Machine Learning

Lecture 10: Dimensionality Reduction & Matrix Factorization

Prof. Dr. Stephan Günnemann

Data Mining and Analytics
Technical University of Munich

08.01.2019 & 14.01.2019

Reading material

Reading material

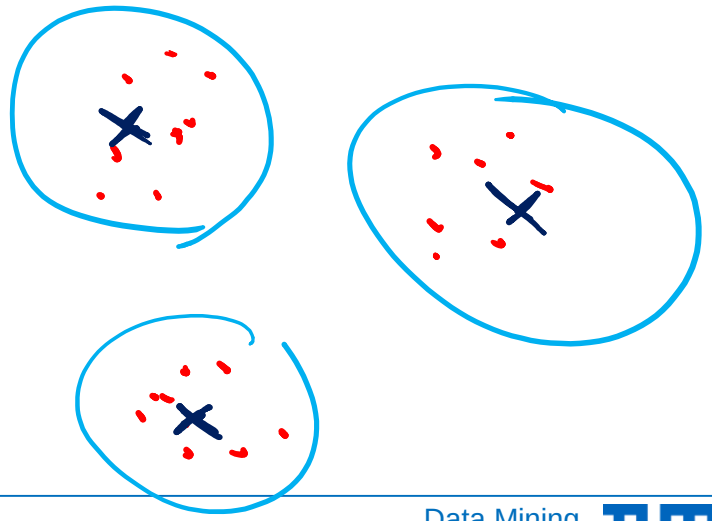
- Bishop, chapters: 12.1, 12.2.1
- Leskovec, Rajaraman, Ullman - Mining of Massive Datasets: chapter 11
- Goodfellow - Deep Learning: chapter 14

Roadmap

- Chapter: Dimensionality Reduction & Matrix Factorization
  1. **Introduction**
  2. Principal Component Analysis (PCA)
  3. Probabilistic PCA (PPCA)
  4. Singular Value Decomposition (SVD)
  5. Matrix Factorization
  6. Autoencoders (Non-linear Dimensionality Reduction)

Introduction: Unsupervised Learning (I)

- Supervised learning aims to map inputs to targets with $y = f(x)$, or in a probabilistic framework it models $p(y|x)$
- Unsupervised learning can be seen as modelling $p(x)$
- We are trying to find the (hidden / latent) structure in the data
  - e.g. find a latent distribution $p(z)$ and a generative transformation $p(x|z)$ we can then obtain $p(x) = \int p(x \mid z)\, p(z)\, dz$
  - latent z usually unknown and has to be estimated
- Example: Clustering
  - the cluster label is the latent state



Introduction: Unsupervised Learning (II)

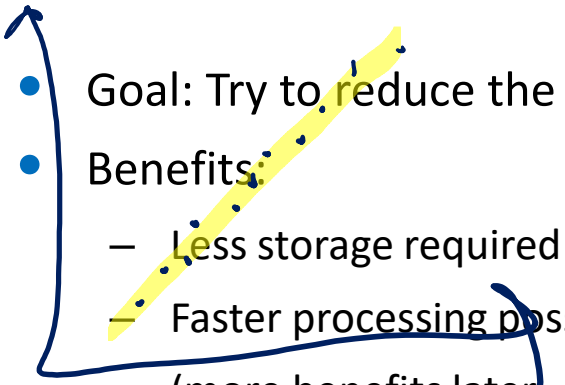
- Unsupervised learning can be viewed as compression
  - compress a data point to a single label corresponding to it's cluster
  - compress a data point from a higher dim. to a lower dim. latent space

- Unsupervised learning can be used as…
  - … a stand-alone method (e.g. to understand your data, visualization)
  - … as a pre-processing step (e.g. use cluster label as feature for subsequent classification task; obtain small number of relevant features)

- This lecture: Dimensionality Reduction (& Matrix Factorization)

Dimensionality Reduction: Motivation

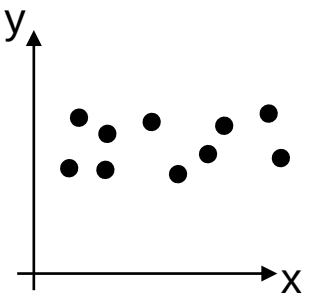
- Often data has very many features, i.e. high dimensional data
- High dimensional data is challenging:
  - Similarity search/computation is expensive because of high complexity of distance functions
  - Highly correlated dimension could cause trouble for some algorithms
  - Curse of dimensionality: we need exponential amounts of data to characterize the density as the dimensionality goes up
  - It is hard to visualize high-dimensional data
- Often the data lies in a low dimensional manifold, embedded in a high-D space



- Goal: Try to reduce the dimensionality while avoiding high information loss
- Benefits:
  - Less storage required
  - Faster processing possible
  - (more benefits later….)

Feature (Sub-)Selection

- Choose "good" dimensions using a-priori knowledge or appropriate heuristics
  - e.g. remove **low-variance** dimensions
  - Depending on the application only a few dimensions might be of interest
    - Example: shoe size interesting for shoe purchases, not so for car purchases

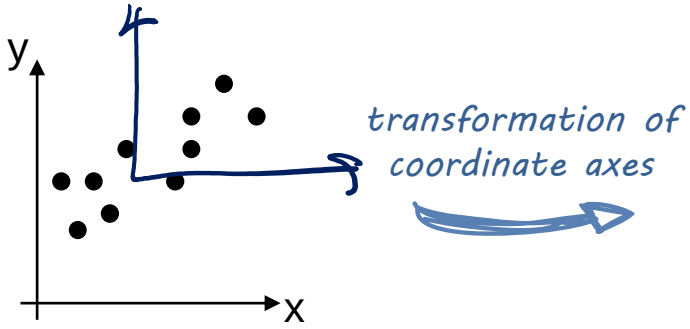


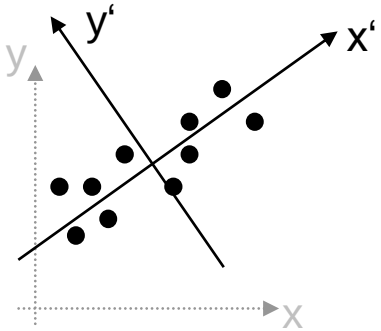
- Advantage:
  - No need for an intensive preprocessing or training phase to determine relevant dimensions
- Disadvantage:
  - Expert knowledge required; misjudgment possible

Beyond Feature (Sub-)Selection

- Can we do
  - better?
  - more intelligent?
  - automatic?

- Obviously: Simply discarding whole features not a good idea
  - Features are often correlated





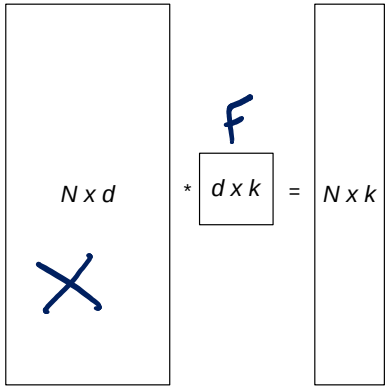
The y' dimension is barely necessary to describe the data in this new coordinate system → The y'-dimension can be ignored

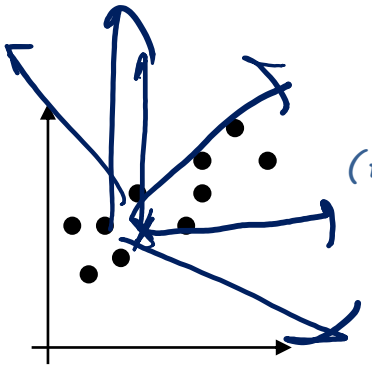
Dim. Reduction via Linear Transformations

- Use linear transformations to represent data in a different coordinate system
  - **change of the basis** (orthogonal basis transformations) **+ potentially discarding dimensions**
- Technical:
  - use orthonormal transformation matrix $F \in R^{d \times k}$
  - $(x')^T = x^T \cdot F$ is the transformation of (column) vector x into the new coordinate system defined by F
  - $X' = X \cdot F$ is the matrix containing all the transformed points x_i'

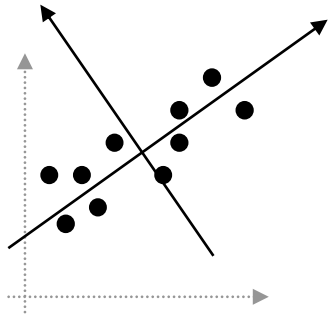
INPUT DIMENSIONALITY

OUTPUT DIM





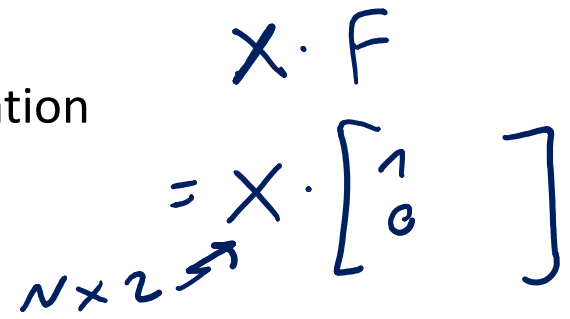
(first centered to the mean, then) transformation with F



IF $k < d$ → DIM. REDUCTION

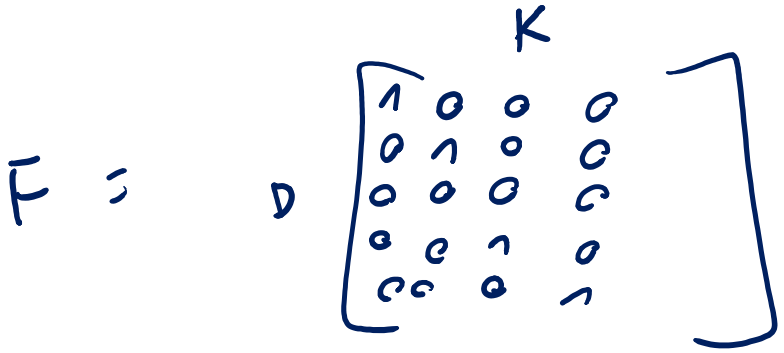
Discussion: Linear Transformations

- Feature selection is a linear transformation
  - What is the matrix F?



- Let $\bar{x}$ be the mean vector (here: row vector) in the original data space, the mean vector in the transformed space is given by $\bar{x'} = \bar{x} \cdot F$
  - Proof?

- Let Σ_X be the covariance matrix in the original data space, the covariance matrix in the transformed space is then $\Sigma_{X'} = F^T \cdot \Sigma_X \cdot F$
  - Proof?

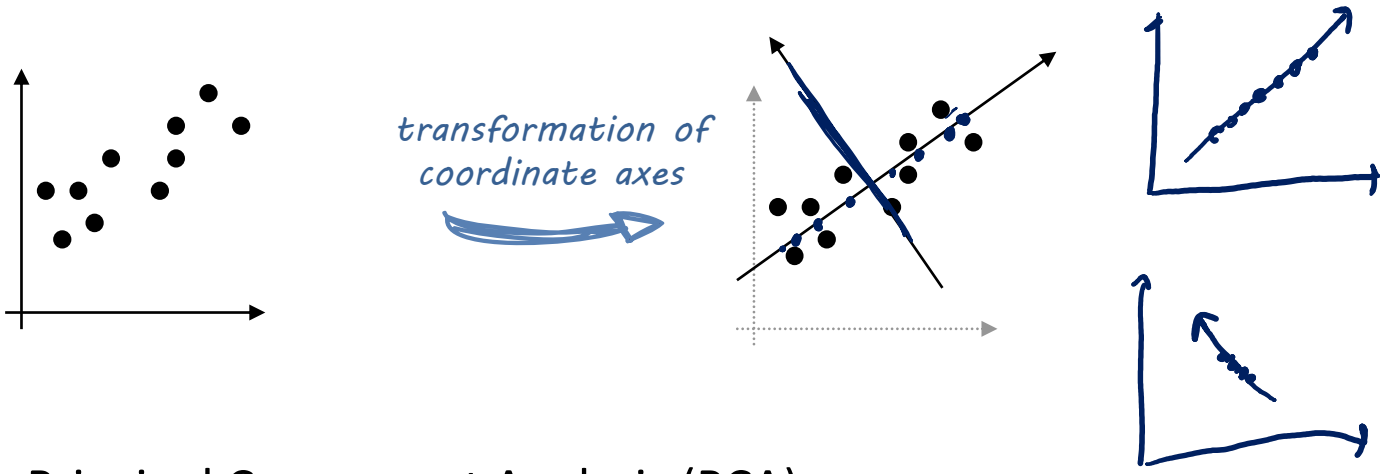


Roadmap

- Chapter: Dimensionality Reduction & Matrix Factorization
  1. Introduction
  2. **Principal Component Analysis (PCA)**
  3. Probabilistic PCA (PPCA)
  4. Singular Value Decomposition (SVD)
  5. Matrix Factorization
  6. Autoencoders (Non-linear Dimensionality Reduction)

Principal Component Analysis: Motivation

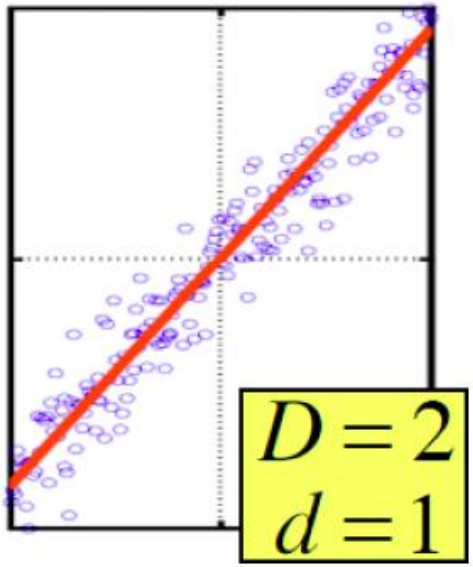
- Question: Which transformation matrix F to use?
  - Is there an **optimal orthogonal transformation** (depending on the data)?
  - Optimality: Approximate the data with few coefficients as good as possible

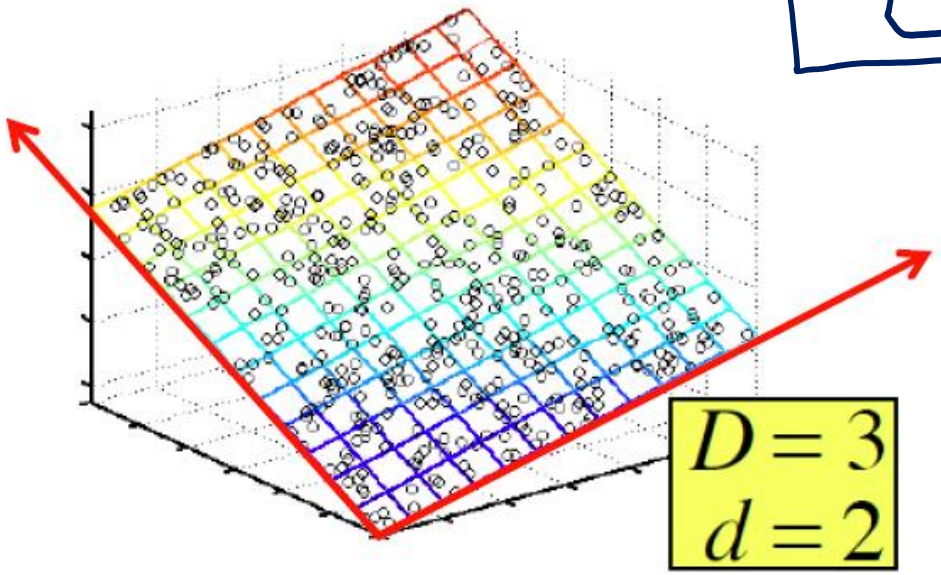


- Approach: Principal Component Analysis (PCA)
  - Find a coordinate system in which the (possibly originally correlated) points are **linearly uncorrelated**
  - The dimensions with no or low variance can then be ignored

Principal Component Analysis: Motivation

- Often the data lies on (or near) a **low dimensional** subspace
- Illustration:

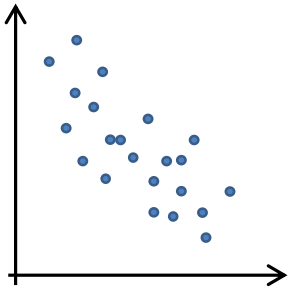




- We effectively need only d dimensions instead of D to describe the data!

Determine the Principal Components

- Goal:
  - Transform the data, such that the **covariance between the new dimensions is 0**
  - The transformed data points are not linearly correlated any more



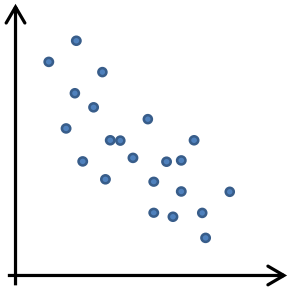
- Given: N d-dimensional data points: : $\{x_i\}_{i=1}^{N}$, $x_i \in \mathbb{R}^d \; \forall i \in \{1, \dots, N\}$
- We represent this set of points by a matrix $\boldsymbol{X} \in \mathbb{R}^{N \times d}$:

$$\boldsymbol{X} = \begin{bmatrix} x_{11} & \cdots & x_{1d} \\ \vdots & \ddots & \vdots \\ x_{N1} & \cdots & x_{Nd} \end{bmatrix}$$

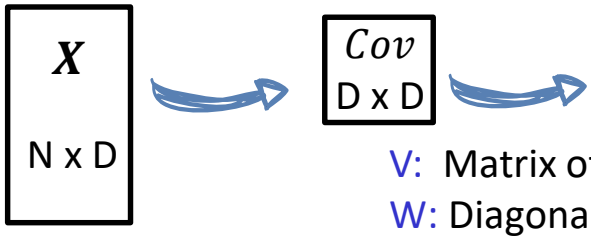
  - The row $x_i = \{x_{i1}, \dots, x_{id}\} \in \mathbb{R}^d$ denotes the i-th point and the column X_j denotes the vector containing all values from the j-th dimension

Determine the Principal Components

- Goal:
  - Transform the data, such that the **covariance between the new dimensions is 0**
  - The transformed data points are not linearly correlated any more



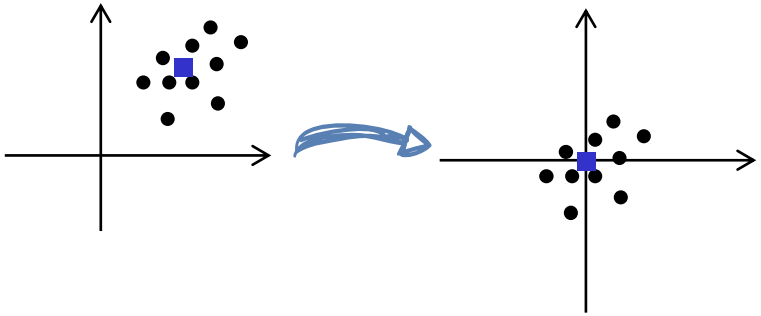
- General approach
  1. Center the data
  2. Compute the covariance matrix
  3. Use the Eigenvector decomposition to transform the coordinate system



V: Matrix of eigenvectors
W: Diagonalmatrix of eigenvalues

Determine the Principal Components

- Given: $X \in \mathbb{R}^{N \times d}$: $X = \begin{bmatrix} x_{11} & \cdots & x_{1d} \\ \vdots & \ddots & \vdots \\ x_{N1} & \cdots & x_{Nd} \end{bmatrix}$



- Shift the points in their mean $\bar{x} \in \mathbb{R}^d$ (centralized data): $\tilde{x}_i = x_i - \bar{x}$

Statistics:

Zero order statistic : number of points N

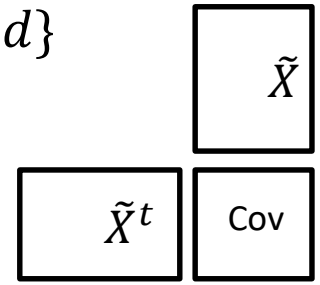
First order statistic: the mean of the N points, the vector $\bar{x} \in \mathbb{R}^d$:

$$\bar{x} = \begin{bmatrix} \bar{x}_1 \\ \vdots \\ \bar{x}_d \end{bmatrix} = \frac{1}{N} \cdot X^T \cdot 1_N$$

where 1_N is an N-dimensional vector of ones

Determine the Principal Components

- Determine the variances $Var(\tilde{X}_j)$ for each dimension $j \in \{1, \dots d\}$
- Determine the covariance $cov(\tilde{X}_{j_1}, \tilde{X}_{j_2})$ between dimensions j_1 and j_2, $\forall\, j_1 \neq j_2 \in \{1, \dots d\}$
- ➢ Leads to the covariance matrix $\Sigma_{\tilde{X}} \in \mathbb{R}^{d \times d}$



Statistics:

Second order statistic: variance and covariance

The variance within the j-th dimension in X is:

$$var(X_j) = \frac{1}{N}\sum_{i=1}^{N}(x_{ij} - \bar{x}_j)^2 = \frac{1}{N} \cdot X_j^T X_j - \bar{x}_j^2$$

The covariance between dimension j_1 and j_2 is:

$$cov(X_{j_1}, X_{j_2}) = \frac{1}{N}\sum_{i=1}^{N}(x_{ij_1} - \bar{x}_{j_1}) \cdot (x_{ij_2} - \bar{x}_{j_2}) = \frac{1}{N} \cdot X_{j_1}^T X_{j_2} - \bar{x}_{j_1}\bar{x}_{j_2}$$

Determine the Principal Components

Statistics (continued):

For the set of points contained in X the corresponding covariance matrix is defined as:

$$\Sigma_X = \begin{bmatrix} var(X_1) & cov(X_1, X_2) & \dots & cov(X_1, X_d) \\ cov(X_2, X_1) & var(X_2) & & \\ \vdots & & \ddots & \vdots \\ cov(X_d, X_1) & \cdots & & var(X_d) \end{bmatrix} = \frac{1}{N} X^{\mathrm{T}} X - \bar{x}\, \bar{x}^T$$

- Remark: covariance matrices are symmetric

Determine the Principal Components

$$Cov' = \begin{pmatrix} Var(1)' & 0 & \cdots & 0 \\ 0 & Var(2)' & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & Var(D)' \end{pmatrix}$$

