Recommender systems

COLLABORATIVE FILTERING

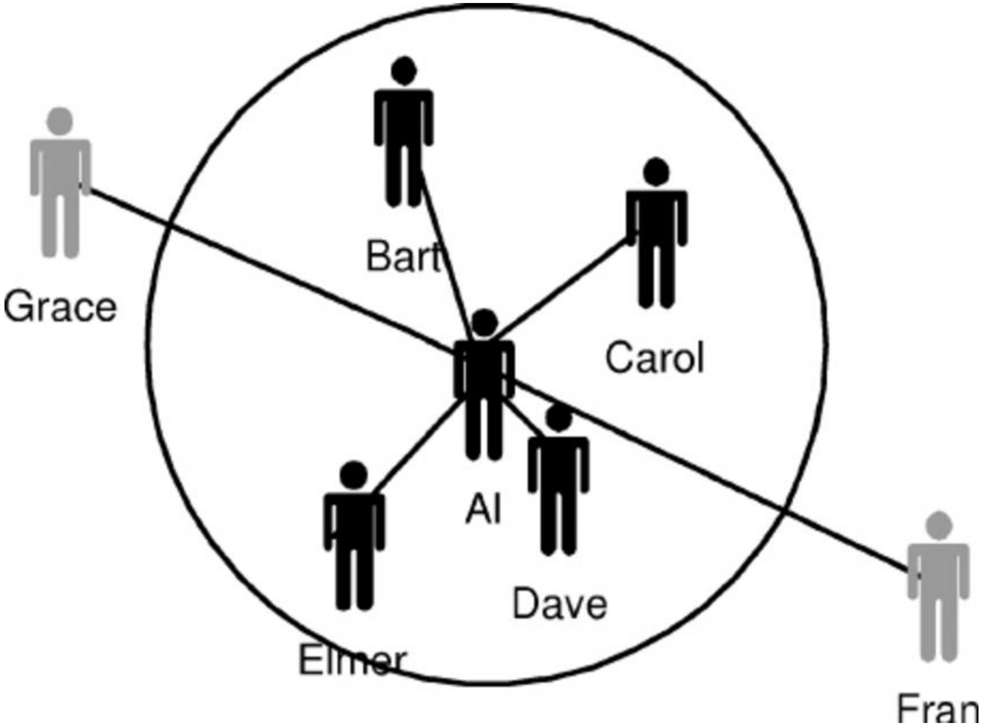
Nguyen Ngoc Thao
nnthao@fit.hcmus.edu.vn

Approaches to Collaborative filtering

- **k-Nearest Neighbors**
- Association rules
- Matrix factorization
- Deep learning

CF using k-Nearest Neighbors

- k-Nearest Neighbors recommendation uses the entire user-item database to make predictions without model building.



- **Two phases:** neighborhood formation and recommendation

k-NN UBCF: Neighborhood formation

- **Pearson's correlation coefficient** rates the similarity between target user u, and a possible neighbor v.

$$sim(\mathbf{u}, \mathbf{v}) = \frac{\sum_{i \in C} (r_{\mathbf{u},i} - \overline{r_{\mathbf{u}}})(r_{\mathbf{v},i} - \overline{r_{\mathbf{v}}})}{\sqrt{\sum_{i \in C} (r_{\mathbf{u},i} - \overline{r_{\mathbf{u}}})^2} \sqrt{\sum_{i \in C} (r_{\mathbf{v},i} - \overline{r_{\mathbf{v}}})^2}}$$

  - C is the set of items that are co-rated by users u and v.
  - $\overline{r_{\mathbf{u}}}$ is the average rating of the target user $\boldsymbol{u}$ and $r_{\mathbf{u},i}$ is the rating given to item i by $\boldsymbol{u}$.
  - Similarly, $\overline{r_{\mathbf{v}}}$ and $r_{\mathbf{v},i}$ are notations for a possible neighbor v.

