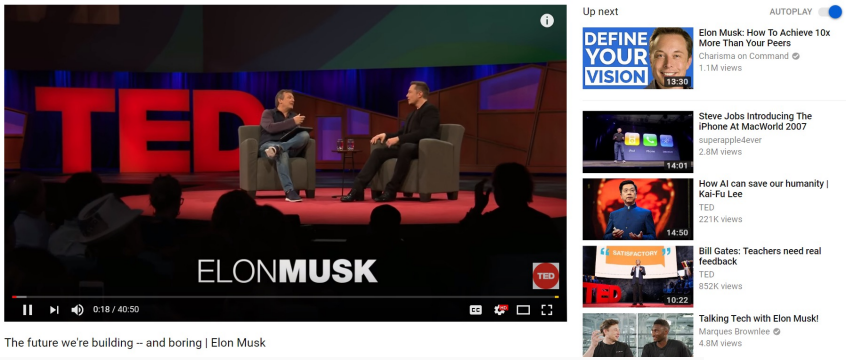
Traditional Recommender Systems

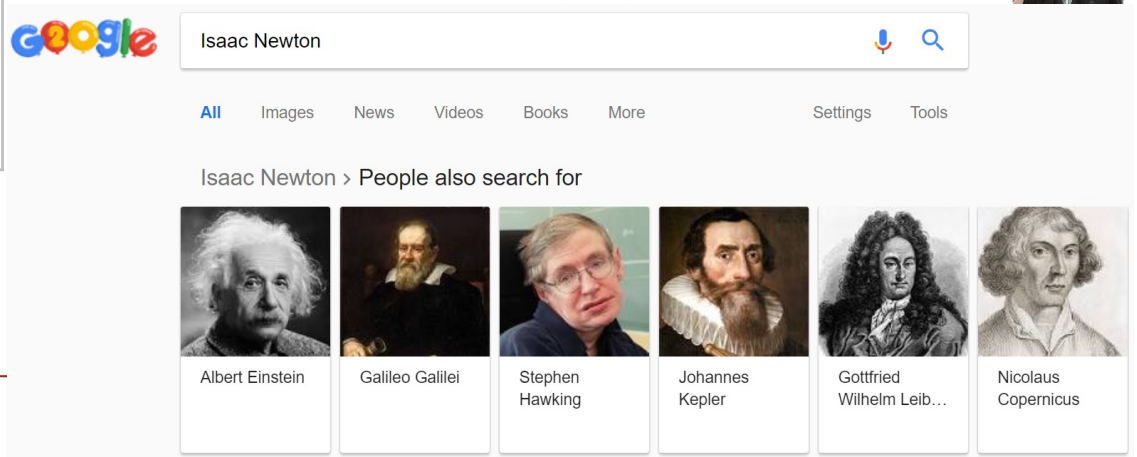
Michel.RIVEILL@univ-cotedazur.fr

Recommender Systems

- A recommender system (RS) helps users that have no sufficient competence or time to evaluate the, potentially overwhelming, number of alternatives offered by a web site.
  - In their simplest form, RSs recommend to their users personalized and ranked lists of items



The future we're building -- and boring | Elon Musk



2

The Impact of RecSys

- 35% of the purchases on Amazon are the result of their recommender system, according to McKinsey.
- During the Chinese global shopping festival of November 11, 2016, Alibaba achieved growth of up to 20% of their conversion rate using personalized landing pages, according to Alizila.
- Recommendations are responsible for 70% of the time people spend watching videos on YouTube.
- 75% of what people are watching on Netflix comes from recommendations, according to McKinsey

The Age of Recommendation

Search:

User ➡ Items

Recommend:

Items ➡ User

Picture from: http://insidebigdata.com/wp-content/uploads/2014/06/Humor_recommender.jpg

Amazon: A personalized online store

Picture from: amazon.com

Recommender Problem

A good recommender

- Show programming titles to a software engineer and baby toys to a new mother.
- Don't recommend items user already knows or would find anyway.
- Expand user's taste without offending or annoying him/her…

Challenges

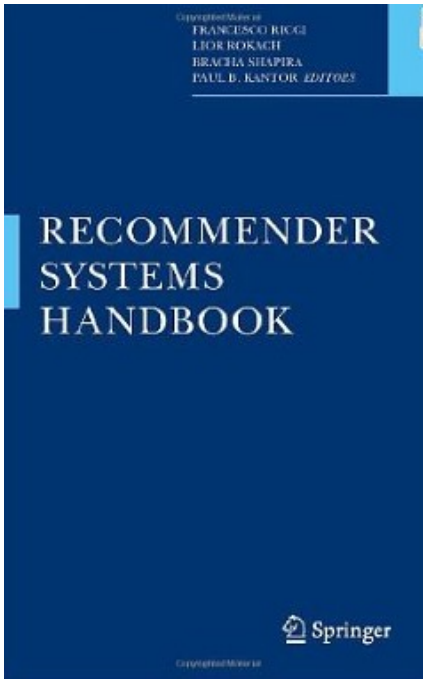
- Huge amounts of data, tens of millions of customers and millions of distinct catalog items.
- Results are required to be returned in real time.
- New customers have limited information.
- Old customers can have a glut of information.
- Customer data is volatile.

About this lecture

- Will give you a **very short introduction** to the field of Recommender Systems

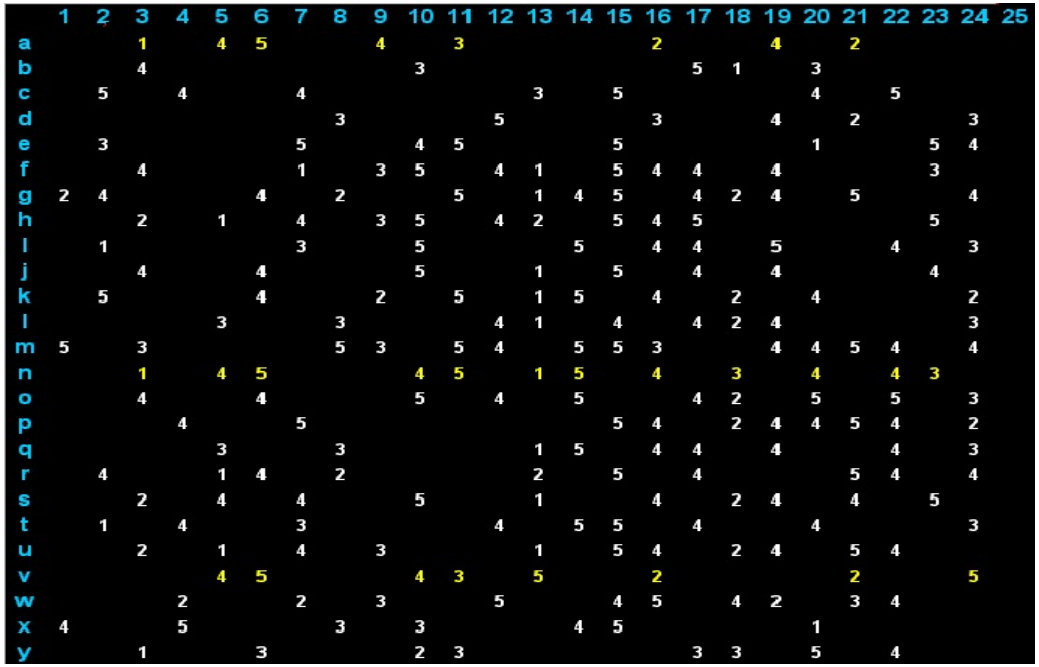
- **"FIT" → How do recommender systems (RS) work ?**
  - How can you compute recommendations?
    - From the basis
      - Non personalized recommendation
    - To more accurate proposal
      - Content-based filtering
      - Collaborative filtering
    - It's just an introduction…

- **"EVALUATE" → How do we measure their success?**
  - How can we know that the recommendations are good?

