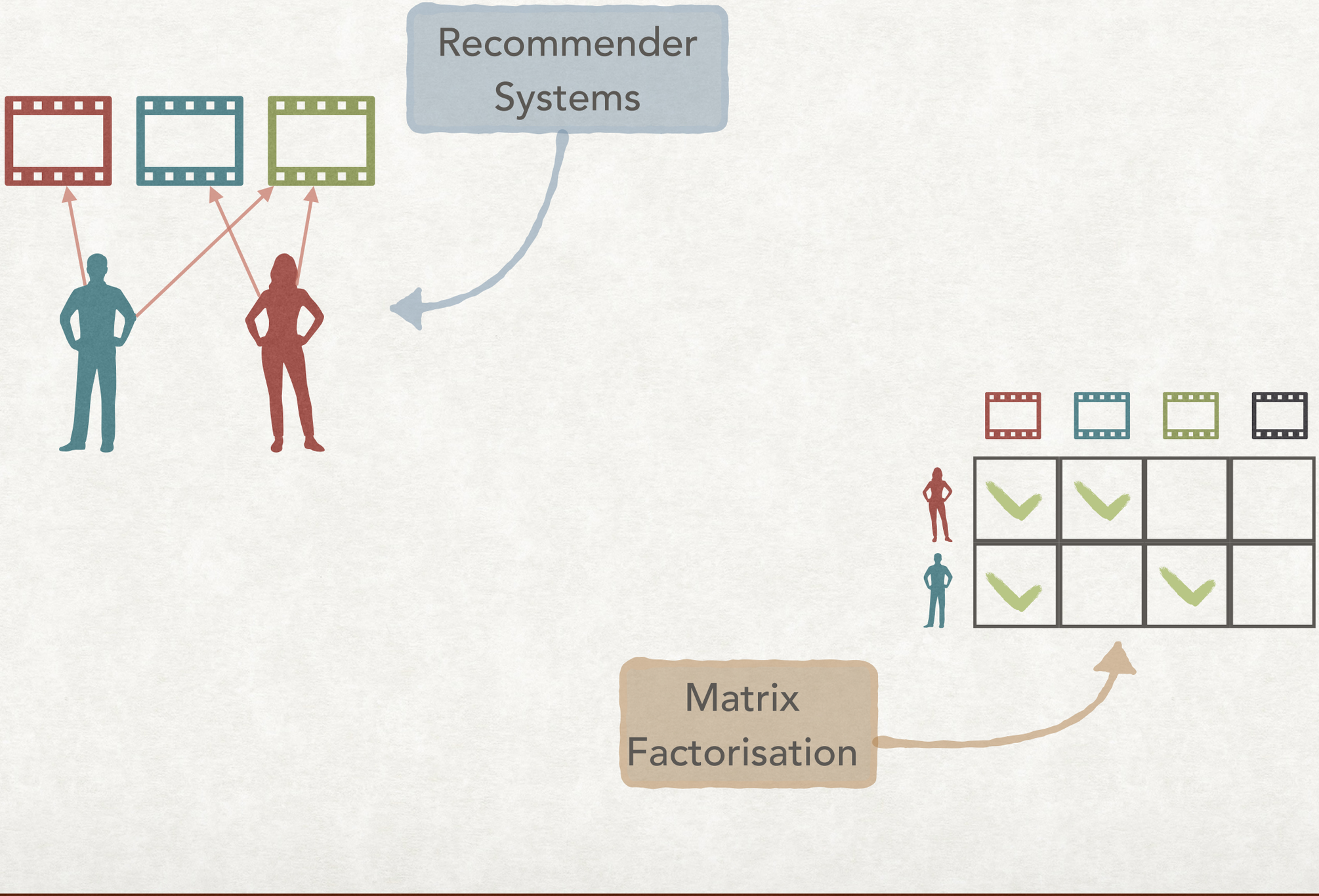
INFORMATION RETRIEVAL

Laura Nenzi
lnenzi@units.it

Lecture 9

LECTURE OUTLINE

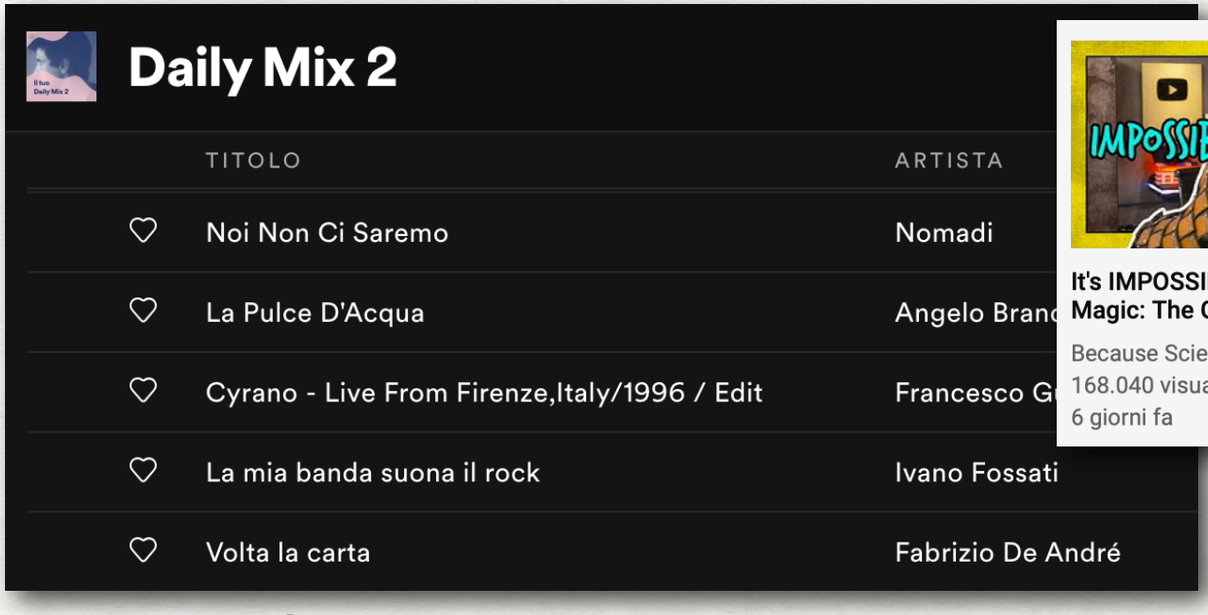
TODAY WITH MATRICES



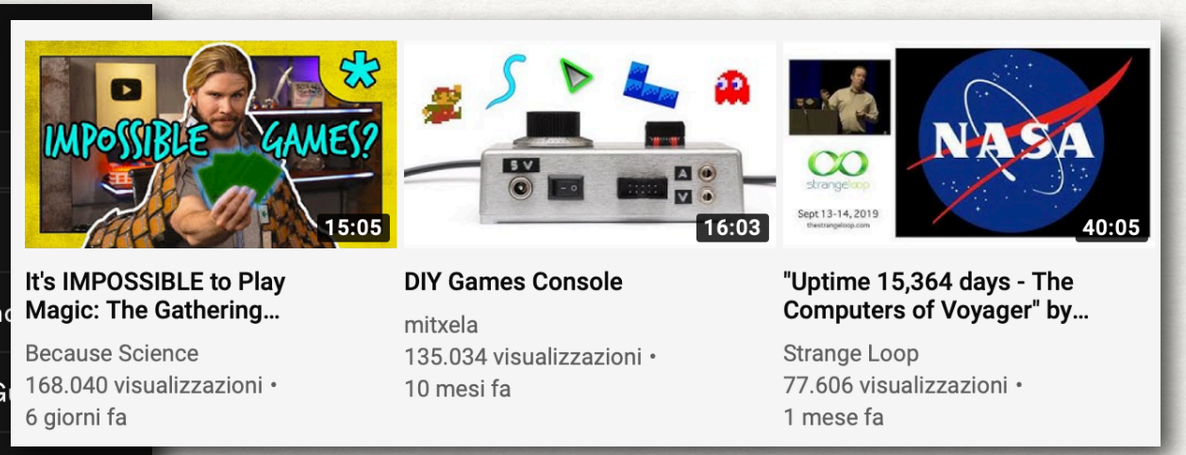
RECOMMENDER SYSTEMS

EXAMPLE OF USES OF RECOMMENDER SYSTEMS

YOU PROBABLY KNOW THEM



Spotify



Youtube

Amazon

Netflix

BASIC CHARACTERISTICS

WHAT PROBLEMS NEED SOLVING

- We do not have a “normal” query, only the previous choices of the user and of similar users.
- We have to provide the user with a collection of suggested items/ documents that he/she might like.
- This is an important feature: according to Google
  “60% of watch time on YouTube comes from recommendations.”
- Recommendation systems are a kind of **information filtering systems**: we already have all the information, but we need to filter the *relevant* information.