- Top-$\boldsymbol{k}$ users that are most similar to the target user u are deemed as the neighborhood of u.

k-NN UBCF: Neighborhood formation

Anne	**5**	**4**	**1**	**4**	**?**
Bob	**3**	**1**	**2**	**3**	**3**
Carl	**4**	**3**	**4**	**3**	**5**
Dave	**3**	**3**	**1**	**5**	**4**
Eddie	**1**	**5**	**5**	**2**	**1**

	$\sqrt{\sum_{i\in C}(\boldsymbol{r}_{\boldsymbol{i}} - \bar{\boldsymbol{r}})^2}$	$\sum_{i\in C}\left(r_{u,i} - \bar{r_u}\right)\left(r_{v,i} - \bar{r_v}\right)$	$\bar{\boldsymbol{r}}$	$\boldsymbol{sim}$
Anne	3		3.5	
Bob	1.685	1.5	2.4	0.297
Carl	1.166	-1	3.8	-0.286
Dave	2.857	6	3.2	0.7
Eddie	3.682	-7.5	2.8	-0.679

k-NN UBCF: Recommendation

- Compute the rating prediction of item i made by the target user $\boldsymbol{u}$

$$p(\mathbf{u}, i) = \overline{r_{\mathbf{u}}} + \frac{\sum_{\mathbf{v} \in V} sim(\mathbf{u}, \mathbf{v}) \times \left(r_{\mathbf{v},i} - \overline{r_{\mathbf{v}}}\right)}{\sum_{\mathbf{v} \in V} |sim(\mathbf{u}, \mathbf{v})|}$$

  - V is the set of k similar users determined in the first phase.
    - It is common to ignore users with negative similarity to the target user.
  - $sim(\mathbf{u}, \mathbf{v})$ is the similarity value between two users, $\boldsymbol{u}$ and $\boldsymbol{v}$.
    - Note that, a similar user is removed from calculation if he did not rate the item i.

$\boldsymbol{p}(\mathbf{Anne},$ $) = 3.5 + \dfrac{0.297 \times (3 - 2.4) + 0.7 \times (4 - 3.2)}{0.297 + 0.7}$

$= 4.24$

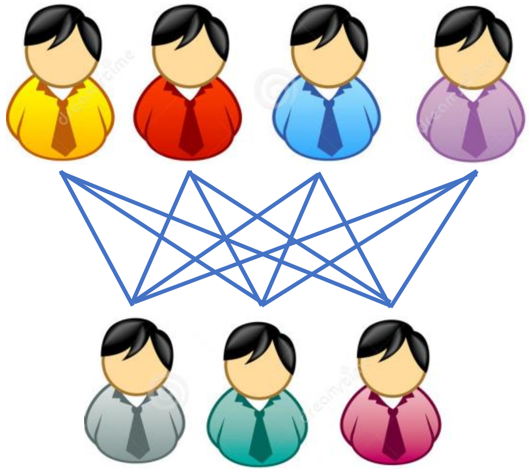
- Choose those highly rated items to recommend to the user

k-NN UBCF: Considerations

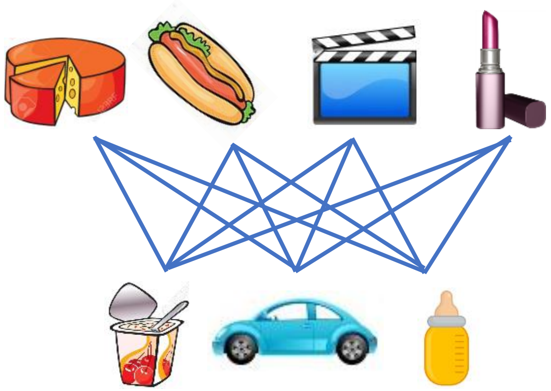
- Neighborhood selection
  - Which similarity metric? Pearson coefficients vs. Cosine similarity
  - Use similarity threshold or fixed number of neighbors
- Not all neighbor ratings might be equally "valuable"
  - Agreements on commonly liked items are less informative than those on controversial items → give more weight to items that have a higher variance
- The value of number of co-rated items
  - “Significance weighting“: linearly reducing the weight when the number of co-rated items is low
- Case amplification
  - Give more weight to "very similar" neighbors (similarity value is close to 1).

k-NN UBCF vs. k-NN IBCF

- k-NN UBCF is lack of scalability, in which predictions require real-time matching the target user to all user records.



vs.