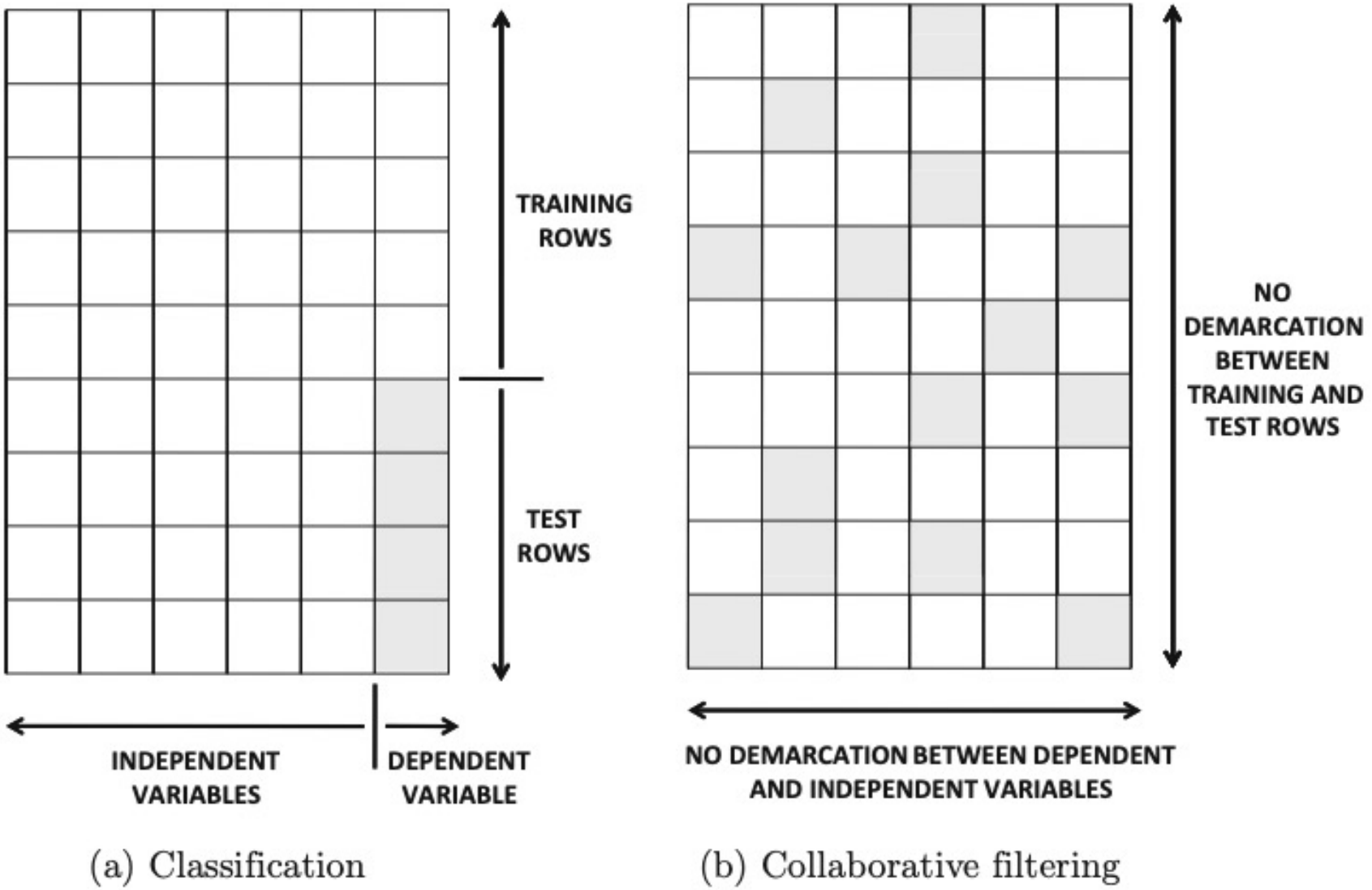


Problem characterization

- Given
  - The profile of the "active" user
  - The description of a set of items
  - And possibly some situational context
- Compute
  - A relevance (ranking) score for each recommendable item
- The problem …
  - … is to **learn a function** that predicts the relevance score for a given (typically unseen) item
  - … it is a Matrix completion problem



ML dataset vs RS dataset



ML dataset vs RS dataset

- Collaborative filtering models are closely related to missing value analysis.

- In traditional ML approach
  - Few missing value
  - Try to impute them

- In RS approach
  - Missing value >> actual rating
  - Data matrix is very large and sparse.

The problem

1. Collect the data

User	Item	Score	Time
0	0	5	1
2	1	4	2
4	2	5	3
0	2	2	4
2	0	5	5
Etc.			

The problem

1. Collect the data
2. Construct the pivot matrix

User	Item	Score	Time
0	0	5	1
2	1	4	2
4	2	5	3
0	2	2	4
2	0	5	5
Etc.			

Item / User	0	1	2	3	4
0	5				
1					
2					
3					
4					

The problem

1. Collect the data
2. Eventually construct the pivot matrix

User	Item	Score	Time
0	0	5	1
2	1	4	2
4	2	5	3
0	2	2	4
2	0	5	5
Etc.			

Item / User	0	1	2	3	4
0	5				
1					
2		4			
3					
4					

The problem

1. Collect the data
2. Eventually construct the pivot matrix

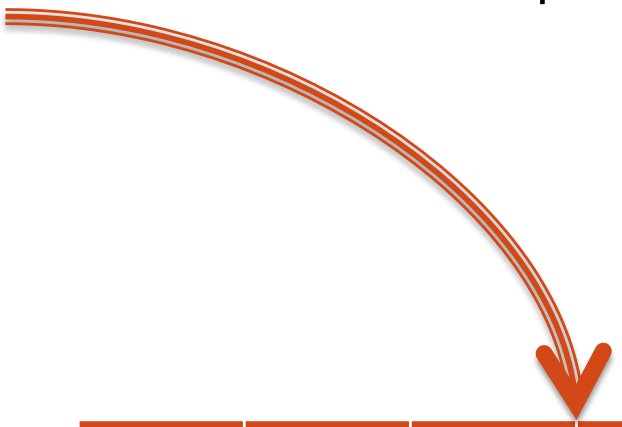
User	Item	Score	Time
0	0	5	1
2	1	4	2
4	2	5	3
0	2	2	4
2	0	5	5
Etc.			

Item / User	0	1	2	3	4
0	5		2		
1					
2	5	4			
3					
4			5		

The problem

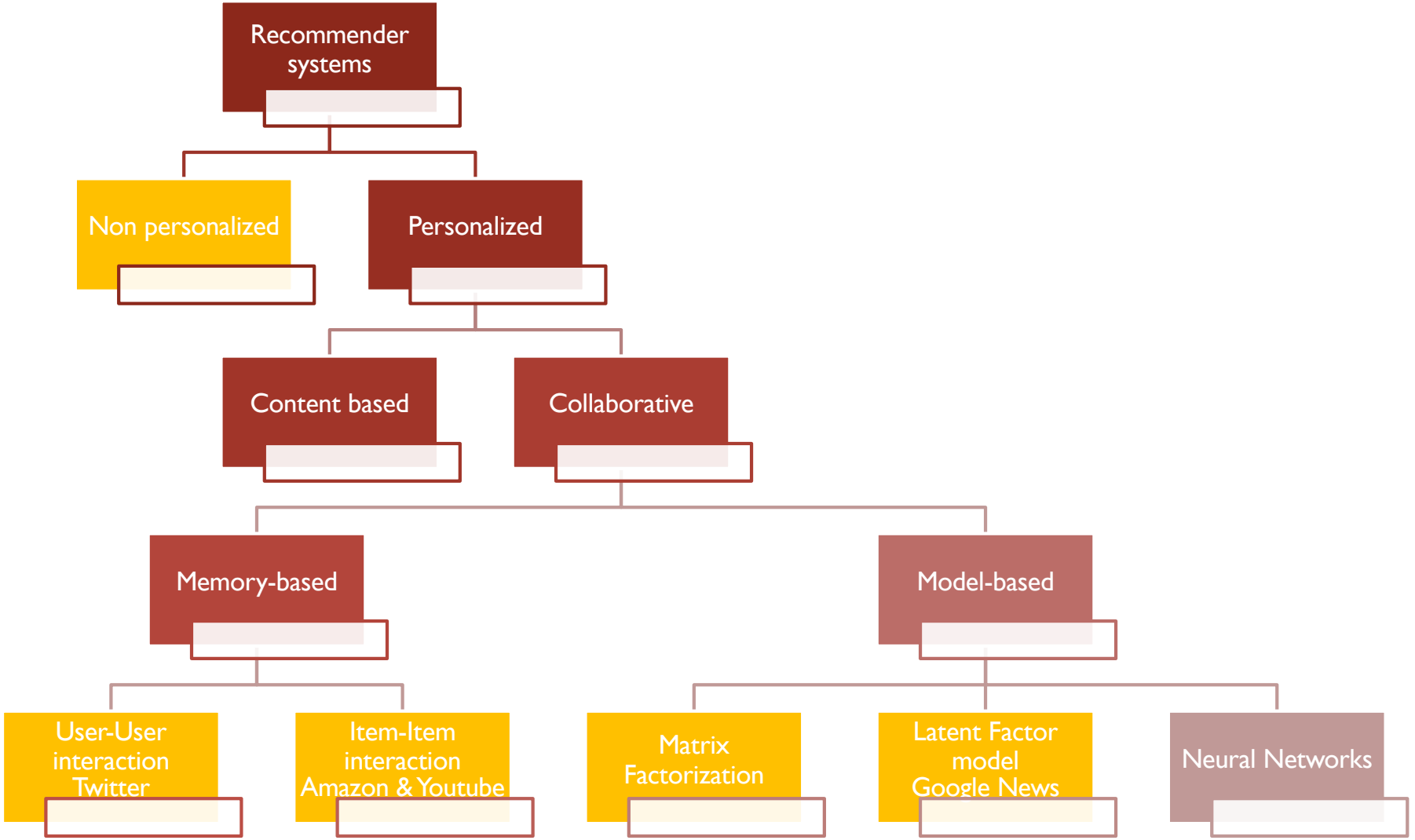
1. Collect the data
2. Eventually construct the pivot matrix
3. Predict missing scores for the desired used
   - Depend of the strategy

User	Item	Score	Time
0	0	5	1
2	1	4	2
4	2	5	3
0	2	2	4
2	0	5	5
Etc.			