BASIC CHARACTERISTICS

WHAT IS A QUERY

- A “query” for a recommender system is also called a **context**.
- It is a combination of information about the user, like:
  - An identifier of the user.
  - The history of interaction by the user
    (e.g. liked video, music listened, watched items).
  - Some additional information, like the time of the day.

TYPES OF RECOMMENDER SYSTEMS

CONTENT-BASED AND COLLABORATIVE

Content-based filtering

Based on the similarity between **items**

The user likes cat videos…
…we will suggest more cat video

Many real-world systems

Collaborative filtering

Based on the similarity between **queries** and **items** *simultaneously*

User A is similar to user B…
…user B likes the video
“cute cat #37”…
…we will propose it to user A

PROBLEMS FOR RECOMMENDER SYSTEMS

- There are multiple issues that a recommender system must address. :
  - **Cold start**. New documents have no ratings/watching/etc., and new users haven't rated/watched/listened anything.
  - **Sparsity**. Most users rate/watch/listen only a small subset of the entire collection.
  - **Scalability**. The collection can be very large, and the time available to make a recommendation quite small.

A FEW HISTORY ON RS

- Grundy, the first recommender system,1979, created by Elaine Rich. It recommended users books they might like based on users specific questions, classifying them into classes of preferences depending on their answers.

- Item-based Collaborative Filtering (late 1990s): Introduced by Amazon—faster and more scalable

- Netflix Prize (2006–2009): Offered $1M for improving Netflix's recommendation algorithm. The winning approach used an ensemble of many methods, and in particular it used matrix factorization.