- k-NN IBCF pre-computes all pairwise item to item similarity values based on their pattern of ratings across users.

k-NN IBCF: Neighborhood formation

- **Adjusted cosine similarity** estimates the similarity between target item i, and a possible neighbor j.

$$sim(i,j) = \frac{\sum_{\mathbf{u}\in C}\left(r_{\mathbf{u},i} - \bar{r}_{\mathbf{u}}\right)\left(r_{\mathbf{u},j} - \bar{\mathrm{r}}_{\mathbf{u}}\right)}{\sqrt{\sum_{\mathbf{u}\in C}\left(r_{\mathbf{u},i} - \bar{r}_{\mathbf{u}}\right)^2}\sqrt{\sum_{\mathbf{u}\in C}\left(r_{\mathbf{u},j} - \bar{\mathrm{r}}_{\mathbf{u}}\right)^2}}$$

  - C is the set of all users who have rated both items i and j
  - $\bar{r}_{\mathbf{u}}$ is the average rating of the user $\boldsymbol{u}$
  - $r_{\mathbf{u},i}$ and $r_{\mathbf{u},j}$ are the ratings given to items i and j by $\boldsymbol{u}$, respectively.

- Top-$\boldsymbol{k}$ items that are most similar to the target item i are deemed as the neighborhood of i.

k-NN IBCF: Neighborhood formation

	🍔	🍣	🥟	🍝	🍜
Anne	**5**	**4**	**1**	**4**	**?**
Bob	**3**	**1**	**2**	**3**	**3**
Carl	**4**	**3**	**4**	**3**	**5**
Dave	**3**	**3**	**1**	**5**	**4**
Eddie	**1**	**5**	**5**	**2**	**1**

	🍔	🍣	🥟	🍝	🍜
$\sqrt{\sum_{\mathbf{u}\in C}\left(r_{\mathbf{u},i}-\bar{r}_{\mathbf{u}}\right)^2}$	1.918	2.375	3.143	2.209	2.383
$\sum_{\mathbf{u}\in C}\left(r_{\mathbf{u},i}-\bar{r}_{\mathbf{u}}\right)\left(r_{\mathbf{u},j}-\bar{r}_{\mathbf{u}}\right)$	3.68	-5.92	-5.72	2.28	5.68
$\boldsymbol{sim}$	0.805	-0.908	-0.764	0.433	

k-NN IBCF: Recommendation

- Compute the rating prediction of target item i made by the user $\mathbf{u}$

$$p(\mathbf{u}, i) = \frac{\sum_{j \in J} r_{\mathbf{u},j} \times sim(i,j)}{\sum_{j \in J} |sim(i,j)|}$$

  - J is the set of k similar items and $r_{\mathbf{u},j}$ is the rating of user $\mathbf{u}$ on item j
  - $sim(i,j)$ is the value of the similarity metric used in the first phase

$$\boldsymbol{p}(\textbf{Anne},$$ $$) = \frac{5 \times 0.805 + 4 \times 0.433}{0.805 + 0.433} = 4.65$$

- It is also common to ignore items with negative similarity to the target item.

k-NN IBCF: Considerations

- Item-based filtering does not solve the scalability problem itself
- Pre-processing approach by Amazon.com (in 2003)
  - Calculate all pair-wise item similarities in advance
  - The neighborhood to be used at run-time is typically rather small, because only items which the user has rated are considered.
- Item similarities are supposed to be more stable than user similarities
- Memory requirements
  - N^2 pair-wise similarities to be memorized (N = number of items).

    In practice, this is significantly lower (items with no co-ratings)
  - Further reductions possible
    - Minimum threshold for co-ratings
    - Limit the neighborhood size (might affect recommendation accuracy)