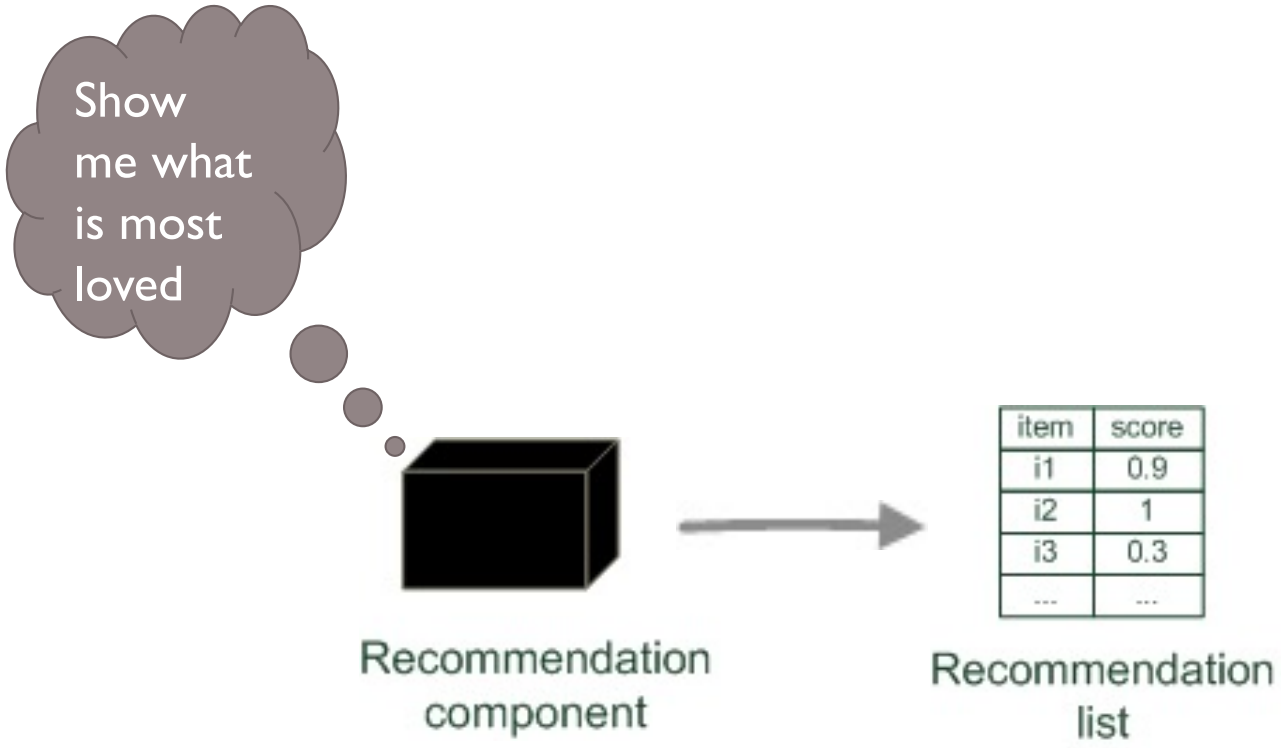
Item / User	0	1	2	3	4
0	5		2		
1					
2	5	4			
3					
4			5		

Paradigms of recommender systems



Non personalized recommendation

Non personalized recommendation



Non personalized recommendation Approach 1 = mean rating

- Goal: provides a list
  - For all user the list is the same
  - But, what do you put at the top of the list?
- A simple approach: rank by score
  - Score : $\frac{\sum_u r_{ui}}{n}$

Item / User	0	1	2	3	4
0	5	3	2		
1		5	3	3	
2	5	4	2	1	
3		3		1	2
4			5	2	
Proposed rankink		3.75			

(3+5+4+3)/4

Non personalized recommendation Approach 1 = mean rating

- Goal: provides a list
  - For all user the list is the same
  - But, what do you put at the top of the list?
- A simple approach: rank by score
  - Score : $\frac{\sum_u r_{ui}}{n}$

Item / User	0	1	2	3	4
0	5	3	2	1,75	2
1	5	5	3	3	2
2	5	4	2	1	2
3	5	3	3	1	2
4	5	3.75	5	2	2
Proposed rankink	5	3.75	3	1,75	2

Ranking list for user 4: [0, 1, 4]

Non personalized recommendation Approach 2 = normalized mean rating

- Goal: provides a list
  - For all user the list is the same
  - But, what do you put at the top of the list?
- A simple approach: rank by score
  - Another proposal for calculating score: normalize each rating by the mean of the rating of each user: $r_u + \frac{\sum_u (r_{ui} - r_u)}{n}$

Item / User	0	1	2	3	4	Mean rating r_u
0	5	3	2			
1		5	3	3		3.66
2	5	4	2	1		
3		3		1	2	
4			5	2		
$\frac{\sum_u (r_{ui} - r_u)}{n}$						

STEP 1: Calculate the mean rating for each user (5+3+3)/3

Non personalized recommendation Approach 2 = normalized mean rating

- Goal: provides a list
  - For all user the list is the same
  - But, what do you put at the top of the list?
- A simple approach: rank by score
  - Another proposal for calculating score: normalize each rating by the mean of the rating of each user: $r_u + \frac{\sum_u (r_{ui} - r_u)}{n}$

Item / User	0	1	2	3	4	Mean rating r_u
0	5	3	2			3.33
1		5	3	3		3.66
2	5	4	2	1		3
3		3		1	2	2
4			5	2		3.5
$\frac{\sum_u (r_{ui} - r_u)}{n}$						

Non personalized recommendation Approach 2 = normalized mean rating

- Goal: provides a list
  - For all user the list is the same
  - But, what do you put at the top of the list?
- A simple approach: rank by score
  - Another proposal for calculating score: normalize each rating by the mean of the rating of each user: $r_u + \frac{\sum_u (r_{ui} - r_u)}{n}$

Item / User	0	1	2	3	4	Mean rating r_u
0	5	3	2			3.33
1		5	3	3		3.66
2	5	4	2	1		3
3		3		1	2	2
4			5	2		3.5
$\frac{\sum_u (r_{ui} - r_u)}{n}$	1.835					

STEP 2: compute the normalized rating for each movie
((5-3.33)+(5-3)) / 2 =

Non personalized recommendation Approach 2 = normalized mean rating

- Goal: provides a list
  - For all user the list is the same
  - But, what do you put at the top of the list?
- A simple approach: rank by score
  - Another proposal for calculating score: normalize each rating by the mean of the rating of each user: $r_u + \frac{\sum_u (r_{ui} - r_u)}{n}$

Item / User	0	1	2	3	4	Mean rating r_u
0	5	3	2			3.33
1		5	3	3		3.66
2	5	4	2	1		3
3		3		1	2	2
4			5	2		3.5
$\frac{\sum_u (r_{ui} - r_u)}{n}$	1.835	0.75	-0.37	-1.29	0	

Non personalized recommendation Approach 2 = normalized mean rating

- Goal: provides a list
  - For all user the list is the same
  - But, what do you put at the top of the list?
- A simple approach: rank by score
  - Another proposal for calculating score: normalize each rating by the mean of the rating of each user: $r_u + \frac{\sum_u (r_{ui} - r_u)}{n}$

Item / User	0	1	2	3	4	Mean rating r_u
0	5	3	2			3.33
1	5.5	5	3	3		3.66
2	5	4	2	1		3
3	3.8	3		1	2	2
4			5	2		3.5
$\frac{\sum_u (r_{ui} - r_u)}{n}$	1.835	0.75	-0.37	-1.29	0	

STEP 3: add the mean user rating to the item rating
- 3.66+1.835 for item 0/user1
- 2+1.835 for item 3/user1