STRUCTURE OF A RECOMMENDER SYSTEM

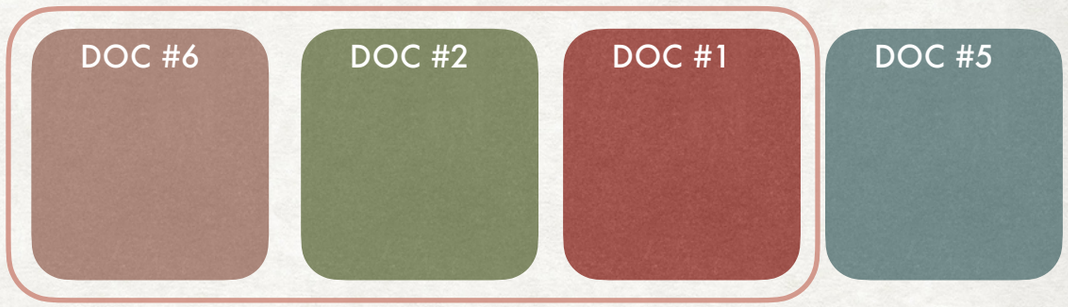
AN EXAMPLE FROM GOOGLE

Possibly billions of documents

Corpus

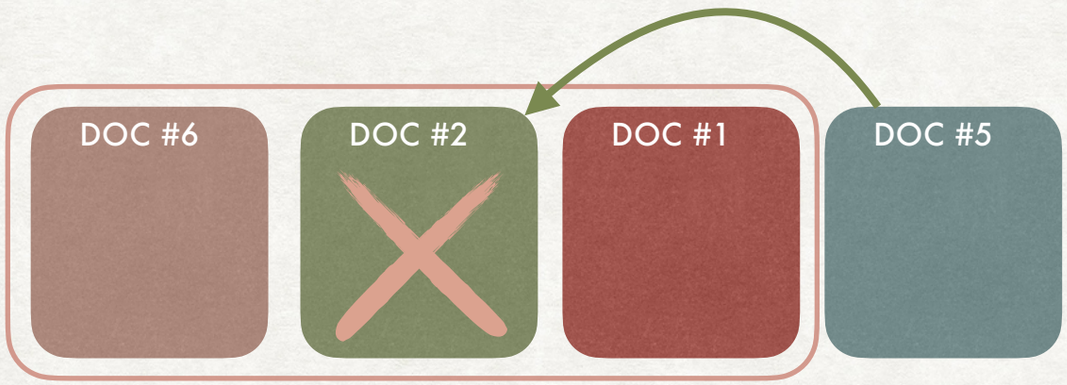
Up to hundreds or thousands of documents

Candidate generation



Up to tens of suggestions to show to the user

Scoring



Re-ranking

Take into account additional information: disliked content, freshness, etc.

CANDIDATE SELECTION

WHY A SEPARATE STEP

- We need to provide a subset of the corpus for the next step
- The corpus can be enormous, thus the retrieval must be fast
- There can be multiple candidate selection methods:
  - Based on similar items and queries
  - Based on popularity
  - Based on specific user preferences, etc.
- We can run all of them, it will be the scoring function the one performing the actual choice.

SCORING

RANKING THE CANDIDATES

- The same method used for candidate selection can be used for scoring…
- …but we might have multiple candidate selection methods…
- …and a separate scoring function can also take additional features into account, since it operates on fewer documents.
- For the scoring we can take into account the user history, the time of the day, the feature of the document, etc.