- Goal of PCA: Transformation of the coordinate system such that the covariances between the new axes are 0
- Approach:
  - Diagonalization by changing the basis (= adapt the coordinate system)
  - According to the spectral theorem, the eigenvectors of a symmetric matrix form an orthogonal basis

➢ Eigendecomposition of the covariance matrix: $\Sigma_{\tilde{X}} = \Gamma \cdot \Lambda \cdot \Gamma^T$

Eigendecomposition (spectral decomposition) is the factorization of $X \in \mathbb{R}^{d \times d}$:

$$X = \Gamma \cdot \Lambda \cdot \Gamma^T$$

→ matrices $\Gamma, \Lambda \in \mathbb{R}^{d \times d}$ with columns of Γ being the normalized eigenvectors γ_i

→Γ is an orthonormal matrix: $\Gamma \cdot \Gamma^T = \Gamma^T \cdot \Gamma = Id$ $(\Gamma^T = \Gamma^{-1})$

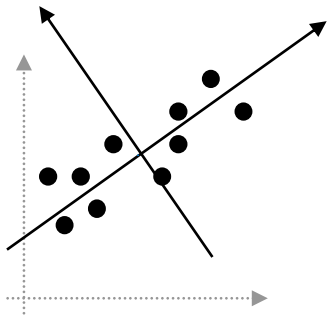
→Λ is a diagonal matrix with eigenvalues λ_i as the diagonal elements

Determine the Principal Components

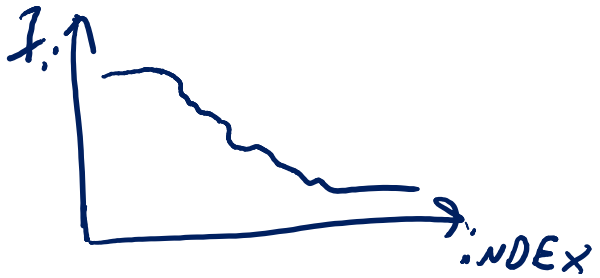
- Eigendecomposition of the covariance matrix: $\Sigma_{\tilde{X}} = \Gamma \cdot \Lambda \cdot \Gamma^T$

$$\Sigma_Y = \Gamma^T \cdot \Sigma_{\tilde{X}} \cdot \Gamma = \underbrace{\Gamma^T \cdot \Gamma}_{I_D} \cdot \Lambda \cdot \underbrace{\Gamma^T \cdot \Gamma}_{I_D} = \Lambda = \begin{bmatrix} \lambda_1 & & 0 \\ & \lambda_2 & \\ 0 & & \ddots \\ & & & \lambda_D \end{bmatrix}$$

- The **new coordinate system** is defined by the eigenvectors γ_i:
  - Transformed data: $Y = \tilde{X}\,\Gamma$
  - Λ is the covariance matrix in this new coordinate system
  - New system has in dimension i the variance λ_i
  - $\forall i_1 \neq i_2$: $cov(Y_{i_1}, Y_{i_2}) = 0$



Dimensionality Reduction with PCA



- Approach
  - The coordinates with a low variance (hence a low λ_i) can be ignored
  - W.l.o.g. let us assume $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$
- Truncation of Γ
  - **Keep only columns (i.e. eigenvectors) of Γ corresponding to the largest k eigenvalues $\lambda_1, \ldots, \lambda_k$**
  - $Y_{reduced} = \tilde{X} \cdot \Gamma_{truncated}$
- How to pick k?
  - Frequently used: 90% rule; the *k* variances should explain 90% of the energy
  - k = smallest value ensuring $\sum_{i=1}^{k} \lambda_i \geq 0.9 \cdot \sum_{i=1}^{d} \lambda_i$
- The modified points (transformed and truncated) contain most of the information of the original points and are low dimensional

Complexity

- Complexity of PCA:

$$\underbrace{O(N \cdot d^2)}_{\text{Compute covariance matrix}} + \underbrace{O(d^3)}_{\text{Eigenvalue decomposition}} + \underbrace{O(N \cdot d \cdot k)}_{\text{Project data onto the k-dimensional space}} = O(N \cdot d^2 + d^3)$$

- Remarks on eigenvalue decomposition:
  - Usually we are interested in the reduced data only
  - ➢ **Only the *k* largest eigenvectors required** (i.e. not all of them)
  - Use iterative approaches (next slide) for finding eigenvectors
    - Complexity: $O(\#it \cdot d^2)$ // #it = number of iterations
    - For sparse data even faster: $O(\#it \cdot \#nz)$ // #nz = number of nonzero elements in the matrix

How to Compute Eigenvectors?

$\lambda \cdot v = X \cdot v$

- Eigenvalues are important for many machine learning/data mining tasks
  - PCA, Ranking of Websites, Community Detection, …
  - How to compute them efficiently?

// see our other lecture!

$v^{t+1} \leftarrow \frac{X \cdot v^t}{\|X \cdot v^t\|}$

STOP $\|v^{t+1} - v^t\| \leq \epsilon$

- Power iteration (a.k.a. Von Mises iteration)
  - Iterative approach to compute **a single** eigenvector
- Let X be a matrix and v be an arbitrary (normalized) vector
  - Iteratively compute $v \leftarrow \frac{X \cdot v}{\|X \cdot v\|}$ until convergence
    - in each step, v is simply multiplied with X and normalized
  - **v converges to the eigenvector of X with greatest absolute value**
  - Highly efficient for sparse data

v^∞ is top-1 eigenvector

How to Compute Eigenvectors?

- Convergence:
  - Linear convergence with rate $|\lambda_2/\lambda_1|$
  - Fast convergence if first and second eigenvalue are dissimilar

- How to find **multiple (the k largest) eigenvectors**?
  - Let us focus on symmetric matrices X
  - Eigenvalue decomposition leads to: $X = \Gamma \cdot \Lambda \cdot \Gamma^T = \sum_{i=1}^{d} \lambda_i \cdot \gamma_i \cdot \gamma_i^T$
  - Define deflated matrix: $\hat{X} = X - \lambda_1 \cdot \gamma_1 \cdot \gamma_1^T$
    - $\hat{X}$ has the same eigenvectors as X except the first one has become zero
  - ➢ Apply power iteration on $\hat{X}$ to find the second largest eigenvector of X

PCA: Summary

- PCA finds the optimal transformation by deriving uncorrelated dimensions
  - Exploits eigendecomposition

- Dimensionality reduction
  - After transformation simply remove dimensions with lowest variance (or simply use only the k largest eigenvectors for transformation)

Alternative views of PCA

$\max_{u_1} u_1^T S u_1$ s.t. $\|u_1\|^2 = 1$

- Maximum Variance Formulation
  - Goal: Project the data to a lower dimensional space $\mathrm{R}^{\mathrm{k}}, k \ll d$ while maximizing the variance of the projected data
  - e.g. for $k = 1$ define the projection as a d dimensional unit vector u_1
  - the variance of the projected data is $\frac{1}{n}\sum_{i=1}^{N}(u_1^T x_i - u_1^T \bar{x})^2 = u_1^T S u_1$ where S is the sample covariance matrix
  - $Su_1 = \lambda_1 u_1$ is a stationary point of $u_1^T S u_1 + \lambda_1(1 - u_1^T u_1)$
  - the variance is given by $u_1^T S u_1 = \lambda_1$
- Minimum Error Formulation
  - Goal: Find an orthogonal set of k linear basis functions $w_j \in R^d$ and corresponding low-dimensional projections $z_j \in R^k$ such that the average reconstruction error $\mathrm{J} = \frac{1}{N}\sum_i ||x_i - \hat{x}_i||^2$ is minimized with $\hat{x}_i = W z_i + \mu$
  - Solution: W = $\mathrm{U_k}$: the first k eigenvectors of S and $J = \sum_{j=k+1}^{d} \lambda_j$

Roadmap

- Chapter: Dimensionality Reduction & Matrix Factorization
  1. Introduction
  2. Principal Component Analysis (PCA)
  3. **Probabilistic PCA (PPCA)**
  4. Singular Value Decomposition (SVD)
  5. Matrix Factorization
  6. Autoencoders (Non-linear Dimensionality Reduction)

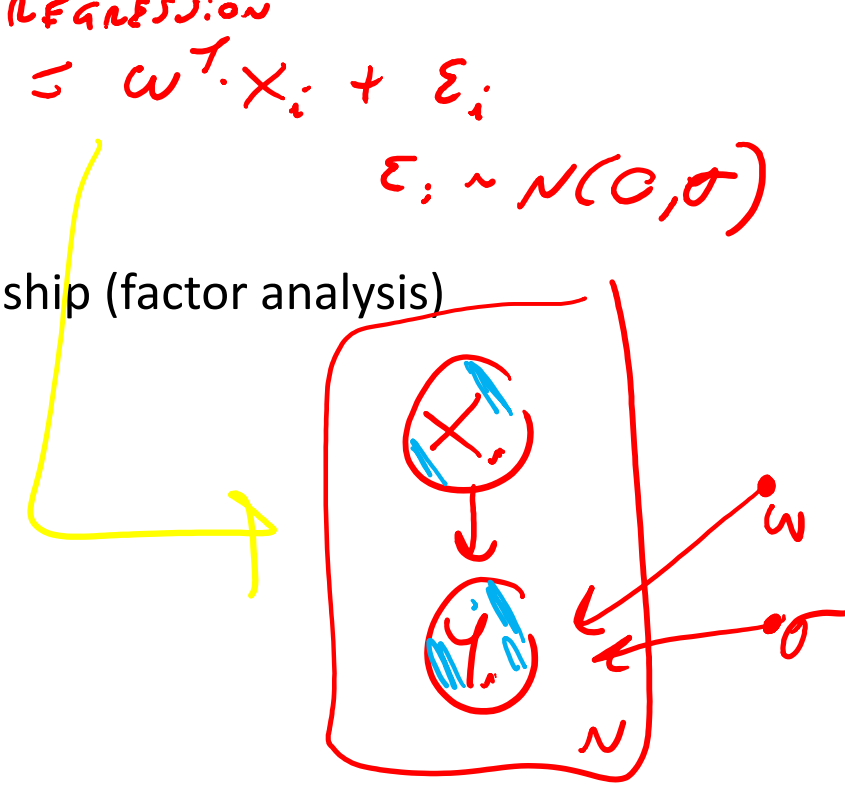
Probabilistic PCA

- PPCA provides a probabilistic formulation
- Addresses limitations of PCA
  - PCA can't deal with missing data
  - Outliers can skew the results significantly

- PPCA is a simple latent variable generative model
  - $x_i \in R^d$: observed data
  - $z_i \in R^k$: latent variable
  - $k \ll d$: latent space usually of much lower dimension
  - Transform a simple latent distribution $p(z)$ into the observable space

$p(x|z)$

PPCA: Formal Definition



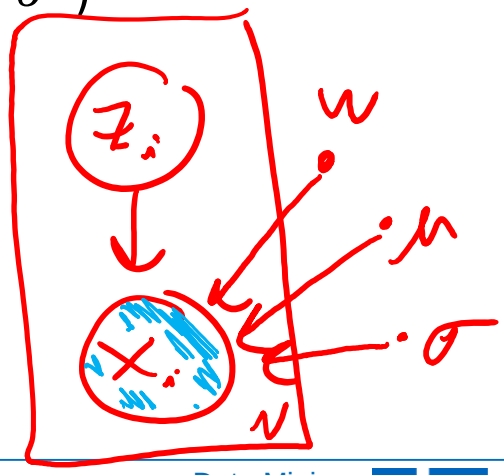
- Latent variable model with linear relationship (factor analysis)

  $x_i = Wz_i + \mu + \epsilon_i$

  - latent variable: $z_i \sim N(0, I)$
  - error (or noise): $\epsilon_i \sim N(0, \Psi)$
  - location term (mean): μ
  - weight matrix (factor loading matrix) W

- PPCA: Noise variances constrained to be equal ($\psi_i = \sigma^2$)
  - isotropic noise model $\epsilon_i \sim N(0, \sigma^2 I)$

$$z_i \sim N(0, I)$$
$$x_i | z_i \sim N(Wz_i + \mu, \sigma^2 I)$$



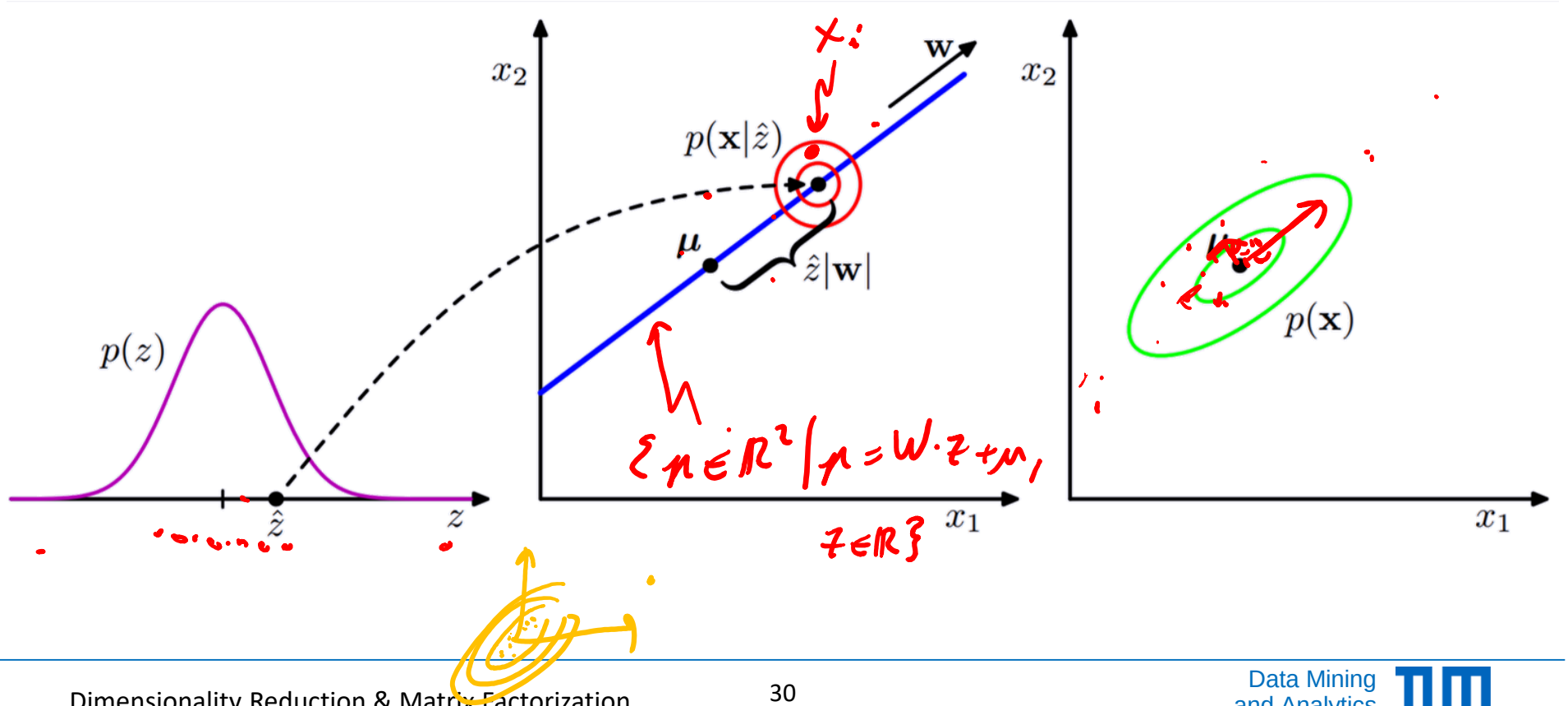
Generative view of PPCA

- $z_i \sim N(0, I)$
- $x_i | z_i \sim N(Wz_i + \mu, \sigma^2 I)$

$= N(Wz_i, \sigma^2 I) + \mu$

$z_i \in \mathbb{R}$

$W \in \mathbb{R}^{2\times 1}$ $\mu \in \mathbb{R}^{2\times 1}$

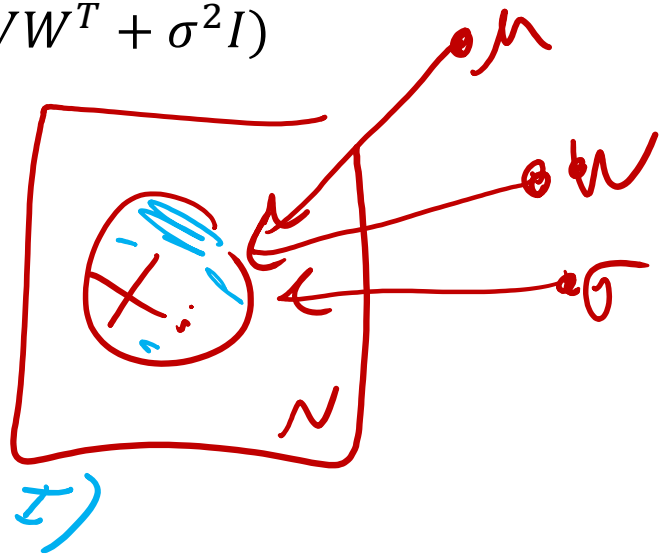


PPCA: Integrating out z

$$z_i \sim N(0, I)$$
$$x_i | z_i \sim N(Wz_i + \mu, \sigma^2 I)$$

by integrating z out, $p(x) = \int p(x \mid z)\, p(z)\, dz$

$$x_i \sim N(\mu,\ WW^T + \sigma^2 I)$$



MLE for PPCA

- The log-likelihood for the PPCA model

$$LL = -\frac{N}{2}\left(d\ln(2\pi) + \ln|C| + tr(C^{-1}S)\right)$$

where $C = WW^T + \sigma^2 I$ and $S = \frac{1}{N}\left(\sum_{i=1}^{N}(x_i - \mu)(x_i - \mu)^T\right)$

- MLE estimate for μ:
  - $\mu_{ML} = \frac{1}{N}\sum_{i=1}^{N} x_i$ = sample mean
  - accordingly, S becomes $\frac{1}{N}\left(\sum_{i=1}^{N}(x_i - \mu_{ML})(x_i - \mu_{ML})^T\right)$ = sample covariance

- MLE estimates for W and σ^2 can be solved in two ways:
  - Closed form
  - EM algorithm (more on this in an upcoming lecture)

Closed form MLE estimates for W and σ^2

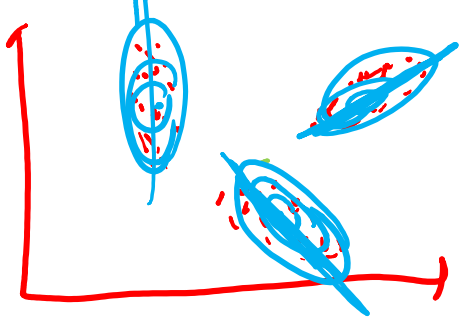
$\Sigma_x = U \Lambda U^T$

$\sigma \to 0 \quad W_{ml} = U (\Lambda)^{1/2} V$

- $W_{ML} = \mathrm{U_k}(\Lambda_\mathrm{k} - \sigma^2 I)^{\frac{1}{2}} V$
  - Represents the mapping of the latent space to that of the principal subspace
  - Columns of $\mathrm{U_k} \in R^{d\ x\ k}$ - principal eigenvectors of S
  - $\Lambda_\mathrm{k} \in R^{k\ x\ k}$ - diagonal matrix of corresponding eigenvalues of S
  - $V \in R^{k\ x\ k}$ - arbitrary rotation matrix, can be se to $I_{k\ x\ k}$

- $\sigma^2_{ML} = \frac{1}{d-k} \sum_{j=k+1}^{d} \lambda_j$
  - Represents the variance lost in the projection, averaged over the number dim decreased
  - λ_j are the corresponding eigenvalues

Relation to PCA and advantages



- PCA is equivalent to PPCA with variance σ^2 going to 0
  - details and proof in the tutorial

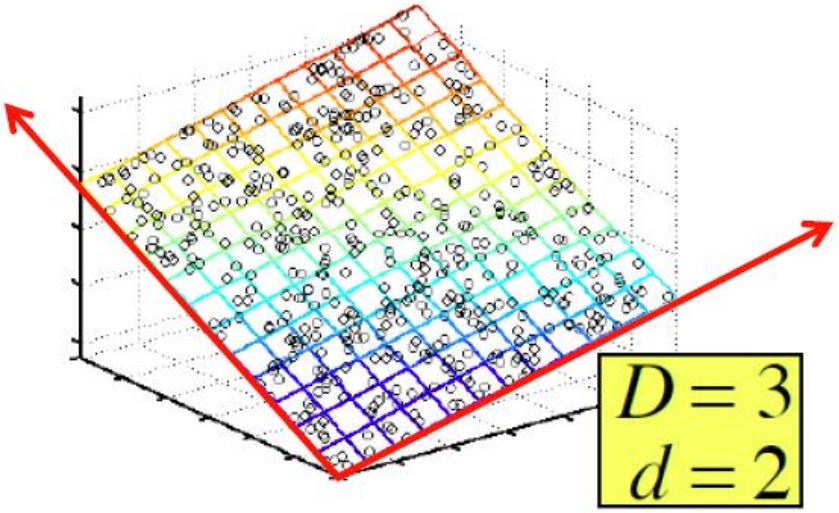
- Advantages of PPCA :
  - We can optimize the model without ever explicitly computing the covariance matrix that is computationally expensive
  - The model can be sampled from to generate new data
  - The existence of the likelihood function allows direct comparison of different models
  - It is easy to combine multiple PCA models into a mixture of PCA
  - Can handle missing data

Roadmap

- Chapter: Dimensionality Reduction & Matrix Factorization
  1. Introduction
  2. Principal Component Analysis (PCA)
  3. Probabilistic PCA (PPCA)
  4. **Singular Value Decomposition (SVD)**
     - **Idea: Low Rank Approximation**
     - SVD & Latent Factors
     - Dimensionality Reduction
  5. Matrix Factorization
  6. Autoencoders (Non-linear Dimensionality Reduction)

Low Dimensional Manifold

- Data often lies on a low dimensional manifold embedded in higher dimensional space



- How can me measure the dimensionality of this manifold?
  - put differently how to measure the intrinsic dimensionality of the data
- How can we find this manifold?

