sigma_m^2) \geq 0.9$
- But if you want to do plotting, you need less…



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. $||\boldsymbol{X} - \boldsymbol{WH}||_F^2$ 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. $||\boldsymbol{X} - \boldsymbol{WH}||_F^2$ is minimized
    - Fix $\boldsymbol{H}$ and find $\boldsymbol{W}$ s.t. $||\boldsymbol{X} - \boldsymbol{WH}||_F^2$ 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 $\boldsymbol{W}$