RE-RANKING

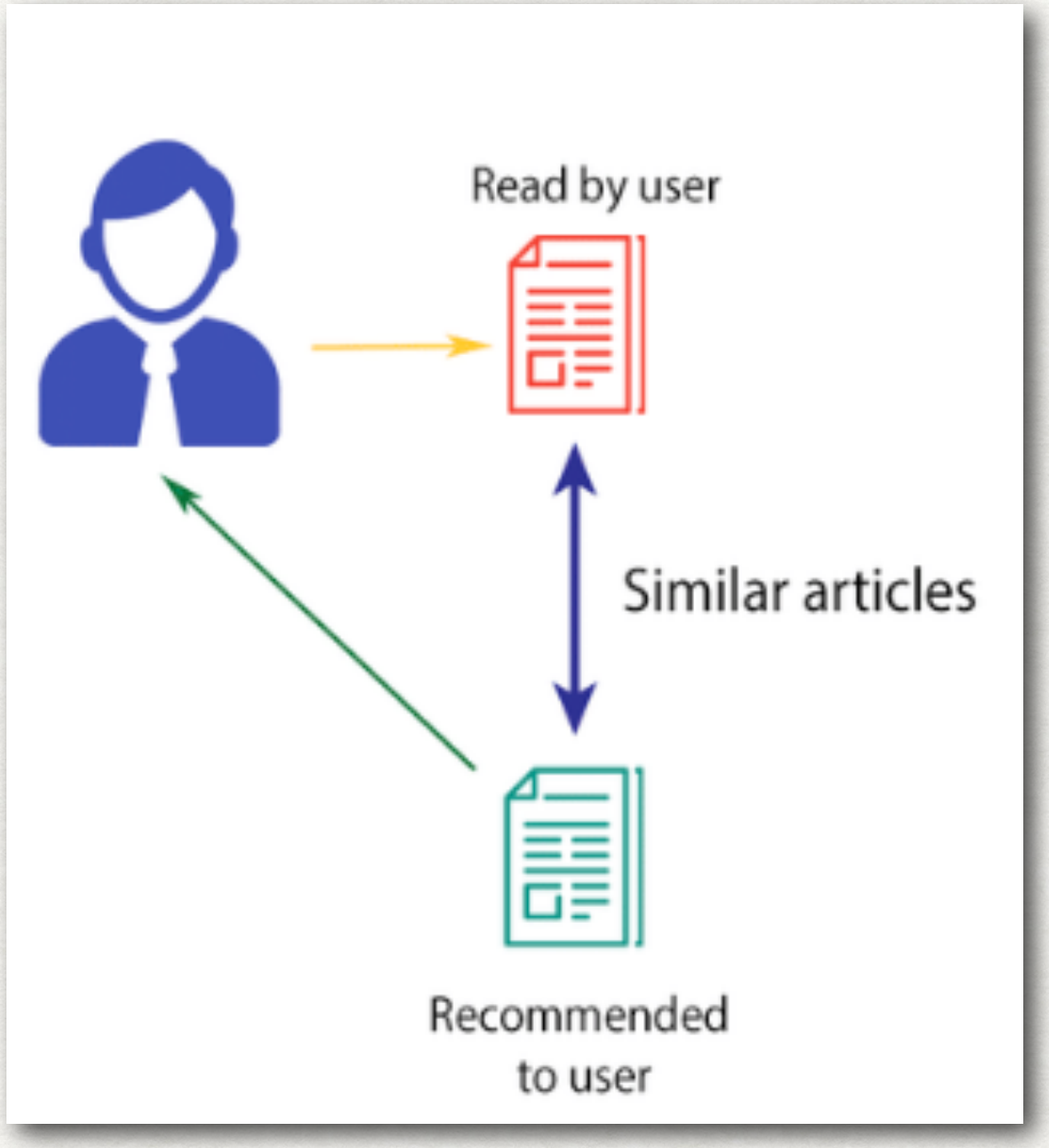
DOING RANKING A SECOND TIME

- Sometimes it is useful to “arrange” the ranking to ensure additional properties, like:
  - Freshness. Take into account new documents, maybe adding the “age” of a document as a feature.
  - Diversity. If a user likes “cute cat video #37”, maybe showing only “cute cat video #n” for all n is not the best choice.

CONTENT-BASED FILTERING

SIMILARITY BETWEEN **ITEMS**

- Based on a description of the item and a profile of the user's preferences
- create a user-profile to indicate the type of item this user likes.
- learn a classifier for the user's likes and dislikes based on an item's features.
- Best suited to situations where there is known data on an item (name, location, description, etc.), but not on the user
- This approach has its roots in information retrieval and information filtering research.