Rank of a Matrix

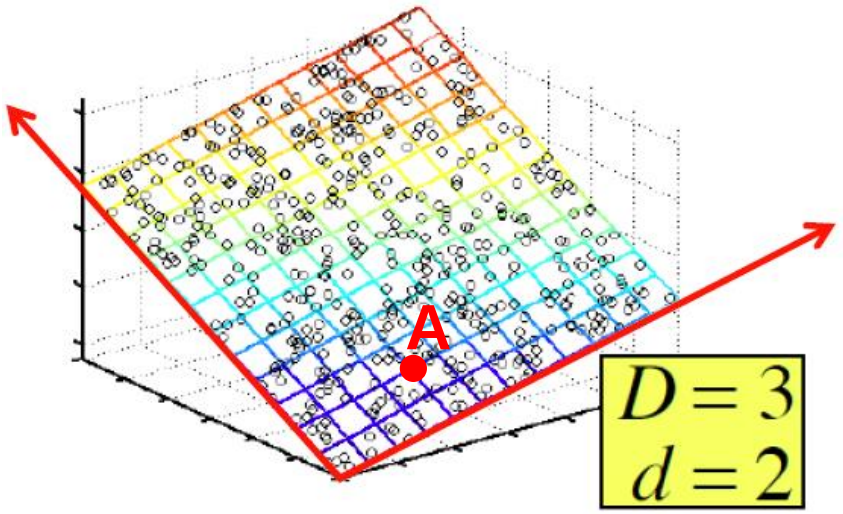
- Q: What is the rank of a matrix A?
- A: Number of linearly independent columns/rows of A
- Example:
  - Matrix A = $\begin{bmatrix} 1 & 2 & 1 \\ -2 & -3 & 1 \\ 3 & 5 & 0 \end{bmatrix}$ has rank r=2
    - Why? The first two rows are linearly independent, so the rank is at least 2, but all three rows are linearly dependent (the first is equal to the sum of the second and third) so the rank must be less than 3.
- Why do we care about low rank?
  - We can write A as two "basis" vectors: [1 2 1] [-2 -3 1]
  - And new coordinates of: [1 0] [0 1] [1 -1]

Rank is "Dimensionality"

- Cloud of points in 3D space:
  - Think of point positions as a matrix:

1 row per point:

$$\begin{bmatrix} 1 & 2 & 1 \\ -2 & -3 & 1 \\ 3 & 5 & 0 \end{bmatrix} \begin{matrix} \mathbf{A} \\ \mathbf{B} \\ \mathbf{C} \end{matrix}$$



- **We can rewrite coordinates more efficiently!**
  - Old basis vectors: [1 0 0] [0 1 0] [0 0 1]
  - **New basis vectors: [1 2 1] [-2 -3 1]**
  - Then A has new coordinates: [1 0], B: [0 1], C: [1 -1]
    - **Notice: We reduced the number of coordinates!**

Low Rank Approximation

- Idea: approximate original data A by a low rank matrix B

$$A = \begin{bmatrix} 1.01 & 2.05 & 0.9 \\ -2.1 & -3.05 & 1.1 \\ 2.99 & 5.01 & 0.3 \end{bmatrix} \approx \begin{bmatrix} 1 & 2 & 1 \\ -2 & -3 & 1 \\ 3 & 5 & 0 \end{bmatrix} = B$$

rank(A)=3
we need 3 coordinates
to describe each point

rank(B)=2
we need only 2 coordinates
per point

Low Rank Approximation

- Idea: approximate original data A by a low rank matrix B

$$A = \begin{bmatrix} 1.01 & 2.05 & 0.9 \\ -2.1 & -3.05 & 1.1 \\ 2.99 & 5.01 & 0.3 \end{bmatrix} \approx \begin{bmatrix} 1 & 2 & 1 \\ -2 & -3 & 1 \\ 3 & 5 & 0 \end{bmatrix} = B$$

- Important: Even though both A and B are $\in R^{n \times d}$ we need only two coordinates per point to describe B
  - rank(A)=3 vs. rank(B)=2 (3 vs. 2 coordinates per point)

- **Goal: Find the best low rank approximation**
  - best = minimize the sum of reconstruction error
  - Given matrix A $\in R^{n \times d}$, find B $\in R^{n \times d}$ with rank(B)=k that minimizes

$$\|A - B\|_F^2 = \sum_{i=1}^{N} \sum_{j=1}^{D} \left(a_{ij} - b_{ij}\right)^2$$

Roadmap

- Chapter: Dimensionality Reduction & Matrix Factorization
  1. Introduction
  2. Principal Component Analysis (PCA)
  3. Probabilistic PCA (PPCA)
  4. **Singular Value Decomposition (SVD)**
     - Idea: Low Rank Approximation
     - **SVD & Latent Factors**
     - Dimensionality Reduction
  5. Matrix Factorization
  6. Autoencoders (Non-linear Dimensionality Reduction)

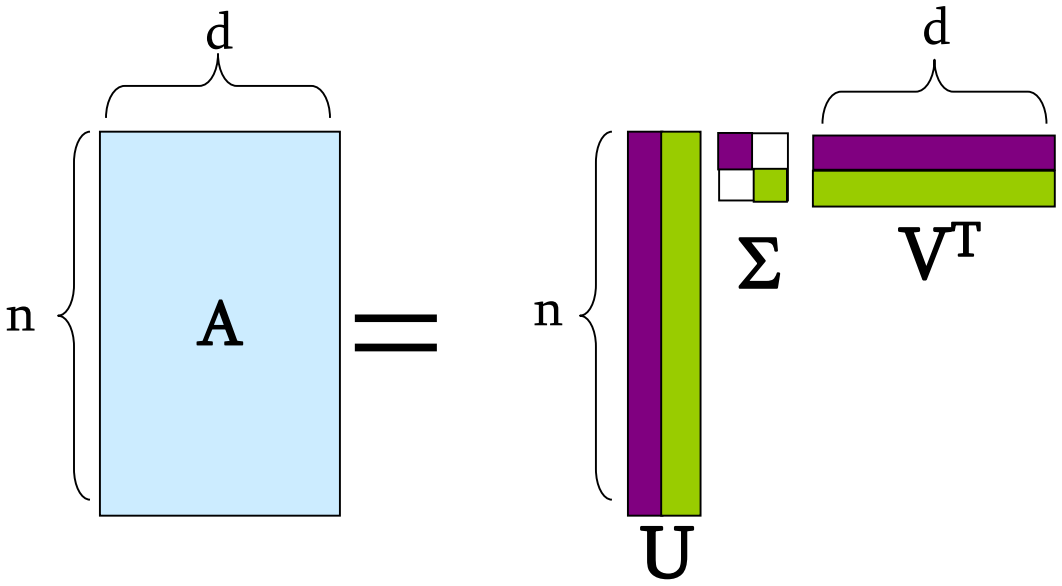
Singular Value Decomposition (SVD): Definition

- Each real matrix $A \in \mathbb{R}^{n \times d}$ can be decomposed into $A = U \cdot \Sigma \cdot \mathrm{V}^{\mathrm{T}}$ (note: **exact representation**, no approximation), where
  - $U \in \mathbb{R}^{n \times r}, \Sigma \in \mathbb{R}^{r \times r}, V \in \mathbb{R}^{\mathrm{d} \times \mathrm{r}}$
  - U, V: column orthonormal
    - i.e. $U^T U = I$; $V^T V = I$ (I: identity matrix)
    - U are called the left singular vectors , V the right singular vectors
  - Σ: diagonal
    - r x r diagonal matrix (r : rank of the matrix A)
    - entries (called singular values) are positive, and sorted in decreasing order ($\sigma_1 \geq \sigma_2 \geq \ldots \geq 0$)

- Remark: The decomposition is (almost) unique
  - see e.g. multiplication by -1

Singular Value Decomposition

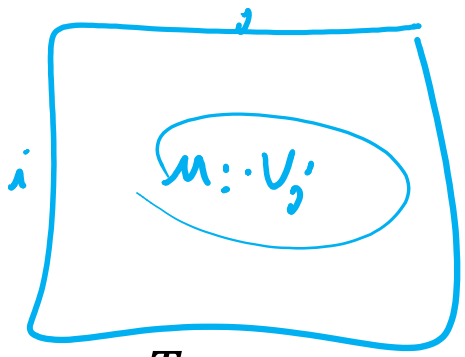
$$A = U\Sigma V^T = \sum_{i=1}^{r} \sigma_i \cdot u_i \circ v_i^T$$



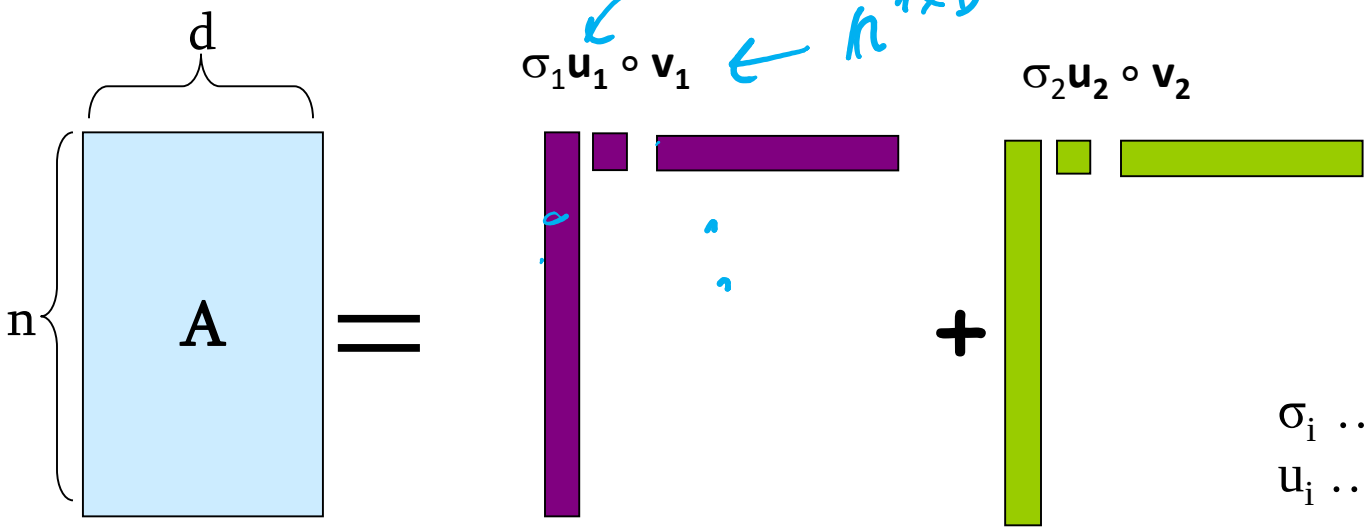
σ_i … scalar
u_i … vector
v_i … vector

here: r=2

Singular Value Decomposition



$$A = U\Sigma V^T = \sum_{i=1}^{r} \sigma_i \cdot u_i \circ v_i^T$$

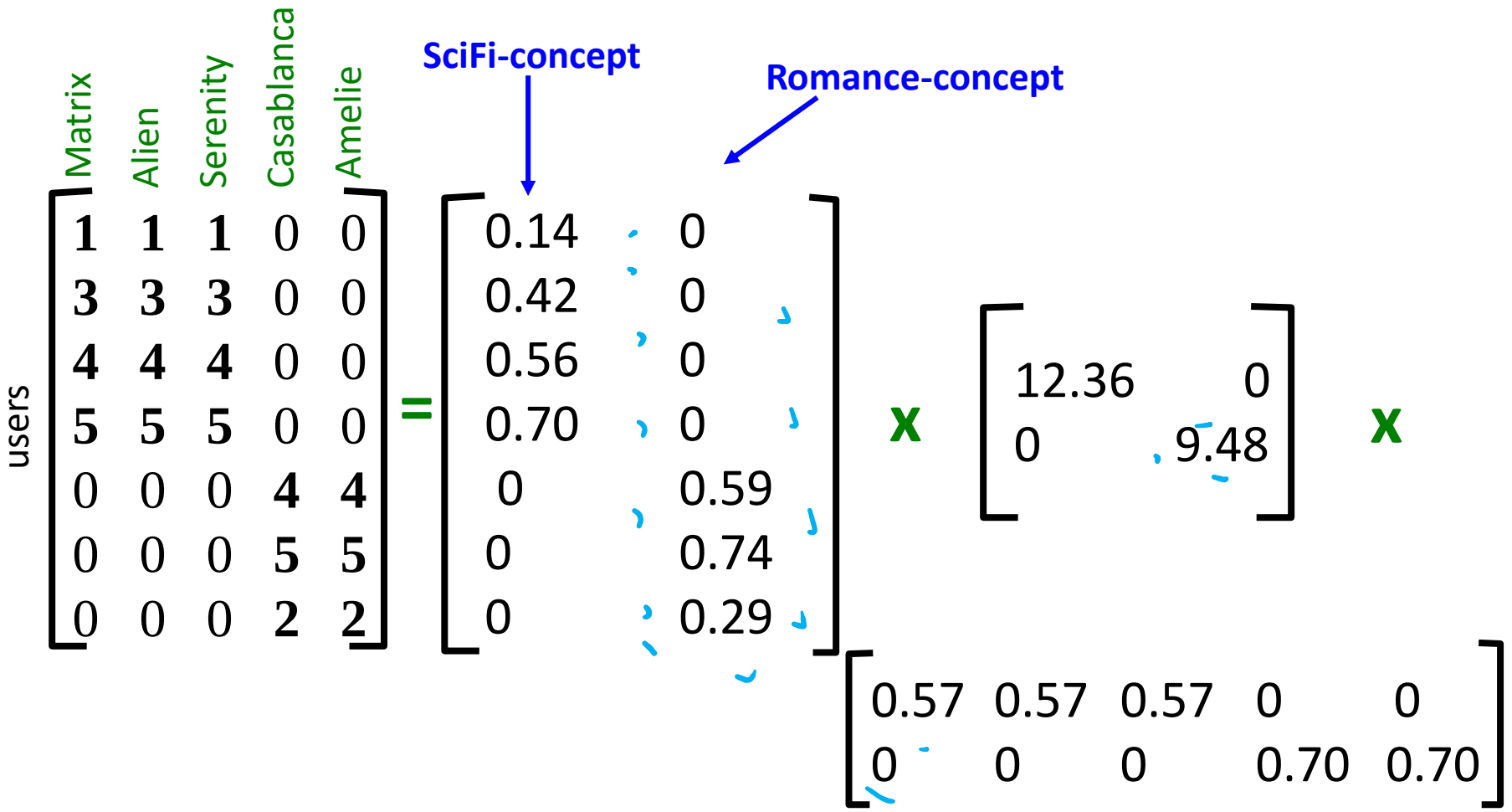


σ_i … scalar
u_i … vector
v_i … vector

here: r=2

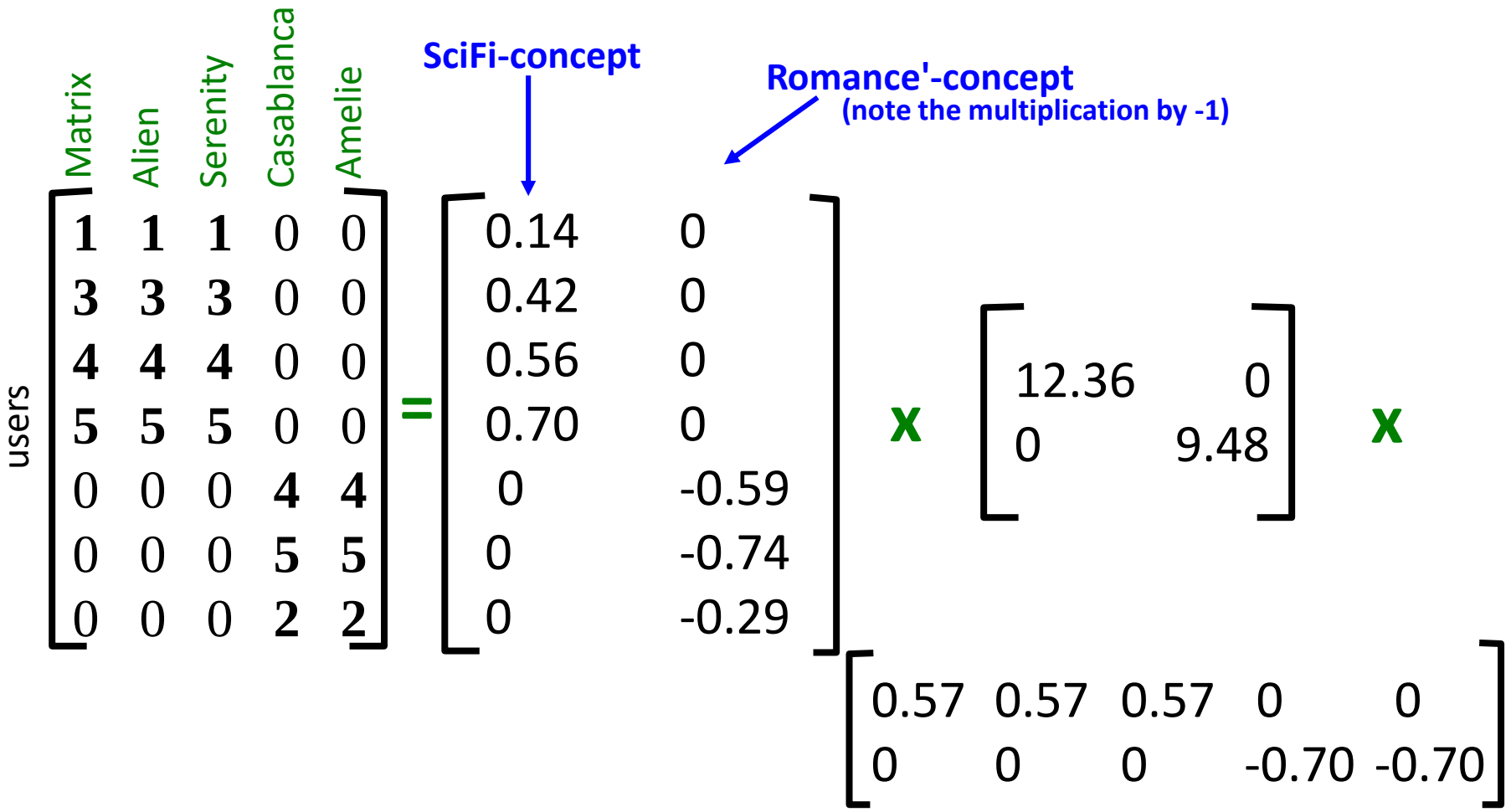
SVD Example: Users-to-Movies

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



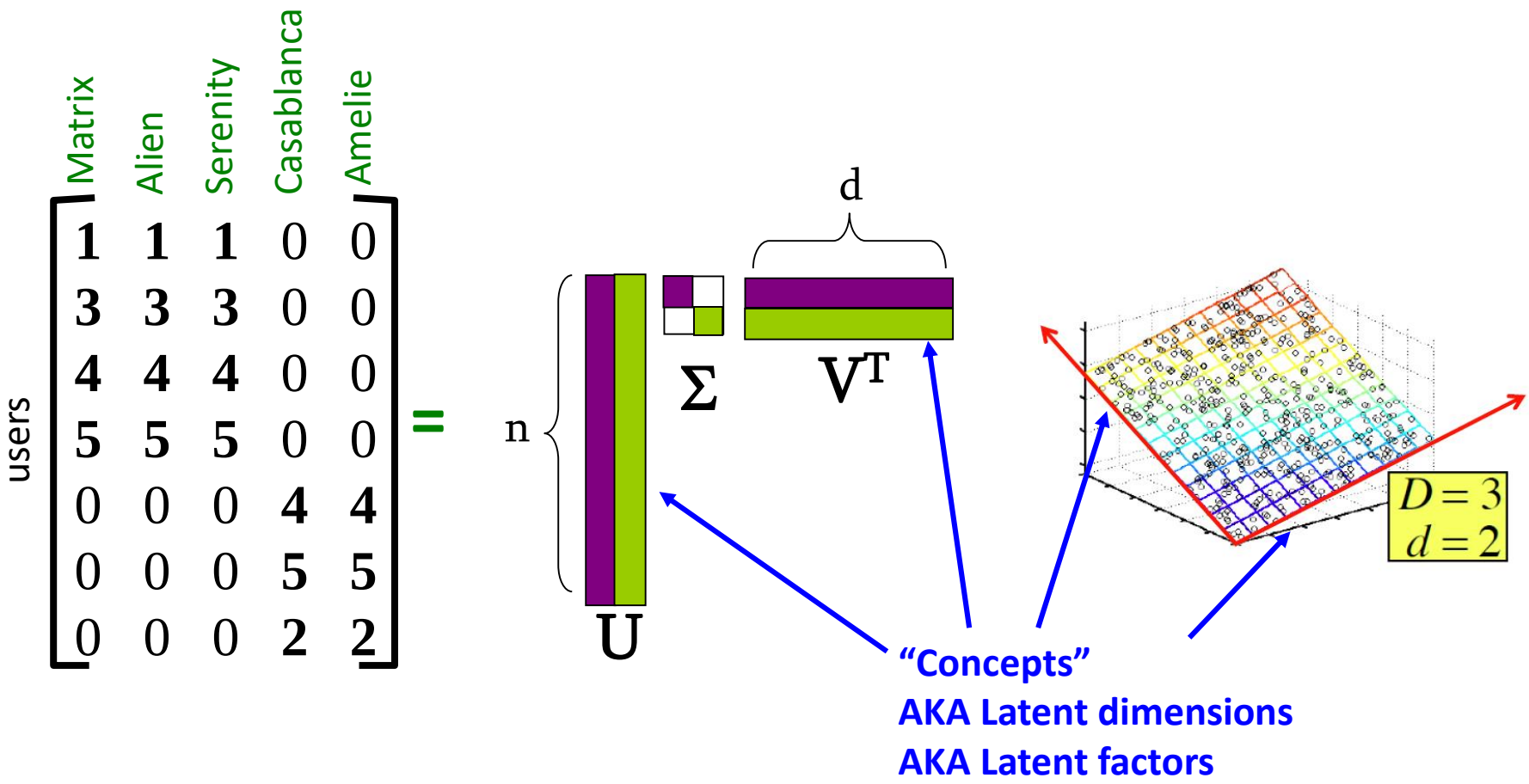
SVD Example: Users-to-Movies

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



SVD Example: Latent Factors

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



SVD Example: Beyond Blocks

	Matrix	Alien	Serenity	Casablanca	Amelie
	1	**1**	**1**	0	0
	3	**3**	**3**	0	0
	4	**4**	**4**	0	0
	5	**5**	**5**	0	0
	0	**2**	0	**4**	**4**
	0	0	0	**5**	**5**
	0	**1**	0	**2**	**2**