CF using k-Nearest Neighbors

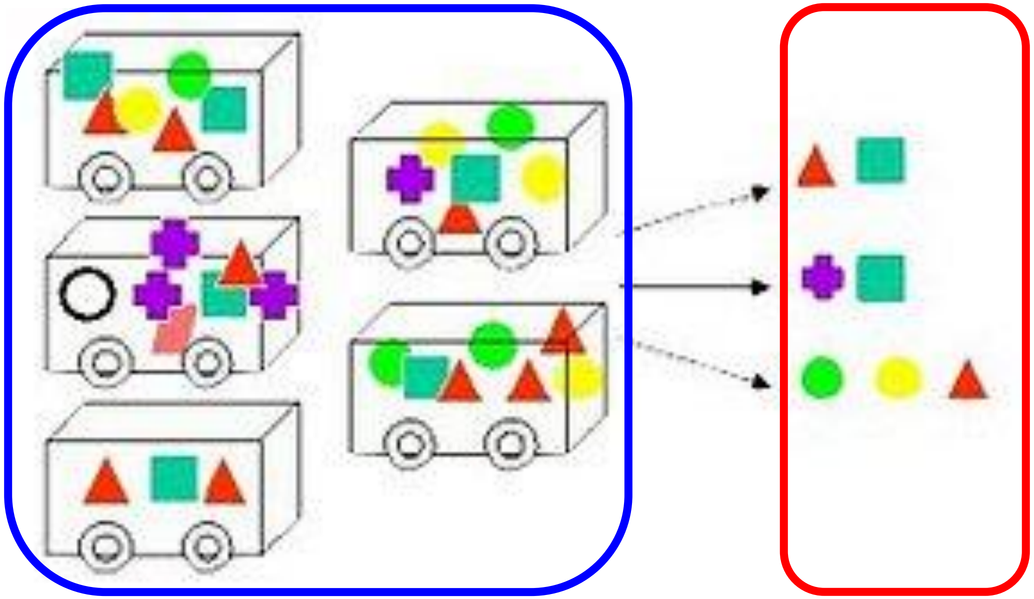
- Computing pairwise user (or item) similarities is intractable due to the high dimensionality of item or user.
- Dimensionality reduction techniques have been applied to downsize the scale of user and item profiles.
  - Project the user-item matrix into a lower dimensional space
    - Linear Discriminant Analysis (LDA), Principal Component Analysis (PCA), Autoencoder, etc.
  - Factorize the user-item matrix to obtain lower-rank representations of users and items
    - Singular Value Decomposition (SVD)

Approaches to Collaborative filtering

- k-Nearest Neighbors
- **Association rules**
- Matrix factorization
- Deep learning

CF using association rules

- Treat the items purchased by each user as a transaction
- Mine association rules from the transactions of all users to prediction the recommendations

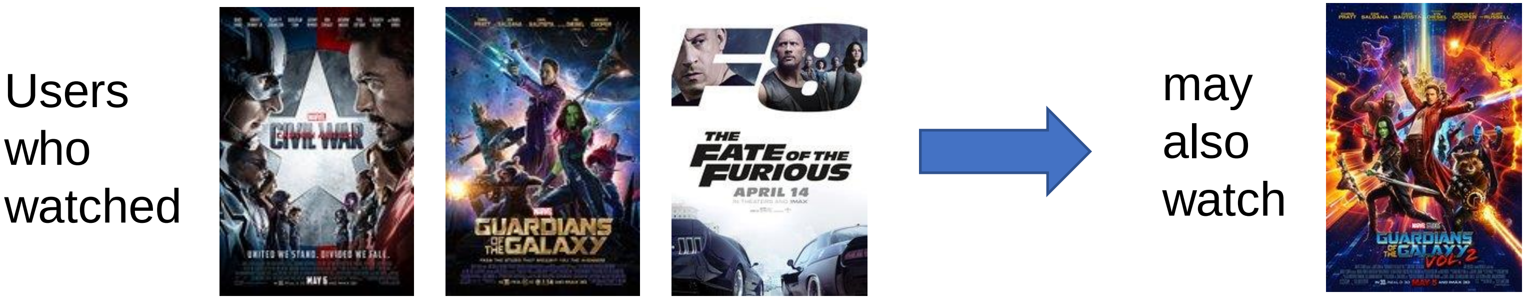


User profiles

Items for recommendation

CF using association rules

- The preferences of the target user are matched against the items on the left-hand side of each association rule $X \to Y$.
- Recommendations are top-N items on the right-hand side of the matching rules that have highest confidence values.