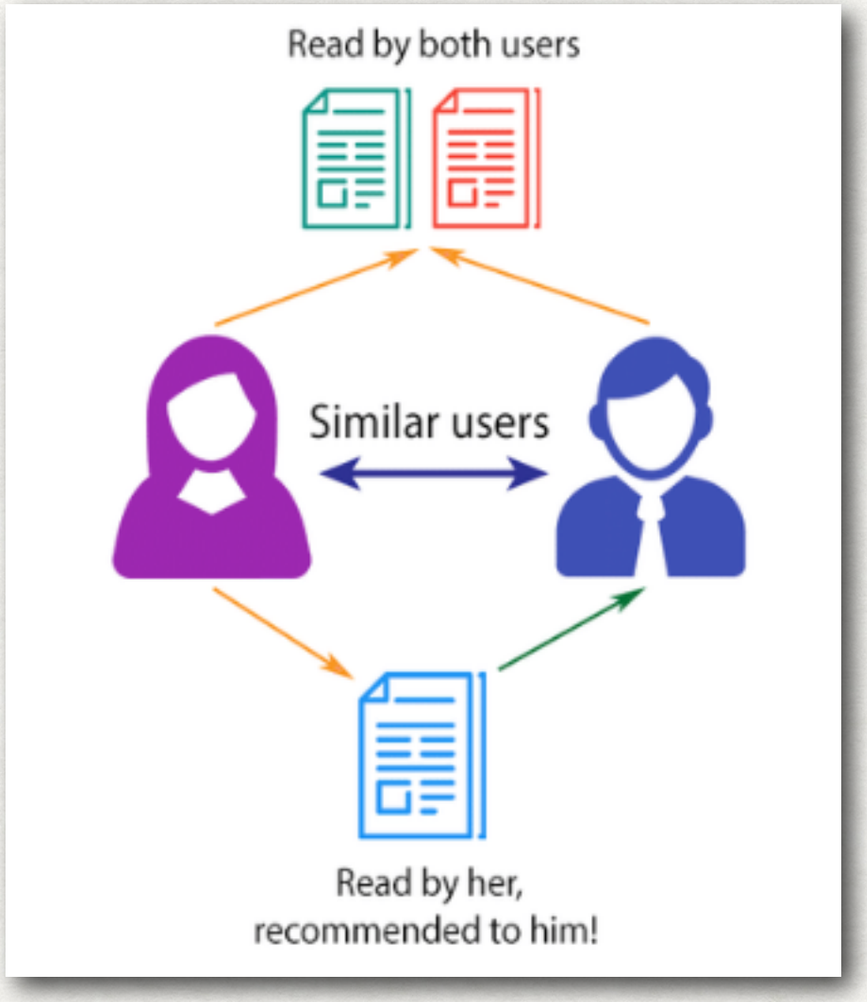
COLLABORATIVE FILTERING

SIMILARITY BETWEEN **USERS**

- based on the assumption that people who agreed in the past will agree in the future

- recommendations using only information about rating profiles for different users or items.

- Collaborative filtering methods are classified as memory-based (rely on similarity between user of items) and model-based (models to predict user preferences.).



USERS AND DOCUMENTS

A REPRESENTATION

We have a matrix C (*feedback* matrix) of users (rows) and of documents (columns).
The position $C_{i,j}$ contains if a user liked a document or not.

	d_1	d_2	d_3	d_4
u_1	?	✓	?	✓
u_2	✓	?	?	?
u_3	?	?	?	✓

MATRIX FACTORISATION

WHAT IS MATRIX FACTORISATION

IN RECOMMENDATION SYSTEMS

- This is a particular technique to map users and documents to a space of features where similarity can be computed.
- This might seem familiar…and it is.
- There are however some important differences.
  - We do not have term-document matrix
  - Users and Documents can have several different information
  - We only have partial information: we know which documents the user likes/dislikes but this is only a small fraction of the documents

USERS AND DOCUMENTS

A REPRESENTATION

We have a matrix C (*feedback* matrix) of users (rows) and of documents (columns).
The position $C_{i,j}$ contains if a user liked a document or not.

	d_1	d_2	d_3	d_4
u_1	?	✓	?	✓
u_2	✓	?	?	?
u_3	?	?	?	✓

WHAT ABOUT UNKNOWN VALUES?

"YOU KNOW NOTHING JON SNOW"

- We can have information about the documents the the user has liked, rated, etc.
- Sometimes we can even obtain information indirectly: e.g., watching an entire video maybe it is an implicit way of "liking" it.
- But for most document we know nothing: the user never accessed them. For example: videos on Youtube.
- Depending on the assumptions that we make about the missing values we can end un with different results.

WHAT WE WANT TO DO

MATRIX FACTORISATION

Given a $M \times N$ feedback matrix C,
we want to find two matrices U and V such that:

- U has M rows and k columns.
- V has N rows and k columns.
- UV^T is an approximation of C according to some criteria.

Where the criteria depends on how we treat missing/not observed entries, and k is the number of *latent factors*.

These U and V are, in general, not the same we used for SVD

LATENT FACTORS

WHAT THEY ARE

User embedding

$$U = \begin{bmatrix} 0.37 & 0 \\ 0 & 1 \\ 0.85 & 0 \end{bmatrix}$$

This is the representation
for the first user
as a vector of two *latent factors*

Item embedding

$$V = \begin{bmatrix} 0 & 1 \\ 0.53 & 0 \\ 0 & 0 \\ 0.85 & 0 \end{bmatrix}$$

This is the representation
for the second item
as a vector of two *latent factors*

The value k (number of latent factors) represents the size
of the space in which we are mapping users and items.