=

0.13	-0.02	-0.01
0.41	-0.07	-0.03
0.55	-0.09	-0.04
0.68	-0.11	-0.05
0.15	**0.59**	**0.65**
0.07	**0.73**	**-0.67**
0.07	**0.29**	**0.32**

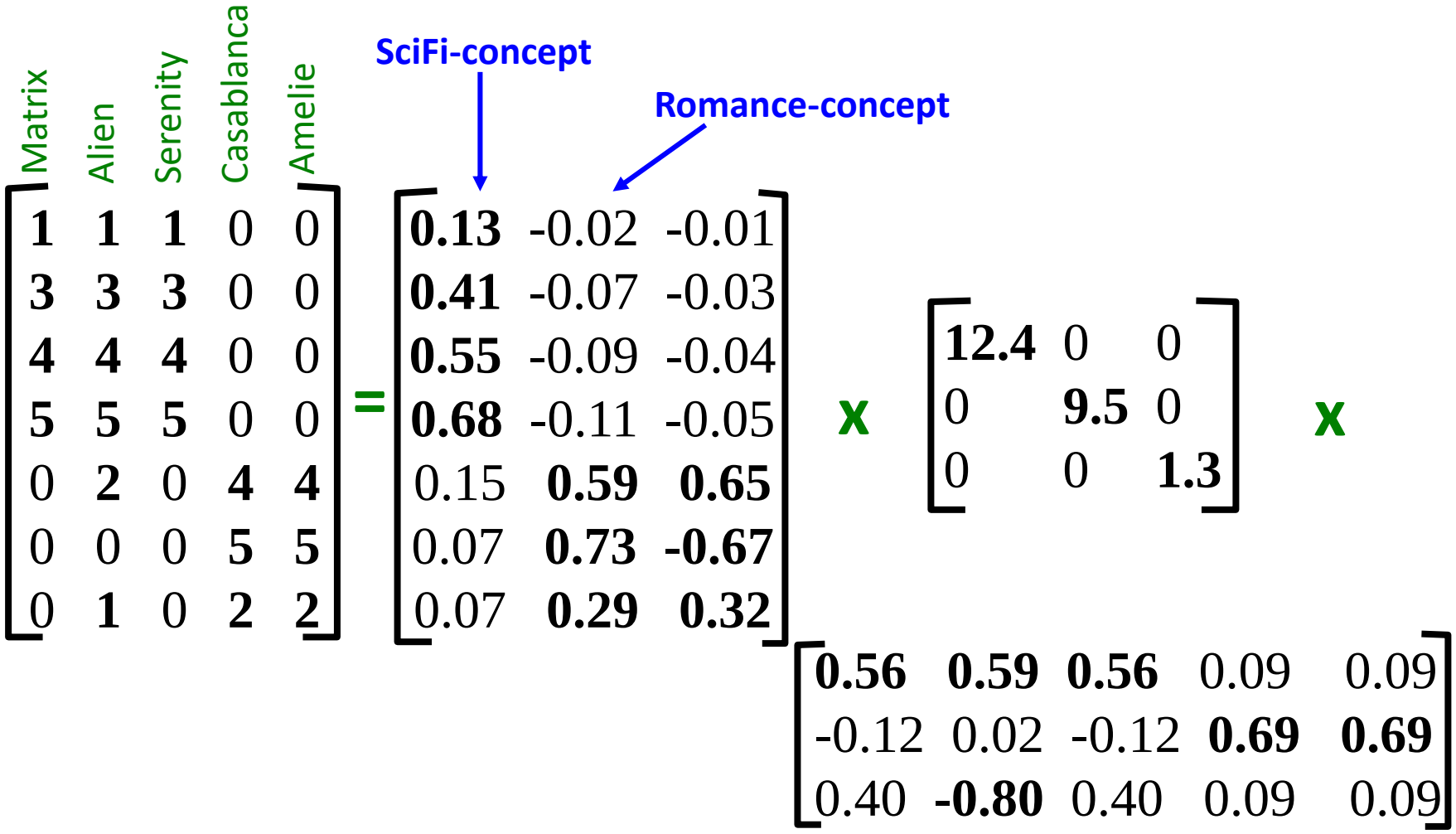
x

12.4	0	0
0	**9.5**	0
0	0	**1.3**

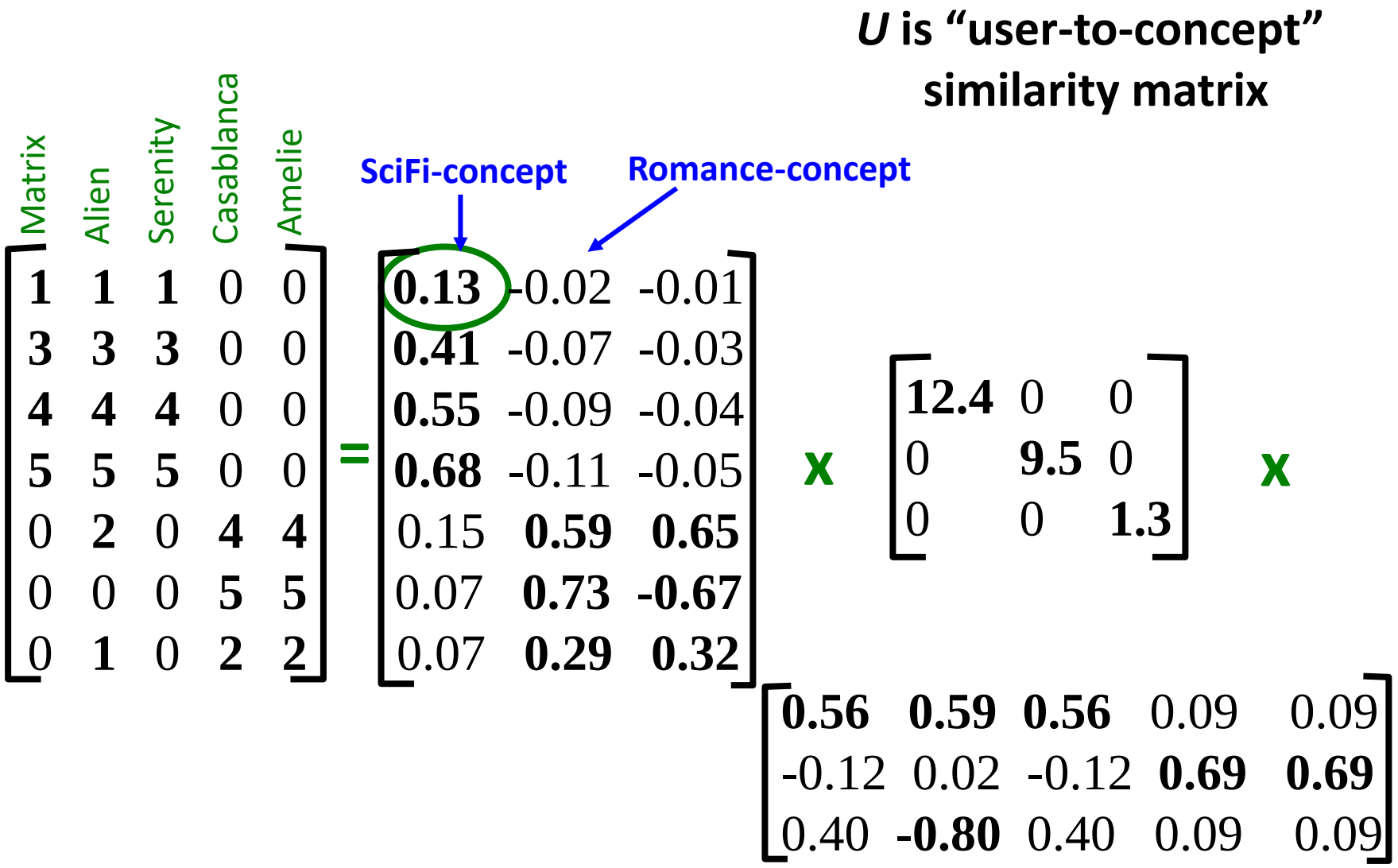
x

0.56	**0.59**	**0.56**	0.09	0.09
-0.12	0.02	-0.12	**0.69**	**0.69**
0.40	**-0.80**	0.40	0.09	0.09

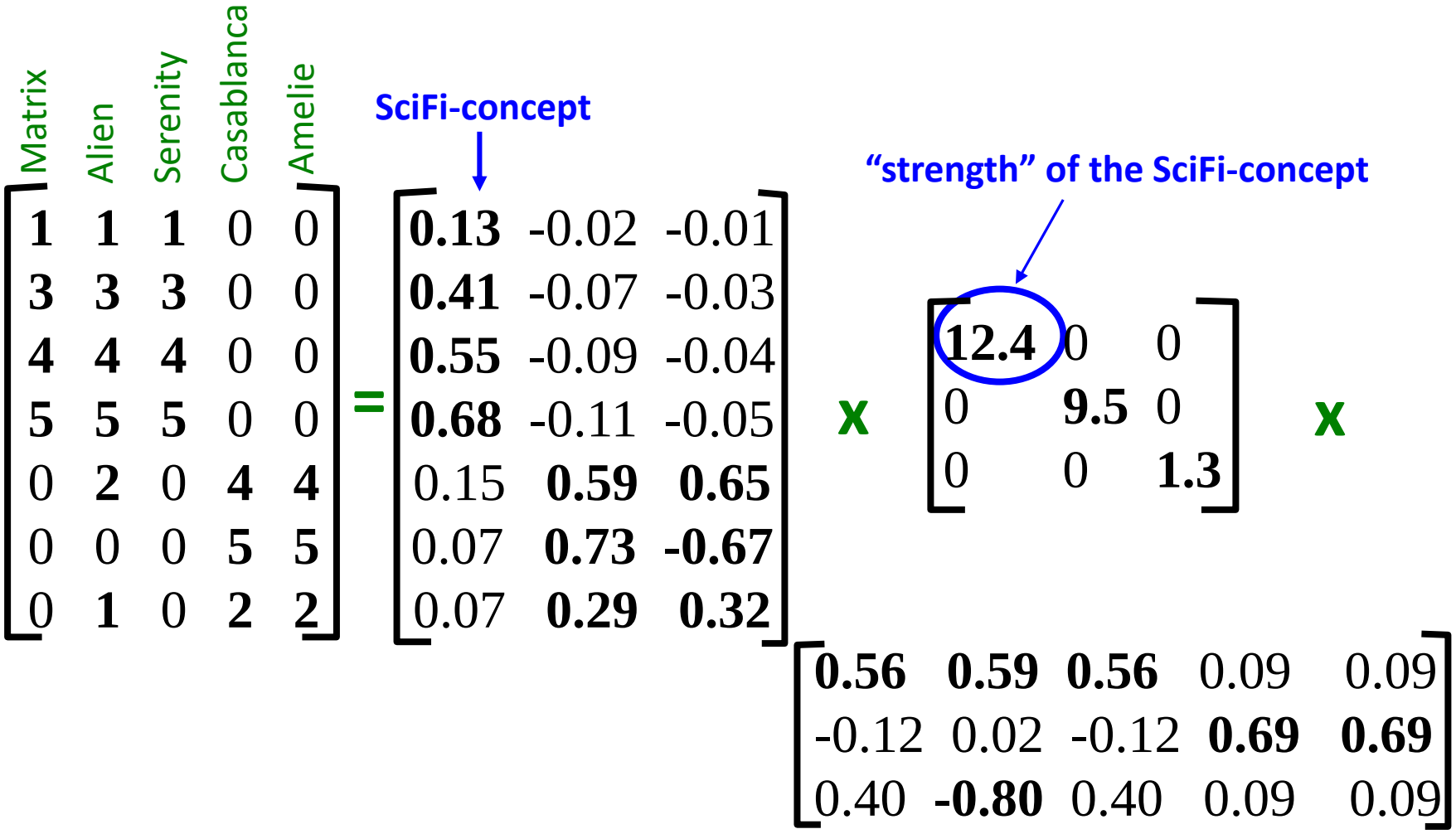
SVD Example: Beyond Blocks



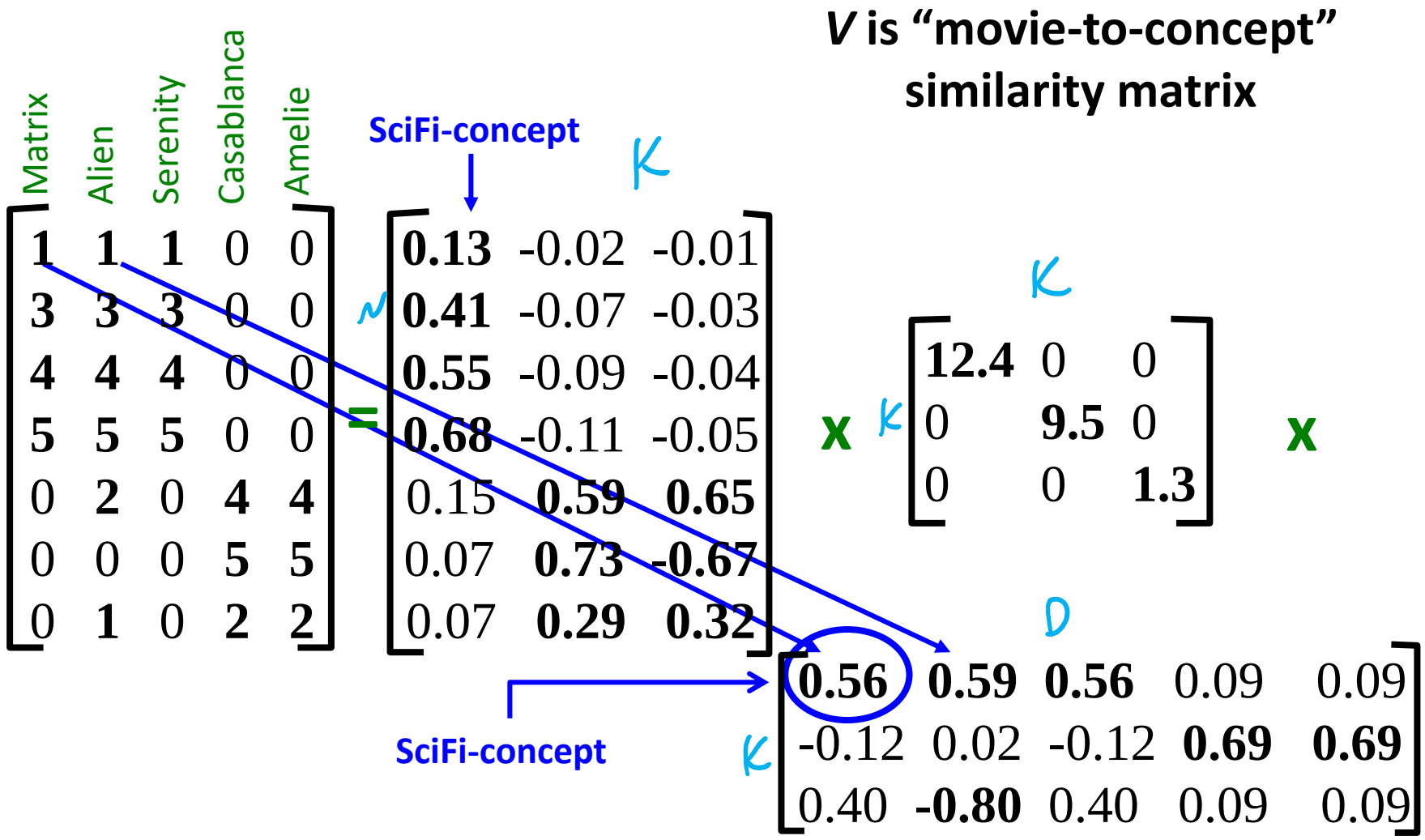
SVD Example: Beyond Blocks



SVD Example: Beyond Blocks



SVD Example: Beyond Blocks



***V* is "movie-to-concept" similarity matrix**

SVD: Interpretation

- A = $U\Sigma V^T$
- 'movies', 'users' and 'concepts':
  - A original data: movies-to-users
  - U: user-to-concept similarity matrix
  - V: movie-to-concept similarity matrix
  - Σ: its diagonal elements: 'strength' of each concept

- **Benefits of SVD (or in general matrix decomposition):**
  - Discover hidden correlations/topics
    - Words that occur commonly together; movies of the same genre; …
  - Interpretation and visualization

Roadmap

- Chapter: Dimensionality Reduction & Matrix Factorization
  1. Introduction
  2. Principal Component Analysis (PCA)
  3. Probabilistic PCA (PPCA)
  4. **Singular Value Decomposition (SVD)**
     - Idea: Low Rank Approximation
     - SVD & Latent Factors
     - **Dimensionality Reduction**
  5. Matrix Factorization
  6. Autoencoders (Non-linear Dimensionality Reduction)

Recap: Dim. Reduction by Low Rank Approx.

- Idea: approximate original data A, by a low rank matrix B

$$A = \begin{bmatrix} 1.01 & 2.05 & 0.9 \\ -2.1 & -3.05 & 1.1 \\ 2.99 & 5.01 & 0.3 \end{bmatrix} \approx \begin{bmatrix} 1 & 2 & 1 \\ -2 & -3 & 1 \\ 3 & 5 & 0 \end{bmatrix} = B$$

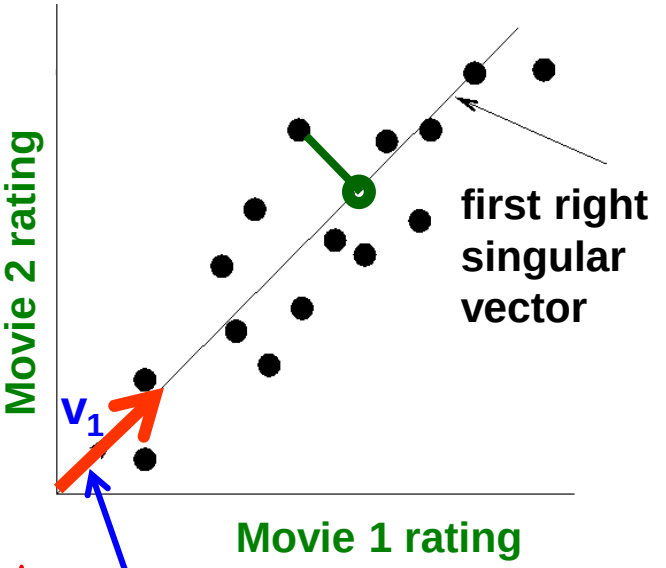
- Important: Even though both A and B are $\in R^{n \times d}$ we need only two coordinates per point to describe B
  - rank(A)=3 vs. rank(B)=2 (3 vs. 2 coordinates per point)

- **Goal: Find the best low rank approximation**
  - best = minimize the sum of reconstruction error
  - Given matrix A $\in R^{n \times d}$, find B $\in R^{n \times d}$ with rank(B)=k that minimizes

$$\|A - B\|_F^2 = \sum_{i=1}^{N} \sum_{j=1}^{D} \left(a_{ij} - b_{ij}\right)^2$$

SVD: Alternative Interpretation

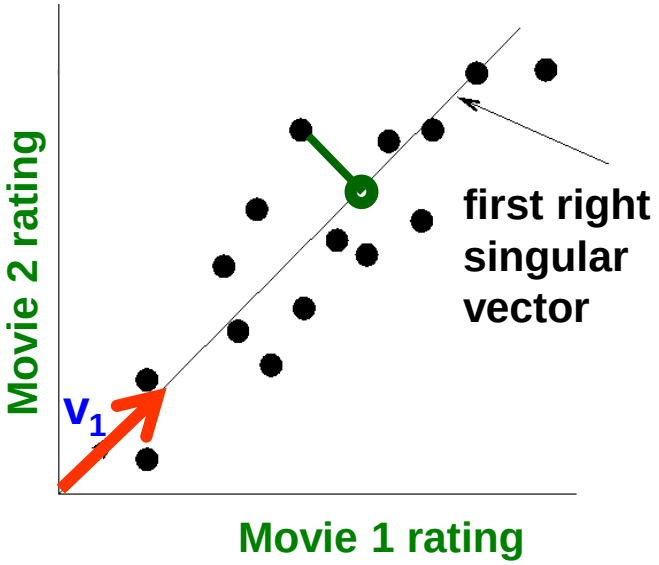
- SVD gives us 'best' axis to project on:
- A = $U\Sigma V^T$ - example:
  - V: "movie-to-concept" matrix
  - U: "user-to-concept" matrix



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

SVD: Alternative Interpretation

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

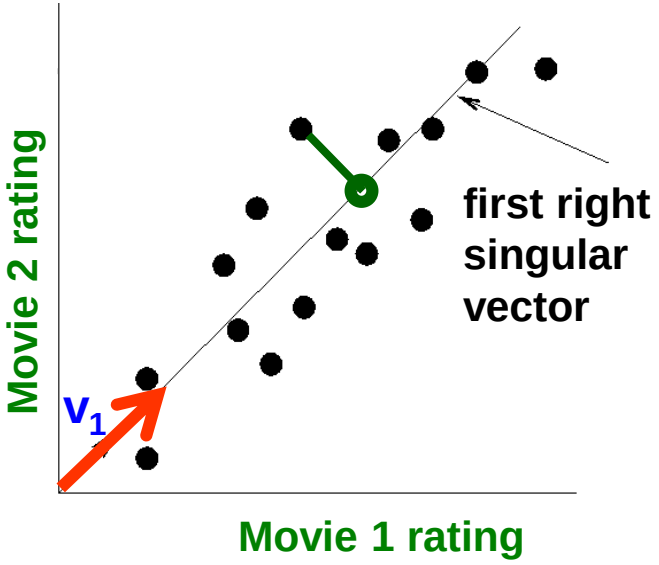


variance ('spread') on the v_1 axis

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

SVD: Alternative Interpretation

- A = $U\Sigma V^T$ - example:
- **U Σ: Gives the coordinates of the points in the projection axis**



$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 2 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 1 & 0 & 2 & 2 \end{bmatrix}$$

Projection of users on the "Sci-Fi" axis (U Σ)^T:

$$\begin{bmatrix} 1.61 & 0.19 & -0.01 \\ 5.08 & 0.66 & -0.03 \\ 6.82 & 0.85 & -0.05 \\ 8.43 & 1.04 & -0.06 \\ 1.86 & -5.60 & 0.84 \\ 0.86 & -6.93 & -0.87 \\ 0.86 & -2.75 & 0.41 \end{bmatrix}$$

SVD: Best Approximation

- How to find the best approximation?

