JOHANNES KEPLER UNIVERSITÄT LINZ

Institute of Computational Perception

MULTIMEDIA MINING AND SEARCH GROUP

UE Learning from User-generated Data S2025

Ex 3: Model-based approaches

Oleg Lesota (oleg.lesota@jku.at)

Shahed Masoudian (shahed.masoudian@jku.at)

Antonela Tommasel (antonela.Tommasel@jku.at)

Agenda

- Exercise 2: Questions?
- Matrix factorization
- Exercise 3: Model-based approaches

Exercise 2

- How is it going?
- Any difficulties?

- Questions?

High dimensional interaction matrices

- Ratings from 1 to 5
- **17K movies**
- **99M ratings**
- **480K viewers**

- 206 movies seen on average
- Average rating of 3.6
- **9B entries but only 99M are non-zero**

High dimensional interaction matrices

- Ratings from 1 to 5
- **17K movies**
- **99M ratings**
- **480K viewers**

- 206 movies seen on average
- Average rating of 3.6
- **9B entries but only 99M are non-zero**

- **Data sparsity challenges.**
  - Large item sets, user ratings just for a small percentage of them.
- **Scalability challenges.**
  - NN requires computations that grows with both the number of users and items.
- **Performance implications.**
  - Item similarities are static (can be pre-computed).
  - User similarities are dynamic (precomputing neighbours can lead to poor predictions).

Dimension reduction

Reducing the number of variables (features) in a dataset while retaining as much important information as possible.

Memory-based approaches like ItemKNN require **storing large interaction matrices**, which can be **inefficient**.

Model-based approaches, such as Matrix Factorization, offer a **more compact and efficient representation** of interaction matrices.

Dimension reduction

Reducing the number of variables (features) in a dataset while retaining as much important information as possible.

Memory-based approaches like ItemKNN require **storing large interaction matrices**, which can be **inefficient**.

Model-based approaches, such as Matrix Factorization, offer a **more compact and efficient representation** of interaction matrices.

Why is it important?

- **Efficiency**: Reduces memory usage and computational load by representing large data in fewer dimensions.
- **Noise Reduction**: Helps to remove less relevant data (noise) and focus on the most important features.
- **Improved Accuracy**: Can help algorithms perform better by focusing on key factors that influence user-item interactions.

Matrix Factorization

- Represent users and items in a compact, latent vector space.

- Decompose the interaction matrix I into two smaller matrices:
  - U $(users \times f)$
  - V $(\text{items} \times f)$

- Reconstruction:
  - $inter[user, item] = U[user, :] \ @ \ V[item, :].T$
  - For example, dot product can be used in the reconstruction.

- More memory-efficient than storing the full interaction matrix.
- Faster computations for recommendations.
- Captures hidden patterns and dependencies in the data.
- Each dimension is called a “latent factor”, which “explain” the user x item interactions.

Singular Value Decomposition

- **Singular Value Decomposition (SVD)** is a mathematical technique used to factorize a matrix into three smaller matrices.

- The matrix I of dimensions $n \times m$ (where n represents the number of users and m the number of items) can be decomposed into three matrices:

$$\mathrm{I} = U\Sigma V^T$$

Represents items ($m \times m$). Orthogonal. Each rows corresponds to a latent factor of items.

Represents users ($n \times n$). Orthogonal. Each column corresponds to a latent factor of users.

Diagonal matrix with singular values on the diagonal. Values represent the importance or strength of each latent factor. The higher the more significant.

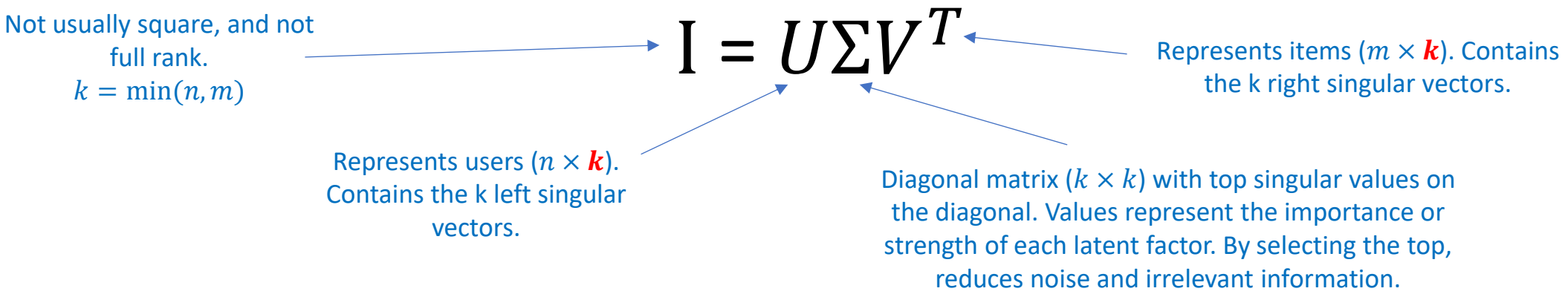
- The SVD represents the original matrix I as a combination of users' and items' latent factors. These factors capture patterns, similarities, or hidden relationships in the data.
- The product of the three matrices U, Σ and V^{T} reconstructs the original matrix I.