Non personalized recommendation Approach 2 = normalized mean rating

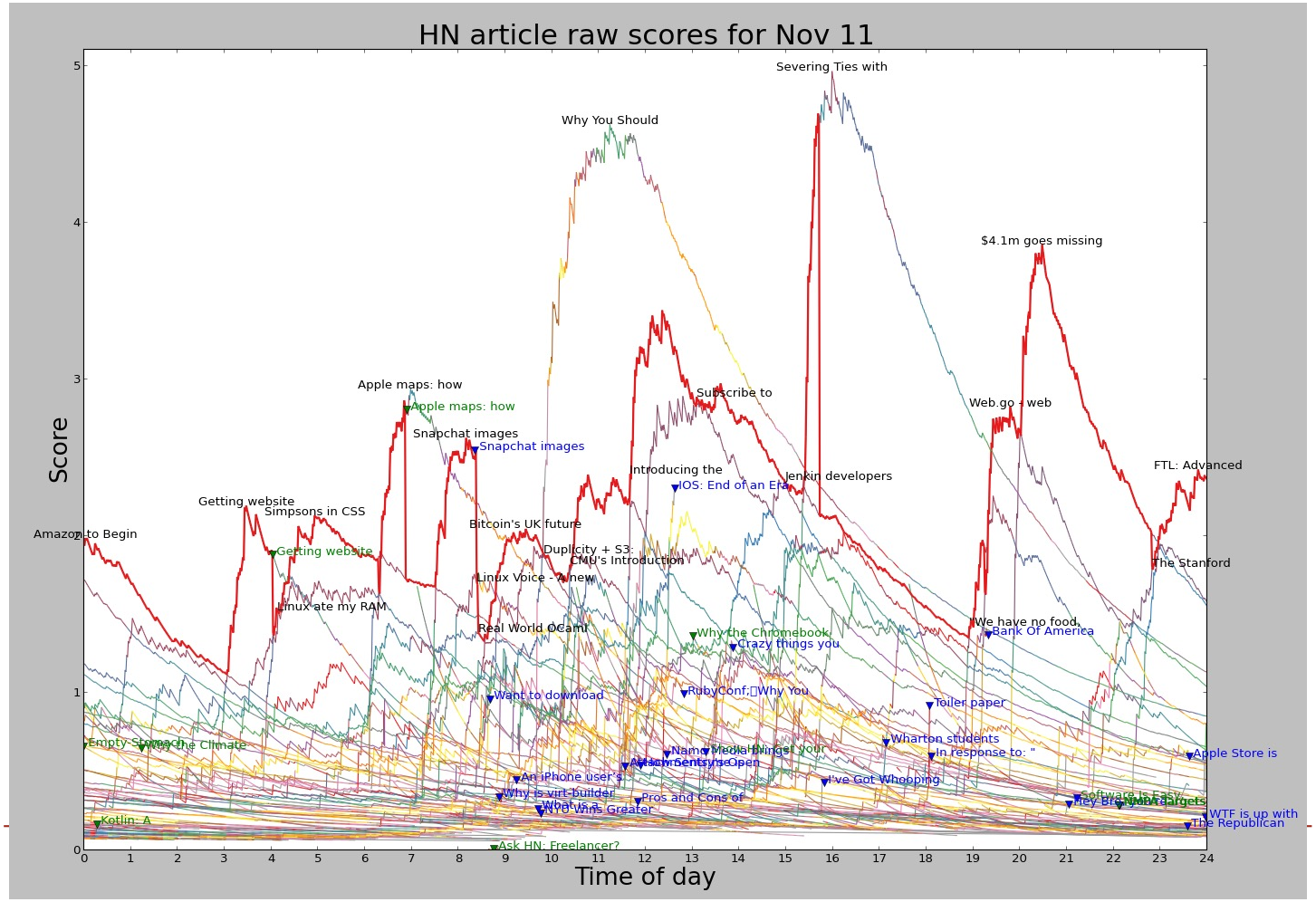
- Goal: provides a list
  - For all user the list is the same
  - But, what do you put at the top of the list?
- A simple approach: rank by score
  - Another proposal for calculating score: normalize each rating by the mean of the rating of each user: $r_u + \frac{\sum_u (r_{ui} - r_u)}{n}$

Item / User	0	1	2	3	4	Mean rating r_u
0	5	3	2	3.33-1.29	3.33-0	3.33
1	5.5	5	3	3	3.66	3.66
2	5	4	2	1	3	3
3	3.8	3	2-0.37	1	2	2
4	5.335	4.25	5	2	3.5	3.5
$\frac{\sum_u (r_{ui} - r_u)}{n}$	1.835	0.75	-0.37	-1.29	0	

Ranking list for user 4: [0, 1, 4]

If the time is important....

- Hacker News algorithms
  - http://www.righto.com/2013/11/how-hacker-news-ranking-really-works.html
  - Score : $\frac{(votes-1)^{0.8}}{(age+2)^{1.8}}$



Personalized recommendation Collaborative filtering

Memory based

Memory-based collaborative filtering

- Use the matrix of user ratings for items of the entire database to find users/or items that are similar to the active user/or items and use their rating to predict new rating

- Advantage
  - Quality of precision are rather good
  - Relatively simple algorithm to implement for any situation
  - New data can be added easily and incrementally

- Disadvantage
  - It depend on human ratings
  - Performance decreases when data gets sparse
  - Prevent scalability and have problem with large dataset

User-Based Algorithms (Breese et al, UAI98)

- $r_{i,j}$= rating of user *i* on item *j*
- I_i = items for which user *i* has rated
- Mean rate for user u is
  - $\bar{r}_u = \frac{\sum_{j \in I_u} r_{uj}}{|I_u|}$

- Predicted vote for "active user" *u* for item j is weighted sum
  - Without normalization: $\widehat{r_{uj}} = \frac{\kappa \sum_{k \in U} w(u,k) \mathrm{r_{kj}}}{\sum_{k \in U} w(u,k)}$
  - With normalization: $\widehat{r_{uj}} = \bar{r}_u + \frac{\kappa \sum_{k \in U} w(u,k)(\mathrm{r_{kj}} - \bar{r}_k)}{\sum_{k \in U} |w(u,k)|}$

weights of users
- Near 1 : users u and k is similar
- Near 0 : users u and k have no similarity

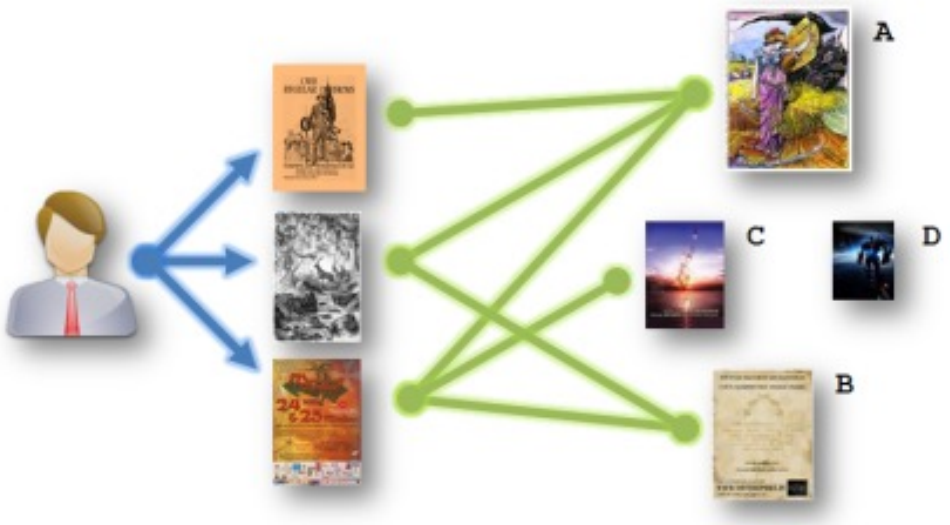
User-Based Algorithms (Breese et al, UAI98)

- K-nearest neighbor
  - $w(a,b) = \begin{cases} 1, if\ b \in neighbors(a) \\ 0, else \end{cases}$
- Pearson correlation coefficient $\in [-1,1]$:
  - $w(a,b) = \frac{\sum_i (a_i - \bar{a})(c_i - \bar{b})}{\sqrt{\sum_i (a_i - \bar{a})^2}\sqrt{\sum_i (b_i - \bar{b})^2}}$
- Cosine similarity $\in [-1,1]$:
  - $w(a,b) = \frac{\sum_i a_i b_i}{\sqrt{\sum_i a_i^2}\sqrt{\sum_i b_i^2}}$

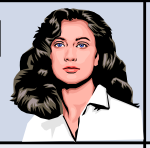
Item-based Approach

- Similar with user-based approach but is on the item side

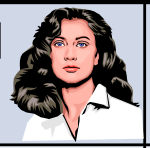
- Choosed by Amazon / Youtube



An example : Item-based approach
Example: infer – r13

	Item 1	Item 2	Item 3	Item 4	Item 5
User 1 	8	1	?	2	7
User 2 	2	?	5	7	5
User 3 	5	4	7	4	7
User 4 	7	1	7	3	8
User 5 	1	7	4	6	?
User 6 	8	3	8	3	7

How to calculate similarity between items ?

	Item 1	Item 2	Item 3	Item 4	Item 5
User 1 	8	1	?	2	7
User 2 	2	?	5	7	5
User 3 	5	4	7	4	7
User 4 	7	1	7	3	8
User 5 	1	7	4	6	?
User 6 	8	3	8	3	7

Sim(I3, I5)

	Item 3	Item 5
	?	7
→	5	~~5~~
→	7	7
→	7	8
	4	?
→	8	7
$\bar{r}$	6,75	6,75

- Calculate Pearson correlation
- $per(I_3, I_5) = \frac{\sum(ri_3 - \bar{r}_3)(ri_5 - \bar{r}_5)}{\sqrt{\sum(r_{i3} - \bar{r}_3)^2}\sqrt{\sum(r_{i5} - \bar{r}_5)^2}}$
- $per(I_3, I_5) = \frac{0.9375}{1,1875} = 0.79$
- Calculate cosine similarity
- $\cos(I_3, I_5) = \frac{I_3 . I_5}{\|I_3\| \|I_5\|}$
- $\cos(I_3, I_5) = \frac{5*5+7*7+7*8+8*7}{\sqrt{5^2+7^2+7^2+8^2}\sqrt{5^2+7^2+8^2+7^2}}$
- $\cos(I_3, I_5) = 0.99$

How to calculate r_{13} ?