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

SVD: Best Approximation

- How to find the best approximation?
- Set smallest singular values to zero!

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

SVD: Best Approximation

- How to find the best approximation?
- Set smallest singular values to zero!

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

SVD: Best Approximation

- How to find the best approximation?
- Set smallest singular values to zero!

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

SVD: Best Approximation

- How to find the best approximation?
- Set smallest singular values to zero!

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

SVD: Best Approximation

- How to find the best approximation?
- Set smallest singular values to zero!

by construction, the rank of the new matrix is 2

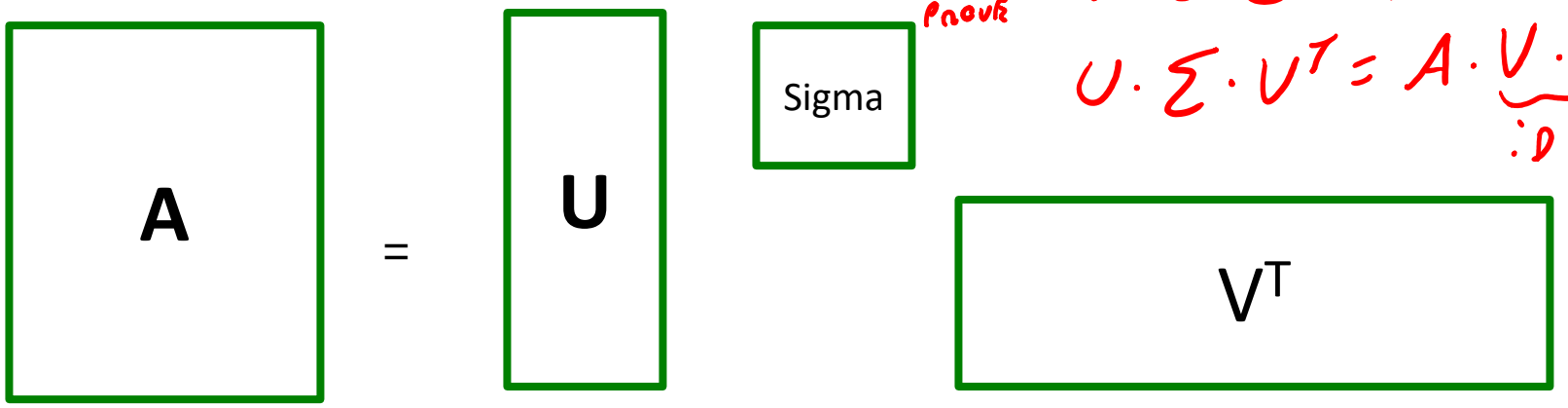
$$\begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ \mathbf{3} & \mathbf{3} & \mathbf{3} & 0 & 0 \\ \mathbf{4} & \mathbf{4} & \mathbf{4} & 0 & 0 \\ \mathbf{5} & \mathbf{5} & \mathbf{5} & 0 & 0 \\ 0 & \mathbf{2} & 0 & \mathbf{4} & \mathbf{4} \\ 0 & 0 & 0 & \mathbf{5} & \mathbf{5} \\ 0 &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

SVD: Best Low Rank Approximation

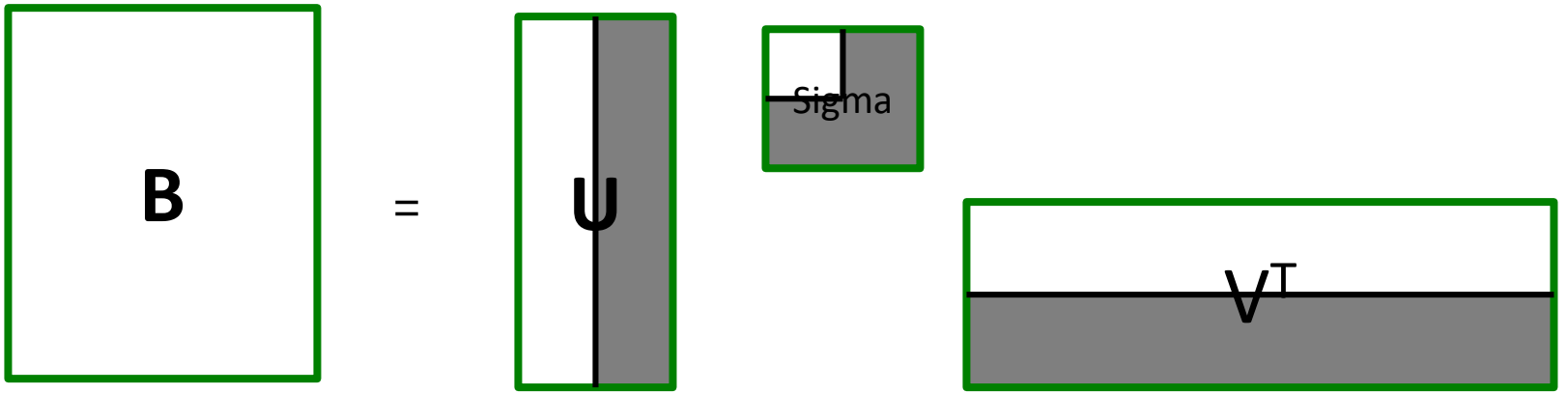
$A = U \cdot \Sigma \cdot V^T$

TO PROVE $\rightarrow$ $U \cdot \Sigma = A \cdot V$

$U \cdot \Sigma \cdot V^T = A \cdot \underbrace{V \cdot V^T}_{:D}$

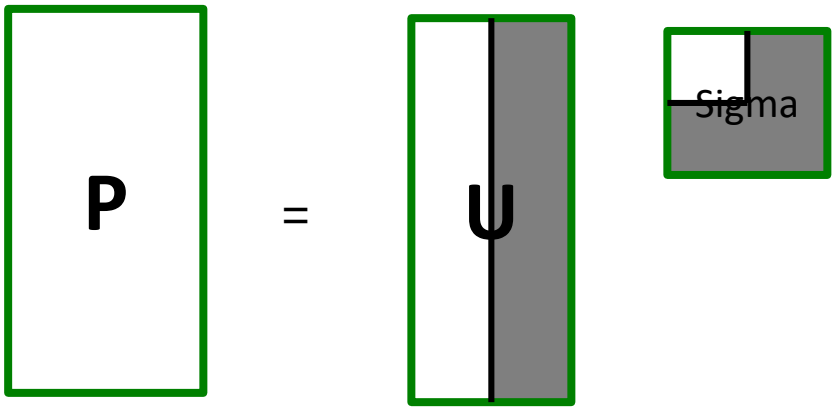


B is best approximation of A



SVD: Projection

- Note: The actual projected/reduced data can be obtained by computing

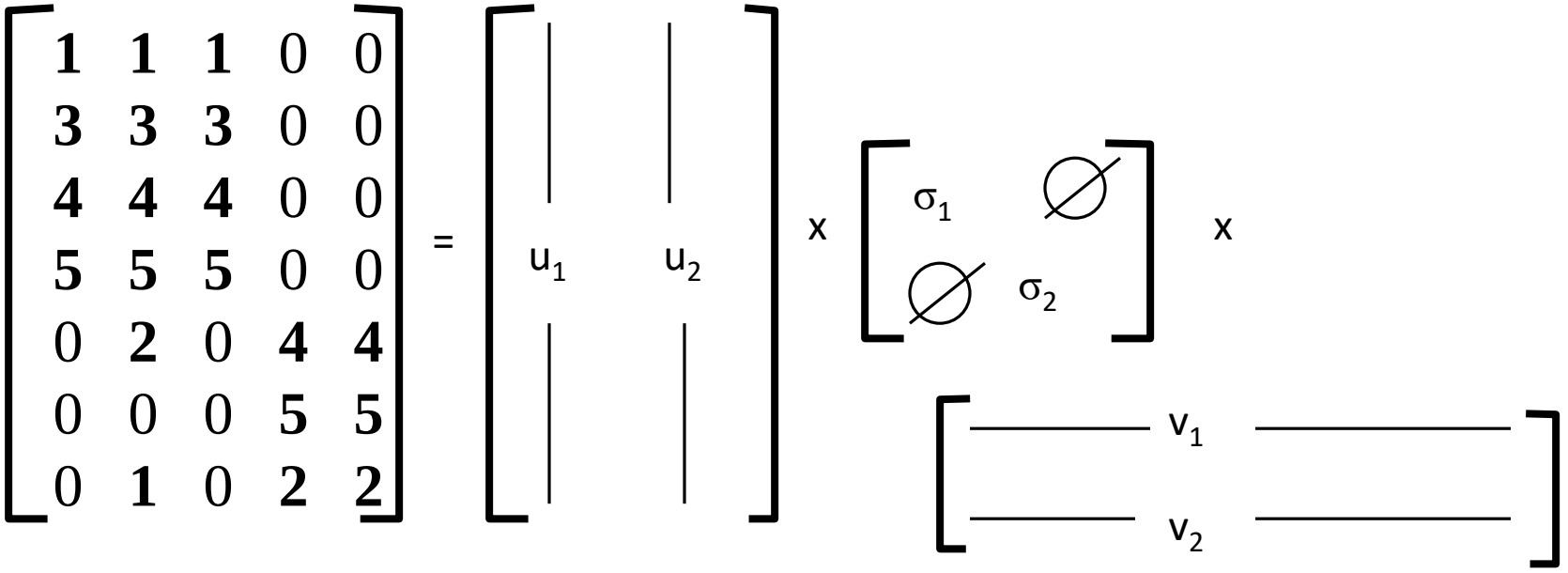


- Or equivalently: $P = A \cdot V$ (since V is orthonormal)

$V_{TRUNCATED}$

Best Approximation – Intuitive Explanation

- Recap: Vectors u_i and v_i are of unit length
- W.l.o.g.: $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \ldots \geq 0$



Best Approximation – Intuitive Explanation

- Recap: Vectors u_i and v_i are of unit length
- W.l.o.g.: $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \ldots \geq 0$

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 2 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 1 & 0 & 2 & 2 \end{bmatrix} = \sigma_1 \; u_1 \; v^T_1 + \sigma_2 \; u_2 \; v^T_2 + \ldots$$

r terms

Q: How many σ_i to pick?
A: Rule of thumb:
keep 90% of 'energy'
$\sum_{i=1}^{k} \sigma_i^2 \geq 0.9 \sum_{i=1}^{r} \sigma_i^2$

- σ_i scales the terms $u_i \cdot v_i^T$
- Zeroing small σ_i introduces less error

SVD: Best Low Rank Approximation - Proof

- Theorem: Let A = $U\Sigma V^T$ ($\sigma_1 \geq \sigma_2 \geq \cdots$, rank(A)=r) and B = USV^T with S being a diagonal $r \times r$ matrix where
  - $s_i = \sigma_i$ for i=1…k and $s_i = 0$ else

  Then B is a best rank-k approximation to A regarding Frobenius norm, i.e. B is a solution to $\min_B \|A - B\|_F$ where rank(B)=k

- We have uploaded a detailed proof to the web
  - Note: Many proofs on the web and on other lecture slides are incorrect!

- Remark: B is also an optimal low-rank approximation regarding the spectral norm (operator 2-norm): $\min_B \|A - B\|_2$
  - $\|X\|_2$ = largest singular value of X

SVD: Best Low Rank Approximation - Proof

- Some facts:
  - $\|X\|_F = \|X^T\|_F$
    - obvious from the definition
  - $\|X\|_F^2 = trace(X^T X)$ // trace = sum of diagonal entries
    - easy homework
  - Frobenius norm is invariant to orthonormal transformations *U*
    - Note: If $U^T U = I$ then also $UU^T = I$
    - $\|UX\|_F^2 = trace((UX)^T(UX)) = trace(X^T U^T UX) = \text{trace}(X^T X) = \|X\|_F^2$
    - $\|XU\|_F^2 = \|(XU)^T\|_F^2 = trace(((XU)(XU)^T)$
      $= trace(XUU^T X^T) = \text{trace}(XX^T) = \|X\|_F^2$
  - Let $A = U\Sigma V^T$ then $\|A\|_F^2 = \|\Sigma\|_F^2 = \sum_i^r \sigma_i^2$
    - follows from above results

SVD: Complexity

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

- To compute SVD:
  - $O(n \cdot d^2)$ or $O(n^2 \cdot d)$ (whichever is less)
- But:
  - Less work, if we just want singular values
  - or if we want first k singular vectors
  - or if the matrix is sparse
- Implemented in linear algebra packages like
  - LINPACK, Matlab, SPlus, Mathematica ...

SVD & PCA: Comparison

- Given data X (let's assume it is already centered)
- SVD gives us:
  - $X = U\Sigma V^T$
  - Projected data obtained by $X \cdot V$ (or truncated V)
- PCA computes the eigendecomposition of the covariance matrix
  - Covariance matrix: $X^T X$
  - Eigendecomposition leads to $X^T X = \Gamma \cdot \Lambda \cdot \Gamma^T$
  - Projected data obtained by $X \cdot \Gamma$ (or truncated Γ)
- Let us calculate:
  - $X^T X = (U\Sigma V^T)^T U\Sigma V^T = V\Sigma^T U^T (U\Sigma V^T) = V\Sigma\Sigma^T V^T = V\Sigma^2 V^T$
  - $\Gamma = V$ **PCA and SVD are equivalent!**
  - $\Sigma^2 = \Lambda$ **squared singular values are variances in new space!**

SVD & PCA: Comparison

transform the data such that dimensions of new space are uncorrelated + discard (new) dimensions with smallest variance

=

find optimal low-rank approximation

(regarding Frobenius norm)

Remark: Computation of SVD

- We can use the eigendecomposition to calculate the singular value decomposition

- $X^T X = \left(U\Sigma V^T\right)^T U\Sigma V^T = V\Sigma^T U^T (U\Sigma V^T) = V\Sigma\Sigma^T V^T = V\Sigma^2 V^T$
  - V = eigenvectors of $X^T X$
- $XX^T = U\Sigma V^T (V\Sigma^T U^T) = U\Sigma\Sigma^T U^T$
  - U = eigenvectors of XX^T

- Drawback: Numerically instable
  - better to use specialized algorithms

Roadmap

- Chapter: Dimensionality Reduction & Matrix Factorization
  1. Introduction
  2. Principal Component Analysis (PCA)
  3. Probabilistic PCA (PPCA)
  4. Singular Value Decomposition (SVD)
  5. **Matrix Factorization**
     - **Motivation & Approach**
     - Regularization & Sparsity
     - Further Factorization Models
  6. Autoencoders (Non-linear Dimensionality Reduction)

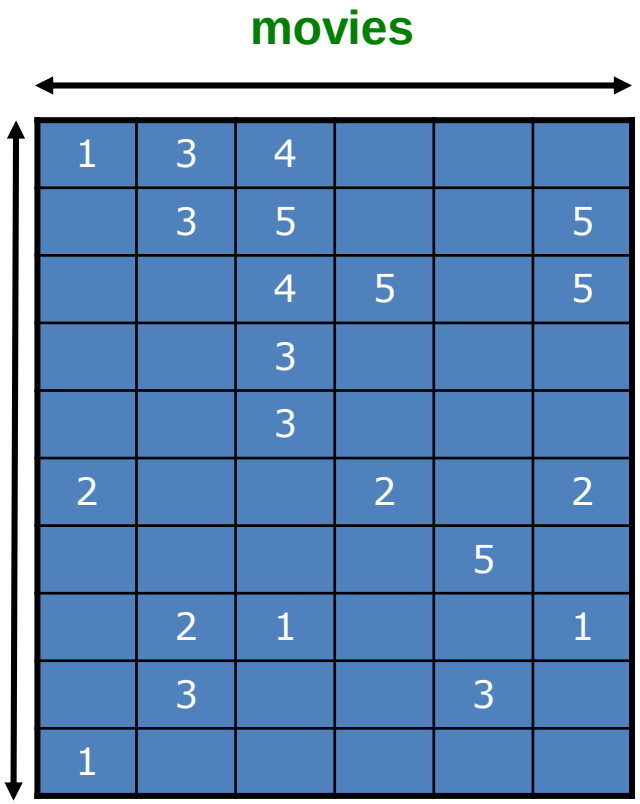
Motivation: The Netflix Prize

- Training data
  - 100 million ratings, 480,000 users, 17,770 movies
  - 6 years of data: 2000-2005
- Test data
  - Last few ratings of each user (2.8 million)
  - Evaluation criterion:

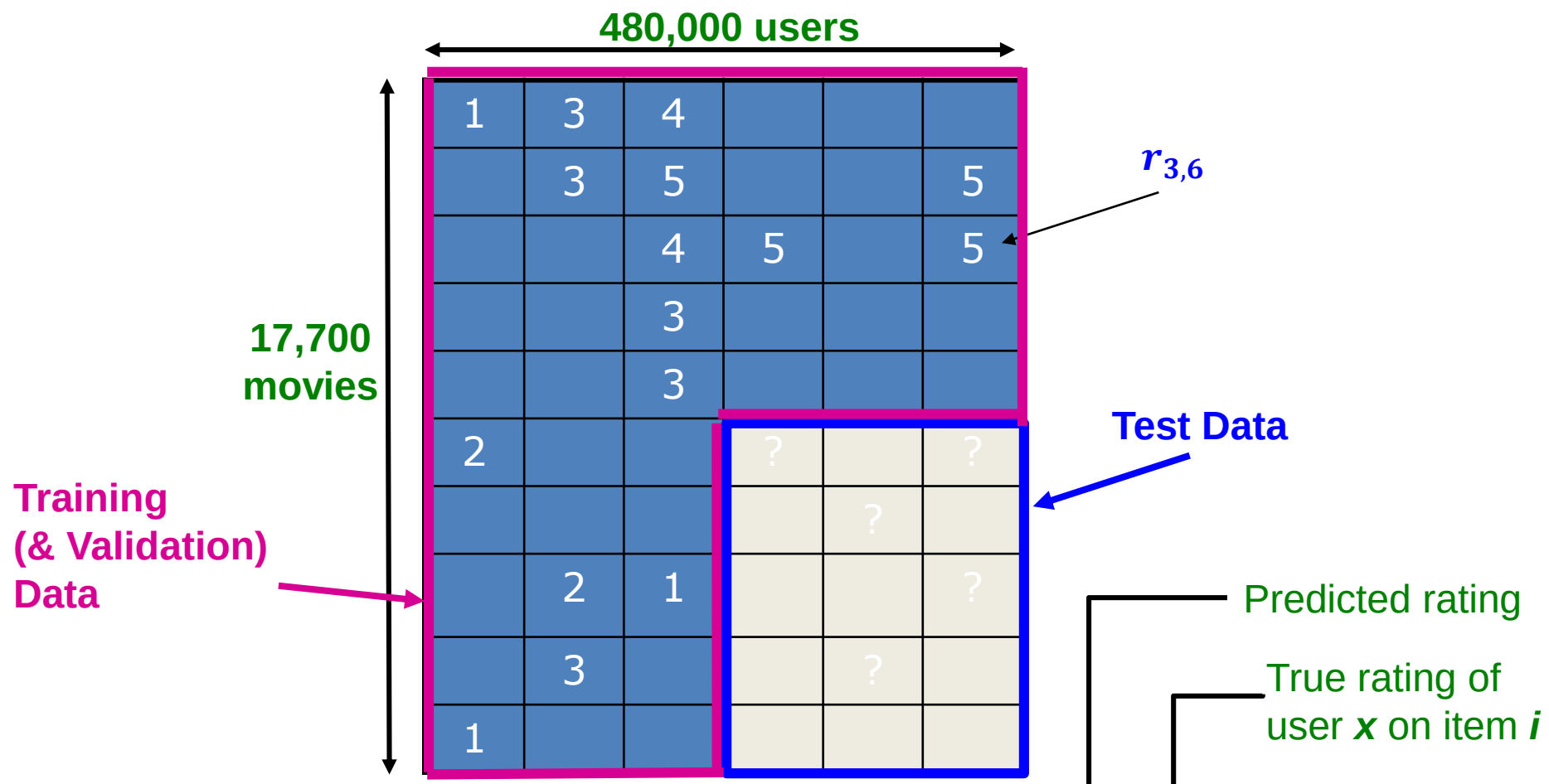
    Root Mean Square Error (RMSE)

    $$\text{RMSE} = \frac{1}{|R|}\sqrt{\sum_{(i,x)\in R}(\hat{r}_{xi} - r_{xi})^2}$$

  - Netflix's system RMSE: 0.9514
- Competition
  - 2,700+ teams
  - $1 million prize for 10% improvement on Netflix