Users who watched

may also watch

Association rule CF: Considerations

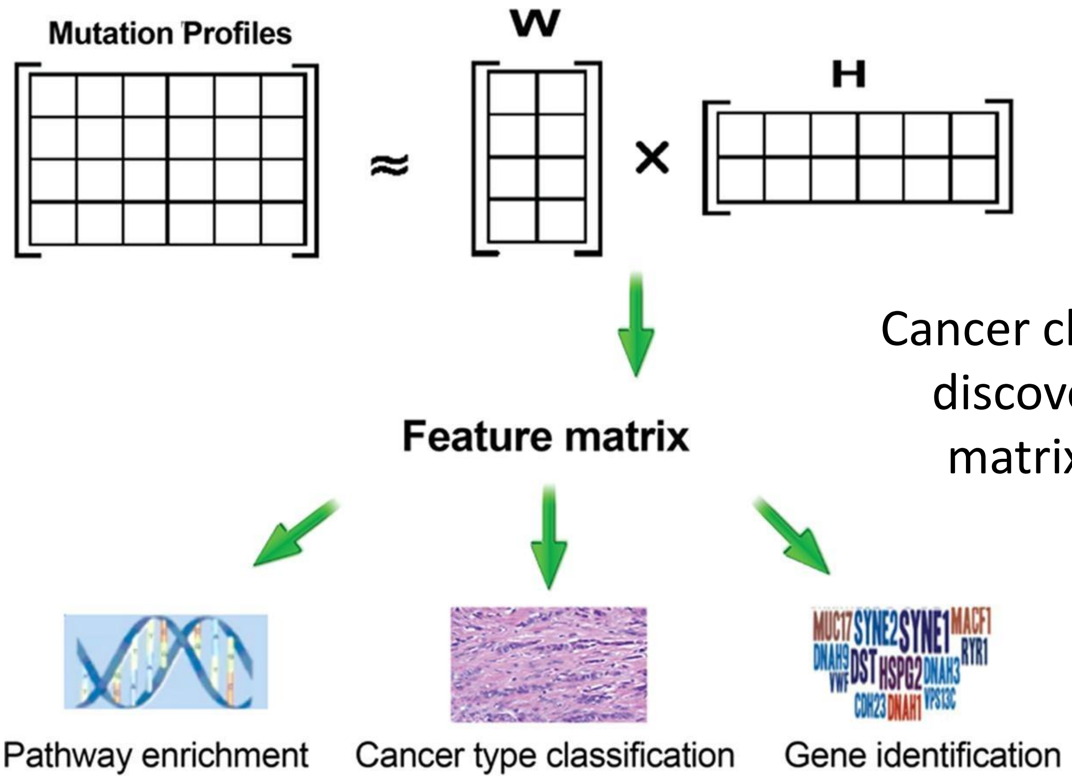
- It is difficult to find enough common items in multiple user profiles.
  - Any given user visits only a very small fraction of the available items.
  - **Solution:** dimensionality reduction, user associations vs. item associations
- Finding a matching rule antecedent with a full user profile is challenging.
  - **Solution:** use a sliding window w over the user profile and iteratively decrease its size until an exact match is found
- Patterns will not include infrequent but important items
  - E.g., for Web pages, references to deeper content or product-oriented pages occur far less frequently than those of top-level navigation-oriented pages
  - **Solution:** multiple minimum support

Approaches to Collaborative filtering

- k-Nearest Neighbors
- Association rules
- **Matrix factorization**
- Deep networks

Matrix factorization (MF)

- Decompose a matrix $\boldsymbol{M}$ into the product of several factor matrices, $\boldsymbol{F_1 F_2 \ldots F_n}$
  - n can be any number, but it is usually 2 or 3.