DIFFERENT OBJECTIVE FUNCTIONS

AND ASSUMPTIONS ON UNOBSERVED VALUES

Let C_k be the approximation of C built using k latent factors.

Let Obs be the set of observed positions and Nobs be the set of unobserved ones

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

All unobserved values are 0

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

All unobserved values are 0,
but we weight them with w_0

$$\begin{bmatrix} ? & 1 & ? & ? \\ 1 & ? & ? & ? \\ ? & ? & ? & 1 \end{bmatrix}$$

We do not count
unobserved values

We want to minimise $\|C - C'\|_F$

This actually means that we are performing SVD.

Usually not a good choice since we do not want to force to zero the unknown values!

DIFFERENT OBJECTIVE FUNCTIONS

AND ASSUMPTIONS ON UNOBSERVED VALUES

Let C_k be the approximation of C built using k latent factors.

Let Obs be the set of observed positions and Nobs be the set of unobserved ones

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

All unobserved values are 0

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

All unobserved values are 0,
but we weight them with w_0

$$\begin{bmatrix} ? & 1 & ? & ? \\ 1 & ? & ? & ? \\ ? & ? & ? & 1 \end{bmatrix}$$

We do not count
unobserved values

We want to minimise $\sum_{i,j \in \text{Obs}} (C_{i,j} - C'_{i,j})^2$

This is called **Observed-only Matrix Factorisation**

DIFFERENT OBJECTIVE FUNCTIONS

AND ASSUMPTIONS ON UNOBSERVED VALUES

Let C' be the approximation of C built using k latent factors.

Let Obs be the set of observed positions and Nobs be the set of unobserved ones

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

All unobserved values are 0

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

All unobserved values are 0, but we weight them with w_0

$$\begin{bmatrix} ? & 1 & ? & ? \\ 1 & ? & ? & ? \\ ? & ? & ? & 1 \end{bmatrix}$$

We do not count unobserved values

We want to minimise $\sum_{i,j \in \text{Obs}} (C_{i,j} - C'_{i,j})^2 + w_0 \sum_{i,j \in \text{Nobs}} (C_{i,j} - C'_{i,j})^2$

The factor w_0 decides how important it is to set the unknown weights to 0

This is called **Weighted Matrix Factorisation** (weighted MF)

WEIGHTED MF

SOME OBSERVATIONS

- We will focus on the Weighted MF, since by changing the parameter w_0 it also includes the other two cases.
- The choice of the parameter w_0 is important, but in practice you might also want to weight the *observed* values:
- We optimise the function:

$$\sum_{i,j \in \text{Obs}} w_{i,j}(C_{i,j} - C'_{i,j})^2 + w_0 \sum_{i,j \in \text{Nobs}} (C_{i,j} - C'_{i,j})^2$$

WEIGHTED MF

SOME OBSERVATIONS

- How can we perform the optimisation?
- Start with two matrices U and V and iteratively change them. How?
  - Stochastic Gradient Descend (SGD)
  - Weighted Alternating Least Squares (WALS)
- The last one is specific to this task.

WEIGHTED ALTERNATING LEAST SQUARES

GENERAL IDEA

The main idea of the algorithm is the following:

- Start with U and V randomly generated, $C'_{i,j} = \mathbf{u}_i^\top \mathbf{v}_j$
- Fix V and find, by solving a linear system, the best U.

$$\min_{\mathbf{u}_i} \sum_{j \in \mathsf{Obs}_i} w_{i,j}(C_{i,j} - \mathbf{u}_i^\top \mathbf{v}_j)^2 + w_0 \sum_{j \notin \mathsf{Obs}_i} (\mathbf{u}_i^\top \mathbf{v}_j)^2$$

- Fix U and find, by solving a linear system, the best V.
- Repeat as needed.

The algorithm is guaranteed to converge and can be parallelised.