- r_{13} = ranking for user1 and item 3 without normalization

- $$r_{13} = \frac{r_{11}*sim(I_1,I_3)+r_{12}*sim(I_2,I_3)+r_{14}*sim(I_4,I_3)+r_{15}*sim(I_5,I_3)}{|sim(I_1,I_3)+sim(I_2,I_3)+sim(I_4,I_3)+sim(I_5,I_3)|}$$

	Item 1	Item 2	Item 3	Item 4	Item 5
User 1 	8	1	?	2	7
User 2 	2	?	5	7	5

- $$r_{13} = \frac{8*sim(I_1,I_3)+1*sim(I_2,I_3)+2*sim(I_4,I_3)+7*sim(I_5,I_3)}{|sim(I_1,I_3)+sim(I_2,I_3)+sim(I_4,I_3)+sim(I_5,I_3)|}$$

Generalization (without normalization)

- User based: $r_{ui} = \frac{\sum_v r_{vi} * sim(u,v)}{\sum_v |sim(u,v)|}$, v ∈ *similar User*
  - Calculate sim, only with common item
  - Rating matrix is almost stable → pre-calculate sim matrix
  - Trick I:
    - normalize the rating
  - Trick II:
    - If two users have very few items in common,
      - For example : 1 rating and it is the same
        --> hight similarity but low confidence
    - Weigh similarity by $\frac{\min(N, |P_{uv}|}{|P_{uv}|}$
      - $|P_{uv}|$ is the number of common items shared by user u and v
- Item based: $r_{ui} = \frac{\sum_j r_{uj} * sim(i,j)}{\sum_j |sim(i,j)|}$, j ∈ *similar Item*
  - Adapt the previous recommendation

Offline computation : Online Recommendation

Offline Computation:

- **builds a similar-items table** which is extremely time intensive, $O(N^2M)$
  - M is the number of customers ± 10 millions
  - N is the number of items ± 1 millions
- In practice, it's closer to O(NM), as most customers have very few purchases
- Sampling customers can also reduce runtime even further with little reduction in quality.

Online Recommendation:

- Given a similar-items table, the algorithm
  - **finds items** similar to each of the user's purchases and ratings,
  - **aggregates** those items, and then
  - **recommends the most popular or correlated items**.

- For User based, exchange Item and User
  - Item similarity is more stable than User similarity

  - N >> M → Item based is more efficient → choosed by Amazon / Youtube

Scalability and Quality: Comparison

User based collaborative filtering:

- little or no offline computation
- impractical on large data sets, unless it uses dimensionality reduction, sampling, or partitioning
- dimensionality reduction, sampling, or partitioning reduces recommendation quality

Item based collaborative filtering:

- scalability and performance are achieved by creating the expensive similar-items table offline
- online component "looking up similar items“ scales independently of the catalog size or the number of customers

  → fast for extremely large data sets
- recommendation quality is excellent since it recommends highly correlated similar items
- unlike traditional collaborative filtering,
  - the algorithm performs well with limited user data,
  - producing high-quality recommendations based on as few as two or three items

User-based or Item-based Approach

- PRO
  - Few modeling assumptions
  - Few tuning parameters to learn
  - Easy to explain to users
    - Dear Amazon.com Customer, We've noticed that customers who have purchased or rated How Does the Show Go On: An Introduction to the Theater by Thomas Schumacher have also purchased Princess Protection Program #1: A Royal Makeover (Disney Early Readers).
- CONS
  - Computationally expensive, $O(M^2N)$ for User based in the worst case, where
    - M is the number of customers ± 10 millions
    - N is the number of items ± 1 millions
  - Dimensionality reduction can increase the performance,
    - BUT, also reduce the quality of the recommendation
  - For very large data sets
    - the algorithm encounters severe performance and scaling issues

Work on scalability: Cluster Models

- Approach
  - Divide the customer base into many segments and treat the task as a classification problem
  - Assign the user to the segment containing the most similar customers
  - Uses the purchases and ratings of the customers in the segment to generate recommendations
  - Cluster models have better online scalability and performance than collaborative filtering because they compare the user to a controlled number of segments rather than the entire customer base.
- Problems
  - Quality of the recommendation is low
  - The recommendations are less relevant because the similar customers that the cluster models find are not the most similar customers
  - To improve quality, it needs online segmentation, which is almost as expensive as finding similar customers using collaborative filtering

Comparative results:

- The MovieLens dataset contains
  - 1 million ratings
  - 6,040 users
  - 3,900 movies.
- The best overall results are reached by the item-by-item based approach. It needs 170 seconds to construct the model and 3 seconds to predict 100,021 ratings.

	User Based	Model Based	Item Based
model construction time (sec.)	730	254	170
prediction time (sec.)	31	1	3
MAE	0.6688	0.6736	**0.6382**

Table from: Candillier, L., Meyer, F., & Boull'e Marc. (2007). Comparing state-of-the-art collaborative filtering systems [3]

Personalized recommendation Collaborative filtering

Model based

Model-based collaborative filtering

- Find patterns based on training data, and these are used to make predictions for real data
- Extract some information from dataset, and use that as a "model" to make recommendations without having to use complete dataset every time
- Advantage
  - Handle sparsity better than memory based ones
  - Scalable with large dataset
  - Improve prediction speed
- Disadvantage
  - Expensive model building
  - Can lose useful information due to reduction models

Latent based recommendation

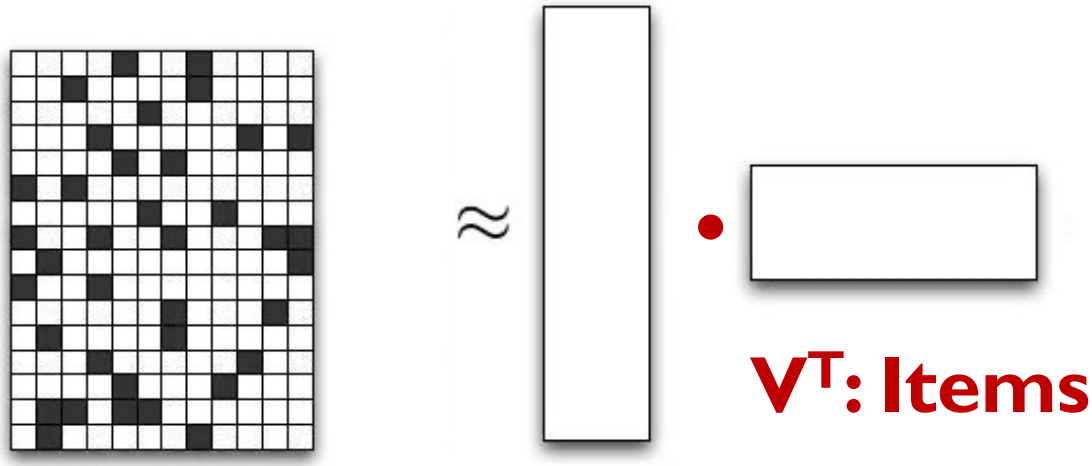
- Models with latent classes of items and users
  - Individual items and users are assigned to either a single class or a mixture of classes

- Could be implemented by
  - Latent Factor Models
    - Items and users described by unobserved factors
    - Matrix factorization or SVD decomposition
  - Probabilistic latent semantic analysis (PLSA)
  - Neural networks

Matrix factorization

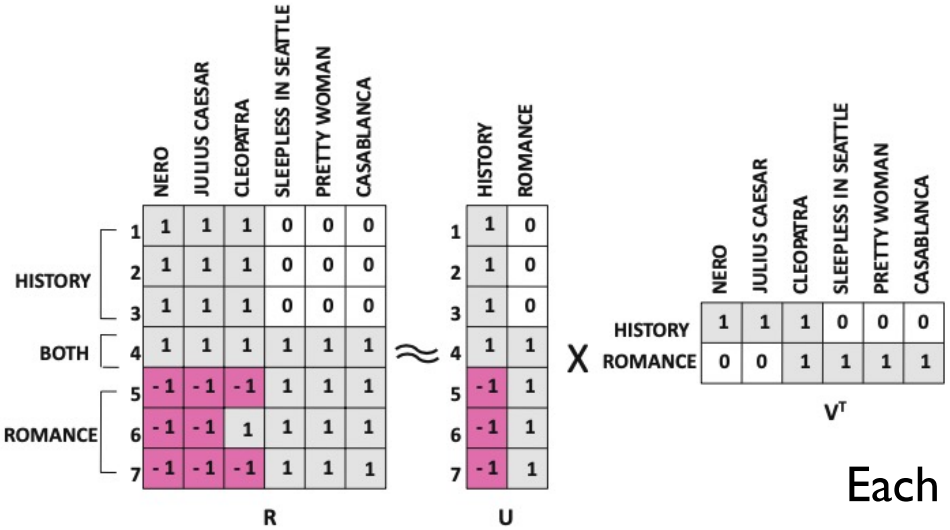
- Mathematic concept of matrix factorization
  - Set of Users (U)
  - Set of Items (V)
  - R = rating matrix