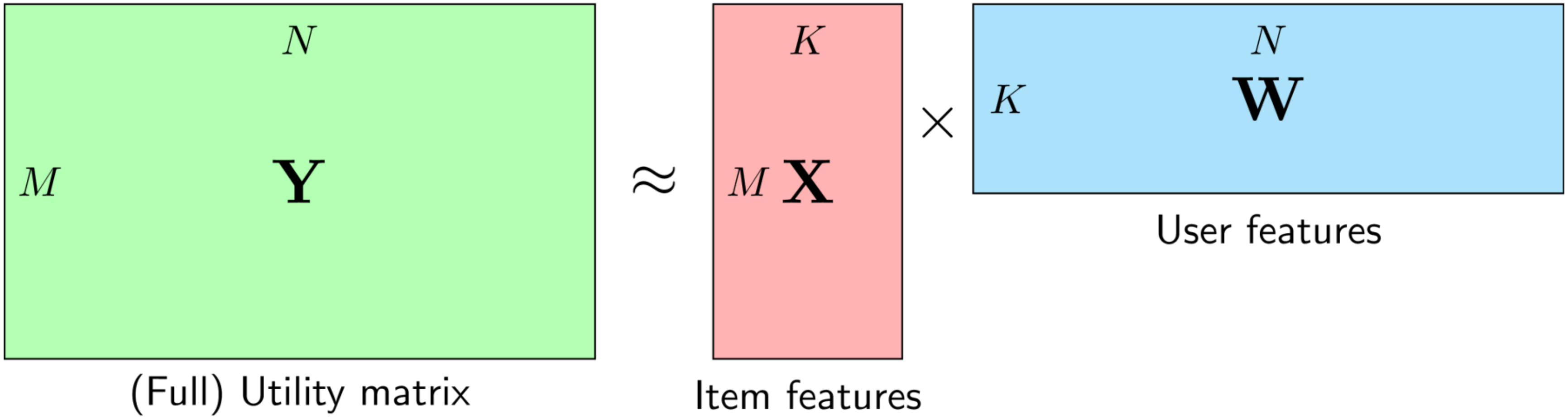


Cancer classification and pathway discovery using non-negative matrix factorization. Source

MF for recommender systems

- Decompose the utility matrix into two smaller matrices, capturing the latent factors of items and user tastes

$$\mathbf{Y} \approx \hat{\mathbf{Y}} = \mathbf{X}\mathbf{W}$$



- Matrix factorization is less popular than before, yet it is still essential for recommendation systems.

Latent factors in MF

- Latent factors account for the underlying reasons of a user purchasing / using a product.
  - E.g., overall quality, whether it is an action movie or a comedy, what stars are in it, etc.
- The connections between latent variables and observed variables are estimated during the training phase.
- Recommendations to a user is computed from his *possible interaction* with each product through the latent factors.

Singular Value Decomposition (SVD)

- Let A be any $m \times n$ matrix.
- There are orthogonal matrices U, V and a diagonal matrix Σ such that $\boldsymbol{A = U\Sigma V^T}$.
- The ordering of the vectors comes from the ordering of the singular values (largest to smallest).
- The columns of U are the eigenvectors of AA^T, while those of V are the eigenvectors of $A^T A$.
- The diagonal elements of Σ are the singular values, $\sigma_i = \sqrt{\lambda_i}$
- There are relationships between v_i and u_i (with normalization):

$$Av_i = \sigma_i u_i \qquad A^T u_i = \sigma_i v_i$$

- A numerical example can be found here.

SVD for recommender systems

- Let $\boldsymbol{R}$ be the given user–movie–rating matrix, in which a non-empty cell r_{ij} represents a known rating of user i on movie j.
  - E.g., in the Netflix Prize contest (2006), there were 17,000 movies and 500,000 users → the rating matrix has 8.5 billion entries.
- SVD decomposes R into two matrices, user-aspect $\boldsymbol{U}$ and movie-aspect $\boldsymbol{M}$.
  - $\boldsymbol{U} = [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_I]$ and $\boldsymbol{M} = [\mathbf{m}_1, \mathbf{m}_2, \dots, \mathbf{m}_J]$, where $\mathbf{u}_i$ and $\mathbf{m}_j$ are $K \times 1$ vectors and K is the number of aspects.
- That is, we want to approximate $\boldsymbol{R} \approx \boldsymbol{U}^{\boldsymbol{T}}\boldsymbol{M}$