Utility Matrix R: Evaluation

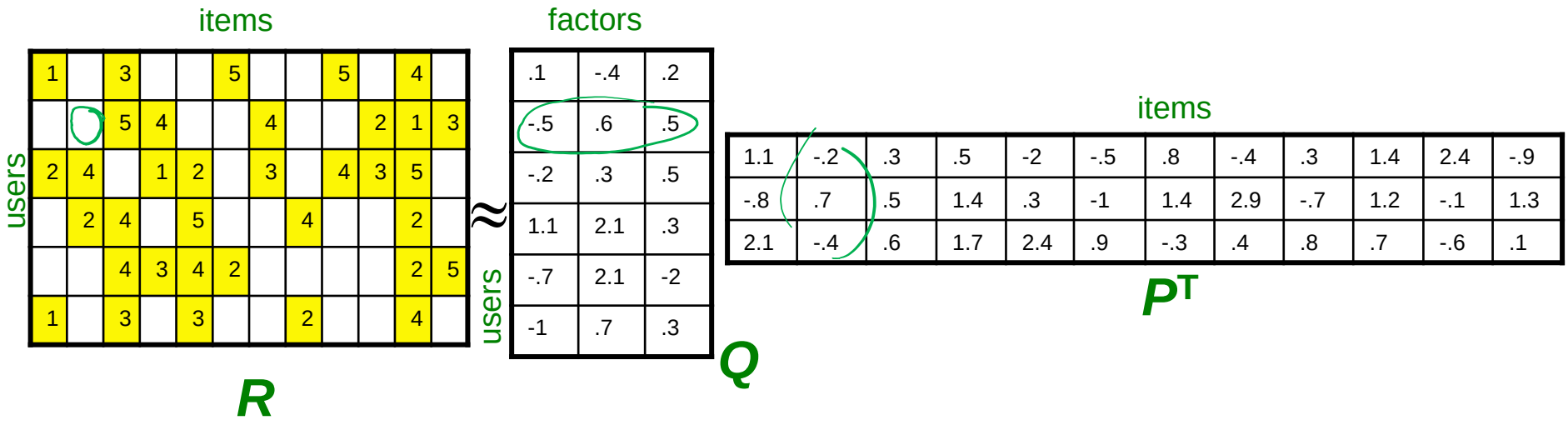


R = set of tuples (i, x) of users x that have rated item i with a rating of r_{xi}

$$\text{RMSE} = \frac{1}{|R|}\sqrt{\sum_{(i,x)\in R}(\hat{r}_{xi} - r_{xi})^2}$$

"SVD" on Rating Data

- Goal: Make good recommendations
  - Lower RMSE ⇒ better recommendation
  - Design a machine learning model that performs well on the observed (user,item) ratings -- and generalizes to the unseen test data
- SVD seems like a solution! → optimal reconstruction error
  - "SVD" on rating data: $R \approx Q \cdot P^T$ // SVD: A = U Σ V^T → Q = U Σ



SVD on Rating Data

- Known: SVD gives **minimum reconstruction error** (Sum of Squared Errors):

$$\min_{U,V,\Sigma} \sum_{ij \in A} \left(A_{ij} - \left[U \Sigma V^{\mathrm{T}} \right]_{ij} \right)^2$$

  - SSE and RMSE are monotonically related:
    - $RMSE = \frac{1}{c}\sqrt{SSE}$
    - Great news: SVD is minimizing RMSE

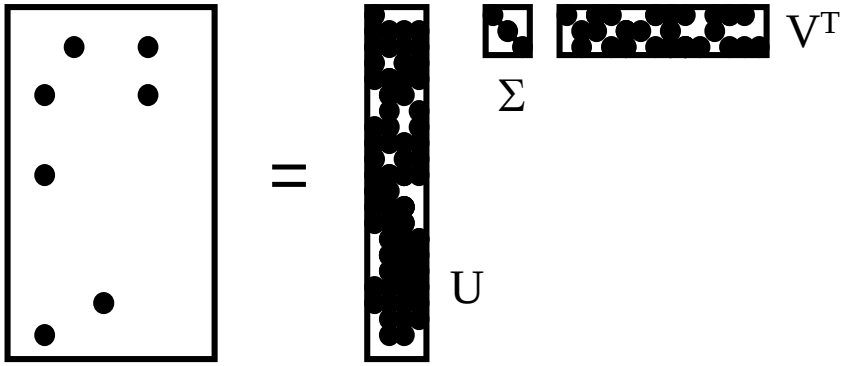
- **Complication:**
  - The sum in the SVD error term is over <u>all</u> entries (no-rating = zero-rating)
  - But our R has <u>missing</u> entries!
  - (Classical) SVD isn't defined when entries are missing!
- Also: We actually don't care about orthogonality and normalization

Discussion: SVD/PCA

+ **Optimal low-rank approximation**
  - In terms of Frobenius norm (and also in terms of the spectral norm)

– **Missing elements (= we have no information/not observed) are treated as 0 (= a very low rating)**
  - Critical for many applications (e.g. recommender systems)
  - General problem: two kinds of "zeros" (not observed/missing != 0)

– **Lack of Sparsity**
  - Singular vectors are **dense!**
  - Potential interpretability problem



– **Orthogonality really required/useful?**

Latent Factor Models

$(s - x \cdot y)^2$

- Our goal: Find Q $\in \mathbb{R}^{n \times k}$ and $P \in \mathbb{R}^{d \times k}$ such that:

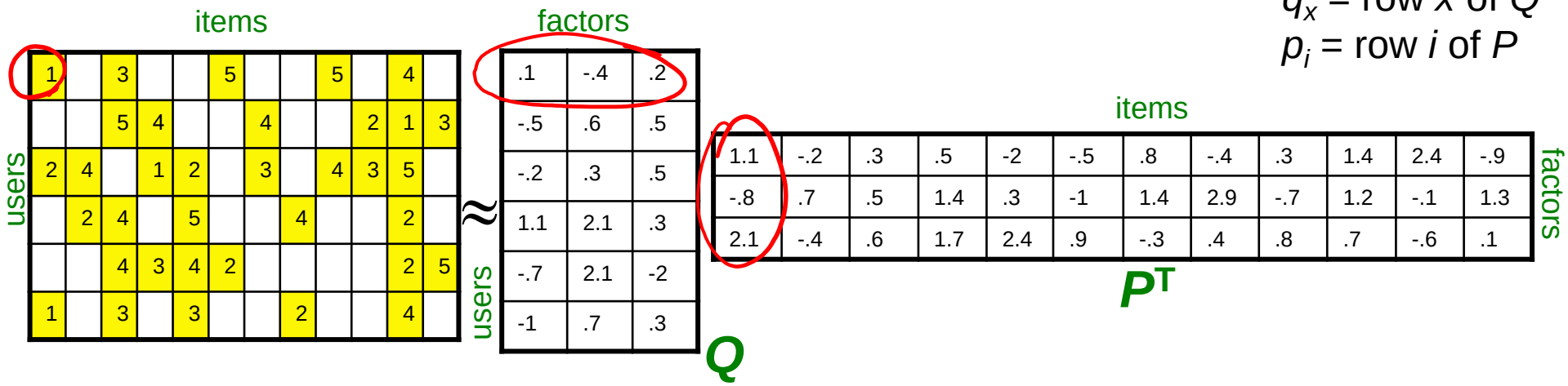
$$\min_{P,Q} \sum_{(i,x) \in R} \left(r_{xi} - q_x \cdot p_i^{\mathrm{T}}\right)^2$$

  - Note 1: We only sum over existing entries *R*
  - Note 2: We don't require columns of P, Q to be orthogonal or unit length

➢ Use optimization techniques to solve this problem

$\hat{r}_{xi} = q_x \cdot p_i^T$

q_x = row x of Q
p_i = row i of P



Finding the Latent Factors (I)

- Goal: $\min_{P,Q} \sum_{(i,x)\in R} \left(r_{xi} - q_x \cdot p_i^{\mathrm{T}}\right)^2 =: \min_{P,Q} f(P,Q)$

- One approach: **Alternating Optimization** // a.k.a. block coordinate minimization
  - Pick initial values for P and Q
  - Alternatingly keep one variable fix and optimize for the other; repeat until convergence

- Pseudo-Code:
  1. initialize $P^{(0)}$, $Q^{(0)}$, t=0
  2. $\mathrm{P}^{(t+1)} = argmin_P \ f(P, \mathrm{Q}^{(\mathrm{t})})$
  3. $\mathrm{Q}^{(t+1)} = argmin_Q \ f\left(P^{(t+1)}, Q\right)$
  4. t=t+1
  5. goto 2 until values convergence

Finding the Latent Factors (II)

- Initialization of P and Q:
  - Use, e.g., SVD where missing entries are replaced by 0 (or overall mean)

- How to solve $P^{(t+1)} = \underset{P}{argmin}\, f(P, Q^{(t)}) = \underset{P}{argmin} \sum_{(i,x)\in R} \left(r_{xi} - q_x \cdot p_i^T\right)^2$
  - *Observation 1:* since Q is fixed, the problem can be solved **for each vector $\boldsymbol{p_i}$ independently**

$$\min_P \sum_{(i,x)\in R} \left(r_{xi} - q_x \cdot p_i^{\mathrm{T}}\right)^2 = \sum_{i=1\ldots d} \min_{p_i} \sum_{x\in R_{*,i}} \left(r_{xi} - q_x \cdot p_i^{\mathrm{T}}\right)^2$$

where $R_{*,i} = \{x \mid (i,x) \in R\}$ // = users who have rated item i

  - Equivalently for vector q_x based on the set $R_{,x*} = \{i \mid (i,x) \in R\}$

Finding the Latent Factors (III)

- *Observation 2:* $\min_{p_i} \sum_{x \in R_{*,i}} \left(r_{xi} - q_x \cdot p_i^T\right)^2$ is an **ordinary least squares regression** problem
  - $\min_b \sum_{j=1}^{n} \left(y_j - x_j^T b\right)^2$ // y_j is scalar, x_j and b (column) vectors
  - Optimal solution in closed form: $b = \left(\frac{1}{n}\sum_{j=1}^{n} x_j x_j^T\right)^{-1} \cdot \frac{1}{n}\sum_{j=1}^{n} x_j y_j$
  - In our case: $p_i^T = \left(\frac{1}{|R_{*,i}|}\sum_{x \in R_{*,i}} q_x^T q_x\right)^{-1} \cdot \frac{1}{|R_{*,i}|}\sum_{x \in R_{*,i}} q_x^T r_{xi}$
  - Equivalently for q_x

- Computation of $\underset{P}{argmin}(P, Q^{(t)})$ reduces to a standard optimization problem

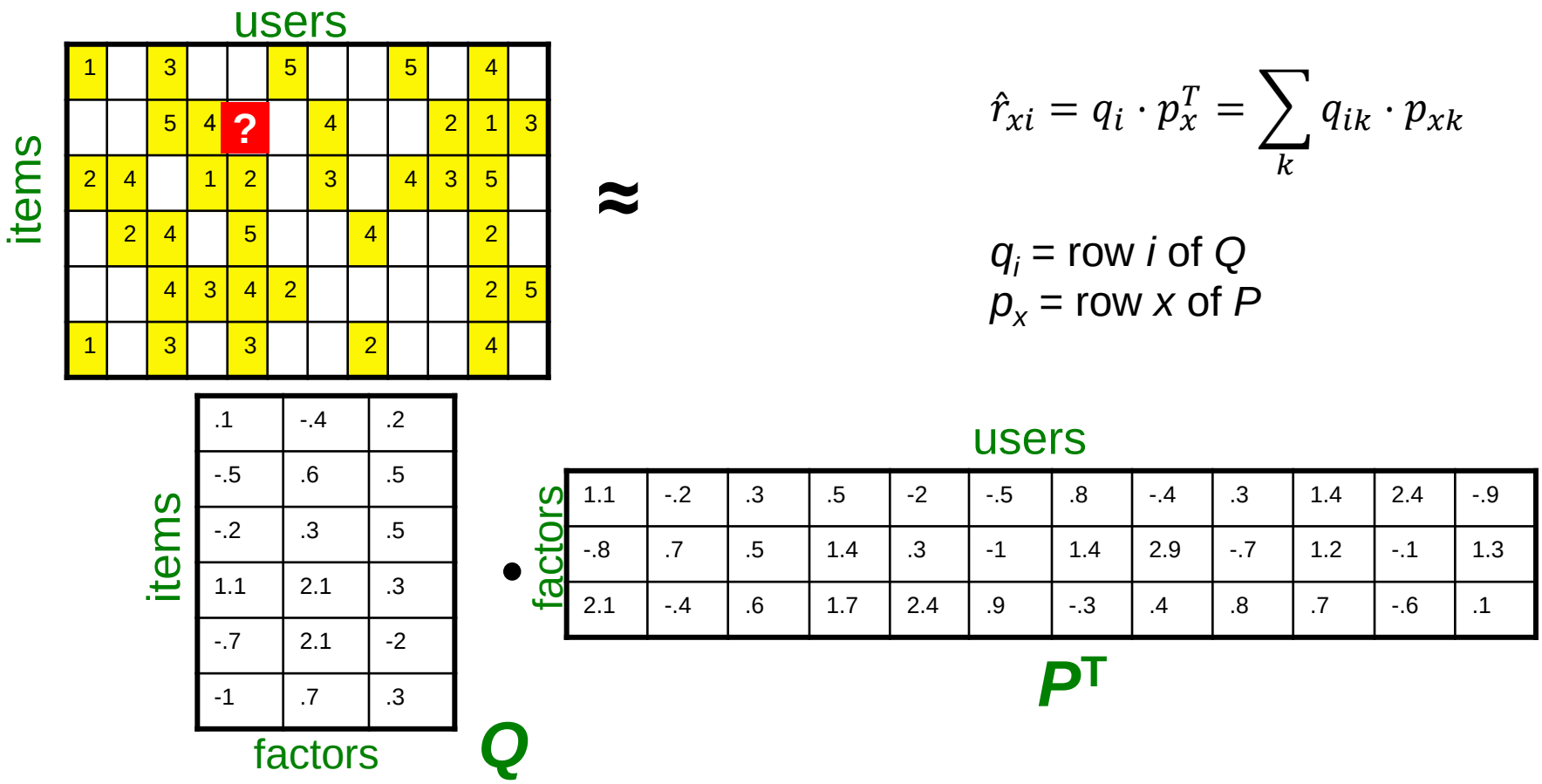
Alternating Optimization: Discussion

- Alternating optimization may provide solution to complex optimization problems
- Here: **Sequence of simple OLS problems**
  - Overall algorithm can be implemented in a few lines in, e.g., Matlab
  - Quite efficient: since data is sparse, the sets $R_{*,i}$ (and $R_{x,*}$) are relatively small

- **Drawback** of Alternating Optimization:
  - Solution is only an approximation
  - No guarantee that close to the optimal solution
  - Highly depends on initial solution

Rating Prediction

- We succeeded in learning Q and P!
- Next: How to estimate the missing rating of user x for item i?

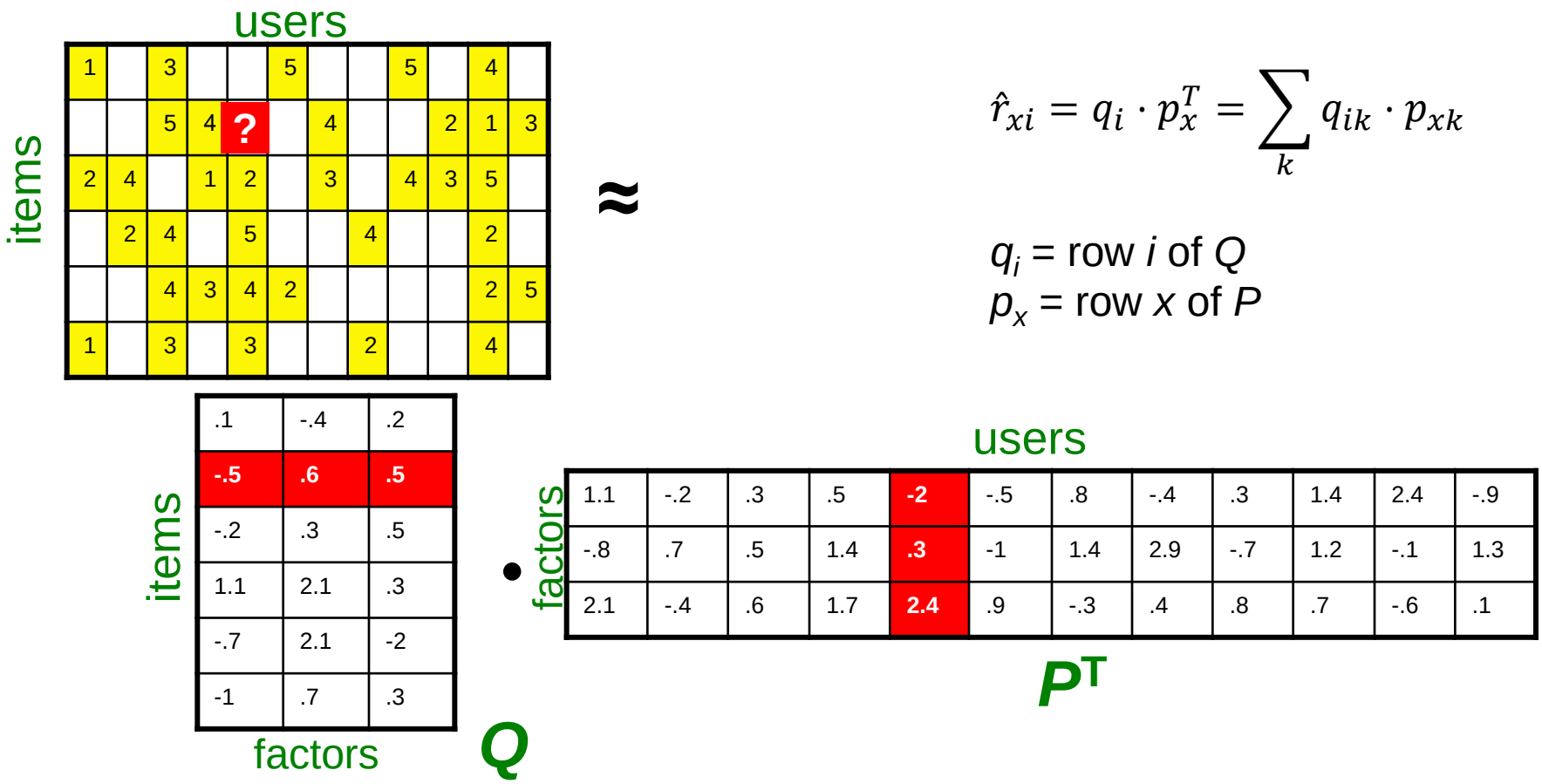


$$\hat{r}_{xi} = q_i \cdot p_x^T = \sum_k q_{ik} \cdot p_{xk}$$

q_i = row i of Q
p_x = row x of P

Rating Prediction

- We succeeded in learning Q and P!
- Next: How to estimate the missing rating of user x for item i?

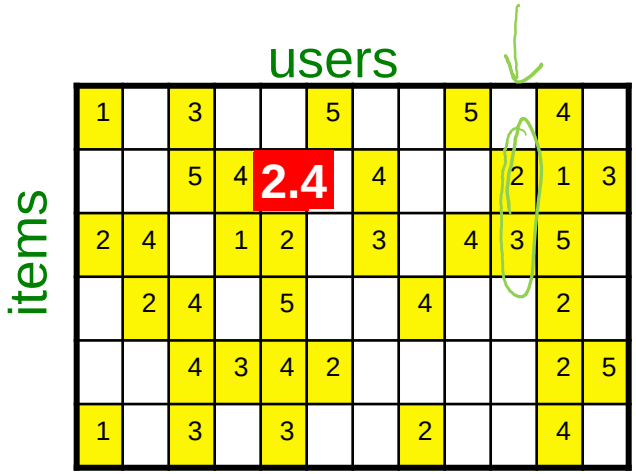


$$\hat{r}_{xi} = q_i \cdot p_x^T = \sum_k q_{ik} \cdot p_{xk}$$

q_i = row i of Q
p_x = row x of P

Rating Prediction

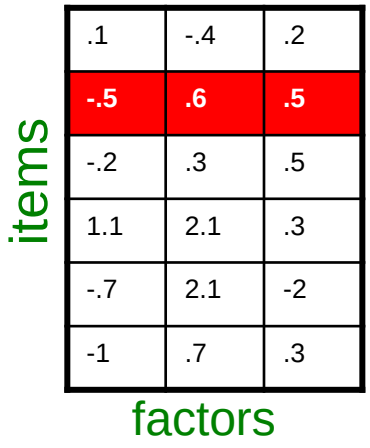
- We succeeded in learning Q and P!
- Next: How to estimate the missing rating of user x for item i?