  - The goal is to discover K latent features, with the input of the 2 matrix U,V
    - U(|U|*k)
    - V(|V|*k)
    - $R \approx U.V^T = \hat{R}$
    - $\widehat{r_{ui}} = u_u^T v_i = \sum_k u_{uk} v_{ki}$

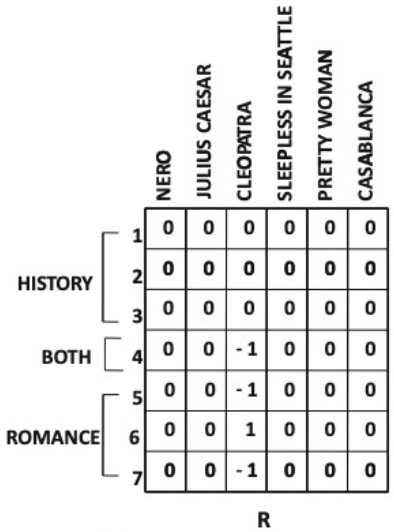


- Challenge: How to compute a mapping of users and items to factor vectors?

Matrix factorization



(a) Example of rank-2 matrix factorization



(b) Residual matrix

Each row of U or V is latent factor

The ith row of U is user factor

→ contain k entries corresponding to the affinity of user i towards the k concepts in the ratings matrix

→ Here k=2

Each rating r_{ui} in R can be approximately Expressed by $r_{ui} \approx \overline{u_u}.\ \overline{v_i}$

Matrix factorization
Leaning algorithms

- Cost function error : MSE
  - $Cost = \sum(error)^2 = \sum(e_{ui})^2 = \sum$ (actual rating – predicted rating)2

$$= \sum_{u,i}(r_{ui} - \widehat{r_{uj}})^2 = \sum_{u,i}(r_{ui} - u_u^T v_i)^2$$

- Goal: minimize the cost function i.e. minimize the MSE
  - It's an optimization problem
- Generic solution : gradient descent approach
  - Gradient: $\frac{\partial}{\partial u_{uk}} e_{uj}^2 = -2(r_{uj} - \widehat{r_{uj}})\, v_{jk} = -2e_{uj}v_{jk}$

$$\frac{\partial}{\partial v_{ik}} e_{ui}^2 = -2(r_{ui} - \widehat{r_{ui}})\, u_{uk} = -2e_{uj}u_{uk}$$

  - Modification of parameters in the opposite direction of the gradient, with α (learning rate)
    - $u'_{uk} \leftarrow u_{uk} + \alpha\, e_{uj}v_{jk}$
    - $v'_{jk} \leftarrow v_{jk} + \alpha\, e_{uj}v_{ik}$
    - $\forall\, k \in \{1..K\}$

Matrix decomposition through SVD decomposition

- Singular value decomposition (SVD)
  - is a form of matrix factorization
  - For a m*n matrix A, we want to compute 3 matrix U, S, V such as $A = USV^T$
    - U is an **m*m** square matrix
    - S is an **m*n** diagonal matrix, $S_{i+1,i+1} <= S_{i,i}$
    - V is an **n*n** matrix
  - in which the columns of U and V are constrained to be mutually orthogonal
    - Mutual orthogonality = the concepts can be completely independent
    - U and V respectively contain the eigenvecyors of AA^T and A^TA
    - S contains the square roots of the eigenvalues
- For fully specified matrix, it easy to perform SVD decomposition
  - A lot of methods
  - A lot of tools

Truncated SVD decomposition

- In order to approximate the rating matrix,
  - Use truncated SVD decomposition at rank $k \ll \min\{m, n\}$
  - $R \approx Q_k \Sigma_k P_k^T$
- The factorization here is into three matrices rather than two
  - However, the diagonal matrix Σ, can be absorbed in either the user factors Qk or the item factors Pk
  - By convention
    - $U \approx Q_k \Sigma_k$
    - $V \approx P_k^T$

SVD decomposition uses

- Each row u_i of U is a vector representation of user i
- Each row v_j of V is a vector representation of item j
- Sd,d assigns a weight to each dimension d of these vectors
- Score $A_{i,j}$ is the weigthed do product of u_i and v_j, where the weight of each dimension is value $S_{d,d}$
  - $A_{i,j} = \sum_d U_{i,d} * Sd_{,d} * Vd_{,j}$

- Example:

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6	Movie 7	Movie 8	Movie 9
User 1	0	1	1	1	0	0	1	0	0
User 2	1	0	1	0	0	1	0	0	0
User 3	0	0	0	0	1	0	0	0	1
User 4	0	1	0	1	0	0	0	1	0
User 5	1	0	0	0	1	0	0	0	0
User 6	0	0	1	0	0	0	0	0	1
User 7	1	0	0	0	0	0	0	0	0

SVD decomposition uses

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6	Movie 7	Movie 8	Movie 9
User 1	0	1	1	1	0	0	1	0	0
User 2	1	0	1	0	0	1	0	0	0
User 3	0	0	0	0	1	0	0	0	1
User 4	0	1	0	1	0	0	0	1	0
User 5	1	0	0	0	1	0	0	0	0
User 6	0	0	1	0	0	0	0	0	1
User 7	1	0	0	0	0	0	0	0	0

USV^T, With M = 4

$$\begin{bmatrix} 0.00 & 1.00 & 1.01 & 1.00 & -0.17 & 0.17 & 0.63 & 0.37 & 0.17 \\ 1.12 & 0.00 & 1.02 & 0.00 & -0.02 & 0.69 & 0.17 & -0.17 & -0.03 \\ 0.06 & -0.02 & 0.08 & -0.02 & 0.97 & -0.18 & -0.05 & 0.03 & 0.99 \\ 0.00 & 1.00 & -0.01 & 1.00 & 0.18 & -0.17 & 0.37 & 0.63 & -0.17 \\ 1.02 & 0.02 & -0.09 & 0.02 & 0.86 & 0.16 & -0.12 & 0.14 & 0.16 \\ -0.11 & 0.01 & 0.94 & 0.01 & 0.18 & 0.16 & 0.22 & -0.21 & 0.88 \\ 0.78 & -0.02 & 0.08 & -0.02 & 0.24 & 0.26 & -0.05 & 0.03 & -0.18 \end{bmatrix}$$

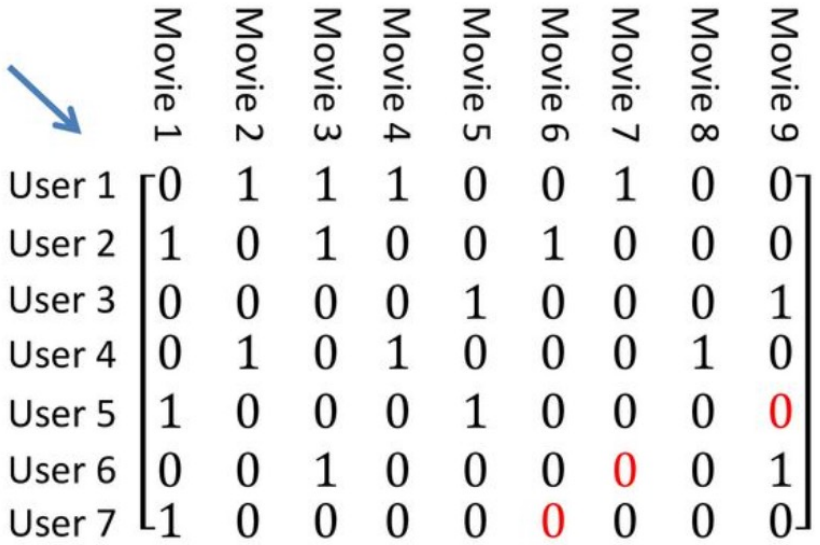
Colors:
Red: best value for users 5,6,7
Green : movies already seen
Black : unseen movies

Recommendation
- Movies 6 and 9 for user 5
- Movies 7 for user 6
- Movie 6 for user 7