Orthogonal: $AA^T = A^T A = I$

(Thin) Singular Value Decomposition

- **Thin SVD** is a **reduced** version of SVD that focuses on the most important components, and it is especially useful for large, sparse matrices like those found in recommendation systems.

- The matrix I of dimensions $n \times m$ (where n represents the number of users and m the number of items) can be decomposed into three matrices:



- **SVD**: Computes a full decomposition, where both U and V are square matrices, and includes all singular values, which can be computationally expensive for large datasets.
- **Thin SVD**: Reduces the size of the matrices to k ($k = \min(n, m)$).

Full rank: The number of linear independent columns/rows matches the smallest matrix dimension.

Singular Value Decomposition

- **Singular Values** are the diagonal elements of the Σ matrix in SVD.
- These singular values are non-negative and are typically arranged in **descending order**, from the largest to the smallest.

$$\Sigma = \begin{bmatrix} 2.58 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.49 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.41 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.37 \end{bmatrix}$$

What Do Singular Values Represent?

- **Magnitude.** Singular values indicate the **weight** of the corresponding latent factors.
  - Larger singular values correspond to factors that capture the most significant patterns in the data (e.g., key features influencing user-item interactions).
- **Data Variance.** Singular values indicate how much of the original matrix's variance (or information) is captured by each latent factor.
  - How much variance is “situated” along each of the vectors.
  - Larger singular values account for more of the variation in the data.
- **Importance.** Singular values that are close to zero or very small typically represent noise or less significant patterns.
  - They can often be discarded in methods like Thin SVD to reduce the complexity of the data.

Truncated Singular Value Decomposition

- **Choosing Dimensions**:
  - We can choose $f < k$ dimensions to represent the entire data matrix (where f is the number of latent features, and k is the total number of dimensions).

  - By selecting the dimensions corresponding to the largest singular values, we ensure that we retain most of the important information while reducing the size of the matrix and potentially filtering out noise.

- **Matrix Shape Change**:
  - The matrices U and V^{T} become:
    - $U : n \times f$
    - $\mathrm{V}^{\mathrm{T}}: f \times m$
  - This reduction allows for faster computations and more efficient storage.

Truncated Singular Value Decomposition

- **Truncated SVD** is an **approximation** of the original matrix by retaining only the top k singular values (and their corresponding vectors).
  - Similar to Thin SVD, but with the goal of **approximating the original matrix**.

  - Using **only a subset** of the singular values (and their vectors) in order to reduce the dimensionality of the matrix while still capturing the most significant information.

  - Typically, **less accurate** than the full SVD (because we discard some information), but it often retains most of the important patterns in the data while reducing its size.

- **Compression.**
  - By using fewer latent factors, we create a compressed version of the original matrix, retaining the most important features for prediction.

- **Noise Reduction.**
  - Less significant singular values (which may represent noise or irrelevant information) are discarded, improving the model's accuracy and robustness.

Making recommendations with MF

- With **two sets of embeddings** U_{final} for users and V_{final} for items, we can generate recommendations by leveraging the latent vector space.
- Both users and items are now represented in the same f**-dimensional vector space**, allowing us to directly compute similarities between them.

- **Step-by-Step Process**:
  - **Dot Product for Similarity**:
    - To recommend items to a user u, we compute the **dot product** between the user's latent vector $U_{final}[u, :]$ and the corresponding item-vectors from V_{final} (e.g., $V_{final}[i, :]$, careful with matrix orientations).
    - It gives us an estimate of how much user u will interact with item i.
  - **Ranking Items.**
    - Create a list of all items ranked according to their dot product scores with the user's vector.
    - Items with higher scores are considered more relevant to the user.
  - **Filter Already Seen Items.**
    - Remove items that the user has already interacted with.
  - **Select Top K Recommendations**:
    - Finally, select the top-k items with the highest dot product scores to recommend to the user.

Summary

- Matrix Factorization reduces memory usage and computation time by using compact representations of users and items.
- SVD provides a mathematical foundation to decompose interaction matrices into smaller, interpretable matrices.
- By applying these methods, we can create efficient recommendation systems with reduced complexity.
- We use user and item embeddings in a shared latent space to compute similarities and make personalized recommendations using the dot product.
  - Top-k recommendations are generated by selecting the items with the highest similarity scores to a user's vector.

Exercise 3: Model-based approaches

Task 1. Matrix Factorization with SVD

- Implement the SVD decomposition.
- Use the SVD decomposition for making recommendations.

Task 2. Iterative Matrix Factorization with PyTorch

- Implement Matrix Factorization with PyTorch.
  - Helper code is available :)
  - Make recommendations with the trained model.

Questions so far?


NOTEBOOK

Thank you & Good luck!