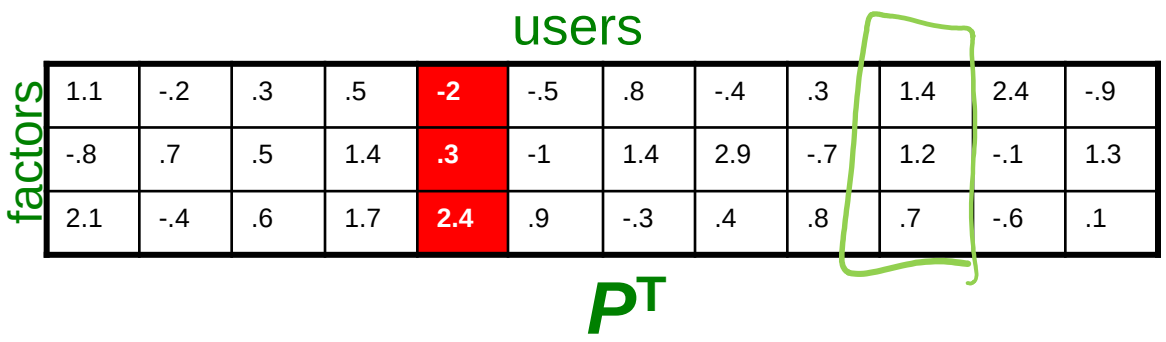
≈

$$\hat{r}_{xi} = q_i \cdot p_x^T = \sum_k q_{ik} \cdot p_{xk}$$

q_i = row i of Q
p_x = row x of P



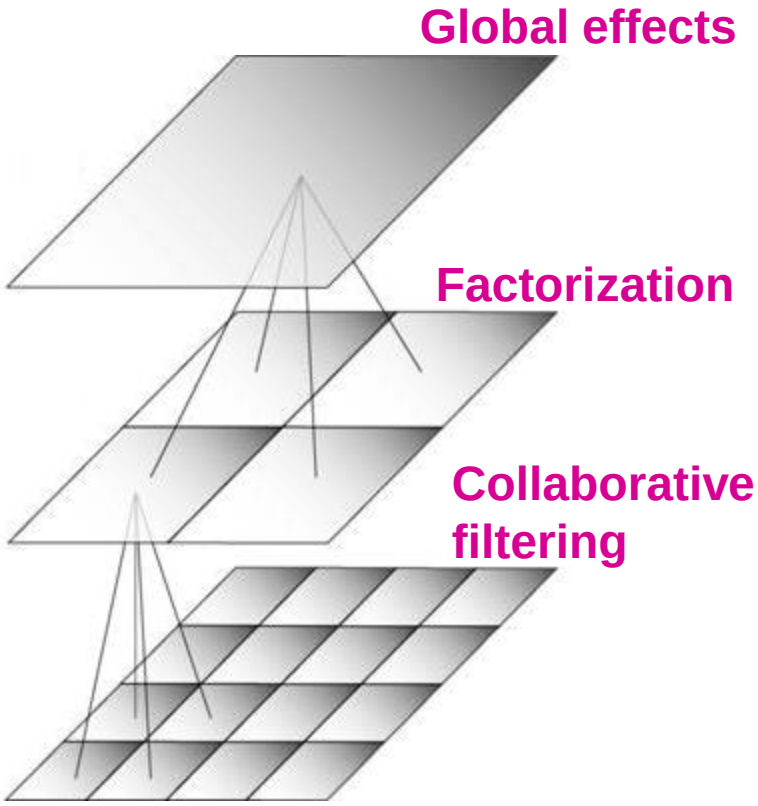
•



Side-note: BellKor Recommender System

- The winner of the Netflix Challenge uses matrix factorization as **one** building block
  - The overall model is a combination of multiple ideas

- Multi-scale modeling of the data: Combine top level, "regional" modeling of the data, with a refined, local view:
  - **Global**:
    - Overall deviations of users/movies
  - **Factorization**:
    - Addressing "regional" effects
  - **Collaborative filtering**:
    - Extract local patterns

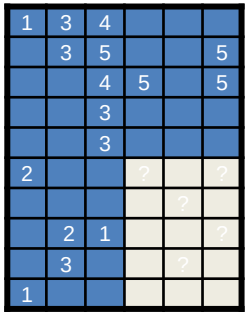


Roadmap

- Chapter: Dimensionality Reduction & Matrix Factorization
  1. Introduction
  2. Principal Component Analysis (PCA)
  3. Probabilistic PCA (PPCA)
  4. Singular Value Decomposition (SVD)
  5. **Matrix Factorization**
     - Motivation & Approach
     - **Regularization & Sparsity**
     - Further Factorization Models
  6. Autoencoders (Non-linear Dimensionality Reduction)

Challenge: Overfitting

- Final Goal: Minimize SSE for **unseen test data**
- Proxy: Minimize SSE on **training data**
  - Want large k (# of factors) to capture all the signals
  - But, SSE on **test data** begins to rise for larger k
  - Why?



- Classical example of overfitting:
  - With too many degrees freedom (too many free parameters) the model starts fitting noise
  - The model fits too well the training data but does not generalize well to unseen test data

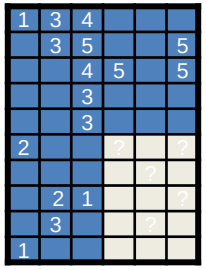
Challenge: Overfitting

- Problem can easily be seen in our scenario:
  - In each step of the *alternating optimization* we solve an OLS regression
    - Number of regression parameters = k = number of latent factors
    - Number of data points used for regression = cardinality of $R_{*,x}$ / $R_{i,*}$
  - Lots of parameters but not enough data points → regression can overfit
  - ➢ If k is large this might even lead to an **underdetermined** system of equations
  - ➢ Problem is ill-posed

- Solution: **Regularization**
  - Interpretation for underdetermined systems: "In the absence of any other information, the parameter vector should be small (i.e. only small effect of features)“
  - Equivalently: We put a prior on the weights

Regularization

- To solve overfitting we introduce regularization:
  - Allow rich model where we have sufficient data
  - Shrink aggressively where data are scarce



$$\min_{P,Q} \underbrace{\sum_{training} (r_{xi} - q_i p_x)^2}_{\text{“error”}} + \underbrace{\left[\lambda_1 \sum_x \|p_x\|^2 + \lambda_2 \sum_i \|q_i\|^2 \right]}_{\text{“length”}}$$

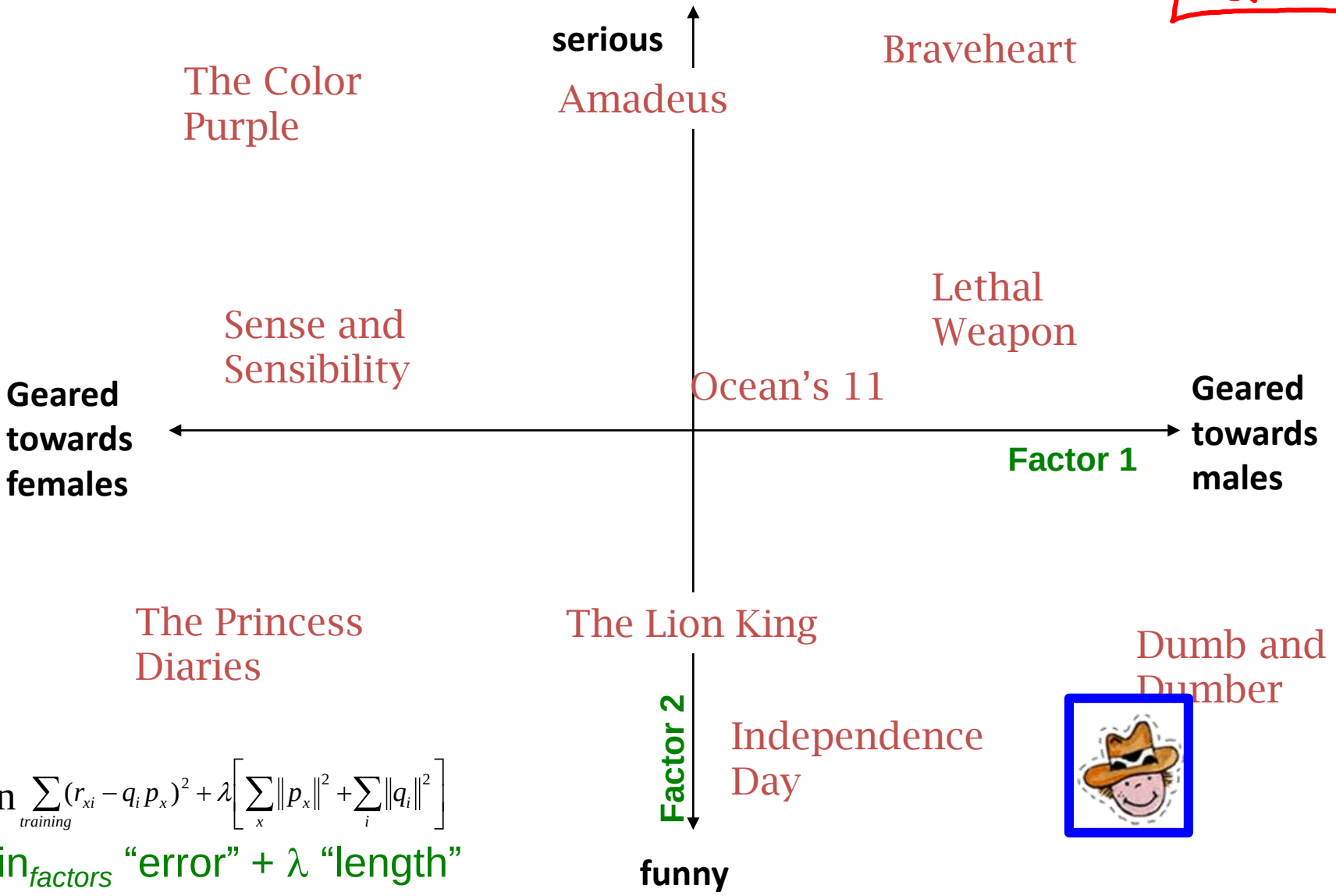
λ_1, λ_2 … user defined regularization parameters

- Note: We do not care about the actual value of the objective function, but we care in the P,Q that achieve the minimum of the objective

The Effect of Regularization

$A = U \Sigma V^T$

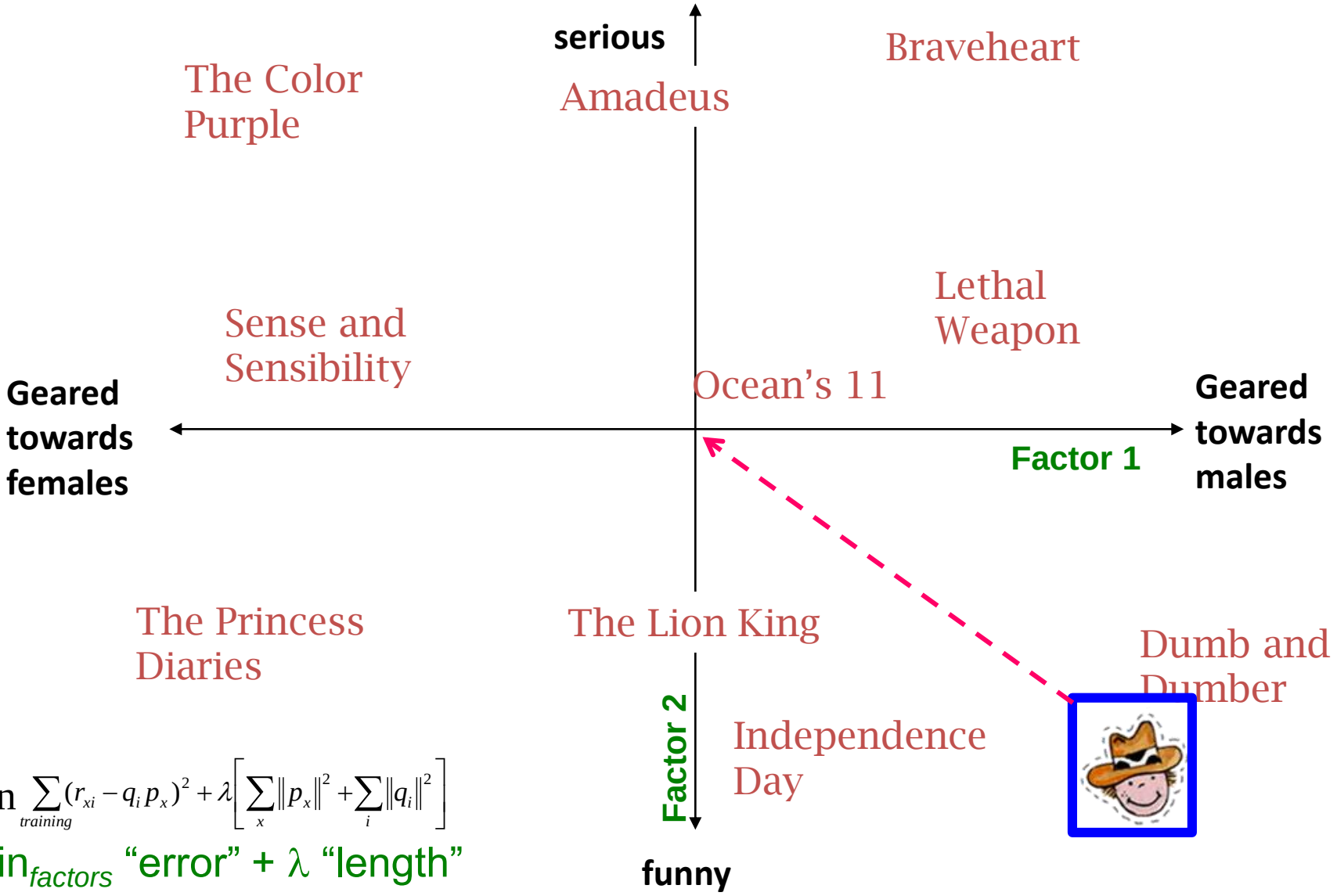
$\rightarrow A \cdot V = U \Sigma$



$$\min_{P,Q} \sum_{training} (r_{xi} - q_i p_x)^2 + \lambda \left[\sum_x \|p_x\|^2 + \sum_i \|q_i\|^2 \right]$$

$\min_{factors}$ "error" + λ "length"

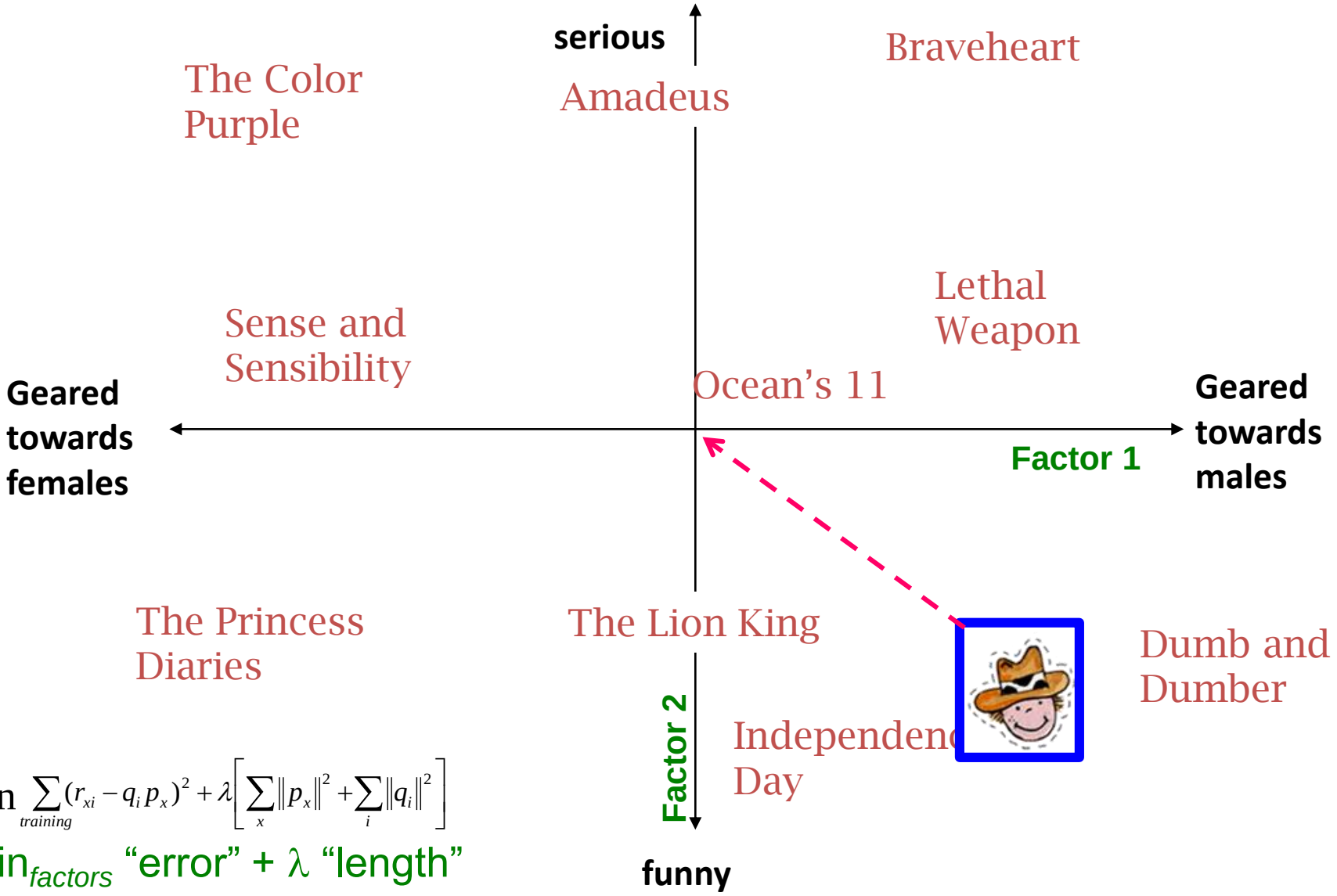
The Effect of Regularization



$$\min_{P,Q} \sum_{training} (r_{xi} - q_i p_x)^2 + \lambda \left[\sum_x \|p_x\|^2 + \sum_i \|q_i\|^2 \right]$$

$\min_{factors}$ "error" + λ "length"

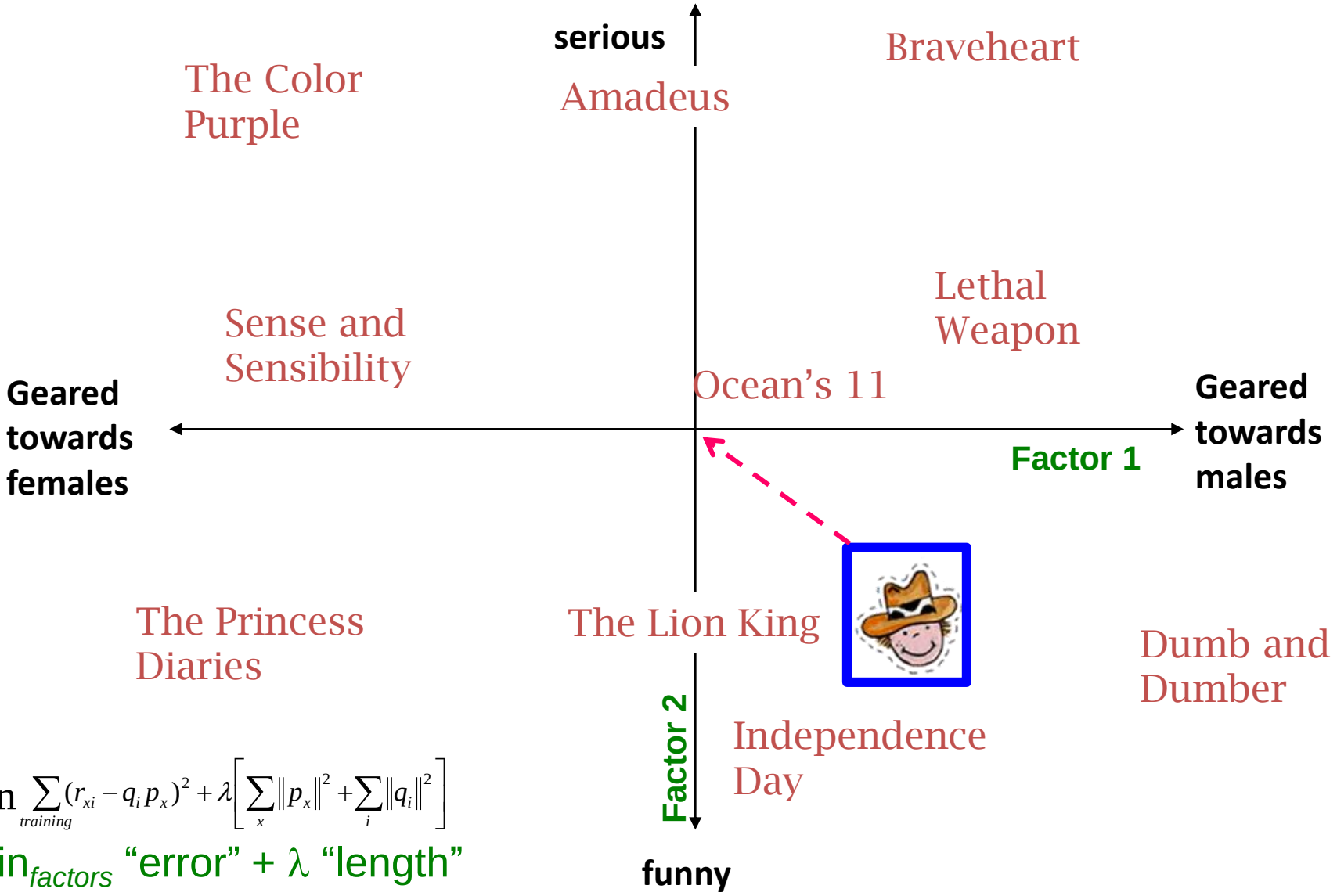
The Effect of Regularization



$$\min_{P,Q} \sum_{training} (r_{xi} - q_i p_x)^2 + \lambda \left[\sum_x \|p_x\|^2 + \sum_i \|q_i\|^2 \right]$$

$\min_{factors}$ "error" + λ "length"

The Effect of Regularization



$$\min_{P,Q} \sum_{training} (r_{xi} - q_i p_x)^2 + \lambda \left[\sum_x \|p_x\|^2 + \sum_i \|q_i\|^2 \right]$$

$\min_{factors}$ "error" + λ "length"

Regularization in Our Use Case

- Regularized Problem: $\min_{P,Q} \sum_{training} (r_{xi} - q_i p_x)^2 + \left[\lambda_1 \sum_x \|p_x\|^2 + \lambda_2 \sum_i \|q_i\|^2 \right]$

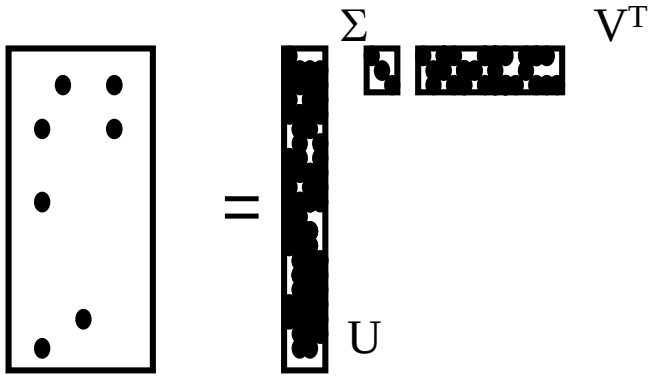
- Recap of unregularized problem: In each iteration of the alternating optimization we solved an OLS regression problem
- For the regularized version this becomes
  - $\min_{p_x} \sum_{i \in R_{*,x}} \left(r_{xi} - q_i \cdot p_x^T\right)^2 + \lambda_1 \|p_x\|^2$
  - a.k.a. **ridge regression**
- Effect: parameter values are forced to become smaller
  - Large values that only capture noise are avoided