SVD decomposition uses

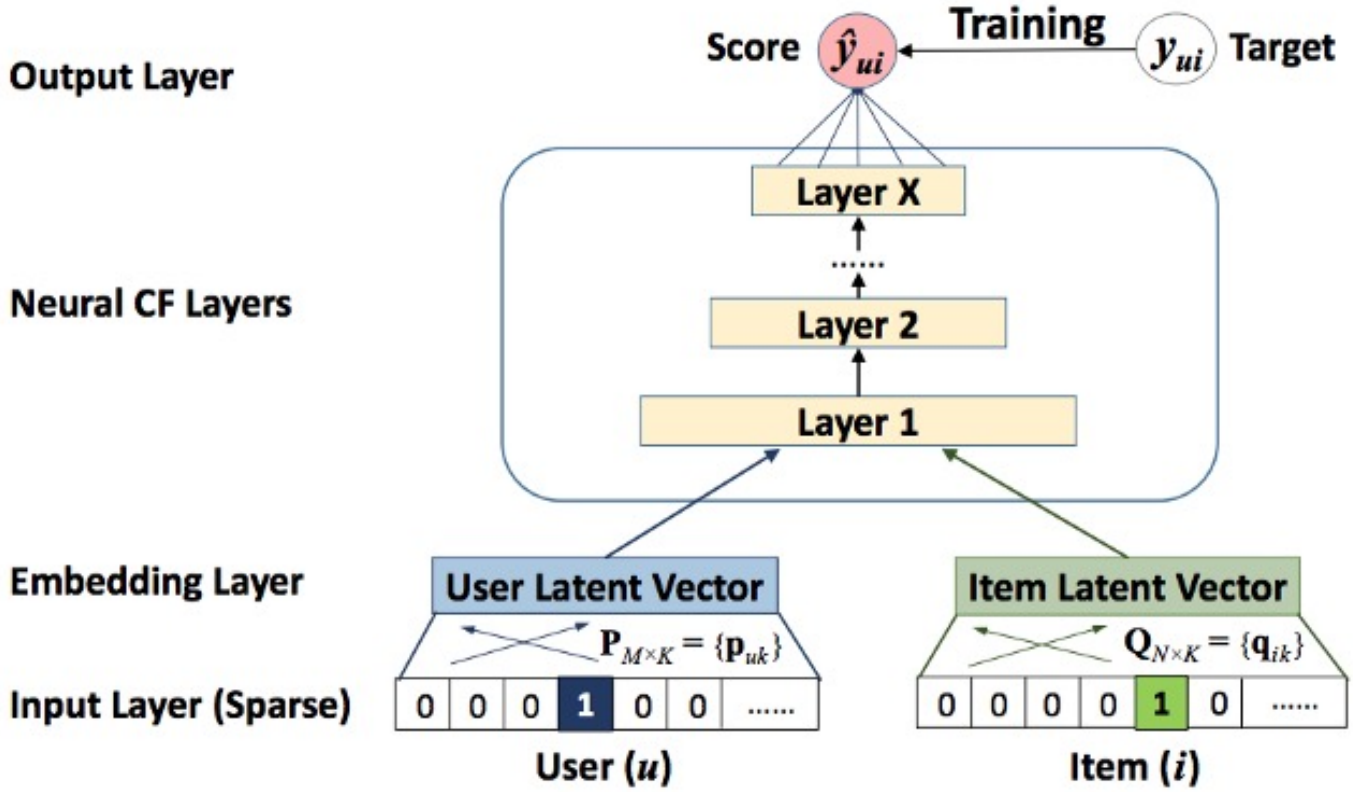
- Each row u_i of U is a vector representation of user i
- Each row v_j of V is a vector representation of item j
- Sd,d assigns a weight to each dimension d of these vectors
- Score $A_{i,j}$ is the weigthed do product of u_i and v_j, where the weight of each dimension is value $S_{d,d}$
  - $A_{i,j} = \sum_d U_{i,d} * Sd_{,d} * Vd_{,j}$

- Example:



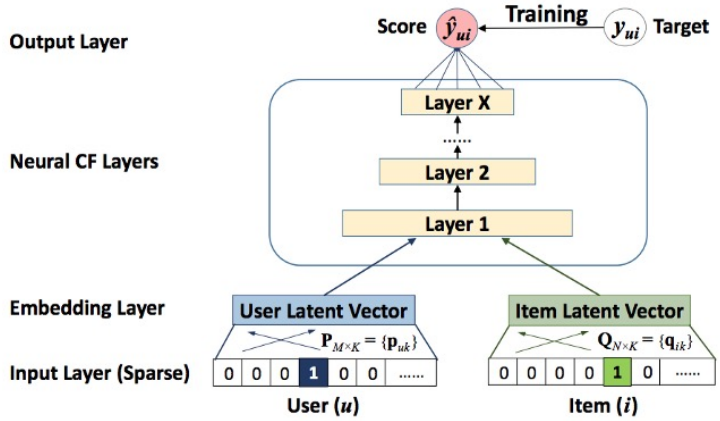
How to use neural network ?

- Neural collaborative filtering (NCF)
  - Learn interaction with a deep neural network



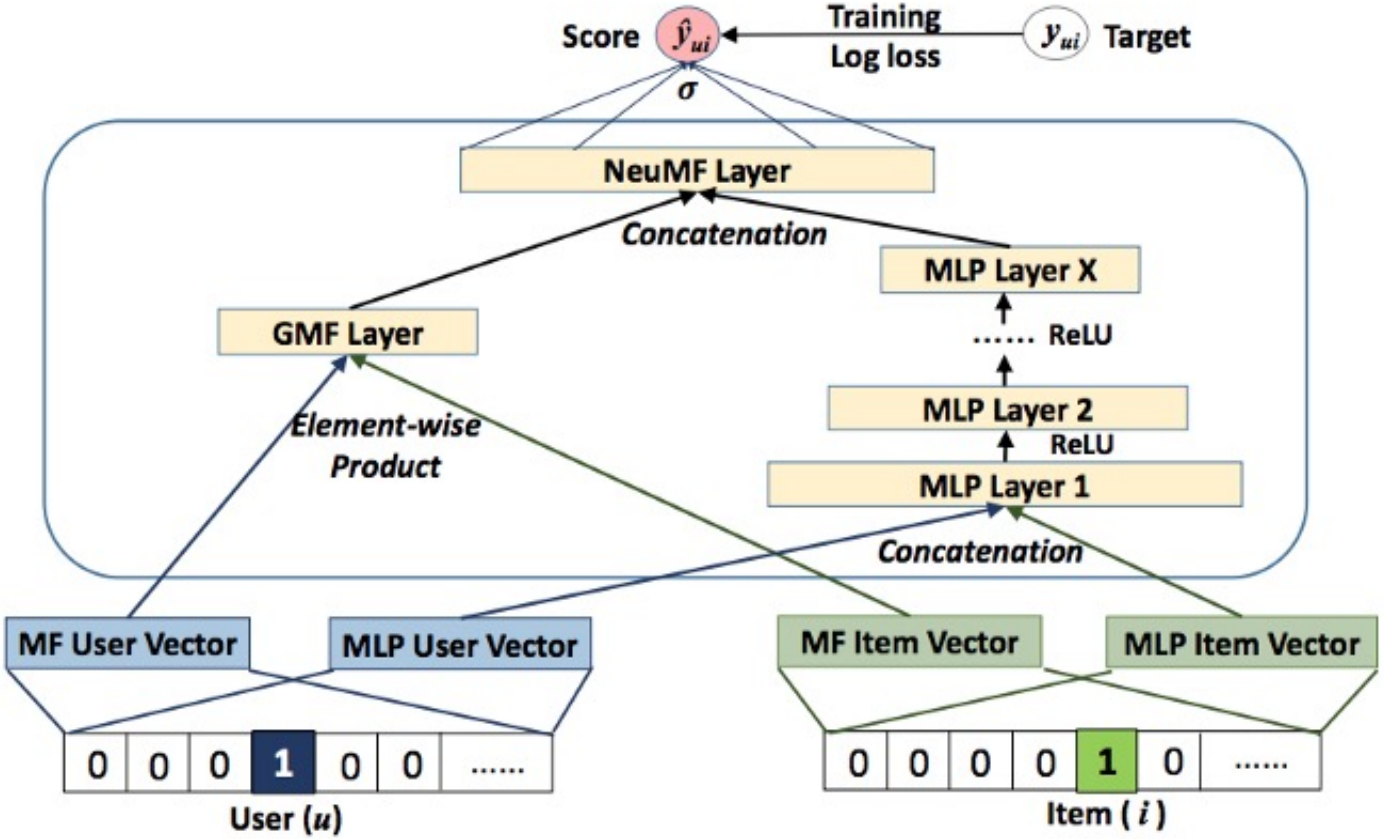
Neural collaborative filtering (NCF)