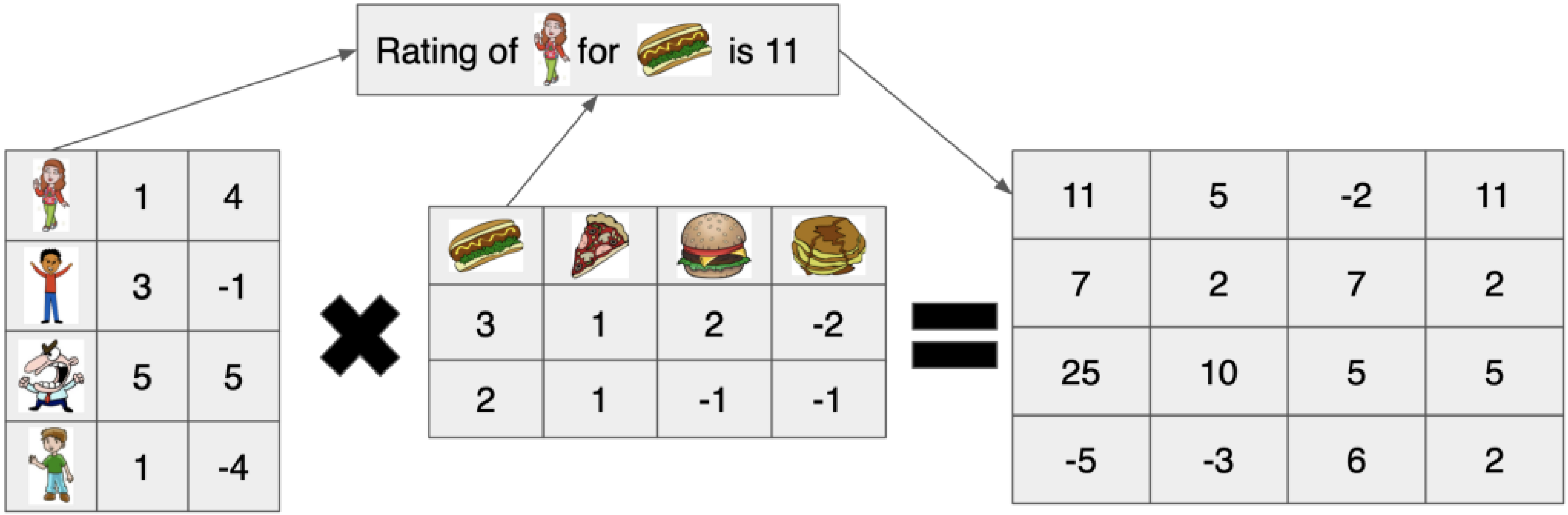
SVD for recommender systems

- Each movie is described by ***K*** aspects indicating how much that movie exemplifies each aspect.
  - The value of latent aspects K is usually empirically set.
- Each user is also represented by ***K*** aspect indicating how much he prefers each aspect.
- SVD determines the rating of how much a user ***i*** likes a movie ***j***.

$$r_{ij} \approx \mathbf{u}_i^T \mathbf{m}_j = \sum_{k=1}^{K} \mathbf{u}_{ki} \mathbf{m}_{kj}$$

  - where $\mathbf{u}_{ki}$ and $\mathbf{m}_{kj}$ are the $k\text{th}$ aspect value for user i and movie j, respectively.

SVD for recommender systems



Rating of for is 11

	1	4
	3	-1
	5	5
	1	-4

3	1	2	-2
2	1	-1	-1

11	5	-2	11
7	2	7	2
25	10	5	5
-5	-3	6	2

- Let p_{ij} be the predicted rating of user i on movie j.
- Use the resulting matrices $\boldsymbol{U}$ and $\boldsymbol{M}$ to predict the ratings in the test set

$$p_{ij} = \sum_{k=1}^{K} \mathbf{u}_{ki}\mathbf{m}_{kj}$$

SVD for recommender systems

- SVD finds these two smaller matrices which minimizes the resulting approximation error, the mean squared error (MSE)
- Traditional factorization methods for SVD does not work with sparse matrix.
  - E.g., in the Netflix Prize contest, the rating matrix has 100 million entries and 8.4 billions empty cells.