- You know how to solve this!
  - Simple solution: gradient descent

L2 vs. L1 Regularization

- Comparison: L2 vs. L1 regularization
  - L2 tries to "shrink" the parameter vector b **equally**
    - Large values are highly penalized due to the square in the L2 norm
    - Unlikely that any component will be **exactly 0**
  - L1 allows large values in individual components of b by shrinking other components to 0
  - ➢ L1 is suited to enforce **sparsity** of the parameter vector

- Why sparsity?
  - Better interpretation
    - Only few values contribute to result
    - Unintuitive that **sparse input data** is generated based on **dense signal**
  - Less storage, faster processing



Roadmap

- Chapter: Dimensionality Reduction & Matrix Factorization
  1. Introduction
  2. Principal Component Analysis (PCA)
  3. Probabilistic PCA (PPCA)
  4. Singular Value Decomposition (SVD)
  5. **Matrix Factorization**
     - Motivation & Approach
     - Regularization & Sparsity
     - **Further Factorization Models**
  6. Autoencoders (Non-linear Dimensionality Reduction)

Further Factorization Models

- Matrix factorization is extremely powerful
  - Dimensionality reduction, data analysis/data understanding, prediction of missing values, …
- Various extensions have been proposed
  - Enforcing different constraints or operating on different data types
  - Important goal: better interpretation of result

Non-Negative Matrix Factorization

- Often data is given in form of non-negative values
  - Rating values between 0 to 5; income, age, … of persons; number of words in a document; grayscale images; etc.
- However: SVD (and the other approaches presented before) might lead to factors containing **negative values**
  - Difficult to interpret: non-negative data is "generated" based on negative factors; what do these negative factors mean?
  - Predicted values might also become negative
- Solution: Non-Negative Matrix Factorization

Non-Negative Matrix Factorization

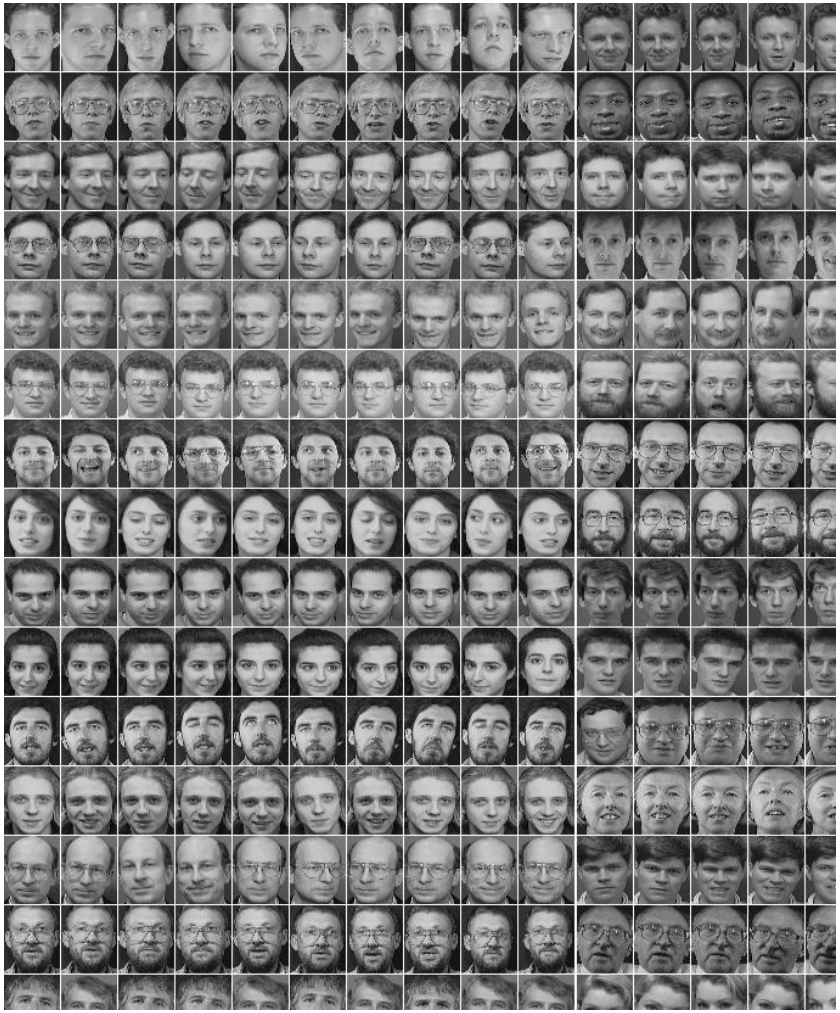
- Task: Factorize non-negative A in non-negative P and Q, i.e. $A \approx P \cdot Q$

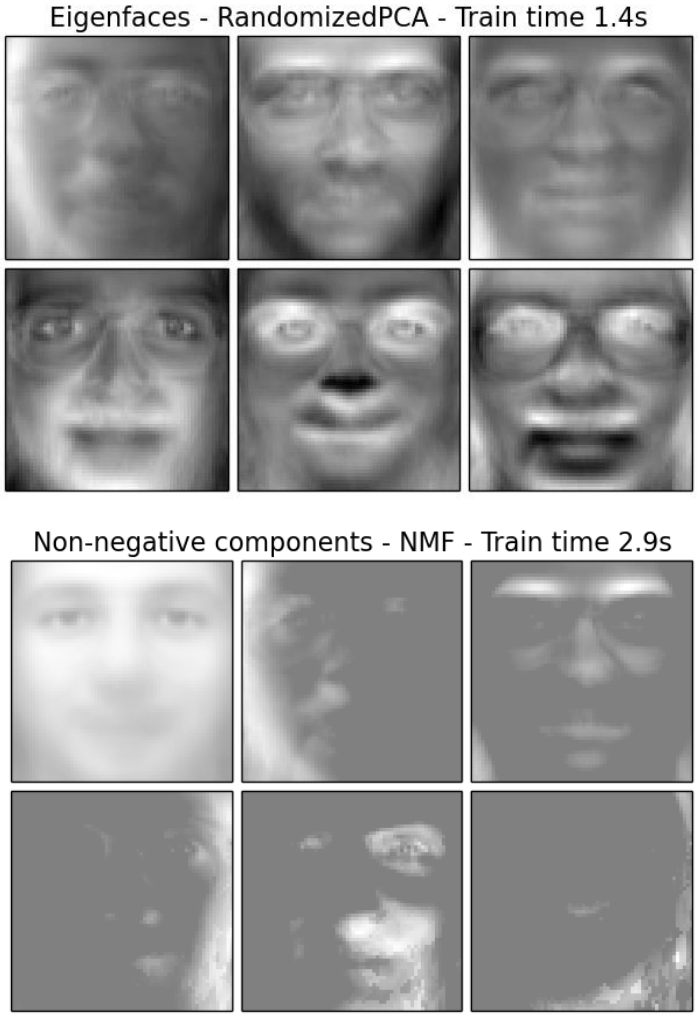
- Formally:
  - Given $A \in R^{n \times d}$ with $A \geq 0$ and integer k, find $P \in R^{n \times k}, Q \in R^{k \times d}$ such that $\|A - P \cdot Q\|_F$ is minimized subject to $P \geq 0$ and $Q \geq 0$

$$\min_{P \geq 0, Q \geq 0} \|A - P \cdot Q\|_F$$

  - ➢ Constrained optimization

NNMF Example: Olivetti Faces Data





results according to scikit-learn

Boolean Matrix Factorization

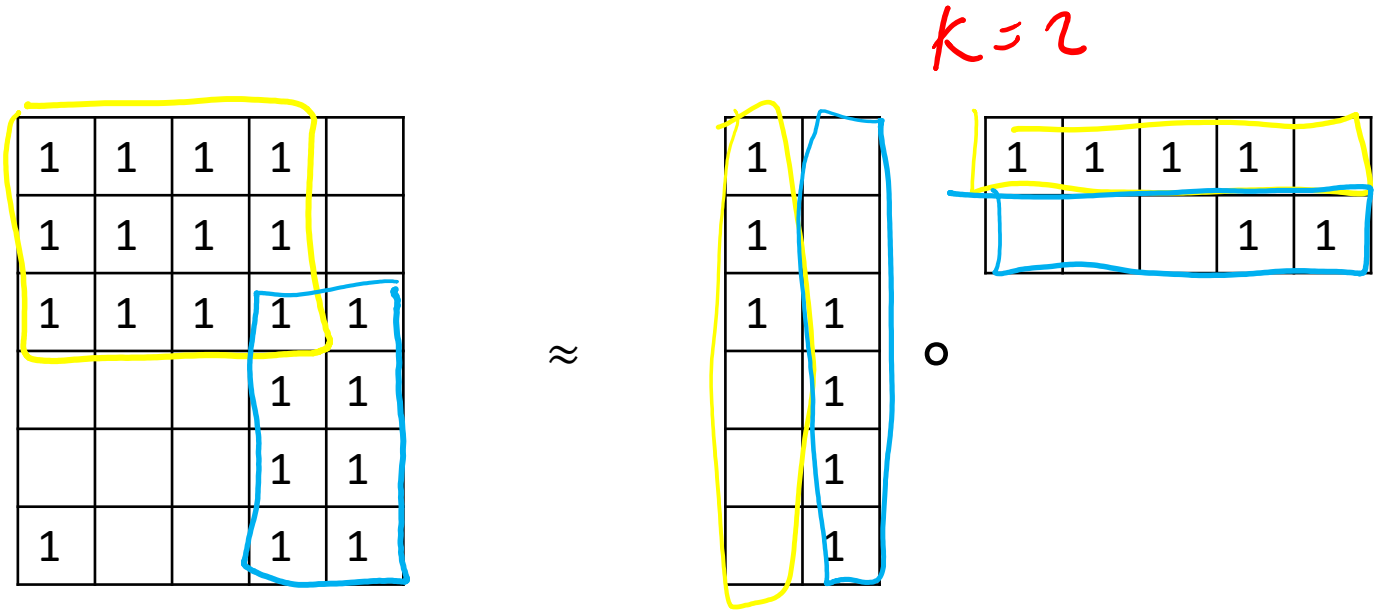
- Special case: Binary data
  - e.g. indicator matrices: objects having certain properties; users and products they bought
- Factorization based on "usual" linear algebra leads to factors containing **numerical values**
  - What do such values represent for binary data?

1	1	1	
1	1		1
		1	1
		1	1
1		1	1

- Alternative: **Boolean Matrix Factorization**
  - Exploits the idea of Boolean algebra (1+1 = 1)
    - i.e. "+" becomes "or" and "·" becomes "and"; 1=true, 0=false
    - Denote with $X \circ Y$ the adapted matrix multiplication using these rules

Boolean Matrix Factorization

- Task: Factorize Boolean A in Boolean B and C, i.e. $A \approx B \circ C$
- Formally:
  - Given $A \in \{0,1\}^{n \times d}$ and integer k, find $B \in \{0,1\}^{n \times k}$ and $C \in \{0,1\}^{k \times d}$ such that $|A - B \circ C|$ is minimized
- NP-hard to optimize → approximate solutions required



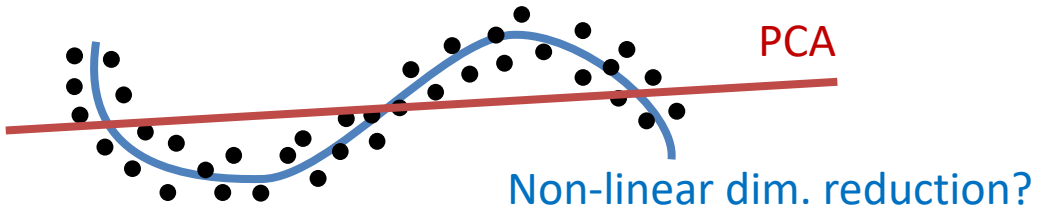
Roadmap

- Chapter: Dimensionality Reduction & Matrix Factorization
  1. Introduction
  2. Principal Component Analysis (PCA)
  3. Probabilistic PCA (PPCA)
  4. Singular Value Decomposition (SVD)
  5. Matrix Factorization
  6. **Autoencoders (Non-linear Dimensionality Reduction)**

Motivation

- PCA / SVD can only capture linear structure (linear variations in the data)
  - Recall: transformed data is $Y = \tilde{X} \cdot \Gamma$
  - Linear projection by the eigenvectors Γ

- However, data often lies on a non-linear low-dimensional manifold



- Idea: find a non-linear projection of the data

Autoencoders

- An **autoencoder** is a neural network that:
  - finds a compact representation of the data
  - by learning to reconstruct its input

$$f(\boldsymbol{x}, \boldsymbol{W}) := \hat{\boldsymbol{x}} \approx \boldsymbol{x}$$

- Goal: minimize the reconstruction error between $\hat{\boldsymbol{x}}$ and $\boldsymbol{x}$

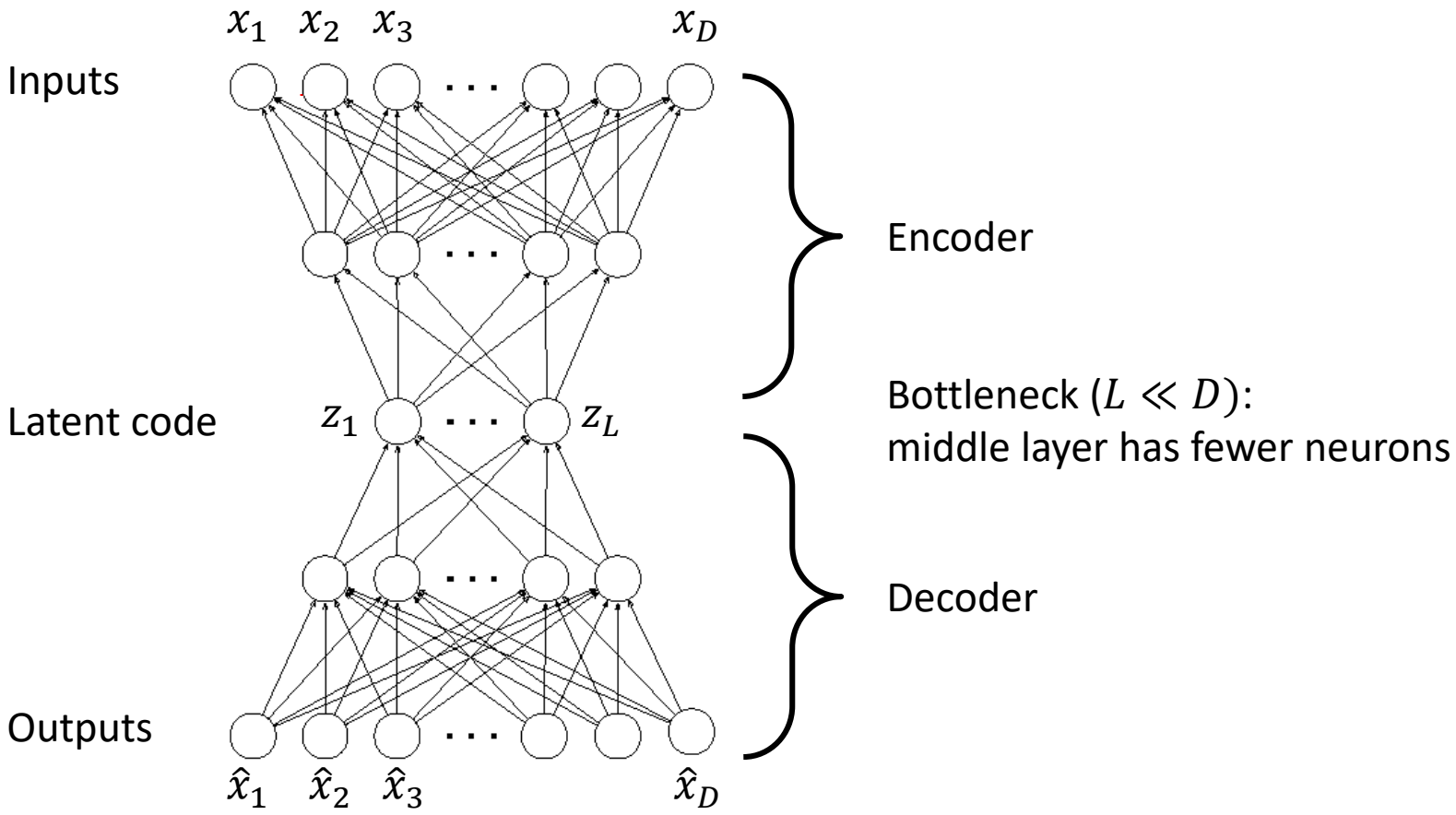
$$\min_{\boldsymbol{W}} \frac{1}{N} \sum_{i=1}^{N} \|f(\boldsymbol{x}_i, \boldsymbol{W}) - \boldsymbol{x}_i\|^2$$

- Alternative view: find a **latent representation** $\boldsymbol{z} \in \mathbb{R}^L$ which is a compact code for $\boldsymbol{x} \in \mathbb{R}^D$ since $L \ll D$
  - $f_{enc}(\boldsymbol{x}) = \mathbf{z}$ # encoder: project the data to a lower dimension
  - $f_{dec}(\boldsymbol{z}) \approx \boldsymbol{x}$ # decoder: reconstruct the data from the latent code

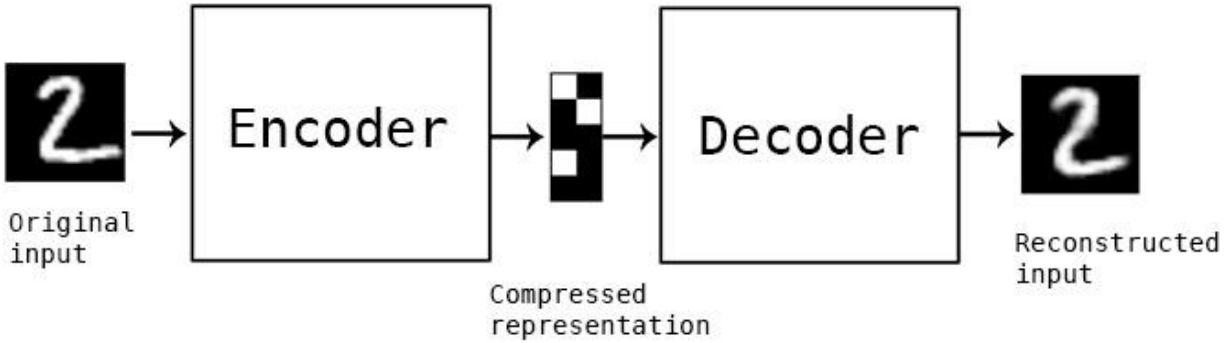
$$f_{dec}\big(f_{enc}(\boldsymbol{x})\big) \approx \boldsymbol{x}$$

Autoencoders

- f_{enc} and f_{dec} are non-linear functions implemented by a neural network



Autoencoders: Example



Original image:

Reconstruction:

Autoencoders

- An autoencoder whose code dimension is less than the input dimension ($L \ll D$) is called **undercomplete**
  - Forces the autoencoder to capture the most salient features of the data
  - $L \geq D$ (**overcomplete**) the autoencoder can simply learn the identity function

- Training autoencoders in practice:
  - Add a regularization term to the SSE loss to prevent overfitting
  - Weight tying: f_{enc} and f_{dec} share the same weights

- Other extensions:
  - Denoising autoencoders (DAEs): receive a corrupted (noisy) training data point as input and predict the "clean" uncorrupted data point as output
  - Variational autoencoders (potentially later in this lecture)

(Linear) Autoencoders & PCA: Comparison

LINEAR AUTOENC IS EQUIVALENT TO PCA

- What if f_{enc} and f_{dec} are linear functions?
  - $f_{enc}(\boldsymbol{x}, \boldsymbol{W}_1) = \boldsymbol{x}\boldsymbol{W}_1$ # $\boldsymbol{W}_1 \in \mathbb{R}^{D \times L}$
  - $f_{dec}(\boldsymbol{z}, \boldsymbol{W}_2) = \boldsymbol{z}\boldsymbol{W}_2$ # $\boldsymbol{W}_2 \in \mathbb{R}^{L \times D}$

NOT PCA → $f_{ENC}(x) = \sigma(x \cdot W_1)$, $f_{DEC}(z) = z \cdot W_2$

- We have: $f_{dec}\big(f_{enc}(\boldsymbol{x})\big) = \boldsymbol{x}\boldsymbol{W}_1\boldsymbol{W}_2$ and

$$\min_{\boldsymbol{W}_1\boldsymbol{W}_2} \frac{1}{N}\sum_{i=1}^{N} \|f(\boldsymbol{x}_i, \boldsymbol{W}) - \boldsymbol{x}_i\|^2 = \min_{\boldsymbol{W}_1\boldsymbol{W}_2} \frac{1}{N}\sum_{i=1}^{N} \|\boldsymbol{x}_i\boldsymbol{W}_1\boldsymbol{W}_2 - \boldsymbol{x}_i\|^2 = \min_{\boldsymbol{W}} \frac{1}{N}\sum_{i=1}^{N} \|\boldsymbol{x}_i\boldsymbol{W} - \boldsymbol{x}_i\|^2$$

Equivalent: $\boldsymbol{W}_1\boldsymbol{W}_2 = \boldsymbol{W}$ s.t. rank of $\boldsymbol{W}$ is L

- Optimal solution (assuming normalization):
  - PCA: $\boldsymbol{W}^* = \boldsymbol{\Gamma}$ where $\boldsymbol{\Gamma}$ are the (top-L) eigenvectors of $\boldsymbol{X}^T\boldsymbol{X}$

Summary

- Dimensionality Reduction and Matrix Factorization are highly related
  - And can be used for various purposes
- PCA, SVD (and linear Autoencoders) are equivalent
  - Optimal low-rank approximation
- Matrix factorization for rating prediction
  - Very general formulation: allows to handle, e.g., missing entries
- Autoencoders perform non-linear dimensionality reduction
- Why are these techniques useful?
  - Less storage required
  - More efficient processing of the data
  - Remove redundant and noisy features
  - Discover hidden correlations/topics/concepts
  - Interpretation and visualization