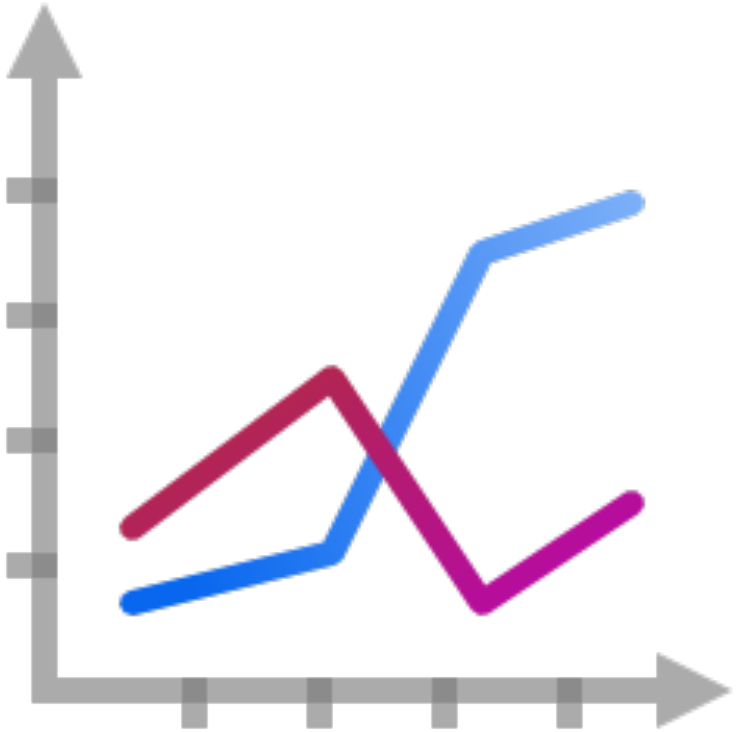
- Input Layer binarise a sparse vector for a user and item identification where:
  - Item (i): 1 means the user u has interacted with Item(i)
  - User (u): user id
- Embedding layer is a fully connected layer that projects the sparse representation to a dense vector.
  - The obtained user/item embeddings are the latent user/item vectors.
- Neural CF layers use multi-layered neural architecture to map the latent vectors to prediction scores.
- The final output layer returns the predicted score by minimizing the pointwise loss/pairwise loss.



Neural collaborative filtering (NCF)

- According to the construction of layer 1, we can build different models
  - concatenation -> Multi-layer Perceptron (MLP)
  - dot-product -> Generalized Matrix Factorisation (GMF)
- It is also possible to combine the models





Evaluating recommender systems

How to evaluate RS ?

- Test with real users
  - A/B tests
  - Example measures: sales increase, click through rates
- Laboratory studies
  - Controlled experiments
  - Example measures: satisfaction with the system (questionnaires)

- Offline experiments
  - Based on historical data
  - Example measures: prediction accuracy, coverage

Traditional accuracy metrics

- Prediction accuracy metrics include the mean absolute error (MAE), root mean square error (RMSE).
  - Use in regression task
  - Focus on comparing the actual vs predicted ratings
  - Easy to calculate and easy to use
  - They are not relevant for recommander systems **because we are more interest in assessing the relevance of the best recommendations** than the totality of the recommendations

Record

- MAE: Mean absolute error
  - sklearn.metrics.mean_absolute_error
- MSE: Mean square error
  - sklearn.metrics.mean_squared_error
- RMSE: Root MSE
  - √MSE

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

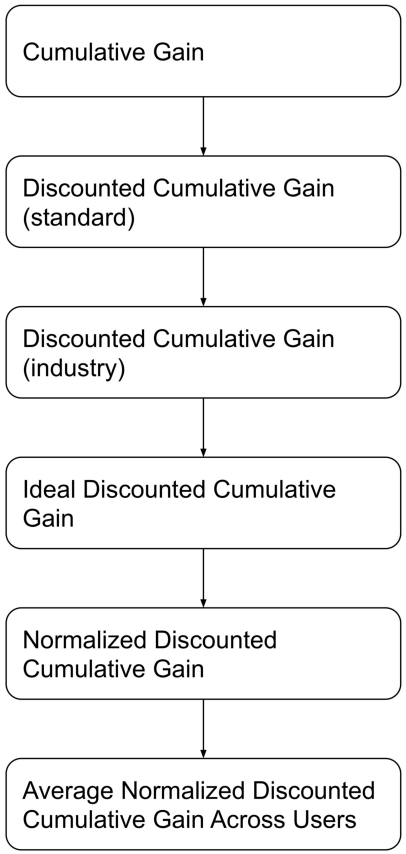
Ranking metrics

- In order to have more suitable metrics, in recommender systems we use mainly
  - *MRR: Mean Reciprocal Rank*
  - *MAP: Mean Average Precision*
  - *NDCG: Normalized Discounted Cumulative Gain --> implemented in sklearn*

- *Others specific metrics available in sklearn*
  - See : multilabel ranking metrics on sklearn documentation.

NDCG: Normalized Discounted Cumulative Gain

- Goal of NDCG
  - Place more emphasis on relevant recommendations at the beginning of lists



$$\mathrm{CG_p} = \sum_{i=1}^{p} rel_i$$

p elements in the recommended ranking

graded relevance of the result at position i (gain)

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)}$$

logarithmic reduction factor to penalize proportionally to the position of the result

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

graded relevance to get a stronger emphasis on retrieving relevant documents

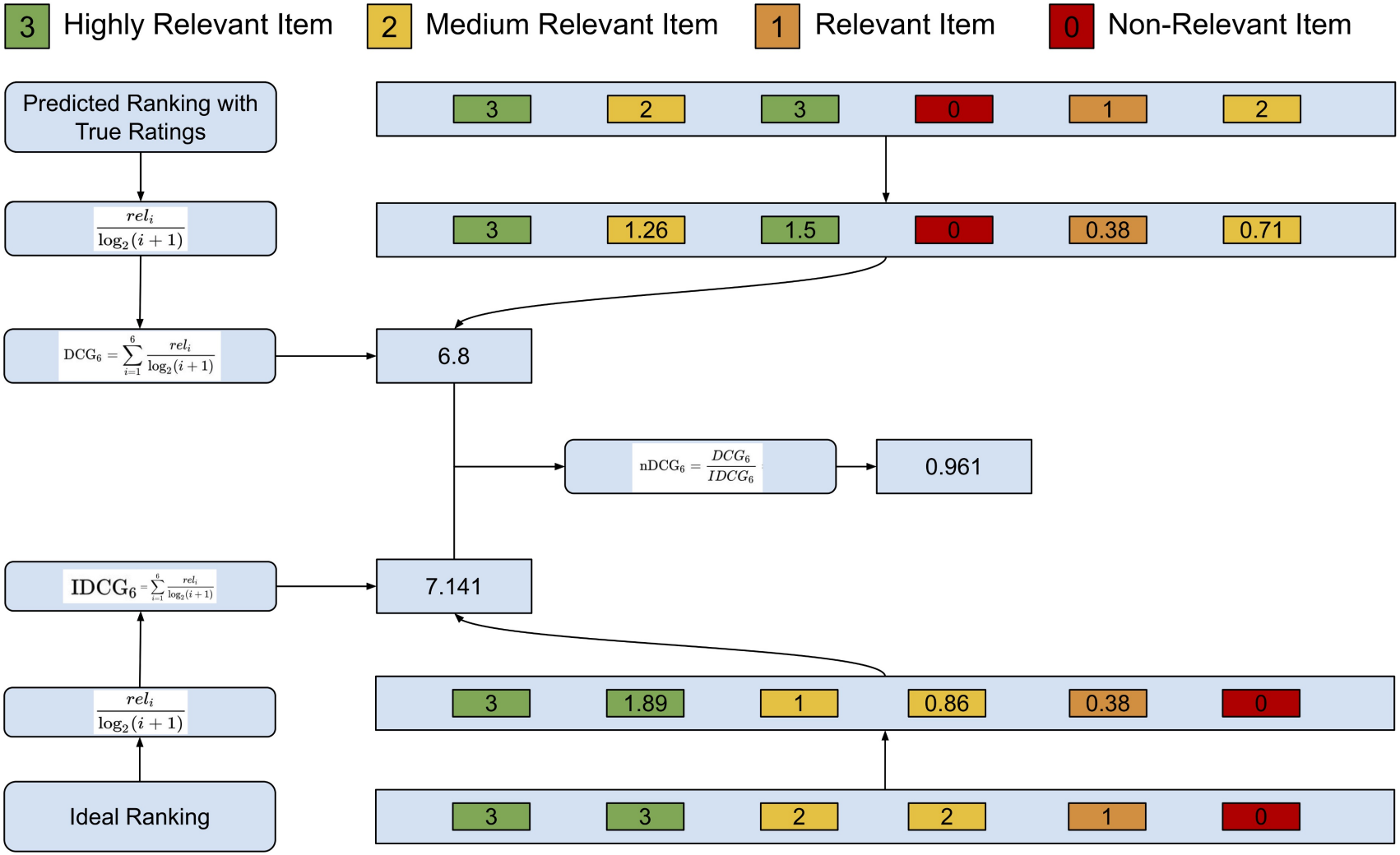
$$\mathrm{IDCG_p} = \sum_{i=1}^{|REL_p|} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

list of relevant documents (ordered by their relevance) in the corpus up to position p

$$\mathrm{nDCG_p} = \frac{DCG_p}{IDCG_p}$$

77

NDCG in practical



NDCG Pros and Cons

- NDCG Pros
  - The primary advantage of the NDCG is that it takes into account the graded relevance values. When they are available in the dataset, the NDCG is a good fit.

- NDCG Cons
  - The NDCG has some issues with partial feedback.

  - This happens when we have incomplete ratings.
  - This is case for the majority of recommender systems situations.
  - If we had complete ratings there would be no real task to achieve!

  - Generally the missing values have a default value of 0