Incremental solution for SVD

- A simple incremental solution based on gradient descent by taking derivatives of the approximation error (Simon Funk)
- Train one single singular vector at a time to approximate $\boldsymbol{R}$ with the current $\boldsymbol{U}$ and $\boldsymbol{M}$
- Add one by one more singular vectors until K is reached
- Major advantage: ignore the errors on 8.4 billion empty cells

Incremental solution for SVD

- Let $e_{ij} = r_{ij} - p_{ij}$ be the error in the prediction of rating r_{ij}.
- Stochastic gradient descent: $\frac{\partial (e_{ij})^2}{\partial u_{ki}} = 2e_{ij}\frac{\partial e_{ij}}{\partial u_{ki}} = 2(r_{ij} - p_{ij})(-m_{kj})$
  - r_{ij} is a constant given in the training data → $\frac{\partial e_{ij}}{\partial u_{ki}} = -\frac{\partial p_{ij}}{\partial u_{ki}}$
  - p_{ij} is a sum over K terms, and one of them is a function of u_{ki}, i.e., $u_{ki} \times m_{kj}$
- The gradient descent update rule (from iteration t to $t+1$)

$$u_{ki}^{t+1} = u_{ki}^{t} - \gamma \frac{\partial (e_{ij})^2}{\partial u_{ki}} = u_{ki}^{t} + 2\gamma(r_{ij} - p_{ij})m_{kj}^{t}$$

  - where $\boldsymbol{\gamma}$ is the **learning rate**.
- Follow the same procedure to take the partial derivative and update m_{kj}.

Incremental solution for SVD

- After adding regularization to deal with overfitting

$$u_{ki}^{t+1} = u_{ki}^{t} + \gamma\left[2\left(r_{ij} - p_{ij}\right)m_{kj}^{t} - \lambda u_{ki}^{t}\right]$$
$$m_{kj}^{t+1} = m_{kj}^{t} + \gamma\left[2\left(r_{ij} - p_{ij}\right)u_{ki}^{t} - \lambda m_{kj}^{t}\right]$$

  - where λ is the **regularization constant**

- The whole training process thus has three main loops

```
For → the incremental addition of aspects
  For → the training of each aspect using SGD
    For → run through all given rating triplet (user, movie, rating)
          .........
    End For
  End For
End For
```

Technical issues for effective learning

- Vector initialization and stopping criterion greatly affect the generalization and accuracy result on unseen test data.
- Some initial values are required for $\mathbf{u}_k$ and $\mathbf{m}_k$
  - E.g., all assigned 0.1's or small random numbers with zero mean.
- Stopping criterions for the gradient descent are necessary.
  - E.g., SSE has little change or a fixed number of iterations
  - We often needs to stop before hitting the bottom of the gradient, which tends to overfit the training data.
- Training process: train the aspects incrementally vs. train all at once
- Some priors or biases can improve the final rating prediction
  - E.g., the overall rating mean, individual movie rating mean (some are good movies and get higher average ratings), user rating mean (some users tend to give higher or lower ratings), etc.

Tensor decomposition

- The interest of users is usually dynamic (change with time) $\rightarrow$ deal with the long-term/short-term interest of users
  - E.g., timeSVD++ (Netflix Prize winning team)
- Traditional matrix factorization methods are often extended to tensor decomposition to deal with the new dimension.

Q1. k-NN collaborative filtering

- This table shows the rating scores (from 1 to 5) of three IT figures, Mark Zuckerberg, Bill Gates and Guido van Rossum, on four programming languages, PHP, Apache Spark, Microsoft .NET and Python.

	php	Spark	Microsoft .NET	
	4.5	4.0	1.5	4.5
	3.0	1.0	4.0	2.0
	4.5	🟨	2.0	5.0

- Predict the rating score of Guido van Rossum on Apache Spark using k-Nearest Neighbors (with k = 1) user-based (or item-based) CF.

Q2. k-NN collaborative filtering

- The following table show the rating scores (from 1 to 5) of four characters on five book titles. An entry with question marks means no rating yet.

	Book1	Book2	Book3	Book4	Book5
Alice	1	2	5	?	1
George	5	?	1	2	5
Mary	?	3	4	3	4
Tom	1	1	5	4	?

- Predict the rating score of Tom on Book5 using k-Nearest Neighbors (with k = 2) user-based (or item-based) CF.

List of references

- Bing Liu. 2007. *Web Data Mining-Exploring Hyperlinks, Contents, and Usage Data.* Springer Series on Data-Centric Systems and Applications. **Chapter 12.4**.
- Math 77B: Collaborative Filtering, Chapter 02. (link)

THE END